University of Exeter
Department of Computer Science

Multi-Objective ROC learning for classification

Andrew Robert James Clark

December 2011

Submitted by Andrew Robert James Clark, to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Computer Science , December 2011.

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

(signature) .....

Declaration

Elements of this thesis have appeared previously as follows:

Chapter 3

Clark, A. and Everson, R. (2008). *Multi-objective optimisation of relevance vector machines: Selecting sparse features for face verification.* ICML/UAI/COLT Workshop on Sparse Optimisation and Variable Selection.

Clark, A. and Everson, R. (2011a). *Evolving sparse multi-resolution RVM classifiers.* In Dupenois, M. and Walker, D., editors, Proceedings of the 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), pages 53 – 60, Exeter, UK. PCCAT, College of Engineering, Mathematics and Physical Sciences, University of Exeter.

Clark, A. and Everson, R. (2011b). *Multi-objective learning of relevance vector machine classifiers with multi-resolution kernels.* Pattern Recognition. In Press.

Chapter 5

Clark, A. and Everson, R. (2011c). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at ABCiL 2011, Imperial College London.

Clark, A. and Everson, R. (2011d). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), University of Exeter.

Andrew Robert James Clark

To my family and friends for all their love and support.

Abstract

Receiver operating characteristic (ROC) curves are widely used for evaluating classifier performance, having been applied to e.g. signal detection, medical diagnostics and safety critical systems. They allow examination of the trade-offs between true and false positive rates as misclassification costs are varied. Examination of the resulting graphs and calculation of the area under the ROC curve (AUC) allows assessment of how well a classifier is able to separate two classes and allows selection of an operating point with full knowledge of the available trade-offs.

In this thesis a multi-objective evolutionary algorithm (MOEA) is used to find classifiers whose ROC graph locations are Pareto optimal. The Relevance Vector Machine (RVM) is a state-of-the-art classifier that produces sparse Bayesian models, but is unfortunately prone to overfitting. Using the MOEA, hyper-parameters for RVM classifiers are set, optimising them not only in terms of true and false positive rates but also a novel measure of RVM complexity, thus encouraging sparseness, and producing approximations to the Pareto front. Several methods for regularising the RVM during the MOEA training process are examined and their performance evaluated on a number of benchmark datasets demonstrating they possess the capability to avoid overfitting whilst producing performance equivalent to that of the maximum likelihood trained RVM.

A common task in bioinformatics is to identify genes associated with various genetic conditions by finding those genes useful for classifying a condition against a baseline. Typically, datasets contain large numbers of gene expressions measured in relatively few subjects. As a result of the high dimensionality and sparsity of examples, it can be very easy to find classifiers with near perfect training accuracies but which have poor generalisation capability. Additionally, depending on the condition and treatment involved, evaluation over a range of costs will often be desirable. An MOEA is used to identify genes for classification by simultaneously maximising the area under the ROC curve whilst minimising model complexity. This method is illustrated on a number of well-studied datasets and applied to a recent bioinformatics database resulting from the current InChianti population study.

Many classifiers produce "hard", non-probabilistic classifications and are trained to find a single set of parameters, whose values are inevitably uncertain due to limited available training data. In a Bayesian framework it is possible to ameliorate the effects of this parameter uncertainty by averaging over classifiers weighted by their posterior probability. Unfortunately, the required posterior probability is not readily computed for hard classifiers. In this thesis an Approximate Bayesian Computation Markov Chain Monte Carlo algorithm is used to sample model parameters for a hard classifier using the AUC as a measure of performance. The ability to produce ROC curves close to the Bayes optimal ROC curve is demonstrated on a synthetic dataset. Due to the large numbers of sampled parametrisations, averaging over them when rapid classification is needed may be impractical and thus methods for producing sparse weightings are investigated.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

Acronyms and abbreviations used in this work:

Term	**Meaning**
ABC	Approximate Bayesian computation
AUC	Area under the ROC curve
AUC Error	1 - AUC
EA	Evolutionary Algorithm
fRVM	Fast RVM
MCMC	Markov chain Monte Carlo
MLP	Multi-layer perceptron
MOEA	Multi-objective Evolutionary Algorithm
ROC curve	Receiver operating characteristic curve
RVM	Relevance vector machine
sRVM	Smooth RVM
SVM	Support vector machine
vRVM	Variational RVM

1. Introduction

A common task in machine learning is learning how to accurately predict a variable whose value is thought to depend on some independent variables or patterns. In supervised learning the models with which to make these predictions are learnt from a training dataset for which we have both input patterns and their targets. In regression problems, the predictions we wish to make based on the input patterns are real valued e.g. the amount of rainfall based on temperature readings. In classification, the subject of this thesis, predictions are the assignment of an input to a particular class. For example a doctor screening biopsy samples for the presence of cancer would want to assign them to either the cancerous class or the non-cancerous class.

A desirable characteristic of a classifier is for it to give a measure of certainty of class assignment. We refer to classifiers that provide a measure of certainty of class membership as "soft" classifiers and classifiers which only return a class labelling and no measure of certainty as "hard" classifiers.

Often, few training data samples are available so there will be uncertainty in the parametrisation of the classification model learnt. One appealing way of accounting for this uncertainty is by calculating posterior probabilities of class membership, taking account of the training data paucity and uncertainty in the parametrisation. Having obtained these posterior probabilities of class membership, thresholds can be set above which an assignment to one class or another is made. This threshold is referred to as the "decision threshold". As we increase the decision threshold so we require the model to be increasingly certain of its assignment.

In order to perform classification, classifiers learn a boundary, the "decision boundary", between the classes in the form of a hyper-plane. Such hyper-planes are either linear or non-linear. Linear classifiers use linear hyper-planes to separate classes and as such produce decision boundaries in the form of straight lines. Non-linear classifiers make use of non-linear hyper-planes to separate classes and can produce curved decision boundaries.

Early classifiers were based on Linear Discriminant Analysis [LDA e.g. Duda et al., 2001; McLachlan, 2004], sometimes referred to as Fisher's linear discriminant [Fisher, 1936]. In these, a linearly weighted combination of the input features is used to produce classifications. Decision trees [Breiman, 1984] are another form of linear classifier. In these the classifications are made by repeatedly splitting the input space. As with LDA this leads to a linear decision boundary.

One of the simplest non-linear classifiers is the K-nearest neighbours classifier [kNN Cover and Hart, 1967]. Using a training or reference dataset for which class labels are known, input patterns are classified by finding the nearest K reference patterns to the input and using their class labels to vote on the input's class membership. A particularly useful property of the kNN classifier is that when $K = 1$ its error rate is no worse than

twice the Bayes error rate. Inspired by the structure of neurons in the brain, Artificial Neural Networks (ANNs) were developed [Bishop, 1995]. Using layers of artificial neurons, whose interconnection strengths are learnt during the training process, ANNs are capable of learning more complex non-linear relationships between inputs and targets.

More recently, kernel based classifiers have become popular which, while linear themselves, make use of non-linear transformations of the data to obtain non-linear decision boundaries. Using kernels, data is mapped from its original space into a higher dimension in which it is hoped it will be linearly separable. Perhaps the most established and commonly used kernel based classifier is the Support Vector Machine [Boser et al., 1992; Vapnik, 1998; Schölkopf et al., 1999] which learns binary classifiers. Recent work has seen the Relevance Vector Machine [Tipping, 2000, 2001; Faul and Tipping, 2002; Tipping and Faul, 2003] and Gaussian Processes [Rasmussen, 2004] introduced, both yielding state-of-the-art performance.

In the vast majority of datasets there is the added problem of "noise" in the samples usually relating to imprecision in the measuring device. Equally importantly, the distributions from which classes are drawn will commonly overlap and as such there will be some mixing of the classes. When the classification models being learnt are too flexible they will often learn this noise in the training data and attempt perfect class separation, resulting in overly convoluted decision boundaries. This is referred to as "overfitting" and models which do so fail to perform as well on previously unseen data as they did on their training data. A model's performance on such previously unseen datasets is its generalisation performance and a classifier is said to generalise well when it performs consistently well on new datasets[1]. Models with markedly reduced performance on new datasets generalise poorly.

To avoid overfitting and give good generalisation capability one often looks to learn sparse models. One way of doing this is to penalise complexity during the training process. This penalty may be applied to the model structure or to the magnitude of the learned weights. Sparsity may be induced by simplifying model structure or by learning weightings for the input features in which the majority of values are 0, effectively discarding them, or some combination of the two. Removal of hidden units in a neural network is an example of simplifying model structure. Support Vector Machines and Relevance Vector Machines are examples of models where sparsity is obtained through sparse weighting of input features and it is this approach that is followed in this thesis. By encouraging sparsity, the learning of overly flexible models is reduced and thus simpler solutions are learned with smoother decision boundaries and improved generalisation capability.

When analysing classifier performance we typically denote the class we are interested in as the positive class and the other class as the negative. For example when screening for cancer we would denote the cancerous samples as the positive class and the non-cancerous ones as the negative class. Classifier performance can be assessed in many different ways of which the most commonly used is accuracy. Learnt models are applied to a dataset and predictions made for each of the samples in it. This dataset might be the data on which the model was trained or some previously unseen data for which class labels are available.

[1] N.B. While a classification model might perform equally badly on both training and unseen test data, we would not say that it generalises well.

Using these predictions and targets performance can then be measured.

Accuracy measures the fraction of data points that are assigned to their true class. This gives a single measure of performance that can be straightforwardly calculated and interpreted. Unfortunately, there are a number of problems with using accuracy to assess model performance. Firstly, it can be misleading when class proportions are unequal, a not uncommon situation. Consider a dataset consisting of 90 examples from the positive class and 10 from the negative class. A classifier simply classifying everything as positive would achieve 90% accuracy on this dataset but have completely failed to model the negative class. Secondly, accuracy fails to take account of unequal misclassification costs. In many real problems the misclassification costs for each of the classes are often different. For example, when attempting to diagnose cancer from a biopsy sample, it is much more costly to fail to classify a cancerous cell sample as cancerous than misclassify a non-cancerous one when carrying out an initial screening leading to further tests. Failure to identify some cancers at an early stage is potentially life threatening (e.g. colon cancer) and treatment in later stages may be increasingly expensive and invasive with reduced chances of survival.

An alternative measure to accuracy is true and false positive rates. Using the model predictions and knowledge of their true classes one can produce counts for correct classifications and misclassifications. A correct classification is referred to as a "true" classification whereas a misclassification is a "false" classification. For a binary classification problem we thus receive counts for the positive and negative class members in the form of True Positives, False Positives, True Negatives and False Negatives. Note that as the count of True Positives increases so the number of False Negatives decreases since the sum of the number of True Positives and False Positives is equal to the number of members of the positive class. Similarly increasing the number of True Negatives leads to a reduction in the number of False Positives. Dividing the number of True Positives by the total number positive class examples yields the True Positive Rate and similarly, division of the number of False Positives by number of negative class examples yields the False Positive Rate. Unlike accuracy, calculation of each of these rates is dependent on one class at a time (either the positive or negative class) and are thus unaffected by changes in class proportions giving an unbiased measure of class-wise performance. Classifiers giving a "hard" class labelling yield a single true and false positive rate. However, those classifiers giving a measure of certainty of class membership (e.g. in the form of a score or posterior probability) can yield multiple true and false positive rate combinations by varying the decision threshold, the threshold value above which an assignment to one class or another is made.

Receiver Operating Characteristic (ROC) curves are a popular way of visualising and comparing classifier performance over the range of misclassification costs by plotting true and false positive rates [Hanley and McNeil, 1982; Provost and Fawcett, 1997, 1998; Fawcett, 2006]. By varying the decision threshold from the minimum to maximum possible classification certainty, all true and false positive rate combinations for a classifier are found and a full ROC curve obtained. This represents the range of cases from the case where it is infinitely more costly to misclassify something as a member of the negative class to the case where it is infinitely more costly to misclassify something as a member

of the positive class. For any given decision threshold there is a trade-off between true and false positive rates. At lower decision thresholds it is easier to classify patterns as belonging to the positive class and thus we will obtain high true positive rates and high false positive rates. Conversely, at higher decision thresholds it becomes harder to assign points to the positive class leading to lower true and false positive rates. Since there is this trade-off between true and false positive rates one can consider them to be competing objectives.

By presenting the full set of true and false positive trade-offs the user is able to select the operating point best suiting their misclassification costs with knowledge of the possible alternatives. Calculation of the area under the ROC curve (AUC) yields a measure of how well a classifier is able to discriminate between the two classes over the range of misclassification costs [Hanley and McNeil, 1982; Hand and Till, 2001; Fieldsend and Everson, 2008a]. In this thesis models are sought which give the best trade-offs between ROC performance and model complexity.

Multi-Objective Evolutionary Algorithms (MOEAs) are an increasingly popular method used to learn classifiers when there are multiple competing measures of performance, such as true positive rates, false positive rates, AUC and complexity. Inspired by evolution and in a similar manner to Genetic Algorithms [Goldberg, 1989], model parametrisations defining the structure and/or weights of a classifier are perturbed (mutated) and the fittest solutions kept, maintaining an archive of "elite" solutions containing the best set of trade-offs found between the competing objectives. Parametrisations are selected from the archive and perturbed to give new models which are evaluated on each of the objectives. If a new solution is no worse than the existing members of the elite archive it is added to the archive and any members which are worse on all objectives than the new solution are eliminated from the archive. This process is repeated for some number of generations to locate the best set of trade-offs between the competing objectives. From this elite archive the user can select the solution that best meets their desired trade-off combination with knowledge of the possible alternatives. By using an MOEA one can incorporate any calculable measures of choice into the model training process and thus it is possible to simultaneously optimise ROC curves and model complexities. In Chapter 3 an MOEA is used to find the best set of trade-offs between RVM classifier true and false positive rates along with a novel measure of complexity which encourages sparse models with optimal ROC performance. Results on a number of benchmark datasets demonstrate the ability of the MOEA trained classifiers to avoid overfitting whilst yielding performance equivalent to that of the maximum-likelihood trained RVM.

A common task in Bioinformatics is the identification of genes for classifying a genetic condition against some baseline. By identifying differences in particular genes it is hoped that an individual's propensity to particular conditions can be detected and potentially countered through gene therapy or possible changes in lifestyle. For example, researchers are often interested in identifying the genetic links to cancer in the hopes of early detection and possible treatments. Typically, datasets are very high dimensional, thousands of features are common, but few training examples are available – sometimes only tens [e.g. Notterman et al., 2001; Hedenfalk et al., 2001]. Such paucity of data means that it is often possible to learn numerous models achieving near perfect training accuracy but

which fail to generalise well to new data. Given the ramifications of identifying genes as being associated with a condition, it is important that *only* those genes needed for the classification are selected, i.e. that overfitting is avoided. Since the severity of conditions, cost of any resulting treatment and invasiveness of any additional tests required varies, so will the costs of misclassification. It is therefore desirable to learn classifiers giving the best possible performance over the range of misclassification costs. AUC provides a measure of how well a classifier is able to discriminate between two classes [Hanley and McNeil, 1982; Hand and Till, 2001; Fieldsend and Everson, 2008a] and thus, in Chapter 4, an MOEA is used to identify gene combinations giving the best trade-offs between AUC and model complexity.

Many classifiers provide hard, non-probabilistic, classifications taking the form of a class labelling but without any measure of the prediction certainty. Such classifiers are common in industrial applications for example the NATS Short Term Conflict Alert (STCA) system which gives hard classifications predicting whether or not pairs of aircraft trajectories will become dangerously close [Everson and Fieldsend, 2006b; Reckhouse et al., 2010].

During classifier training a single set of model parameters is learnt whose values are uncertain due to the limited availability of training data e.g. in the STCA case there are few training examples for aircraft on collision trajectories. As a result there is uncertainty in the predictions which we are not made aware of. In the Bayesian framework one can ameliorate the effects of this parameter uncertainty by averaging over classifiers weighted by the posterior probability of their parametrisations [Bernardo and Smith, 1994]. Unfortunately, as hard classifiers do not supply a posterior probability the Bayesian framework cannot straightforwardly be used.

Approximate Bayesian Computation methods [Marjoram et al., 2003] allow one to compensate for the inability to evaluate the posterior probability of model parametrisations through a surrogate measure of posterior probability and sampling procedures. Such methods are usually used in regression [Toni et al., 2009] or "model fitting" context [Wilkinson and Tavaré, 2009] to sample parameters for a model simulating measured data e.g. a system of ordinary differential equations (ODEs). Datasets are simulated using sampled model parametrisations and compared with the original dataset using some distance function. If this distance is lower than a specified error threshold then the parametrisation is kept. The resulting set of model parametrisation samples are thus drawn from a distribution conditioned on the distance function and value for the error threshold. Using these one can then produce predictions by averaging over the sampled model parametrisations.

Since many datasets are high-dimensional, calculation of the distance between the original and simulated dataset can be expensive. Instead, summary statistics can be used in the distance function, chosen based on knowledge of the problem at hand to be informative regarding the choice of model parametrisation. In this thesis (Chapter 5), an Approximate Bayesian Computation Markov Chain Monte Carlo algorithm is used to sample parametrisations for a hard classifier using AUC as the summary statistic in the distance function. By averaging over the sampled classifiers' predictions an approximation of posterior probability of class membership is obtained.

1.1. Structure of the thesis

Chapter 2. An overview of relevant literature is given, beginning with a discussion of supervised learning. Methods for evaluating classifier performance are examined and their various merits and problems discussed. Algorithms for learning sparse models with the aim of reducing overfitting and obtaining good generalisation performance are then discussed with a particular focus on the Relevance Vector Machine. Finally an introduction to multi-objective optimisation and multi-objective evolutionary algorithms is given.

Chapter 3. A multi objective optimising evolutionary algorithm (MOEA) is used to locate Pareto optimal classifiers in terms of true and false positive rates along with a novel measure of complexity, encouraging sparseness. The evolutionary optimisation process is further regularised using two different proposed cross validation schemes. Performance for the MOEA located classifiers is compared with Relevance Vector Machines (RVM) trained using maximum likelihood on a number of benchmark datasets. The results demonstrate the ability of the MOEA trained classifiers to avoid overfitting whilst producing performance equivalent to that of the RVM.

Chapter 4. The task of gene identification for classification of genetic conditions is addressed. Given the sparsity of bioinformatics datasets it can be hard to learn classifiers with good generalisation performance. A MOEA is used to select genes for classification, simultaneously maximising the area under the ROC curve while minimising classifier complexity. In combination with a data-partitioning scheme the MOEA and RVM are used to identify genes for classification on a number of well-studied bioinformatics datasets as well as a more recent database arising from the InChianti population study [Ferrucci et al., 2000].

Chapter 5. An Approximate Bayesian Computation algorithm is used to obtain estimates for posterior classification probabilities from hard classifiers. Since large numbers of model parametrisations are generated by the sampling process it may be impractical to average over them when rapid classification is desired or computational resources are limited. The possibility of obtaining a sparse weighting of the sampled classifiers that reproduces the performance of the whole chain is investigated.

Chapter 6. The work carried out in each of the chapters is summarised along with possible further extensions. An overview of the thesis is then given.

1.2. Principal Contributions

The main contributions of this thesis are as follows:

- A novel measure of relevance vector machine classifier complexity.
- A multi-objective optimising evolutionary algorithm for training relevance vector machine classifiers by optimising true and false positive rates along with model complexity.

- An integration of K-fold cross validation into evolutionary optimisation for preventing overfitting.
- A multi-objective optimising evolutionary algorithm for gene selection that optimises models in terms of class discrimination, as measured by the area under the ROC curve, and complexity.
- An Approximate Bayesian Computation method for obtaining estimates of posterior probabilities of class membership from hard classifiers.

2. Background

A common task in machine learning is learning how to accurately predict a variable, t, whose value is thought to be dependent on some independent variables or pattern, $\mathbf{x}$ [Duda et al., 2001; Hastie et al., 2004; Bishop, 2006]. Given a training dataset of N sample-target pairs $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ we seek to model the relationship between the samples, $\mathbf{x}$, and their targets, t.

Consider the case of a steam powered factory. Based on some amount of steam fed into the factory, $\mathbf{x}$, we would like to know how much productivity, t, we will obtain from it. By feeding N different amounts of steam, $\mathbf{x}_n$, into the factory we obtain different associated values of productivity, t_n, to obtain a training dataset $\mathcal{D}_{\text{tr}}$ consisting of N steam-productivity pairs. If the factory was in perfect working order then the productivity would only be dependent on the amount of steam input and so we could model t as:

$$t = f(\mathbf{x}) \tag{2.1}$$

where $f(\mathbf{x})$ is the mapping between the input amount of steam and level of productivity obtained, t. However, the pipes feeding the steam through the factory may occasionally leak and, as a result, a random element, ϵ, is introduced into the relationship between t and $\mathbf{x}$:

$$t = f(\mathbf{x}) + \epsilon. \tag{2.2}$$

Thus, in measuring factory productivity for an input amount of steam, we obtain a value consisting of a systematic element, $f(\mathbf{x})$, and a stochastic element, ϵ.

In another task, we might build a model to predict a person's propensity to having a heart attack based on their age and amounts they drink and smoke. Here our samples, $\mathbf{x}$, take the form of a vector containing an individual's age, how much they drink and how much they smoke. Our targets, t, take the form of a label for whether or not they have had a heart attack. A training dataset can then be built by asking people how old they are, how much they drink and smoke and whether or not they have had a heart attack.

In general, someone who is old, drinks a lot and smokes heavily is more likely to have a heart attack, whereas someone who is young and neither drinks or smokes is less likely to have a heart attack. There are, however, people who are old and drink and smoke heavily who have never had a heart attack. Likewise, there are young people who have never drunk or smoked who have heart attacks. In this case the stochastic element has been introduced into the model by these unlikely, but real, cases.

We desire to learn a model for the systematic component of the relationship between sample and target and ignore this stochastic component, often referred to as "noise". Errors may also be introduced by unmeasured systematic factors, e.g. a familial history of heart attacks not recorded in the data, or in the form of systematic errors, e.g. genetic

samples being prepared in different batches [Ferrucci et al., 2000], both of which we need to consider when selecting a model. Once the chosen model has been trained we can then produce predictions $y(\mathbf{x}')$ of the most likely target for some previously unseen sample, $\mathbf{x}'$, as:

$$y(\mathbf{x}') = f(\mathbf{x}'; \boldsymbol{\theta}) \tag{2.3}$$

where $f(\mathbf{x}; \boldsymbol{\theta})$ is the model and $\boldsymbol{\theta}$ is the model parametrisation. The task of machine learning is to model $f(\mathbf{x}; \boldsymbol{\theta})$ and learn the "best" value for $\boldsymbol{\theta}$ on the basis of some training dataset, $\mathcal{D}_{\text{tr}}$.

Predictions types fall into two categories: *classification* and *regression*. In *classification* tasks [Duda et al., 2001] one labels input samples as belonging to one of two or more categories, referred to as classes. Tasks where there are only two classes to assign between are referred to as *binary* classification tasks (such as the heart attack classification discussed above); those consisting of more than two are referred to as *multi-class* classification tasks. In this thesis we are concerned with binary classification problems. Typically, we denote the class we are interested in detecting as the "positive" class and the other as the "negative" class e.g. when screening for the presence of cancer in tissue samples we are interested in detecting cancer and would therefore denote the cancerous samples as the positive class and non-cancerous samples as the negative cancer. Regardless of whether a classification problem is binary or multi-class, the output of the classification model will be one of a number of discrete values i.e. the predicted class. This differs from *regression* where one looks to produce a continuous output e.g. predicting height based on age or productivity based on steam use.

In both regression and classification we will refer to the model prediction as y, where the prediction for the n^{th} sample is y_n. In classification these predictions are the class labels, typically denoting each class with a discrete value. For the case of binary classification the classes are usually labelled as class 1 for the "positive" class and class 0 for the "negative" class. In regression t will normally be a scalar and thus y will also be scalar. It is worth noting that t could also be a vector for cases when one predicts multiple values at the same time. However, in this thesis the main concern is binary classification so t, and therefore y, takes the form of a scalar label for membership to one of the two classes.

Depending on the task, different types of sample data, $\mathbf{x}_n$, may be provided; for example Gabor wavelet convolutions may be fed to a face recognition system [e.g. Shen et al., 2007], aircraft trajectory data may be used in a collision alert system [e.g. Everson and Fieldsend, 2006b; Reckhouse et al., 2010] or gene expression data may be used to differentiate between genetic conditions [e.g. Li et al., 2002]. Such data may be continuous, discrete or some mix of the two e.g. continuous height values combined with a Boolean gender flag.

Often, the size of these training datasets is limited. Usually this is related to the "cost" in procuring labelled training examples; it may be that is monetarily expensive to run a series of crash tests, time consuming for an expert to label a biopsy slide or overly invasive to procure samples from patients. Other reasons for small sample sizes may be that it is simply rare to observe and thus capture cases from one of the classes e.g. Moyamoya disease is a very rare genetic condition only present in 1 in a million people in the UK [NHS, 2010] and thus hard to locate examples of gene expression data.

As discussed, using this training data we seek to model the function that generated it.

The processes involved in learning the parametrisations for these models fall into three main categories: *unsupervised* learning, *reinforcement* learning and *supervised* learning,

In *unsupervised* learning [e.g. Sejnowski, 1999] the training data consists of input samples, $\mathbf{x}$, without any associated targets. Instead of learning how to predict target values, here learning is focused on attempting to discern some latent structure in the data, perhaps clustering together similar examples within the data [e.g. Moody and Darken, 1989], determining the distribution of the data in the input space, referred to as density estimation [e.g. Bishop, 2006], or projecting the data from a high-dimensional space down to two or three dimensions to enable visualisation [e.g. Jolliffe, 1986; Walker et al., 2010].

During *reinforcement* learning [e.g. Sutton and Barto, 1998], feedback is given to the algorithm but not necessarily in a direct way, typically taking the form of a cost function or reward indicating when performance is good or bad. Rather than being given the targets, the algorithm must discover them for itself, often by a process of trial and error. An example might be for a robot arm balancing a pencil. The robot arm will receive an input indicating how well the pencil is balanced and then make a decision on how it should move. Based on this movement the pencil will either be either better or worse balanced, and it is this which is fed back to the arm. At no point is the robot told the best way to move to best balance the pencil, i.e. the target, it has to learn the best movements to make for itself.

In *supervised* learning [e.g. Kotsiantis, 2007], the machine learning algorithm is given direct feedback as to what the true targets, $\mathbf{t}$, are during the training phase. Many examples exist where one can provide the algorithm with training data for which the true targets are known. Examples include: gene expression data with gender labels as targets, face images with associated identities and music samples with their genre.

In this chapter a number of background topics pertinent to this thesis are discussed. In section 2.1 methods for learning model parametrisations are introduced. Section 2.2 discusses various methods for evaluating the performance of a classifier. In section 2.3 an introduction to sparse modelling is given with a particular focus on the Relevance Vector Machine and its variants in section 2.3.1. Finally, in section 2.4, multi-objective optimisation of classifiers is discussed which enables one to incorporate desired model performance metrics into the learning process.

2.1. Supervised Learning

In supervised learning we desire to learn the function $t = f(\mathbf{x}; \boldsymbol{\theta})$ with which to make predictions, y, for a given $\mathbf{x}$ based on some vector of parameters $\boldsymbol{\theta}$. During training the algorithm is presented with the training data $\mathcal{D}_{\text{tr}}$, consisting of N sets of inputs, $\mathbf{x}_n$, and their corresponding targets, t_n:

$$\mathcal{D}_{\text{tr}} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}. \tag{2.4}$$

Having learned $\boldsymbol{\theta}$, the function $f(\mathbf{x}; \boldsymbol{\theta})$ can be used to make predictions on a previously unseen dataset during the test phase: $f(\mathbf{x}'; \boldsymbol{\theta})$.

Many different supervised learning algorithms exist with different forms for $f(\mathbf{x}; \boldsymbol{\theta})$ and different ways of learning $\boldsymbol{\theta}$ based on the training data. Examples of $f(\mathbf{x}; \boldsymbol{\theta})$ include

multi-layer perceptrons (MLP), Support Vector Machines (SVM) and Relevance Vector Machines (RVM). The process by which $\boldsymbol{\theta}$ is learned, based on $\mathcal{D}_{\text{tr}}$, will depend on the specific algorithm.

In general, the performance of supervised learning algorithms is measured according to some *cost function* $C(f(\mathbf{x};\boldsymbol{\theta}),t)$. This is also called the *error* or *risk* function and is used to compare model predictions with the true targets. A discussion of various classifier performance measures is presented in section 2.2.

Given the limited nature of the training datasets, an important question is how best to generalise from the training data in order to make new predictions on previously unseen data. If $f(\mathbf{x};\boldsymbol{\theta})$ is too complex it may fit the training data well but generalise poorly, referred to as "overfitting", by learning any noise present in the training data and not just the underlying structure. In the case of classification, overfitting results in overly complex class boundaries being learned which try to perfectly separate classes as presented in the training data, $\mathcal{D}_{\text{tr}}$. Less complex models yield smoother class boundaries and, while they may not fit the training data as well as the more complex ones, may generalise better. This trade-off between complexity and generalisation is an important issue in supervised learning, applying to both regression and classification. In chapters 3 and 4 we investigate methods for preventing overfitting during training by penalising model complexity to obtain the best generalisation performance for a given model complexity. In the following sections we present the most commonly used methods for learning the model parametrisation, $\boldsymbol{\theta}$.

Maximum Likelihood

In maximum likelihood (ML) [e.g. Rice, 1988; Sivia and Skilling, 2006] one seeks the parameter set, $\boldsymbol{\theta}$, that is most likely to have produced the training data, $\mathcal{D}_{\text{tr}}$. In order to do this we need an expression for the likelihood of observing $\mathcal{D}_{\text{tr}}$ given $\boldsymbol{\theta}$:

$$p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}) = p(\mathbf{x}_1, t_1, \ldots, \mathbf{x}_N, t_N|\boldsymbol{\theta}). \tag{2.5}$$

The maximum likelihood parameters, $\boldsymbol{\theta}_{ML}$, are those that maximise the likelihood i.e.:

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}). \tag{2.6}$$

Assuming each of the feature-target pairs $(\mathbf{x}_n, t_n)$ are independent of each other and drawn from the same unknown, but fixed, distribution $p(\mathbf{x},t)$, $(\mathbf{x},t)$ are independently and identically distributed (iid), and thus the joint density of observing the data can be modelled as the product of $p(\mathbf{x}_n, t_n|\boldsymbol{\theta})$ for each $(\mathbf{x}_n, t_n)$:

$$p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{x}_n, t_n|\boldsymbol{\theta}). \tag{2.7}$$

Realising that the target, t_n, depends only on the value of the associated sample, $\mathbf{x}_n$, we obtain:

$$p(\mathbf{x}_n, t_n|\boldsymbol{\theta}) = p(t_n|\mathbf{x}_n, \boldsymbol{\theta})p(\mathbf{x}_n|\boldsymbol{\theta}) \tag{2.8}$$

for a specific value of $\boldsymbol{\theta}$. By combining these we get:

$$p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}) = p(\mathbf{x}_1, t_1, \ldots, \mathbf{x}_N, t_N|\boldsymbol{\theta}) \tag{2.9}$$

$$= \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \boldsymbol{\theta}) p(\mathbf{x}_n, \boldsymbol{\theta}). \tag{2.10}$$

Rather than working with equations of the form of 2.10, it is often easier to work with log likelihoods as they produce summations rather than products and, for cases where the likelihood function is exponential, by taking logs one simplifies the resulting equations. Taking logs, equation 2.10 becomes:

$$\log(p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})) = \sum_{n=1}^{N} \log(p(t_n|\mathbf{x}_n, \boldsymbol{\theta}) p(\mathbf{x}_n|\boldsymbol{\theta})) \tag{2.11}$$

which is simplified to give:

$$\log(p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})) = \sum_{n=1}^{N} \log(p(t_n|\mathbf{x}_n, \boldsymbol{\theta}) + \sum_{n=1}^{N} \log(p(\mathbf{x}_n|\boldsymbol{\theta})). \tag{2.12}$$

Since the log function is monotonic, maximisation of the log-likelihood (2.11) is equivalent to maximisation of the original likelihood function (2.10). Noting that the second summation in (2.12) is not dependent on the targets, t_n, location of the maximum likelihood for $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})$ is equivalent to locating the maximum log-likelihood for the underlying distribution $p(\mathbf{x}_n|\boldsymbol{\theta})$ and distribution $p(t_n|\mathbf{x}_n, \boldsymbol{\theta})$, conditional on t_n, separately. Often, one models only the conditional distribution $p(t_n|\mathbf{x}_n, \boldsymbol{\theta})$ and only maximises the first term of the log-likelihood, $\sum_{n=1}^{N} \log(p(t_n|\mathbf{x}_n, \boldsymbol{\theta}))$. By negating this, we obtain an error, or cost, function.

Kullback-Leibler divergence

The Kullback-Leibler divergence (KL-divergence) [Kullback and Leibler, 1951; Kullback, 1959], also known as the relative entropy, is a non-symmetric measure of the difference, or *divergence*, between two probability distributions, $p(\mathbf{a})$ and $q(\mathbf{a})$. The KL-divergence for $p(\mathbf{a})$ and $q(\mathbf{a})$ is defined as:

$$D_{KL}(p||q) = -\int p(\mathbf{a}) \log\left(\frac{q(\mathbf{a})}{p(\mathbf{a})}\right) d\mathbf{a}. \tag{2.13}$$

KL-divergence acts like a distance measure in that is is always non-negative:

$$D_{KL}(p||q) \geq 0 \tag{2.14}$$

with $D_{KL}(p||q) = 0$ if and only if the distributions $p(\mathbf{a})$ and $q(\mathbf{a})$ are equal almost everywhere [1] . However, as mentioned, the KL-divergence is asymmetric as:

$$D_{KL}(p||q) \neq D_{KL}(q||p) \quad \text{if} \quad p \neq q \tag{2.15}$$

[1] that is, equal everywhere except on a set of measure zero

and is thus not a distance.

Taking $\mathbf{a}$ to denote a feature-target pair $(\mathbf{x}, t)$, we can link Kullback-Leibler divergence and maximum likelihood. If we model $p(\mathbf{a})$ using some family of distributions $q(\mathbf{a}|\boldsymbol{\theta})$, then the maximum likelihood conditioned on $\boldsymbol{\theta}$ will minimise the KL-divergence between $p(\mathbf{a})$ and $q(\mathbf{a})$ as the size of the dataset becomes large. By rearranging 2.13 we get:

$$D_{KL}(p||q) = \int p(\mathbf{a}) \log(p(\mathbf{a}))d\mathbf{a} - \int p(\mathbf{a}) \log(q(\mathbf{a}|\boldsymbol{\theta}))d\mathbf{a}. \tag{2.16}$$

Notice that the first term in equation 2.16 is independent of $\boldsymbol{\theta}$ and thus unaffected by maximisation over $\boldsymbol{\theta}$. The second term can be approximated by the average log-likelihood:

$$\int p(\mathbf{a}) \log(q(\mathbf{a}|\boldsymbol{\theta}))d\mathbf{a} \approx \frac{1}{N} \sum_{n=1}^{N} \log(q(\mathbf{a}_n|\boldsymbol{\theta})) \tag{2.17}$$

where the summation is the likelihood of observing the data $\mathcal{D}_{\text{tr}} = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ sampled from $p(\mathbf{a})$. As $N \to \infty$, so this approximation becomes an equality. Thus, at the limit, selecting $\boldsymbol{\theta}$ to maximise the log-likelihood, based on the summation in (2.17), minimises the KL-divergence $D_{KL}(p||q)$.

We consider two model types; *open* and *closed* models [Bernardo and Smith, 1994]. In open models the true distribution, $p(\mathbf{a})$, lies outside the set of parametrisations of $q(\mathbf{a}|\boldsymbol{\theta})$. In a closed model $p(\mathbf{a})$ lies within the set of parametrisations of $q(\mathbf{a}|\boldsymbol{\theta})$ such that there is some parametrisation whereby $q(\mathbf{a}|\boldsymbol{\theta}) = p(\mathbf{a})$.

For *closed* models this implies that the maximum likelihood estimate will yield the true distribution $p(\mathbf{a}) = q(\mathbf{a}|\boldsymbol{\theta}_{ML})$ when there exists a model parametrisation where $p(\mathbf{a}) = q(\mathbf{a}|\boldsymbol{\theta})$. For open models, where the true distribution lies outside the set of model parametrisations, the KL-divergence $D_{KL}(p(\mathbf{a})||q(\mathbf{a}|\boldsymbol{\theta}_{ML}))$ is minimised. Thus the maximum likelihood estimate $\boldsymbol{\theta}_{ML}$ of the parametrised model $q(\mathbf{a}|\boldsymbol{\theta}_{ML})$ yields a distribution as close as possible, from the set of possible distributions parametrised by $\boldsymbol{\theta}$, to that of the true distribution $p(\mathbf{a})$.

Bayesian modelling

Under the Bayesian paradigm [Bernardo and Smith, 1994; Gelman et al., 2003; Sivia and Skilling, 2006] one models not only $f(\mathbf{x}; \boldsymbol{\theta})$ but also any prior beliefs about the model parametrisation, $\boldsymbol{\theta}$, treated as random variables. Using a prior distribution $p(\boldsymbol{\theta})$, one can encode any prior belief about the parameters before any data is observed. For example, one might have a prior belief that model parameters should be small and encode this in the prior distribution. Making use of Bayes rule, one can then update the prior beliefs about $\boldsymbol{\theta}$ following exposure to new data:

$$p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}) = \frac{p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D}_{\text{tr}})}. \tag{2.18}$$

That is, the *posterior* belief $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})$ about the parameters $\boldsymbol{\theta}$ having observed the data $\mathcal{D}_{\text{tr}}$, is proportional to the *prior* belief about the parameters $p(\boldsymbol{\theta})$ multiplied by the likelihood of observing the data given the parameters $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})$. The normalisation constant $p(\mathcal{D}_{\text{tr}})$

is calculated as the integral of the right hand side of (2.18) with respect to $\boldsymbol{\theta}$:

$$p(\mathcal{D}_{\text{tr}}) = \int p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.19}$$

where $p(\mathcal{D}_{\text{tr}})$ is known as the *evidence* or *marginal likelihood*. Once $\boldsymbol{\theta}$ is integrated out the evidence is the model's average prediction for $p(\mathcal{D}_{\text{tr}})$, the likelihood of the data.

Unlike the maximum likelihood approach, in which the parameters are optimised to find a specific parameter $\boldsymbol{\theta}_{ML}$, the Bayesian approach locates the entire *posterior distribution*. $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})$, for the model parameters based on some training data $\mathcal{D}_{\text{tr}}$. Parameter uncertainty can then be integrated out and predictions made. Thus for binary classification, when we have a likelihood model $p(t|\mathbf{x}, \boldsymbol{\theta})$ the model makes a prediction of probability of membership to the $t = 1$ class for a new input vector $\mathbf{x}'$ as:

$$p(t = 1|\mathbf{x}', \mathcal{D}_{\text{tr}}) = \int p(t = 1|\mathbf{x}', \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})d\boldsymbol{\theta}. \tag{2.20}$$

Since this model averages over all model parametrisations, it is thus more robust to parameter uncertainty arising from the finite available training data [Bernardo and Smith, 1994]. This Bayesian approach gives one the ability to not only express prior beliefs about parameters in the model but allows one to update these beliefs on the basis of new data.

Depending on the forms of $p(\boldsymbol{\theta})$ and $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})$, calculating the posterior distribution for the model parametrisations may be more or less difficult. In particular, for high dimensional $\boldsymbol{\theta}$ parametrisations, $\int p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$ may be hard to calculate.

When the prior is conjugate with the likelihood one finds that the posterior distribution is of the same form as the prior and thus can be written in closed form. A prior is said to be a conjugate prior for a distribution if the prior and posterior distributions are from the same family, and thus the posterior distribution has the same form as the prior distribution. For example, if the likelihood is a Binomial distribution, $Bin(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})$, a conjugate prior on $\boldsymbol{\theta}$ is the Beta distribution and thus the posterior distribution is also a Beta distribution. Other commonly used conjugate prior/likelihood combinations include the Gaussian/Gaussian, Gamma/Poisson, Gamma/Gamma and Gamma/Beta. For a comprehensive report on the many conjugate families see [Fink, 1997].

Unfortunately, a conjugate prior is does not always exist and thus calculation of the posterior is more difficult, requiring evaluation of the evidence in order to find the posterior. As this requires integration over the parameters and, for many real problems, $\boldsymbol{\theta}$ can be quite large, evaluation of the evidence may become hard or impractical.

In the following sections we briefly survey methods for approximating the posterior when a formula for it is not available, in particular maximum a posteriori estimation and sampling methods. Other methods include the Laplace approximation [MacKay, 1992a], in which the posterior is approximated by a Gaussian centred on the mode and a covariance matrix determined by the Hessian at the mode, and Variational methods [e.g. Attias, 1999] and expectation propagation [e.g. Minka, 2001] which depend on minimising the Kulback-Leibler divergence between the posterior and parametrised approximation to it.

2.1.1. Maximum a posteriori estimation

In maximum *a posteriori* (MAP) estimation the single most probable set of parameters from the posterior distribution are returned, the mode of the distribution, giving a single estimate, $\boldsymbol{\theta}_{MAP}$, for the true model parametrisation. Since the mode of a distribution remains unchanged when multiplied by a constant one can locate the mode of a non-normalised distribution which is proportional to the true distribution. Thus, the most probable model parametrisation can be found without requiring evaluation of the evidence as:

$$\boldsymbol{\theta}_{MAP} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D}_{\text{tr}})} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{2.21}$$

Incorporation of a prior over the parameters, $p(\boldsymbol{\theta})$, allows MAP learning to include any pre-conceptions about how the model should be parametrised in the estimation process, unlike in maximum likelihood learning where the prior is not included. Taking logs of the non-normalised posterior distribution:

$$\log p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}) = \log p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \tag{2.22}$$

we see that this is equivalent to penalised maximum likelihood and regularisation methods, whereby the log of the prior is a penalty term. Note that for uniform priors i.e. where all parametrisations are equally likely, the maximum likelihood estimate and maximum *a posteriori* estimate are identical.

The main limitation of this method is that it only gives a single point estimate for the model parametrisation, $\boldsymbol{\theta}_{MAP}$, and thus does not model the full posterior distribution of parameters. As we do not have the full distribution, only this point sample, we cannot integrate out parameter uncertainty. This can be particularly problematic when distributions are multi-modal or have large variance.

2.1.2. Sampling approaches

Sampling methods allow one to draw samples from the posterior distribution in situations where it is not feasible to write the formula for the posterior distribution. Crucially, sampling can often be carried out with only a knowledge of the un-normalised posterior, $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. A few distributions exist, notably Gaussians, uniform, gamma and exponential, from which samples can be drawn directly [Devroye, 1986; Gentle, 2003], but generally sampling must be carried out by drawing samples from one of these distributions and either accepting or rejecting the sample depending on the density function from which we want to sample.

Using a set of S parameter samples one can approximate integrals:

$$p(t = 1|\mathbf{x}', \mathcal{D}_{\text{tr}}) = \int p(t = 1|\mathbf{x}', \boldsymbol{\theta}, \mathcal{D}_{\text{tr}})p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})d\boldsymbol{\theta} \tag{2.23}$$

$$\approx \frac{1}{S}\sum_{i=1}^{S} p(t = 1|\mathbf{x}', \boldsymbol{\theta}_i) \tag{2.24}$$

where $\boldsymbol{\theta}_i$ are samples from $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})$, and thereby reduce parameter uncertainty.

Numerous Monte Carlo approaches exist for sampling from high-dimensional distribu-

tions, of which rejection sampling, Gibbs sampling, and Metropolis-Hastings Markov chain Monte Carlo methods are now discussed, for which a good introduction is given by MacKay [1998].

Rejection Sampling

In rejection sampling one seeks to draw samples from the distribution $p(\boldsymbol{\theta})$, which is too complicated to sample from directly. Assuming some simpler distribution $q(\boldsymbol{\theta})$ is available which we *can* sample from and has the property:

$$\forall \boldsymbol{\theta}, cq(\boldsymbol{\theta}) > p(\boldsymbol{\theta}) \tag{2.25}$$

where c is a constant real number, we can sample from $p(\boldsymbol{\theta})$ with rejection sampling.

First, we sample a $\boldsymbol{\theta}$ from $q(\boldsymbol{\theta})$ and draw a uniformly random variable, u, from the interval $[0, cq(\boldsymbol{\theta})]$. Using the sampled $\boldsymbol{\theta}$, $p(\boldsymbol{\theta})$ is evaluated and the sample rejected or accepted by comparing u with $p(\boldsymbol{\theta})$. If u is greater than $p(\boldsymbol{\theta})$ then $\boldsymbol{\theta}$ is rejected. If $u \leq p(\boldsymbol{\theta})$ then $\boldsymbol{\theta}$ is added to the set of samples $\{\boldsymbol{\theta}_i\}$. This process is repeated, drawing new values for u and $\boldsymbol{\theta}$ each time, until the desired number of samples have been obtained.

Unfortunately, when dealing with high-dimensional $\boldsymbol{\theta}$ the requirement that $cq(\boldsymbol{\theta})$ be an upper bound for $p(\boldsymbol{\theta})$ will result in large values for c and thus poor sample acceptance rates. Additionally, location of a suitable value for c may prove difficult as in many problems we may not be aware of *all* the mode locations for $p(\boldsymbol{\theta})$ or their heights. Therefore, while rejection sampling is useful for low-dimensional problems it is often impractical for sampling from high dimensional distributions.

Gibbs Sampling

Gibbs sampling attempts to ameliorate the problems associated with sampling from high-dimensional distributions by sampling one dimension at at time, leaving the others fixed. Here an example is given for a three element sample.

Assuming a sample $\boldsymbol{\theta}$, such that $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$. A new value for θ_1, θ_1^*, is then sampled from $p(\theta_1^*|\theta_2, \theta_3)$ whilst θ_2 and θ_3 are fixed. θ_2^* is then sampled from $p(\theta_2|\theta_1^*, \theta_2)$ with θ_1^*, from the previous step, and θ_3 fixed. Similarly this is then done for θ_3^*. The sampling is then repeated by updating θ_1^* etc.

Markov Chains

Markov Chains are sequences of non-independent samples, each of which is dependant only on the previous sample i.e. $p(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_{i-2}, \ldots, \boldsymbol{\theta}_0) \equiv p(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i-1})$.

Using a *proposal distribution*, $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$, Markov chain Monte Carlo (MCMC) algorithms propose new $\boldsymbol{\theta}^*$ based on the previous sample, $\boldsymbol{\theta}_{i-1}$. Depending on the choice of proposal distribution, $q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i-1})$ may be significantly different from $q(\boldsymbol{\theta}_{i+1}|\boldsymbol{\theta}_i)$ if it is dependent on the value of the previous sample.

The *Metropolis* algorithm and, its extension, the *Metropolis Hastings* algorithm are the two main MCMC approaches. In Metropolis algorithms [Metropolis et al., 1953] the

proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1})$ is assumed to be symmetric i.e.

$$q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1}) \equiv q(\boldsymbol{\theta}_{i-1}|\boldsymbol{\theta}^*). \tag{2.26}$$

For example, the proposal distribution could be a Gaussian centred on the previously accepted sample:

$$q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1}) = \mathcal{N}(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1}, \sigma^2\mathbf{I}). \tag{2.27}$$

Metropolis Hastings algorithms [Hastings, 1970] extend the Metropolis algorithm to asymmetric proposal distributions i.e. where:

$$q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1}) \neq q(\boldsymbol{\theta}_{i-1}|\boldsymbol{\theta}^*). \tag{2.28}$$

Using the proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1})$ and the previous accepted sample, $\boldsymbol{\theta}_i$, new candidates, $\boldsymbol{\theta}^*$, are sampled. The probability, ρ, of accepting the newly proposed $\boldsymbol{\theta}^*$ is then calculated as:

$$\rho = \min\left(1, \frac{p(\boldsymbol{\theta}^*)q(\boldsymbol{\theta}_{i-1}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}_{i-1})q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1})}\right). \tag{2.29}$$

In situations where the proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1})$ is symmetric, (2.29) simplifies to:

$$\rho = \min\left(1, \frac{p(\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}_i)}\right). \tag{2.30}$$

If $\rho = 1$ then $\boldsymbol{\theta}_i = \boldsymbol{\theta}^*$; otherwise:

$$\boldsymbol{\theta}_i = \begin{cases} \boldsymbol{\theta}^*, & \text{with probability } \rho \\ \boldsymbol{\theta}_{i-1}, & \text{with probability } 1-\rho. \end{cases} \tag{2.31}$$

Note that this is different from rejection sampling where rejected points are discarded and result in no addition to the set of samples. Here, when a newly drawn sample is rejected then the current state is added to the set of samples again.

When running a Metropolis algorithm to obtain S samples, one does not obtain S *independent* samples from the target distribution $p(\boldsymbol{\theta})$; the samples are correlated. Since each sample is correlated with the previous one, the MCMC algorithm may need running for many iterations to draw independent samples from $p(\boldsymbol{\theta})$. Alternatively, one might only use every n^{th} sample from the chain.

Typically, the Markov chain is initialised with an arbitrary value, $\boldsymbol{\theta}_0$, and the practice of "burning-in" carried out in which some initial number of samples are drawn until the chain is judged to be stationary and sampling from the target distribution. This is done so as to minimise the effects of the initial samples on the posterior distribution. Consider a target distribution $p(\boldsymbol{\theta}) = \mathcal{N}(0, 1)$, and a Markov chain initialised at 10^6. Over a few initial samples the chain should move towards drawing samples in the region around 0. If one includes the initial samples drawn from round 10^6 when estimating the mean it will have a large bias on the estimated value. Were one to use the chain to draw an infinite number of samples then the effect of these initial samples would be minimised. However, in practice, one can only draw some finite number of samples. Thus, once the chain is judged to be sampling from the target distribution, one discards these initial "burn-in" samples.

Determination as to whether or not the chain has "burned-in" can be difficult and it is worth noting that for some sampling problems this burn-in process can take longer than assumed; for example when sampling parameters for a multi-layer perceptron [Lampinen and Vehtari, 2001]. Once these burn-in samples have been discarded, the remaining set of accepted $\boldsymbol{\theta}_i$ should represent samples from the target distribution $p(\boldsymbol{\theta})$.

2.1.3. Summary

In this section methods for supervised learning of model parametrisations, $\boldsymbol{\theta}$, have been discussed, specifically maximum likelihood and Bayesian inference. Methods for approximating the Bayesian posterior have been examined which allow for the incorporation of prior belief regarding model parametrisations into the learning process. For cases where conjugate priors are not used or do not exist, formulation of the posterior distribution is hard due to the need for evaluation of the evidence and so methods for overcoming this have been investigated. Maximum *a posteriori* learning finds $\boldsymbol{\theta}_{MAP}$, the mode for the distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})$, but does not model the full posterior distribution. Monte Carlo sampling methods have been shown to allow one to sample parametrisations from the posterior and allow marginalisation of parameter uncertainty. They do however often require large numbers of iterations in order to obtain sufficiently many independent samples.

We note that these sampling methods require evaluation of the likelihood, something that is not always possible. Indeed, for "hard" classifiers, where only the class labelling is given with no measure of certainty of the classification, no likelihood is available. For situations where likelihoods are not available, Approximate Bayesian Computation (ABC) implementations of these sampling methods exist [Marjoram et al., 2003]. ABC methods are discussed later in this thesis in chapter 5.

Having obtained a model parametrisation, $\boldsymbol{\theta}$, with which one can make predictions, y, the next step is to evaluate its performance. In the following section a selection of commonly used methods for classifier performance evaluation are discussed along with their various strengths and weaknesses.

2.2. Measuring classifier performance

Given a model parametrised by $\boldsymbol{\theta}$ that makes predictions, y, for input feature vectors $\mathbf{x}$, such that $y = f(\mathbf{x}|\boldsymbol{\theta})$, one seeks to evaluate the model performance on a dataset consisting of N input-target pairs, $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, using some metric. This dataset may be the training data, $\mathcal{D}_{\text{tr}}$, or some previously unseen test dataset, $\mathcal{D}_{\text{te}}$. In particular, one is interested in the generalisation capabilities of said model i.e. how well it will perform on previously unseen data. Depending on the domain to which the model is being applied, the choice of performance measure may vary. For instance, in classification problems there may be different costs associated with failing to correctly classify feature vectors belonging to different classes e.g. it would be more costly to fail to detect cancer in a cell biopsy than mis-classifying a healthy cell as cancerous when screening patients.

Since the focus of this thesis is classification, the measures discussed here all concern classification performance. In this section a number of the most common methods for analysing performance are discussed along with their various merits and flaws. For all the

metrics presented here knowledge of the set of true classification targets, $\mathbf{t}$, is required.

When performing classification we want to assign an input feature vector $\mathbf{x}$ a *discrete* class label. Depending on the classifier used, the prediction made, $f(\mathbf{x}|\boldsymbol{\theta})$, may either be a discrete class labelling, a score or a probability of class membership. Classifiers yielding discrete *class labels* as their predictions are referred to as *hard* classifiers, examples of which include nearest neighbour classifiers [Cover and Hart, 1967], support vector machines [Boser et al., 1992; Vapnik, 1998; Schölkopf et al., 1999] and decision trees [Breiman, 1984]. Classifiers assigning a score for class membership or a probability of an example belonging to a particular class are referred to as *soft* classifiers. Examples include logistic regression [Myers and Forgy, 1963; Bishop, 2006], multi-layer perceptrons [Bishop, 1995] and relevance vector machines [Tipping, 2000; Faul and Tipping, 2002; Tipping and Faul, 2003].

Since the targets we are comparing against take the form of discrete labels, in order to evaluate model performance the predictions also need to be in the form of a discrete labelling. Hard classifiers already provide this as their output. Soft classifiers, however, need their outputs converting to a discrete class label. This is done through the use of the *decision function*. Taking the output to be a probability of an input belonging to the $t = 1$ class we transform it to give y, the prediction, as:

$$y = \begin{cases} 1, & \textit{if } f(\mathbf{x}|\boldsymbol{\theta}, \mathcal{D}_{\text{tr}}) > \lambda \\ 0, & \texttt{otherwise} \end{cases} \tag{2.32}$$

where λ is the *decision threshold* [Duda et al., 2001]. Depending on how certain the user wants the classifier to be in assigning inputs to a class the value of λ can be adjusted accordingly. Increasing λ requires the prediction of class membership to be increasingly certain in order to assign $\mathbf{x}$ to the $t = 1$ class. Decreasing it makes it easier for $\mathbf{x}$ to be assigned to the $t = 1$ class. For soft classifiers, when evaluating performance at a specific decision threshold, we can write the label prediction y as a function of λ:

$$y = f(\mathbf{x}|\boldsymbol{\theta}, \lambda, \mathcal{D}_{\text{tr}}). \tag{2.33}$$

We discuss the costs of misclassification later in this section.

A discussion of methods for comparing these predicted class labels with the true class labels to evaluate classifier performance follows. Methods discussed include the Confusion Matrix, Accuracy and Error rates, Receiver Operating Characteristic (ROC) curves and Area Under the ROC curve.

2.2.1. Confusion Matrix

The confusion matrix is a tool used to visualise classifier performance in the form of a tabulation, from which a number of different performance measures can be calculated. Each column of the confusion matrix represents the true, i.e. actual, class and each row the class predicted by the model being evaluated. The elements of the confusion matrix are counts of the number of instances that the corresponding column's class was predicted as belonging to the corresponding row's class. By visualising the performance in this way

		True		
		■	★	▲
Predicted	■	6	4	0
	★	4	5	1
	▲	0	1	9
Totals		$N_{■}$	$N_{★}$	$N_{▲}$

Table 2.1.: Example confusion matrix for classification between three different shapes

		True	
		p	**n**
Predicted	**p**	**T**rue **P**ositives	**F**alse **P**ositives
	n	**F**alse **N**egatives	**T**rue **N**egatives
Totals		N_{+ve}	N_{-ve}

Table 2.2.: Confusion matrix

one can see if a classifier is confusing, or struggling to discriminate between, two classes.

A confusion matrix for an illustrative problem of shape classification is shown in table 2.1. A classifier is being used to classify input shapes as either being squares, ■, stars, ★, or triangles, ▲. The true class labels for the inputs are known and thus the confusion matrix can be constructed using these labels and the predictions from the model being evaluated.

In this example there are 30 shapes being classified, 10 of each type; 10 squares, 10 stars and 10 triangles. Examining the confusion matrix, we can see that of the 10 triangles, 9 are correctly classified as triangles and 1 is misclassified as a star. Of the squares 6 are correctly classified as squares but 4 are misclassified as stars while 4 of the stars are misclassified as squares and 1 is misclassified as a triangle. From these results we can deduce that the classifier being evaluated is able to distinguish the triangle class from the others but is struggling to discriminate between, i.e. separate, the square and star classes.

For binary classification, the subject of this thesis, we consider two classes typically denoting one as the positive class and the other as the negative class. The way round in which the classes are labelled as either the positive or negative class does not matter, but in general we denote the class in which we are interested as the positive class and the baseline class as the negative class. For example, when using aircraft trajectory pairs to predict collisions we are interested in the class consisting of trajectory pairs leading to collisions and would thus denote that as the positive class.

Given a classifier parametrisation, and decision threshold for smooth classifiers, a set of data divided between two classes is classified. Using knowledge of the true classification targets one constructs the confusion matrix shown in table 2.2.

The entries of the constructed confusion matrix are counts of the number of True Positives, False Positives, True Negatives and False Negatives, as seen in table 2.2. The perfect classifier would be one for which the number of True Positives was equal to the number of positive class members, $TP = N_{+ve}$, and the number of True Negatives was equal to the number of negative class members, $TN = N_{-ve}$. Note that maximising the number of True Positives thus minimises the number of False Negatives and minimising the number of False Positives maximises the number of True Negatives.

Using the entries in this confusion matrix one can calculate a number of standard performance measures which are now discussed.

Accuracy and Error rates

Accuracy and error rates are perhaps the most commonly used and simplest methods for evaluating classifier performance. For accuracy, we are concerned with correctly matching predicted class and true class and thus the True Positive, TP, and True Negative, TN, entries of the confusion matrix. Using these, accuracy is calculated as:

$$Accuracy = \frac{TP + TN}{N_{+ve} + N_{-ve}}. \tag{2.34}$$

Conversely, for error rates we are interested in misclassification i.e. when the predicted and target class for a sample do not match. Therefore to calculate error rates the entries in the confusion matrix for False Positives and False Negatives are used:

$$Error = \frac{FP + FN}{N_{+ve} + N_{-ve}}. \tag{2.35}$$

If either the accuracy or error rate is already calculated one can easily calculate the other without needing to consult the confusion matrix since:

$$Accuracy = 1 - Error. \tag{2.36}$$

As only the model predictions and true targets are required, accuracy is straightforward to assess. However, all classifications make an equal contribution to the calculation, regardless of originating class and any associated costs. As a result there are several concerns when using these measures to analyse performance.

Consider a two class problem for which there are 10 examples of the positive class and 990 examples of the negative class. By simply classifying everything as members of the negative class a classifier will achieve 99% accuracy while completely failing to model the positive class. This can be particularly concerning when correct classification of the rarer positive class examples is important e.g. aircraft trajectories that will result in collisions.

Another problem is encountered when class proportions differ between training and test datasets. Such changes may be as a result of data availability for training consisting of a non-representative class proportion or, more likely, when some change in some external influence results in different class frequencies when it comes to testing. Consider again the classifier that always classifies inputs as negative. Given training data consisting of 90 negative examples and 10 positive examples a training accuracy of 90% will be obtained. In the test dataset, however, the class proportions are different: 50 negative examples and 50 positive examples giving a test error of 50%. The classifier is still working in the same way, but the change in proportions result in a very different apparent performance.

The final concern voiced here is that of misclassification cost. In accuracy and error rates, all misclassifications are treated equally regardless of their true class. This is acceptable if misclassifications have no difference in cost on a class-wise basis, but a problem if there is. For example, if one is trying to classify cells for the presence of cancer indicators that will lead to further tests, it is considerably more costly to mis-classify a cancerous

cell as healthy than mis-classify a healthy cell as potentially cancerous. A classifier that is better at identifying cancerous cells is preferable to one that is better at identifying non-cancerous ones.

True and False Positive Rates

An alternative to measuring overall performance in the form of accuracy or error, is to use class-wise performance rates in the form of true and false positive rates. Having constructed the confusion matrix, one can calculate the true positive rate, T, and false positive rate, F, as:

$$T = \frac{TP}{N_{+ve}}, \tag{2.37}$$

$$F = \frac{FP}{N_{-ve}}. \tag{2.38}$$

Notice that, unlike accuracy and error rates, these values are only dependent on the number of examples from that class. Thus (2.37) provides a measure of successful classification of positive class examples and (2.38) a measure of misclassifying negative class examples as positive class members.

Using these measures in preference to accuracy or error rates has several advantages. Firstly, since T and F are only dependent on their own class, differing class proportions between training and test data will not result in a change in these measures. Secondly, one can determine from these measures if a model is struggling to separate the two classes. Ideally, we would like to have a classifier with a true positive rate of 1 and false positive rate of 0, giving perfect class separation.

Consider again a classifier that always predicts the positive class. The training data consists of 90 positive examples and 10 negative examples and the test dataset consists of 50 negative examples and 50 positive examples. Using accuracy we would obtain a 90% training rate and a 50% test rate but would be unable to determine where the classification difficulty had arisen from. Analysing performance in terms of true and false positive rates the classifier will achieve a true positive rate of 1 and a false positive rate of 1 on the training data. Applying this classifier to the test data will also achieve a true positive rate of 1 and false positive rate of 1. Whereas accuracy has been affected by the changes in class proportions, true and false positive rates have not.

In many classification problems there are differing costs associated with each type of mis-classification i.e. a false negative might be more costly than a false positive or vice-versa. If we know the values associated with these costs then we can calculate the classification cost for a classifier on a particular dataset as:

$$\text{Cost} = F \times c_{FP} + (1 - T) \times c_{FN} \tag{2.39}$$

where c_{FP} is the cost associated with a false positive and c_{FN} is the cost associated with a false negative, assuming the cost of a correct classification to either class is zero. When we have a precise cost function such as 2.39 and the costs are known then, given a set of classifiers and a dataset on which to evaluate their performance, we can rank classifiers according to their cost and select the minimum. Indeed, we can use this cost function to

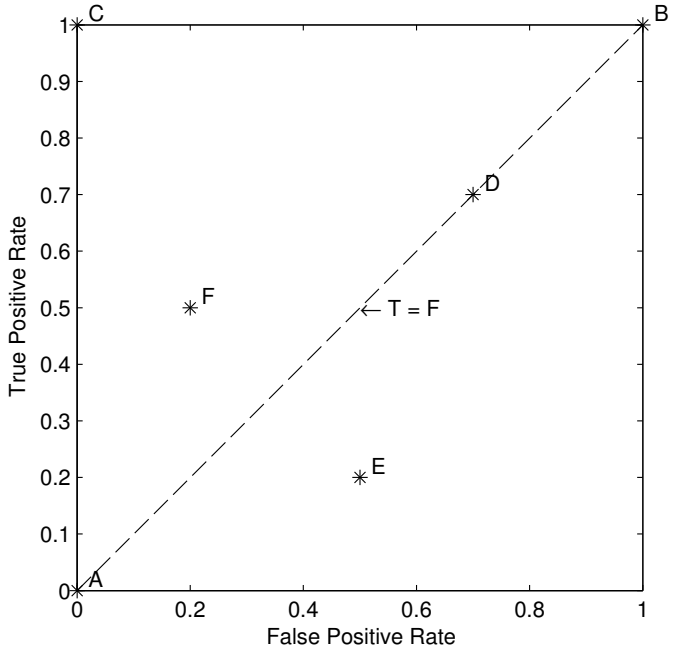


Figure 2.1.: An example ROC graph on which the performances of a number of different classifiers are plotted for comparison.

locate the best decision threshold for a classifier as the one giving the minimum cost [Duda et al., 2001]. It is, however, rare to know the exact costs associated with misclassifications.

True and false positive rates are important in another method of performance evaluation, Receiver Operating Characteristic (ROC) graphs, another way of visualising classifier performance commonly used to assess model performance over the full range of misclassification costs. ROC graphs are discussed in the next section.

2.2.2. Receiver Operating Characteristic graphs

As we have seen, classification error alone is a poor metric for analysing classifier performance as it fails to account for unbalanced class proportions and unequal misclassification costs. One common alternative technique to visualise, organise and select classifiers based on their performance is the use of Receiver Operating Characteristic (ROC) graphs [Hanley and McNeil, 1982; Provost and Fawcett, 1997, 1998; Fawcett, 2006]. Typically, these are used to display the trade-offs between true and false positive rates as misclassification costs, and by extension decision thresholds, are varied allowing the user to select the operating point best meeting their needs with knowledge of the available alternatives. Examples of their use include: analysis and comparison of radar based systems for the identification of Japanese aircraft during World War II [Green and Swets, 1966], visualisation of medical imaging techniques [Swets, 1979; Hanley and McNeil, 1982], comparison of different classification algorithms [Spackman, 1989], evaluation of new radiology techniques [Obuchowski, 2003] and the visualisation and comparison of the performance of safety-critical system parametrisations [Everson and Fieldsend, 2006b; Reckhouse et al., 2010].

In figure 2.1 an example ROC graph is plotted in which each point corresponds to the performance of a different classifier. False positive rates are displayed on the horizontal

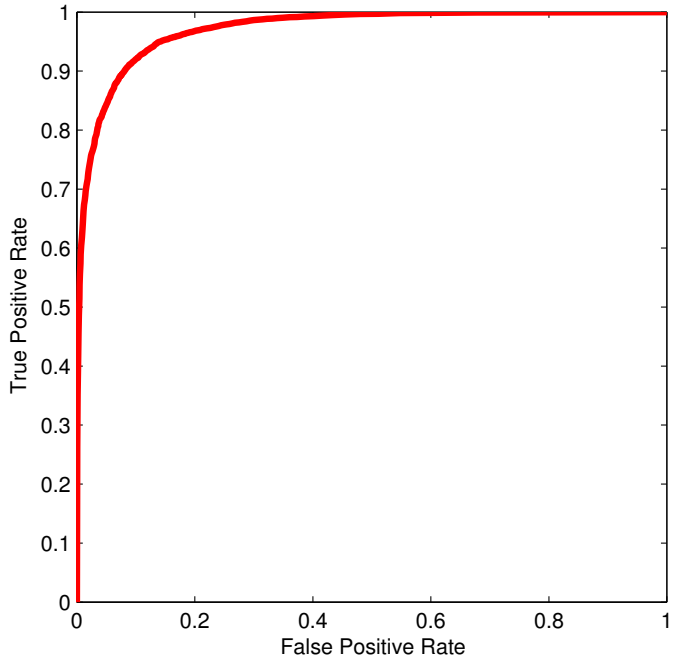


Figure 2.2.: An example ROC curve generated by varying the decision threshold for a soft classifier. The $T = 1, F = 1$ end of the curve corresponds to a decision threshold of $-\infty$ whereby every sample is classified as a member of the positive class. The $T = 0, F = 0$ end of the curve corresponds to a decision threshold $+\infty$ whereby every sample is classified as a member of the negative class.

axis and true positive rates on the vertical axis. Classifiers located at the origin, classifier A, are those that classify everything as negative whilst those at the top right, classifier B, location always issue a classification of positive class membership. The ideal, or "perfect", classifier is one whose performance gives a false positive rate of 0 and true positive rate of 1, locating it at classifier C. Classifiers such as classifier D, lying on the true positive rate equals false positive rate line, perform as well as random guessing with a bias towards a certain class propensity. Any classifier whose performance places it below the $T = F$ line thus performs worse than random guessing. Such classifiers have learnt information about the classes but are applying it in the wrong direction (classifier E). By simply reversing the classifier predictions one can move it from this lower right triangle of the ROC graph to the upper left; classifier F is generated by reversing the predictions of classifier E.

For hard, binary, classifiers each model parametrisation, $\boldsymbol{\theta}$, will map to a *single* point on the ROC graph. For soft classifiers, where the prediction takes the form of a score or probability of class membership, one can sweep out a full ROC *curve* for a classifier by varying the decision threshold, λ, from 0 to 1 to find all the true and false positive rate combinations as seen in figure 2.2. By varying the decision threshold we obtain performance under different misclassification costs and, thus, this represents moving from the case where it is infinitely more costly to misclassify something as a member of the *negative* class, at the true and false positive rates equal to 1 end of the ROC curve, to the case where it is infinitely more costly to misclassify something as a member of the *positive* class, at the true and false positive rates equal 0 end of the ROC curve.

Using an ROC graph one can not only visualise the performance of a single classifier, but can also compare the performance of multiple different classifiers by plotting their

performances on the same graph as per figure 2.3. When comparing two classifiers X and Y, if the operating point for classifier X lies North-West of the operating point for classifier Y, classifier X performs better than classifier Y. Thus, in figure 2.3 we say that classifier A performs better than classifier B. Classifier A performs better than classifier C in terms of false positive rates but classifier C performs better than classifier A in terms of true positive rates.

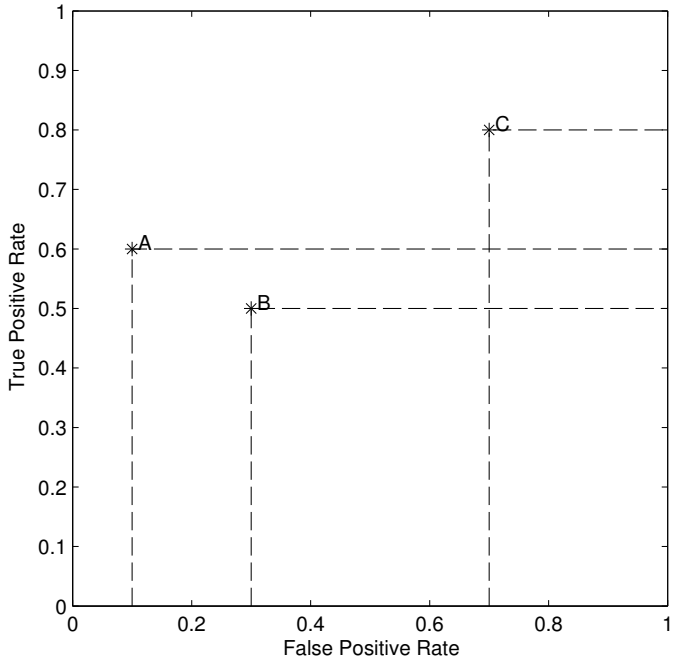


Figure 2.3.: Comparing ROC graph operating points.

Letting the prior probability of a positive class sample be $p(t = 1)$ and the prior probability of a negative class sample be $p(t = 0)$ such that $p(t = 0) = 1 - p(t = 1)$, then the cost of misclassification by the classifier represented by the point (F, T) in ROC space is:

$$\text{Cost} = p(t = 1) \times (1 - T) \times c_{FN} + p(t = 0) \times F \times c_{FP}. \tag{2.40}$$

where c_{FP} and c_{FN} are the the costs associated with false positive and false negative errors respectively. Therefore two ROC points (F_1, T_1) and (F_2, T_2) have the same performance if:

$$\frac{T_2 - T_1}{F_2 - F_1} = \frac{p(t = 0)c_{FP}}{p(t = 1)c_{FN}}. \tag{2.41}$$

This equation defines the slope of an *iso-performance* line [Provost and Fawcett, 1997]. A set of classifiers corresponding to points on the same iso-performance line have the same expected cost. Each different combination of class and cost proportions will define its own set of iso-performance lines. Lines located "more North-West" in the ROC space with higher true positive rate intercept values are better as they correspond to classifiers with lower expected cost.

When using ROC curves to analyse classifier performance we obtain a visualisation of a classifier's ability to distinguish between classes over the range of misclassification costs. Calculation of area under the ROC curve provides a measure of the ability of a classifier, or set of classifiers, ability to separate out two classes [Hanley and McNeil, 1982; Hand

and Till, 2001]. Thus, in the next section methods for calculating this area are discussed.

Area under the ROC curve

The area under the ROC curve (AUC) provides a single value between 0 and 1 which can be used to evaluate a classifier's ability to distinguish between two classes [Hanley and McNeil, 1982; Hand and Till, 2001; Fieldsend and Everson, 2008a]. For any pair of samples, one from each of the classes, the AUC gives the probability of correctly ranking the positive class sample higher than the negative class sample [Hanley and McNeil, 1982]. The larger the AUC for a classifier, the more likely it is that it will successfully distinguish between the two classes; the classifier obtaining perfect class separation has a point on its ROC curve located at $(F = 0, T = 1)$ on the ROC graph, resulting in an AUC of 1.

Here, a number of different methods for calculating this area are discussed. To begin with we discuss methods for calculating the AUC for an ROC curve generated by either a single classifier or from the ROC operating points from several different classifier parametrisations before discussing the Mann-Whitney-Wilcoxon test which allows us to calculate the AUC for a single classifier without the need to calculate all true and false positive rate combinations.

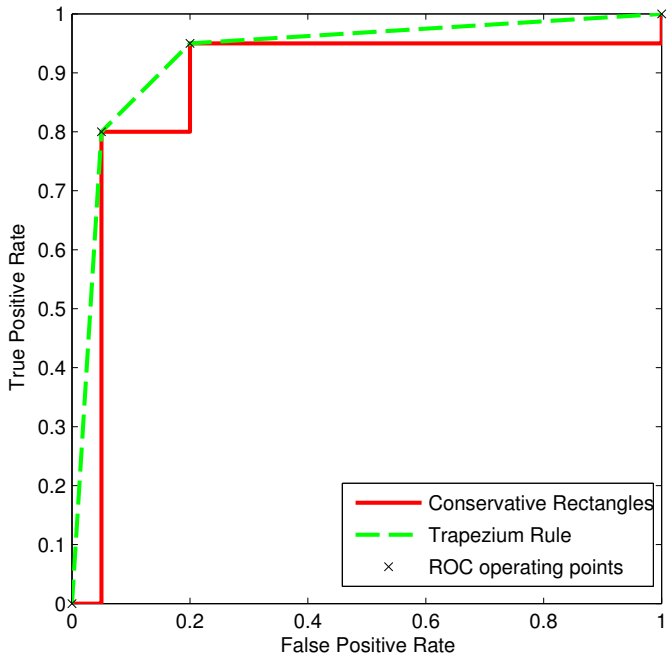


Figure 2.4.: Different methods for estimating AUC based on a set of ROC points.

In figure 2.4 an example of an ROC curve is shown in which each of the located ROC operating points have been marked with the black x's. These operating points can represent the performance of a single classifier using different decision thresholds. Alternatively, each point might be the performance of a *different* classifier. Two different methods are available for calculating the area under the ROC curve: "conservative" rectangles, marked on the ROC graph by the red line, and the trapezium rule, marked by the green dashed line.

In the conservative rectangles method, the AUC is calculated as the sum of a set of rectangles defined by the points on the ROC curve. The conservative rectangles AUC,

as marked by the red line, calculates the AUC as being the sum of the area *beneath* the points in the ROC graph. As seen from the figure, calculating the AUC using this method gives a conservative estimate for its value. Note that as the number of points on the ROC curve increases, so the estimates obtained by the rectangle methods will become closer to the true AUC.

The trapezium rule calculates the AUC as a sum of trapezoids defined by the points on the ROC curve. Its area is marked on the ROC curve by the dashed green line between ROC points. In this case, the AUC is calculated on the basis that one can obtain ROC performance along the green line between points using a random weighted selection between the classifiers at either end. For example, we might randomly select one classifier 50% of the the time and the other 50% of the time to obtain a new ROC point halfway between those of the two classifiers being selected between. Using this, we obtain a larger estimate for the AUC than the conservative rectangles method but rely on random weighted selection between classifiers in order to do so.

Of these methods the sum of conservative rectangles is preferred as it only calculates the AUC in terms of the available ROC points rather than obtaining an overestimate suggesting better performance than is obtained. While the trapezium rule produces an estimate of the AUC it relies on randomly weighted selection between classifiers to obtain that area and in many classification problems such random selection between classifiers is undesirable. For example, in the case of a safety critical system, it would be unacceptable for loss of lives to have resulted from the random selection of classifier and thus those ROC points would not be generated. Therefore, we do not use this measure.

When presented with the predictions from a *single* soft classifier, we can calculate the ROC curve for that classifier by varying the decision threshold to obtain all possible true and false positive rate combinations. In this instance, rather than calculating the AUC by calculating and summing the areas of each of the rectangles defined by the ROC points, we can instead use the more efficient Mann-Whitney-Wilcoxon two sample test to calculate the AUC [Hand and Till, 2001].

To begin with, the classifier being evaluated is used to produce predictions of membership of the negative class for each of the samples in a dataset. Let y_i^- be the estimated probability of the i^{th} negative class sample belonging to the negative class. Similarly, let y_i^+ be the estimated probability of the i^{th} positive class sample belonging to the negative class. The predictions are then ranked in increasing order as $\{y_1^+, \ldots, y_{N_{+ve}}^+, y_1^-, \ldots, y_{N_{-ve}}^-\}$ where N_{+ve} is the number of positive class samples and N_{-ve} is the number of negative class samples.

Let r_i be the rank of the i^{th} negative class point. There are $(r_i - i)$ positive class samples whose probability of membership to the negative class is less than the i^{th} negative class sample. Summing over the negative class samples, we see that the total number of sample pairs, one from each of the classes, in which the negative class sample has a smaller predicted probability of belonging to the negative class than the positive class sample is:

$$\sum_{i=1}^{N_{+ve}} (r_i - i) = \sum_i r_i - \sum_i i = S_{-ve} - N_{-ve}(N_{-ve} + 1)/2 \tag{2.42}$$

where S_{-ve} is the sum of the ranks of the negative class samples. As there are $N_{-ve}N_{+ve}$

such sample pairs in total, one can estimate the probability that a randomly chosen positive class sample has a lower predicted probability of membership to the negative class than a randomly chosen negative class sample as:

$$A = \frac{S_{-ve} - N_{-ve}(N_{-ve}+1)/2}{N_{-ve}N_{+ve}}. \tag{2.43}$$

Since the area under the ROC is composed of elements of area $(r_i - i)/N_{-ve}N_{+ve}$, with one element for each point in the negative class, one can see that adding these over the negative class points results in A as above.

Calculation of AUC using this method is straightforward and, importantly, does not require generation of a classifier's ROC curve using lots of thresholds prior to calculation. Note that one can only apply this AUC calculation to the ROC curve for a *single* classifier at a time. For situations where different classifiers are used at different ROC operating points (such as in chapter 3) methods such as the sum of conservative rectangles method must be used. When using the Mann-Whitney-Wilcoxon method to calculate the AUC for soft classifier predictions it is equivalent to calculating the AUC using the conservative rectangles method [Hanley and McNeil, 1982].

Summary

In this section methods for analysing and comparing model performance have been discussed. Construction of the confusion matrix for hard classifiers or soft classifiers at a specific decision threshold, λ, gives a simple visualisation of classifier performance in the form of a table, allowing one to inspect the classifier's performance.

Using the confusion matrix one can calculate a number of standard measures, of which accuracy/error rates and true and false positive rates have been discussed. Accuracy provides a single measure of classifier performance in the form of a rate or percentage but fails to account for class imbalances and unequal misclassification costs. True and false positive rates give one a measure of how well a classifier is performing on a class-wise basis. Because they are only concerned with one class at a time true and false positive rates are not subject to problems with class imbalance and allow one to consider performance in terms of misclassification costs on a class-wise basis.

Receiver Operating Characteristic graphs are a commonly used method for visualising classifier performance over the range of misclassification costs. Using true and false positive rates to plot points on the graph, different classification rules can be compared over a range of misclassification costs – these can be obtained either by varying the decision threshold for a soft classifier or from different classifier parametrisations. By calculating the area under the ROC curve one obtains a measure of how well a classifier, or set of classifiers, is able to distinguish between two classes. When calculating this measure for a set of different classifiers one has to use either the sum of rectangles or trapezium rule. Should the AUC be being evaluated for a single classifier's ROC curve then this can be done efficiently using the Mann-Whitney-Wilcoxon test.

Typically, ROC curves are evaluated following classifier training and thus performance levels are already set. When large amounts of data are available for training and the correct model type is used, maximum likelihood methods will learn good parametrisations

and thus produce optimal ROC curves. Unfortunately, this is rarely the case with training data commonly limited in size and/or insufficient knowledge about the underlying data distribution available to enable selection of the correct model type.

In this thesis we therefore introduce a number of methods in which the optimisation of ROC curves is incorporated into the training of binary classifiers. In chapter 3 we optimise ROC curves for relevance vector machine classifiers during training, obtaining the best ROC curves for RVM classifiers over a range of complexities. In chapter 4 we optimise the AUC values for RVM classifiers in order to obtain the best class discrimination possible for a particular RVM model complexity on a number of gene selection problems, successfully identifying genes associated with various genetic conditions. In chapter 5 we use AUC as a distance measure in an Approximate Bayesian Computation Markov chain Monte Carlo (ABC MCMC) algorithm, in which we demonstrate the ability to reproduce the performance of the Bayes optimal ROC curve using the averaged prediction of the ABC MCMC sampled classifiers.

In the next section we discuss sparse models. By learning sparse models, we hope to avoid overfitting and obtain good generalisation performance. In particular, we focus on the Relevance Vector Machine [Tipping, 2001, 2000], which casts sparse model learning in a Bayesian framework.

2.3. Sparse Kernel Models

In real-world problems there is commonly signal noise (regression) and/or class overlap (classification) present which may be learnt by an overly flexible model during the training process. This is referred to as *overfitting* and leads to reduced, often poor, generalisation performance. As a result, there has been increasing interest in learning sparse models for machine learning problems by penalising overly-complex models during the training process. Methods for learning these sparse models are referred to as *regularisation* methods and typically involve a penalty term, calculated based on the model parametrisation, being added to the error being minimised during training.

In recent years there has been increasing interest and research into algorithms for learning sparse models for machine learning problems. In particular, L1-regularisation methods have proven effective for preventing overfitting. Examples include ridge regression [Tikhonov and Arsenin, 1977] and LASSO [Tibshirani, 1996], typically used to control the magnitude of weights in artificial neural networks [Bishop, 1995], Elastic Net [Zou and Hastie, 2005], L1-regularised generalised linear models [Park and Hastie, 2007] and sparse classifiers such as the support vector machine [SVM, Boser et al., 1992; Vapnik, 1998; Schölkopf et al., 1999].

In particular, we are interested in sparse kernel models. In these models non-linear kernels are centred on each of the samples in the training data mapping the data from a lower dimensional space to a higher dimensional space in the hope that they will be linearly separable in the higher dimensional space. Such models make predictions, $y(\mathbf{x}; \mathbf{w})$, using

models typically of the form:

$$y(\mathbf{x}; \mathbf{w}) = \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}) \tag{2.44}$$

where $\phi_m(\mathbf{x})$ is some basis function applied to the input vector $\mathbf{x}$ and w_m is the weight associated with the m^{th} basis function, $\boldsymbol{\phi}_m$. Given a training set of sample-target pairs, $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, sparse kernel models attempt to learn weightings $\mathbf{w} = (w_1, \ldots, w_M)^T$ that lead to good prediction generalisation with the majority of the w_m set to zero. By setting weights to zero, as opposed to limiting them to small non-zero values, model complexity is reduced and generalisation capability of the learnt models should be improved due to the reduction in potential overfitting. The added benefit is that, on model deployment, computational overheads are reduced due to the reduction in the number of calculations to be carried out. This can be an important consideration when dealing with mobile devices where computing resources are limited and in systems requiring real-time prediction.

One of the most popular approaches for learning sparse kernel models is the Support Vector Machine [SVM, Boser et al., 1992; Vapnik, 1998; Schölkopf et al., 1999]. In the SVM predictions are made as:

$$y(x; w) = \sum_{n=1}^{N} w_n K(\mathbf{x}, \mathbf{x}_n) + w_0 \tag{2.45}$$

where $K(\mathbf{x}, \mathbf{x}_n)$ is a *kernel* function, defining a basis function, $\phi_n(\mathbf{x}) \equiv K(\mathbf{x}, \mathbf{x}_n)$, centred on each $\mathbf{x}_n$ in $\mathcal{D}_{\text{tr}}$, w_0 is the bias and $\{w_n\}_{n=1}^{N}$ a sparse set of weightings for kernels centred on each of the N elements in $\mathcal{D}_{\text{tr}}$. Since there are multiple hyperplanes which one might use to separate and classify data, a sensible choice would be to locate the one that gives the largest separation, or margin, between the classes. Therefore, in the SVM we look to select basis functions, in the form of support vectors, defining the hyperplane for which the distance from it to the nearest data point on each side is maximised.

Before training the SVM, a number of parameters need defining. For classification an error/margin trade-off parameter, C, has to be defined and in regression an "insensitivity parameter", ϵ, is also required. C controls how wide the margin between classes should be while ϵ effectively places a "tube" around the learnt function within which errors are not penalised. An illustration of SVM classification can be seen in figure 2.5 and SVM regression in figure 2.6.

Selection of appropriate values for C and ϵ is usually carried out using cross-validation and thus expensive to perform. Having set these parameters, training of the SVM focuses on locating a sparse set of kernel weightings which simultaneously minimises the error on the training data while maximising the margin. The result is an effective method of controlling complexity and avoiding overfitting which leads to the learning of sparse models with good generalisation capability, dependent on only a subset of the kernel functions. The selected kernels are centred on particular training data examples, $\mathbf{x}_n$, referred to as "support vectors", lying on either the margin or the "wrong" side of it.

While the SVM succeeds in learning sparse models with good performance, it has a num-

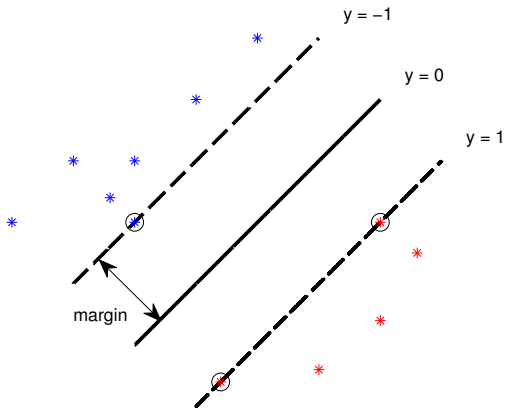


Figure 2.5.: Example of SVM classification. Data points with circles around them are support vectors.

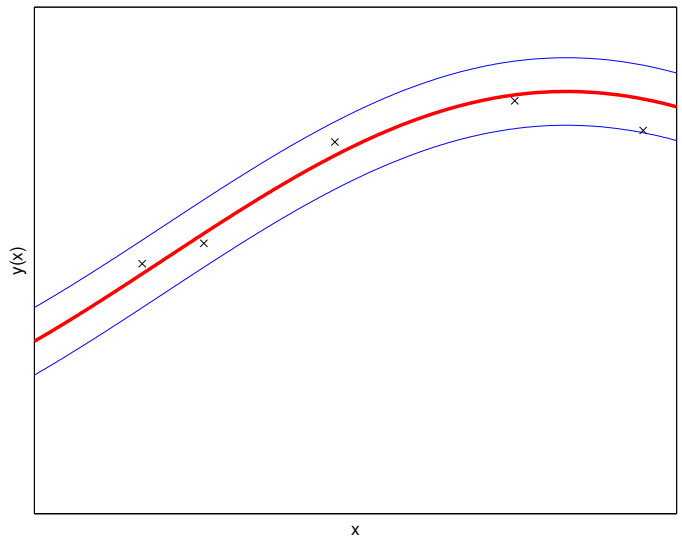


Figure 2.6.: Example of SVM regression showing the regression curve along with ϵ-insensitive 'tube', marked here as the region between the blue lines.

ber of disadvantages. First, and foremost, the SVM does not make use of a probabilistic framework and, as a result, the predictions made are non-probabilistic. In regression these take the form of a point estimate and in classification a hard binary prediction. The desirable prediction, however, would be one for which we have an estimate of the uncertainty of the prediction. For regression this could be in the form of a predictive probability density around a point estimate, while in classification this would take the form of a probability of class membership, $p(t|\mathbf{x})$. Platt [2000] obtains posterior probability estimates from SVMs through post-processing but Tipping [2001] argues these are unreliable.

Secondly, while sparse, SVM models make "unnecessarily liberal use" [Tipping, 2001] of basis functions. This is because the number of basis functions used tends to grow linearly with the number of examples in the training data. In order to reduce the complexity of these models, approaches have been proposed based on post-processing [Burges, 1996; Burges and Schölkopf, 1997] and data partitioning for basis function selection [Graf et al., 2004].

Thirdly, prior to training the SVM, the trade-off parameter, C, and insensitivity pa-

rameter, ϵ, need setting, usually on the basis of expensive cross-validation. It would be desirable for these to set automatically and not require the use of cross-validation to determine their optimal values.

Finally, the kernel functions $K(\mathbf{x}, \mathbf{x}_n)$ used by the SVM must satisfy Mercer's condition, requiring that the chosen kernels used must be positive semi-definite. In [Smola et al., 1998; Schölkopf, 2001] it is shown that this can be relaxed to include conditionally positive kernels.

The Relevance Vector Machine [RVM, Tipping, 2001, 2000] is a state-of-the-art machine learning tool that casts the task of selecting a sparse weighting for (2.44) in a Bayesian framework and avoids the limitations associated with the SVM methodology. Unlike the SVM, no parameters need setting in the RVM to control model sparsity, instead the model complexity is controlled through automatic relevance determination [ARD, MacKay, 1992b] which removes basis functions from the model for which there is no evidence in the data, i.e. sets their associated weights to 0. A discussion of the Relevance Vector Machine and its various implementations follows.

2.3.1. Relevance Vector Machines

In the Relevance Vector Machine (RVM) a probabilistic framework is used to associate priors with each of the weights, w_m. These priors make the weight values tend to zero unless there is evidence for them to be non-zero in the data. The variances of these priors are controlled by a set of hyper-parameters whose values are estimated by maximising the Bayesian evidence or type II maximum likelihood. The posterior distributions for the weights are, in the majority of cases, sharply peaked at zero and thus sparse models are obtained. Those training vectors included in the final model, i.e. those with non-zero weights associated with them, are referred to as "relevance vectors".

While the RVM takes longer to train than the SVM, this is offset by not having to perform cross-validation to determine and set model complexity parameters. Additionally, the RVM typically yields sparser models than the SVM and therefore has shorter prediction times than the SVM – something that may be particularly useful when rapid prediction is required. Finally, whereas the SVM is non-probabilistic and produces predictions without any measure of the uncertainty of the prediction, the RVM is probabilistic giving error bars for regression predictions and posterior probabilities of class membership when applied to classification problems.

To begin with we describe the classic RVM in section 2.3.1. We then describe its faster implementation the "fast" RVM (fRVM) in section 2.3.1, which exploits a decomposition of the log marginal likelihood to yield an efficient greedy learning algorithm for determining the model parameters. In section 2.3.1 we briefly discuss some other RVM variants of note.

Classic RVM

In this section we detail the original, or "classic", RVM implementation [Tipping, 2001, 2000]. To begin with we consider the RVM model for regression before extending it to classification.

Given a training dataset of sample-target pairs $\mathcal{D}_{\text{tr}} = \{\mathbf{x}_1, t_1, \ldots, \mathbf{x}_N, t_N\}$ it is asumed that the targets are samples from a model plus some noise:

$$t_n = y(\mathbf{x}_n; \mathbf{w}) + \epsilon_n \tag{2.46}$$

where each ϵ_n is an independent noise sample assumed to come from a zero-mean Gaussian of variance σ^2. Thus the probability of a particular target value, t_n, for an input $\mathbf{x}_n$ is:

$$p(t_n|\mathbf{x}) = \mathcal{N}(t_n|y(\mathbf{x}_n; w), \sigma^2) \tag{2.47}$$

which specifies a Gaussian distribution over t_n of mean $y(\mathbf{x}_n)$ and variance σ^2 where $y(\mathbf{x}_n)$ is:

$$y(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{N} w_i K(\mathbf{x}, \mathbf{x}_i) + w_0 \tag{2.48}$$

and of the same form as the SVM model found in equation 2.45.

Since each t_n is assumed to be independent, the likelihood for the dataset can be written as:

$$p(\mathbf{t}|\mathbf{w}, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}||\mathbf{t} - \mathbf{\Phi}\mathbf{w}||^2\right\} \tag{2.49}$$

where $\mathbf{t} = (t_1, \ldots, t_N)^T$, $\mathbf{w} = (w_0, \ldots, w_N)^T$ and $\mathbf{\Phi}$ is the $N \times (N+1)$ "design" matrix where $\mathbf{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N)]^T$ given:

$$\boldsymbol{\phi}(\mathbf{x}_n) = [1, K(\mathbf{x}_n, \mathbf{x}_1), \ldots, K(\mathbf{x}_n, \mathbf{x}_N)]^T. \tag{2.50}$$

Since maximum likelihood estimation of $\mathbf{w}$ and σ^2 from 2.50 would most likely lead to overfitting, Tipping [2001, 2000] "adopts a Bayesian perspective" and penalises overly-complex models by placing prior distributions over the model parametrisation.

To encourage less complex functions a Gaussian centred on 0 is used for the prior distribution on $\mathbf{w}$:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{N} \mathcal{N}(w_i|0, \alpha_i^{-1}) \tag{2.51}$$

where $\boldsymbol{\alpha}$ is a vector of $N+1$ *hyper-parameters*, α_i. Notice that each weight, w_i has its own individual hyper-parameter, α_i, associated with it, controlling the strength of the prior on that particular weight.

Hyperpriors are then placed over $\boldsymbol{\alpha}$

$$p(\boldsymbol{\alpha}) = \prod_{i=0}^{N} Gamma(\alpha_i|a, b) \tag{2.52}$$

and the noise variance σ^2:

$$p(\beta) = Gamma(\beta|c, d) \tag{2.53}$$

where $\beta \equiv \sigma^{-2}$, the noise precision, and:

$$Gamma(\alpha|a, b) = \Gamma(a)^{-1} b^a \alpha^{a-1} e^{-b\alpha} \tag{2.54}$$

is the gamma probability density function.

By setting the parameters $a = b = c = d = 0$ the predictions become independent of the linear scaling of $\mathbf{t}$ and the basis functions, $\phi_i(\mathbf{x})$. Thus, in this case, the results are independent of the unit of measurement of the targets.

Using prior distributions to control model complexity in this way makes them a type of automatic relevance determination (ARD) prior [MacKay, 1992b] though, in neural networks, these priors are usually associated with *groups* of weights rather than individual ones as seen here. By using a broad prior over the hyper-parameters, evidence in the training data will allow the probability mass to concentrate at very large values for some of the α_i. As a result of these large α_i values, the posterior probability of the associated weights is concentrated at 0, "switching off" the corresponding weights and deeming them "irrelevant". It is this association of individual hyper-parameters with each weight, and therefore each basis function, that allows the RVM to learn sparse models.

Having defined this prior structure, from Bayes' rule the posterior over all the parameters, $\boldsymbol{\theta} = \{\mathbf{w}, \boldsymbol{\alpha}, \sigma^2\}$, given the data is computed as:

$$p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}}) = \frac{p(\mathcal{D}_{\text{tr}}|\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)}{p(\mathcal{D}_{\text{tr}})}. \tag{2.55}$$

For a new input sample, $\mathbf{x}'$, predictions are made for the corresponding target, t' in terms of the predictive distribution as:

$$p(t'|\mathcal{D}_{\text{tr}}) = \int p(t'|\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}})d\mathbf{w}d\boldsymbol{\alpha}d\sigma^2. \tag{2.56}$$

Unfortunately we cannot calculate, $p(\mathcal{D}_{\text{tr}}) = \int p(\mathcal{D}_{\text{tr}}|\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)d\mathbf{w}d\boldsymbol{\alpha}d\sigma^2$, and are thus unable to calculate the posterior $p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}})$. Instead, the posterior is decomposed as:

$$p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}}) = p(\mathbf{w}|\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}}). \tag{2.57}$$

Since the normalising integral for the posterior distribution over the weights is a convolution of Gaussians the posterior distribution over the weights is given by:

$$\begin{aligned} p(\mathbf{w}|\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}, \sigma^2) &= \frac{p(\mathcal{D}_{\text{tr}}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})}{p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2)} && (2.58) \\ &= (2\pi)^{-(N+1)/2}|\boldsymbol{\Sigma}|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{w}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{w}-\boldsymbol{\mu})\right\} && (2.59) \end{aligned}$$

where the posterior covariance is calculated as:

$$\boldsymbol{\Sigma} = (\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \mathbf{A})^{-1} \tag{2.60}$$

given:

$$\mathbf{A} = \text{diag}(\alpha_0, \alpha_1, \ldots, \alpha_N) \tag{2.61}$$

and the mean of the distribution is:

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\mathbf{t}. \tag{2.62}$$

Approximation is used for the hyper-parameter posterior, $p(\boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}})$, representing it by

a delta function at its mode; its most probable values $\boldsymbol{\alpha}_{MAP}$ and σ^2_{MAP}.

Training and learning in the RVM then becomes the search for the posterior mode for the hyper-parameters; the maximisation of $p(\boldsymbol{\alpha}, \sigma^2|\mathcal{D}_{\text{tr}}) \propto p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha})p(\sigma^2)$ with respect to $\boldsymbol{\alpha}$ and β. For uniform hyperpriors only the first term of $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2)$ needs maximising, given by:

$$p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2) = \int p(\mathcal{D}_{\text{tr}}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w} \tag{2.63}$$

$$= (2\pi)^{-(N/2)}|\sigma^2\mathbf{I} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T|^{-1/2} \exp\left\{-\frac{1}{2}\mathbf{t}^T(\sigma^2\mathbf{I} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)^{-1}\mathbf{t}\right\} \tag{2.64}$$

which is known as the *marginal likelihood*, or evidence. Its maximisation is known as *type-II maximum likelihood* maximisation. Equivalently, one can maximise the logarithm of the marginal likelihood $\mathcal{L}(\boldsymbol{\alpha})$:

$$\mathcal{L}(\boldsymbol{\alpha}) = \log p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2) \tag{2.65}$$

$$= \log \int_{-\infty}^{\infty} p(\mathcal{D}_{\text{tr}}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w} \tag{2.66}$$

$$= -\frac{1}{2}\left[N \log 2\pi + \log|\mathbf{C}| + \mathbf{t}^T\mathbf{C}^{-1}\mathbf{t}\right] \tag{2.67}$$

where:

$$\mathbf{C} = \sigma^2\mathbf{I} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T. \tag{2.68}$$

Since the values of $\boldsymbol{\alpha}$ and σ^2 cannot be obtained in closed form, an iterative process is used instead. In this process the vector of hyper-parameters, $\boldsymbol{\alpha}$, is initialised with all α_i set to finite values and iteratively re-estimated to maximise $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\alpha}, \sigma^2)$, removing basis functions from the model by setting their associated α_i to ∞. Details for this can be found in [Tipping, 2000, 2001]. A faster, more efficient, method for setting the values of the hyper-parameters was introduced in [Tipping and Faul, 2003; Faul and Tipping, 2002] and is described in section 2.3.1.

Once the hyperparameter estimation procedure has converged to find the most probable values for $\boldsymbol{\alpha}$ and σ^2, predictions are made based on the posterior distribution over the weights, conditioned on $\boldsymbol{\alpha}_{MAP}$ and σ^2_{MAP}. The predictive distribution for a new sample, $\mathbf{x}'$, can then be computed as:

$$p(t'|\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}_{MAP}, \sigma^2_{MAP}) = \int p(t'|\mathbf{w}, \sigma^2_{MAP})p(\mathbf{w}|\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}_{MAP}, \sigma^2_{MAP})d\mathbf{w} \tag{2.69}$$

which, given that both terms in the integral are Gaussian, can be computed as:

$$p(t'|\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}_{MAP}, \sigma^2_{MAP}) = \mathcal{N}(t'|y', \sigma^{2'}) \tag{2.70}$$

where:

$$y' = \boldsymbol{\mu}^T\boldsymbol{\phi}(\mathbf{x}'), \tag{2.71}$$

$$\sigma^{2'} = \sigma^2_{MAP} + \boldsymbol{\phi}(\mathbf{x}')^T\boldsymbol{\Sigma}\boldsymbol{\phi}(\mathbf{x}'). \tag{2.72}$$

Thus, the predictive mean is $y(\mathbf{x}';\boldsymbol{\mu})$; the sparse weighting of the basis functions. The predictive variance for this is the sum of the estimated data noise and the uncertainty in the weight predictions. In practice, point predictions are obtained by setting $\mathbf{w}$ to $\boldsymbol{\mu}$, and retaining $\boldsymbol{\Sigma}$ should computation of error bars be required (see Appendix D.1 of [Tipping, 2001] for details of how to estimate the error bars).

Extending this regression framework to classification only requires the adaptation of the likelihood function and link function to deal with the different targets. To do so an additional approximation step has to be added to the algorithm.

In the case of binary classification, we desire predictions to take the form of a posterior probability of an input, $\mathbf{x}'$, belonging to one of the classes. Following convention, the RVM applies the logistic sigmoidal link function to $y(\mathbf{x}')$:

$$\sigma(y) = \frac{1}{1+e^{-y}}. \tag{2.73}$$

Using the Bernoulli distribution for $p(t|\mathbf{x})$, the likelihood can be written as:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} \sigma(y(\mathbf{x}_n;\mathbf{w}))^{t_n} \left[1-\sigma(y(\mathbf{x}_n;\mathbf{w}))\right]^{1-t_n} \tag{2.74}$$

where the targets $t_n \in \{0,1\}$. Notice that there is no noise variance for the targets here, unlike in the regression case.

Unlike regression, however, the weights cannot be calculated analytically and thus closed form expressions are not available for the weight posterior $p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha})$ or the marginal likelihood $p(t|\boldsymbol{\alpha})$. Instead, an approximation procedure based on Laplace's method and previously used by MacKay [1992a] is used.

Under this methodology the most probable weights $\mathbf{w}_{MAP}$ are found for the current, fixed, values of $\boldsymbol{\alpha}$ to obtain the mode of the posterior distribution. Since $p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha}) \propto p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})$ this is equivalent to finding the maximum over $\mathbf{w}$ of:

$$\log(p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})) = \sum_{n=1}^{N} \left[t_n \log y_n + (1-t_n)\log(1-y_n)\right] - \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} \tag{2.75}$$

where $y_n = \sigma(y(\mathbf{x}_n;\mathbf{w}))$. As this is a penalised log-likelihood function iterative maximisation must be used. Typically the iteratively-reweighted least-squares algorithm [Nabney, 1999] is used to find $\mathbf{w}_{MAP}$.

Equation 2.75 is then differentiated twice with respect to w to obtain:

$$\nabla_{\mathbf{w}}\nabla_{\mathbf{w}} \log p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha})|_{\mathbf{w}_{MAP}} = -(\boldsymbol{\Phi}^T\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}) \tag{2.76}$$

where $\mathbf{B}$ is a diagonal matrix, $\mathbf{B} = \mathrm{diag}(\beta_1,\ldots,\beta_N)$, in which:

$$\beta_n = \sigma(y(\mathbf{x}_n))\left[1-\sigma(y(\mathbf{x}_n))\right]. \tag{2.77}$$

The value obtained from (2.76) is then negated and inverted to obtain $\boldsymbol{\Sigma}$, the covariance for a Gaussian approximation to the posterior over the weights centred on $\mathbf{w}_{MAP}$. Using $\boldsymbol{\Sigma}$ and $\mathbf{w}_{MAP}$ (rather than $\boldsymbol{\mu}$) the hyper-parameters, $\boldsymbol{\alpha}$, can then be updated as per the

regression version of the RVM.

From 2.76) and using knowledge that $\nabla_{\mathbf{w}} \log p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha})|_{\mathbf{w}_{MAP}} = 0$ we obtain:

$$\boldsymbol{\Sigma} = (\boldsymbol{\Phi}^T \mathbf{B} \boldsymbol{\Phi} + \mathbf{A})^{-1} \tag{2.78}$$

and

$$\mathbf{w}_{MAP} = \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \mathbf{B} \mathbf{t} \tag{2.79}$$

at the mode of $p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha})$.

From these results we can see that the Laplace approximation has mapped the classification problem to a regression one with data-dependent, heteroscedastic, noise with the noise precision given by $\beta_n = \sigma(y(\mathbf{x}_n)) \left[1 - \sigma(y(\mathbf{x}_n))\right]$.

For multi-class problems, i.e. where there are more than two target classes, the likelihood can be generalised to the multinomial form as:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \sigma(y_k(\mathbf{x}_n; \mathbf{w}_k))^{t_{nk}} \tag{2.80}$$

where K is the number of classes and the targets take the form of the conventional one-of-K encoding. In this case the classifier has multiple outputs $y_k(\mathbf{x}; \mathbf{w}_k)$, each with its own parameter vector $\mathbf{w}_k$ and associated hyper-parameters $\boldsymbol{\alpha}_k$. Unlike in the SVM, there is no need to combine multiple binary classifiers. For an implementation of the RVM for multi-class problems see [Damoulas et al., 2008; Psorakis et al., 2010].

Using this framework, the RVM successfully learns sparse probabilistic models for pattern recognition tasks whose performance is equivalent to that of the SVM but with even sparser models [Tipping, 2001]. The main drawback of this method is the length of the training process. In the next section we discuss the "fast" RVM, an implementation of the RVM in which this training process is sped up.

Fast RVM

While the classic RVM successfully learns sparse probabilistic models for regression and classification tasks, its training procedure, based on the optimisation of the marginal likelihood, is typically slow. In [Tipping and Faul, 2003; Faul and Tipping, 2002] the "fast" RVM (fRVM) is introduced in which the marginal likelihood is maximised via a process of addition and deletion of basis functions, $\boldsymbol{\phi}_i$.

By considering the dependence of the log marginal likelihood, $\mathcal{L}(\boldsymbol{\alpha})$ on an individual hyper-parameter α_i, $\mathbf{C}$ from equation 2.68 can be decomposed as:

$$\mathbf{C} = \sigma^2 \mathbf{I} + \sum_{m \neq i} \alpha_m^{-1} \boldsymbol{\phi}_m \boldsymbol{\phi}_m^T + \alpha_i^{-1} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \tag{2.81}$$

$$= \mathbf{C}_{-i} + \alpha_i^{-1} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \tag{2.82}$$

where $\mathbf{C}_{-i}$ is $\mathbf{C}$ with the contribution from the i^{th} basis function, $\boldsymbol{\phi}_i$, excluded. Using

standard matrix determinant and inverse identities the following can be written:

$$|\mathbf{C}| = |\mathbf{C}_{-i}||1 + \alpha_i^{-1}\boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i| \tag{2.83}$$

$$\mathbf{C}^{-1} = \mathbf{C}_{-i}^{-1} - \frac{\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}}{\alpha_i + \boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i} \tag{2.84}$$

This enables $\mathcal{L}(\boldsymbol{\alpha})$ to be written as:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\Bigg[&N\log(2\pi) + \log|\mathbf{C}_{-i}| + \mathbf{t}^T\mathbf{C}_{-i}^{-1}\mathbf{t} - \log\alpha_i \\ &+ \log(\alpha_i + \boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i) - \frac{(\boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\mathbf{t})^2}{\alpha_i + \boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i}\Bigg] \qquad (2.85)\\ = \mathcal{L}(\boldsymbol{\alpha}_{-i}) &+ \frac{1}{2}\left[\log\alpha_i - \log(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right] \qquad (2.86)\\ = \mathcal{L}(\boldsymbol{\alpha}_{-i}) &+ l(\alpha_i) \qquad (2.87)\end{aligned}$$

where s_i, the "sparsity factor", and q_i, the "quality factor" are:

$$s_i = \boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i, \tag{2.88}$$

$$q_i = \boldsymbol{\phi}_i^T\mathbf{C}_{-i}^{-1}\mathbf{t}. \tag{2.89}$$

The sparsity factor, s_i, is used to measure the extent to which the associated basis vector, $\boldsymbol{\phi}_i$, "overlaps" those already included in the model. The quality factor, q_i, can be written as:

$$q_i = \sigma^{-2}\boldsymbol{\phi}_i^T(\mathbf{t} - \mathbf{y}_{-i}) \tag{2.90}$$

and thus can be seen to be measuring the alignment of $\boldsymbol{\phi}_i$ with the prediction error of the model with that basis function excluded from it.

In [Faul and Tipping, 2002], analysis of $l(\alpha_i)$ showed that $\mathcal{L}(\boldsymbol{\alpha})$ has a unique maximum with respect to α_i as:

$$\alpha_i = \frac{s_i^2}{q_i^2 - s_i} \qquad \text{if } q_i^2 > s_i \tag{2.91}$$

$$\alpha_i = \infty \qquad \text{if } q_i^2 \leq s_i \tag{2.92}$$

Calculation of the values of s_i and q_i for all the available basis functions, $\boldsymbol{\phi}_i$, is relatively straightforward – including those not currently in the model as their associated $\alpha_i = \infty$.

This can be made easier by maintaining and updating:

$$S_m = \boldsymbol{\phi}_m^T\mathbf{C}^{-1}\boldsymbol{\phi}_m \tag{2.93}$$

$$Q_m = \boldsymbol{\phi}_m^T\mathbf{C}^{-1}\mathbf{t} \tag{2.94}$$

Algorithm 1 Fast RVM

Require: $\mathbf{\Phi}$ *Set of M basis functions*
$\mathbf{t}$ *Vector of N targets*

```
if regression
  σ² := initialise()                                   Initialise σ²
end
α := initialise_alphas()                               Initialise α with a single α_i ≠ ∞
if regression
  Σ, μ, s, q := reg_calculate(Σ, μ, s, q)              Calculate Σ, μ, s and q
else
  Σ, w_MAP, s, q := cla_calculate(Σ, w_MAP, s, q)      Calculate Σ, w_MAP, s and q
end
while not converged
  φ_i := select(Φ)                                     Select a φ_i from Φ
  θ_i := q_i² − s_i
  if(θ_i > 0 & α_i < ∞)
    α_i := new_estimate(α_i)                           Re-estimate the value of α_i
  end
  if(θ_i > 0 & α_i = ∞)
    α_i = alpha_update(α_i)                            Add φ_i to the model
  end
  if(θ_i ≤ 0 & α_i < ∞)
    α_i := ∞                                           Delete φ_i from the model
  end
  if regression
    σ² := sigma_update(σ²)
    Σ, μ, s, q := reg_update(Σ, μ, s, q)               Update Σ, μ, s and q
  else
    Σ, w_MAP, s, q := cla_update(Σ, w_MAP, s, q)       Update Σ, w_MAP, s and q
  end
end
```

from which it follows that:

$$s_m = \frac{\alpha_m S_m}{\alpha_m - S_m} \tag{2.95}$$

$$q_m = \frac{\alpha_m Q_m}{\alpha_m - S_m} \tag{2.96}$$

noting that when $\alpha_m = \infty$, $s_m = S_m$ and $q_m = Q_m$.

From the calculation of s_i and q_i rules for adding and deleting a basis function, $\boldsymbol{\phi}_i$, from the model can be implied as follows: if $\boldsymbol{\phi}_i$ is in the model with associated $\alpha_i < \infty$ yet $q_i^2 \leq s_i$ then $\boldsymbol{\phi}_i$ may be deleted from the model by setting $\alpha_i = \infty$, and, if $\boldsymbol{\phi}_i$ is excluded from the model with associated $\alpha_i = \infty$ and $q_i^2 > s_i$ then $\boldsymbol{\phi}_i$ may be added to the model by setting α_i to some optimal finite value. Should a $\boldsymbol{\phi}_i$ already be in the model with associated $q_i^2 > s_i$, then the α_i can be re-estimated. In these situations discrete alterations are made to the structure whilst simultaneously increasing the marginal likelihood.

By sequentially carrying out these operations for varying i, as summarised in algorithm 1, the fRVM obtains an efficient (greedy) learning algorithm for the values of α_i and,

thus, the model weights. Since the calculations are only carried out in terms of the basis functions already in the model and a candidate basis function $\boldsymbol{\phi}_i$ this learning process is quicker than in the classic RVM. Whereas the classic RVM starts with all basis functions "switched on" (all $\alpha_m < \infty$), the fRVM is initialised using a *single* basis function, leading to less expensive calculation overheads. Since the fRVM learns slightly sparser models than the RVM with comparable performance [Tipping and Faul, 2003], it is the fRVM that is used in the experiments in this thesis.

Other RVM variants

While the conventional RVM methodology produces point estimates for the vector of hyper-parameters, $\boldsymbol{\alpha}$, the Variational RVM [vRVM, Bishop and Tipping, 2000] makes use of a "more complete Bayesian treatment" of the original RVM methodology. Using variational inference, posterior distributions for the hyper-parameters, $\boldsymbol{\alpha}$, and thus parameters, are obtained. This variational solution for the RVM is slower to train than the classic type-II maximum likelihood version. It is expected that when faced with limited training data, however, that using this Bayesian approach will be more advantageous and since less data is available the increased computational costs during the training phase will be minimal. However, Tipping notes[2] that the vRVM is "pretty much identical to the non-variational version, but is a lot slower to train".

Bae and Mallick [2004] present a two-level hierarchical Bayesian regression model in which priors are placed over the model weights in order to promote sparse models for the task of gene selection. A Gaussian prior of unknown variance is placed over each of the model weights, w_m, and priors assigned to the variances. Three different prior models for the variance are considered, each promoting a different level of model sparseness. In the first model, an Inverse Gamma distribution is placed over the variances, resulting in a prior structure of the same form as the RVM in which a Gamma distribution is placed over the inverse variances. In the second model, an exponential distribution is placed over the weight variances, resulting in a Laplace prior for the weights. In the third model, the non-informative Jeffries prior recommended by Figueiredo [2001, 2003] is used. This differs from the other two priors as Jeffreys prior requires no setting of hyper-parameters, enabling automatic variable selection and strongly inducing model sparseness. Unlike the RVM, where type-II maximum likelihood estimation is used to obtain the most probable set of hyper-parameters, $\boldsymbol{\alpha}_{MAP}$, MCMC sampling is used to obtain vectors of hyper-parameters for the regression model, leading to potentially long training times.

Sparsity in the RVM is dependent on the choice of kernel, or basis function, and thus overfitting can be a problem, in particular when multi-resolution kernels are used. In [Schmolck and Everson, 2007] the Smooth RVM (SRVM) builds on the fRVM framework [Faul and Tipping, 2002] making use of a noise dependent smoothness prior to prevent overfitting in regression. Although Schmolck and Everson [2007] state that the smoothness prior can be easily adapted to handle classification problems, Schmolck [2008] notes difficulties in applying it for classification. This is discussed later in chapter 3 of this thesis.

Clark and Everson [2008, 2011a,b] addressed the issue of RVM over-fitting when using

[2] `http://www.miketipping.com/index.php?page=rvm`, 29/09/11

multi-resolution kernels in classification problems. A novel measure of RVM classifier complexity was proposed and successfully used to penalise overly complex RVM classifiers using an MOEA to set the RVM hyper-parameters, simultaneously optimising true and false positive rates along with model complexity to produce Pareto fronts of RVM classifiers from which to select. These EA located RVMs were shown to give equivalent performance to that of RVMs trained using the fRVM methodology whilst learning sparser models. This work forms the basis of chapter 3 of this thesis.

2.3.2. Summary

Overfitting is an important concern in supervised pattern recognition. In order to combat this, there has been interest in learning sparse models with good generalisation capability. The Support Vector Machine is one such method for learning sparse models but produces non-probabilistic predictions. Tipping [2000, 2001] introduced the Relevance Vector Machine which cast the task of sparse modelling in a probabilistic Bayesian framework. The RVM learns sparse probabilistic models with good generalisation performance and equivalent performance to the SVM. Typically, these models are sparser than those learned by the SVM and have the added benefit of providing a measure of the certainty of the model prediction.

Unfortunately, when multi-resolution kernels are used the RVM tends to overfit. For the case of regression this was addressed by Schmolck and Everson [2007] through the use of a noise dependent smoothness prior. In this thesis we present a novel measure of RVM classifier complexity and use a multi-objective evolutionary algorithm to simultaneously optimise classifier complexity and performance. In chapter 3 we locate the best set of trade-offs between true and false positive rates and RVM complexity on a number of benchmark datasets. Chapter 4 demonstrates the ability to successfully identify genes associated with particular genetic conditions using RVM classifiers optimised in terms of area under the ROC curve and complexity.

In the next section we discuss the multi-objective optimisation. Using multi-objective optimisation methods, one is able to incorporate different measures of classifier performance and suitability into the model training and selection process. In particular, multi-objective optimisation allows us to optimise ROC curves during training.

2.4. Multi-Objective Optimisation

In many problems there are two or more competing measures, or *objectives*, that we would like to consider when generating solutions. For example, when generating potential water network designs there is a trade-off between the cost of building it and the resulting customer experience [e.g. McClymont et al., 2010]. Cheaper networks may result in poor water quality, water pressure problems or poor network robustness and thus a poor customer experience. More expensive networks should suffer from fewer of these problems, leading to a better customer experience. In the case of classification problems we may be concerned with competing measures such as model complexity, true and false positive rates, AUC, classification time etc.

Given a set of competing objectives, we want to find the *best* set of trade-offs between

them. For the case of the water network planning we would like to get the best customer experience possible for a given amount of money spent – why spend more money to obtain worse performance than a cheaper alternative? Once we have the best set of trade-offs the user can then select the solution most closely fitting their needs with knowledge of the available alternatives. The set of solutions giving the best trade-offs between a set of objectives is known as the Pareto front.

Multi-objective optimisation methods [Deb, 2001; Jin, 2006; Coello et al., 2007; Knowles et al., 2008] enable one to locate sets of the best trade-offs between multiple competing objectives, and have been successfully applied to a range of classification problems. In [Lofstrom et al., 2009] multi-objective optimisation was used to generate ensembles of classifiers in which the ensemble accuracy and diversity were simultaneously optimised. Multi-objective optimisation of the Support Vector Machine was used by Mierswa [2007] for simultaneously optimising training error and model complexity by perturbing model hyper-parameters. More recently, Reckhouse et al. [2010] used multi-objective optimisation to optimise Receiver Operating Characteristic curves for the NATS Short Term Conflict Alert system which classifies aircraft trajectory pairs as either resulting in the aircraft becoming dangerously close or maintaining a safe distance.

One early method of optimising the multiple objectives was to aggregate them together to form a composite measure of solution fitness [e.g., Schaffer, 1985b,a] i.e. the objectives are added together to give a single scalar value. By individually weighting the contributions of each of the objectives their perceived relative importance can be encoded in this composite function. Unfortunately, in order to obtain the full set of optimal trade-offs between the objectives, known as Pareto fronts, will often require multiple runs of the optimisation algorithm. Other problems arise depending on whether the Pareto front is convex or not [Das and Dennis, 1997]. Should the front not be convex then no prior weighting of the objectives exists which locate solutions in the nonconvex part, though these solutions can be located if one allows the weights to be adjusted during the optimization process [Jin et al., 2001]. In the case of *convex* fronts, an even spread of weightings rarely produces an even spread of points in the Pareto front. Thus, researchers moved away from methods based on the aggregation of objectives and instead moved towards a Pareto-dominance based methodology. Modern Evolutionary Algorithms [EAs, Coello et al., 2007; Knowles et al., 2008] produce entire Pareto fronts of solutions in a single run and are more appropriate for handling the commonly occurring situation in which the costs associated with the objectives are unknown. It is these Pareto-dominance based EAs that we discuss in this chapter.

To begin with, in section 2.4.1, we discuss the concepts of Dominance and Pareto optimality, used by multi-objective methods to compare models based on their performance on the objectives. Section 2.4.2 gives an overview of Multi-Objective Evolutionary Algorithms and in section 2.4.3 multi-objective optimisation of ROC curves is discussed.

2.4.1. Dominance and Pareto Optimality

When carrying out multi-objective optimisation, one looks to minimise (or maximise) a set of D objective functions $f_i(\boldsymbol{\theta})$, $i = 1, \ldots, D$ of P parameters or decision variables, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$. Since any function to be maximised can be converted to one that is to

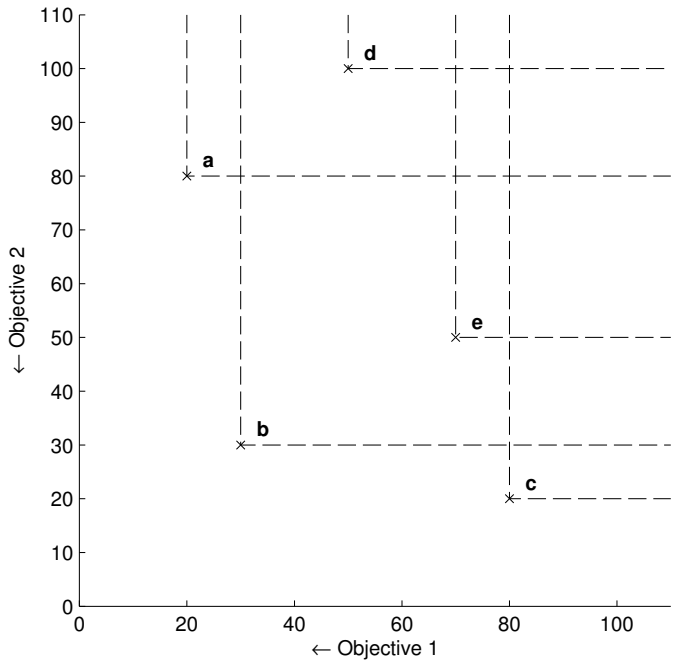


Figure 2.7.: An example of dominance on a two-objective minimisation problem. Labeled x's mark each feasible solution. Dashed lines indicate the area dominated by each solution.

be minimised we can therefore assume, without loss of generality, that all objectives are to be minimised and express the problem as:

$$\text{Minimise} \quad \mathbf{f}(\boldsymbol{\theta}) \equiv (f_1(\boldsymbol{\theta}), \ldots, f_D(\boldsymbol{\theta})) \tag{2.97}$$

noting that there may be additional constraints on the values of $\boldsymbol{\theta}$ e.g. all $\{\theta_i\}$ must be positive or some combinations of θ_i are not allowed.

In order to compare two multi-objective quantities, $\mathbf{f}$ and $\mathbf{g}$, we make use of the *dominates* relation. If $\mathbf{f}$ is no worse than $\mathbf{g}$ on *all* objectives and wholly better than $\mathbf{g}$ on *at least* one objective than $\mathbf{f}$ is said to dominate $\mathbf{g}$: $\mathbf{f} \prec \mathbf{g}$. Thus $\mathbf{f}$ dominates $\mathbf{g}$ iff:

$$f_i \leq g_i \quad \forall i \in \{1, \ldots, D\} \text{ and } f_i < g_i \text{ for at least one } i \tag{2.98}$$

This objective space dominance can be extended to parameter space. Thus, for two parametrisations $\mathbf{a}$ and $\mathbf{b}$, if $\mathbf{f}(\mathbf{a}) \prec \mathbf{f}(\mathbf{b})$, it is said that $\mathbf{a}$ dominates $\mathbf{b}$: $\mathbf{a} \prec \mathbf{b}$.

In figure 2.7 an example two-objective minimisation problem is plotted for which 5 different solutions, $\mathbf{a} - \mathbf{e}$, are available. Of these solutions, solution $\mathbf{a}$ dominates solution $\mathbf{d}$, $\mathbf{a} \prec \mathbf{d}$, and solution $\mathbf{b}$ dominates solutions $\mathbf{d}$ and $\mathbf{e}$, $\mathbf{b} \prec \mathbf{d}$ and $\mathbf{b} \prec \mathbf{e}$.

The dominates relation is not a total order and therefore two solutions are deemed *mutually non-dominating* if neither dominates the other. A set of solutions, E, is said to be *non-dominating* if no member of E dominates any other:

$$\mathbf{a} \not\prec \mathbf{b} \quad \forall \mathbf{a}, \mathbf{b} \in E. \tag{2.99}$$

Slightly abusing the notation, we write $E \prec \mathbf{a}$ if at least one member of the set E dominates the solution $\mathbf{a}$.

A set of solutions, in the form of parametrisations, which no other feasible solution dominates any members of, is considered *Pareto optimal* [Srinivas and Deb, 1994], or *globally non-dominated*. The set of all Pareto-optimal solutions is referred to as the *Pareto set* and the image in objective space under $\mathbf{f}(\cdot)$ of the Pareto set is known as the *Pareto-optimal front*, $\mathcal{P}$. Thus, the Pareto front represents all possible trade-offs between the competing objectives for a problem.

Returning to figure 2.7 we see solutions **a**, **b** and **c** are mutually non-dominating since none of them dominates another. Solutions **d** and **e** are also mutually non-dominating as **d** does not dominate **e** and **e** does not dominate **d**. Solutions **a**, **b** and **c** form the Pareto front here, approximating the Pareto-optimal front. If we were to remove the globally non-dominated solutions, **a**, **b** and **c**, then solutions **d** and **e** would form a new, second, set of non-dominated solutions. In some algorithms, e.g. [NSGA, Srinivas and Deb, 1994], solutions are ranked based on which non-dominated front they are in. Solutions in the first non-dominated front would be assigned rank 1, solutions in the second non-dominated front assigned rank 2, etc.

In the next section multi-objective evolutionary algorithms are discussed which use the concepts of dominance and Pareto-optimality to locate sets of solutions giving the best trade-offs between competing objectives, approximating the true Pareto-optimal front.

2.4.2. Multi-objective Evolutionary Algorithms

Inspired by evolution, Multi-Objective Evolutionary Algorithms (MOEAs), and similarly Genetic Algorithms, a type of MOEA, vary (crossover/mutate/perturb) "parent" solutions, in the form of model parametrisations, to produce new "child" solutions whose performance is evaluated on multiple competing objectives. The better a child solution performs, the more likely it is to survive to the next generation and form the basis of new solutions and thus, over successive generations, better and better solutions should be found. When discussing MOEAs and the operations they carry out, it is useful to consider them in terms of the Unified Model for Multi-Objective Evolutionary Algorithms [UMMEA, Laumanns et al., 2000].

Multi-Objective Evolutionary Algorithms are run for a number of generations either until some convergence criteria is met or a predefined maximum number of generations is exceeded. At each generation MOEAs carry out four steps in turn; *sample*, *vary*, *evaluate* and *update*. A solution is selected (sampled) from those in the population, varied to form a new solution(s) and evaluated using some fitness function. Based on the fitness assigned to the new solution, a decision is made whether or not the new solution should enter the population and if any existing population members should be removed from it. This is the *update* step.

A fitness function is used to determine the population ranking and drive the search towards the Pareto optimal front. As such, it needs to account for each objective being optimised, typically without any knowledge of the associated costs. In many modern EAs, this fitness is based on dominance.

A suitable combination of selection and update functions should promote fit solutions while maintaining diversity of solutions in the population and, thus, hopefully avoid becoming stuck in any *local* optima – a problem that may occur with non-diverse popu-

lations. However, should the population be too diverse then selective pressure may be reduced leading to slow convergence to the *global* optima.

The *vary* step refers to application of the set of operations in which new child solutions are generated from their parent solutions. These operations, often referred to as genetic operators, fall into two categories: *mutation*, sometimes referred to as perturbation, and *cross-over*. In mutation, individual elements, θ_i, of the solution, $\boldsymbol{\theta}$, are modified by increasing or decreasing their value. In cross-over two or more parametrisations are "spliced" together taking a subset of the parameters from one parent and replacing the corresponding subset in another parent to generate new child parametrisations.

Two categories of evolutionary algorithms exist: *elitist* and *non-elitist*. In non-elitist EAs each successive population of solutions is evaluated and sorted by its fitness, replacing the parent population with the best of the child population. This ensures that the best of each generation is used to form the next. In elitist EAs both the parent and child populations are *combined* and the best solutions from the combined population kept. This has been shown to enhance the convergence of MOEAs [Laumanns et al., 2000; Deb, 2001] and, more importantly, means that good solutions will not be lost once they have been found – something that occurs in non-elitist EAs since the parent population is discarded each generation. We briefly discuss some of the more popular examples of non-elitist and elitist EAs – see Deb [2001] and [Coello et al., 2007] for comprehensive reviews of MOEAs.

Non-elitist EAs

The Multi-Objective Genetic Algorithm [MOGA, Fonseca and Fleming, 1993] assigns cost to a solution as 1 plus the number of solutions dominating it, where cost can be thought of as an inverse fitness. As a result, the non-dominated solutions in the current generation approximating the Pareto front are assigned a rank of 1. Those members of the population dominated by other members are assigned a cost proportional to the surrounding population density. Thus, in any population there must be at least one member with rank 1 and the maximum rank attributed to any population member will be N, the number of population members. Using this dominance based cost assignment, MOGA simultaneously emphasises non-dominated solutions in the GA population whilst maintaining diversity of solutions by penalising solutions with similar locations in objective space, resulting in a spread of Pareto-optimal solutions in the objective space. Its performance, however, can be sensitive to the shape of the Pareto front and density of solutions.

In the Non-dominated Sorting Genetic Algorithm (NSGA) Srinivas and Deb [1994] solutions are initially ranked based on "layers" of non-domination, whereby all solutions of the same rank are given an initially identical fitness value giving them equal chances of selection. See section 2.4.1 for a discussion of how these layers are obtained by removing successive sets of globally non-dominated solutions. "Sharing" is then used to penalise more densely populated regions, dividing a member's initial fitness by the number of solutions in a specified niche area. Starting with the rank 1 solutions, some initial random fitness value will be assigned to all rank 1 solutions. Using sharing, these initial values will be updated for each of the solutions. Once this is complete, the non-dominated rank 1 solutions will be ignored and the rank 2 solutions identified. The highest fitness assigned to any of the rank 2 solutions will be less than the minimum assigned to the rank 1

solutions. This process is repeated until the whole population has been ranked and these sharing based fitnesses assigned. Since those solutions on or near the current estimated Pareto front will possess the highest fitness values, a greater proportion of them will be used to form the next population and thus the population should move towards the true Pareto front over successive generations. Unfortunately, this repeated ranking and fitness assignment is computationally expensive and requires the manual setting of a parameter for the sharing calculations, upon which algorithm performance is sensitive.

Elitist EAs

More recent EAs have made use of the concept of "elitism" as it has been shown to enhance the convergence of MOEAs [Laumanns et al., 2000; Deb, 2001]. Whereas non-elitist discard and replace the parent population with the child population, in elitist EAs a percentage of the "best" solutions are retained from generation to generation in a secondary archive, referred to as the elite archive. Use of this secondary archive reduces wasted time rediscovering good solutions lost between successive generations, something that occurs in non-elitist EAs, and as such, convergence to the Pareto front should be quicker.

One of the earliest examples of an elitist MOEA was the Strength Pareto Evolutionary Algorithm [SPEA, Zitzler and Thiele, 1998]. In SPEA, two populations are maintained; a standard population, $\mathcal{P}_t$, containing non-dominating and dominated solutions, and a second population, E_t containing only elite non-dominated solutions. At each iteration of the algorithm, any non-dominated members of $\mathcal{P}_t \cup E_{t-1}$ are copied to E'_t. As a result previously elite solutions in E_{t-1} that are dominated by a new solution in $\mathcal{P}_t$ are removed from the elite population. The elite population is therefore a combined population of both the old and new elite solutions.

When the size of the elite population becomes too great, exceeding some pre-set number of members, truncation is carried out using clustering to prune the more densely populated regions of the Pareto set, encouraging diversity. Fitness is assigned to each of the members of the elite population, $\mathcal{P}'_t$, based on the number of solutions in $\mathcal{P}_t$ that it dominates, where t is the current generation. Members of $\mathcal{P}_t$ then have their fitness assigned as "one plus the sum of the strength values of all external population members" that dominate it [Deb, 2001]. Selection is then carried out using binary tournament selection with these fitness values and cross-over and mutation operators used to create a new population, $\mathcal{P}_{t+1}$, while retaining the elite population $\mathcal{P}'_t$. This tournament selection gives preference to solutions near the Pareto-optimal front in the generation of the new population.

SPEA2 [Zitzler et al., 2001] attempted to address problems with the fitness assignment, density estimation and archive truncation that could be present in the original algorithm. In SPEA solutions that are dominated by the same members of the elite population will be assigned an identical fitness value and thus, if the elite population only contains one solution, all members of the population will be assigned the same rank leading to low selection pressure. If this occurs then SPEA behaves like a random search algorithm [Zitzler et al., 2001]. SPEA2 addresses this in the assignment of fitness to population members by not only taking into account how many individuals dominate a solution but also how many it dominates. To further prevent fitness value ties, nearest neighbour density estimation is used to discriminate between solutions. In SPEA2 density estimation

is based on both the elite *and* general population rather than just the non-dominated elite solutions. The clustering method used to prune solutions from the elite archive is also amended so as not to unnecessarily prune the outer solutions of the estimated Pareto set in the elite population. Comparisons of SPEA2 with the original SPEA implementation by Zitzler et al. [2001] demonstrated it to outperform the original.

In [Knowles and Corne, 2000] the Pareto-Archived Evolution Strategy (PAES) was presented. Canonical PAES uses a (1+1)-evolution strategy in which a single child solution is generated from a single parent. The child solution is compared against an elite archive of non-dominated solutions in order to determine if it should be accepted or not. If a child is not dominated by its parent and the elite archive members it is accepted into the archive. As with SPEA, there is a limit on the number of elite solutions that can be kept i.e. there is a fixed archive size. A crowding factor is determined by the the number of solutions that occupy cells in a uniform grid of objective space. Should the archive be full, a non-dominated solution will only be accepted into it if it can replace an existing solution that has a higher crowding factor. As a result, sparser areas of the Pareto front are filled out, promoting an even distribution of members. If a child solution does not dominate its parent or any member of the elite archive then the solution with the lower grid count becomes the new parent solution.

In NSGA-II, so named to highlight its genesis and place of origin [Deb, 2001], Deb et al. [2000] sought to address the large computational overheads, lack of elitism and requirement to specify a sharing factor in the NSGA algorithm. Ranking efficiency was improved by modifying the method used to locate layers of non-dominated solutions so that it was of complexity $O(MN^2)$ rather than NSGA's $O(MN^3)$, where N is the population size and M the number of objectives. See Deb et al. [2000] for details. Elitism is implemented by combining all the child solutions with the parent solutions and then carrying out ranking, thus retaining the best solutions from generation to generation. Instead of specifying a sharing factor, crowding distance for solutions in the final sampled layer is estimated by averaging the distance to the nearest two solutions on either side of a solution in objective space.

The Multiple Single Objective Sampling algorithm [MSOPS, Hughes, 2003] runs multiple *single objective* optimisation searches in parallel in which a set of target vectors are generated *a priori* with which to evaluate the fitness values of the population members. By defining these target vectors, interesting areas of the objective space can be deliberately targeted for investigation. Hughes later improved on this algorithm to create MSOPS-II [Hughes, 2007] in which two improvements were made to the original algorithm: automatic generation of the target vectors, removing the requirement of *a priori* user specification of targets, and an improved fitness assignment method allowing better solution constraint handling.

Summary

In this sub-section we have introduced the basic concepts behind MOEAs. Inspired by nature, MOEAs generate child solutions from parent solutions, typically using the concepts of Pareto-optimality and dominance to assess and compare individuals' fitness. Over successive generations solutions should get better and better, converging to the Pareto

front for the problem.

Two types of MOEAs were discussed, non-elitist and elitist, and a number of their popular implementations outlined. While non-elitist EAs retain the best child solutions to form the new parent solutions, elitist EAs retain the best solutions found so far. As a result, time is not wasted rediscovering good solutions. Indeed, elitism has been shown to enhance the convergence of MOEAs [Laumanns et al., 2000; Deb, 2001].

Typically, elitist MOEAs have a restriction on the size of the elite archive. As the number of members of the elite archive increases so do the computational overheads; more comparisons between candidate and elite solutions need making and more memory is required for storage as the size of the elite archive increases. By maintaining a fixed size elite archive one can ameliorate this. However, Fieldsend et al. [2003] demonstrate that restriction of the number of solutions in the elite archive can lead to shrinking and oscillating/retreating estimated Pareto fronts.

Using an elitist MOEA one obtains a set of mutually non-dominating solutions for the best trade-offs found between the competing objectives. From these, the user can select the solution most closely meeting their requirements with knowledge of the available alternatives. Since one can use almost any competing objectives in an MOEA, this enables us to incorporate ROC curves into the training procedure. In the next section, we discuss multi-objective optimisation of ROC curves.

2.4.3. Multi-objective Optimisation of ROC curves

The training of binary classifiers can be viewed as an optimization problem in which we desire to maximise the classifier performance on a dataset. As discussed in section 2.2, one often tries to optimise measures such as accuracy and error rates but these give poor indication of class-wise performance. True and false positive rates allow one to measure how well a classifier performs on a class-wise basis and are essentially two objective functions [Kupinski and Anastasio, 1999]: one describing how well the classifier classifies examples of the positive class and one describing how well it classifies examples of the negative class. These two objectives are non-commensurable and thus it may not be possible to simultaneously improve both the true and false positive rate for a classifier. While one could aggregate these two measures into a single objective, the best way to do so is usually unknown. Indeed, there are an infinite number of ways one could combine two objective functions into a single scalar function. Typically, one is unsure about the exact costs associated with each of these objectives and even if such costs were known precisely *a priori* it is still not always clear how best to use them to construct the aggregate function [Kupinski and Anastasio, 1999; Anastasio et al., 1998]. For example, in [Kupinski and Anastasio, 1999; Anastasio et al., 1998] it is unclear how best to construct an aggregate function based on true and false positive rates for medical diagnosis systems. As a result it would seem natural to cast the problem of training classifiers as a multi-objective optimisation problem in which we simultaneously optimise the true and false positive rates for a classifier, removing the need to construct some aggregate function.

The solution to a multi-objective optimisation problem is the Pareto-optimal set, the set of solutions for which no other feasible solution exists that is better on all objectives. When no information is available about the preference between objectives all solutions in

the Pareto-optimal set are equally valid solutions to the problem. In the case of classifier optimisation in terms of true and false postive rates, the members of the Pareto-optimal set are points on an optimal receiver operating characteristic curve. The performances of these members describe the limiting true and false positive rate trade-offs that the classifier can yield for the training data [Kupinski and Anastasio, 1999].

Multi-objective Evolutionary Algorithms allow us to locate Pareto-sets of solutions in terms of multiple competing objectives. Thus, by using an elitist EA, we can obtain a set of classifier parametrisations approximating the Pareto front of solutions in terms of true and false positive rates. In doing so, one is optimising the ROC curve for that classification problem and family of classifier models. Examples of evolutionary multi-objective optimisation of ROC graphs include; optimisation of diagnostic classifiers [Kupinski and Anastasio, 1999; Anastasio et al., 1998], optimising the parameters of NATS STCA system [Everson and Fieldsend, 2006b; Reckhouse et al., 2010] and the optimisation of the ROC curves of Relevance Vector Machine classifiers at varying model complexities [Clark and Everson, 2008, 2011a].

While this thesis is focused on ROC optimisation for binary classifiers, it is noted that optimisation of ROC curves is not limited to binary classification problems. Indeed, in [Everson and Fieldsend, 2006a] multi-class ROC curves are optimised.

2.4.4. Summary

In many problems we often have multiple competing measures of performance or suitability, referred to as objectives. Rather than simply aggregating these objectives into a single scalar objective function, one can use multi-objective optimisation to locate the best set of trade-offs between these objectives. Multi-objective evolutionary algorithms are a multi-objective optimisation technique inspired by nature, which use the concepts of Pareto-optimality and dominance to locate sets of solutions containing the best set of trade-offs between the competing objectives. In particular, we can use MOEAs to locate optimal ROC curves for classification problems [Kupinski and Anastasio, 1999; Anastasio et al., 1998; Everson and Fieldsend, 2006b].

While MOEAs allow one to simultaneously optimise multiple competing objectives, caution should be used when selecting the number of objectives. As the number of objectives increases so it becomes more likely that two different solutions will be mutually non-dominating. Thus, there is reduced selection pressure during the optimisation process and convergence to the Pareto set may not occur, or take considerable numbers of iterations [Ishibuchi et al., 2008]. As an added consequence there will be an increased number of mutually non-dominating solutions with which to compare newly generated models for dominance leading to increasing computation overheads – though the maximum number of comparisons can be reduced using fixed population sizes.

Since elitist MOEAs generate Pareto sets of solutions, final model selection will be the responsibility of the user. Depending on the number of competing objectives, this may prove difficult. For two objectives, visualisation is straight-forward enough, plotting solutions on a two-axis graph. Solutions competing on three objectives can be plotted in three-dimensions, though this may be difficult to display in hard copy form. For larger numbers of competing objectives, referred to as *many* objective optimisation problems,

visualisation to enable solution selection is difficult. Methods for visualising these many-objective solutions is an active area of research [e.g. Everson and Fieldsend, 2006a; Walker et al., 2010] as are methods for reducing the number of objectives in order to enable visualisation in lower dimensional spaces [e.g. Walker et al., 2011]. However, this is beyond the scope of this thesis where, *at most*, three competing objectives are considered and thus such methods are not needed.

In this thesis MOEAs are used to optimise ROC curves while attempting to prevent overfitting. In [Clark and Everson, 2011a] a multi-objective optimising evolutionary algorithm was applied to setting hyper-parameters for Relevance Vector Machine classifiers, simultaneously optimising classifier true and false positive rates along with a novel measure of RVM classifier complexity, used as a third objective with which to encourage sparsity. Details of this can be found in chapter 3. An MOEA is also applied to the problem of gene selection for genetic condition classification in chapter 4, where the area under the ROC curve and RVM complexity are simultaneously optimised in order to locate sparse classifiers giving the best possible class discrimination.

3. Controlling Complexity in RVMs

Parts of the work in this chapter have appeared as:

> Clark, A. and Everson, R. (2008). *Multi-objective optimisation of relevance vector machines: Selecting sparse features for face verification.* ICML/UAI/COLT Workshop on Sparse Optimisation and Variable Selection.
>
> Clark, A. and Everson, R. (2011a). *Evolving sparse multi-resolution RVM classifiers.* In Dupenois, M. and Walker, D., editors, Proceedings of the 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), pages 53 – 60, Exeter, UK. PCCAT, College of Engineering, Mathematics and Physical Sciences, University of Exeter.
>
> Clark, A. and Everson, R. (2011b). *Multi-objective learning of relevance vector machine classifiers with multi-resolution kernels.* Pattern Recognition. In Press.

3.1. Introduction

The Relevance Vector Machine [RVM, Tipping, 2001, 2000] and the faster implementation, the fast RVM [fRVM, Tipping and Faul, 2003; Faul and Tipping, 2002], train sparse probabilistic models for pattern recognition problems using a Bayesian framework. By making use of the kernel trick [Aizerman et al., 1964] models can be built in high-dimensional feature spaces at low computational cost with the advantage of a probabilistic formulation, unlike the Support Vector Machine. The RVM produces sparse models using the Automatic Relevance Determination prior [ARD, MacKay, 1992b], "switching off" basis functions for which there is little or no support in the data, thus reducing the complexity of the trained model. Unfortunately, this sparsity is not only controlled by the ARD prior but is also subject to choice of kernel [Schmolck and Everson, 2007] and severe overfitting can occur when multi-resolution kernels are used [Clark and Everson, 2008]. Despite this, it is still advantageous to use the RVM and its Bayesian framework, as the choice of basis functions is determined by the data rather than by manual control of regularising parameters such as kernel width or, in the case of the SVM, error/margin trade-off and insensitivity parameters.

In the case of regression, overfitting when using multi-resolution kernels has been ameliorated by making use of a smoothness prior [Schmolck and Everson, 2007] but that method cannot be easily applied to classification. Additionally, the costs of misclassification may be unknown and one commonly looks to assess performance over a *range* of misclassification costs rather than a single one. Such evaluation over a range of costs is typically carried out using a Receiver Operating Characteristic (ROC) curve [e.g. Provost

and Fawcett, 1997; Fawcett, 2006] which allow analysis of true and false positive rates for a classifier over a range of misclassification costs by varying the decision threshold.

In order to address the problem of overfitting and allow for performance assessment over a range of cost ratios, this chapter presents a multi-objective optimisation (MO) method using an evolutionary algorithm (EA) in which RVM models are simultaneously optimised not only by true positive rate, T, and false postitive rate, F, but also in terms of a new measure of model complexity, C. This simultaneous optimisation of T, F and C leads to the generation of an approximation to the *Pareto Front* containing the best trade-offs between true and false positive rates and model complexity. Thus, at a given complexity, the Pareto front can be considered to be the ROC curve optimised for RVMs of that complexity. By controlling complexity in this way overfitting is reduced and so sparser models are learnt with equivalent generalisation performance to those trained by the standard fRVM methodology. As with most training procedures, MOEAs are prone to overfitting and so schemes to control overfitting during the evolutionary optimisation process are examined and their effects on test error rates investigated.

The chapter begins with an outline of the theory behind the Relevance Vector Machine (section 3.2) and multi-objective optimisation before introducing an algorithm for the multi-objective evolutionary optimisation of RVM classifiers (section 3.4) and providing a comparison between fRVM and MOEA performance on a small problem. Following that, a number of different cross-validation schemes are investigated (section 3.5) and results are presented on a number of benchmark binary classification problems not only in terms of accuracy but also in terms of the area under the receiver operator characteristic (ROC) curve (section 3.6).

3.2. The Relevance Vector Machine

The Relevance Vector Machine uses a logistic link function to model classification by mapping a linear combination of basis functions to a posterior class probability; thus if $\mathbf{x}$ is an input vector to be classified and $t \in \{0, 1\}$ is the associated target class label, then the posterior class probability is estimated as:

$$p(t \,|\, \mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{x})}} \tag{3.1}$$

where

$$y(\mathbf{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}) \tag{3.2}$$

and the w_m are the weights associated with each of the M, usually nonlinear, basis functions $\{\phi_m\}_{m=1}^{M}$. For notational convenience the bias unit is incorporated as $\phi_0 \triangleq 1$ and the $(M+1)$-dimensional vector of weights is denoted $\mathbf{w}$. The basis functions are commonly derived from kernels centred on each of the observations $\{\mathbf{x}_n\}_{n=1}^{N}$ in a provided training set, $\mathcal{D}_{\text{tr}}$, so that $\phi_m(\mathbf{x}) = K(\mathbf{x}_m, \mathbf{x})$. However the kernels used may be quite general functions such as multi-resolution Gabor wavelets used in image processing tasks [Shen, 2006; Shen et al., 2007; Clark and Everson, 2008]. As with Support Vector Machines [Boser et al., 1992; Vapnik, 1998; Schölkopf et al., 1999], using basis functions defined via ker-

nels permits inner product computations in the high-dimensional feature space spanned by the basis functions to be performed in the low-dimensional data space, affording great computational savings [Aizerman et al., 1964; Schölkopf and Smola, 2002].

An Automatic Relevance Determination [MacKay, 1992b] prior is used to enforce sparsity by placing the prior

$$p(w_m \,|\, \alpha_m) \propto \alpha_m^{\frac{1}{2}} \exp\left(-\frac{1}{2}\alpha_m w_m\right) \tag{3.3}$$

over each of the weights, w_m, $m = 0, \ldots, M$. As Tipping [2001] shows, this induces a heavy tailed Student-t prior over each of the weights w_m, thus favouring solutions in which probability mass is concentrated along the coordinate axes in weight space. As a result, sparse solutions where weights are either "switched on" ($\alpha_m \to 0$) or "switched off" ($\alpha_m \to \infty$) are favoured over solutions with many intermediate values of α_m. Those basis functions which are "switched on" are referred to as *relevance vectors*.

Learning is achieved by type-II maximum likelihood estimation, in which the marginal likelihood

$$\mathcal{L}(\boldsymbol{\alpha}) = p(\mathbf{t} \,|\, \boldsymbol{\alpha}) = \int p(\mathbf{t} \,|\, \mathbf{w}) p(\mathbf{w} \,|\, \boldsymbol{\alpha}) d\mathbf{w} \tag{3.4}$$

is maximised with respect to the vector of hyperparameters $\boldsymbol{\alpha}$. When carrying out regression, with Normally distributed observational noise of variance σ^2, the likelihood of a single observation is $p(t \,|\, \mathbf{w}, \sigma^2) = \mathcal{N}(t \,|\, y(\mathbf{x}), \sigma^2)$, and so (3.4) can be integrated analytically. Writing the training targets as a N-dimensional vector $\mathbf{t}$, we have

$$\log p(\mathbf{t} \,|\, \boldsymbol{\alpha}, \sigma^2) = -\frac{1}{2}\left[N \log 2\pi + \log |\mathbf{C}| + \mathbf{t}^T \mathbf{C}^{-1} \mathbf{t}\right] \tag{3.5}$$

with

$$\mathbf{C} = \sigma^2 \mathbf{I} + \boldsymbol{\Phi} \operatorname{diag}(\boldsymbol{\alpha})^{-1} \boldsymbol{\Phi}^T \tag{3.6}$$

where $\boldsymbol{\Phi}$ is the N by $(M+1)$ design matrix: $[\boldsymbol{\Phi}]_{nm} = \phi_m(\mathbf{x}_n)$. The "fast" RVM [fRVM, Faul and Tipping, 2002; Tipping and Faul, 2003] exploits a decomposition of the model covariance matrix $\mathbf{C}$ to yield an efficient greedy search procedure that at each step maximises the marginal likelihood with respect to a *single* α_m. See Chapter 2.3.1 for details.

For classification problems (3.4) cannot be integrated exactly. Instead, a Laplace approximation [Ó Ruanaidth and Fitzgerald, 1996] may be used to approximate the integrand around its mode, located using iterated reweighted least squares [IRLS, Nabney, 1999], giving an expression for the log marginal likelihood similar to (3.5):

$$\log p(\mathbf{t} \,|\, \boldsymbol{\alpha}) \approx -\frac{1}{2}\left[N \log 2\pi + \log |\mathbf{C}| + \hat{\mathbf{t}}^T \mathbf{C}^{-1} \hat{\mathbf{t}}\right] \tag{3.7}$$

where

$$\mathbf{C} = \mathbf{B} + \boldsymbol{\Phi} \operatorname{diag}(\boldsymbol{\alpha})^{-1} \boldsymbol{\Phi}^T, \tag{3.8}$$

$$\hat{\mathbf{t}} = [\boldsymbol{\Phi}^T \operatorname{diag}(\boldsymbol{\alpha})^{-1} \boldsymbol{\Phi}^T + \mathbf{B}^{-1}](\mathbf{t} - \mathbf{y}). \tag{3.9}$$

Here $\mathbf{B} = \operatorname{diag}(y_n(1 - y_n))$ is a matrix of heteroscedastic noise precisions required to map

the classification problem to a regression one. As the form of (3.8) is identical to (3.6), the fRVM scheme may also be used to learn $\boldsymbol{\alpha}$ for classification.

3.3. Measuring complexity of RVM classifiers

As mentioned, the ARD prior does not control model complexity enough to prevent, sometimes severe, overfitting; an issue that can become particularly apparent when using highly resolving basis functions. Schmolck and Everson [2007] show that for the case of regression sparsity and overfitting can be controlled effectively by a noise-dependent smoothness prior $p(\alpha_m \,|\, \sigma^2) \propto \exp\{-DF\}$.

Given the posterior over the weights in the RVM:

$$p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\mathbf{w}|\mathbf{t}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})}{p(\mathbf{t}|\boldsymbol{\alpha}, \sigma^2)} \tag{3.10}$$

$$= \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.11}$$

where

$$\boldsymbol{\Sigma} = (\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \mathrm{diag}(\boldsymbol{\alpha}))^{-1} \tag{3.12}$$

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\mathbf{t} \tag{3.13}$$

we obtain the mean posterior prediction:

$$\mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\mu} = (\boldsymbol{\Phi}\sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T)\mathbf{t} \equiv \mathbf{S}\mathbf{t} \tag{3.14}$$

where $\mathbf{S}$ is referred to as the *smoothing matrix* [Hastie and Tibshirani, 1990], which calculates the smoothed projection of $\mathbf{t}$.

It is possible to evaluate the degree of smoothing in the model using the degrees of freedom of $\mathbf{S}$, given by its trace:

$$DF = \mathrm{tr}\mathbf{S}. \tag{3.15}$$

In the case of orthogonal basis functions, such as wavelets, where:

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \mathbf{I}_M \tag{3.16}$$

then the trace of the smoothing matrix, $\mathbf{S}$, can be calculated as:

$$DF = \mathrm{tr}\mathbf{S} = \sum_{m=0}^{M} \frac{1}{1 + \sigma^2\alpha_m}. \tag{3.17}$$

showing that DF is counting the number of active basis functions where their level of activity is measured relative to the noise magnitude [Schmolck and Everson, 2007]. Basis functions with associated $\alpha_m \to \infty$ make no contribution to the degrees of freedom while basis functions with associated $\alpha_m = 0$ make a full contribution.

Unfortunately, this framework is difficult to apply to the RVM for the classification case [Schmolck, 2008]. This is primarily due to the association of an effective noise precision $y_n(1-y_n)$ with each observation, whereas the degrees of freedom in the smoothing matrix

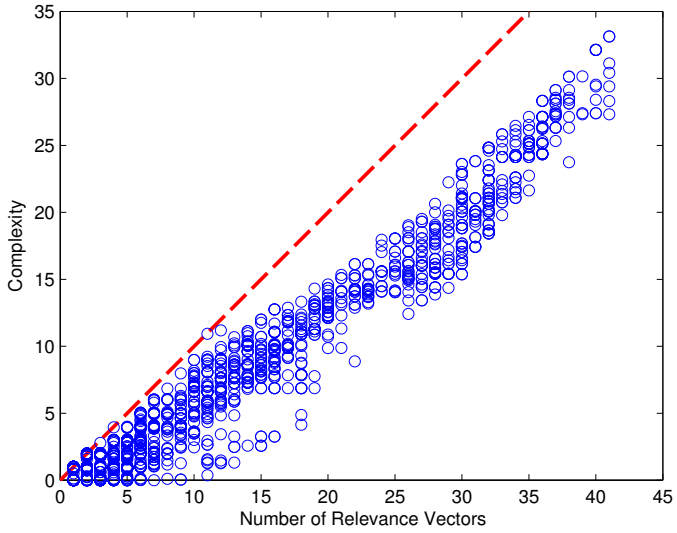


Figure 3.1.: Complexity vs. number of relevance vectors for EA front trained on the first fold of the Banana data. The red dashed line marks where Complexity is equal to the number of Relevance Vectors. The blue o's mark each of the solutions in the EA front.

are calculated in respect to a *single* observational noise variance. Merely averaging the heteroscedastic noises is ineffective in controlling model complexity and computation of the trace of the N by N smoothing matrix is computationally expensive. Put more simply: the noise appears in the rows of the design matrix, $\mathbf{\Phi}$ but we look to control model complexity by selecting basis functions i.e. the columns of $\mathbf{\Phi}$.

In order to measure RVM classifier complexity, $C(\boldsymbol{\alpha})$, the degrees of freedom of the smoothing matrix, as given by (3.17), is used:

$$C(\boldsymbol{\alpha}) = \sum_{m=0}^{M} \frac{1}{1 + \sigma^2 \alpha_m}. \tag{3.18}$$

From this, it can be seen that as $\alpha_m \to \infty$ so the associated basis function, $\boldsymbol{\phi}_m$, is "switched off" making no contribution to the complexity measure. Conversely, basis functions with small associated α_m contribute in full. Thus $C(\boldsymbol{\alpha})$ counts the number of active $\boldsymbol{\phi}_m$ and the magnitude of their activation. In figure 3.1 a scatter plot of complexity vs. number of relevance vectors is shown for solutions for the first fold of the Banana dataset located by the MOEA presented in this chapter, illustrating the close correlation of this complexity measure with the number of selected basis functions, or relevance vectors. At most, an individual active α_m can contribute 1 to the complexity measure and thus the scatter plot of complexity vs. relevance vectors seen in figure 3.1 is constrained to lie below the diagonal. Since α_m values can be set to partially turn on the associated relevance vectors, thus the number of apparently active basis functions in a model can be *greater* than the associated complexity value.

It should be noted that, despite σ^2 being poorly determined for classification problems, $C(\boldsymbol{\alpha})$ orders solutions by complexity independently of the value of σ^2. Thus, if $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ are two solutions so that

$$C(\boldsymbol{\alpha}, \sigma^2) < C(\boldsymbol{\alpha}', \sigma^2) \tag{3.19}$$

then since

$$C(\boldsymbol{\alpha}', r\sigma^2) - C(\boldsymbol{\alpha}, r\sigma^2) = \sum_m \frac{r\sigma^2(\alpha'_m - \alpha_m)}{(1 + r\sigma^2\alpha_m)(1 + r\sigma^2\alpha'_m)} \tag{3.20}$$

we have

$$C(\boldsymbol{\alpha}, r\sigma^2) < C(\boldsymbol{\alpha}', r\sigma^2) \text{ for all } r > 0. \tag{3.21}$$

Therefore it is proposed to choose $\boldsymbol{\alpha}$ by minimising $C(\boldsymbol{\alpha}, \sigma^2)$ with fixed $\sigma^2 = 1$ whilst simultaneously minimising the false positive rate and maximising the true positive rate.

As noted, greedy optimisation schemes similar to the RVM are not available using (3.17) and therefore evolutionary optimisation methods are examined, allowing for simultaneous optimisation of true and false positive rates and model complexity.

3.4. Evolving Sparse RVMs

Multi-objective optimising evolutionary algorithms [MOEAs, e.g. Deb, 2001; Coello et al., 2007] are a commonly used method to obtain the set of solutions containing the best trade-offs between multiple competing objectives. In [Suttorp and Igel, 2006] multi-objective optimisation was used to locate Support Vector Machine (SVM) classifiers giving the best trade-offs between true and false positive rates and complexity, as measured by the number of support vectors used. Similarly, Mierswa [2007] used multi-objective optimisation to simultaneously optimise SVM training error and model complexity by perturbing model hyper-parameters. Other examples of optimising model complexity against performance include the optimisation of Regression Error Characteristic curves over varying complexities of multi-layer perceptrons to identify the minimum level of complexity needed to achieve specific error combinations [Fieldsend, 2006]; the generation of neural network ensemble members by locating trade-offs between complexity and accuracy [Jin et al., 2004a]; and the optimisation of neural networks for forecasting in terms of complexity and accuracy [Fieldsend and Singh, 2004].

Here an MOEA is used to locate a set of RVM classifiers containing the best trade-offs between true and false positive rates along with the complexity measure previously introduced through the use of the concepts of dominance and Pareto optimality discussed previously in Chapter 2.4.1. In essence, the algorithm is looking to optimise ROC curves [Everson and Fieldsend, 2006c; Reckhouse et al., 2010] at each complexity value to achieve the best class separation possible at that complexity.

3.4.1. Multi-objective optimisation of RVM

The algorithm presented here makes use of multi-objective optimisation to identify the Pareto set in terms of $(T(\boldsymbol{\theta}), F(\boldsymbol{\theta}), C(\boldsymbol{\theta}))$. Using the dominates relation the algorithm compares solutions in terms of T, F and C, maintaining an elite archive, E, of non-dominated solutions approximating to the Pareto set. On each iteration of the optimisation process a member of E, $\boldsymbol{\theta}$, is selected and perturbed to create a new solution $\boldsymbol{\theta}'$. If none of the members of E dominate $\boldsymbol{\theta}'$ then it is added to E and any newly dominated elements of E are eliminated. Therefore, the archive can only approach the true Pareto front for the training data, $\mathcal{D}_{\text{tr}}$. The algorithm is outlined in algorithm 2.

Algorithm 2 Multi-Objective Evolutionary Algorithm

Require: $\mathcal{D}_{\text{tr}}$ *Data feature-target pairs*
$\boldsymbol{\alpha}$ *Initial solution*

1: $E := \texttt{initialise}(\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha})$ *Initialise the archive*
2: `while not converged`
3: $\boldsymbol{\alpha} := \texttt{select}(E)$ *Select $\boldsymbol{\alpha}$ from the set of unique $\boldsymbol{\alpha}$ in E*
4: $\boldsymbol{\alpha}' := \texttt{perturb}(\boldsymbol{\alpha})$ *Perturb the $\boldsymbol{\alpha}$ values*
5: $\mathbf{w} := \texttt{IRLS}(\boldsymbol{\alpha}, \mathcal{D}_{\text{tr}})$ *Use IRLS to learn $\mathbf{w}$*
6: `for` $\lambda' := 0 : \delta : 1$ *Thresholds λ'*
7: $\{F(\boldsymbol{\alpha}', \lambda'), T(\boldsymbol{\alpha}', \lambda'), C(\boldsymbol{\alpha}')\} := \texttt{evaluate}(\mathbf{w}, \boldsymbol{\alpha}', \lambda', \mathcal{D})$
8: $\boldsymbol{\theta}' := \{\boldsymbol{\alpha}', \lambda'\}$
9: $E := \texttt{nondom}(E \cup \{\boldsymbol{\theta}'\})$ *Update archive E*
10: `end`
11: `end`

For numerical convenience the values of all α_m are restricted to a finite range: $\alpha_{\min} \leq \alpha_m \leq \alpha_{\max}$. Should the value of an α_m exceed $\alpha_{\max}$ then the basis function is deemed to be "switched off" and so α_m is set to ∞. Conversely, where an α_m becomes less than $\alpha_{\min}$ its value is set to $\alpha_{\min}$. For the experimental results presented here $\alpha_{\min} = 10^{-12}$ and $\alpha_{\max} = 10^{12}$. It should be noted that results are not sensitive to the particular threshold values used.

Denoting the training features $\{\mathbf{x}\}_{n-1}^{N}$ and associated labels $\mathbf{t}$, which together form the training data $\mathcal{D}_{\text{tr}}$, the algorithm begins by using a provided $\boldsymbol{\alpha}$ evaluated on $\mathcal{D}_{\text{tr}}$ to initialise the elite archive, E. This could be a randomly generated vector of α_m values or an $\boldsymbol{\alpha}$ found by training an fRVM on $\mathcal{D}_{\text{tr}}$. Here, the algorithm is initialised using an $\boldsymbol{\alpha}$ with a randomly chosen $\alpha_m = \alpha_{\min}$ and all other $\alpha_n = \infty$ where $n \neq m$, 'switching off' those basis functions. This $\boldsymbol{\alpha}$ is evaluated over $0 \leq \lambda \leq 1$ in steps of size δ, in order to create a mutually non-dominating set E; the ROC curve for $\boldsymbol{\alpha}$.

On each iteration an $\boldsymbol{\alpha}$ is selected at random from the unique $\boldsymbol{\alpha}$s in the elite archive, E, copied and perturbed to form $\boldsymbol{\alpha}'$ (lines 3 and 4). Selecting from only the unique $\boldsymbol{\alpha}$ in E means that bias towards any particular combination is avoided. Between 1 and 3 components of the selected $\boldsymbol{\alpha}$ are then perturbed to form $\boldsymbol{\alpha}'$. These perturbations are carried out in one of the following ways, selected with equal probability: (i) an α_m corresponding to an active basis function is randomly selected from the active α_m and perturbed as follows: $\log_{10} \alpha_m' = \log_{10} \alpha_m + \epsilon$ where ϵ is drawn from a heavy-tailed Laplacian distribution, $p(\epsilon) \propto e^{-|\epsilon|/2}$ encouraging exploration [Yao et al., 1999]; (ii) a basis function selected at random from the active basis function is "switched off", setting α_m to ∞; (iii) a basis function is selected at random from the inactive basis functions and "switched on" by setting α_m to a number uniformly drawn in logarithmic space between $\alpha_{\min}$ and $\alpha_{\max}$.

Weights, $\mathbf{w}$, are trained based on $\boldsymbol{\alpha}'$ using iterative reweighted least squares (line 5), allowing the true positive rate, T, false positive rate, F, and complexity, C, to be evaluated over a range of thresholds, λ' (lines 6 and 7). A new perturbed parametrisation, $\boldsymbol{\theta}'$, is formed from each $(\boldsymbol{\alpha}', \lambda')$ pair which, if not dominated by any member of the elite archive, E, is added to E and any elements of E dominated by it are eliminated from the archive. In the algorithm these operations are represented by the $\texttt{nondom}(A)$ procedure which returns the mutually non-dominating elements of the set A.

At each iteration a large number of solutions, all of the same complexity, C, may be generated cheaply using the newly perturbed parametrisation, $\boldsymbol{\alpha}'$, simply by varying the threshold value, λ, making this a particularly efficient algorithm. Furthermore, the algorithm is able to make simultaneous perturbations of more than one α_m at each iteration, helping it avoid local maxima encountered by the fRVM. Since the fRVM can only perturb a single α_m at each step it may experience difficulties with exchanging a solution for a better one when the intermediate step is worse, a problem that the EA can avoid by perturbing multiple α_m on a given iteration. In the event that no new additions have be made to E for 20 iterations the algorithm attempts to force more diversity into the front by deliberately "switching on" and "off" α_ms. Should no additions have been made to the archive for over 100 iterations or the maximum number of allowed iterations reached then the algorithm stops.

While there are a number of existing MOEA implementations that could have been used here, such as SPEA2 [Zitzler et al., 2001], PAES [Knowles and Corne, 2000] or NSGA-II [Deb et al., 2000], concerns raised by Fieldsend et al. [2003] regarding the fixed archive sizes used by these algorithms are noted. In [Fieldsend et al., 2003] it was demonstrated that restriction of the number of solutions in the elite archive can lead to shrinking and oscillating/retreating estimated Pareto fronts. Thus, this elitist MOEA with an unconstrained archive size was implemented instead.

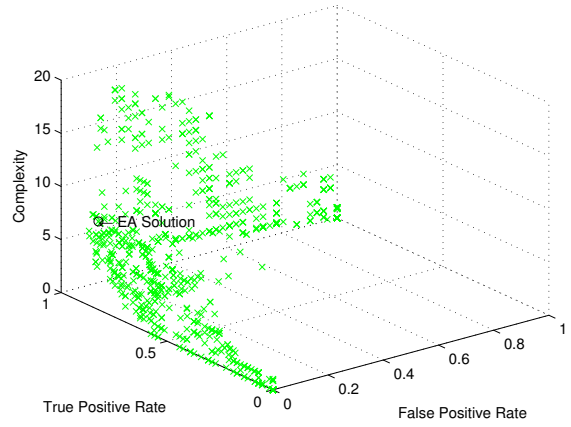
3.4.2. Illustration

Here an initial illustration of the MOEA is provided by training it and the fRVM on a subset of the Banana dataset [Rätsch et al., 2001]. This is a 2-dimensional dataset for which Rätsch et al. [2001] provide several standard partitions into training and test sets each containing 400 training and 4900 test examples with an overall positive class fraction of 44.7%. As a result, the dataset is one in which the training points are relatively closely packed making it relatively easy to approach the Bayes error rate.

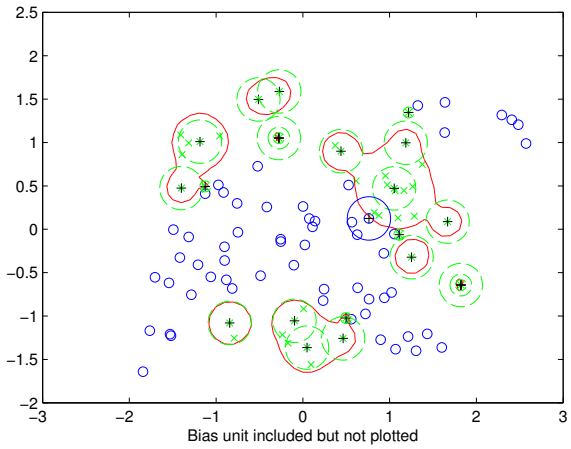
To provide a more challenging problem and demonstrate the efficacy of the MOEA when working with multi-resolution kernels, for each of the first ten folds of the Banana dataset a random subset of 100 points were selected from the training data in that fold, maintaining class balance, to form the "Banana Subset" dataset. The MOEA and fRVM were trained using Gaussian kernels centred on each of the training points $\phi_m(\mathbf{x}) \propto \exp\{-||\mathbf{x}-\mathbf{x}_m||^2/r^2\}$ using widths $r = r^*, r^*/2$ and $r^*/4$ where $r^* = 0.25$. These widths were deliberately chosen so as to present the situation in which a set of highly resolving kernels have been selected.

In figure 3.2a the Pareto front generated by the MOEA for the first fold of the Banana subset is shown. The front contains 798 solutions with true and false positive rates ranging from 0 to 1 and complexities ranging from 1.4120×10^{-12} to 18.1685. As a result of the algorithm generating multiple solutions from a *single* $\boldsymbol{\alpha}$ by varying the decision threshold, λ, bands of solutions can be seen at different complexity levels where several solutions based on the same $\boldsymbol{\alpha}$ combination have been accepted into the archive.

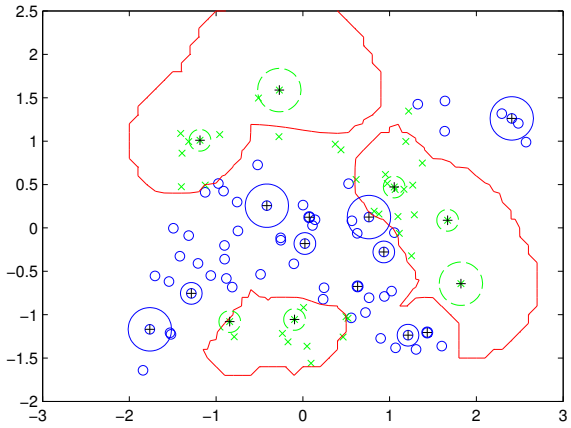
Despite having obtained an entire Pareto front consisting of 798 solutions with a range of true and false positive rates, in order to compare with the fRVM solution it is useful to select a single solution from it. In figure 3.2c the decision boundary for the MOEA solution



(a)



(b)



(c)

Figure 3.2.: (a): The approximate Pareto front of solutions trained by the MOEA on fold 1 of the Banana data subset. (b): Decision boundary for the fRVM solution. (c): Decision boundary for the EA solution with the highest accuracy on the training data. Training data are denoted by x's and o's with the positive class represented by the x's and negative by o's. Selected relevance vectors marked with +'s and the circles surrounding them represent the width of the selected kernel. Decision boundaries are marked with the red line

with the highest training accuracy (94%) is plotted and its location ($T = 0.9831, F = 0.1220, C = 5.0057$) marked on the the front in figure 3.2a.

Examining figures 3.2b and 3.2c, we can see that fRVM and MOEA solutions produce somewhat different decision boundaries. The fRVM has used more relevance vectors than the selected MOEA solution (25 as opposed to 18) and has clearly overfit to the training data, resulting in a decision boundary tightly wrapped around members of the positive class and only achieving a 68.08% test rate despite having a 99% training accuracy. In contrast, the sparser EA solution has generalised comparatively well and has selected basis functions centred on points from both classes, giving decision boundary less tightly wrapped around the selected relevance vectors and attaining a training rate of 94% and test rate of 84.63%.

Over the ten folds examined, the fRVM achieved an average training accuracy of 98.80% and test accuracy of 77.38% using an average of 28.40 relevance vectors. Meanwhile, the most accurate solutions on the training data selected from the fronts generated by the MOEA achieved an average training accuracy of 94.50% and test accuracy of 80.93% with an average of 21.00 relevance vectors. Thus the MOEA has produced slightly sparser solutions than the fRVM but with comparable accuracies and a smaller training-test rate discrepancy.

The large training-test accuracy discrepancy for the fRVM indicates that the fRVM has overfit to the data and therefore generalises poorly, a phenomenon also noticed for regression using multiple-scale kernels with the RVM [Schmolck and Everson, 2007]. Thus, even on this small example the fRVM can be seen to have overfit while the MOEA incorporates some regularisation through its use of the dominates relation to exclude from the elite archive, E, complex solutions that do not have better true and false positive rates than existing members.

When working with single-width kernels it is possible to carry out regularisation through "tuning" the kernel widths. Unfortunately this is not possible when employing *multiple-resolution* kernels. Despite the MOEA providing an entire Pareto front of solutions from which to choose, selection of the model with the highest *training* accuracy will lead to choosing models which have overfit to the training data. In fact, most solutions located by the MOEA with sufficiently high complexity will have overfit to the training data as these will be the ones which best represent the training data, thus maximising T and minimising F, but not necessarily generalising well. Despite $C(\boldsymbol{\alpha})$ measuring solution complexity it is not obvious how one should choose a suitable $C(\boldsymbol{\alpha})$ *a priori*. Therefore, in the following section a number of cross validation methods for controlling overfitting during the evolutionary optimisation process are examined.

3.5. Cross Validation Methods for EAs

In machine learning a common approach to controlling model complexity is to use a regularisation term that increasingly penalises the likelihood (or training error) as the model complexity increases. Typically, cross validation is used to determine the value of a hyperparameter used to control the degree of regularisation [Braga-Neto and Dougherty, 2004]. Bayesian methods avoid the need to calculate the best value by setting priors

for the problem parameters and averaging over posterior distributions. However, maximum *a posteriori* learning can be regarded as a penalised likelihood scheme and as such care may be required when setting the prior (hyper-)parameters in order to achieve good generalisation.

Since it would be computationally expensive to perform multiple runs of the MOEA to determine the best regularising hyper-parameter, two different cross validation schemes are examined for their effect on preventing solutions in the elite archive, E, overfitting during the optimisation process. The first scheme examined here splits the data, $\mathcal{D}$, into training and validation datasets, $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, and maintains separate training and validation solution archives, E_{tr} and E_{val} corresponding to $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ respectively. Several different methods for carrying out the dataset splitting and archive maintenance for this scheme are presented. The second scheme examined maintains a single archive of solutions and evaluates an average true and false positive rate based on a K-fold splitting of the training data. Related work in which validation sets are used to assess solutions optimised in terms of performance and complexity includes the use of a validation set to identify when neural networks optimised in terms of complexity and mean squared error have overfit to a training set [Fieldsend and Singh, 2004; Fieldsend and Everson, 2008a] and using a validation set to assess performance of ensembles of neural networks generated using an MOEA [Jin et al., 2004b]. It is emphasised though that in those cases solution assessment on a validation set is carried out *after* the MOEA has located its estimate of the Pareto front rather than as part of the optimisation process itself.

3.5.1. Two-archive methods

All of the two-archive methods presented here depend on a partitioning of the training data into a training, $\mathcal{D}_{\text{tr}}$, and validation, $\mathcal{D}_{\text{val}}$, split whilst maintaining separate elite archives, E_{tr} and E_{val}, for each. On each iteration of the EA, when a new solution is added to E_{tr}, the training archive, the algorithm also tries to add it to E_{val}, the validation archive. All these methods follow the same general structure as given in algorithm 3; a modification of algorithm 2. The difference between the implementations is the method used for carrying out the data partitioning into $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ and management of the associated archives E_{tr} and E_{val}. In algorithm 3 these operations are summarised by the `CrossVal` procedure found on line 15.

The algorithm begins (line 1) by carrying out the partitioning of the data into the two equally-sized training and validation datasets, $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$. Using an initial solution, $\boldsymbol{\alpha}$, archives of mutually non-dominating $(\boldsymbol{\alpha}, \lambda)$ pairs, E_{tr} and E_{val}, are initialised over a range of λ values using the associated datasets (lines 2-3).

As with algorithm 2, the algorithm proceeds by selecting, copying and perturbing an $\boldsymbol{\alpha}$ from the set of unique $\boldsymbol{\alpha}$'s in E_{tr}. True and false positive rates and complexity are evaluated on the training partition, $\mathcal{D}_{\text{tr}}$, over a range of λ' values by calculating the corresponding weights, $\mathbf{w}$, using iterated reweighted least squares (IRLS) based on $\mathcal{D}_{\text{tr}}$ and $\boldsymbol{\alpha}'$ (line 9). Where $(\boldsymbol{\alpha}', \lambda')$ is non-dominated by any member of E_{tr} then it is added to E_{tr} and any newly dominated solutions in it deleted (line 12). Following successful addition to E_{tr} the $(\boldsymbol{\alpha}', \lambda)$ pair is then evaluated for addition to E_{val} by calculating true and false positive rates for the solution on $\mathcal{D}_{\text{val}}$ using $\mathbf{w}$ previously calculated on $\mathcal{D}_{\text{tr}}$ using IRLS. Should

Algorithm 3 Two-Archive Cross Validation Evolutionary Algorithm

Require: $\mathcal{D}$ *Data feature-target pairs*
$\boldsymbol{\alpha}$ *Initial solution*

1: $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}} = \texttt{split}(\mathcal{D})$ *Training and validation data*
2: $E_{\text{tr}} := \texttt{initialise}(\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha})$ *Training archive*
3: $E_{\text{val}} := \texttt{initialise}(\mathcal{D}_{\text{val}}, \boldsymbol{\alpha})$ *Validation archive*
4: `while not converged`
5: $\quad\boldsymbol{\alpha} := \texttt{select}(E_{\text{tr}})$
6: $\quad\boldsymbol{\alpha}' := \texttt{perturb}(\boldsymbol{\alpha})$
7: $\quad\mathbf{w} := \texttt{IRLS}(\boldsymbol{\alpha}, \mathcal{D}_{\text{tr}})$ *Use IRLS to learn* $\mathbf{w}$
8: $\quad$`for` $\lambda' := 0 : \delta : 1$ *Thresholds,* λ
9: $\quad\quad\{F(\boldsymbol{\alpha}', \lambda'), T(\boldsymbol{\alpha}', \lambda'), C(\boldsymbol{\alpha}')\} := \texttt{evaluate}(\mathbf{w}, \boldsymbol{\alpha}', \lambda', \mathcal{D}_{\text{tr}})$
10: $\quad\quad\boldsymbol{\theta}' := (\boldsymbol{\alpha}', \lambda')$
11: $\quad\quad$`if` $\neg(E_{\text{tr}} \prec \boldsymbol{\theta}')$ *If* $\boldsymbol{\theta}'$ *not dominated by any element of* E_{tr}
12: $\quad\quad\quad E_{\text{tr}} := \texttt{nondom}(E_{\text{tr}} \cup \{\theta'\})$ *Update* E_{tr}
13: $\quad\quad\quad\{F_{val}(\boldsymbol{\alpha}', \lambda'), T_{val}(\boldsymbol{\alpha}', \lambda')\} := \texttt{evaluate}(\boldsymbol{\alpha}', \lambda', \mathcal{D}_{\text{val}})$
14: $\quad\quad\quad$`if` $\neg(E_{\text{val}} \prec \boldsymbol{\theta}')$ *If* $\boldsymbol{\theta}'$ *not dominated by any element of* E_{val}
15: $\quad\quad\quad\quad E_{\text{val}} := \texttt{nondom}(E_{\text{val}} \cup \{\theta'\})$ *Update* E_{val}
16: $\quad\quad\quad\quad\{E_{\text{tr}}, E_{\text{val}}, \mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}}\} := \texttt{CrossVal}(E_{\text{tr}}, E_{\text{val}}, \mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}})$
17: $\quad\quad\quad$`end`
18: $\quad\quad$`end`
19: $\quad$`end`
20: `end`

the solution be non-dominated by any member of E_{val} then it is added to E_{val} and any newly dominated members of E_{val} deleted (line 15). Finally, a possible new $E_{\text{tr}}, \mathcal{D}_{\text{tr}}, E_{\text{val}}$ and $\mathcal{D}_{\text{val}}$ are chosen (line 16) using the `CrossVal` procedure of the chosen method.

Using this algorithm as an outline, a number of two-archive cross validation procedures are now presented whose efficacy is discussed later in section 3.6.

EA Swap

Perhaps the simplest and most straightforward cross validation scheme would be to divide the data, $\mathcal{D}$, into training and validation sets and learn $\boldsymbol{\alpha}$ based on $\mathcal{D}_{\text{tr}}$ and E_{tr} at each iteration, only accepting solutions into E_{val} if non-dominated by other solutions in E_{val}. Unfortunately, as a consequence of this, the likely effect would be to overfit solutions to the *validation data*. Therefore, in this method, the roles of the training data and archives are swapped each time a new solution is added to to validation archive.

Upon initialisation of the EA, $\mathcal{D}$ is randomly partitioned into two equal-sized splits, $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ (line 1). The `CrossVal` procedure at line 16 of algorithm 3 is therefore:

16.1: $E_{\text{tr}}, E_{\text{val}} := \texttt{swap}(E_{\text{tr}}, E_{\text{val}})$
16.2: $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}} := \texttt{swap}(\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}})$

It should be highlighted that whatever the initial data split made by the algorithm on initialisation, the partitions are maintained throughout the optimisation process; only the role of the data splits change.

EA Shuffle

A possible consequence arising from the initial partitioning of the data by the swapping procedure is that, in randomly splitting it, a non-representative split may be produced for training and validation, resulting in poorly generalising solutions. To combat this, and further reduce overfitting to a particular dataset, this method re-partitions $\mathcal{D}$ into new $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ sets whenever a new solution is added to the validation archive, E_{val}. Solutions that have overfit to a specific training dataset are likely to be eliminated by the repartitioning of the data while solutions that generalise well should survive and remain in E_{tr} and E_{val}. The following procedure is carried out by the `CrossVal` procedure at line 15 whenever a new solution enters the current validation archive:

16.1: $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}} := \texttt{split}(\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{val}})$ *Repartition at random*
16.2: $E'_{\text{tr}} := E_{\text{val}} := \emptyset$
16.3: `for` $\boldsymbol{\theta}$ `in` E_{tr}
16.4: $\quad \boldsymbol{\alpha}, \lambda := \boldsymbol{\theta}$
16.5: $\quad F(\boldsymbol{\alpha}, \lambda), T(\boldsymbol{\alpha}, \lambda), C(\boldsymbol{\alpha}) := \texttt{evaluate}(\boldsymbol{\alpha}, \lambda, \mathcal{D}_{\text{tr}})$
16.6: $\quad$ `if` $\neg(E'_{\text{tr}} \prec \boldsymbol{\theta})$ *If $\boldsymbol{\theta}$ is not dominated by elements of E'_{tr}*
16.7: $\quad\quad E'_{\text{tr}} := \texttt{nondom}(E'_{\text{tr}} \cup \{\boldsymbol{\theta}\})$ *Add to E'_{tr}*
16.8: $\quad\quad F(\boldsymbol{\alpha}, \lambda), T(\boldsymbol{\alpha}, \lambda), C(\boldsymbol{\alpha}) := \texttt{evaluate}(\boldsymbol{\alpha}, \lambda, \mathcal{D}_{\text{val}})$
16.9: $\quad\quad E'_{\text{val}} := \texttt{nondom}(E'_{\text{val}} \cup \{\boldsymbol{\theta}\})$ *Add to E_{val} if not dominated*
16.10: $\quad$ `end`
16.11: `end`
16.12: $E_{\text{tr}} := E'_{\text{tr}}$

Whilst the EA Swap method swapped $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, along with their associated archives E_{tr} and E_{val}, the shuffling method presented here recombines all the data and randomly splits it into new training and validation partitions. Members of E_{tr} have their true and false positive rates re-evaluated on the *new* training dataset in order to form a new training archive, E_{tr}, of mutually non-dominating solutions. Note that complexity is dependent only on the $\boldsymbol{\alpha}$ values of the solutions so does not need to be re-calculated. Solutions that are accepted into E_{tr} are evaluated on $\mathcal{D}_{\text{val}}$, the validation dataset, to form a new mutually non-dominating validation archive, E_{val}. The optimisation process proceeds as before and this shuffling procedure is carried out upon addition of a new solution to E_{val}.

EA Stratified Shuffle

While the shuffling procedure described above should reduce overfitting through its random repartitioning of the data, a drawback is that the generated partitions may have non-representative class balances. Therefore the shuffling procedure is refined so that whenever data is split, each class is divided up separately so as to maintain class proportions when creating $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$.

EA Homogeneous Shuffle

Although the EA Stratified Shuffle method ensures that class proportions are preserved when creating $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, and thus when optimising E_{tr} and E_{val}, there is the potential

Algorithm 4 K-Fold Multi-Objective Evolutionary Algorithm

$\{\mathcal{D}^{(k)}\} := \texttt{split}(\mathcal{D})$ $\quad k = 1, \ldots, K$
$E := \texttt{initialise}(\{\mathcal{D}^{(k)}\}, \boldsymbol{\alpha})$ $\quad$ *Initialise archive*
`while not converged`
  $\boldsymbol{\alpha} := \texttt{select}(E)$
  $\boldsymbol{\alpha}' := \texttt{perturb}(\boldsymbol{\alpha})$
  $C(\boldsymbol{\alpha}') := \texttt{complexity}(\boldsymbol{\alpha}')$ $\quad$ *Complexity of $\boldsymbol{\alpha}'$ using (3.17)*
  `for` $\lambda' := 0 : \delta : 1$ $\quad$ *Thresholds, λ'*
    `for` $k = 1, \ldots, K$
      $\mathbf{w} := \texttt{IRLS}(\boldsymbol{\alpha}', \mathcal{D}_{\setminus k})$ $\quad$ *Find weights*
      $\{F_k(\boldsymbol{\alpha}', \lambda'), T_k(\boldsymbol{\alpha}', \lambda')\} := \texttt{evaluate}(\boldsymbol{\alpha}', \lambda', \mathbf{w}|\mathcal{D}_k)$
    `end`
    $F(\boldsymbol{\alpha}', \lambda') := \frac{1}{K}\sum_{k=1} F_k$ $\quad$ *Average T and F*
    $T(\boldsymbol{\alpha}', \lambda') := \frac{1}{K}\sum_{k=1} T_k$
    $\boldsymbol{\theta}' := \{\boldsymbol{\alpha}', \lambda'\}$
    $E := \texttt{nondom}(E \cup \{\boldsymbol{\theta}'\})$ $\quad$ *Update archive*
  `end`
`end`

that, particularly for small datasets, the partitioning may result in quite dissimilar data splits. For example, where data may actually consist of two clusters a poor split might assign most of one cluster to $\mathcal{D}_{\text{tr}}$ and most of the other to $\mathcal{D}_{\text{val}}$. Even though repeated re-partitioning of $\mathcal{D}$ into new $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ splits will lead to the discarding of poor partitions, at a particular iteration a poor partition could result in a drastic reduction in the number of members of one of the archive and therefore inhibit convergence.

In order to address this the "nearly homogeneous partitioning" proposed by Aupetit [2009] is implemented to carry out the splitting of $\mathcal{D}$ into $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$. Starting at a randomly chosen data point, the homogeneous partitioning algorithm plots a path through the dataset from nearest neighbour to nearest neighbour assigning alternate points on the path to $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, resulting in training and validation splits with similar statistics. As per the recommendation by Aupetit [2009], the partitioning procedure is initialised by first selecting a random point from $\mathcal{D}$ and then using the point *furthest* from it as the algorithm's starting point to reduce chances of partitioning being carried out along a class boundary which may result in classes being divided between two separated partitions.

3.5.2. K-fold Cross Validation

K-fold cross validation is a method commonly used in statistical pattern recognition in order to ascertain the best regularisation parameter(s) [Devijver and Kittler, 1982]. Training data is randomly partitioned into K disjoint folds so that each of the K folds can be used as a surrogate test set, the *validation* set, in order to estimate the generalisation error of a model trained on the remaining $K - 1$ folds. Regularisation parameters that result in the lowest validation errors can then be identified and used. By averaging the validation error over the K validation folds the variance in the error estimate is reduced; Braga-Neto and Dougherty [2004] empirically find that 5 or 10 folds are enough.

The second cross validation scheme presented here treats the α_m, the ARD parameters, as regularising hyperparameters and searches for those which produce the best generalisa-

tion performance. Unlike the previous scheme, wherein separate training and validation archives were maintained, a single archive is maintained into which solutions are only accepted if they are non-dominated by members of the archive as measured in terms of average performance over K validation sets. This is implemented through a simple modification of algorithm 2 to form algorithm 4.

Using Aupetit's method [Aupetit, 2009], the algorithm initialises by splitting the data into K "nearly homogeneous" folds $\mathcal{D}^{(k)}$. An initial solution, $\boldsymbol{\alpha}$, has its performance evaluated on these K folds and is used to initialise the archive E (line 2). As with the original algorithm, it then proceeds to select a unique $\boldsymbol{\alpha}$ from the archive E and perturbs it to form $\boldsymbol{\alpha}'$ (lines 4 - 5). The algorithm then loops through each of the K folds, using iteratively reweighted least squares (IRLS) to train weights, $\mathbf{w}$, based on $\boldsymbol{\alpha}'$ based on data from all but the k^{th} fold evaluating true and false positive rates on the k^{th} fold (lines 8 - 11). The true and false positive rates found are then averaged over the number of folds used (lines 12 - 13) and if the solution is non-dominated by any member of the archive E in terms of these average T and F as well as complexity C (which depends only on $\boldsymbol{\alpha}'$ so does not change over the folds), is then added to it. Any solutions in E dominated by it are deleted from the archive.

3.6. Benchmark Dataset Results

Experiments were performed with the MOEA and its variants and the results compared with those of the fRVM on a number of standard benchmark datasets. As previously discussed, the efficacy of the RVM is linked with the regularisation provided by the kernels used. In [Tipping, 2001] results are presented comparing the RVM with the SVM in which the RVM obtained better training accuracy and sparsity than the SVM. Therefore, initial experiments were carried out using the fRVM for each of the benchmark datasets investigated here in which the kernel widths were adjusted so as to produce equivalent sparsity and test error rates with the fRVM as those given in Tipping [2001]. This tuned kernel width, denoted here by r^*, was used as the basis upon which to construct an over-complete dictionary comprising of kernels of widths r^*, $r^*/2$ and $r^*/4$ centred on each training point. This set of multi-resolution kernels was then used to train RVMs, using the fRVM methodology, and the various MOEA implementations were used to locate the approximate true positive vs false positive vs complexity Pareto front. In the case of the K-fold cross validation scheme the optimisation was carried out using 2, 5 and 10 folds.

To enable comparison with the results in [Tipping, 2001], the datasets assembled by Rätsch et al. [2001] were used; details of these can be found in table 3.1. In addition, to assess algorithm performance when dealing with more extreme cases of class imbalance, the MNIST numerals dataset [Lecun et al., 1998] was used with the goal of distinguishing between images of the number 5 and images of the other numerals. Of the 60000 available training samples only 1000 were used, drawn in equal proportions from each of the 10 digit classes (0 - 9). Additionally, the images were down-sampled from 28×28 pixels to 13×13 pixels and a Gaussian kernel was employed.

In figure 3.3 fronts of solutions generated by the MOEA are plotted for the first folds of the Banana, Breast Cancer, Titanic, Waveform, German and Pima datasets. From

Data Set	No. Training	No. Testing	Dimensions	Positive class fraction (%)
Pima	200	332	7	33.27
Banana	400	4900	2	44.70
Banana Subset	100	4900	2	44.70
Breast Cancer	200	77	9	30.65
Titanic	150	2051	3	32.30
Waveform	400	4600	21	32.95
German	700	300	20	29.93
MNIST: 5 v All	1000	10000	169	9.04

Table 3.1.: Details of benchmark datasets. Note that for the MNIST data only 1000 training points were used and the data downsampled, reducing the number of features to 169.

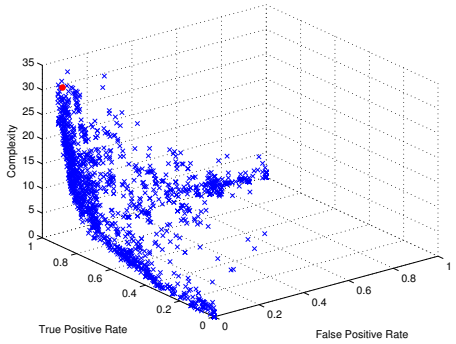
these plots we see the MOEA has successfully located solutions over a range of true and false positive rates and complexities, effectively optimising ROC curves at each complexity value. In general, as the complexity value increases, so the ROC curves get closer to the optimal operating point ($T = 1$, $F = 0$), as seen in figures 3.3a, 3.3b, 3.3d and 3.3f, and to a lesser extent in figure 3.3e. On the datasets which are harder to classify, Waveform and German, we see less of a movement towards the optimal operating point with ROC curves of increasing complexity. Instead, it appears that certain operating points on the ROC curve can only be obtained by increasingly complex solutions. This particularly appears to be the case for the Titanic dataset (figure 3.3c) in which a much sparser front of solutions is obtained than for any of the other datasets.

Despite providing an approximation to the Pareto front consisting of solutions over a range of varying true and false positive rates as well as complexity, a single solution generated by each MOEA method had to be selected in order to allow a direct comparison with the fRVM. In order to do so the EA solution from the generated fronts which had the highest accuracy over *all* the training data in that fold was selected, i.e. where training data had been split for cross validation it was recombined along with the archives and the solution which performed best *overall* in terms of accuracy was selected. Results comparing the selected solutions from the MOEA schemes and the fRVM, with the highest performing method highlighted in bold, are presented in table 3.2. Average accuracies over 10 splits of the data (as provided by Rätsch et al. [2001][1]) are given along with their standard deviations. Test accuracies for the Banana subset, Breast Cancer, Titanic and Pima datasets were indistinguishable at the 5% significance level using the Mann-Whitney-Wilcoxon test. In the case of the complete Banana data, the two-archive methods' performances were significantly different from that of the fRVM, whilst for the German data the basic MOEA and K-fold cross validation methods are indistinguishable from the fRVM, but some two-archive methods produced significantly different results. On the Waveform data, the fRVM achieved significantly better accuracies than the various EA schemes. Therefore, the MOEA methods have located solutions of comparable accuracy

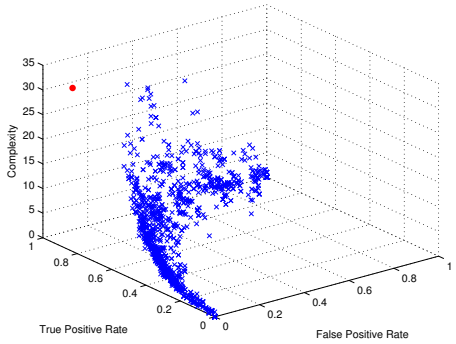
[1]The Pima data, a popular dataset available at `http://www.stats.ox.ac.uk/pub/PRNN`, is not part of the collection provided by Rätsch et al. [2001]. Results are given for the split used by Ripley [1996] and by Tipping [2001] and also for 9 random splits thereof for which the number of training and test examples were of equal size to those in Ripley's split.

	Banana			Banana Subset		
Classifier	Train (%)	Test (%)	RVs	Train (%)	Test (%)	RVs
fRVM	94.05 (1.09)	**88.59 (0.75)**	29.70 (4.14)	98.80 (1.87)	77.38 (6.72)	28.40 (7.06)
EA RVM	93.28 (1.27)	87.83 (0.87)	25.90 (5.57)	94.50 (1.78)	80.93 (2.20)	21.00 (5.87)
Swap	91.58 (1.34)	86.64 (1.04)	21.70 (6.93)	87.30 (2.21)	77.91 (2.62)	11.90 (3.66)
Shuffle	89.85 (2.32)	85.68 (1.80)	15.30 (5.42)	87.60 (3.13)	75.07 (3.46)	8.50 (3.34)
Shuffle Strat	91.00 (1.81)	87.45 (1.74)	17.50 (5.30)	87.40 (2.50)	75.04 (5.17)	8.60 (3.03)
H. Shuffle	91.43 (1.66)	86.97 (1.53)	15.70 (7.90)	86.70 (4.11)	74.78 (6.42)	9.20 (5.03)
2 Fold	92.48 (1.03)	88.09 (1.11)	25.30 (6.25)	93.60 (2.37)	**82.37 (2.31)**	17.20 (4.54)
5 Fold	92.73 (1.19)	87.26 (0.68)	27.20 (4.16)	93.60 (2.50)	81.31 (1.72)	16.20 (6.39)
10 Fold	92.97 (1.08)	88.19 (0.99)	24.70 (7.80)	92.70 (1.49)	80.29 (1.58)	16.80 (6.12)
	Breast Cancer			Titanic		
Classifier	Train (%)	Test (%)	RVs	Train (%)	Test (%)	RVs
fRVM	93.25 (1.34)	69.35 (4.42)	36.30 (3.83)	79.80 (2.95)	**77.71 (1.38)**	69.90 (22.12)
EA RVM	83.45 (1.52)	68.83 (5.61)	19.90 (8.23)	80.00 (2.85)	77.54 (1.37)	7.90 (3.00)
Swap	80.05 (1.26)	70.26 (5.35)	13.70 (8.06)	79.60 (2.48)	77.22 (1.25)	5.90 (2.73)
Shuffle	79.75 (1.74)	70.13 (4.66)	8.60 (4.12)	79.87 (2.77)	77.54 (1.35)	5.80 (2.25)
Shuffle Strat	79.10 (1.37)	**71.56 (5.03)**	7.20 (2.30)	79.67 (2.69)	77.13 (1.63)	6.80 (4.34)
H. Shuffle	80.50 (1.31)	68.31 (5.52)	10.50 (3.50)	79.60 (2.78)	77.09 (1.67)	6.90 (2.69)
2 Fold	81.50 (1.49)	66.10 (6.61)	14.80 (4.66)	79.87 (2.77)	77.54 (1.38)	7.70 (3.06)
5 Fold	82.25 (1.67)	70.26 (5.59)	15.60 (6.42)	79.87 (2.77)	77.27 (1.78)	6.90 (3.93)
10 Fold	82.15 (1.63)	69.48 (4.21)	15.50 (6.29)	79.87 (2.77)	77.22 (1.77)	7.20 (2.39)
	Waveform			German		
Classifier	Train (%)	Test (%)	RVs	Train (%)	Test (%)	RVs
fRVM	100 (0)	**89.42 (0.59)**	35.90 (5.13)	98.60 (0.48)	**76.13 (1.92)**	137.90 (4.20)
EA RVM	94.83 (1.15)	88.47 (0.73)	28.90 (4.38)	79.44 (0.96)	75.13 (2.23)	30.10 (6.31)
Swap	92.18 (1.62)	87.34 (0.61)	13.20 (6.88)	78.21 (0.97)	73.83 (2.17)	17.60 (6.88)
Shuffle	91.88 (1.70)	87.47 (0.59)	13.20 (6.23)	75.63 (1.01)	74.17 (2.35)	12.00 (8.58)
Shuffle Strat	91.25 (1.47)	88.10 (0.64)	9.10 (4.77)	76.26 (1.07)	74.57 (2.73)	12.20 (6.58)
H. Shuffle	92.10 (1.51)	88.20 (0.38)	12.70 (2.87)	76.29 (0.95)	73.97 (2.34)	11.00 (5.66)
2 Fold	93.27 (1.27)	88.38 (0.75)	18.30 (4.90)	78.63 (0.88)	75.80 (2.36)	17.70 (4.64)
5 Fold	93.68 (0.93)	88.34 (0.91)	18.70 (8.27)	78.77 (0.94)	75.23 (2.35)	24.90 (7.19)
10 Fold	93.53 (1.30)	88.36 (0.43)	18.90 (4.93)	78.79 (0.96)	75.37 (2.72)	22.00 (7.23)
	Pima			MNIST 5 v All		
Classifier	Train (%)	Test (%)	RVs	Train (%)	Test (%)	RVs
fRVM	93.25 (1.92)	**75.93 (1.32)**	31.40 (4.27)	100	**97.36**	61
EA RVM	87.05 (1.82)	75.00 (2.89)	26.20 (7.48)	96.70	95.36	28
Swap	82.85 (1.65)	73.98 (2.97)	12.80 (5.20)	96.00	94.40	20
Shuffle	81.35 (3.94)	75.45 (2.68)	7.80 (4.32)	97.50	95.44	23
Shuffle Strat	82.15 (2.42)	75.27 (2.19)	10.80 (3.77)	95.40	94.93	19
H. Shuffle	81.25 (2.79)	**75.93 (1.23)**	5.90 (2.85)	96.40	94.76	19
2 Fold	83.55 (2.89)	75.27 (2.16)	12.00 (4.47)	97.30	96.13	35
5 Fold	84.10 (2.65)	75.51 (2.45)	14.80 (4.59)	96.40	93.84	17
10 Fold	84.65 (2.40)	75.75 (2.24)	14.60 (4.20)	95.70	93.88	15

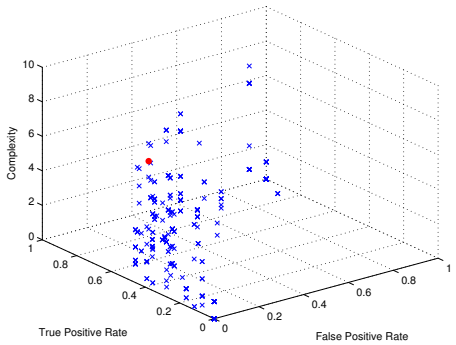
Table 3.2.: Average training and test accuracies and number of relevance vectors used for the fRVM and EA variants for solutions with highest accuracy in training archive. Standard deviations over 10 folds are given in brackets. Note that the MNIST dataset has a single partitioning and therefore no standard deviations.



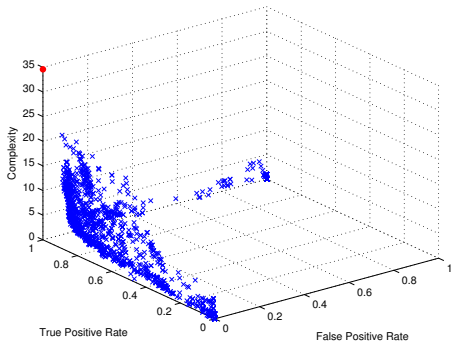
(a) Banana



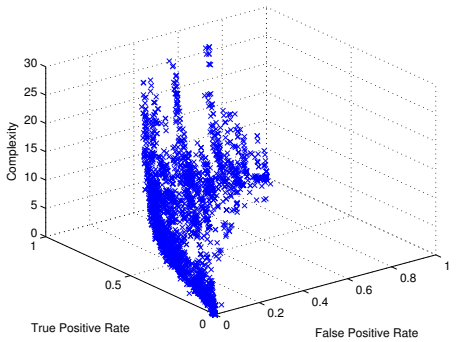
(b) Breast Cancer



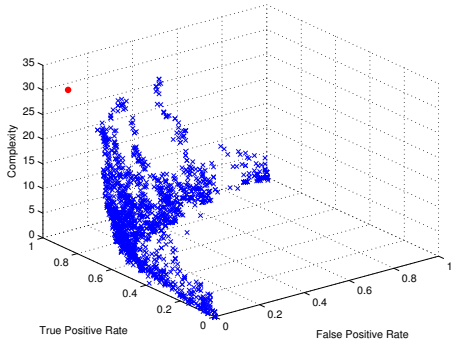
(c) Titanic



(d) Waveform



(e) German



(f) Pima

Figure 3.3.: Solutions located for each of the first folds of the benchmark datasets. EA solutions are marked with blue x's and fRVM solutions are marked with red filled o's. Since the fRVM solution is considerably more complex than those located by the EA RVM it is not plotted here. A plot for the German data with the fRVM solution marked can be found in figure A.1 of appendix A.

to those learnt by the fRVM but with markedly lower complexity. Particular attention should be drawn to the Titanic and German dataset results where solutions of comparable accuracy to the fRVM were found by the MOEAs but which used much lower numbers of relevance vectors to achieve them.

In general, the training and test accuracies of the solutions selected from the fronts generated by the basic MOEA are lower than those of the fRVM alone whilst their training-test discrepancies are smaller than those of the fRVM, which appears to overfit to the training data. The basic MOEA provides some regularisation of the generated RVMs since solutions cannot enter the elite archive, E, if an existing member has equal or better true and false positive rates and has lower complexity, ensuring that only low complexity

solutions make it into the archive. The most severe regularisation is generally provided by the homogeneous and stratified shuffle methods; the results suggest that this regularisation is often too severe. It is noted that the basic MOEA is not always finding a solution with training accuracy equal to or greater than that of the fRVM solution despite it lying within the search space. This is investigated later in section 3.7.

In addition to locating single sparse solutions of high accuracy, the multi-objective schemes presented provide a range of solutions over the full range of true and false positive rates. In order to compare these, the areas dominated by the Pareto fronts in the false positive - true positive planes are calculated and presented in table 3.3. The calculated areas are analogous to the area under the ROC curve and thus represent the classifier's ability to separate the two classes [Hanley and McNeil, 1982; Hand and Till, 2001; Fieldsend and Everson, 2008a]. For all cases results are presented for both the training and test data. The projection of the Pareto fronts into the false positive - true positive plane is carried out by ignoring the complexity, effectively flattening the front, and finding the non-dominated set of points on the *training* data. The area dominated by this same set of solutions on the *test* data is reported in the column labelled 'Test'. The calculated area is that strictly dominated by the front, i.e., the area dominated by the attainment surface [Zitzler, 1999; Grunet da Fonseca et al., 2001]. The results shown in the table indicate that the MOEA schemes locate a range of solutions that perform as well as, if not better than, those found by the fRVM. Of the various MOEA schemes presented, it is the K-fold cross-validation method that performs best, usually outperforming the base MOEA and two-archive methods.

3.7. Locating fRVM equivalent solutions

In the benchmark experiments carried out in section 3.6 it was noted that the fronts of solutions located by the EA RVM did not always contain a solution with equal or better training accuracy than the fRVM, despite it being within in the set of possible solutions[2]. In this section an investigation is carried out to see if this is due to the perturbation approach being carried out for which experiments are carried out using two of the benchmark datasets: Banana and Titanic.

One possible explanation for the EA RVM not consistently locating solutions with equivalent training accuracy to the fRVM is that the small number of mutations being carried out at each iteration, 1-3 possible mutations, has resulted in it either becoming stuck in local minima or unable to traverse gaps in the search space. In order to determine if this was the case, the EA RVM was run with an increased number of potential mutations per iteration: 1-10 mutations per iteration. In table 3.4 the training and test accuracies obtained by the most accurate solution located by the EA RVM with this increased number of mutations are presented. In both cases solutions with training accuracy greater than that of the fRVM solution have been located whose accuracy dropped on application to the test data. In the case of the Banana data this increased training accuracy has come with an increased number of relevance vectors while on the Titanic data the EA RVM located

[2]N.B. this is not the case for the cross validation EA implementations as only a subset of the data is available for training and evaluation at a given iteration.

	Banana		Breast Cancer		Titanic		Waveform		German		Pima		MNIST: 5 v All	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
fRVM	0.99	**0.95**	0.98	0.68	0.68	0.64	1.00	0.94	1.00	0.75	0.98	0.81	1.00	**0.94**
EA RVM	0.98	0.94	0.86	**0.71**	0.74	0.70	0.98	**0.95**	0.82	**0.77**	0.93	**0.82**	0.87	0.90
Swap	0.94	0.92	0.78	0.68	0.70	0.69	0.96	0.94	0.78	0.75	0.85	0.79	0.86	0.90
Shuffle	0.89	0.92	0.76	0.68	0.71	0.69	0.95	0.94	0.69	0.74	0.83	0.80	0.77	0.90
Shuffle Strat.	0.90	0.92	0.75	0.68	0.71	0.69	0.95	0.94	0.70	0.73	0.84	0.80	0.70	0.89
H. Shuffle	0.91	0.93	0.77	0.68	0.70	0.70	0.95	0.94	0.70	0.73	0.85	0.81	0.74	0.89
2 Fold	0.96	0.93	0.82	0.69	0.72	0.70	0.98	**0.95**	0.79	**0.77**	0.88	0.81	0.84	0.91
5 Fold	0.97	0.94	0.83	0.70	0.72	**0.71**	0.98	0.85	0.80	**0.77**	0.89	**0.82**	0.90	0.90
10 Fold	0.97	0.94	0.82	0.70	0.72	0.70	0.98	**0.95**	0.80	**0.77**	0.89	**0.82**	0.83	0.88

Table 3.3.: Average area under the curve for the training and test fronts.

solutions are much sparser than the fRVM (average 5.2 RVs) whilst still locating solutions of higher training accuracy than the fRVM. This, again, demonstrates that use of the complexity measure as an objective results in the penalisation of more complex solutions in favour of less complex ones with equal or better performance. As before results are also presented in terms of AUC by flattening the front of solutions into the TP-FP plane and selecting those which are Pareto optimal with which to calculate training and test AUC values. For the Banana dataset we see an increase in training AUC from the 0.98 obtained when using only 3 perturbations to 0.99 when using 10 perturbations, an AUC value which is in line with that obtained by the fRVM. On application to the test data, the AUC drops to 0.94, as found previously when using the 3 perturbations EA RVM. On the Titanic dataset, the AUC values remain consistent with that found previously by the 3 perturbations EA RVM, obtaining training and testing AUC values of 0.74 and 0.70 respectively and outperforming the AUC obtained by the fRVM.

Another possible explanation for the 3 perturbation EA RVM not finding the fRVM solutions is that not enough diversity is present in the elite archive. With this in mind experiments were also carried out using the set-based NSGA-II MOEA [Deb et al., 2000]. For an overview of the algorithm see Chapter 2.4.2 and for more in-depth technical details see [Deb et al., 2000; Deb, 2001].

Two variants of the NSGA-II algorithm were implemented: one using only mutation and one which used both 5-point cross-over and mutation. In both cases the mutation scheme used was identical to that used in the 3 point perturbation EA RVM. For his 3-objective DTLZ experiments Deb et al. [2002] used a population size of 100 and, for the modified 3-objective experiments in [Deb et al., 2006], a population size of 200. A population size of 300 was therefore deemed sufficient and the NSGA-II implementations run for 2000 generations. In both NSGA-II variants the populations were initialised in a similar manner to the EA RVM initialisation, generating a different $\boldsymbol{\alpha}$ for each in which a randomly chosen $\alpha_m = \alpha_{min}$ and all other $\alpha_n = \infty$ where $n \neq m$. Results for these two implementations can be found in tables 3.4 and 3.5.

The mutation only NSGA-II implementation successfully located solutions with training accuracy better than that of the fRVM on both the Banana and Titanic datasets. On the Banana dataset, the selected solutions' complexity in terms of used relevance vectors was greater than that of those selected from the 3 mutations EA RVM and suffered from slightly worse test performance. On the Titanic dataset, however, the selected solutions from the mutation only NSGA-II implementation were slightly sparser than those found by the 3 perturbation EA RVM and obtained slightly better test rates - although these are indistinguishable at the 5% significance level using the Mann-Whitney-Wilcoxon test. In terms of AUC, performance was the same as that obtained by the EA RVM. Results for the NSGA-II implementation using both mutation and crossover were similar to those obtained when only using mutation, albeit with slightly worse training performance.

Finally, experiments were also run using the basic 3 mutations EARVM initialised using the set of αs located by the fRVM in order to ensure it was capable of moving away from this initial overly-complex solution. Note that this is unlike the population initialisation used for the investigations with the other MOEAs presented here whose initial members are generated *at random*. On both the Banana and Titanic datasets training accuracies

	Banana		
Classifier	Train (%)	Test (%)	RVs
fRVM	94.05 (1.09)	**88.59 (0.75)**	29.70 (4.14)
EARVM: 3 Mutations	93.28 (1.27)	87.83 (0.87)	25.90 (5.57)
EARVM: 10 Mutations	97.43 (1.42)	86.43 (1.12)	97.40 (22.83)
NSGA-II: No Crossover	94.98 (1.27)	87.46 (0.80)	33.30 (10.57)
NSGA-II: Crossover	95.43 (1.40)	87.50 (1.14)	38.00 (8.18)
EARVM: 3 Mut., fRVM init.	95.55 (1.12)	88.23 (0.85)	29.50 (5.72)
	Titanic		
Classifier	Train (%)	Test (%)	RVs
fRVM	79.80 (2.95)	77.71 (1.38)	69.90 (22.12)
EARVM: 3 Mutations	80.00 (2.85)	77.54 (1.37)	7.90 (3.00)
EARVM: 10 Mutations	80.00 (2.85)	77.65 (1.70)	5.20 (1.69)
NSGA-II: No Crossover	80.00 (2.85)	**77.96 (0.78)**	3.80 (1.03)
NSGA-II: Crossover	80.00 (2.85)	77.56 (1.69)	3.50 (0.97)
EARVM: 3 Mut., fRVM init.	80.00 (2.85)	77.77 (1.18)	4.30 (2.16)

Table 3.4.: Average training and test accuracies and number of relevance vectors used by the solutions with highest training data accuracy on the Banana and Titanic datasets as located by the fRVM, EARVM using varying numbers of perturbations and NSGA-II variants. Standard deviations over 10 folds are given in brackets.

	Banana		Titanic	
Classifier	Train	Test	Train	Test
fRVM	0.99	0.95	0.68	0.64
EARVM: 3 Mutations	0.98	0.94	0.74	**0.70**
EARVM: 10 Mutations	0.99	0.94	0.74	**0.70**
NSGA-II: No Crossover	0.99	0.94	0.74	**0.70**
NSGA-II: With Crossover	0.95	0.93	0.73	**0.70**
EARVM: 3 Mut., fRVM init.	0.99	0.94	0.74	**0.70**

Table 3.5.: Average area under the curve for the training and test fronts located by the fRVM, EARVM with varying number of mutations, and NSGA-II variants.

greater than those of the fRVM are successfully obtained, with the added benefit of sparser solutions - most notably on the Titanic dataset. On the Banana dataset this initialisation leads to training AUC equal to that of the fRVM, though test AUC is slightly worse but still as good as that found by the other MOEAs. On the Titanic dataset the EA RVM initialised with the fRVM solution gives AUC performance equal to that of the randomly initialised EA RVM, outperforming the fRVM.

As seen here, the EARVM algorithm is capable of overfitting to the training data using only 3 possible mutations per iteration and, thus, in the rest of the experiments carried out in this thesis only 3 mutations are used. Should one be concerned that equivalent solutions to the fRVM are not being located then it is recommended that either the number of perturbations carried out by the EARVM is increased or initialisation is carried out using an fRVM solution since the EARVM can successfully move away from it to locate solutions with equal or better performance and sparsity. While the population based NSGA-II implementations yielded equivalent performance to that obtained by the EARVM implementations, it remains unclear how best to set the archive size without leading to potential shrinking and oscillation/retreating of the estimated Pareto front [Fieldsend et al., 2003]. Therefore, it is preferable to use the EARVM algorithm, or one of its variants, as they have unconstrained archive sizes.

3.8. Conclusion

A multi-objective evolutionary algorithm for training relevance vector machines for classification tasks which allows location of solutions over the full range of true and false positive rates has been presented. When working with multi-resolution kernels, such as wavelets, the RVM tends to overfit to the training data and thus generalise poorly. By including a measure of model complexity as an objective for minimisation a degree of regularisation has been provided by filtering out complex solutions that do not have superior true and false positive rates to those already in the elite archive.

A number of cross-validation schemes intended to prevent overfitting have been presented which, in conjunction with the evolutionary algorithm, locate solutions with far fewer relevance vectors than an RVM trained using the fRVM methodology, but which have comparable accuracies. Of the schemes presented, the K-fold cross validation methods have been most successful at achieving good generalisation, whilst the two-archive methods examined tended to over-regularise by removing too many basis functions. Preventing overfitting is a common problem when performing evolutionary optimisation, where it is important not to overfit [e.g. Reckhouse et al., 2010], and it is anticipated that the K-fold method presented will be useful for controlling overfitting in general evolutionary optimisation.

The stochastic search inherent in evolutionary methods helps in the location of the global optimum, whereas greedy methods such as the fRVM may become stuck in local optima. Entire Pareto fronts of solutions are found (e.g figure 3.3) giving the set of optimal trade-offs between true and false positive rates and complexity from which a solution fitting a particular application may be chosen.

3.9. Future Work

As a result of the MOEA training process we are effectively presented with ROC curves optimised over varying RVM complexities. It is unclear, however, how to identify an appropriate complexity from which to select a solution and so automatic selection of an appropriate complexity remains a problem. One possible solution would be to carry out manual inspection of the front to identify "knees" in the complexity from which to select. The user of the system is then able to select their preferred choice of trade-off between true and false positive rates and complexity from those available. Unfortunately, this may be time consuming or impractical with larger numbers of optimised objectives, requiring alternative visualisation methods e.g. [Walker et al., 2010]. While end user selection of a preferred solution is advantageous, ideally a particular complexity, or range of complexities would be selected, and the Pareto fronts flattened down to the TP-FP plane. Thus, it would be preferable to have some more principled automatic method for identifying an appropriate complexity, or range of complexities, automatically. Such a method might also prove useful in determining whether an RVM trained with the standard maximum likelihood method had overfit. A potential starting point for this could involve the use of normalised performance gain [Jin et al., 2006] which has previously been used to identify "knees" in two-dimensional Pareto fronts of neural networks located for complexity vs. mean squared error.

The smoothness of the MOEA generated fronts and apparent performance of the solutions within is dependent on the finite amount of underlying data. Prediction uncertainty is introduced as a result of the finite nature of the training data and uncertainty in the chosen model parametrisation. Additionally, noise in the data itself may result in solutions being added to the elite archive that would otherwise be dominated and, conversely, solutions that would be dominated by the archive being added to it when they shouldn't. Thus, two different but statistically equivalent equal sized samples could lead to different ROC performance. Ideally, one would like to incorporate this uncertainty into the generation of Pareto sets of classifiers.

Further work might centre around using measures such as dominance probability [Fieldsend and Everson, 2005, 2008b] when deciding whether or not to add a solution to the elite archive. This is similar to the K-fold cross validation scheme presented in section 3.5, in that solutions only enter the elite archive if it is very likely they dominate others. Underlying [Fieldsend and Everson, 2008b] is a sampling procedure used to produce estimates for the probability of dominance by comparing solution dominance over a number of samples. Combination of the sampling method and the K-fold cross validation scheme presented here could form the basis of future work.

4. Gene Selection from Classification

4.1. Introduction

A common task in bioinformatics is to identify features, in the form of gene expressions, with which to differentiate between instances of particular genetic conditions compared to some baseline. If selection of a particular gene leads to good classifier performance, it is therefore likely to be linked to the genetic causes of a condition. Typically, blood samples are passed over a Micro-Array chip on which there are a large number of genetic probes. Each individual probe on the chip is associated with particular gene and returns a value for the level at which that gene is expressed in the sample. These values, the "gene expressions", are used in the classification process.

Once a gene has been identified as linked to the genetic cause of a condition it may then be used in screening processes or targeted as part of treatment. Typically, samples are taken from a relatively small number of people compared to the very high number of gene expressions in the sample; often only tens of samples will be available but each may well consist of thousands of gene expression values e.g. [Hedenfalk et al., 2001]. Since the dimensionality of the data, M, is so high and the number of examples, N, so low i.e. $M \gg N$ it is often possible to use a large number of different hyperplanes to separate the two classes, those samples from the condition concerned and those from the baseline group, and still obtain good training performance. The InChianti dataset [Ferrucci et al., 2000] used in the experiments in section 4.7 is a particularly extreme example of this with only 698 samples in 16571-dimensions[1]. When it comes to testing, however, very few of these classification hyperplanes will yield good generalisation performance.

Given the paucity of the data, any noise in the samples may have a significant impact on the learnt models. Such noise may arise from extraction methods; precision of the equipment used to process genetic samples, as well as whether gene expressions are obtained either *in vitro* or *in vivo*, affecting the obtained gene expression levels. Additionally, noise may present in the form of mislabelled samples e.g. [Golub et al., 1999] and InChianti as used in 4.7 for Extreme Cognitive Divergence prediction. This may lead to overly flexible models modelling the noise and selecting more genes for classification than needed or potentially selecting noisy genes over important ones due to the rarity of samples. Since so few examples are available, any mislabelling may have a major impact on the genes selected.

As a result of the potential impact of noise along with the invasiveness and cost (both financial and human) of misclassification, it is therefore not only important to identify genes giving reliable classification performance but also to avoid any overfitting, identifying

[1]The InChianti dataset was provided by David Melzer of the Peninsula College of Medicine and Dentistry, University of Exeter. The work in 4.7 benefited from the insight of David Melzer, Luke Pilling and Lorna Harries.

only those genes which are relevant to the condition being classified.

By making use of Receiver Operating Characteristic curves one is able to assess classifier performance over a range of misclassification costs [Hanley and McNeil, 1982; Provost and Fawcett, 1997] and select the operating point giving the most appropriate trade-offs for the task at hand with knowledge of the range of options available. In gene selection problems misclassification costs are not always known beforehand – for instance they may relate to the particular genes that have to be targeted by treatment and thus vary on a classifier by classifier basis. Therefore rather than focusing solely on classification accuracy, one seeks classifiers that give the best possible class discrimination over the full range of misclassification costs. The area under the ROC curve (AUC) is a measure of how well a classifier is able to discriminate between two classes [Hanley and McNeil, 1982; Hand and Till, 2001] and therefore it is desirable to optimise classifier AUC performance during training.

In this chapter a Multi-objective Evolutionary Algorithm (MOEA) is presented which trains RVM classifiers by simultaneously maximising AUC whilst minimising model complexity. Pareto fronts of solutions are generated containing the best trade-offs between complexity and class discrimination ability, as measured by AUC. Using AUC and complexity as the objectives promotes models giving the best possible class separation at a particular complexity and reduces overfitting by excluding models that give no improvement in AUC for increased complexity. Thus, those genes key for differentiating between different conditions should be identified whilst other, less important ones playing an insignificant role in the classification, are pruned from solutions.

To ensure robust gene selections, the repeated partitioning scheme proposed by Li et al. [2002] was used. In this, the data is randomly split into two equal sized halves and gene selections recorded from classifiers trained separately on each half. The partitioning and recording process is then repeated some suitable number of times. Examining these gene selections, counts can be produced for how many times a particular gene is selected. If a gene selection frequently occurs in both halves of a partition it is likely to be important for classification.

Using the partitioning scheme the fast Relevance Vector Machine (fRVM) [Tipping and Faul, 2003; Faul and Tipping, 2002] and the Multi-Objective Evolutionary Algorithm were used to carry out gene selection for classification of a number of well studied bioinformatics cancer classification datasets and gene selection counts recorded. The MOEA's ability to select genes important for distinguishing between different genetic conditions is demonstrated, albeit with lower gene selection counts than the fRVM. An investigation carried out using the Leukaemia dataset [Golub et al., 1999] indicated this to be due to the paucity of the dataset and that the fRVM appears to overfit during training.

Experiments were also carried out on the newer bioinformatics dataset arising from the InChianti population study [Ferrucci et al., 2000] for a number of current research questions [Harries et al., 2011a,b]. As a baseline experiment the MOEA and fRVM were used to classify sample gender, both successfully identifying genes lying on the X and Y chromosomes. Following this experiments were carried out to identify genes associated with ageing, an important factor in many conditions. Finally, genes associated with extreme cognitive decline, as measured by Mini-Mental State Exam [MMSE Folstein et al., 1975]

scores, were sought.

This chapter begins with an overview of the existing methods for gene identification in section 4.2. Following that details of the MOEA are given in section 4.3 and the data partitioning routine in section 4.4. Results on the well-studied bioinformatics databases are given in section 4.5 with a more in-depth analysis of the counts carried out using the Leukaemia dataset in section 4.6. Experimental results on the newer InChianti dataset are given in section 4.7 with some further analysis in section 4.8. Conclusions and future work are given in sections 4.10 and 4.11 respectively.

4.2. Background

Perhaps the most commonly used method in bioinformatics for examining gene association with genetic condition is regression screens. Typically, linear regression models are fitted to each of the individual gene probes, [e.g. Harries et al., 2011a], along with any confounding cofactors, such as age, height, smoking, etc., regressing against the target condition. Using the regression model predictions and the true targets, hypothesis testing is carried out to obtain p-values for the probability the two distributions were generated by different models.

Since comparisons are being carried out separately for each of the gene probes, the Bonferroni correction [e.g. Rice, 1988] is then used to reduce the probability of obtaining false positives[2] giving adjusted p-values. Genes with p-values lower than some threshold are then determined to be significant for prediction. The main drawback of the regression screen method is that it only works with a single gene probe at a time. As each gene is regressed independently of the others, interactions between groups of genes are not considered and thus any combinations leading to a condition may be missed.

Using a partitioning scheme, discussed later in section 4.4, Li et al. [2002] use RVM classifiers to select sparse weightings of gene expressions on disjoint halves of bioinformatics datasets and compare the resulting selections over a number of different partitionings. Given the high dimensionality of the datasets it is unlikely that the same genes would be selected at random by two classifiers trained on disjoint halves of the datasets. Thus, genes repeatedly selected over a number of different partitions are likely to be significant for classification. Since all the genes are considered by the RVM at the same time, any gene interactions important for classification that would otherwise be overlooked by individual analyses can be identified. By using this repeated partitioning procedure the potential effects of overfitting to a particular training dataset are ameliorated in the counting process.

In [Bae and Mallick, 2004] a two-level hierarchical Bayesian regression model is used in which priors over model weights are used to promote sparse models in terms of the numbers of genes selected. Three different prior structures are investigated, promoting differing levels of model sparseness: a Gaussian prior over the model weights with an independent prior distribution over the weight variances; a sparsity promoting Laplacian prior equivalent to the Lasso regularisation penalty [Williams, 1995]; the non-informative Jeffreys prior recommended in [Figueiredo, 2001, 2003] is used. Unlike the other two priors the Jeffreys prior requires no setting of hyper-parameters, enabling automatic variable selection and strongly inducing model sparseness.

[2]It should be noted that this comes at the cost of increased false negatives

The classification model used is defined as:

$$y(\mathbf{x}) = p(t = 1|\mathbf{x}) = \phi(\mathbf{x}^T\mathbf{w}) \tag{4.1}$$

where $\mathbf{w}$ is a vector of weights and ϕ is the Normal cumulative density function used to link the probability of membership of a sample, $\mathbf{x}$, to class 1, $y(\mathbf{x})$, with the linear weighting, $\mathbf{x}^T\mathbf{w}$.

In the models proposed by Bae and Mallick [2004] a two-level hierarchical Bayesian model is used in which different priors are used for $\mathbf{w}$. A zero-mean Gaussian prior is placed over $\mathbf{w}$ with unknown variance. Priors are then assigned to the variances, assuming they are independent. The prior distribution for $\mathbf{w}$ is thus:

$$p(\mathbf{w}|\boldsymbol{\alpha}^{-1}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \boldsymbol{\alpha}^{-1}) \tag{4.2}$$

where $\mathbf{0} = (0, \ldots, 0)^T$, $\boldsymbol{\alpha}^{-1} = \text{diag}(\alpha_1^{-1}, \ldots, \alpha_M^{-1})$ and α_m^{-1} is the variance of w_m. Three different choices of prior distribution for $\boldsymbol{\alpha}^{-1}$ are then considered, resulting in differing levels of sparsity in the numbers of genes selected.

In the first model, a Student-t prior is placed on $\mathbf{w}$ from a scale mixture of zero mean Gaussians with Inverse Gamma distributed $\boldsymbol{\alpha}^{-1}$. An Inverse Gamma prior is applied to each α_m^{-1} in $\boldsymbol{\alpha}^{-1}$ as:

$$p(\alpha_m^{-1}|a, b) = IG\left(\alpha_m^{-1}|\frac{a}{2}, \frac{b}{2}\right) = (\alpha_m^{-1})^{-a/2-1}\exp(-b/2\alpha_m^{-1}) \tag{4.3}$$

where the values of a and b are typically set so as to give a large variance over α_m^{-1}. This prior structure is equivalent to the ARD model used in the RVM [Tipping, 2001, 2000], in which a Gamma prior is placed over $\boldsymbol{\alpha}$. Assuming independence between individual α_m^{-1}, the conditional prior distribution for $\boldsymbol{\alpha}^{-1}$ is:

$$p(\boldsymbol{\alpha}^{-1}|a, b) = \prod_{m=1}^{M} IG\left(\alpha_m^{-1} \mid \frac{a}{2}, \frac{2}{b}\right). \tag{4.4}$$

This results in a Student-t model for $\mathbf{w}$ (see Chapter 2), as $p(\mathbf{w}|a, b)$ is a scale mixture of Gaussians:

$$p(w_m|a, b) = \int_0^\infty \mathcal{N}(w_m|0, \alpha_m^{-1})IG(\alpha_m^{-1}|a, b)d\alpha_m^{-1} = \textit{Student-t}\,(w_m|a, b)\,. \tag{4.5}$$

In the second model, instead of placing an Inverse Gamma prior on α_m^{-1} an exponential prior is used instead, resulting in a Laplace prior on $\mathbf{w}$ in which sparseness is promoted by setting "irrelevant" w_m to zero. Since a Laplacian prior distribution can be expressed as a scale mixture of Gasussian priors with independent expontentially distributed variance, it is thus equivalent to a two-level hierarchical Bayesian model:

$$p(w_m|\gamma) = \int_0^\infty \mathcal{N}(w_m|0, \alpha_m^{-1})expon(\alpha_m^{-1}|\gamma)d\alpha_m^{-1} = \textit{Laplace}\left(w_m|0, \frac{1}{\sqrt{\gamma}}\right) \tag{4.6}$$

where the exponential distribution used as the prior distribution for α_m^{-1} is:

$$p(\alpha_m^{-1}|\gamma) = expon\left(\alpha_m^{-1}|\gamma\right) = \frac{\gamma}{2}\exp\left(-\frac{\gamma\alpha_m^{-1}}{2}\right). \tag{4.7}$$

resulting in a model which is equivalent to using a Laplace prior for w. As with the previous prior structure the hyper-parameter, in this case γ, is set so the variance of α^{-1} is high. This is related to the Lasso model [Williams, 1995] but whereas the Lasso method only works with a single α^{-1}, this model has the added flexibility of working with multiple α^{-1} values, allowing some to be active and others to be "switched off".

In the third and final model considered by Bae and Mallick [2004], the problem of fixing distribution hyper-parameters is circumvented through the use of a non-informative Jeffreys prior over $\boldsymbol{\alpha}^{-1}$, a model proposed in [Figueiredo, 2003]:

$$p(\boldsymbol{\alpha}^{-1}) = |\boldsymbol{\alpha}^{-1}|^{1/2} = \prod_{m=1}^{M}\frac{1}{\alpha_m^{-1}} \tag{4.8}$$

As shown in [Figueiredo, 2001] and demonstrated in [Figueiredo, 2003; Bae and Mallick, 2004] using this prior for α_m^{-1} strongly induces sparseness and results in good performance.

While the RVM uses type II maximum likelihood to obtain model parametrisations, Bae and Mallick [2004] make use of MCMC Gibbs sampling to sample parameters from the posterior distribution. As with the RVM method, these models are able to consider multiple genes at once, and thus able to consider gene interactions in the selection process, unlike regression screens. However, the main drawback of this model in comparison with the RVM is the potentially long training times required for the MCMC sampling process. All three of the prior structures used are shown to give good model performance and of them, it is the Jeffreys prior that is recommended since it does not require setting of hyper-parameters and strongly induces sparseness while giving good performance.

In the next section we present a Multi-Objective Evolutionary Algorithm to learn sparse models for classification in which we simultaneously optimise RVM classifier complexity and class discrimination ability.

4.3. Multi-Objective Evolutionary Algorithm

Given the paucity of the data, overfitting and unreliable gene selections are a concern. Not only do we desire gene selections that lead to models that generalise well we *only* want to select those genes related to discrimination between genetic conditions – it could be expensive both in terms of time and money to carry out the follow-up biological investigations. It is therefore proposed to use a Multi-Objective Evolutionary Algorithm to penalise model complexity whilst promoting class discrimination ability.

The Multi-Objective Evolutionary Algorithm (MOEA) presented in this chapter seeks to locate the Pareto set of Relevance Vector Machine classifiers by simultaneously maximising the area under the ROC curve, $AUC(\boldsymbol{\alpha})$, and minimising model complexity, $C(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is the vector of RVM hyper-parameters used to control the magnitude of the model weights. For an explanation of RVM classifiers and the complexity measure, $C(\boldsymbol{\alpha})$, see Chapter 3. Making use of the dominates relation, the MOEA compares solutions in terms

Algorithm 5 Multi-Objective Evolutionary Algorithm optimising AUC and complexity

Require: $\mathcal{D}_{\text{tr}}$ *Data feature-target pairs*
$\boldsymbol{\alpha}$ *Initial solution*

1: $E := \texttt{initialise}(\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha})$ *Initialise the archive*
2: `while not converged`
3: $\boldsymbol{\alpha} := \texttt{select}(E)$ *Select $\boldsymbol{\alpha}$ from the set of $\boldsymbol{\alpha}$ in E*
4: $\boldsymbol{\alpha}' := \texttt{perturb}(\boldsymbol{\alpha})$ *Perturb the $\boldsymbol{\alpha}$ values*
5: $AUC(\boldsymbol{\alpha}'), C(\boldsymbol{\alpha}') := \texttt{evaluate}(\mathcal{D}_{\text{tr}}, \boldsymbol{\alpha}')$
6: $E := \texttt{nondom}(E \cup \{\boldsymbol{\alpha}'\})$ *Update archive E*
7: `end`

of AUC and C, maintaining an elite archive, E, of mutually non-dominating solutions which approximate the Pareto set. At each iteration of the optimisation process an $\boldsymbol{\alpha}$ is selected from those in E and perturbed to create a new solution, $\boldsymbol{\alpha}'$. If $\boldsymbol{\alpha}'$ is non-dominated by the members of E it is added to it and any newly dominated solutions in E removed. Thus, the elite archive E can only approach the Pareto front for the training data $\mathcal{D}_{\text{tr}}$. The algorithm is outlined in algorithm 5.

For numerical convenience the values of the α_m in $\boldsymbol{\alpha}$ are constrained between $\alpha_{\min} = 10^{-12}$ and $\alpha_{\max} = 10^{12}$ with any newly perturbed $\alpha_m < \alpha_{\min}$ being set to $\alpha_{\min}$ and any $\alpha_m > \alpha_{\max}$ being set to ∞. Note that the algorithm is not sensitive to the particular values of $\alpha_{\min}$ and $\alpha_{\max}$ used.

To begin with, the algorithm takes in a provided training dataset of feature-target pairs, $\mathcal{D}_{\text{tr}}$, and, if provided, an initial vector of $\boldsymbol{\alpha}$ values. The value of $\boldsymbol{\alpha}$ can either be specified by the user, perhaps using an $\boldsymbol{\alpha}$ learnt by the fRVM, or is otherwise initialised at random with a single α_m set to $\alpha_{\min}$ and all others set to ∞. In the experiments here a randomly selected α_m is set to $\alpha_{\min}$.

Using this initial $\boldsymbol{\alpha}$, the MOEA initialises the elite archive of solutions, E. The complexity $C(\boldsymbol{\alpha})$ is calculated and, along with the provided data, $\mathcal{D}_{\text{tr}}$, $\boldsymbol{\alpha}$ is used to train classifier weights using iterative reweighted least squares (IRLS) and the AUC, $AUC(\boldsymbol{\alpha})$, of the model obtained (line 1).

The algorithm proceeds by selecting an $\boldsymbol{\alpha}$ at random from the archive, E, and perturbing between 1 and 3 of its components to form a new parametrisation, $\boldsymbol{\alpha}'$ (lines 3 - 4). As with the algorithm in Chapter 3, these perturbations are carried out in one of three different ways, selected with equal probability: (i) an α_m corresponding to an active basis function is selected at random and perturbed as: $\log_{10} \alpha'_m = \log_{10} \alpha_m + \epsilon$ where ϵ is drawn from a heavy-tailed Laplacian distribution, $p(\epsilon) \propto e^{-|\epsilon|/2}$, to encourage exploration [Yao et al., 1999]; (ii) an active basis function is selected at random and "switched off" by setting the associated α_m to ∞; (iii) an inactive basis function is selected at random and "switched on" by setting α_m to a number uniformly drawn at random in logarithmic space between $\alpha_{\min}$ and $\alpha_{\max}$. This newly perturbed vector of α_m values, $\boldsymbol{\alpha}'$, is used to train weights using IRLS. The resulting model $AUC(\boldsymbol{\alpha}')$ for the ROC curve on $\mathcal{D}_{\text{tr}}$ is calculated, along with the model complexity measure $C(\boldsymbol{\alpha}')$ (line 5).

Having obtained $AUC(\boldsymbol{\alpha}')$ and $C(\boldsymbol{\alpha}')$ the solution is compared with those already in the elite archive, E. If $\boldsymbol{\alpha}'$ is non-dominated by the set of $\boldsymbol{\alpha}$ in E, it is added to E and any newly dominated members of E deleted (line 6). In this way, the archive E can

only approach the set of Pareto optimal solutions in terms of AUC and complexity. This process of perturbing and evaluating $\boldsymbol{\alpha}'$ for inclusion in E (lines 2 - 7) is repeated until either no additions have been made to the archive for 100 iterations or the number of iterations exceeds some specified maximum number.

In order to calculate the AUC the Mann-Whitney-Wilcoxon two sample test is used as outlined by Hand and Till [2001] and discussed in Chapter 2. Calculation of AUC using this method is straightforward, only requiring the model predictions and data targets to calculate, and, importantly, does not require setting of a decision threshold. Having obtained the AUC for a particular classifier we have a measure of how well the classifier is able to discriminate between the two classes [Hand and Till, 2001; Hanley and McNeil, 1982]. Since the aim in these Bioinformatics problems is the identification of genes to discriminate between particular genetic conditions, AUC provides a suitable measure with which to compare model performance.

4.4. Splitting Procedure

Given the typically high dimensionality of gene sample data and relatively few available training examples, feature selection becomes unreliable. As data is so high dimensional there will be numerous combinations of features that could be selected to give good classification performance on the training data. Few, however, will generalise well to new data. The task therefore is to successfully identify *only* the set of features that are actually associated with the classification problem.

As the dimensionality of the datasets is so high, so the probability of selecting an individual feature at random is low. Therefore those genes which are repeatedly selected by classifiers over numerous different training runs should have a high chance of being significant for the desired classification. This is the idea behind the procedure in [Li et al., 2002], which is outlined here.

In order to identify genes associated with particular genetic conditions Li et al. [2002] make use of a partitioning scheme. Training data is randomly partitioned into two disjoint sets of equal size and the classifier trained independently on each set. The selected features are then analysed in order to identify those which are common, or coincident, to both halves of the partition. This process is repeated some number of times, generating new partitions with which to train models and the resulting gene selections recorded.

Using the gene selections produced by this procedure one can identify potentially useful genes. As a result of the repartitioning, genes that give good classification should be selected over a number of different partitions whereas those which may only give good performance on a particular partition will appear less often.

If m_1 is the number of features selected by classifier 1, m_2 the number of features selected by classifier 2 and m the maximum number of features possible to select, then the conditional probability of having m_c features in common by drawing m_1 and m_2 features from m uniformly at random is:

$$p(M_c = m_c | m, m_1, m_2) = \frac{\binom{\max(m_1,m_2)}{m_c}\binom{m-\max(m_1,m_2)}{\min(m_1,m_2)-m_c}}{\binom{m}{\min(m_1,m_2)}}. \tag{4.9}$$

Thus, for large numbers of features, m, and small numbers of features selected by the trained models, m_1 and m_2, then the probability of $M_c = 1$ or $M_c = 2$ is quite small.

While high counts of the total number of times a gene is selected will give a good indication of that gene being important for the classification, this method is more stringent, only counting gene selections where they are coincident to models trained on both halves of a partition. From equation 4.9 we know that small numbers of gene selections coincident to two models trained on disjoint datasets are unlikely by random selection. The chances of these coincident selections recurring over several partitions at random are also low. Therefore, genes with large total coincident selection counts can be considered highly likely to be those useful for carrying out the desired classification.

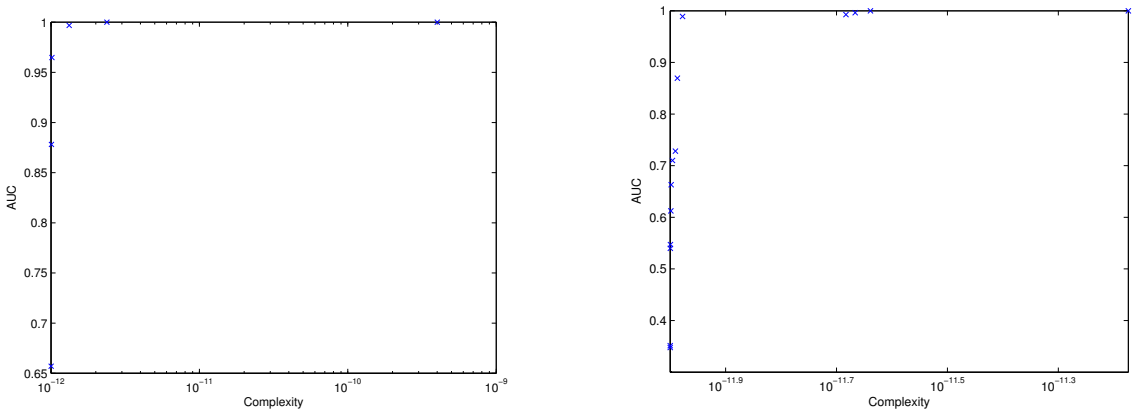
Given the paucity of the observations in most bioinformatics datasets, any splits resulting in non-representative class proportions could have a significant impact on model learning. Therefore a minor amendment is made to the splitting procedure implementation in the experiments carried out here. Whereas Li et al. [2002] split the whole dataset in half at random, here the partitioning is carried out on a class-wise basis i.e. each class is split in half between the two sets. By performing this "stratified partitioning", the potential for creating splits where class proportions are markedly different from that of the overall training dataset is avoided. This will go some way to addressing the situation where the noise in a small number of samples used to represent a class in a training split is falsely learnt as important for classification.

4.5. Experimental Results

When evaluating a new classifier, one would normally utilise a synthetically generated dataset to analyse classifier performance and feature selections. Since the dataset would have been constructed from a pre-determined distribution it would be known whether or not classifier feature selections, in this case genes, were correct. Unfortunately no suitable dataset is available and it is unclear how to realistically construct one that has the properties of a true bioinformatics dataset. Therefore, in order to investigate the efficacy of the MOEA for carrying out gene selection, the fRVM and MOEA were trained on a number of commonly used bioinformatics datasets using the splitting procedure outlined in section 4.4. Since these are well studied datasets, gene selections can be compared with those identified in the existing literature. As discussed, given the paucity of the datasets, partitioning was carried out on a class-wise basis.

Classifiers were trained on both halves of each partitioning and their selections recorded for analysis both in terms of overall counts of gene selections and by the number of times a gene selection was coincident to both halves of a partition. While this is simple to do for the fRVM, some thought is needed for counting gene selections in the fronts of solutions generated by the MOEA.

Each front generated by the MOEA will contain multiple classifiers of varying AUC and complexity. It is quite possible the useful genes are present in several classifiers in the front, making a full contribution to the classification. However, it is also possible that other less useful genes are also present in several classifiers but only making a minor contribution to the classification. Furthermore, different MOEA generated fronts will likely contain



(a) (b)

Figure 4.1.: MOEA located solutions for two different splits of the Leukemia data. AUC and log-complexity of the solutions are plotted on the vertical and horizontal axes respectively.

different numbers of classifiers. A straightforward counting of gene selections for every classifier located by the MOEA might therefore obfuscate important genes in favour of more prolific less important genes that only make minor contributions to the classification. Additionally if one counted selections from each classifier in the front, it would lead to bias towards the gene selections in larger fronts giving a particular split more importance in the selection counting over the others. Therefore, when recording gene selections for the MOEA fronts generate in the experiments here, it was decided to give a gene a selection count of 1 if it appeared anywhere in the front i.e. while a gene may be included in multiple solutions in a front, it will only count once towards the recorded selections. This avoids selection count bias towards larger fronts and means that prolific uninformative genes making only a minor contribution to classifications will only be counted once.

In figures 4.1a and 4.1b two different fronts of solutions generated by running the MOEA on two different splits of the Leukemia dataset are plotted. On the first split, figure 4.1a, the MOEA located 6 different solutions with AUC values ranging from 0.66 to 1. The maximum number of genes selected by any one solution in the front was 3 and the minimum selected was 1. Six different genes were selected across the whole front of solutions. On the second split, figure 4.1b, the MOEA located 14 different solutions whose AUC values range from 0.35 to 1. The maximum number of genes selected by any one solution in this front was 3 and the minimum was 1. Across the whole front 15 different genes were selected. In both cases the MOEA has located solutions over a range of AUC and complexity values, but with different numbers of solutions. No gene selections were common to both fronts.

For each of the datasets investigated below, gene selections were analysed for the MOEA and fRVM trained classifiers. Total and coincident gene selection counts are given and the top hits compared with those found in the existing literature. Since the MOEA yields much lower gene selection counts than the fRVM, an investigation into the actual performance of these classifiers was carried out using the Leukaemia data [Golub et al., 1999] and is presented later in section 4.6.

4.5.1. Leukaemia Dataset

The Leukaemia dataset [Golub et al., 1999] is a well studied Bioinformatics dataset [e.g. Golub et al., 1999; Li et al., 2002; Bae and Mallick, 2004] containing 72 examples of 7129 dimensional gene expression data divided into a training set of 38 examples and test set of 34 examples. The goal is to distinguish between examples of Acute Myeloid Leukaemia (AML) and Acute Lymphoblastic Leukaemia (ALL).

As with [Li et al., 2002] the training and test data were combined to give a dataset of 25 AML examples and 47 ALL examples. Using the previously outlined splitting procedure, the fRVM and MOEA were trained for 100 splits and the gene selections recorded. Stem plots of the total and coincident selection counts are plotted for the fRVM and MOEA in figure 4.2 and the top hits recorded in tables 4.1 and 4.3.

The top hits found by the fRVM were Zyxin with 136 selections, TCF3 Transcription factor 3 with 93 selections, CCND3 Cyclin D3 with 75 selections and MacMarcks with 73 selections. Zyxin and MacMarcks are identified as leading genes for classification by [Bae and Mallick, 2004] and they, along with the other top fRVM selections feature in the list of strongly significant genes for classification of the Leukaemia data given by [Lee et al., 2003].

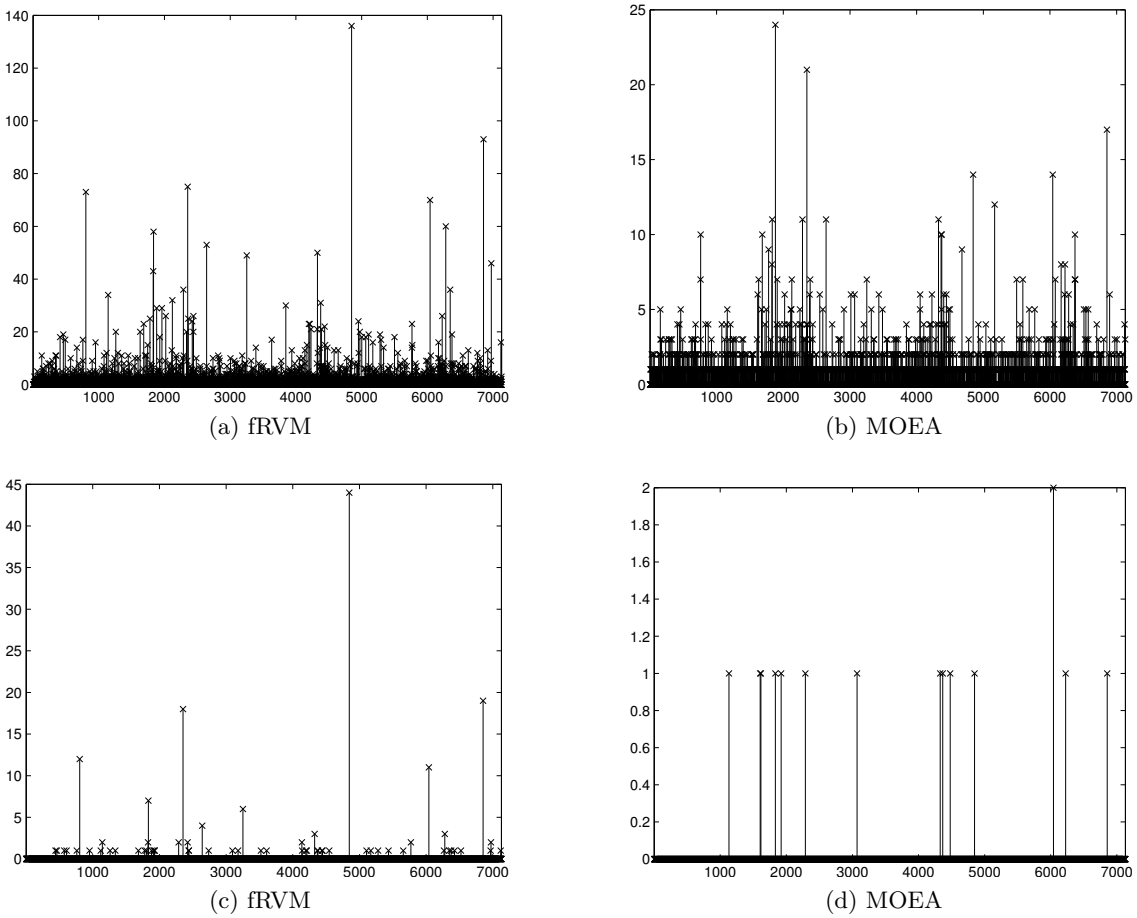
Analysing the coincident gene selections for the fRVM pairs trained on each split, Zyxin was selected in common 44 times, TCF3 Transcription factor 3 selected 19 times, CCND3 Cyclin D3 19 times, MacMarcks 12 times and APLP2 Amyloid beta (A4) precursor-like protein 2 selected 11 times. A longer list of these can be found in table 4.2.

The top hits found by the MOEA were CST3 Cystatin C with 24 selections, CCND3 Cyclin D3 with 21 selections, TCF3 Transcription factor 3 with 17 selections and Zyxin and APLP2 Amyloid beta (A4) precursor-like protein 2 with 14 selections. CST3 Cystatin C is the gene identified as most significant for classification by Lee et al. [2003], while Zyxin has previously been identified as an important gene [Lee et al., 2003; Bae and Mallick, 2004]. The other top hits also feature in the list of significant genes provided by Lee et al. [2003]. Counts of gene selections coincident to fronts trained on each half of the splits are given in table 4.4 with APLP2 Amyloid beta (A4) precursor-like protein occurring most times in common with a count of 2.

Both the MOEA and fRVM have successfully identified Zyxin as an important gene for classification of this dataset in their overall selection count top hits. Interestingly, while Zyxin is the top coincident hit for the fRVM it is not for the MOEA. The top MOEA coincident gene selection, APLP2 Amyloid beta (A4) precursor-like protein, appears in the top fRVM coincident gene selections in fifth place. Regardless, both the fRVM and MOEA are selecting genes already identified in the literature as important for classification. An analysis of the comparative performance of the learnt classifiers is presented in section 4.6.

4.5.2. Colon Cancer

In the Colon Cancer dataset [Notterman et al., 2001] the task is to distinguish between cancerous and "normal" tissue samples. Data was collected from 40 cancerous and 22 non-cancerous tissue samples each with 2000 features. As before, the fRVM and MOEA were trained over 100 splits and their gene selections recorded. Stem plots for gene selections



(a) fRVM (b) MOEA

(c) fRVM (d) MOEA

Figure 4.2.: Gene Selections for the Leukaemia dataset over 100 splits. Top row: all selections. Bottom row: selections coincident to both halves of each split.

Rank	**Count**	**Gene**
1	136	Zyxin
2	93	TCF3 Transcription factor 3
3	75	CCND3 Cyclin D3
4	73	Macmarcks
5	70	APLP2 Amyloid beta (A4) precursor-like protein 2
6	60	MYL1 Myosin light chain (alkali)
7	58	CD33 CD33 antigen (differentiation antigen)
8	53	MB-1 gene
9	50	PROTEASOME IOTA CHAIN

Table 4.1.: fRVM Gene Selections on Leukaemia Dataset.

Rank	Count	Gene
1	44	Zyxin
2	19	TCF3 Transcription factor 3
3	18	CCND3 Cyclin D3
4	12	Macmarcks
5	11	APLP2 Amyloid beta (A4) precursor-like protein 2
6	7	CD33 CD33 antigen (differentiation antigen)
7	6	GLUTATHIONE S-TRANSFERASE, MICROSOMAL
8	4	MB-1 gene
9	3	PROTEASOME IOTA CHAIN
		MYL1 Myosin light chain (alkali)

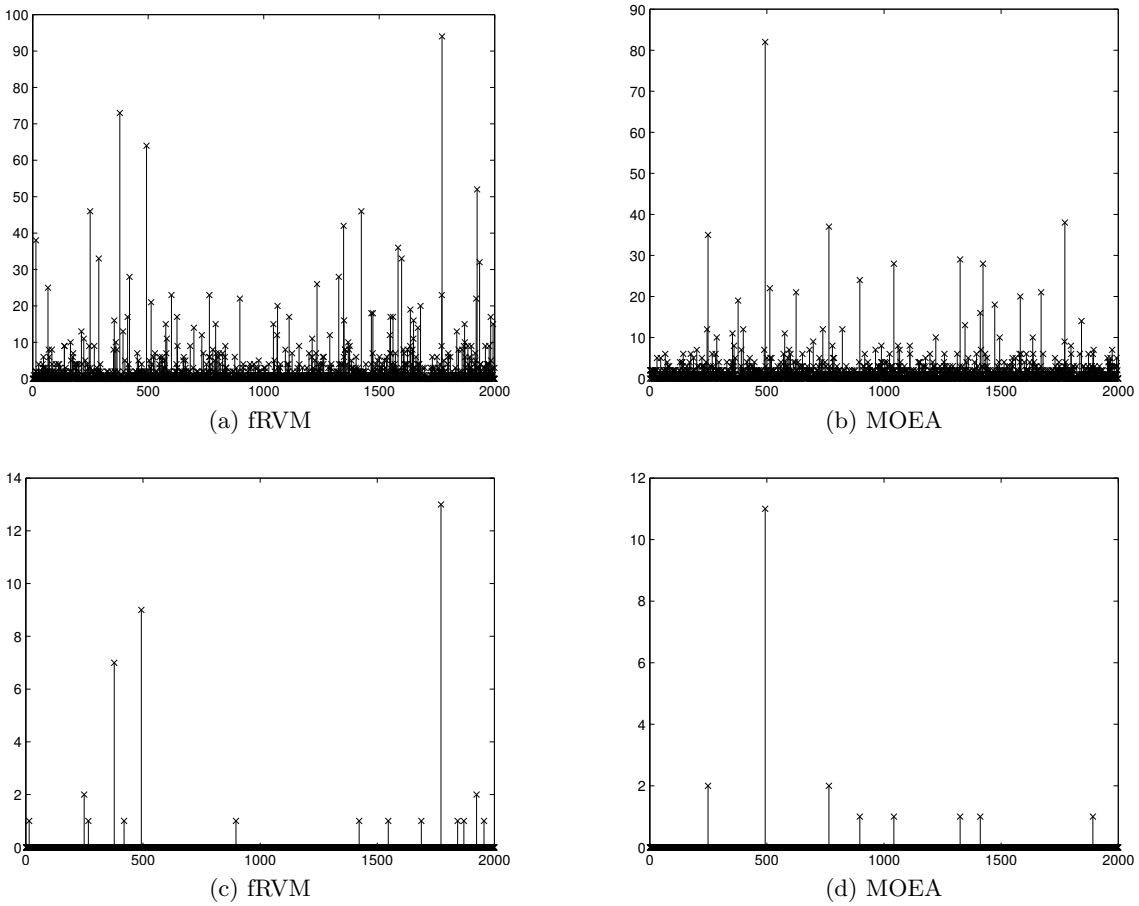
Table 4.2.: Top 10 fRVM Coincident Gene Selections on Leukaemia Dataset.

Rank	Count	Gene
1	24	CST3 Cystatin C (amyloid angiopathy and cerebral hemorrhage)
2	21	CCND3 Cyclin D3
3	17	TCF3 Transcription factor 3
4	14	Zyxin
		APLP2 Amyloid beta (A4) precursor-like protein 2
5	12	OBF-1 mRNA for octamer binding factor 1
6	11	CD33 CD33 antigen (differentiation antigen)
		DF D component of complement (adipsin)
		MB-1 gene
		PROTEASOME IOTA CHAIN

Table 4.3.: MOEA Gene Selections on Leukaemia Dataset.

Rank	Count	Gene
1	2	APLP2 Amyloid beta (A4) precursor-like protein
2	1	CATHEPSIN G PRECURSOR
		RB1 Retinoblastoma 1 (including osteosarcoma)
		Quiescin (Q6) mRNA, partial cds
		CD33 CD33 antigen (differentiation antigen)
		MYH9 Myosin, heavy polypeptide 9, non-muscle
		DF D component of complement (adipsin)
		Tax1-binding protein TXBP181 mRNA
		PROTEASOME IOTA CHAIN
		ARHG Ras homolog gene family, member G (rho G)
		ID3 Inhibitor of DNA binding 3
		Zyxin
		CD19 gene
		TCF3 Transcription factor 3

Table 4.4.: Top EA Front Coincident Gene Selections on Leukaemia Dataset.



(a) fRVM (b) MOEA

(c) fRVM (d) MOEA

Figure 4.3.: Gene Selections for the Colon cancer dataset over 100 splits. Top row: all selections. Bottom row: selections coincident to both halves of each split.

are given in figure 4.3 and the top hits tabulated in table 4.5.

As with the results presented in [Li et al., 2002], the top three genes identified by the fRVM were; gene 1772 (H08393) Collagen Aalpha 2(XI) chain (Homo sapiens), gene 377 (Z50753) mRNA for GCAP-II/uroguanylin precursor and gene 493 (R87126) Myosin Heavy Chain, nonmuscle (Gallus gallus). Gene 1772 (H08393) was also been found to be an important gene for classifying this data by Guyon et al. [2002]. Examination of gene selections coincident to the fRVM pairs trained on each split half showed gene 1772 was selected in common 13 times, gene 493 9 times and gene 377 7 times. The full list can be found in table 4.5c.

For the MOEA, genes 493 and 1772 are also identified as top hits with 83 and 38 selections respectively. Examination of selections coincident to fronts trained on each half of the splits reveal gene 493 to have been selected 11 times in common and genes 249 and 765 twice in common. Interestingly, gene 1772 was not found to be selected in common a significant number of times, unlike with the fRVM. See tables 4.5b and 4.5d for gene selection counts.

4.5.3. Hereditary Breast Cancer

The Hereditary breast cancer dataset [Hedenfalk et al., 2001] consists of tissue samples taken from patients carrying mutations in the genes BRCA1 or BRCA2 and from patients

Rank	Count	Gene ID
1	94	1772
2	73	377
3	64	493

(a) fRVM

Rank	Count	Gene ID
1	82	493
2	38	1772
3	37	765
4	35	249

(b) MOEA

Rank	Count	Gene ID
1	13	1772
2	9	493
3	7	377
4	2	249
		1924

(c) fRVM

Rank	Count	Gene ID
1	11	493
2	2	249
		765

(d) MOEA

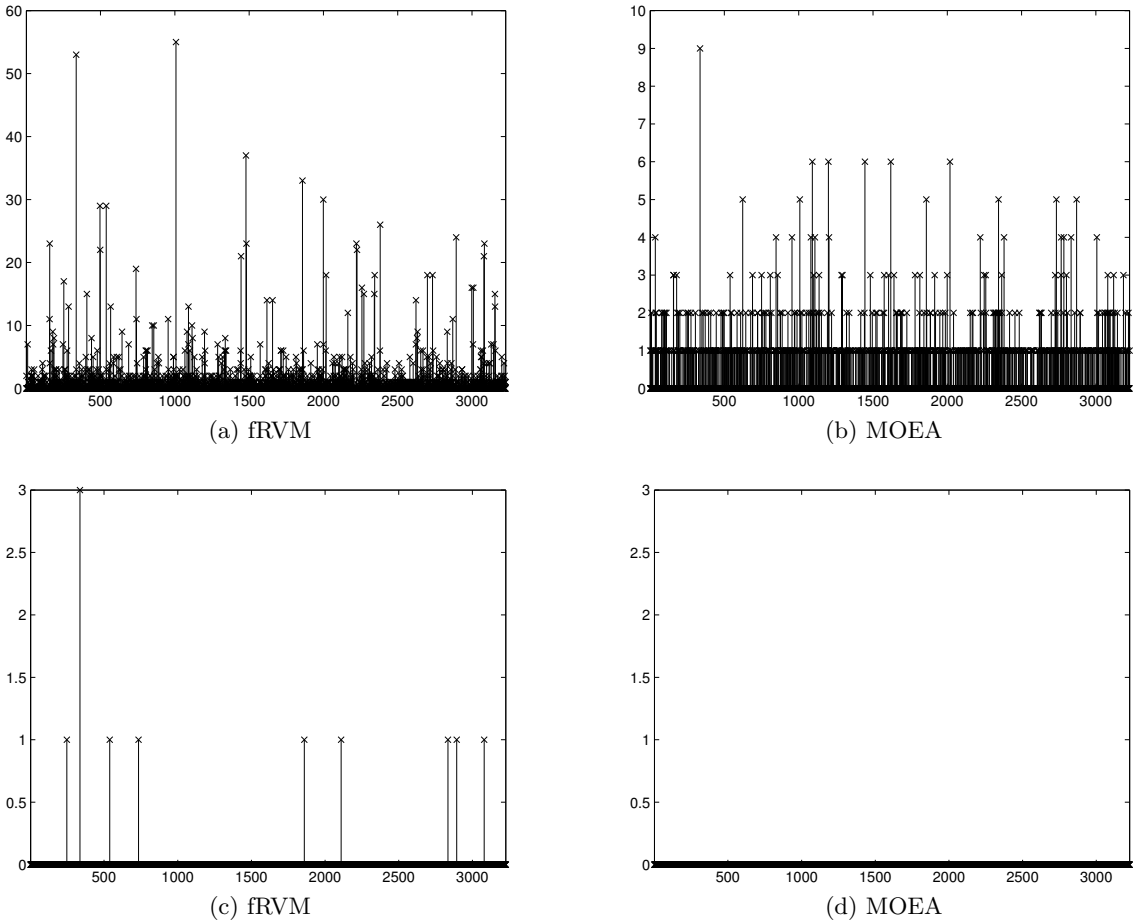
Table 4.5.: Top gene selections for the Colon Cancer dataset over 100 splits. Top row: total selections. Bottom row: selections coincident to both halves of each split

not expected not to have mutations in those predisposing genes. 22 samples were taken from 21 breast cancer patients of which 15 of the women concerned had hereditary cancer: 7 with tumors and BRCA1 and 8 with tumors and BRCA2. In each case the sample contained 3226 gene expressions. Following Bae and Mallick [2004] experiments were carried out casting the dataset as a 1 vs all two-class problem of distinguishing between the BRCA1 cases versus BRCA2 and sporadic (the non-hereditary group) grouped together. Gene selection counts are plotted in figure 4.4.

The top two gene selections found by the fRVM are keratin 8 with 55 selections and transducer of ERBB2, 1 (sometimes referred to as TOB1) with 53 selections, both of which feature as top hits in [Bae and Mallick, 2004; Lee et al., 2003]. Examination of the gene selections finds transducer of ERBB2, 1 to have been selected in common to both halves of a split 3 times. Selection counts for the fRVM can be found in tables 4.6 and 4.7.

The MOEA trained classifiers identify the transducer of ERBB2, 1 as the most important gene for classification, the gene identified as a close second most important by the fRVM. However, it does so with at total selection count of only 9. These low selection counts are further encountered when examining the number of coincident gene selections on each split, in which one can see that *no* genes are selected in common (see figure 4.4d).

Again, it is noted that the fRVM and MOEA have produced very different gene selection counts. While the maximum number of times a gene was selected by the fRVM was 55, the highest selection count given by the MOEA was only 9. It is also noted that, overall, the numbers of gene selection counts for *both* the fRVM and MOEA are lower on this dataset than they are on the Leukaemia (136 max) and Colon cancer datasets (94 max) studied previously. This may be due to the fact that, under the partitioning scheme used here, at most 4 BRCA1 samples will be used in training at a time. Given such paucity of data it is likely numerous different classifiers can be learned that will give good training performance and this is being reflected in lower gene selection counts arising from different gene selections on different splits.



(a) fRVM (b) MOEA

(c) fRVM (d) MOEA

Figure 4.4.: Gene Selections for Hedenfalk's Breast Cancer dataset over 100 splits. Top row: all selections. Bottom row: selections coincident to both halves of each split.

Rank	**Count**	**Gene**
1	55	keratin 8
2	53	transducer of ERBB2, 1
3	37	cyclin D1 (PRAD1: parathyroid adenomatosis 1)
4	33	ESTs
5	30	ESTs
6	29	cell division cycle 4-like
		Human GABA-A receptor pi subunit mRNA, complete cds
7	26	catenin (cadherin-associated protein), alpha 1 (102kD)
8	24	mutS (E. coli) homolog 2 (colon cancer, nonpolyposis type 1)

Table 4.6.: fRVM Gene Selections on Breast Cancer dataset.

Rank	Count	Gene
1	3	transducer of ERBB2, 1
2	1	transcription factor AP-2 gamma
		Human GABA-A receptor pi subunit mRNA, complete cds
		microsomal glutathione S-transferase 1
		ESTs
		EST
		B-cell CLL/lymphoma 7b
		mutS (E. coli) homolog 2 (colon cancer, nonpolyposis type 1)
		transmembrane 4 superfamily member 1

Table 4.7.: Coincident fRVM Gene Selections on Breast Cancer dataset.

Rank	Count	Gene
1	9	transducer of ERBB2, 1
2	6	nuclear factor I/B
		KIAA0020 gene product
		fatty acid binding protein 5 (psoriasis-associated)
		ESTs
		ESTs

Table 4.8.: MOEA Front Gene Selections on Breast Cancer dataset.

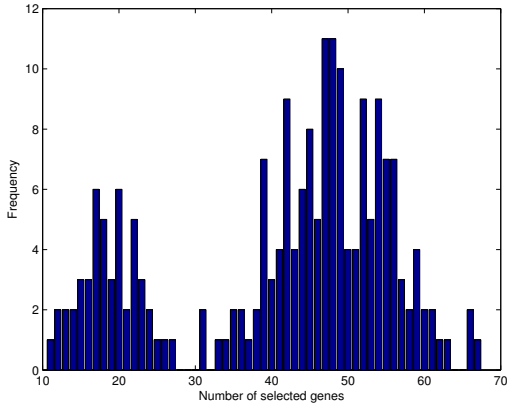
4.6. Analysis of gene selection counts

In the experiments presented above the gene selection total and coincident counts for the MOEA were consistently lower than for the fRVM. Therefore, to better understand these gene selection count discrepancies, further analysis was carried out. Using the Leukaemia dataset, the most studied of the three cancer datasets, the EA and fRVM were run for 100 partitionings using identical splittings for training both sets of classifiers. As before, gene selections were recorded and counted. In addition to the gene selection counts model predictions were obtained for each of the splittings. Training data predictions were produced using the half of the dataset on which the models were trained and test predictions using the halves they were not trained on. Thus for 100 partitionings we obtain 200 sets of training predictions and 200 sets of test predictions.

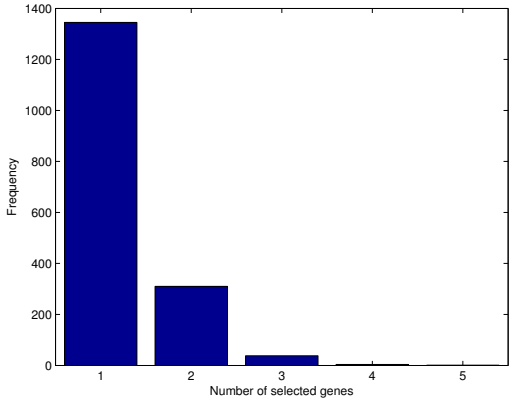
4.6.1. Feature selection counts

As an initial part of the analysis, examination of the numbers of genes being used by each of the trained models was carried out to identify any large differences in model complexities (in terms of numbers of genes used). In figure 4.5 histograms of the number of genes selected by each of the classifiers in the MOEA fronts and trained fRVMs are plotted. A plot of the number of unique genes selected across all the classifiers in each of the MOEA fronts is also given.

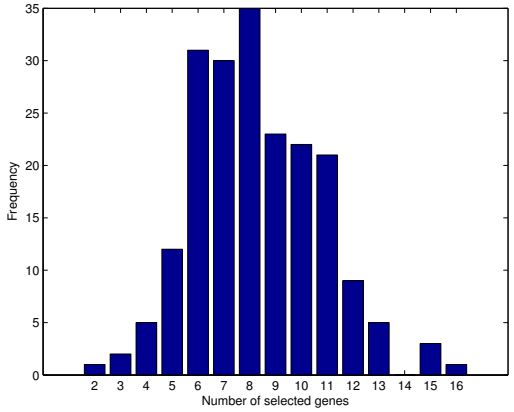
Comparing figures 4.5a and 4.5b, it can be seen that the EA learned classifiers contain significantly fewer gene selections than the fRVM solutions. Overall, the MOEA solutions used at most 5 genes while the fRVM classifier using the fewest genes selected 11 genes. In fact, as can be seen from figure 4.5b, the MOEA classifiers only make use of 1 gene in the majority of cases. It is noted that while the MOEA classifier gene selection counts



(a) fRVM



(b) MOEA



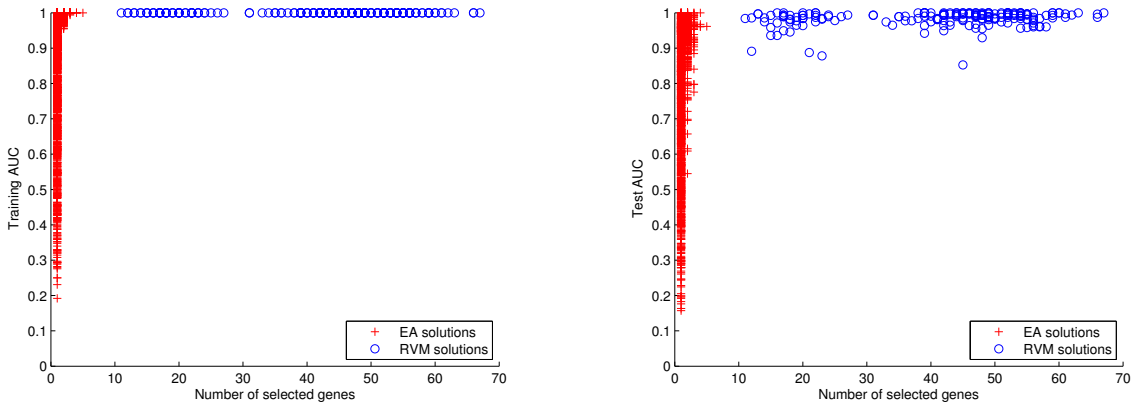
(c) MOEA-front

Figure 4.5.: Number of selected features for Leukemia data gene selections analysis. 4.5a shows the number of genes selected by the fRVM classifiers, 4.5b shows the number of genes selected by each of the MOEA learned classifiers and 4.5c shows the number of unique genes selected across each MOEA located front.

plot has a single mode, the fRVM classifier selection counts plot appears bimodal, though one mode is almost twice the height of the other.

In figure 4.5c counts of the number of unique genes selected by the classifiers in each MOEA generated front are plotted. From this we can see that at most a front contains 16 different selected genes and at least 2 different selected genes across all the MOEA located classifiers in a single front. This distribution has some overlap with the minimum number of different genes selected by the fRVM classifiers (as seen in figure 4.5a) but the majority of fronts still contain fewer gene selections than the fRVM. Taking this in combination with figure 4.5b, we can deduce that the EA is regularly generating fronts consisting of a number classifiers using only 1 gene, but that each classifier is not necessarily using the same gene. This can arise as a result of the complexity measure being minimised by the MOEA – two different solutions may contain the same number of selected features, but it is the $\boldsymbol{\alpha}$ values associated with the features that are used in the complexity measure.

Given the paucity of the dataset it may be possible to obtain good training data performance with very few genes. The MOEA fronts would seem to have been successfully regularised as a result of the complexity measure, $C(\boldsymbol{\alpha})$, and thus located these simpler solutions whilst the fRVM is overfitting during training and selecting more genes than



(a) Training (b) Test

Figure 4.6.: Comparison of features with Training and Test AUC for Leukaemia analysis. EA solutions are marked with red +'s and fRVM solutions by blue o's.

necessary to classify the training data.

To better understand this behaviour and identify an explanation for it, analysis of training and test performance was carried out. This analysis was carried out both in terms of AUC and accuracy, the results for which are presented below in sections 4.6.1 and 4.6.1 respectively.

Area Under the Curve

During the training process the MOEA locates fronts of solutions whose complexity measure has been minimised whilst maximising the AUC obtained on the training data. Thus, the solutions found by the MOEA cover a range of AUC values and gene selection counts on each partitioning. The fRVM, however, learns a single solution, giving a single AUC and count of the number of selected genes for each partition.

In figure 4.6 the genes selection counts from each of the models are compared with their training and test AUC values. While training AUC values for the EA fronts are spread out, those obtained by the fRVM are consistently equal to 1 over a range of model complexities. As we might expect, when using lower numbers of selected genes the EA solutions perform less well and performance improves as the gene selection counts increase. In figure 4.6b it can be seen that model AUC performances have changed on application to the "test" splits. As might be expected, both the EA learnt solutions and fRVM classifier have reduced AUC performance on the test split. However, the fRVM does appear to have overfit on a number of folds dropping from training AUC values of 1.

Recalling the bimodal plot in figure 4.5a showing the number of genes selected by the fRVM classifiers, we can see these two groups in figures 4.6a and 4.6b forming two clusters separated at 29-30 selected genes. Given the varying model complexities one might imagine that the sparser fRVMs may generalise better than the more complex ones. However, both groups perform equally well on the training data and have similar performance changes on the test data.

To further understand these training-test AUC discrepancies, figure 4.7 was plotted comparing AUC values on the training data against those obtained on the test data for each of the trained models. Again, the drops in performance for the fRVM are apparent

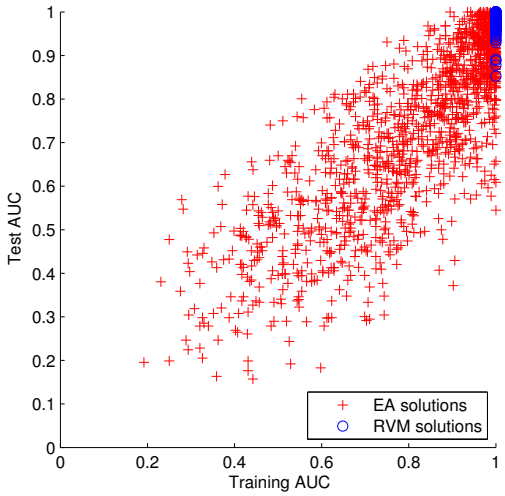


Figure 4.7.: Comparing Training and Test AUC for each learnt classifier on the Leukaemia dataset. EA solutions are marked with red +'s and fRVM solutions by blue o's.

and indicate overfitting. In addition, a better view of performance changes for the MOEA solutions is given. Overall, training and test AUC rates for the MOEA solutions appear correlated but the spread is, at times, rather large – the correlation coefficient matrix is $[1, 0.85; 0.85, 1]$.

Interpreting these changes in AUC, it would appear that the MOEA solutions have located sparse solutions that perform well on the training data but do not always perform as well on the test data. The fRVM solutions, however, appear to be overfitting to the training data using many more gene selections, and thus suffer from reduced generalisation performance.

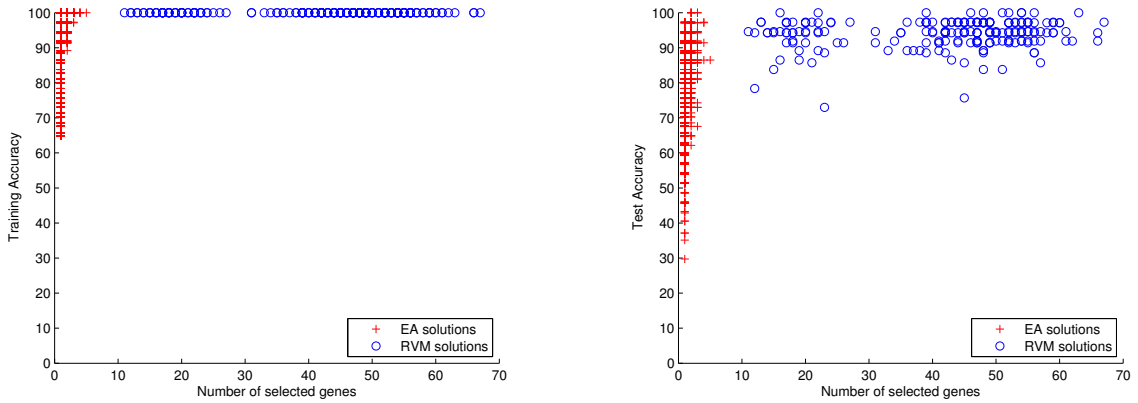
Accuracy

Although the fRVM produces good ROC curves, it is not explicitly trained to do so, unlike the solutions learnt by the MOEA. Thus, a comparison in terms of accuracy was carried out for completeness.

Since the MOEA optimises solution performance in terms of AUC and not accuracy, no decision thresholds are associated with the generated models. Therefore, in order to facilitate comparison between the MOEA and fRVM classifiers, the decision threshold which gave the best accuracy *on the training data* was found for each EA solution and used to produce the training and test accuracies presented here.

As before, training and test performance was compared with the number of selected genes, this time using accuracy rather than AUC, and the results plotted in figure 4.8. Again, the fRVM achieves perfect, 100% accuracy, training performance which drops when applied to the test data. As before there is no meaningful difference between the two fRVM groups. The MOEA, while not specifically trained to optimise accuracy, performs as one would anticipate based on the previous AUC analysis and again, performance drops on application to the test data.

In figure 4.9 model training and test accuracies are compared to better visualise how accuracy has altered. Once again, the RVM appears to have overfit to the training data and



(a) Training (b) Test

Figure 4.8.: Comparison of numbers of selected features with training and test rates on the Leukaemia dataset. For the MOEA the threshold that produces the best training accuracy is used. EA solutions are marked with red +'s and fRVM solutions by blue o's.

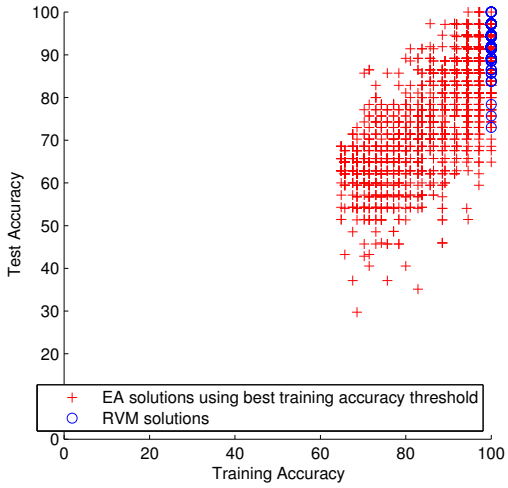


Figure 4.9.: Comparing Training and Test accuracies for Leukaemia analysis

failed to generalise well. Meanwhile, EA solution training-test performance remains loosely correlated dropping slightly when applied to the test data with a correlation coefficient matrix of $[1, 0.76; 0.76, 1]$.

4.6.2. Summary

Examination and comparison of training and test performance supports the belief that the fRVM is overfitting to the training data and thus failing to generalise well. Solutions generated by the MOEA use fewer selected genes than those learnt by the fRVM methodology but still obtain good training accuracies. Drops in performance when applied to the test data show that the relative sparseness of the data compared to the dimensionality make it easy to achieve good training performance with few selected genes but that such sparse models do not always generalise well.

Since the MOEA locates classifiers with fewer selected genes than the fRVM, when one compares the fRVM and MOEA gene selection counts, those obtained for the MOEA are thus lower. Again it is emphasised that although a gene may be selected several times

across an MOEA located front of solutions, it only makes a contribution of 1 to the overall count. It is interesting to note that, although each of the fRVM classifiers use many more gene selections than the EA learned classifiers, the number of coincident selections for the fRVM that appear to be significant is much closer to the lower bound for the counts of fRVM total gene selections than their average (see figure 4.2) i.e. roughly 7 genes appear to be significant in the fRVM coincident gene selections which is closer to the minimum overall number of genes selected, 11 genes, rather than the mean number of selected genes, 43 genes.

Despite lower coincident gene selection counts, the EA still identifies genes useful for classifying the Leukaemia dataset in its top hits. The top coincident gene selection for the EA solutions, ALP2 Amyloid beta (A4) precursor-like protein 2, is also found by fRVM and, more importantly, has previously been identified amongst useful genes [Lee et al., 2003].

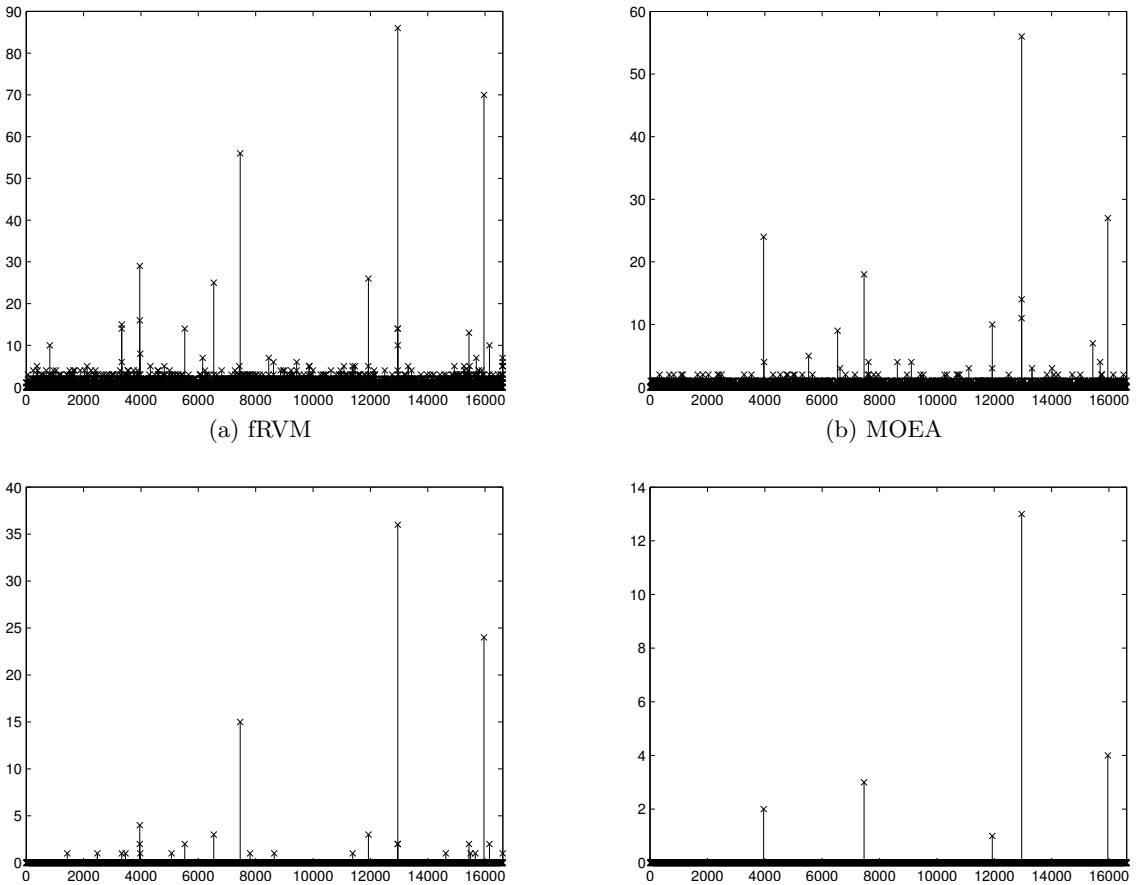
4.7. InChianti Dataset

The InChianti Dataset was collected as a population study in the Chianti area of Italy [Ferrucci et al., 2000; InChianti Group, 2011]. Blood samples were taken, examinations carried out and questionnaires completed from September 1998 to March 2000. Follow-up assessments were carried out 3 and 6 years after the initial examination in the years 2001 - 2003 and 2004 - 2006. Included in the dataset are 16571 genetic probes for 698 people. Some of these probes are not annotated and so manual inspection is required to establish which genes they are associated with. Along with these probes, technical cofactors such as the site the sample was taken at, percentages of cell types, batch numbers and "other cofactors" such as gender, age, body mass index (BMI), education etc. are provided. For a full list of cofactors see Appendix B. Since some of these cofactors were recorded during the follow-up assessments when some individuals were absent, not every individual has a complete set of them. Therefore, the following experiments on this dataset used only those individuals who have a complete set of the required cofactors recorded.

Gender Prediction

As an initial experiment with the InChianti dataset, experiments were carried out seeking to identify the probes associated with gender. Data was grouped into the two classes, male and female, along with the associated cofactors to give 201 members of the male class and 235 members of the female class (obviously gender was not included as a cofactor here). The fRVM and MOEA were then trained on this data over 50 partitions and gene selections recorded. Stem plots of the gene selection counts can be found in figure 4.10 for visual analysis.

The top gene probes selected by the fRVM were RPS4Y1, XIST and ilmn_1685690 with 86, 70 and 56 selections respectively. RPS4Y1 is a female specific gene and XIST a male specific one. Probe ilmn_1685690 represents a sequence unique to the Y-chromosome and therefore male specific. Although the probe is not annotated it does lie close to the Y-specific genes KDM5D and EIF1AY, and is possibly located in a regulatory region [Pilling et al., 2011]. Examination of the gene selections coincident to both halves of



(a) fRVM (b) MOEA

(c) fRVM (d) MOEA

Figure 4.10.: Gene Selections for predicting sex based on 50 splits. Top row: all selections. Bottom row: selections coincident to both halves of each split.

each partitioning reveals RPS4Y1, XIST and ilmn_1685690 to be the top hits again, with counts of 36, 25 and 15. Further details of gene selection counts can be found in tables 4.9 and 4.10. The fRVM has therefore succeeded in identifying important genes for gender classification.

Analysis of the probe selections for the MOEA revealed similar results. The top probe selections for the MOEA (shown in table 4.11) were associated with RPS4Y1 with 56 selections, XIST with 27 selections, ilmn_1755537 with 24 selections, ilmn_1685690 with 18 selections and two probes associated with RPS4X with selection counts of 14 and 11. ilmn_1755537 represents another probe without annotation which, like ilmn_1685690, lies on the Y-chromosome [Pilling et al., 2011]. RPS4X is linked to the X-chromosome. As with the fRVM, the top MOEA coincident probe selection was RPS4Y1 with 13 selections followed by XIST and ilmn_1685690 with 4 and 3 selections each (see table 4.12).

These results, as plotted in figure 4.10 and tabulated in tables 4.9, 4.10, 4.11 and 4.12, have demonstrated the ability of the two classifier types to successfully identify gender specific genes for the purpose of predicting gender on the InChianti dataset. We note that, again, the MOEA is selecting fewer genes for classification than the fRVM.

Rank	Count	Gene
1	86	RPS4Y1
2	70	XIST
3	56	ilmn_1685690
4	29	ilmn_1755537
5	26	PRKY
6	25	BM666601

Table 4.9.: Top fRVM gene selections on the InChianti dataset for sex prediction.

Rank	Count	Gene
1	36	RPS4Y1
2	24	XIST
3	15	ilmn_1685690
4	4	ilmn_1755537
5	3	BM666601
		PRKY

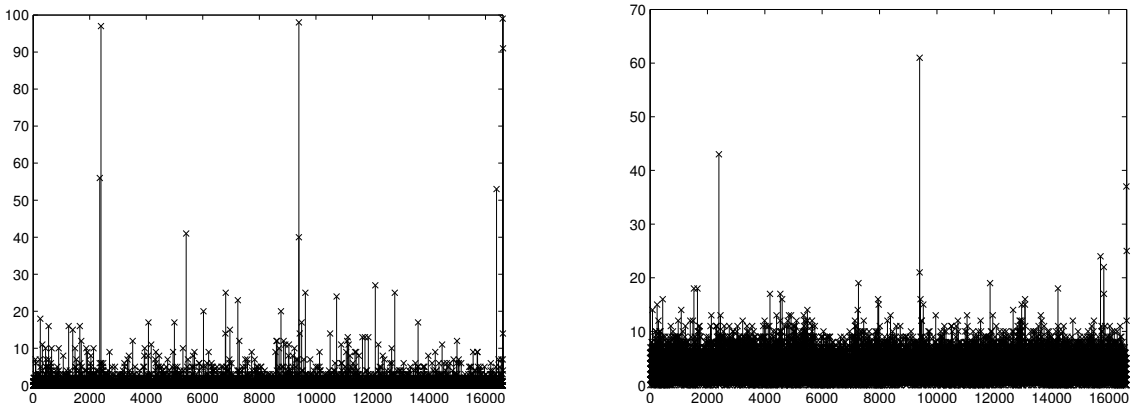
Table 4.10.: Top fRVM coincident gene selections on the InChianti dataset for sex prediction.

Rank	Count	Gene
1	56	RPS4Y1
2	27	XIST
3	24	ilmn_1755537
4	18	ilmn_1685690
5	14	RPS4X
6	11	RPS4X

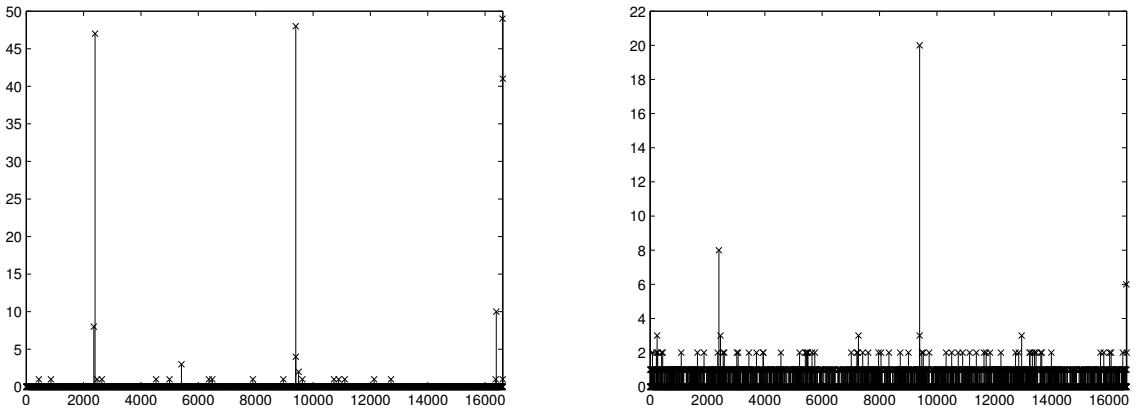
Table 4.11.: Top EA front gene selections on the InChianti dataset for sex prediction.

Rank	Count	Gene
1	13	RPS4Y1
2	4	XIST
3	3	ilmn_1685690
4	2	ilmn_1755537
5	1	PRKY

Table 4.12.: Top EA front coincident gene selections on the InChianti dataset for sex prediction.



(a) fRVM (b) MOEA



(c) fRVM (d) MOEA

Figure 4.11.: Gene Selections for predicting Age group based on 50 splits. Top row: all selections. Bottom row: selections coincident to both halves of each split.

Age Classification

Since ageing is an important risk factor for many conditions, such as dementia, type 2 diabetes and cardiovascular diseases [Harries et al., 2011a], an important investigation would be to see what, if any, gene expressions change as individuals age. Following Harries et al. [2011a] the InChianti dataset was used to create two classes; one consisting of 105 people younger than 65 and the other consisting of 287 people aged 75 and over. The fRVM and EA were trained using 50 splits and their gene selections recorded. Stem plots for gene selections can be found in figure 4.11.

The top hits identified by the fRVM are the level of education cofactor, q_education, with 99 selections, LRRN3 with 98 selections, CD248 with 97 selections and the activity cofactor, q_act_last, with 91 selections. Examination of coincident gene selections reveals an identical ordering with counts of 49, 48, 47 and 41 respectively. As level of education is commonly associated with income, standard of living and, thus, better health and life expectancy, finding q_education as a top hit is not unexpected. The probes associated with LRRN3 and CD248 are also found to be important for age classification by Harries et al. [2011a]. It is also not unexpected that the cofactor q_act_last is a top hit as it provides a measure of physical activity which one might expect to reduce with age.

Examining gene selections from the MOEA fronts reveals a similar picture, albeit with reduced gene selection count levels (see tables 4.15 and 4.16). The probes associated with

Rank	Count	Gene
1	99	q_education
2	98	LRRN3
3	97	CD248
4	91	q_act_last
5	56	CCR6
6	53	ZNF518B
7	41	AY262164
8	40	LRRN3

Table 4.13.: Top fRVM gene selections on InChianti dataset for age prediction.

Rank	Count	Gene
1	49	q_education
2	48	LRRN3
3	47	CD248
4	41	q_act_last
5	10	ZNF518B
6	8	CCR6

Table 4.14.: Top fRVM coincident gene selections on InChianti dataset for age prediction.

LRRN3 and CD248 are selected 61 and 43 times each in total over the 50 splits while cofactors q_education and q_act_last each receive 37 and 25 selections. Probes associated with VAMP5 and WARS receive selection counts of 24 and 22 along with another probe associated with LRRN3 receiving 21 selections. Harries et al. [2011a] also found VAMP5 and WARS to be associated with age group discrimination. Coincident selection counts for the MOEA show LRRN3 to be the most significant with a selection count of 20 followed by CD248 with 8 selections and the education cofactor with a count of 3 selections. LRRN3 was found to be the most important gene for this classification by Harries et al. [2011a].

Extreme Cognitive Divergence

Over 50% of adults over the age of 85 suffer from age-related cognitive decline relating to loss of function of comprehension, language, memory and orientation. Such cognitive decline is one of the "great health threats of old age" [Harries et al., 2011b] and as it progresses not only do social and functional ability decline but there are also increased risks of mortality and institutionalisation. With an increasingly ageing population and worldwide prevalence of dementia predicted to quadruple in the next 40 years there is

Rank	Count	Gene
1	61	LRRN3
2	43	CD248
3	37	q_education
4	25	q_act_last
5	24	VAMP5
6	22	WARS
7	21	LRRN3

Table 4.15.: Top MOEA gene selections on InChianti dataset for age prediction

Rank	Count	Gene
1	20	LRRN3
2	8	CD248
3	6	q_education
4	3	ADM
		CD74
		IL7R
		LRRN3
		RPS5

Table 4.16.: Top MOEA coincident gene selections on InChianti dataset for age prediction

interest in identifying factors leading to cognitive decline.

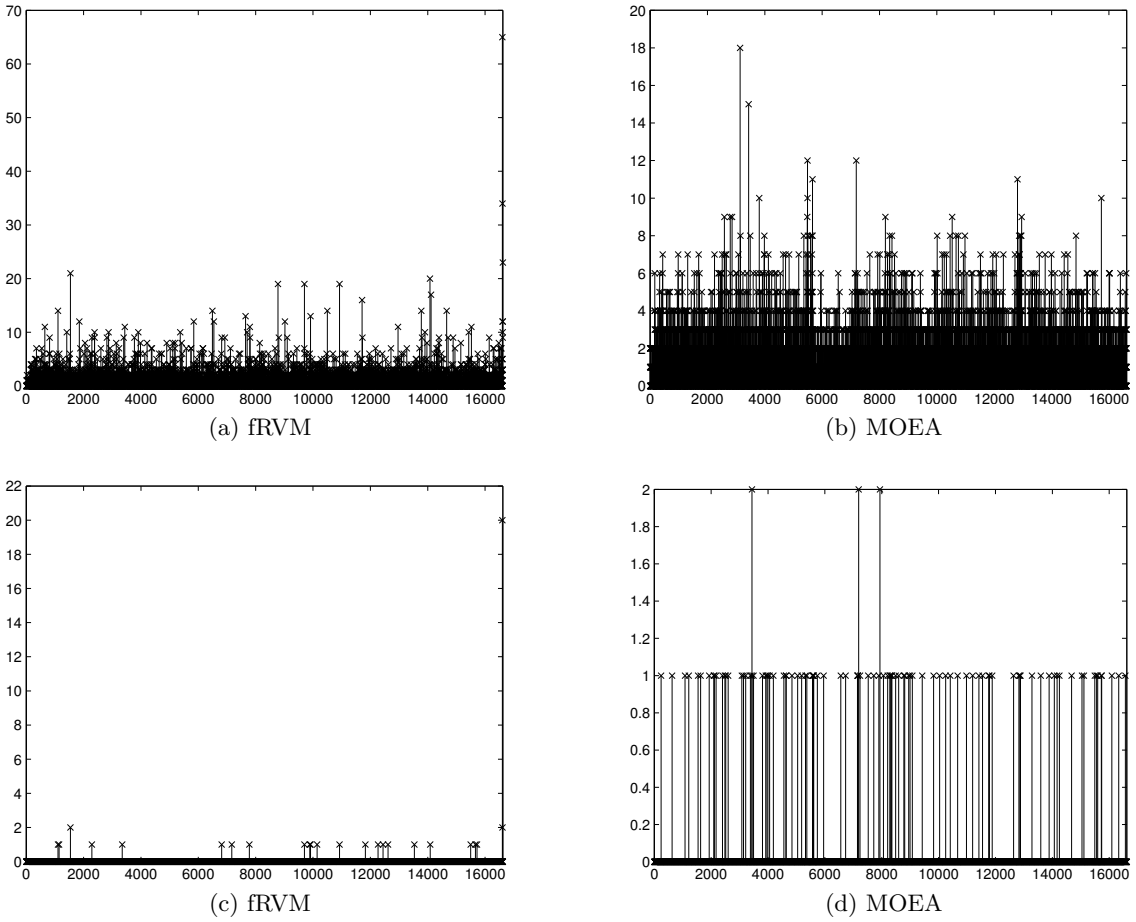
As part of the InChianti study subjects were assessed for overall cognitive function using the Mini-Mental State Examination [MMSE Folstein et al., 1975] at the original sampling and each of the 3 year follow-ups. The MMSE is a 30-point test of global cognitive function that takes roughly 10 minutes and involves simple questions and problems testing various functions including arithmetic, memory and orientation. Scores of 25 or higher are deemed normal, 21 - 24 indicate mild cognitive impairment, 10 - 20 moderate and 9 or less severe cognitive impairment. It is noted that these raw scores may need correcting for age and level of education before interpretation. Low scores indicate the presence of dementia.

An investigation arising from this data is the identification of factors leading to Extreme Cognitive Divergence; i.e., what leads to larger than normal changes in cognitive function as measured by MMSE score? The dataset was used to create two groups of individuals with 'extreme' values of cognitive divergence - very bad and very good. Individuals with an age at the start of the study (baseline) of 60 or more and a baseline MMSE score of at least 23 were used to create the two groups as:

$$t = \begin{cases} 1, & \text{if } S_3 \leq 24 \text{ and } S_0 - S_3 \geq 5 \\ 0, & \text{if } S_3 \geq 28 \text{ and } S_3 - S_0 \geq 0 \end{cases} \tag{4.10}$$

where S_0 was the MMSE score at baseline and S_3 the MMSE score at the 3rd follow-up session 9 years after the initial sample. Using these class assignments and only those samples for which a complete set of cofactors were available produced 35 members of class 1, the "very bad" class, and 97 members of class 0, the "very good" class. The fRVM and EA were trained using 50 splits of this dataset and their gene selections recorded. Stem plots for gene selections can be found in figure 4.12.

The top selection by the fRVM was the age_at_extraction cofactor with 65 hits followed by the q_education cofactor with 34 hits and q_act_last cofactor with 23 hits. It is not surprising that education and age should appear as they are often used to adjust MMSE scores. Similarly, activity levels are not unexpected either since they tend to decline with cognitive function. The top gene selection was C17orf39, a gene located within the Smith-Magenis syndrome region on chromosome 17. Smith-Magenis syndrome is a developmental disorder whose major features include mental retardation. The other top gene selections can be found in table 4.17. Analysis of coincident gene selections showed the age cofactor, age_at_extraction, to be the top hit with 20 selections, followed by C17orf39 and education



(a) fRVM (b) MOEA

(c) fRVM (d) MOEA

Figure 4.12.: Gene Selections for predicting extreme cognitive decline based on 50 splits. Top row: all selections. Bottom row: selections common to both halves of each split.

Rank	Count	Gene
1	65	age_at_extraction
2	34	q_education
3	23	q_act_last
4	21	C17orf39
5	20	SPP1
6	19	IGKV1D-8
		MCM7
		OR2A5
7	17	SRGAP3
8	16	POU2AF1
9	14	DQ895923
		AW204581
		NEK3
		SLC7A6
		THBS1
10	13	KIAA1324
		MKKS

Table 4.17.: Top fRVM gene selections on InChianti dataset for extreme cognitive divergence classification.

Rank	Count	Gene
1	20	age_at_extraction
2	2	C17orf39
		q_education

Table 4.18.: Top fRVM coincident gene selections on InChianti dataset for extreme cognitive divergence classification.

cofactor, q_education, with 2 selections, as shown in table 4.18.

Analysis of MOEA gene selections found a number of different gene selections in the top hits, as shown in table 4.19, the most significant of which was CSF3R with 18 hits. This was followed by DDT with 15 hits and by HBG1 and IGLL1 with 12 hits. Of these, none are known to be linked with cognitive impairment. Interestingly, however, DDT is a gene closely linked to the macrophage migration inhibitory factor (MIF) on chromosome 22 which is an inflammatory mediator associated with rheumatoid arthritis. Rheumatoid arthritis is a particularly painful condition most often appearing in later life that can lead to loss of mobility if untreated. It may be that this is being selected in place of the age cofactor.

Examining the coincident gene selections (table 4.20) showed DDT, IGLL1 and LDHA to have been selected in common 2 times each. LDHA is a protein predominantly found in muscular tissue. Again, none of these genes are known to be linked with cognitive function.

In this experiment the fRVM and MOEA do not agree on the important genes and it is interesting that the MOEA does not identify the education and activity cofactors in its top gene selections. Nor do the top hits identify genes in common with those found by the simple probe analysis carried out by Harries et al. [2011b] who identify CCR2 as the most significant gene associated with cognitive function as humans age. However, in order

Rank	Count	Gene
1	18	CSF3R
2	15	DDT
3	12	HBG1
		IGLL1
4	11	HLA-DRB1
		RPL10A
5	10	DUSP1
		HBG2
		VHL

Table 4.19.: Top MOEA gene selections on InChianti dataset for extreme cognitive divergence classification.

Rank	Count	Gene
1	2	DDT
		IGLL1
		LDHA

Table 4.20.: Top MOEA common gene selections on InChianti dataset for extreme cognitive divergence classification.

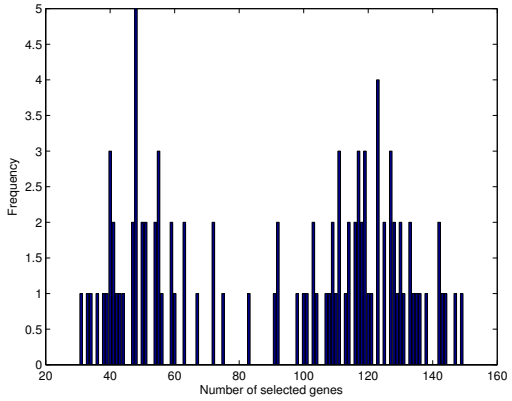
to obtain sufficiently low p-values Harries et al. [2011b] were forced to restrict the gene expressions used down to 635 of those available based on a search for inflammatory and immune related pathways of the Gene Ontology project [Ashburner et al., 2000] website. It is also noted that the gene selection counts in these experiments are lower than those in the other experiments carried out with the InChianti dataset here. To better understand this further analysis was carried out in the following section.

4.8. Further Analysis of Extreme Cognitive Divergence Results

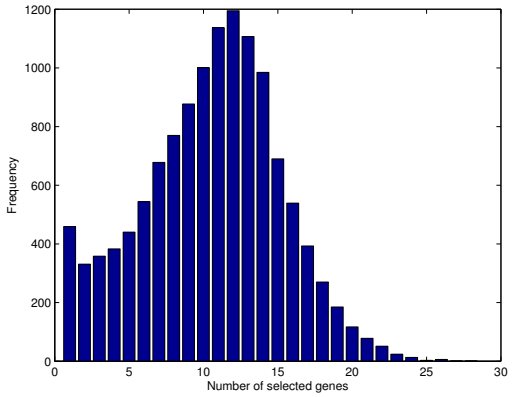
Since the gene selections for the MOEA and RVM do not agree with each other nor do they locate the top hits found by Harries et al. [2011b], an analysis of performance and comparison between the MOEA and RVM solutions was carried out and the results reported below.

The first part of the investigation focused on the number of genes selected by the models to see how complex the underlying models used to produce the selection counts were. Histograms of the numbers of genes used by the models in the MOEA learnt fronts and by the fRVM classifiers are given in figure 4.13 along with a histogram of the number of unique genes selected in each MOEA front. As with the previous investigation using the Leukaemia data, the MOEA is seen to be selecting far fewer genes than the RVM. At *most* the EA selects 28 genes while the minimum number of genes selected by the RVM is 31. This again indicates that the lower selection counts obtained by the EA are due to the sparser models. The presence of a bimodal distribution of fRVM gene selection counts is again noted for later investigation.

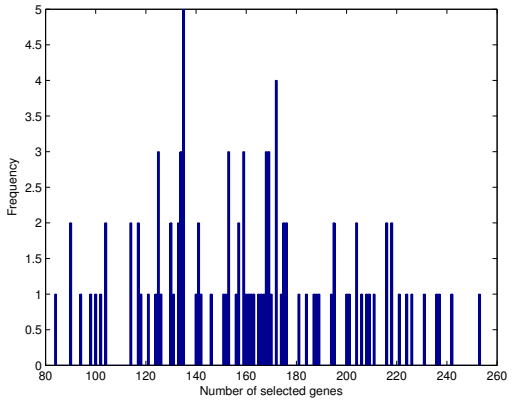
Examination of the number of unique genes selected by classifiers in each EA fronts, as plotted in figure 4.13c, reveals an interesting picture. At minimum, 84 different genes



(a) fRVM



(b) MOEA



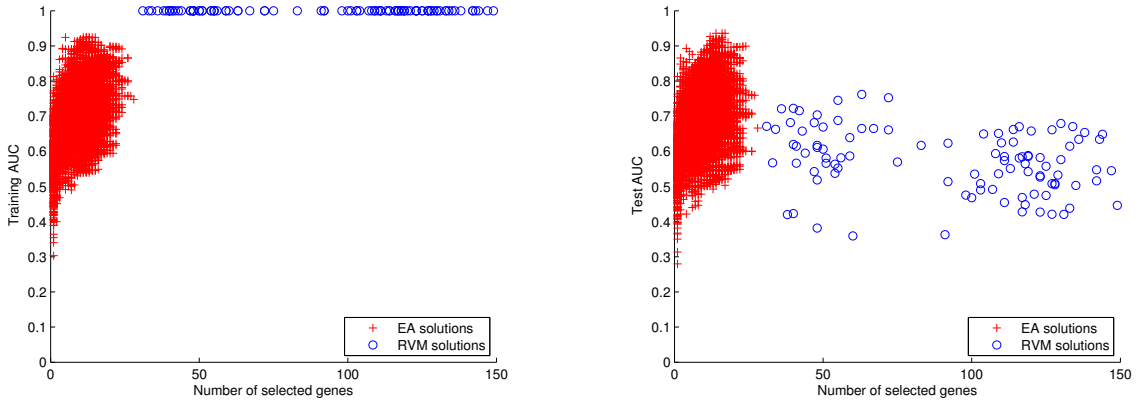
(c) MOEA-front

Figure 4.13.: Number of selected features for Extreme Cognitive Divergence data.

were selected across an EA generated front and at maximum 253 genes were selected across the front. Coupled with figure 4.13b, this shows that the MOEA generated fronts contain multiple models with different gene selection combinations. From this one might infer that there are multiple different gene combinations contributing to extreme cognitive divergence. As numerous different models exist to make these predictions using different gene combinations gene selection counts will be affected. In particular, those for coincident gene selections will be reduced due to model variation.

Since class separation is our primary performance concern, the EA fronts and RVM solutions were compared in terms of AUC and gene selection counts with the results plotted in figure 4.14. From figure 4.14a we can see the RVM has repeatedly achieved a training AUC of 1 while the EA solutions have AUCs centred round 0.7, achieving at best 0.93. Notably, the EA is using fewer genes than the RVM. Moving to figure 4.14b we see that the RVM markedly reduced in AUC performance while the EA performance has remained consistent. In figure 4.15 the training versus testing AUC performance is better visualised further showing the more consistent performance of the EA solutions.

The reduced fRVM performance on the test dataset seems to indicate that it is again generating overly complex models overfitting to the training data. It is observed that the group of less complex fRVM classifiers observed in figure 4.13a generalises better than the more complex one, though only slightly so. Meanwhile, the sparser MOEA learnt models perform more consistently and appear to generalise well, even when using relatively few



(a) Training (b) Test

Figure 4.14.: Comparison of features with Training and Test AUC.

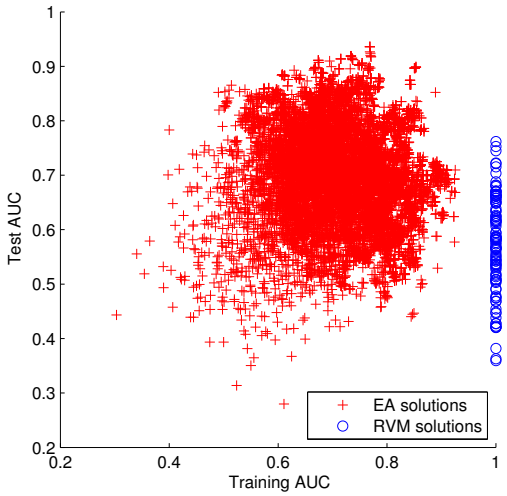


Figure 4.15.: Comparing Training and Test AUC for MOEA and fRVM classifiers predicting Extreme Cognitive Decline.

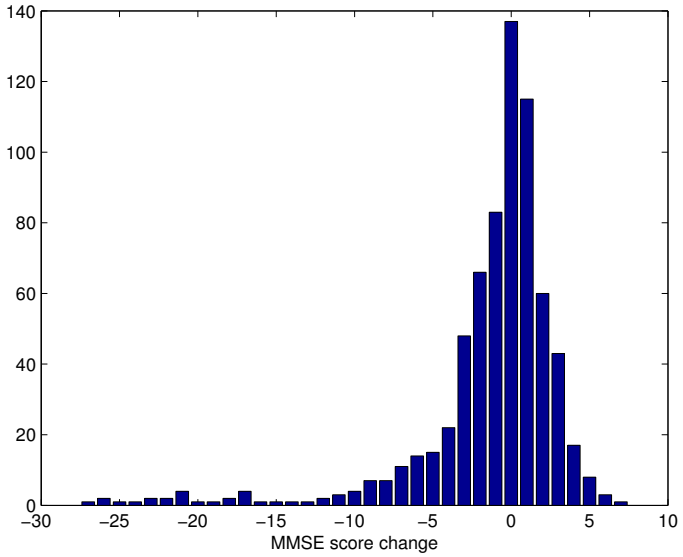


Figure 4.16.: Change in MMSE score over 9 years for each subject in the InChianti study

genes to perform classifications. Results when comparing models in terms of accuracy are similar and can be found in Appendix C.

From these results, we can see that the RVM has repeatedly overfit to the training data and thus failed to generalise well to the test data. The MOEA solutions, while performing less well on the training datasets, generalise well to the test data. Given the reduced numbers of genes selected by the EA solutions and good generalisation performance both in terms of AUC and accuracy compared with the RVM which we see has overfit, we believe the EA to have successfully regularised its solutions during the model training process.

In addition to calculating the training and test AUC for the fRVM and MOEA located classifiers, AUC performance was also compared with that of a classifier using CCR2. Using the partitioning scheme, RVM classifiers were trained using only the CCR2 probe indicated by Harries et al. [2011b] to be significant for predicting cognitive decline, along with a bias unit. As before, predictions were made for the training and test data and the AUC values calculated and plotted in figure 4.17. In this, the CCR2 classifier test performance is clearly negatively correlated with the training performance. Training AUC is much lower than for that of the fRVM, though the average *test* performance is marginally better than the fRVM. In relation to the EA learned classifiers, the CCR2 classifiers' performance lies in the bottom left of the EA cluster, giving performance better than some, but by no means all, of the EA solutions. Clearly CCR2 alone is not enough to get the best training-test AUC performance on this dataset, but does lead to some good classifications. Examining the fRVM and MOEA gene selection counts shows that the CCR2 probe only appears once in the total gene selection counts for the fRVM and three times for the MOEA learned classifiers. It does not occur coincidently in the gene selections for either the fRVM or MOEA learned classifiers. In the next section we investigate the performance of classification using CCR2 alone.

The question remains, however, as to why there is no overlap between the gene selections found by the MOEA and fRVM and why they do not locate the top genes identified by Harries et al. [2011b]. Several conclusions can be drawn as to why this has occurred. First

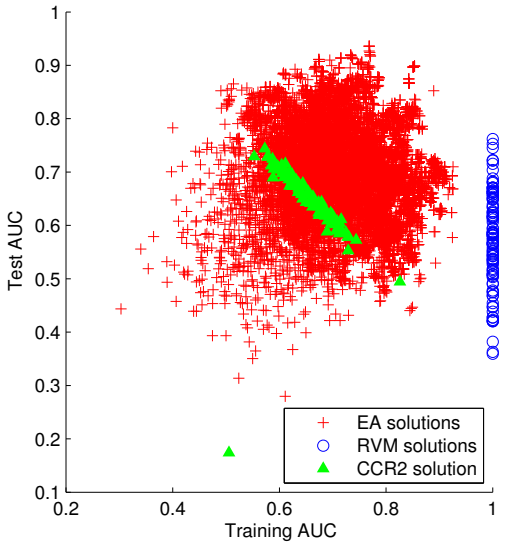


Figure 4.17.: Comparing Training and Test AUC for MOEA and fRVM classifiers predicting Extreme Cognitive Decline with that of an RVM classifier using CCR2.

the RVM has overfit whilst the MOEA has successfully regularised the solutions leading to different gene selections between the two different model learning processes. Second, examination of the changes in MMSE score for the whole InChianti population show it to be somewhat variable with many examples of MMSE score changes (both positive and negative) in the 0 - 5 points range, as seen in figure 4.16, while few values are in the larger range. As a result there is most likely noise in the assigned targets making it hard to correctly identify the responsible genes and thus different key genes were located by different methods. Third, with only 35 of the 132 16618-dimensional samples belonging to the extreme cognitive divergence class, the dataset is likely so sparse as to allow myriad combinations of genes all giving good classification performance.

4.9. Potential models for Extreme Cognitive Divergence classification

In this section we investigate and compare the performance of some of the promising potential classification models highlighted as a result of the experiments with the InChianti Extreme Cognitive Divergence classification dataset. In each case, weights for an RVM classifier were learned by setting the α_m associated with the genes and/or cofactors and the bias unit to 0.

Prior to training these specific classifiers, we first investigated the performance that would be obtained by a random classifier. On any split, the RVM and MOEA classifiers see half the "very good" class examples and half the "very bad" class examples. Thus, a set of 67 targets consisting of 49 very good class targets and 18 very bad class targets was constructed. 100000 random sets of 67 predictions were generated by drawing numbers uniformly at random from between 0 and 1 and AUC values calculated by comparing them with the targets. The average AUC obtained was 0.50, as one might expect since this is the area below the random performance line of the ROC graph, and the standard deviation was 0.08. A histogram of the AUC values obtained by these 100000 random sets

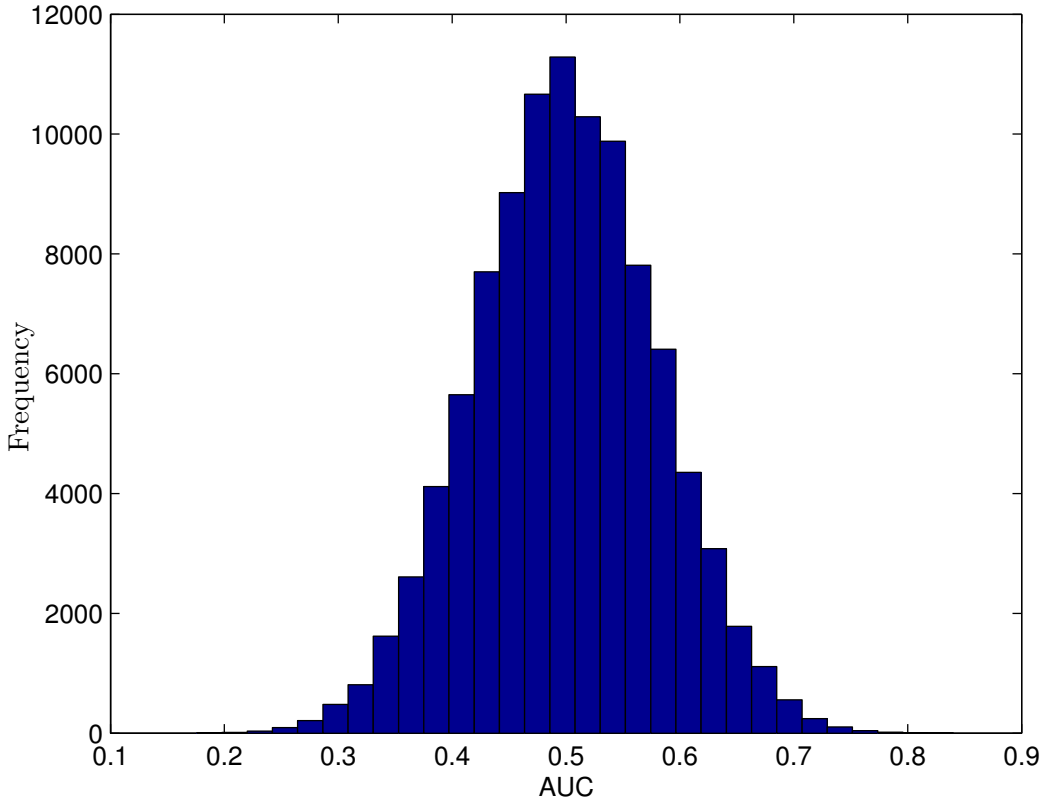


Figure 4.18.: Histogram of AUCs obtained by 100000 random classifications of the Extreme Cognitive Divergence data.

of classifications is shown in figure 4.18. Having obtained these values, we can now compare them with the performance of the gene/cofactor combinations of interest investigated here to see if the performance obtained is better than that of a random classifier.

Two of the top hits identified by the fRVM were the age at extraction cofactor and the education cofactor. In the overall selection counts, age had 65 hits and education had 34 hits while, in the coincident selection counts, age had 20 hits and education had 2 hits. It was therefore decided to investigate the performances that would be obtained using only these cofactors and so models were trained for age only, education only and age and education combined. Training was carried out using the 50 random partitionings of the Extreme Cognitive Divergence dataset and training and test AUC values obtained as before. In figures 4.19a, 4.19b and 4.19c the training-test AUC values are plotted for each of these models.

Using education alone to classify the data (figure 4.19b) yields poor AUC performance, operating within one standard deviation of the random classifier AUC performance. However, classifying based on age alone (figure 4.19a) gives high levels of AUC performance, as does combining education and age (figure 4.19c).

It is not unexpected to obtain good performance using age to classify samples as people are more likely to suffer from dementia as they get older. The AUC results for using education as the basis of classification could suggest that a person's level of educational attainment will not necessarily be reflected in their day to day use of mental faculties. For example, someone with a degree could end up working on the checkout at Tesco not using their full mental capacity whereas someone without a degree might be regularly reading

lots of books keeping their mind active. For the case of age and education combined, we believe the majority of the classification accuracy to be coming from the age cofactor.

Intriguingly, there is a negative correlation between training and test AUC values obtained by these models, along with the other specific models investigated here, such that if a model performs well on the training data it performs poorly on the test data and, conversely, if a model performs poorly on the training data it performs well on the test data. We investigate this later in section 4.9.1

In [Harries et al., 2011b], CCR2 was indicated as being significantly associated with cognitive decline. Surprisingly, it was not found as a significant gene by either the fRVM or the MOEA. In fact, comparison of a classification model using only CCR2 and a bias unit with models learnt by the MOEA in figure 4.17 showed that classification based on CCR2 alone did not obtain optimal performance on the Extreme Cognitive Decline dataset. In this section we investigate and compare the performance of a number of promising genes identified by the MOEA with that of CCR2. Noting that in [Harries et al., 2011b] the data was adjusted for all the cofactors, we present performance for CCR2 alone in figure 4.19d and CCR2 with all cofactors in figure 4.19e. From these we see no significant improvement in performance was obtained by including all the technical cofactors and that CCR2 gives worse AUC performance than the age cofactor based classifier (figure 4.19a).

Examination of coincident gene selections for the MOEA shows the top hits, DDT, IGLL1 and LDHA, to have been selected coincidently twice each. In figures 4.19g, 4.19h and 4.19i we present the training-test AUC performances obtained by classifiers using each of these genes along with a bias unit over the 50 splits. Of the three, DDT performs the best, giving similar performance to that obtained by CCR2. IGLL1 appears to be performing a little better than random though appears to have learnt the class associations in the wrong direction for roughly half the splits. Recall though, from Chapter 2, that this can be corrected for by flipping the model predictions. The performance obtained by LDHA is equivalent to that of the random classifier.

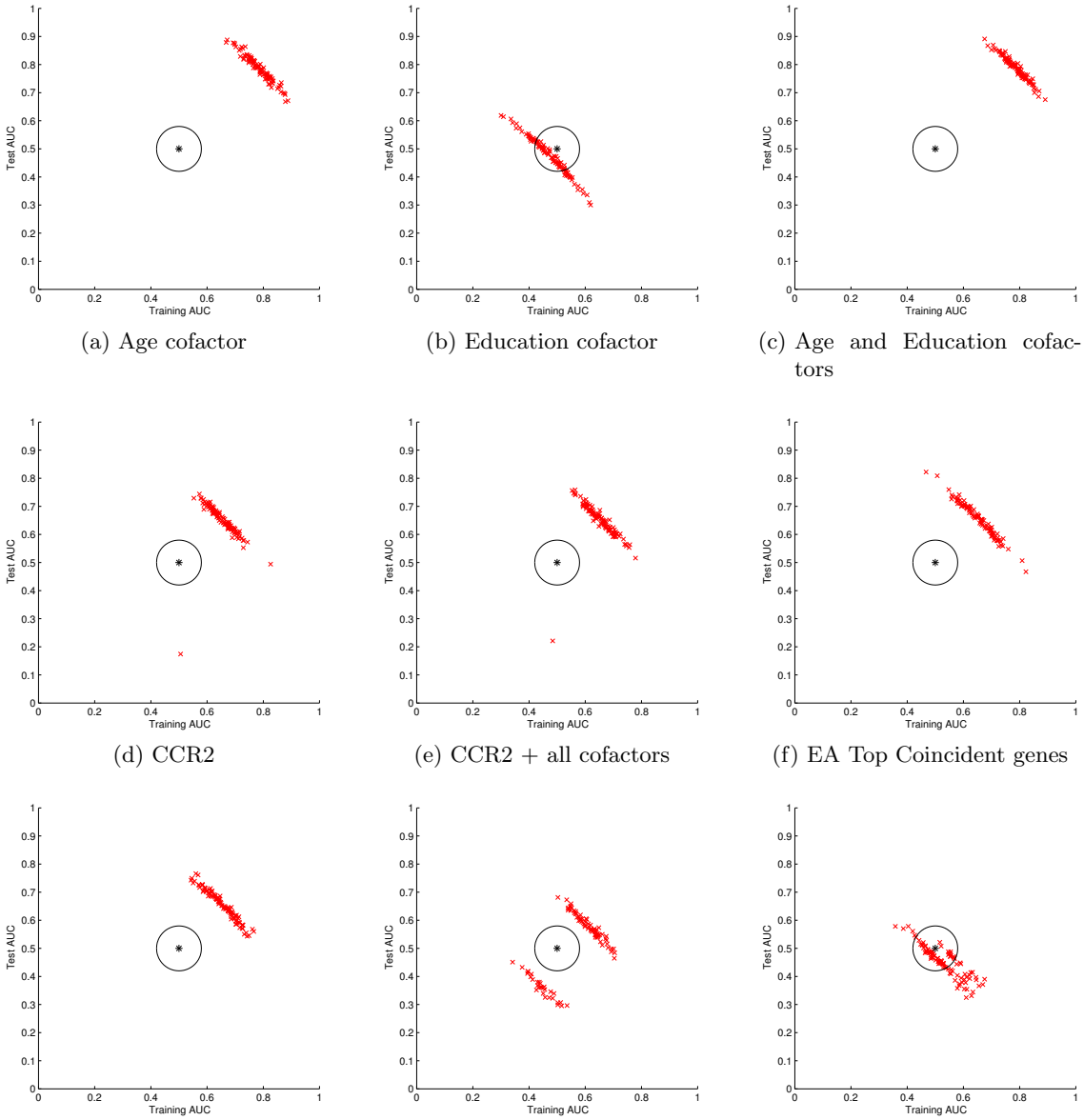
Since all three of these genes, DDT, IGLL1 and LDHA, were selected coincidently an equal number of times, a model was also constructed using all three of these genes and its performance over the 50 splits plotted in figure 4.19f. Here, we see that the combination is performing better than random and marginally better than CCR2, though not significantly so. It is, however, still outperformed by the age based classifier.

Throughout these investigations, there has been a recurring negative correlation between the training and test AUC values obtained by the models. In the following section we investigate possible reasons for it.

4.9.1. Negative Correlation Investigation

In the previous section experiments were carried out on the InChianti Extreme Cognitive Divergence data using models based on genes and cofactors of interest. Oddly, there was a negative correlation between training and testing AUC values across the different models such that; if a model performed well on the training data it would perform poorly on the test data and if a model performed poorly on the training data it would perform well on the test data. Here we investigate the possible cause of this negative correlation.

One possible explanation for these negative correlations is that the models may not



(a) Age cofactor (b) Education cofactor (c) Age and Education cofactors

(d) CCR2 (e) CCR2 + all cofactors (f) EA Top Coincident genes

(g) DDT (h) IGLL1 (i) LDHA

Figure 4.19.: Comparison of Training and Test AUC for gene selection combinations of interest. Red x's mark the performance of the gene combination being evaluated. Black * marks the average performance obtained by a random classifier. The black circle around it shows the region encompassed by one standard deviation of the AUCs obtained by a random classifier from the mean performance based on 100000 randomly generated sets of classifications.

be learning one of the classes well during training and therefore, when testing occurs, performance will not be consistent. To investigate this, we plot training and test data ROC curves for three different age based classifiers. In figure 4.20a we plot the training and test ROC curves for the model with the highest training AUC and in figure 4.20b we plot the ROC curves for the model obtaining the highest test AUC. Perhaps unsurprisingly, the training and test curve locations appear to be swapped between figures 4.20a and 4.20b. In figure 4.20c the ROC curves for the model with the minimum training-test AUC difference are plotted. As expected the training and test curves lie close together. Had the negative correlations been due to misrepresentation of one of the classes we would expect there the ROC curves to be skewed, resulting in the training-test AUC differences, but none appears in these plots.

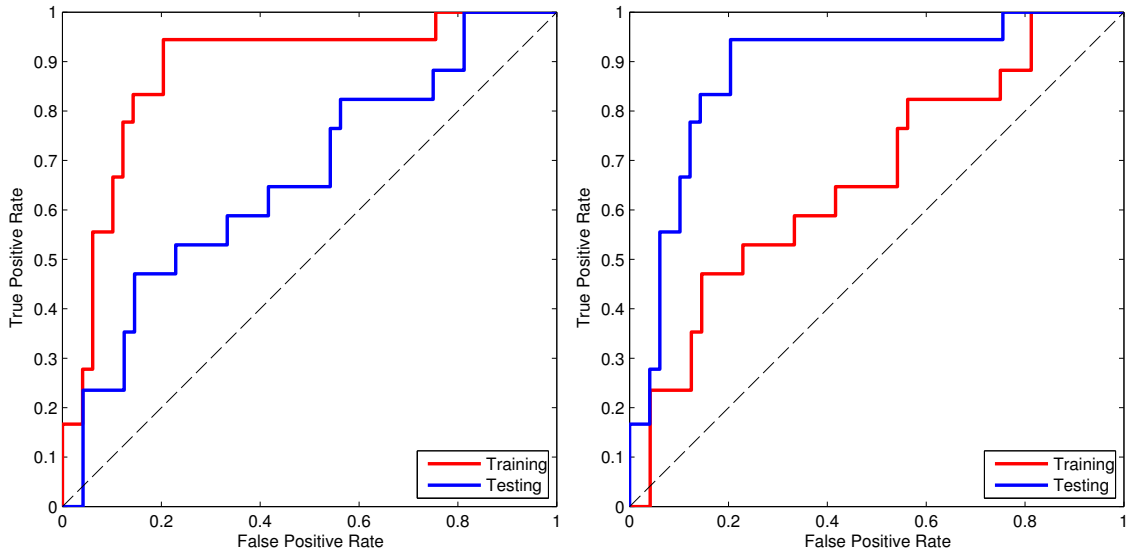
Another possible explanation is that these negative correlations are the result of mislabelling some small number of the samples in the data. During training, the weights learned by the RVM classifier are those that give the minimal error across the training data and thus a small number of mislabelled points will only have a minor impact on the weights learned. If a small number of points were to be mislabelled in each class, the splitting procedure would lead to different numbers of them being divided between training and test data for each of the splits and thus performance bounded by the number of mislabellings present in the data on which the model was being evaluated.

To determine if this was the cause, a simple synthetic dataset consisting of 132 points was constructed. 35 points were generated for class 1 and 97 points were generated for class 0. This gives a dataset with the same number of samples and class proportions as the Extreme Cognitive Decline data.

Training runs were then carried out on this synthetic data using the RVM (with a bias unit) using 50 stratified splits of the data. On each training run we deliberately mislabelled a number of samples from each class; on the 1st run of 50 partitionings 1 example per class was mislabelled, on the 2nd run 2 examples per class were mislabelled, on the 3rd 3 were mislabelled etc. Using these mislabelled targets, training and test AUC values were calculated and plotted in figure 4.21. From this we can see that with just 1 mislabelling per class we obtain a negative correlation, with the performances moving increasingly southwest as the number of mislabellings grows.

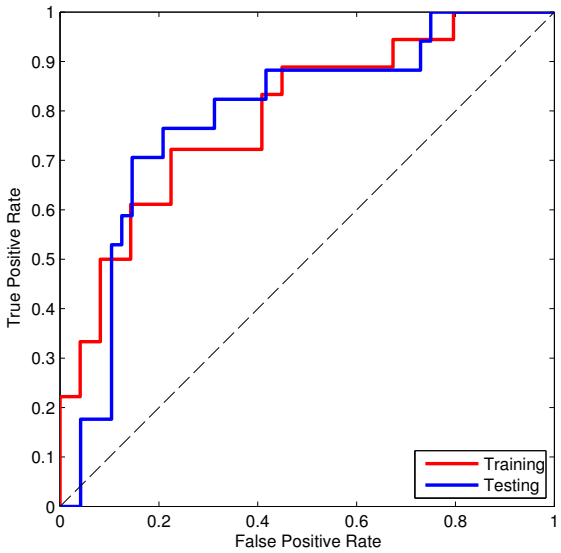
To better understand why this is happening we need to consider how the mislabelled points relate to the overall dataset and the splitting procedure. In the complete dataset, consisting of 35 class 1 points and 97 class 0 points, a mislabelling of 1 point in the class 1 samples is equivalent to 2.7% of the class 1 samples being mislabelled while a mislabelling of 1 point in class 0 is equivalent to 1.0% of the class 0 samples being mislabelled.

Recall, however, that the splitting procedure repeatedly randomly divides the data into training and test datasets. Consider the case of a single mislabelling per class. Upon partitioning the dataset, the training and test splits will consist of 66 points each with the mislabelled points randomly divided between them. In table 4.21 we see the four possible ways in which these mislabelled points could be distributed between training and test partitions in terms of both counts and percentages. As we can see, a particular half of the data could contain none, one or two of the mislabelled points with the other half containing the remaining ones. These four possible distributions of the mislabelled points correspond



(a) Train AUC $\gg$ Test AUC

(b) Train AUC $\ll$ Test AUC



(c) Train AUC $\approx$ Test AUC

Figure 4.20.: ROC curves for extrema and center of distribution of Training vs. Test AUC for Age based classification of Extreme Cognitive Divergence dataset. Red curves mark the training performance and blue curves mark the test performance. The dashed black line marks the $T = F$ line obtained by random performance.

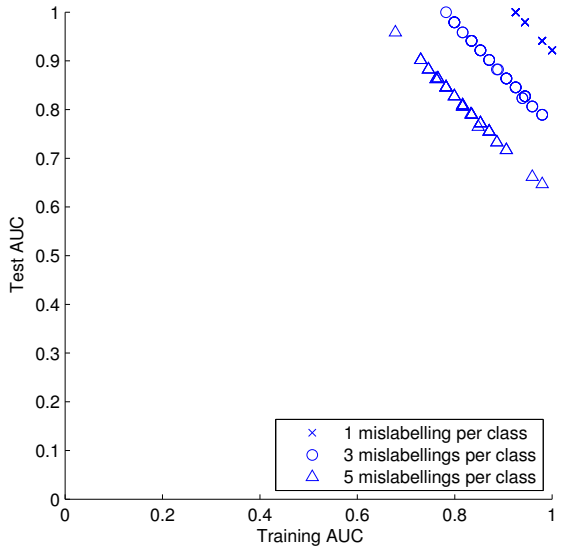


Figure 4.21.: Training-Test AUC values obtained by deliberate mislabelling of 1, 3 and 5 samples per class.

Training Partition			Test Partition		
C_0	C_1	Overall (%)	C_0	C_1	Overall (%)
1 (2.1%)	0 (0%)	1 (1.5%)	0 (0%)	1 (5.7%)	1 (1.5%)
0 (0%)	1 (5.7%)	1 (1.5%)	1 (2.1%)	0 (0%)	1 (1.5%)
0 (0%)	0 (0%)	0 (0%)	1 (2.1%)	1 (5.7%)	2 (3.0%)
1 (2.1%)	1 (5.7%)	2 (3.0%)	0 (0%)	0 (0%)	0 (0%)

Table 4.21.: Counts and percentages for possible divisions of mislabelled points between training and test partitions with a single mislabelling per class.

to the four training-test AUC combinations plotted for a single mislabelling per class in figure 4.21, as marked with blue x's. Over the 50 partitionings these four different divisions of mislabelled points between training and test datasets have arisen with the situations where all the mislabelled points fall into either the training or test partition defining the extrema. If only a single training-test partitioning were used only one distribution of the mislabelled points would be obtained. However, by using the partitioning procedure we obtain the different distributions of the mislabelled points between the training and test data and as a result see this negative correlation in our results.

Based on these results, we believe the negative correlations presenting in the experiments carried out in this section to be the result of a mislabelling of a small number of points in the Extreme Cognitive Divergence data, confirming the concerns expressed earlier in section 4.8.

4.9.2. Summary

In this section we have investigated a number of potentially useful genes and cofactors for classification of the Extreme Cognitive Divergence data and compared their performance with that obtained by a random classifier and a CCR2 based classifier. Of the models investigated, the age cofactor based classifier was found to perform best, outperforming models based on CCR2. As people get older they have a higher likelihood of suffering from dementia so it is not unexpected that models based on this cofactor have good classification

performance. In section 4.7 age was identified among the top gene selections by the fRVM but not by the MOEA in the experiments carried out. In fact, over the 50 splits the MOEA had only selected the age cofactor 6 times in total and 0 times coincidently.

Examination of the top coincident gene selections by the MOEA highlighted three possible genes for classification, albeit with low hit numbers. Models based on each of these genes individually and combined were trained and evaluated. Of the three genes DDT was found to perform the best, giving performance equivalent to that obtained by CCR2. A combination of the three was seen to be marginally better than CCR2 based classification.

Throughout these investigations a negative correlation was found between the training and test AUC values obtained by the models. Investigation using a synthetic dataset in section 4.9.1 indicated that this was likely due to a mislabelling of a number of the samples in the dataset confirming concerns raised in section 4.8.

4.10. Chapter Summary and Conclusion

In this chapter a Multi-objective Evolutionary Algorithm (MOEA) has been presented that produces Pareto fronts of solutions containing the best trade-offs between model complexity and area under the ROC curve. Using the partitioning method described by Li et al. [2002], experiments were carried out on a number of bioinformatics datasets using the MOEA and 'fast' Relevance Vector Machine to select genes for classification demonstrating the MOEA's ability to successfully identify important genes for classification. As a result of penalising model complexity the number of times genes were selected by the MOEA were thus lower than those of the fRVM whilst still finding key genes and giving good generalisation performance.

Several different experiments were carried out on the newer InChianti dataset with the fRVM and MOEA utilising the repartitioning methodology. For the case of gender classification, genes associated with sex, particularly the X and Y chromosomes, were successfully identified by both the MOEA and fRVM. Experiments carried out to identify genes associated with ageing by classifying two different age groups identified LRRN3 as an important gene, confirming recent findings by Harries et al. [2011a]. However, when applied to the task of classifying examples of extreme cognitive divergence the MOEA and fRVM selected different genes to each other as well as those found in a recent paper on the subject [Harries et al., 2011b] and so further results analysis was carried out.

While the fRVM classifiers were found to achieve near perfect performance on the training data, they frequently failed to perform as well on the test data, apparently overfitting. In contrast, the sparser solutions learnt by the MOEA generalised well to new data. As a result of these gene selection discrepancies and performance changes several conclusions were drawn. First, the RVM classifiers had overfit to the training data whilst the inclusion of the complexity measure aided regularisation of the MOEA located solutions. Secondly, noise was present in the targets and thus it was hard to correctly identify those genes responsible for extreme cognitive divergence and thus different methods selected different gene combinations. Finally, the dataset was so sparse as to allow numerous different combinations of gene selections all of which yielded good classification performance on the training data.

When carrying out gene selection for genetic condition classification problems correct gene selection and avoiding overfitting are important. The MOEA presented here has been shown to not only consistently avoid overfitting, producing sparse models that generalise well, but also successfully identify key genes for classification. By using AUC and model complexity the MOEA located models providing the best class discrimination for a given complexity. Results analysis showed that it was possible to achieve good training performance with very few selected genes further highlighting the problems of such sparse datasets.

4.11. Future Work

In some classification tasks it is conceivable that there may be a number of underlying sub-classes that have been grouped together to form one of the classes the model is trying to predict. For example, if we sought to screen someone for cancer based on a set of gene expressions then we know there are different cancer types: Breast cancer, Colon cancer, Leukaemia etc. Each of these types of cancer has different genes associated with it by all are examples of the cancer class. However, it is unlikely we would be aware as to which of the samples belonged to each of the underlying classes when building our model. For the case of the InChianti dataset when trying to predict cognitive decline, it may be that a number of different conditions affect MMSE scores e.g. Alzheimers, stroke propensity, depression propensity, genetic (as opposed to physical) age etc. However, this information is not available. Future work could involve building a model for automated detection of these underlying sub-classes and the genes associated with them. As a starting point one might use the Hereditary breast cancer dataset [Hedenfalk et al., 2001] which consists of three cancer classes; BRCA1, BRCA2 and sporadic. Typically this dataset is used for a 1 vs all classification task of BRCA1 vs BRCA2 and sporadic. Since this is an actual bioinformatics dataset for which we know the underlying classes involved in the two-class problem it would prove useful for testing the efficacy of a newly developed algorithm seeking to detect multiple causes.

Since we have been learning sparse models for classification of bioinformatics datasets, there is the possibility that selection of one particular gene may result in others being disregarded, effectively "masking" them as a result of selecting the first gene. Finding less important genes is still important though. Investigation into this possible masking effect could be carried out as part of future work so as to identify any other important genes that may be hidden but which still yield good classification performance and, more importantly, are linked to the genetic condition of interest. Initial investigations might involve the removal of the top gene selections from the dataset and then re-running the training process with the remaining genes. Any masked genes should then be selected and appear a significant number of times in the gene selection counts. Although such experiments could demonstrate the existence of these masked genes, a more principled method for detection of masking as part of the training process would be desirable.

4.12. Acknowledgements

The work carried out on the InChianti dataset in section 4.7 benefited greatly from the input of David Melzer, Luke Pilling and Lorna Harries of the Peninsula College of Medicine and Dentistry, Exeter, who provided a copy of the latest InChianti dataset along with draft copies of research papers and personal correspondence helping identify unlabelled gene probes in the dataset.

5. Soft classifiers from hard: Using Approximate Bayesian Computation to average hard classifiers

Parts of the following chapter have appeared as:

> Clark, A. and Everson, R. (2011c). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at ABCiL 2011, Imperial College London.
>
> Clark, A. and Everson, R. (2011d). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), University of Exeter.

5.1. Introduction

Classification is a common task in supervised machine learning in which models are learnt, based on some training data, with which to label an input as belonging to a particular class. For example, a doctor screening for cancer would want to assign tissue samples to either the cancerous class or non-cancerous class. These assignments can either be 'hard', in which only the labelling is given, or 'soft', whereby some measure of the certainty of the assignment of the input to a particular class is provided either in the form of a score or posterior probability. A desirable feature in a classifier is for it to give a soft classification in the form of a posterior probability of class membership taking into account any uncertainty in the prediction.

Unfortunately, many classifiers, produce only hard, binary, yes/no classifications, and, in general, learn a single set of parameters during training whose values are uncertain due to finite available training data and possible mismatch between the learnt model and the data generating model. Such classifiers are frequently used in industrial applications e.g. the NATS Short Term Conflict Alert (STCA) system which gives hard classifications predicting whether or not pairs of aircraft trajectories will become dangerously close [Reckhouse et al., 2010].

In most real problems of interest, data is limited and thus correct model and its parameters are uncertain, e.g. in the STCA problem there are few training examples for aircraft collision trajectories. Ideally one would make use of the Bayesian framework to ameliorate this parameter uncertainty by averaging over classifiers weighted by the posterior probability of their parametrisations. Thus, for binary classification when we have a likelihood model $p(t|\mathbf{x}', \boldsymbol{\theta})$ for a new input $\mathbf{x}'$ we make the prediction of probability of membership

to the $t = 1$ class as:

$$\hat{y} = p(t = 1|\mathcal{D}_{\text{tr}}) = \int p(t = 1|\mathbf{x}', \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}) d\boldsymbol{\theta} \tag{5.1}$$

where $\boldsymbol{\theta}$ is a model parametrisation and $\mathcal{D}_{\text{tr}}$ is the training data. Unfortunately, as hard classifiers do not supply a likelihood from which to calculate the posterior, the Bayesian framework cannot straightforwardly be used to obtain these posterior probabilities.

Approximate Bayesian Computation (ABC) methods [Marjoram et al., 2003] allow one to circumvent this inability to evaluate the posterior probabilities of model parametrisations by using a surrogate measure of posterior probability and sampling procedures. ABC methods are commonly used in regression tasks [Toni et al., 2009] or in a "model fitting" context [Wilkinson and Tavaré, 2009] to sample parametrisations for a model simulating measured data, e.g., a system of ODEs. Using the sampled model parametrisations, datasets, $\mathcal{D}^*$, are simulated and compared with the original dataset using some distance function to evaluate the difference between the two datasets. Should the distance be lower than a pre-defined error threshold then the sampled parametrisation is kept; otherwise, it is discarded. Thus, the resulting set of sampled model parametrisations is drawn from a distribution conditioned on the distance function and value for the error threshold.

Having obtained S model parametrisations for a hard classifier, $f(\mathbf{x}; \boldsymbol{\theta})$, one can obtain an estimate of the posterior prediction for an input vector $\mathbf{x}'$ belonging to the $t = 1$ class as:

$$\hat{y} = \frac{1}{S} \sum_{i=1}^{S} f(\mathbf{x}'; \boldsymbol{\theta}_i) \tag{5.2}$$

where $\boldsymbol{\theta}_i$ is i^{th} sampled parametrisation.

Unfortunately, for many real datasets of interest, it is unclear how to carry out a Euclidean distance based comparison of the training and simulated datasets. Instead, one can use knowledge of the problem to select informative summary statistics for use in the distance function in order to capture the desired structure in $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}^*$. For example, one might compare the means and standard deviations of the original and simulated dataset when one believed the data to be drawn from a Gaussian distribution – though if one believed the data to be distributed in this way ABC methods would be somewhat excessive.

Receiver Operating Characteristic (ROC) curves are a commonly used method for visualising and allowing assessment of the performance of a binary classifier over the full range of misclassification costs. The area under the ROC curve (AUC) is equivalent to the Mann-Whitney-Wilcoxon statistic and provides a measure of how well a classifier is able to separate classes over the full range of misclassification costs [Hand and Till, 2001; Fieldsend and Everson, 2008a]. In chapter 4 AUC was successfully used as an objective in a multi-objective evolutionary algorithm in order to locate Relevance Vector Machine classifiers giving the best class discrimination for a model complexity.

In the preliminary investigation presented in this chapter, an Approximate Bayesian Computation Markov chain Monte Carlo algorithm (ABC MCMC) is used to sample parameters for hard classifiers using AUC as a measure of classification performance. It is the AUC for the sampled classifier that is used as the summary statistic in the dis-

tance function to compare the classifications made by the hard classifier with the training data. In section 5.4 this method is investigated for a model problem using synthetic data, demonstrating that the prediction from the sampled chain provides a good estimate of the posterior classification probability and yields an ROC curve close to that of the Bayes classifier with full knowledge.

As a result of the ABC MCMC sampling process one obtains a large number of different classifier parametrisations. While averaging over the numerous parametrisations sampled by the ABC MCMC chain provides a good estimate of the posterior classification probability, it may be impractical to average over such large numbers of models when rapid classification is needed e.g. in safety critical systems [Reckhouse et al., 2010], or when computing power is limited e.g. portable devices. Therefore, methods for learning a sparse set of model weightings that result in equivalent soft classifications to the equally-weighted average over the whole chain are investigated. A comparison of various model averaging methods is presented in section 5.5 demonstrating the ability of a sparse set of classifiers drawn from the MCMC chain to reproduce the performance of the whole chain.

This chapter begins with an introduction to Approximate Bayesian Computation in section 5.2. We then move on to describe the proposed application of ABC MCMC using AUC in the distance function for sampling parametrisations for hard classifiers in section 5.3. The performance of the algorithm is illustrated on a model problem in section 5.4, demonstrating the ability of the ABC MCMC algorithm to produce ROC curves close to that of the Bayes optimal ROC curve. An investigation into the possibility of learning a sparse weighting of the ABC MCMC sampled classifiers which reproduces the performance of the chain is given in section 5.5. In section 5.6 an experiment is carried out on the Waveform benchmarking dataset [Rätsch et al., 2001] confirming the results found in sections 5.4 and 5.5. Finally a summary of the chapter is given in section 5.7 and possible further work outlined in section 5.8.

5.2. Approximate Bayesian Computation

In order to evaluate posterior distributions one needs to be able to evaluate a likelihood function. Unfortunately for many problems the likelihood is either impossible or too computationally expensive to calculate. In particular, for hard classifiers such a likelihood does not exist. In Approximate Bayesian Computation (ABC) methods, evaluation of the likelihood is replaced by a comparison between the observed training data and the data simulated by the model parametrisation, $\boldsymbol{\theta}$. ABC methods have been successfully applied to problems of non-linear regression [Blum and François, 2010], epidemiology [Toni et al., 2009] and species divergence modelling from fossil record analysis [Wilkinson and Tavaré, 2009; Wilkinson et al., 2011].

The goal of the ABC methods is to approximate the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}) \propto p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ where $p(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta})$ is the likelihood of the observed data $\mathcal{D}_{\text{tr}}$ given $\boldsymbol{\theta}$ and $p(\boldsymbol{\theta})$ is the prior distribution.

Overall, ABC methods follow the same general form. This begins with sampling a model parametrisation to be evaluated, $\boldsymbol{\theta}^*$, from a proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta})$. Using $\boldsymbol{\theta}^*$, a dataset $\mathcal{D}^*$ is 'simulated'. Finally, the simulated dataset $\mathcal{D}^*$ is compared to the

Algorithm 6 ABC MCMC algorithm

Require: $\mathcal{D}_{\text{tr}}$ *Training Data*
ϵ *Error threshold*
samples *Number of samples to generate*

$i := 0$, $\boldsymbol{\theta}_i :=$ `initialise()` *Initialise* $\boldsymbol{\theta}_i$
`while` $i <$ samples
  $\boldsymbol{\theta}^* \sim q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i))$ *Propose* $\boldsymbol{\theta}^*$ *according to* $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$
  $\mathcal{D}^* \sim f(\boldsymbol{\theta}^*)$ *Simulate a dataset using the model* $f(\boldsymbol{\theta}^*)$
  `if` $d(\mathcal{D}_{\text{tr}}, \mathcal{D}^*) \leq \epsilon$
    $\rho :=$ `min` $\left(1, \frac{p(\boldsymbol{\theta}^*)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)}\right)$
    `if rand` $\leq \rho$
      $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}^*$
    `else`
      $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}_i$
    `end`
  `else`
    $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}_i$
  `end`
  $i := i + 1$
`end`

training dataset $\mathcal{D}_{\text{tr}}$ using a distance function $d(\mathcal{D}^*, \mathcal{D}_{\text{tr}})$. If the distance is less than a specified tolerance ϵ then $\boldsymbol{\theta}^*$ is accepted.

Thus, by running an ABC algorithm one obtains a sample of parameters from the distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}, d(\mathcal{D}^*, \mathcal{D}_{\text{tr}}) \leq \epsilon)$. Given a suitably small value of ϵ the distribution of samples $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}, d(\mathcal{D}^*, \mathcal{D}_{\text{tr}}) \leq \epsilon)$ will be a good approximation to the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}})$. By altering the value of ϵ we control how close the sample distribution is to the true data distribution. Reducing ϵ will result in the sample distribution becoming closer to the true data distribution but it becomes more difficult to locate acceptable samples. Conversely, increasing ϵ will result in more samples being accepted but at the cost of the sample distribution diverging from the true distribution.

Instead of performing direct Euclidean distance comparison between the training and simulated datasets, $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}^*$, one can use knowledge of the problem being modelled to select suitable summary statistics that capture the desired characteristics of the data, e.g. for comparing the mean and standard deviations of the real and simulated datasets. The distance calculation is then carried out in terms of these summary statistics.

ABC MCMC implementations exist for the commonly used rejection, Gibbs and MCMC sampling algorithms (see chapter 2.1.2 for a discussion of these sampling methods). For a general introduction to sampling methods see [MacKay, 1998]. In this chapter we are interested in the ABC implementation of the MCMC algorithm which is presented below.

ABC MCMC Algorithms

Metropolis-Hastings enables one to produce samples from a distribution using the outputs from a Markov chain. Whilst rejection sampling compares the probability of each individual sample with an acceptance threshold, in the Metropolis-Hastings algorithm the

probability of the newly sampled parametrisation is compared with that of the previously accepted parametrisation. As a result, Markov chain Monte Carlo (MCMC) approaches tend to accept samples more frequently than in rejection sampling, but with dependencies between subsequent samples.

Since the Bayes factor cannot be evaluated for hard classifiers, one can instead make use of Approximate Bayesian Computation to approximate this. The ABC MCMC algorithm proposes a new parametrisation, $\boldsymbol{\theta}^*$, based on the previous sample, $\boldsymbol{\theta}_i$, by drawing a sample from the proposal distribution $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$. Using this new model parametrisation a dataset, $\mathcal{D}^*$, is simulated using the model $f(\mathbf{x}|\boldsymbol{\theta}^*)$ and compared with the training data, $\mathcal{D}_{\text{tr}}$. If the two datasets are similar enough, the new sample is accepted with probability ρ where ρ is:

$$\rho = \min\left(1, \frac{p(\boldsymbol{\theta}^*)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)}\right) \tag{5.3}$$

Should the proposal distribution be symmetric, i.e. $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i) = q(\boldsymbol{\theta}_i|\boldsymbol{\theta}^*)$, then ρ depends only on the prior, $p(\rho)$.

The result of this sampling is a Markov chain with the distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}, d(\mathcal{D}^*|\mathcal{D}_{\text{tr}}) \leq \epsilon)$ [Marjoram et al., 2003]. As with other MCMC methods, there is the possibility of the ABC MCMC algorithm getting stuck in regions of low probability for long times and there being a high correlation between samples [Gilks et al., 1996; Toni et al., 2009]. Therefore one need ensure the sampling process is run for a sufficiently large number of iterations, and thus long sample chains are obtained [Toni et al., 2009].

The ABC MCMC algorithm is presented in algorithm 6. At each iteration the algorithm samples a potential set of parameters, $\boldsymbol{\theta}^*$, drawn from the proposal distribution (line 3). This newly sampled set of parameters is used to generate the simulated dataset $\mathcal{D}^*$ (line 4). The simulated dataset is then compared with the training data $\mathcal{D}_{\text{tr}}$ using the distance function, $d(\cdot, \cdot)$, and some supplied error tolerance threshold, ϵ. Should $d(\mathcal{D}_{\text{tr}}, \mathcal{D}^*) \leq \epsilon$ then $\boldsymbol{\theta}^*$ is accepted with probability ρ (lines 5 - 8); otherwise the previous sample is kept (lines 9 - 13). As discussed previously, it may be preferable to perform the distance calculation by comparing summary statistics capturing the characteristics of $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}^*$ rather than calculating the Euclidean distance between the two datasets.

When setting the value of ϵ some care needs to be taken. Should the chosen value of ϵ be too small then the ABC MCMC algorithm may not be able to find a viable model parametrisation satisfying the value of ϵ and thus fail to sample any models. This failure to sample may also occur as a result of mismatch between the data generating model and the model whose parameters are being sampled, necessitating a larger value of ϵ. Additionally, requiring the initial parameter perturbations to satisfy the constraint introduced by ϵ may be difficult leading to an initial lack of samples. This may, however, be ameliorated during the burn-in phase during which a larger, slacker, initial ϵ value can be used to get sampling started and then slowly reduced to the desired ϵ value to facilitate sampling from the desired distribution. This approach is used in the experiments presented here.

In this chapter we propose an ABC MCMC algorithm for sampling parametrisations for hard classifiers simulating new targets, $\mathcal{D}_n^* = \{\mathbf{x}_n, \mathbf{y}_n^*\}$, using AUC as the summary statistic in the distance function. By averaging over the hard classifications produced using the sampled model parametrisations, we obtain a soft classification in the form of

an estimate of the posterior probability of class membership for a datapoint. In the next section we outline the idea behind this AUC based ABC MCMC algorithm and discuss how best to determine the value of ϵ used for sampling hard classifier parametrisations when using AUC as the summary statistic.

5.3. AUC error as a distance function

Receiver Operating Characteristic (ROC) curves are a commonly used method for visualising performance of binary classifiers over the range of misclassification costs [Fawcett, 2006]. By using the area under the ROC curve (AUC), calculated here using the Mann-Whitney-Wilcoxon statistic, one is able to measure how well a classifier discriminates between the two classes in binary classification [Hanley and McNeil, 1982; Hand and Till, 2001]. Under this measure, a classifier giving perfect class separation is one that completely dominates the ROC curve space with a true positive rate of 1 and false positive rate of 0 yielding an AUC value of 1. In the limit of a large dataset, the maximum AUC for a classification problem is that obtained by the Bayes classifier, AUC_{Bayes}. This is because the Bayes classifier has full knowledge of the underlying data distribution and thus gives the true probabilities of class membership. Since we want to sample classifier parametrisations from the posterior distribution and thus obtain the best possible class separation, it was decided to use AUC as the summary statistic in the ABC MCMC distance function.

Using knowledge of the distributions from which a dataset was generated, one can sample from them and generate the Bayes optimal ROC curve and calculate its AUC. This might then be compared with the AUC of a model sampled by the ABC MCMC algorithm. The distance comparison against ϵ is then:

$$d(\mathcal{D}_{\text{tr}}, \mathcal{D}^*) = d(\mathbf{t}, \mathbf{y}^*) = AUC_{Bayes} - AUC(\mathbf{t}, \mathbf{y}^*) \leq \epsilon \tag{5.4}$$

where $\mathbf{t}$ is the vector of targets and $\mathbf{y}^*$ the model prediction using the newly sampled parametrisation, $\boldsymbol{\theta}^*$. In the limit of a large dataset $AUC_{Bayes} \geq AUC(\mathbf{t}, \mathbf{y}^*)$, however with finite datasets taking the absolute value of the difference ensures that $d(\mathbf{t}, \mathbf{y}^*) \geq 0$.

Unfortunately, knowledge of the underlying distributions is rarely, if ever, available and so the AUC for the Bayes optimal ROC curve cannot be calculated. Therefore, instead of equation 5.4 we define a measure of the proximity of $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}^*$ as:

$$d(\mathcal{D}_{\text{tr}}, \mathcal{D}^*) = 1 - AUC(\mathbf{t}, \mathbf{y}^*) = c + (AUC_{Bayes} - AUC(\mathbf{t}, \mathbf{y}^*)) \tag{5.5}$$

where c is some unknown constant. We refer to the value $1 - AUC(\mathbf{t}, \mathbf{y}^*)$ as the "AUC error".

Of course the AUC_{Bayes} is not available for real problems. However, in the limit of a large amount of data, $\mathcal{D}_{\text{tr}}$, we may write:

$$1 - AUC(\mathbf{t}, \mathbf{y}^*) = 1 + AUC_{Bayes} - AUC(\mathbf{t}, \mathbf{y}^*) - AUC_{Bayes} \tag{5.6}$$

$$= 1 - AUC_{Bayes} + d(\mathbf{t}, \mathbf{y}^*) \tag{5.7}$$

$$= c + d(\mathbf{t}, \mathbf{y}^*) \tag{5.8}$$

where $c = 1 - AUC_{Bayes}$ is unknown.

We therefore measure the proximity of the training data to the simulated targets $\mathbf{y}^*$ by:

$$\hat{d}(\mathbf{t}, \mathbf{y}^*) = 1 - AUC(\mathbf{t}, \mathbf{y}^*) \geq 0. \tag{5.9}$$

Note however, that ϵ, against which $\hat{d}(\mathbf{t}, \mathbf{y}^*)$ is compared, should never be set to less than approximately $c = 1 - AUC_{Bayes}$ because this represents optimal classification. We recognise however that with finite training data it is possible that $AUC(\mathbf{t}, \mathbf{y}^*) > AUC_{Bayes}$. In practice this is not an impediment because ϵ is always set greater than zero in order to allow the MCMC to explore.

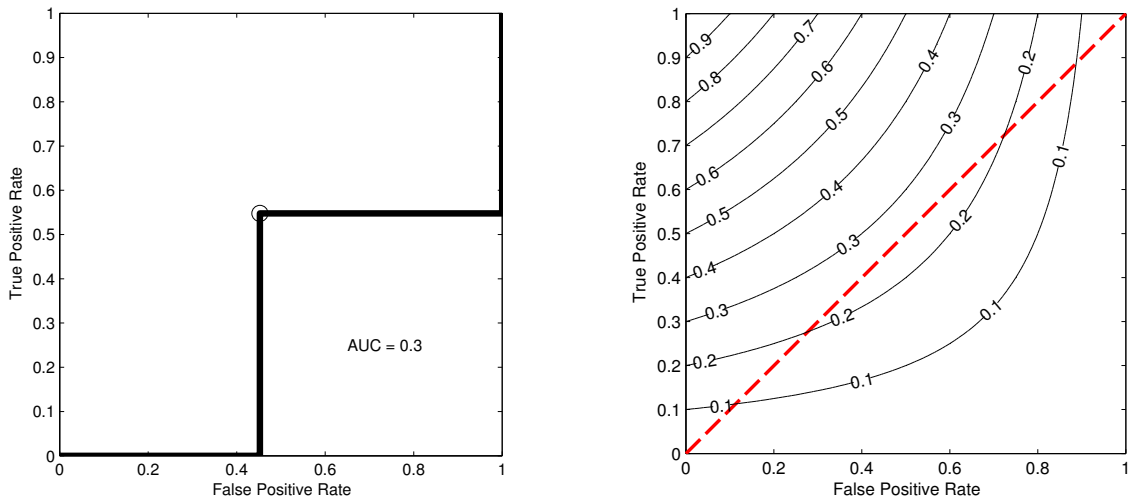
Since we intend to use the ABC MCMC algorithm to sample parameters for hard classifiers and the distance function uses AUC error, care is required in setting the value of ϵ. This is discussed in the following section.

Selecting ϵ

For most classification problems of interest there is overlap between the classes and thus Bayes classifiers with perfect knowledge of the underlying class distributions, giving the best possible class separation, are unable to produce perfect class separation. Furthermore, since many models produce hard classifications rather than soft ones in the form of scores or posterior probabilities, they will only dominate a relatively small area of the ROC graph. As a result, area under the curve measures for such hard classifiers are often surprisingly small.

In figure 5.1a the ROC curve for a hard classifier is plotted. Hard classifiers produce an ROC curve comprising only a *single* operating point (in addition to the trivial (0,0) and (1,1) end-points), unlike soft classifiers whose predictions can be used to produce a complete ROC curve by varying the decision threshold. Thus, hard classifiers only dominate a small area of the ROC space, in this case $AUC = 0.3$ for the example in figure 5.1a. Figure 5.1b shows a contour plot of the AUC values obtained by hard classifiers. Each contour shows the locus of operating points for classifiers with a particular AUC. In particular, attention should be drawn to the fact that from figure 5.1b we can see that hard classifiers whose AUC is only 0.25 lie to the northwest of the diagonal line where true positive rate equals false positive rate i.e. in the better performance than random half of the ROC graph. Conventionally, one looks to obtain classifiers whose ROC curves dominate more that 50% of the ROC curve space in order to obtain performance better than random. However for hard classifiers, such as those investigated in this chapter, AUC target constraints can be relaxed to surprisingly low values whilst still obtaining better than random performance.

Care is therefore needed when deciding the value of ϵ against which $1 - AUC(\mathbf{t}, \mathbf{y}^*)$ is compared. Indeed, for a hard classifier to reproduce the AUC value that would be obtained by the Bayes optimal ROC curve, which has a *continuous* output and therefore produces a full ROC curve rather than single operating point, would require the generation of a model parametrisation whose ROC operating point *dominates* the Bayes ROC curve. Clearly, this is undesirable as such models are unlikely to be sampled and accepted into the sample chain due to being unlikely to obtain such performance levels.



(a) (b)

Figure 5.1.: Left: ROC curve with single operating point for a hard classifier obtaining AUC of 0.3. Right:Contours of Area Under Curve achieved by hard classifiers using Mann-Whitney statistic

5.4. Illustration

In this section we demonstrate the efficacy of the ABC MCMC algorithm by sampling parameters for hard multi-layer perceptrons (MLP) classifying the Ripley synthetic dataset [Ripley, 1994]. Although the MLP is capable of producing soft classifications, here we use it as a hard classifier. Since the MLP contains a large number of connection strengths and associated bias values whose values need setting it is ideal for testing the efficacy of the ABC MCMC algorithm in sampling appropriate model parameters. In the experiments carried out here, the ABC MCMC algorithm treats the MLP as a black-box classifier which performs hard binary classification based on a particular set of parameters, $\boldsymbol{\theta}$.

We begin with a description of how binary classification is carried out using the multi-layer perceptron in section 5.4.1, followed by details of the priors used for sampling its parameters in section 5.4.2. The ABC MCMC algorithm using AUC error is then outlined in section 5.4.3 and results on the Ripley synthetic dataset presented in section 5.4.4.

5.4.1. MLP

Artificial Neural Networks [ANNs, e.g., Bishop, 1995], are a machine learning tool whose development was inspired by the structure and interconnection of neurons in the brain. In ANNs layers of artificial neurons are created whose interconnection strengths are learnt during the training process. Such networks are capable of learning non-linear relationships between inputs and their targets and have been widely used for both regression and classification problems.

In this chapter we make hard binary classifications using the multi-layer perceptron (MLP), an ANN constructed from multiple layers of fully connected artificial neurons in a directed graph. Each of the nodes in the MLP, excluding the input nodes, uses a non-linear activation function allowing the MLP to separate data that that cannot be separated linearly. The layers of neurons between the input and output neurons are often referred

to as the "hidden layer" and the neurons within it as "hidden units". Here, a single layer of hidden units is used.

For a sample, $\mathbf{x}$, in an M dimensional dataset, the input values to the MLP with H hidden units are denoted by x_m where $m = 1, \ldots, M$. The first layer of the MLP forms H linear combinations of the inputs to give the activations, $a_h^{(1)}$, of each hidden unit as:

$$a_h^{(1)} = \sum_{m=1}^{M} w_{hm}^{(1)} x_m + b_m^{(1)} \qquad m = 1, \ldots, M \tag{5.10}$$

where each $w_{hm}^{(1)}$ is a weight in the first layer of the MLP and $b_m^{(1)}$ are the bias values associated with each of the hidden units. The activations, $a_h^{(1)}$, are then transformed using a non-linear sigmoidal activation function. Typically, this is the tanh function. Thus, the outputs of the hidden units are:

$$z_h = \tanh(a_h^{(1)}) \qquad h = 1, \ldots, H \tag{5.11}$$

The z_h are then transformed by a second layer of weights and biases to give the second-layer (often the output layer) activation value $a^{(2)}$ as:

$$a^{(2)} = \sum_{h=1}^{H} w_h^{(2)} z_h + b^{(2)} \tag{5.12}$$

Classifications are carried out by examining the activation of $a^{(2)}$ and seeing if whether it is greater than 0. If the activation is greater than 0 then $\mathbf{x}$ is assigned to class 1. Otherwise it is assigned to class 1. Thus:

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } a^{(2)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

It is also possible to obtain a soft classification and estimate for the posterior probability of class membership by passing the output activation through the logistic link function.

The most commonly used method for learning the values for the weights and biases in an MLP is error back-propagation [Rumelhard et al., 1986]. Error back-propagation seeks to maximise the likelihood of the parametrisation by feeding the MLP prediction errors back through the network structure, using the gradient of the error to update the values of the weights. For a full description see [Rumelhard et al., 1986; Bishop, 1995] and for an implementation see [Nabney, 2002]. Other methods used to learn MLP parametrisations include multi-objective optimisation [Jin et al., 2004b; Fieldsend and Singh, 2005] and MCMC sampling [Lampinen and Vehtari, 2001].

5.4.2. Priors

In this chapter we use this ABC MCMC algorithm to sample sets of parameters, $\boldsymbol{\theta}$, for multi-layer perceptrons performing hard binary classification. These parametrisations consist of the values of the weights and biases for the MLP. The structure of the network, i.e. number of nodes, remains fixed.

Algorithm 7 ABC MCMC algorithm using AUC

Require: $\mathcal{D}_{\text{tr}}$ *Training data feature-target pairs*
ϵ *Error threshold*
samples *Number of samples to generate*

$i := 0$ $\quad \boldsymbol{\theta}_i :=$ `initialise`() *Initialise* $\boldsymbol{\theta}_i$
`while` $i <$ samples
  $\boldsymbol{\theta}^* \sim q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$ *Propose* $\boldsymbol{\theta}^*$ *according to* $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$
  $\mathbf{y}^* := f(\mathcal{D}_{\text{tr}}|\boldsymbol{\theta}^*)$ *Predict the targets using* $\boldsymbol{\theta}^*$
  `if` $1 - AUC(\mathbf{t}, \mathbf{y}^*)) \leq \epsilon$
    $\rho =$ `min` $\left(1, \frac{p(\boldsymbol{\theta}^*)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)}\right)$
    `if rand` $\leq \rho$
      $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}^*$
    `else`
      $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}_i$
    `end`
  `else`
    $\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}_i$
  `end`
  $i := i + 1$
`end`

As with all supervised learning tasks, overfitting is a concern. A common method used to avoid overfitting and produce models that generalise well is to produce sparse models by learning weightings for which many of the values are 0. In the Relevance Vector Machine (RVM) [Tipping, 2001, 2000], used earlier in this thesis in chapters 3 and 4, sparse weightings are learnt by placing Student-t priors over each of the model weights. Here we follow a similar methodology for the weights in the MLP.

To encourage weight values to remain small we place a Student-t prior over each of the weights in the sampled parametrisations as:

$$p(w_m) = St(w_m|\mu, \alpha, \nu) = \frac{\Gamma(\nu/2 + 1/2)}{\Gamma(\nu/2)} \left(\frac{\alpha}{\pi\nu}\right)^{1/2} \left[1 + \frac{\alpha(w_m - \mu)^2}{\nu}\right]^{-\nu/2-1/2} \tag{5.14}$$

where ν is the degrees of freedom of the distribution and α its precision. As we desire the weight values to be small and tend to 0, so we set $\mu = 0$, and $\nu = 3$. Whereas in the RVM different precisions are learned during training and assigned to each of the weights, here we set $\alpha = 0.05$ for all w_m. This is designed to encourage the ABC MCMC algorithm to sample smaller magnitude weight values, providing some regularisation of the MLP.

Unlike the weights, we have no prior belief about sparsity with regards to the bias values. Since we have no preference for the values of the biases, a uniform prior is placed over them.

5.4.3. AUC error ABC MCMC Algorithm Implementation

Here, we present the AUC error ABC MCMC algorithm used to sample the parametrisations for hard binary MLP classifiers in algorithm 7.

The ABC MCMC algorithm begins by taking in the training data feature-target pairs,

$\mathcal{D}_{\text{tr}}$, the desired value of ϵ, and the number of samples to generate. Sampling is then initialised using either a sample from the prior distribution or using a provided initial parametrisation (line 1).

Using the proposal distribution, a new MLP parametrisation, $\boldsymbol{\theta}^*$, is sampled (line 3). This parametrisation is then used by the MLP to make class predictions, $\mathbf{y}^*$, for each of the samples in $\mathcal{D}_{\text{tr}}$ (line 4). Based on the predictions, $\mathbf{y}^*$, and targets, $\mathbf{t}$, the AUC error is then calculated for this new parametrisation and compared with ϵ (line 5). If it is less than ϵ then the new parametrisation is accepted with probability ρ (lines 6 - 7). Otherwise, the previous sample is kept. This process of sampling and checking for acceptance is repeated until the desired number of samples have been obtained from the target distribution conditioned on ϵ.

For the proposal distribution, $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i)$, new parametrisations, $\boldsymbol{\theta}^*$, are sampled from a Gaussian distribution with standard deviation 0.05 (chosen by experiment) centred on the previously accepted sample, $\boldsymbol{\theta}_i$. Since this is a symmetric kernel, i.e. $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_i) = q(\boldsymbol{\theta}_i|\boldsymbol{\theta}^*)$, the acceptance probability, ρ, is only dependent on the prior.

Having obtained a set of model parametrisations, we can then use the MCMC chain to make predictions. Using the chain, we produce hard classifications for each of the S sampled model parametrisations for an input datapoint $\mathbf{x}'$. We then average over the hard predictions from each of the samples to obtain the prediction $\hat{y}$ as:

$$\hat{y} = \frac{1}{S}\sum_{i=1}^{S} f(\mathbf{x}'; \boldsymbol{\theta}_i) \tag{5.15}$$

Notice that $\hat{y}$ is no longer a hard prediction as long as $S > 1$ and *at least* 2 unique parametrisations exist in the S samples. Instead we now have an estimate of the posterior probability of $\mathbf{x}'$ belonging to the $t = 1$ class and can therefore produce full ROC curves. In the next section, we compare the ROC curve generated by an ABC MCMC chain sampling parameters for hard MLP classifiers with the Bayes optimal ROC curve on a synthetic dataset.

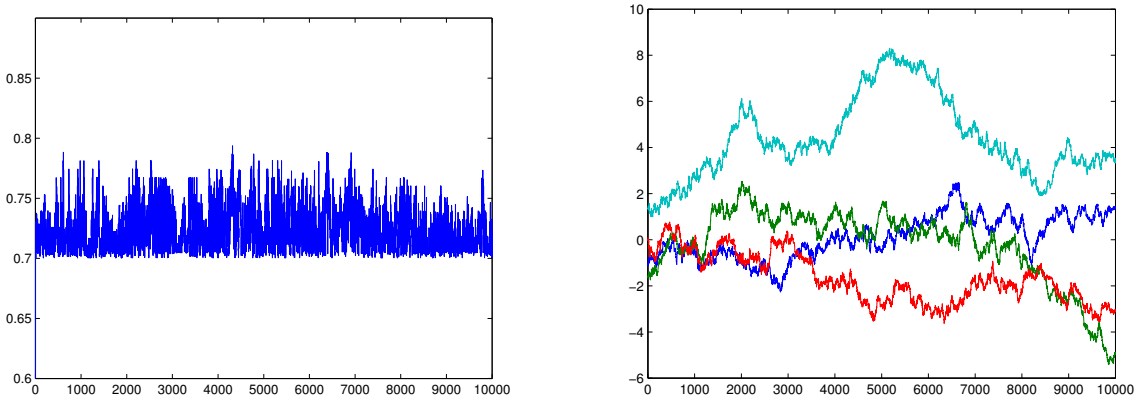
5.4.4. Results

In order to demonstrate the efficacy of the ABC MCMC algorithm the commonly used Ripley synthetic data [Ripley, 1994] is employed. This is a two-dimensional synthetic dataset consisting of two classes generated from Gaussian distributions of centres (-0.7, 0.3), (0.3, 0.3), (0.4, 0.7) and (-0.3, 0.7) each with covariance 0.03I. A total of 250 training examples and 1000 test examples are provided. Since the underlying distributions from which the dataset was created are known, one can calculate the performance of the Bayes classifier and thus generate the Bayes optimal ROC curve, enabling better analysis of how well the model parameters have been sampled from the posterior distribution. Here, the Bayes optimal ROC curve was generated from 25000 data points sampled from the Gaussian distributions used to construct the Ripley data giving an area under the curve of $AUC_{Bayes} = 0.9714$.

As discussed earlier, the distance function chosen was $1 - AUC(\mathbf{t}, \mathbf{y}^*)$. In the case of the Bayes optimal ROC curve the distance is 0.0286. However, as discussed in section 5.3,

it would be difficult for the sampled models to actually achieve this as, while the Bayes ROC curve is based on a *continuous* posterior probability, the sampled models only give *hard* predictions and thus have a considerably smaller area under their ROC curves.

Using the ABC MCMC algorithm outlined in section 5.4.3, the weights and biases, i.e. parameters, for a hard classification multi-layer perceptron (MLP) containing 7 hidden units were sampled. This number of hidden units is indicated by Ripley [1994], and confirmed through experimentation, to give optimal performance in an unregularised neural network on the Ripley synthetic dataset. A Student-t distribution was used for the weights prior, centred at 0 with precision $\alpha = 0.05$ and $\nu = 3$ degrees of freedom, chosen to encourage sparse models. A uniform distribution was placed over the bias values. Since a value of ϵ must be chosen this was selected to be 0.3. Sampling was initialised with a slacker ϵ value of 0.5 which was gradually tightened to $\epsilon = 0.3$ over 7 equally spaced values. At each of these values sampling proceeded until 100 parametrisations had successfully been sampled and accepted before moving on to the next value of ϵ. Once the desired value of $\epsilon = 0.3$ had been reached a further 10000 samples were taken. Figure 5.2a shows the AUC values obtained by each of the 10000 samples. From this we can see that the minimum AUC value obtained by the ABC chain on the training data is 0.70 corresponding to an AUC error value of 0.3. The maximum training AUC value obtained by a sample in the chain was 0.79, corresponding to an AUC error of 0.21 but this is less than the Bayes AUC. In figure 5.2b we see how the values of 4 of the weights from the input to hidden layer of the MLP change over successive samples. From this we can see there are relatively long term correlations with samples being taken from a wide range of parameter values. We note that the AUC samples (as shown in figure 5.2a) are apparently stationary even though the weights display some long term correlations. Of the 10000 samples obtained,



(a) (b)

Figure 5.2.: Left: AUC values of the 10000 sampled parametrisations for the hard MLP classifier on the Ripley datasets. Right: Weight values for each of the 10000 sampled parametrisations

the first 5000 were discarded as being part of the burn in phase and, of the remaining 5000 sampled parametrisations, every 7th sample was kept to give 715 model parametrisations.

Averaging over the hard predictions of the 715 MLP classifiers parametrised by the samples in the ABC MCMC chain, one obtains the soft prediction $\hat{\mathbf{y}}$ (see equation 5.15). From this soft prediction, one can use a range of decision thresholds to generate and

examine the training and test data ROC curves for the composite classifier. As seen in figure 5.3, the ROC curves generated by the averaged prediction $\hat{\mathbf{y}}$ show performance close to that of the Bayes optimal ROC curve with training and test AUC values of 0.937 and 0.965 respectively, close to $AUC_{Bayes} \approx 0.974$. In particular, on the larger test dataset, we see that the ROC curve of the ABC chain prediction almost coincides with the Bayes optimal ROC curve.

In addition to the Bayes optimal ROC curve we also plot the ROC curves for the Bayes optimal classifier prediction on the training and test datasets i.e. the ROC curves obtained by classifying the dataset using the true posterior probabilities of class membership. On the training data the Bayes classifier achieves an AUC of 0.953 and on the test data an AUC of 0.977; explaining how the AUC for the ABC chain prediction has improved between training and test datasets. Comparing the ROC curves on the training and test data for the ABC chain and Bayes classifier, we see that the ABC chain has produced ROC curves close to those of the Bayes classifier.

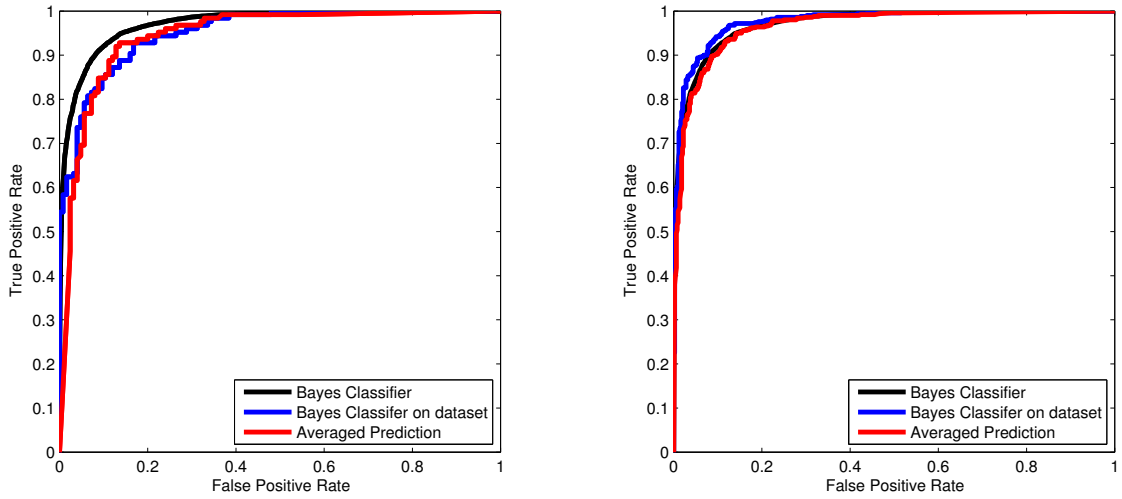
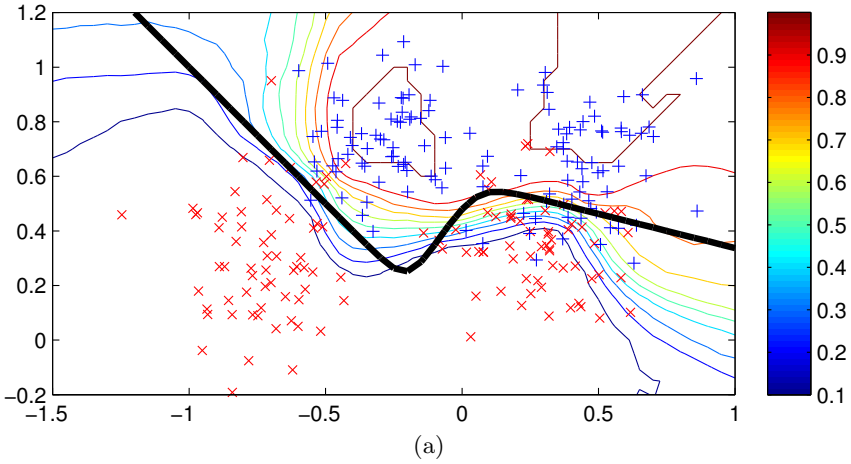


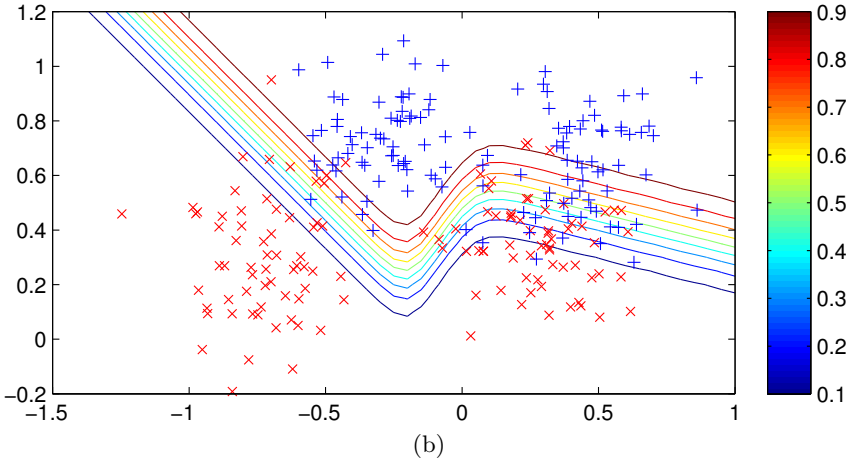
Figure 5.3.: Comparison of the ROC curve of the averaged prediction with the Bayes ROC curve. Left: Training ROC curves, Right: Test ROC curves

In figure 5.4a we plot probability contours for the soft, averaged, prediction, $\hat{\mathbf{y}}$, and compare them with the thick black line marking the Bayes class boundary i.e. where $p(t = 1|\mathbf{x}') = 0.5$. In figure 5.4b we see the contours for the true probabilities of class membership. Examining these, we see that in the denser areas of the training data the ABC chain prediction is close to that of the Bayes classifier; whereas, unsurprisingly, in the less dense areas, where there is less information about the data distribution, the true probability contours and chain prediction contours diverge.

From these results we can see that the ABC MCMC chain has successfully produced estimates of the posterior probabilities of class membership yielding ROC curve performance close to that of the Bayes optimal ROC curve. In the next section we investigate the possibility of pruning the number of classifiers with which these predictions are obtained whilst maintaining performance with the aim of reducing computation time and expense.



(a)



(b)

Figure 5.4.: Contour plots for posterior probabilities of class membership. Top: Coloured lines represent the probability contours for the averaged prediction, thick black line is the Bayes class boundary. Bottom: Coloured lines represent the probability contours for the true probability of class membership

5.5. Sparse Ensembles

As a result of the ABC MCMC sampling procedure, one is presented with a large number of sampled hard classifiers with which to make classifications. The most straightforward method of producing a soft, continuous, output from the sampled classifiers is to average over all of them, giving each model an equal contribution towards the final classification. Unfortunately, this means averaging over the predictions made by a large number of different classifiers and thus costly in terms of both time and computation potentially rendering this method impractical when rapid classification is required or available computational power is limited.

Ideally, one would like to select a subset of the sampled classifiers that performs as well as the averaged classification, $\hat{\mathbf{y}}$, in order to reduce time and computational costs. In this section methods for building sparse ensembles of classifiers from those classifiers sampled by the ABC MCMC algorithm are investigated.

While many methods exist for the generation of ensembles, such as Mixture of Experts [Jacobs et al., 1991], Bagging [Breiman, 1996] and Boosting [Schapire, 1990, 1999], here we are concerned with ensemble pruning. The set of classifiers sampled by the ABC

MCMC algorithm, whose predictions we have previously been averaging to produce $\hat{y}$, is a special ensemble which has been generated by sampling classifier parameters, $\boldsymbol{\theta}$, from the distribution $p(\boldsymbol{\theta}|\mathcal{D}_{\text{tr}}, d(\mathcal{D}^*, \mathcal{D}_{\text{tr}}) \leq \epsilon)$.

In ensemble pruning one seeks to reduce the size of the ensemble whilst maintaining performance. Pruning algorithms fall in two categories: selection-based and weight-based. In selection-based pruning algorithms the ensemble members are either selected or rejected, based on their performance. Typically no weighting of the individual members is used. The most straightforward method is to rank the ensemble members by performance and pick the M best [Chawla et al., 2004]. Other methods look at selecting ensemble members so as to maximise the pair-wise difference between the members [Margineantu and Dietterich, 1997]. The main problem with these methods, however, is they make use of greedy search without any theoretical or empirical guarantees of performance. For example, consider an ensemble created by selecting 3 classifiers each with 95% classification accuracy. There is no guarantee that this will perform better than an ensemble of 3 classifiers each of which has 65% accuracy.

As one might expect from the name, in weight-based ensemble pruning one looks to tune the weights associated with each ensemble member. For regression ensembles weights have been calculated to minimise the mean square error [Hashem, 1993; Zhou et al., 2002], though this approach has problems with real-world applications due to large numbers of ensemble members with similar performance. Additionally such methods are prone to overfitting and often fail to reduce the ensemble size. Since weight-based ensemble pruning can be regarded as a sparse Bayesian learning problem one can apply the relevance vector machine (RVM) [Tipping, 2001, 2000] to it [Chen, 2008]. The RVM makes use of automatic relevance determination (ARD) and thus prunes most of the members, producing a sparsely weighted ensemble. It is noted, however, that there are concerns about potential negative weights produced by the RVM weighting [Chen, 2008]. To address this, Chen et al. [2006]; Chen [2008] make use of a truncated Gaussian prior over the weights so as to ensure only positive weights are generated.

In the following sections methods for pruning the number of sampled classifiers are investigated and the performance of the pruned ensembles compared with that of the whole ABC chain. In the first, a selection-based method is investigated in which classifier AUC performance is used to select classifiers with which to make predictions. In the second section weight-based pruning is investigated using the 'fast' Relevance Vector Machine [fRVM, Tipping and Faul, 2003; Faul and Tipping, 2002] in both the regression and classification context. For regression, the fRVM is regressed against $\hat{\mathbf{y}}$, the prediction from the chain, and for classification the true class targets for the samples, $\mathbf{t}$, are used.

5.5.1. Performance-based selection

As an initial investigation into ensemble pruning, performance-based selection was used. Each of the classifiers sampled by the ABC chain had their individual training AUC calculated. Ensembles were built by selecting and equally weighting all the classifiers which obtained a training AUC greater than some threshold. As the AUC threshold was lowered, so the number of classifiers in the ensemble increased.

In figure 5.5 we see the effects on ensemble size and performance of varying the AUC

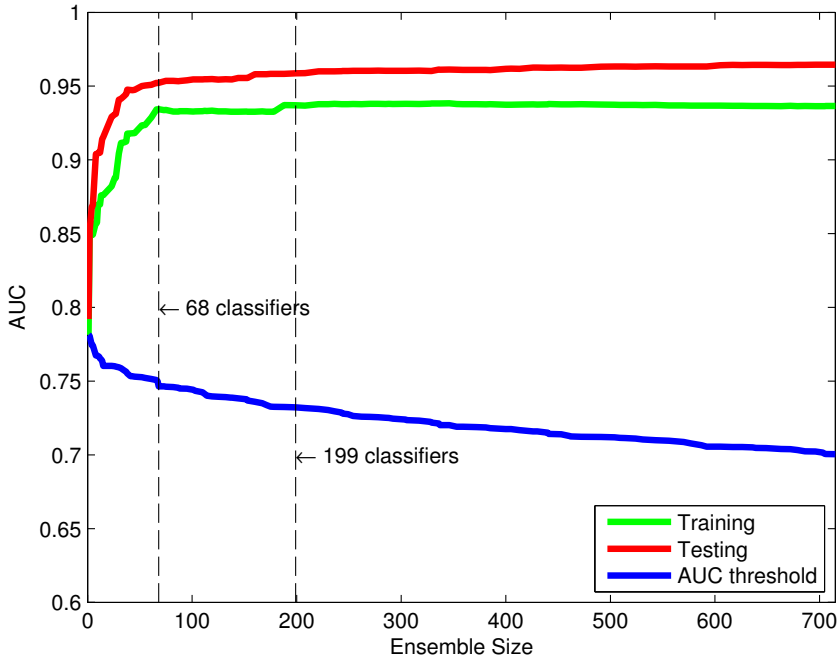


Figure 5.5.: Performance and size of ensembles constructed by equally weighting sampled classifiers selected based on their individual training AUC. Green line marks the training AUC of the ensemble. Red line marks the test AUC of the ensemble. Blue line shows the AUC performance threshold used to select the ensemble.

value above which sampled classifiers are included in the ensemble. Increasing the number of equally weighted classifiers in the ensemble leads to increased training and test data performance. While test data AUC performance continues increasing as the ensemble increases in size, performance appears to level off on the training data at around 200 included classifiers. There is also a region between 75 and 178 included classifiers where no significant improvement in training performance is seen despite increasing ensemble size.

Were one to use an ensemble constructed by equally weighting the top 68 performing classifiers, based on training data AUC, the ensemble would achieve a training AUC of 0.934 and test AUC of 0.952. Alternatively selection of the top 199 classifiers would lead to an ensemble performance of 0.937 on the training data and 0.959 on the test data. The ROC curves for these ensembles are plotted in figures 5.6 and 5.7 respectively.

In figure 5.6 we see the ROC curve generated by creating the ensemble from the top 68 sampled classifiers. On the training data there is some clear deviation from the ROC curve of the whole ABC MCMC chain but on the test data the pruned ensemble performance is closer to that of the ABC chain.

Similarly, in figure 5.7 the training and test ROC curves generated by creating the ensemble from the top 199 sampled classifiers are plotted. Again, there is some clear deviation from the ROC curve of the whole ABC MCMC chain on the training data while the ensemble's ROC curve is closer to that of the ABC chain on the test data.

Although this method appears promising it does suffer from a number of drawbacks. First, determination of the ideal size of pruned ensemble giving equivalent performance to that of the whole ABC chain is not automatic and, thus, manual selection of the

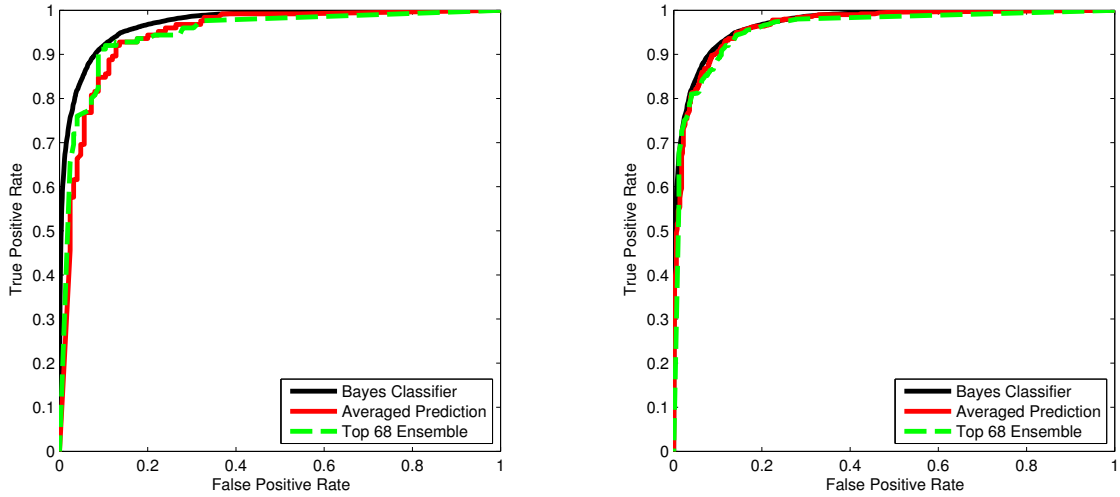


Figure 5.6.: Comparison of the ROC curve generated by selecting the 68 best performing classifiers with the ROC curves of the averaged prediction with the Bayes ROC curve. Left: Training ROC curves, Right: Test ROC curves.

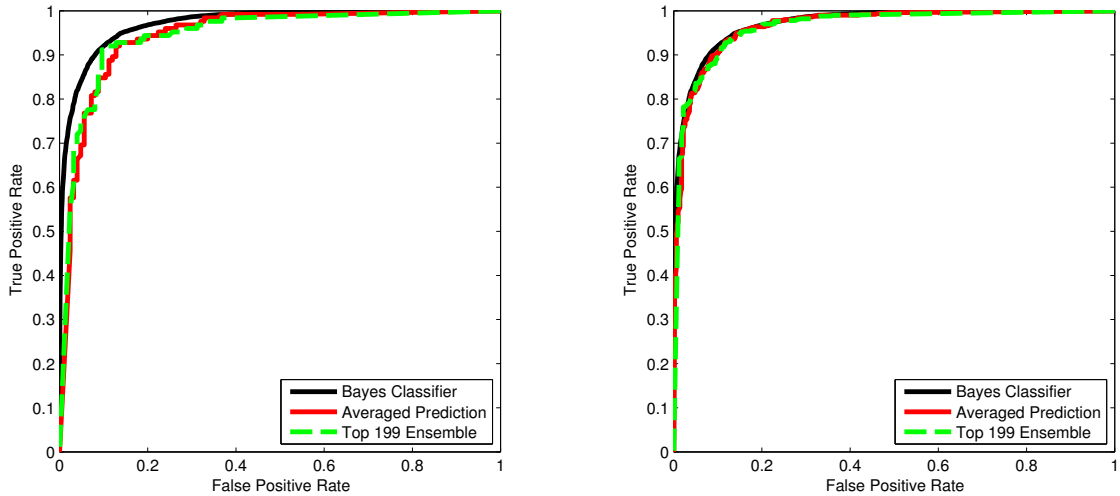


Figure 5.7.: Comparison of the ROC curve generated by selecting the 199 best performing classifiers with the ROC curves of the averaged prediction with the Bayes ROC curve. Left: Training ROC curves, Right: Test ROC curves.

ensemble size is required. Indeed, performance based selection here has not been carried out with a view to reproducing the performance of the ABC chain itself. Second, increasing the ensemble size by lowering the inclusion threshold does not guarantee an increase in performance – though it does here for the test data. Finally, as noted by Chen [2008], there is no guaranteed ensemble performance by pruning the ensemble in this way. Consider the situation where overfitting has occurred. A large number of classifiers may perform particularly well on the training data but an ensemble constructed from them could fail to perform well on the test data. In that situation it would be desirable to include some of the apparently poorer performing classifiers in the ensemble to potentially obtain better generalisation performance.

5.5.2. fRVM based weighting

Using Automatic Relevance Determination, the fast RVM [Tipping and Faul, 2003; Faul and Tipping, 2002], learns sparse probabilistic models for pattern recognition tasks. Typically, the fRVM is given a set of basis functions, ϕ_m, from which select a sparse weighting. Here, we treat the classifiers sampled by the ABC chain as basis functions for selection by the fRVM. Therefore, the RVM should select a sparse ensemble of classifiers from those parametrised by the ABC MCMC chain, each with their own associated weight towards the final classification.

Two approaches to selecting a sparse ensemble with the fRVM are taken here. In the first, the model selection is treated as a regression problem, with the fRVM being used to find a sparse set of classifiers whose weighted combination reproduces $\hat{\mathbf{y}}$, the averaged prediction of the classifiers sampled by the MCMC chain. In the second approach, the model selection is treated as a classification problem, using the fRVM to learn a sparse weighting of the sampled classifiers whose weighted combination reproduces the true targets, $\mathbf{t}$, for the data.

For regression against $\hat{\mathbf{y}}$, the fRVM selected 193 of the 715 available classifiers, 700 of which give unique sets of predictions on the training data. Figure 5.9 shows the weights assigned to each of the sampled classifiers by the fRVM. Of the 193 non-zero weights 176 are positive and 17 are negative. All 193 weight values are unique. It is not unexpected that the majority of the non-zero weights should be positive since the sampled classifiers perform better than random on the training data. Indeed, it seems desirable to have positive weights as the MCMC averaging yielding $\hat{\mathbf{y}}$ weights all the classifiers positively. One might postulate that the small number of negatively weighted classifiers *may* have been so weighted in order to reduce the strength of classification assigned to some of the points in the dataset. Since it is undesirable to have large numbers of weights that almost cancel each other out (e.g. $+10.23$ and -10.31), which is numerically unstable, it is reassuring that only a small number of these weightings are negative. This weighted combination of the sampled classifiers results in a training ROC curve AUC of 0.937 and test AUC of 0.964, plotted in figure 5.8. As can be seen, the ROC curves generated by this sparse weighting of the sampled classifiers lie very close to the curves produced by the ABC MCMC chain averaged prediction.

In order to determine if the fRVM has truly selected and weighted a good subset of the sampled classifiers and the selected ensemble was performing better than a randomly generated one, comparisons of the fRVM ensemble prediction and the averaged predictions of 50 groups of 193 classifiers selected uniformly at random from the ABC MCMC chain (i.e. the same number of classifiers as selected by the fRVM) were compared to the prediction obtained from the ABC MCMC chain. As shown in figure 5.10, the fRVM ensemble predictions are well correlated with those of the ABC MCMC chain for both the training and test datasets and compare well to those obtained from the randomly selected subsets.

Training the fRVM to carry out classification using the true targets, resulted in 101 of the sampled classifiers being selected. The resulting training and testing ROC curves have AUCs of 0.975 and 0.889 respectively. Given the Bayes classifier achieves an AUC of 0.971 and the fRVM training-test AUC discrepancy, the fRVM appears to have overfit

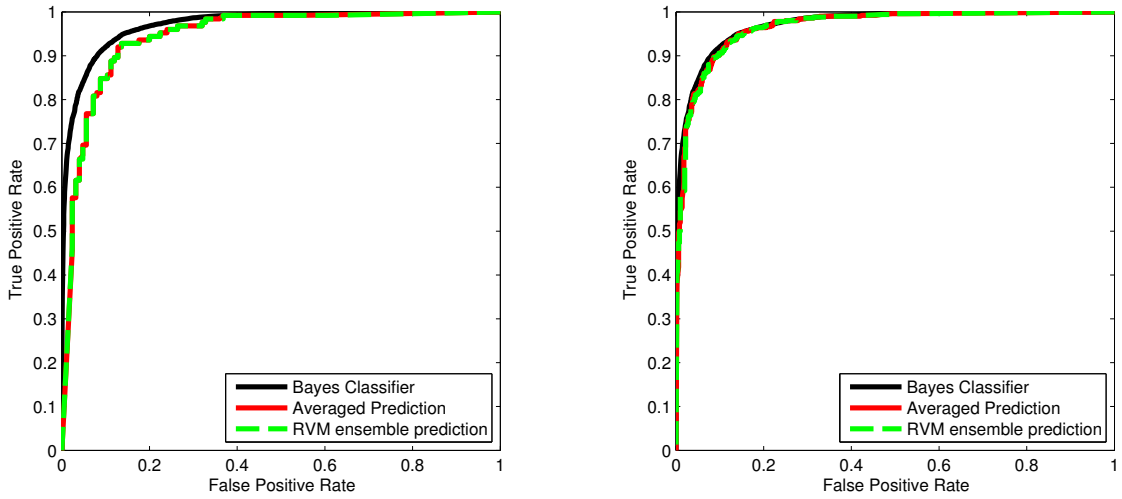


Figure 5.8.: ROC curve comparisons of the classifier subset selected by the RVM regressor with the averaged prediction and Bayes ROC curve. Left: Training ROC curves, Right: Test ROC curves.

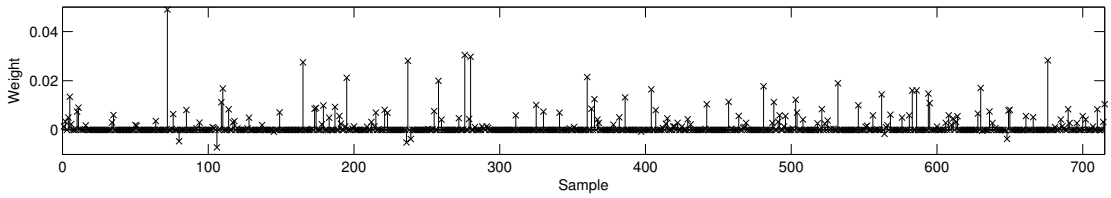


Figure 5.9.: Sample weights assigned by using the fRVM to regress against $\hat{\mathbf{y}}$.

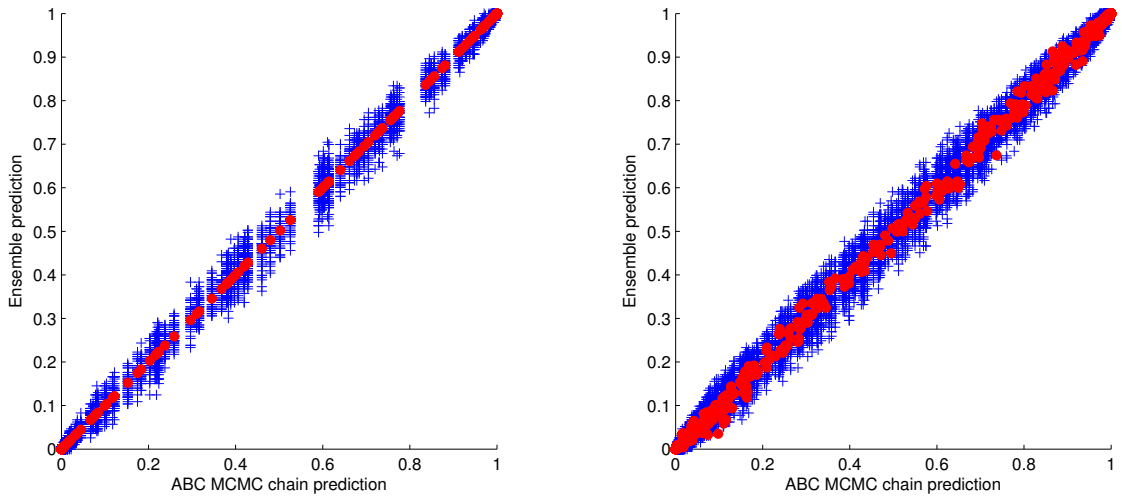


Figure 5.10.: Comparison of predictions of classifier subsets with the prediction obtained by the ABC MCMC chain. Red filled o's denote the predictions obtained by the fRVM using a weighted combination of 193 sampled classifiers. Blue +'s denote predictions obtained taking 50 sets of 193 randomly selected classifiers and averaging their predictions. Left: Training data predictions, Right: Test data predictions.

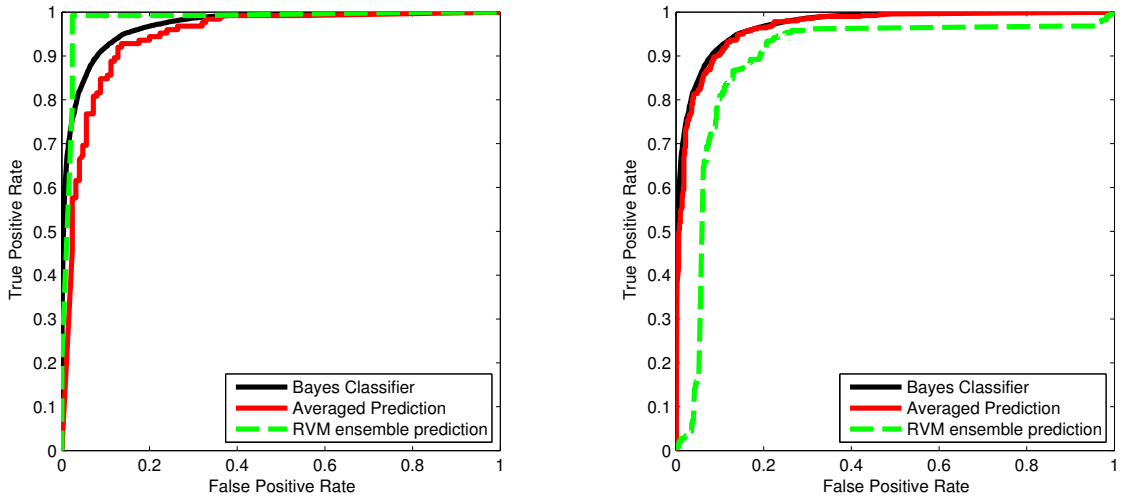


Figure 5.11.: ROC curve comparisons of the classifier subset selected by the RVM classifier with the averaged prediction and Bayes ROC curve. Left: Training ROC curves, Right: Test ROC curves.

here. Indeed, by examining the plots of these ROC curves in figure 5.11 one can quite clearly see that overfitting has occurred. On the training data the fRVM selected ensemble produces an ROC curve that dominates that of the Bayes optimal classifier and has thus likely overfit to the training data. This is further evidenced by its test data ROC curve which has moved behind the Bayes optimal classifier ROC curve. More importantly, perhaps, this test ROC curve for the fRVM selected ensemble does not lie close to that of the original ABC chain ROC curve and therefore fails to reproduce its performance. To prevent this overfitting one might use the k-fold MOEA presented in chapter 3 to control the complexity of the RVM classifier.

5.5.3. Summary

In this section three different methods for pruning the ensemble were investigated. In the first method, performance based selection was used to create ensembles of equally weighted classifiers. While the method is capable of yielding good ensemble performance, it is not an automated process and requires manual selection of the preferred ensemble. Additionally, there are no guarantees of generalisation capability of the pruned ensemble.

The second and third ensemble pruning methods both made use of the fRVM to select a sparse weighting of the sampled classifiers. Experiments using the fRVM in classification mode to select a sparse weighting that reproduced the true data targets indicated that overfitting had occurred and failed to reproduce performance equivalent to that of the ABC chain. Experiments using the fRVM to carry out regression to the chain prediction, $\hat{\mathbf{y}}$, were more promising. The fRVM successfully reduced the ensemble size from 715 equally weighted classifiers to 193 individually weighted classifiers. This pruned ensemble closely reproduced the performance of the ABC MCMC chain.

Of the methods investigated, use of the fRVM to carry out regression against the ABC chain prediction was the most promising. This provides a sparse weighted ensemble of classifiers whose size is automatically determined without need for user interaction and which appears to closely reproduce the ROC and AUC performance obtained from ABC

chain prediction.

5.6. Illustration: Waveform Data

Having investigated ABC MCMC algorithm performance on a synthetic dataset in section 5.4 and the possibility of ensemble pruning in section 5.5, in this section experiments are carried out on the Waveform dataset [Rätsch et al., 2001]. The Waveform dataset is a standard benchmarking dataset for which 100 partitionings into 400 training and 4600 testing data samples are provided. Each sample in the dataset has 21 features. This presents a higher dimensional problem to be solved than the Ripley data, which is only two dimensional. Unlike the Ripley dataset used in the previous experiments, the underlying distributions from which the data was generated are not known and, thus, the Bayes optimal ROC curve is not known. Therefore, performance of the ABC MCMC chain was compared with that of an unregularised MLP trained using a cross entropy error function whose output layer activation is passed through the logistic sigmoidal link function as:

$$y(\mathbf{x}) = \frac{1}{1 + \exp(-a^{(2)}(\mathbf{x}))} \tag{5.16}$$

to obtain a soft classification $y(\mathbf{x})$ which approximates $p(t = 1|\mathbf{x})$.

In order to determine the number of hidden units to use in the MLP, a standard, unregularised, MLP was trained using each of the 100 training-test partitions provided for the Waveform dataset and AUC values for the training data recorded. From these it was decided to sample parameters for a hard classification MLP consisting of 10 hidden units. This requires the setting of 220 weight values and 11 bias unit values, giving a total of 231 parameter values for the ABC MCMC algorithm to sample.

Using the ABC MCMC algorithm outlined in section 5.3, weights and biases were sampled for the MLP using the first training-test split of the the waveform data. As before, a Student-t prior was placed over the weights and a uniform distribution placed over the bias values. The same proposal distribution as before, a Gaussian distribution, was used. The algorithm was initialised with a desired ϵ value of 0.15 chosen so as to not force overfitting and allow for some variation in the model parameters sampled. In total, 100000 samples were generated by the ABC MCMC algorithm. Again, the potentially longer than anticipated time for burning-in when sampling parameters for an MLP [Lampinen and Vehtari, 2001] was considered and thus the first 50000 samples were discarded. Of the remaining 50000 samples, every 100th sample was kept.

Using the retained samples, predictions were made for the training and test data and compared with those obtained from an MLP with 10 hidden units trained using back-propagation. On the training data the MLP obtains an AUC of 1 whilst the ABC MCMC chain obtains an AUC value of 0.974. On the test data the MLP AUC performance drops to 0.941 while the ABC chain achieves an AUC of 0.931. These training and test AUC values are consistent with what one would expect based on the previous experiments in chapter 3 in which the first 10 folds of the dataset were used. In those experiments, the fRVM achieved an average training AUC of 1 and test AUC of 0.94 – although it is is noted that the fRVM had overfit to the training data.

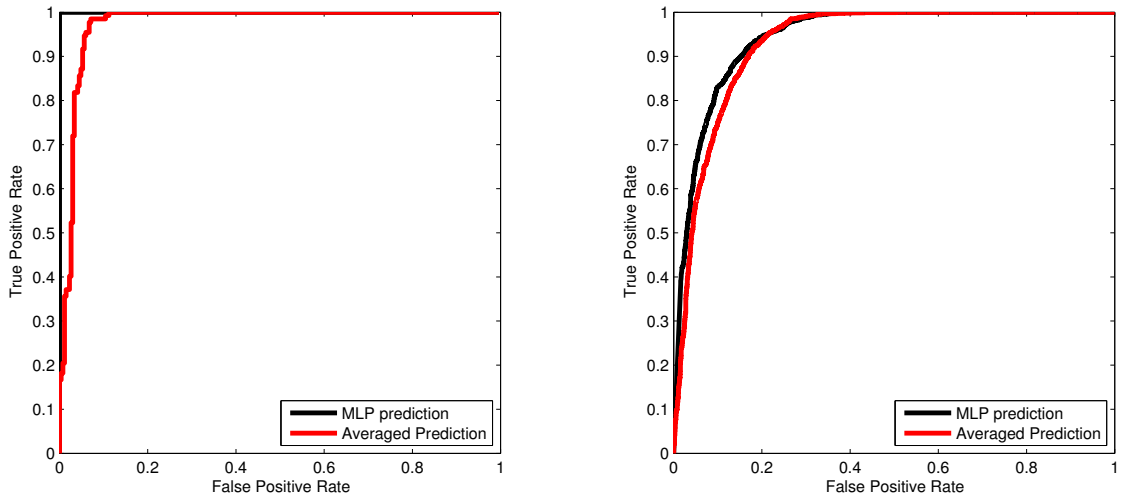


Figure 5.12.: Comparison of the ROC curve of the averaged prediction with the MLP ROC curve for the Waveform dataset. Left: Training ROC curves, Right: Test ROC curves

In figure 5.12 we see a comparison of the ROC curves for the ABC chain and back-propagation trained MLP. On the training data the MLP ROC curve is in the top left hand corner, as one would expect from the AUC of 1, while the ROC curve for the ABC MCMC chain lies behind it. On the test data the MLP ROC curve has moved away from the top left corner, as evidenced by its reduced test data AUC. The ROC curve for the ABC chain prediction lies close to that of the MLP. From these results it would appear that the unregularised MLP trained using back-propagation has overfit to the training data while the ABC MCMC chain has generalised well to new data.

As with the experiments with the Ripley synthetic data, the possibility of pruning this ensemble was explored. Of the pruning methods investigated in section 5.5, fRVM regression proved the most promising for reproducing the performance of the chain with a reduced ensemble size. Therefore, an fRVM was used to find a sparse weighting of the predictions of the sampled classifiers to regress against the averaged chain prediction, $\hat{\mathbf{y}}$. Of the 501 sampled classifiers the fRVM selected 255, reducing the ensemble size by nearly half. On the training data the fRVM selected ensemble yielded an AUC of 0.974 and on the test data an AUC of 0.929.

ROC curves were generated for the fRVM selected ensemble on the training and test data and compared with those for the trained MLP and averaged chain prediction. From figure 5.13 we can see that the sparse ensemble selected by the fRVM regression has once again closely reproduced the performance of the ABC chain. This further supports the idea of ensemble pruning via sparse regression.

5.7. Conclusion

Many classifiers provide hard classifications based on a single set of parameters whose values are uncertain. Ideally, one would make use of the Bayesian framework to ameliorate this parameter uncertainty by averaging over classifiers weighted by the posterior probability of their parametrisations. However, since hard classifiers do not supply a likelihood

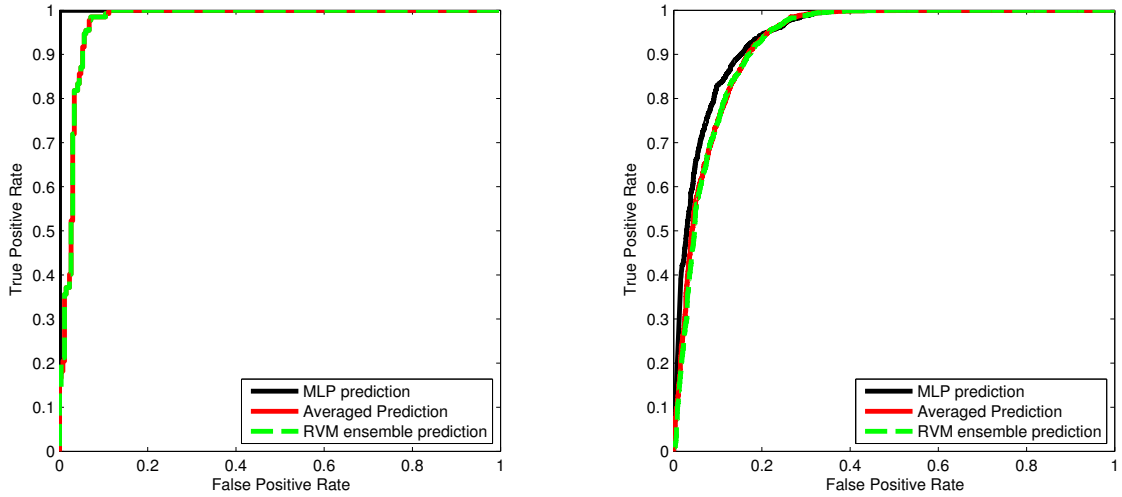


Figure 5.13.: ROC curve comparisons of the classifier subset selected by the RVM classifier with the averaged prediction and MLP ROC curve for the Waveform dataset. Left: Training ROC curves, Right: Test ROC curves

from which to calculate the posterior, the Bayesian framework cannot be used to calculate these posterior probabilities. Approximate Bayesian Computation methods allow one to circumvent this inability to evaluate the posterior probabilities of model parametrisations

An Approximate Bayesian Computation Markov chain Monte Carlo algorithm in which AUC was used as a summary statistic was used to sample parameters for a multi-layer perceptron giving hard classifications. By averaging the predictions of the classifiers sampled by the ABC MCMC chain one can obtain soft classifications from hard classifiers. An initial illustration of the potential of this method was carried out using the Ripley synthetic dataset. Since the Ripley data is synthetic and the underlying distributions from which it was generated are known, it was demonstrated that the prediction obtained from the ABC MCMC chain is capable of performance in line with that of the Bayes optimal ROC curve.

Given the large number of classifier parametrisations sampled by the ABC MCMC chain, it may become impractical to run them all when rapid classification is needed e.g. in a safety critical system. Therefore the possibility of reducing the number of classifiers whose predictions were to be averaged was investigated. By treating the training data predictions from the classifiers sampled by the ABC MCMC chain as kernels from which to select, one can train Relevance Vector Machines to select and weight a subset of classifiers. Training the fRVM to carry out regression against $\hat{\mathbf{y}}$, the averaged prediction of the sampled classifiers, one can produce ROC curves equivalent to those produced by the MCMC chain using markedly fewer classifiers. However, training an fRVM classifier to select a subset of classifiers resulted in overfitting and reduced test performance.

To corroborate the results found on the synthetic dataset, the ABC MCMC algorithm was applied to sampling the parameters for hard classification MLPs classifying the Waveform benchmark dataset and the chain performance compared with that of an MLP trained using the back-propagation method. ROC curve performance of the ABC chain was found to be close to that of the back-propagation trained MLP demonstrating its ability to work in higher dimensional sampling problems. Use of the fRVM to regress a sparsely weighting

of the sampled classifier predictions against the prediction of the whole chain once again demonstrated the ability to prune the ensemble whilst replicating whole chain ROC curve performance.

5.8. Future Work

Future work with the ABC MCMC algorithm would ideally involve using it to sample parameters for an existing industrial hard classifier. Of particular interest would be the NATS Short Term Conflict Alert (STCA) system. Using aircraft trajectory pairs the STCA system gives hard classifications as to whether or not the two aircraft will become dangerously close. While the structure of the classification model is fixed, its parameters are not, having been fine tuned over the years by air traffic control. Work by Reckhouse et al. [2010] focused on optimising ROC curves for the STCA system and, as a result, ROC curves are available against which novel STCA parametrisations can be compared. This is precisely the sort of system we envisage the ABC MCMC algorithm being useful for, and the existence of ROC curves with which to compare the ABC chain performance against makes it an ideal real-world test of the algorithm efficacy.

One of the questions raised in this chapter was how best to prune the ensemble of classifiers generated by the ABC chain. We showed that the fRVM could be used to produce a sparse weighting of the classifiers which reproduced the performance of the whole ABC chain. Future work might involve the development of a more tailored pruning algorithm using a suitable prior structure over the ensemble member weightings.

It may be desirable to ensure that the sparse weightings are all positive and sum to 1. Very preliminary work suggests that this might be achieved by drawing the weights from a Dirichlet distribution, while controlling the sparsity with the Dirichlet's precision – the equivalent of ARD for strictly positive weights.

While Bayesian model averaging is the standard method for creating ensembles of learners using Bayesian methods, it has come under some criticism in recent years [Domingos, 2000; Minka, 2000; Monteith et al., 2011]. Domingos [2000] argues that the problem with Bayesian model averaging is that it places too much weight on the maximum likelihood classifier and that performance of an ensemble is not just dependent on the ability to deal with parameter uncertainty. In [Monteith et al., 2011] Bayesian model averaging is modified to integrate over classifier *combinations* rather than integrating over individual classifiers in a method referred to as "Bayesian model combination". Future work could involve the application of the Bayesian model combination algorithm proposed by Monteith et al. [2011] to generate ensembles of classifiers from the ABC Chain samples. We note, however, that this method comes with an increased computational cost, not just in terms of training, but also in terms of producing the final ensemble prediction as it is generated by combining the predictions of multiple ensembles.

6. Conclusions

Here, each of the three chapters of this work is summarised along with some of the possible future work arising from them. Finally an overview of this work is given.

6.1. Multi-objective optimisation of RVM classifiers

In Chapter 3 a multi-objective evolutionary algorithm was used to locate classifiers whose locations on the ROC graph were optimal. Using the MOEA, hyper-parameters were set for RVM classifiers which were evaluated in terms of true and false positive rates along with a novel measure of RVM classifier complexity. The set of solutions containing the best trade-offs between these competing objectives was found, yielding an approximation to the Pareto front. By including complexity as an objective models whose additional complexity yielded no improvement in true and false positive rates were eliminated, thus encouraging sparseness.

In order to further reduce any potential overfitting during the training process, a number of methods for regularising the RVM classifiers generated by the MOEA were examined. A number of schemes based on maintaining separate training and validation archives were explored. A scheme based on K-fold cross validation was also investigated.

Experiments were carried out on a number of standard benchmark datasets utilising multi-resolution kernels in which the performance of RVM classifiers learnt by the MOEA training methods were compared with the RVM trained using the maximum likelihood method. Performance of MOEA trained classifiers was found to be equivalent to that of the RVM whilst avoiding overfitting.

Of the MOEA training schemes proposed, the K-fold methods were the most successful at preventing overfitting whilst still achieving good generalisation performance. The two-archive schemes, however, appeared to over-regularise the learned models by excluding too many basis functions. The K-fold method should prove useful for controlling overfitting in general evolutionary optimisation, a common and important consideration.

As a result of the MOEA training process we are effectively presented with ROC curves optimised over varying RVM complexities. A remaining question, however, is how to identify an appropriate complexity *a priori* from which to select a solution. The classifiers learnt by the MOEA have been regularised as part of the training process, thanks to the inclusion of the complexity measure, but the most complex solutions located are those which perform best on, and thus overfit to, the training data. The K-fold cross validation scheme successfully reduced the complexity of the learnt solutions whilst maintaining good generalisation performance.

Still, automatic selection of an appropriate complexity remains a problem. One possible solution would be manual inspection of the front to identify "knees" in the complexity from which to select. This gives the end user of the system the ability to select their own

choice of trade-off between true and false positive rates and complexity, though this may be time consuming or impractical with larger numbers of optimised objectives, requiring alternative visualisation methods [e.g. Walker et al., 2010]. While end user selection of a preferred solution is advantageous, ideally one would like to select a particular complexity, or range of complexities, and flatten down the Pareto fronts to the true positive rate - false positive rate plane. Thus, it would be preferable to have some more principled method for identifying appropriate complexities automatically. Such a method could also prove useful in indicating whether an RVM trained with the standard maximum likelihood method had overfit.

Front smoothness and apparent performance is dependent on the underlying data. Prediction uncertainty is introduced as a result of the finite nature of the training data and uncertainty in the chosen model parametrisation. Thus, two different but statistically equivalent equal sized samples could lead to different ROC performance. Ideally, one would like to incorporate this uncertainty into the generation of Pareto sets of classifiers. Further work might centre around using measures such as dominance probability [Fieldsend and Everson, 2005, 2008b] when deciding whether or not to add a solution to the elite archive. This is similar to the K-fold cross validation scheme presented in Chapter 3, in that solutions only enter the elite archive if it is very likely they dominate others.

6.2. Gene Selection

In Chapter 4 the task of gene selection for classification of various genetic conditions was examined. Bioinformatics datasets typically contain very large numbers of different gene expressions but relatively few samples. Due to the high dimensionality of such sparse datasets, classifiers with near perfect training performance can easily be obtained but these often have poor generalisation capability. Depending on the condition involved and the cost and intrusiveness of the treatment needed, evaluation over a range of costs will often be desirable.

A multi-objective optimising evolutionary algorithm was used to identify genes for classification by minimising model complexity while simultaneously maximising the area under the ROC curve. By maximising area under the curve we locate the classifier that gives the best class discrimination at a given model complexity. Inclusion of complexity means that increasingly complex classifiers yielding no improvement in AUC will be excluded, thus encouraging sparseness.

To further avoid the potential incorrect selection of genes for classification the repeated data partitioning scheme introduced by Li et al. [2002] was utilised. In this process, datasets are randomly split in half and models trained on each half with their gene selections recorded. This is repeated some number of times recording gene selections for each partitioning. The chances of a gene, or set of genes, repeatedly being coincident to both halves of numerous partitions are low. Thus, if a gene is selected often there is high confidence in it being significant for classification.

Experiments were carried out with the MOEA and RVM using the partitioning routine on a number of well-studied bioinformatics datasets. Both the RVM and MOEA successfully located genes indicated in the literature as being related to the classification problems

concerned. Comparing gene selection counts, it was noted that the MOEA was selecting genes less often than the RVM though it was still locating important ones. Analysis of results on the Leukaemia dataset found the MOEA to be locating solutions that achieved good class separation on the training data using very few genes. Such models are possible due to the sparsity of the datasets.

The InChianti population study is a recent bioinformatics dataset used in a number of current research tasks. Experiments were carried out using the MOEA and RVM with the partitioning routine in order to identify genes for a number of classification problems on this dataset. Gender classification of the InChianti dataset was carried out as an initial investigative experiment with both the MOEA and RVM successfully identifying sex linked gene expressions – in particular those lying on the X and Y chromosomes. Ageing is an important factor for many conditions and there is much current interest in identifying gene expression changes associated with it. Following Harries et al. [2011a] the dataset was divided into two age groups to form two classes and genes sought for distinguishing between them. Both the MOEA and RVM located genes indicated as important for ageing by Harries et al. [2011a]. When carrying out experiments to identify genes for the prediction of Extreme Cognitive Divergence the RVM and MOEA did not agree on the important genes. Nor did they agree with another recent work on the dataset [Harries et al., 2011b].

In order to better understand why the MOEA and RVM solutions were selecting different gene combinations with which to predict Extreme Cognitive Divergence, further analysis of the results was carried out. While the RVM trained classifiers were found to achieve near perfect training data performance, they failed to generalise well to the test data, apparently overfitting. Meanwhile, the sparser MOEA located solutions generalised well to new data. Since none of the top gene selections agreed and performance discrepancies were found, several conclusions can be drawn. Firstly the RVM had overfit whilst the MOEA had successfully regularised the solutions. Second, the noise in the targets meant that it was hard to correctly identify the responsible genes and so different key genes were located by different methods. Finally, with only 35 of the 132 16618-dimensional samples belonging to the extreme cognitive divergence class, the dataset was so sparse as to allow myriad combinations of genes all giving good classification performance.

For some classification problems there may be a number of underlying sub-classes that result in the class labels the model is trying to predict. For example, were we looking to predict someone having cancer based on gene expression data we know there are different cancer types: Breast cancer, Colon cancer, different types of Leukaemia, etc. Each individual condition has different associated genes but all are part of the cancer class. However, it is likely we will not be aware of which samples belong to each of the underlying classes when building our model. For the case of the InChianti dataset when looking to predict cognitive decline it may be that a number of different conditions affect cognition scores, e.g. Alzheimers, stroke propensity, genetic age (as opposed to physical), etc. However, we do not have this information. Future work could centre around detection and selection of genes associated with these sub-classes for building classification models.

As a result of learning sparse models, the selection of one particular gene may result in others being disregarded, effectively "masking" them as a result of selecting the first

gene. This could occur when a gene pairing leads to a particular condition. It may be that in the training data whenever one gene is active so is the other and so a sparse learning procedure would most likely select one of the pair but not the other. Knowledge of the second gene, however, would still be useful.

Investigation into this masking effect could be carried out as part of future work in order to identify any other important genes that may being hidden but which still yield good classification performance. Initial experiments might involve the removal of the top gene selections from the dataset and re-running the training process with the remaining ones. Thus, any masked genes should be selected and appear a significant number of times in the gene selection counts. While such experiments could demonstrate the existence of these masked genes, a more principled method that detected masking as part of the training process would be desirable, perhaps even informing the user of any gene pairings present in the data.

6.3. Approximate Bayesian Computation MCMC

For many classification problems hard, non-probabilistic classifiers are used. Such classifiers learn a single set of parameters during the training process whose values are uncertain due to limitations on available data. The Bayesian framework allows one to address the effects of this parameter uncertainty by averaging over classifiers weighted by the posterior probability of their particular parametrisations.

When working with hard classifiers computation of the posterior probability is not possible. Therefore, in Chapter 5 an Approximate Bayesian Computation Markov chain Monte Carlo algorithm was used to sample model parameters for hard classifiers using area under the ROC curve as a measure of performance. Thus, it was possible to sample model parametrisations and average over the hard predictions to obtain probabilities.

Experiments were carried out in which ABC MCMC was used to sample parametrisations for MLPs with hard classification thresholds. The capability to produce ROC curves close to the Bayes optimal ROC was demonstrated on the Ripley synthetic dataset [Ripley, 1994]

Future work with the ABC MCMC method could involve sampling of parameters for an existing industrial classifier that gives hard classifications. Of particular interest would be applying it to the NATS Short Term Conflict Alert (STCA) system which predicts aircraft collisions in the form of hard classifications. Work by Reckhouse et al. [2010] has already located approximate Pareto fronts of model parametrisations which could be used to compare performance obtained by ABC MCMC parametrised models.

As a result of the sampling process large numbers of model parametrisations are obtained. Having to make and average over such large numbers of predictions could be undesirable in situations where rapid classification is required or computational power is limited. Ideally, one would like to reproduce the performance obtained by averaging over the whole chain with only a subset. Therefore, methods for learning sparse weightings of the sampled classifiers were investigated.

Treating the chain prediction as the targets and each parametrised classifier as a basis function, the RVM was used to learn a sparse weighted subset of the classifiers with

which to make predictions. Given the significant reduction in the number of classifiers used in the sparse weighting, there was only a minor reduction in ROC performance compared to predictions made using the whole chain. Further investigations into methods for producing sparse weightings could be carried out and algorithms could be written casting this classifier selection in a more appropriate Bayesian framework in which all the weights are kept positive and sum to one.

6.4. Overview

Receiver Operating Characteristic curves are a popular and widely used method for evaluating performance of classifiers. By plotting classifier true and false positive rates on the ROC graph one can compare classifiers over a range of misclassification costs and identify the set of classifiers giving the best set of trade-offs between these competing objectives. Calculation of the area under the ROC curve provides a measure as to how well a classifier (or set of classifiers) is able to discriminate between classes.

The overarching theme of this thesis has been the incorporation of ROC performance into the model training process. Using a novel measure of RVM classifier complexity a multi-objective evolutionary algorithm was successfully used to optimise RVM classifier ROC curves over a range of complexities. This was further built on using a K-fold cross validation scheme for preventing overfitting during the evolutionary optimisation process which should have more general application to evolutionary learning.

An MOEA was used to identify genes for classification of various genetic conditions using area under the ROC curve (AUC), measured by the Mann-Whitney-Wilcoxon statistic, as a measure of class discrimination ability. The set of optimal trade-offs between AUC and model complexity were found, selecting important genes whilst avoiding overfitting.

An Approximate Bayesian Computation method using AUC as a measure of classifier performance was successfully used to obtain estimates of posterior probability of class membership from hard classifiers, giving performance close to the Bayes optimal ROC curve. In order to reduce computational overheads, sparse weightings were investigated demonstrating the capability to closely reproduce the performance of the ABC chain with a smaller ensemble of classifiers.

Throughout these experiments, the use of ROC curves in the training process has led to success. In the absence of sufficiently large numbers of data samples, as is typical with most problems, and knowledge of the underlying data distributions, ROC curves have proven invaluable in the training and performance assessment of classifiers. Equally importantly, by presenting the end user with ROC curves they are able to select the operating point appropriate for their misclassification costs.

A. Additional Plots for Chapter 3

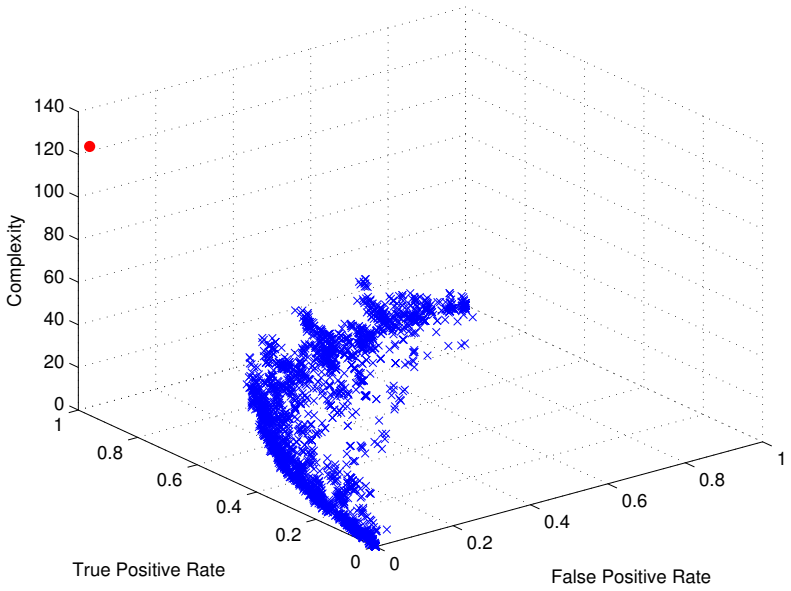


Figure A.1.: Solutions located for the first fold of the German dataset. Blue x's mark the EA RVM solutions and filled red o marks the fRVM solution.

B. InChianti Dataset Cofactors

The following tables contain details of the cofactors for the InChianti dataset used in chapter 4

Cofactor	**Details**
site	binary: people are from one site or the other
amp_batch	amplification-batch during RNA quantification
hyb_batch	hybridization-batch during RNA quantification
q_p_neu	percentage of cells used that we neutrophils
q_p_lin	percentage of cells used that we lymphocytes
q_p_mon	percentage of cells used that we monocytes
q_p_eos	percentage of cells used that we eosinophils

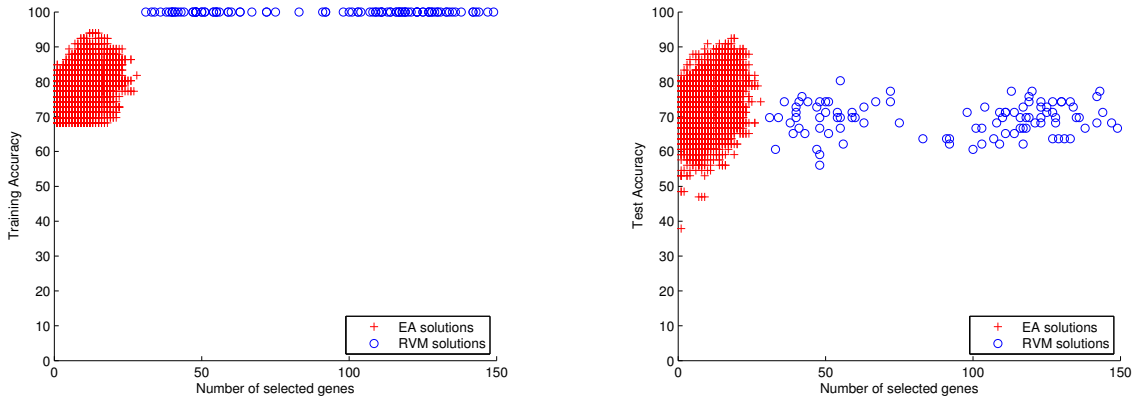
Table B.1.: Technical Cofactors for the InChianti Dataset

Cofactor	**Details**
sex	gender, binary: 1=male, 2=female
age_at_extraction	age at RNA extraction (FUP3), continuous
pbqmi	BMI at FUP3, continuous
q_packyrs_lt	Smoking status: categorical = number of pack-years smoked
q_waist	waist circumference, centimetres
q_education	highest level of education attained, in five categories: none, elementary, secondary, high school and university/professional
evidenceofstroke	binary: 1=evidence of stroke, 0=none
apoe	APOE a gene shown to have an association with cognitive function This is categorical and defines each individual's genotype for this genes
t2d	Type 2 Diabetes status/severity: categorical 0=no, 1=somewhat etc.
x_act_last	Physical activity at in year leading up to baseline
y_act_3yrs	Physical activity in 3yrs prior to FUP1
y_act_last	Physical activity in year leading up to FUP1
z_act_3yrs	Physical activity in 3yrs prior to FUP2
z_act_last	Physical activity in year leading up to FUP2
q_act_3yrs	Physical activity in 3yrs prior to FUP3
q_act_last	Physical activity in year leading up to FUP3

Table B.2.: "Other Cofactors" in the InChianti Dataset

C. Extreme Cognitive Divergence Results analysis

Further analysis of the Extreme Cognitive Divergence results carried out purely in terms of accuracy is presented here for completeness and fair comparison of the EA and RVM trained solutions. As expected from the results in Chapter 4.8, the performance of the RVM is better than the EA on the training dataset (figure C.1a) but drops on the test dataset (figure C.1b). Examining figure C.2, we get a better view of this train-test accuracy drop. Again it appears that the RVM has overfit to the training data and thus failed to generalise well.



(a) Training (b) Test

Figure C.1.: Comparison of features with Training and Test rates. For the EA the threshold that produces the best training accuracy is used.

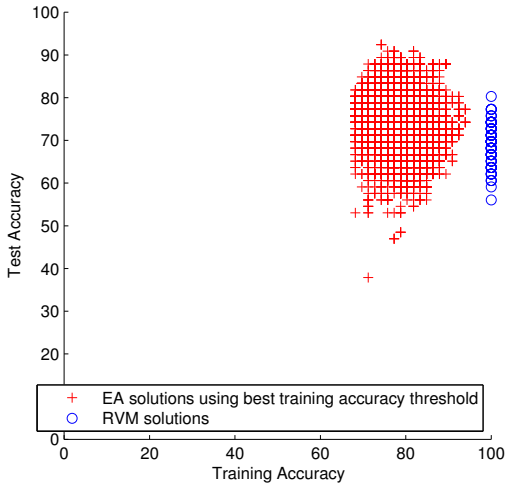


Figure C.2.: Comparing Training and Test accuracies

Bibliography

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

Anastasio, M., Kupinski, M., and Nishikawa, R. (1998). Optimization and FROC analysis of rule-based detection schemes using a multiobjective approach. *Medical Imaging, IEEE Transactions on*, 17(6):1089 –1093.

Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., et al. (2000). Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29.

Attias, H. (1999). Inferring parameters and structure of latent variable models by variational Bayes. In *Proc. 15th Conf. on Uncertainty in Artificial Intelligence*, volume 2.

Aupetit, M. (2009). Nearly homogeneous multi-partitioning with a deterministic generator. *Neurocomput.*, 72:1379–1389.

Bae, K. and Mallick, B. (2004). Gene selection using a two-level hierarchical bayesian model. *Bioinformatics*, 20(18):3423.

Bernardo, J. and Smith, A. (1994). *Bayesian theory*, volume 62. Wiley New York.

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford university press.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer New York.

Bishop, C. and Tipping, M. (2000). Variational relevance vector machines. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 46–53.

Blum, M. and François, O. (2010). Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(1):63–73.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

Braga-Neto, U. M. and Dougherty, E. R. (2004). Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380.

Breiman, L. (1984). *Classification and regression trees*. Chapman & Hall/CRC.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Burges, C. (1996). Simplified support vector decision rules. In Saitta, L., editor, *Thirteenth International Conference on Machine Learning*, pages 71–77. Morgan Kauffman.

Burges, C. and Schölkopf, B. (1997). Improving the accuracy and speed of support vector machines. *Advances in neural information processing systems*, 9:375–381.

Chawla, N., Hall, L., Bowyer, K., and Kegelmeyer, W. (2004). Learning ensembles from bites: A scalable and accurate approach. *The Journal of Machine Learning Research*, 5:421–451.

Chen, H. (2008). *Diversity and Regularization in Neural Network Ensembles*. PhD thesis, University of Birmingham.

Chen, H., Tino, P., and Yao, X. (2006). A probabilistic ensemble pruning algorithm. In *Workshops on Optimization-based Data Mining Techniques with Applications in Sixth IEEE International Conference on Data mining*, pages 878–882. IEEE.

Clark, A. and Everson, R. (2008). Multi-objective optimisation of relevance vector machines: Selecting sparse features for face verification. ICML/UAI/COLT Workshop on Sparse Optimisation and Variable Selection.

Clark, A. and Everson, R. (2011a). Evolving sparse multi-resolution rvm classifiers. In Dupenois, M. and Walker, D., editors, *Proceedings of the 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011)*, pages 53–60, Exeter, UK. PCCAT, College of Engineering, Mathematics and Physical Sciences, University of Exeter.

Clark, A. and Everson, R. (2011b). Multi-objective learning of relevance vector machine classifiers with multi-resolution kernels. *Pattern Recognition*. In Press.

Clark, A. and Everson, R. (2011c). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at 2nd Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), University of Exeter.

Clark, A. and Everson, R. (2011d). Soft classifiers from hard; using ABC to average hard classifiers. Poster presentation at ABCiL 2011, Imperial College London.

Coello, C., Lamont, G., and Van Veldhuizen, D. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 2. Springer-Verlag New York Inc.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27.

Damoulas, T., Ying, Y., Girolami, M., and Campbell, C. (2008). Inferring sparse kernel combinations and relevance vectors: An application to subcellular localization of proteins. In *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pages 577 –582.

Das, I. and Dennis, J. (1997). A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural and Multidisciplinary Optimization*, 14(1):63–69.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. Wiley.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, pages 849–858. Springer.

Deb, K., Sinha, A., and Kukkonen, S. (2006). Multi-objective test problems, linkages, and evolutionary methodologies. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 1141–1148, New York, NY, USA. ACM.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002). Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825 –830.

Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Prentice-Hall, London.

Devroye, L. (1986). *Non-uniform random variate generation*. Springer-Verlag New York.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. In *Proceedings of Seventeenth International Conference on Machine Learning*, pages 223–230.

Duda, R., Hart, P., and Stork, D. (2001). Pattern classification. *John Wiley & Sons*.

Everson, R. and Fieldsend, J. (2006a). Multi-class ROC analysis from a multi-objective optimisation perspective. *Pattern Recognition Letters*, 27:531–556.

Everson, R. and Fieldsend, J. (2006b). Multiobjective optimization of safety related systems: an application to short-term conflict alert. *Evolutionary Computation, IEEE Transactions on*, 10(2):187 – 198.

Everson, R. M. and Fieldsend, J. E. (2006c). Multiobjective optimization of safety related systems: An application to short-term conflict alert. *IEEE Transactions on Evolutionary Computation*, 10(2):187–198.

Faul, A. and Tipping, M. (2002). Analysis of sparse bayesian learning. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14, pages 383–389. MIT Press.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

Ferrucci, L., Bandinelli, S., Benvenuti, E., Di Iorio, A., Macchi, C., Harris, T., and Guralnik, J. (2000). Subsystems contributing to the decline in ability to walk: bridging the gap between epidemiology and geriatric practice in the InCHIANTI study. *Journal of the American Geriatrics Society*, 48(12):1618–1625.

Fieldsend, J. (2006). Regression error characteristic optimisation of non-linear models. *Multi-objective machine learning*, 16:103–123.

Fieldsend, J. and Everson, R. (2005). Multi-objective optimisation in the presence of uncertainty. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, pages 476–483. IEEE.

Fieldsend, J. and Everson, R. (2008a). Multi-objective supervised learning. In Knowles, J., Corne, D., and Deb, K., editors, *Multi-Objective Problem Solving from Nature: From Concepts to Applications*, pages 155–176. Springer-Verlag.

Fieldsend, J. and Everson, R. (2008b). On the efficient use of uncertainty when performing expensive ROC optimisation. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3984 –3991.

Fieldsend, J., Everson, R., and Singh, S. (2003). Using unconstrained elite archives for multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 7(3):305 – 323.

Fieldsend, J. and Singh, S. (2005). Pareto evolutionary neural networks. *Neural Networks, IEEE Transactions on*, 16(2):338 – 354.

Fieldsend, J. E. and Singh, S. (2004). Optimizing forecast model complexity using multiobjective evolutionary algorithms. *Applications of Multi-Objective Evolutionary Algorithms*, pages 675–700.

Figueiredo, M. (2001). Adaptive sparseness using Jeffreys prior. In *NIPS*, pages 697–704.

Figueiredo, M. (2003). Adaptive sparseness for supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1050–1159.

Fink, D. (1997). A compendium of conjugate priors.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188.

Folstein, M. F., Folstein, S. E., and McHugh, P. R. (1975). mini-mental state: A practical method for grading the cognitive state of patients for the clinician. *Journal of Psychiatric Research*, 12(3):189 – 198.

Fonseca, C. and Fleming, P. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the fifth international conference on genetic algorithms*, pages 416–423.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2003). Bayesian data analysis.

Gentle, J. (2003). *Random number generation and Monte Carlo methods*. Springer Verlag.

Gilks, W., Richardson, S., and Spiegelhalter, D. (1996). *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

Goldberg, D. (1989). *Genetic algorithms for search, optimization, and machine learning*. Addison-Wesley.

Golub, T., Slonim, D., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J., Caligiuri, M., et al. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531.

Graf, H. P., Cosatto, E., Bottou, L., Durdanovic, I., and Vapnik, V. (2004). Parallel support vector machines: The cascade SVM. In *NIPS'04*.

Green, D. and Swets, J. (1966). *Signal detection theory and psychophysics*, volume 1974. Wiley New York.

Grunet da Fonseca, V., Fonseca, C. M., and Hall, A. O. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. In Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 213–225. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389–422.

Hand, D. and Till, R. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

Hanley, J. and McNeil, B. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36.

Harries, L., Hernandez, D., Henley, W., Wood, A., Holly, A., Bradley-Smith, R., Yaghootkar, H., Dutta, A. Murray, A., Frayling, T., Guralnik, J., Bandinelli, S., Singleton, A., Ferrucci, L., and Melzer, D. (2011a). Gene expression changes in human aging. *Accepted for publication in: Aging Cell*.

Harries, L. W., Bradley Smith, R. M., J., L. D., Pilling, L. C., Henley, W., Hernandez, D., Guralnik, J. M., Bandinelli, S., Singleton, A amd Ferrucci, L., and Melzer, D. (2011b). Leukocyte CCR2 expression is associated with cognitive function in ageing humans. *TBC*.

Hashem, S. (1993). *Optimal Linear Combinations of Neural Networks*. PhD thesis, Purdue University.

Hastie, T. and Tibshirani, R. (1990). *Generalized additive models*. Chapman & Hall/CRC.

Hastie, T., Tibshirani, R., and Friedman, J. (2004). The elements of statistical learning: Data mining, inference, and prediction. *Springer*.

Hastings, W. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97.

Hedenfalk, I., Duggan, D., Chen, Y., Radmacher, M., Bittner, M., Simon, R., Meltzer, P., Gusterson, B., and Esteller, M. e. a. (2001). Gene-expression profiles in hereditary breast cancer. *New England Journal of Medicine*, 344(8):539–548.

Hughes, E. (2003). Multiple single objective Pareto sampling. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 4, pages 2678 – 2684.

Hughes, E. (2007). MSOPS-II: A general-purpose Many-Objective optimiser. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3944 –3951.

InChianti Group (2011). Private correspondence.

Ishibuchi, H., Tsukamoto, N., and Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2419 –2426.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Jin, Y. (2006). *Multi-objective machine learning*, volume 16 of *Studies in Computational Intelligence*. Springer Verlag.

Jin, Y., Okabe, T., and Sendhoff, B. (2004a). Evolutionary multi-objective optimization approach to constructing neural network ensembles for regression. *Applications of Multi-Objective Evolutionary Algorithms*, pages 653–673.

Jin, Y., Okabe, T., and Sendhoff, B. (2004b). Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 1–8.

Jin, Y., Olhofer, M., and Sendhoff, B. (2001). Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how? In *Genetic and Evolutionary Computation Conference*, pages 1042–1049, San Francisco, USA.

Jin, Y., Sendhoff, B., and Korner, E. (2006). Simultaneous generation of accurate and interpretable neural network classifiers. *Multi-objective machine learning*, 16:291.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer.

Knowles, J. and Corne, D. (2000). Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation*, 8(2):149–172.

Knowles, J., Corne, D., and Deb, K., editors (2008). *Multi-Objective Problem Solving from Nature: From Concepts to Applications*. Springer-Verlag.

Kotsiantis, S. B. (2007). Supervised machine learning : A review of classification techniques. *Informatica*, 31:249–268.

Kullback, S. (1959). Statistics and information theory.

Kullback, S. and Leibler, R. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Kupinski, M. and Anastasio, M. (1999). Multiobjective genetic optimization of diagnostic classifiers with implications for generating receiver operating characteristic curves. *Medical Imaging, IEEE Transactions on*, 18(8):675 –685.

Lampinen, J. and Vehtari, A. (2001). Bayesian approach for neural networks–review and case studies. *Neural Networks*, 14(3):257–274.

Laumanns, M., Zitzler, E., and Thiele, L. (2000). A unified model for multi-objective evolutionary algorithms with elitism. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 46 –53.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 –2324.

Lee, K., Sha, N., Dougherty, E., Vannucci, M., and Mallick, B. (2003). Gene selection: a Bayesian variable selection approach. *Bioinformatics*, 19(1):90.

Li, Y., Campbell, C., and Tipping, M. (2002). Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18(10):1332.

Lofstrom, T., Johansson, U., and Bostrom, H. (2009). Ensemble member selection using multi-objective optimization. In *Computational Intelligence and Data Mining, 2009. CIDM '09. IEEE Symposium on*, pages 245 –251.

MacKay, D. (1992a). Bayesian interpolation. *Neural computation*, 4(3):415–447.

MacKay, D. (1998). Introduction to Monte Carlo methods. *Learning in graphical models*, pages 175–204.

MacKay, D. J. C. (1992b). The evidence framework applied to classification networks. *Neural Computation*, pages 4–720.

Margineantu, D. and Dietterich, T. (1997). Pruning adaptive boosting. In *Machine learning: proceedings of the fourteenth International Conference (ICML'97)*, pages 211–218.

Marjoram, P., Molitor, J., Plagnol, V., and Tavaré, S. (2003). Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 100(26):15324.

McClymont, K., Keedwell, E., Savic, D., and Randall-Smith, M. (2010). Mitigating discolouration risk with optimised network design. In *HIC*, China.

McLachlan, G. (2004). Discriminant analysis and statistical pattern recognition.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., et al. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087.

Mierswa, I. (2007). Controlling overfitting with multi-objective support vector machines. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1830–1837. ACM.

Minka, T. (2000). Bayesian model averaging is not model combination. *MIT Media Lab Note December 2000*.

Minka, T. (2001). Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369.

Monteith, K., Carroll, J., Seppi, K., and Martinez, T. (2011). Turning Bayesian model averaging into Bayesian model combination. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2657 –2663.

Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural computation*, 1(2):281–294.

Myers, J. and Forgy, E. (1963). The development of numerical credit evaluation systems. *Journal of the American Statistical Association*, 58:799–806.

Nabney, I. (2002). *NETLAB: algorithms for pattern recognition*. Springer Verlag.

Nabney, I. T. (1999). Efficient training of RBF networks for classification. In *ICANN99*, pages 210–215, London. IEE.

NHS (2010). Moyamoya disease. Great Ormond Street Hospital for Children, London.

Notterman, D., Alon, U., Sierk, A., and Levine, A. (2001). Transcriptional gene expression profiles of colorectal adenoma, adenocarcinoma, and normal tissue examined by oligonucleotide arrays. *Cancer Research*, 61(7):3124.

Obuchowski, N. A. (2003). Receiver operating characteristic curves and their use in radiology. *Radiology*, 229(1):3–8.

Ó Ruanaidth, J. J. K. and Fitzgerald, W. J. (1996). *Numerical Bayesian Methods Applied to Signal Processing*. Springer.

Park, M. and Hastie, T. (2007). L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677.

Pilling, L., Melzer, D., and Harries, L. (2011). Private correspondence.

Platt, J. (2000). Probabilistic outputs for support vector machines. *Advances in Large Margin Classifiers*.

Provost, F. and Fawcett, T. (1997). Analysis and visualisation of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48, Menlo Park, CA. AAAI Press.

Provost, F. and Fawcett, T. (1998). Robust classification systems for imprecise environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 706–7, Madison, WI. AAAI Press.

Psorakis, I., Damoulas, T., and Girolami, M. (2010). Multiclass relevance vector machines: Sparsity and accuracy. *Neural Networks, IEEE Transactions on*, 21(10):1588 –1598.

Rasmussen, C. (2004). Gaussian processes in machine learning. *Advanced Lectures on Machine Learning*, pages 63–71.

Rätsch, G., Onoda, T., and Müller, K.-R. (2001). Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320.

Reckhouse, W. J., Fieldsend, J. E., and Everson, R. M. (2010). Variable interactions and exploring parameter space in an expensive optimisation problem: Optimising short term conflict alert. In *IEEE Congress on Evolutionary Computation*, pages 1–8.

Rice, J. (1988). *Mathematical Statistics and Data Analysis*. Statistics/probability series. Wadsworth and Brooks/Cole Advanced Books and Software.

Ripley, B. D. (1994). Neural networks and related methods for classification. *Journal of the Royal Statistical Society*, 56(3):409–456.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rumelhard, D. E., McClelland, J. L., and PDP Research Group, editors (1986). *Learning internal representations by error propagation*, volume 1 of *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Schaffer, J. (1985a). Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st international conference on genetic algorithms*, pages 93–100.

Schaffer, J. (1985b). Some experiments in machine learning using vector evaluated genetic algorithms. Technical report, Vanderbilt Univ., Nashville, TN (USA).

Schapire, R. (1990). The strength of weak learnability. *Machine learning*, 5(2):197–227.

Schapire, R. (1999). A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1401–1406.

Schmolck, A. (2008). *Smooth Relevance Vector Machines*. PhD thesis, University of Exeter.

Schmolck, A. and Everson, R. M. (2007). Smooth relevance vector machine: a smoothness prior extension of the RVM. *Machine Learning*, 68(2):107–135.

Schölkopf, B. (2001). The kernel trick for distances. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, volume 13, page 301. The MIT Press.

Schölkopf, B., Burges, C., and Smola, A. (1999). *Advances in kernel methods: support vector learning*. MIT press Cambridge, MA.

Schölkopf, B. and Smola, A. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. the MIT Press.

Sejnowski, T. (1999). *Unsupervised learning: foundations of neural computation.* The MIT Press.

Shen, L. (2006). *Recognizing Faces - An Approach Based on Gabor Wavelets.* PhD thesis, University of Nottingham.

Shen, L., Bai, L., and Fairhurst, M. (2007). Gabor wavelets and general discriminant analysis for face identification and verification. *Image Vision Comput.*, 25:553–563.

Sivia, D. and Skilling, J. (2006). *Data analysis: A Bayesian tutorial.* Oxford University Press, USA.

Smola, A., Schölkopf, B., and Müller, K. (1998). The connection between regularization operators and support vector kernels. *Neural Networks*, 11(4):637–649.

Spackman, K. (1989). Signal detection theory: Valuable tools for evaluating inductive learning. In *Proceedings of the sixth international workshop on Machine learning*, pages 160–163. Morgan Kaufmann Publishers Inc.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248.

Sutton, R. and Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge University Press.

Suttorp, T. and Igel, C. (2006). Multi-objective optimization of support vector machines. *Multi-objective machine learning*, 16:199–220.

Swets, J. (1979). ROC analysis applied to the evaluation of medical imaging techniques. *Investigative Radiology*, (14):109–121.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Tikhonov, A. and Arsenin (1977). *Solutions of ill-posed problems.* Winston Washington, DC:.

Tipping, M. (2000). The Relevance Vector Machine. In Solla, A., Leen, T., and Muller, K., editors, *Advances in Neural Information Processing Systems*, volume 12, pages 652–658. MIT Press.

Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244.

Tipping, M. E. and Faul, A. (2003). Fast marginal likelihood maximisation for sparse Bayesian models. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, 2003.*

Toni, T., Welch, D., Strelkowa, N., Ipsen, A., and Stumpf, M. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31):187.

Vapnik, V. (1998). Statistical learning theory.

Walker, D., Everson, R., and Fieldsend, J. (2010). Visualisation and ordering of many-objective populations. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, Barcelona. IEEE.

Walker, D. J., Everson, R. M., and Fieldsend, J. E. (2011). Rank-based dimension reduction for many-criteria populations. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 107–108, New York, NY, USA. ACM.

Wilkinson, R., Steiper, M., Soligo, C., Martin, R., Yang, Z., and Tavaré, S. (2011). Dating primate divergences through an integrated analysis of palaeontological and molecular data. *Systematic Biology*, 60(1):16.

Wilkinson, R. and Tavaré, S. (2009). Estimating primate divergence times by using conditioned birth-and-death processes. *Theoretical population biology*, 75(4):278–285.

Williams, P. (1995). Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, 7:117–143.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102.

Zhou, Z., Wu, J., and Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial intelligence*, 137(1-2):239–263.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH). Diss ETH No. 13398.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm.

Zitzler, E. and Thiele, L. (1998). An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. Technical Report 43, Computing Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.