Aspects of Online Learning

Edward Francis Harrington

6 December 2004

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

Research School of Information Sciences and Engineering

Declaration

Except where otherwise indicated, this thesis is my own original work. Some of the material in thesis has already been published elsewhere in collaboration with others, the details of which are:

1. Harrington E., Kivinen J. and Williamson R. C., Channel Equalization and the Bayes Point Machine, International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 493-496, 2003.

2. Harrington, E., Kivinen, J., Williamson, R. C., Herbrich, R., & Platt J. Online Bayes Point Machine. Proceedings of PAKDD-03, 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 241-252, Heidelberg: Springer Verlag, 2003.

3. Harrington E., Expert mixture methods for adaptive channel equalization, Kaynak, O. et al. (Eds.): International Conference of Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP), LNCS 2714, pp. 538-545, Springer Verlag, 2003.

4. Harrington E. F., Online Ranking/Collaborative Filtering using the Perceptron Algorithm, Proceedings of the Twentieth International Conference of Machine Learning (ICML-2003), pp. 250-257, Morgan Kaufmann.

Edward Harrington

6 December 2004

Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University

Acknowledgements

My warmest thanks must go to my wife Annie, who provided me with great support and love, especially whilst writing this thesis.

I have been fortunate to have had the supervision of Bob Williamson, the many "fire side" like chats with him have kept my on track to complete this thesis. His guidance in machine learning and signal processing have proven to be invaluable. I would like to thank a number of people in machine learning community for their help over last three or so years: Peter Bartlett, Ralf Herbrich, Jyrki Kivinen, Shahar Mendelson, John Platt, Gunnar Rätsch, and Alex Smola. I am very grateful to Jyrki Kivinen for his supervision and his many thoughtful comments concerning this thesis which have lead to its improvement. I am also grateful to Ralf Herbrich for his guidance and encouragement throughout the last few years. I would like to thank the examiners for their constructive suggestions, as they resulted in a much improved thesis.

I am very grateful to my employer, the Defence Science and Technology Organisation, for their financial support in allowing me to undertake this PhD. Several people at DSTO have provide me with assistance during my PhD: Ian Doherty, Geoff Latham, Nigel McGinty and Garry Newsam. I am especially appreciative of the encouragement and the support given to me by Ian Doherty, and the late Geoff Latham, when applying for study leave to undertake a PhD. I would also like to thank Garry Newsam for his support during the later stages of my PhD.

Thanks to the head of Computer Sciences Laboratory, John Lloyd, for funding some of my travel during 2003. To Michelle Moravec many thanks go to her for her help with all my administration needs whilst at the Computer Sciences Laboratory.

The friendship of my fellow PhD students has helped me during the last three, or so years: Evan Greensmith, Omri Guttiman, Kee Siong Ng and Cheng Soon Ong. I also acknowledge the proof reading efforts of Evan Greensmith.

Lastly, I would like to thank my parents and family for their love and support.

Abstract

Online learning algorithms have several key advantages compared to their batch learning algorithm counterparts: they are generally more memory efficient, and computationally more efficient; they are simpler to implement; and they are able to adapt to changes where the learning model is time varying. Online algorithms because of their simplicity are very appealing to practitioners. This thesis investigates several online learning algorithms and their application. The thesis has an underlying theme of the idea of combining several simple algorithms to give better performance. In this thesis we investigate: combining weights, combining hypothesis, and (sort of) hierarchical combining.

Firstly, we propose a new online variant of the *Bayes point machine* (BPM), called the *online Bayes point machine* (OBPM). We study the theoretical and empirical performance of the OBPM algorithm. We show that the empirical performance of the OBPM algorithm is comparable with other large margin classifier methods such as the *approximately large margin algorithm* (ALMA) and methods which maximise the margin explicitly, like the *support vector machine* (SVM). The OBPM algorithm when used with a parallel architecture offers potential computational savings compared to ALMA. We compare the test error performance of the OBPM algorithm with other online algorithms: the Perceptron, the voted-Perceptron, and Bagging. We demonstrate that the combination of the voted-Perceptron algorithm and the OBPM algorithm, called voted-OBPM algorithm has better test error performance than the voted-Perceptron and Bagging algorithms. We investigate the use of various online voting methods against the problem of ranking, and the problem of collaborative filtering of instances. We look at the application of online Bagging and OBPM algorithms to the telecommunications problem of channel equalization. We show that both online methods were successful at reducing the effect on the test error of label flipping and additive noise.

Secondly, we introduce a new mixture of experts algorithm, the *fixed-share hierarchy* (FSH) algorithm. The FSH algorithm is able to track the mixture of experts when the switching rate between the best experts may not be constant. We study the theoretical aspects of the FSH and the practical application of it to adaptive equalization. Using simulations we show that the FSH algorithm is able to track the best expert, or mixture of experts, in both the case where the switching rate is constant and the case where the switching rate is time varying.

Contents

List of Figures

List of Tables

Introduction

This thesis considers a number of ways of improving online learning algorithms. The improvements are all of the general form: run a number of (variants of) simple online learning algorithms on (variant of) the original problem and data set. Combine the outputs of the simple algorithms in a manner which:

- preserves the online nature of the original algorithm,
- is simple to implement,
- improves the performance above that of the original simple online algorithm.

In considering this topic, connection is made to a number of topical ideas in machine learning including boosting, bagging, large margin algorithms, kernel methods and mixture of experts. The new techniques can be applied (and are experimentally explained in this thesis) to diverse areas including standard classification, ranking, collaborative filtering and channel equalization. The rest of this chapter provides the background to and a roadmap for the rest of this thesis.

1.1 Online versus batch learning

We consider two different algorithmic approaches to learning: online, and batch. The batch learning algorithm has at its disposal the entire training set to learn from. An online learning algorithm makes updates based on a single training example at a time, hence it only makes use of the part of the training set it has seen so far. Most of the online learning algorithms studied in this thesis can be trained in a batch mode where they see the entire training set multiple times or *epochs* of the entire training set. The updates of the online learning algorithms in this batch mode still only use a single training example at a time, therefore maintaining their inherit simplicity. This simplicity of implementation of online learning algorithms is one of its appealing qualities for practitioners when compared to batch learning algorithms with more complex optimisations. Although the online learning algorithms can have simple implementations their analysis is not in general that simple. One way usually used to measure the success of an online learning algorithm is if its performance is asymptotically identical to its batch

counterpart, in which case it is referred to as being *lossless*. Being lossless suggests that the best we can do with an online learning approach is to replicate the performance of a batch learning algorithm. However such a comparison neglects the deficiency of a batch algorithm in a situation where the target solution is non-stationary. Even in a stationary setting there are still a number of advantages of using online learning rather than batch learning, as online learning algorithms are generally: simpler, more memory efficient, ideal for real-time implementations with large data sets. While online learning methods have advantages for when the data set size is large, there is a distinct disadvantage for when the size of the data set is small, and in general batch methods will in that case have superior performance compared to online methods.

1.2 Motivations

Recently Bottou and Le Cun [2003, 2004] showed that an online learning algorithm can learn just as fast (with the same number of examples) as the batch version of the same algorithm. This implies that the online learning algorithm can be more computationally efficient compared to a batch learning algorithm as it requires only the current example to perform an update and not the entire training set. The potential computational savings is one motivating factor for the development of the online algorithms investigated in this thesis, especially when the data sets to be learnt are large in size. One further advantage of using the current training example is that it naturally tends to be able to adapt to changes in the target concept due to time dependencies in the learning model. This ability to adapt to a target concept which is drifting is particularly applicable in time-series prediction problems such as classical equalization problems in telecommunications. There are also rational online variants of more modern learning problems such as sorting documents on web pages where the topic of the documents has some time dependence.

1.3 Notational convention

Scalars are always written using lower case. Both sets and matrices using upper case. A vector, or sequence of vectors, are written using a lower case symbol in bold face. When defining elements within a vector they will be enclosed in square brackets [and]. For example the following defines the elements of a column vector $\mathbf{v} := [v(1), v(2), \ldots, v(d)]'$, where $v(i)$ defines the ith element and $'$ denotes a vector transpose. Sometimes the ith element of a vector will be defined with a subscript i.e. v_i is used. A sequence of vectors, scalars, etcetera, will be enclosed by (and), where the ith element of the sequence is denoted by a subscript. Unless explicitly stated, the Euclidean norm of a vector is denoted by the symbol $\|\cdot\|$. Any inner product between two vectors $\mathbf{x}$ and $\mathbf{s}$ is denoted by $(\mathbf{x} \cdot \mathbf{s})$. Within in this thesis we use the notation $:=$ to denote assignment when in an algorithm, and to denote a definition otherwise.

A glossary of the main symbols used can be found at the end of the thesis.

1.4 Learning setting

The prime concern of this thesis is the supervised learning problem. In the supervised learning setting the learning algorithm sees a sequence of T training examples, $\mathbf{z} = (\mathrm{z}_1, \ldots, \mathrm{z}_T) := ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$, comprising an instance $\mathbf{x}$ in the instance space $\mathcal{X} \subseteq \mathbb{R}^d$ and y in the label space $\mathcal{Y} \subseteq \mathbb{R}$. We assume that the training sequence of T examples $\mathbf{z} = ((\mathbf{x}_i, y_i))_{i=1}^T$ are drawn from an unknown distribution P_z. We consider two different problem settings to supervised learning: regression and classification. In the linear regression setting the learning algorithm updates its weight vector $\mathbf{w} \in \mathbb{R}^d$ at each training example $\mathbf{x}_t$ in $\mathbf{z}$ according to the error made predicting y_t using the predictor "$\hat{y}_t = (\mathbf{w} \cdot \mathbf{x}_t)$". In classification we wish to categorise, or discriminate, the instances into different classes defined by the labels y. The main classification setting considered throughout this thesis is the two class ($m = 2$) or binary label case where $y \in \{-1, +1\}$. We do, however, consider the case of discriminating the instances into m classes in this thesis; we give more detail on how we do that later. The classifier weight vector $\mathbf{w} \in \mathbb{R}^d$ defines the d-dimensional hyperplane[1] $\{\mathbf{x}\colon (\mathbf{w} \cdot \mathbf{x}) = 0\}$ which is the decision boundary associated with the hypothesis $h(\mathbf{x}) = \mathrm{sgn}\,(\mathbf{w} \cdot \mathbf{x})$, where $\mathrm{sgn}(v) = 1$ if $v \geq 0$ and $\mathrm{sgn}(v) = -1$ if $v < 0$. We often assume that the target labels y_i are defined by $y_i = \mathrm{sgn}\,(\mathbf{u} \cdot \mathbf{x}_i)$; in other words, there exists a linear classifier which can correctly classify the data.

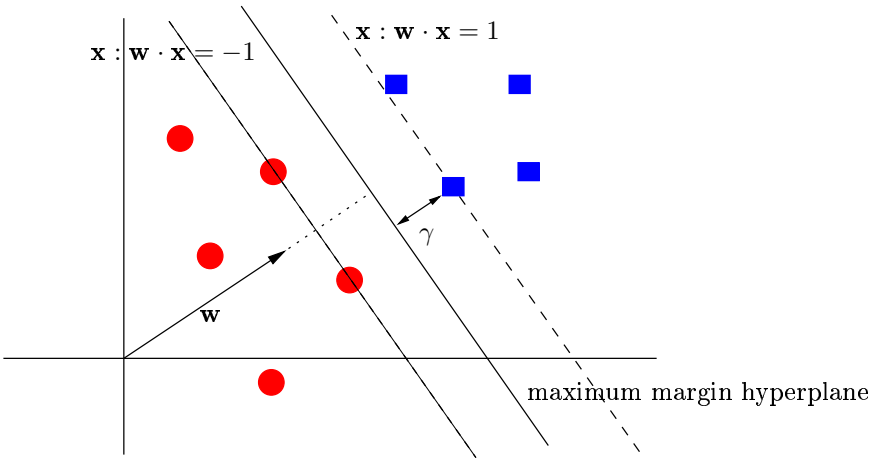

Figure 1.1: Binary classification example in two dimensions illustrating the hyperplane with γ being the maximum margin separating the two classes (squares representing instances in $+1$ class and circles representing instances in -1 class).

[1] Note that in the case of a weight vector with bias term (also referred to as the threshold) an extra dimension with the value of $+1$ is added.

1.5 Measures of performance

In this section we define and discuss different measures of performance for use in both the classification and regression settings. We start with the *zero-one loss* measure of performance for classification: given the true label y and the estimate $\hat{y} = h(\mathbf{x})$, the zero-one loss is given by

$$l_{0-1}(y, \hat{y}) := I(y \neq \hat{y}). \tag{1.1}$$

One strategy to produce a good learning algorithm is to minimise the training error (*empirical risk*)

$$R_{\text{emp}}(h, \mathbf{z}) := \frac{1}{T} \sum_{i=1}^{T} l_{0-1}(y_i, \hat{y}_i). \tag{1.2}$$

It is well known that minimising the training error does not necessarily translate to lower test errors. Ideally we want the learning algorithm to minimise the error we would expect to see, hence have good generalisation, which means minimising the *expected risk*

$$R(h, \mathrm{P_z}) := \mathrm{E_z}(l_{0-1}(y, \hat{y})). \tag{1.3}$$

One way of measuring the potential generalisation error of a classifier is via its *margin*. The margin $\gamma_{\mathbf{z}}(\mathbf{w})$ plays a key role in understanding the performance and behaviour of linear classifier learning algorithms. The definition of the margin for a single example is the minimum Euclidean distance between an instance vector $\mathbf{x}$ and the separating hyperplane produced by $\mathbf{w}$. Formally, the margin of a weight vector on an example $\mathrm{z_i}$ is $\gamma_{\mathrm{z_i}}(\mathbf{w}) := y_i(\mathbf{w} \cdot \mathbf{x}_i)/\|\mathbf{w}\|_2$, and the margin with respect to the whole training sequence is $\gamma_{\mathbf{z}}(\mathbf{w}) := \min\{\gamma_{\mathrm{z_i}}(\mathbf{w}) \colon i \in \{1, \ldots, T\}\}$. To illustrate the concept of the margin and the maximum margin we consider the example of Figure 1.1 for a two class classification problem that having a hyperplane solution which divides the two classes with a large margin. Typically a large margin hyperplane will have better generalisation (see Section 1.6) to as yet unseen examples, and better robustness to the presence of noise in the training set.

A learning algorithm that attains zero training error finds an hypothesis or target function $\mathbf{u}$ which is said to be in the *version space*. The version space $V(\mathbf{z})$ is the set of hypotheses consistent with the training sample. In the case of linear classifiers, we consider $V(\mathbf{z})$ to be a subset of the weight space:

$$V(\mathbf{z}) := \{\mathbf{w} \in \mathbb{R}^d \colon \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x}_i) = y_i \text{ for all } (\mathbf{x}_i, y_i) \in \mathbf{z}\}. \tag{1.4}$$

An alternative approach is the associated regression problem, where a prediction $f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x})$ of the given y is made. The general measure of performance used in regression is the *mean squared error* (MSE) given by $\mathrm{E_z}\left[(y - (\mathbf{w} \cdot \mathbf{x}))^2\right]$. The objective

is find the $\mathbf{w}^*$ to minimise the MSE defined by

$$\mathbf{w}^* := \text{argmin}_{\mathbf{w}} \mathrm{E}_{\mathbf{x},y} \left[(y - (\mathbf{w} \cdot \mathbf{x}))^2 \right]. \tag{1.5}$$

It is not obvious whether regression or classification are better at producing error free discrimination. Figure 1.2 shows a situation where classification is better as minimising the squared distance will result in misclassification. We see in Figure 1.2 that the hyperplane produced by classification correctly classifies the examples, whereas the regression solution tries to minimise the distance between the two classes resulting in a solution which misclassifies two examples.

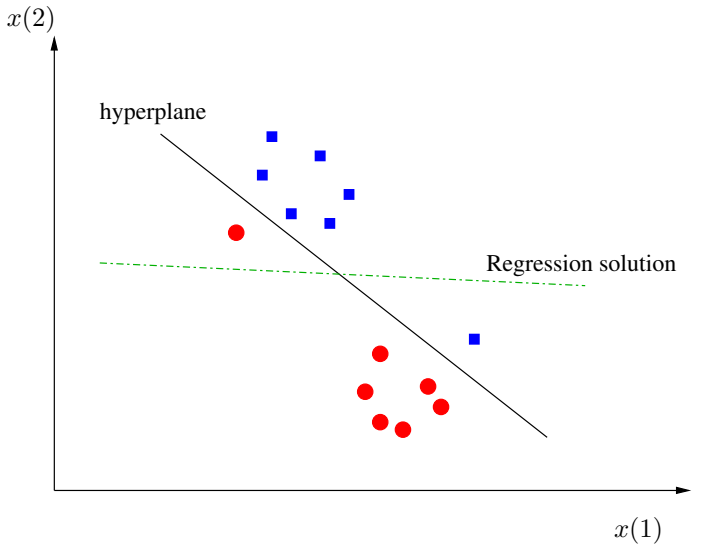

Figure 1.2: Example scenario where classification is better than regression.

We have considered measures which are used in the assessment of whether a classifier or predictor produces a low probability of error. For our learning algorithm to have good generalisation it must also take into account the number of samples required to achieve its hypothesis. This measure of performance is referred to as the *sample* complexity of our algorithm. Most commonly the sample complexity of a learning algorithm is explained with the use of VC Theory (Vapnik and Chervonenkis [1971], Vapnik [1995]), in particular the use of the VC dimension d_{VC} of the hypothesis (function) class $\mathcal{H}$, i.e. $h \in \mathcal{H}$. The VC dimension crudely speaking measures how many training examples can be separated (shattered) for all possible labelings of the hypothesis class $\mathcal{H}$. We will present in Section 1.6 the connection between the VC dimension, the expected risk, and the *margin*. Another important measure of performance of a learning algorithm is its computational complexity, which measures of the amount of computational effort required to determine the algorithm's final hypothesis.

1.6 The Perceptron and large margin variants

In this section we consider, for ease of discussion, a hybrid of the standard Perceptron of Rosenblatt [1958] and the marginalised Perceptron of Duda et al. [2000], which we will refer to as the generalised Perceptron (see Algorithm 1). The algorithm differs from the standard Perceptron algorithm in two ways: 1) an update is made on example z_k if $\gamma_{z_k}(\mathbf{w}_k) \leq \rho_k$, where ρ_k is the *update margin* ($\rho_k = 0$ for all k in the standard Perceptron); and 2) the update step size is η_k instead of always being fixed (often it is fixed to 1). When $\rho_k = 0$, and η_k is fixed, it is known by Theorem 1.6.1 of Novikoff [1962] and Block [1962] that the standard Perceptron algorithm makes no more than $(\beta/\gamma_{\mathbf{z}})^2$ updates, where $\beta := \max_{i=1,\ldots,T} \|\mathbf{x}_i\|_2$ (we assume that the instances, throughout the thesis, are normalised unless otherwise stated, so $\beta = 1$), and $\gamma_{\mathbf{z}}$ is the margin of the maximum margin hyperplane (for example see Figure 1.1) defined by $\gamma_{\mathbf{z}} := \max\{\gamma_{\mathbf{z}}(\mathbf{w}) : \mathbf{w} \in \mathbb{R}^d\}$. We have included here the proof of Duda et al. [2000] of the Perceptron convergence for completeness, and because of its fundamental importance to a number of approaches presented in this thesis.

Theorem 1.6.1. (Novikoff [1962], Block [1962]) *Let $\boldsymbol{z} = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T))$ be a sequence of labelled examples, and let $\beta = \max_{\boldsymbol{x} \in \boldsymbol{z}} \|\boldsymbol{x}\|$. Assume there is a Perceptron weight $\boldsymbol{w}^*$ such that $\gamma_{\boldsymbol{z}} = \min_{(\boldsymbol{x},y)\in\boldsymbol{z}} y\,(\boldsymbol{w}^* \cdot \boldsymbol{x})\,/\|\boldsymbol{w}^*\| \geq 0$ for all $(\boldsymbol{x}, y) \in \boldsymbol{z}$. The number of updates M of the Perceptron algorithm is less than or equal to*

$$\frac{\beta^2 \|\boldsymbol{w}^*\|^2}{\gamma_{\boldsymbol{z}}^2}.$$

Proof. The Euclidean distance between the Perceptron weight after update k, at example t, with a scaled version of the solution $\mathbf{w}^*$ is

$$\|\mathbf{w}_t - \alpha\mathbf{w}^*\|^2 = \|\mathbf{w}_{t-1} - \alpha\mathbf{w}^*\|^2 + 2\left((\mathbf{w}_{t-1} - \alpha\mathbf{w}^*) \cdot \mathbf{x}_t\right) y_t + \|\mathbf{x}_t\|^2.$$

where $\alpha \in \mathbb{R}^+$ is a constant scaling and $\mathbf{w}_t$ is the Perceptron weight after update k.

Using the fact that $\mathbf{w}_{t-1}\mathbf{x}_t y_t \leq 0$ (given the Perceptron at iteration t, example t, made a update) we get

$$\|\mathbf{w}_t - \alpha\mathbf{w}^*\|^2 \leq \|\mathbf{w}_{t-1} - \alpha\mathbf{w}^*\|^2 - 2\alpha\left(\mathbf{w}^* \cdot \mathbf{x}_t\right) y_t + \|\mathbf{x}_t\|^2.$$

By setting $\alpha = \beta^2/\gamma_{\mathbf{z}}$, and using $\|\mathbf{x}_t\|^2 \leq \beta^2$ and $\gamma_{\mathbf{z}} = (\mathbf{w}^* \cdot \mathbf{x}_t)\, y_t$ the above inequality becomes

$$\|\mathbf{w}_t - \alpha\mathbf{w}^*\|^2 \leq \|\mathbf{w}_{t-1} - \alpha\mathbf{w}^*\|^2 - \beta^2.$$

Since $\|\mathbf{w}_t - \alpha\mathbf{w}^*\|^2 \geq 0$ for all k, this decrease may occur at most $M = \frac{\|\mathbf{w}_0 - \alpha\mathbf{w}^*\|^2}{\beta^2}$ times. The proof is complete by substituting the fact that all the elements of $\mathbf{w}_0$ are zero. □

Theorem 1.6.1 guarantees that for separable problems the Perceptron's hypothesis

Algorithm 1 Generalised Perceptron algorithm

Require: A linear separable training sequence $\mathbf{z} = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_T, y_T))$.
Require: Margin parameter sequence $\boldsymbol{\rho}_k$ and learning rate $\boldsymbol{\eta}_k$.

1: $t := 0,\ k := 0,\ \mathbf{w}_0 := \mathbf{0}$.
2: **repeat**
3: $\quad t := t + 1$
4: $\quad$ **if** $y_t (\mathbf{w}_{t-1} \cdot \mathbf{x}_t) / \|\mathbf{w}_{t-1}\|_2 \leq \rho_k$ **then**
5: $\quad\quad \mathbf{w}_t := \mathbf{w}_{t-1} + \eta_t y_t \mathbf{x}_t$
6: $\quad\quad k := k + 1$
7: $\quad$ **else**
8: $\quad\quad \mathbf{w}_t := \mathbf{w}_{t-1}$
9: $\quad$ **end if**
10: **until** No more updates are made
11: **return** $\mathbf{w}_{\text{perc}} := \mathbf{w}_t$

$\mathbf{w}_{\text{perc}}$ is in $V(\mathbf{z})$ after a finite number of updates. The Perceptron algorithm has two other key advantages besides a finite number of updates: it is extremely simple, and it is online. In the case where the classification problem is not linearly separable, and therefore Theorem 1.6.1 does not hold. We can consider a nonlinear mapping $\phi(\mathbf{x})$ of our instances, which is in a very high dimensional space such that the classification problem is linearly separable in this extended space. Using the *kernel trick* one can express the mappings in terms of inner products, resulting in a kernel $K(\mathbf{x}_j, \mathbf{x}_k) = (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}))$. One can simply apply the kernel trick to Algorithm 1 by replacing $(\mathbf{w}_{t-1} \cdot \mathbf{x}_t)$ with $\sum_{j=1}^{k-1} y_j K(\mathbf{x}_j, \mathbf{x}_t)$ in line 4, and storing $(\mathbf{x}_j, y_j)$ each time there is an update, therefore replacing the need for storing $\mathbf{w}$.

Whilst the standard Perceptron algorithm has several advantages we are only guaranteed that the margin of its final hypothesis is $\gamma_{\mathbf{z}}(\mathbf{w}_{\text{perc}}) > 0$. To guarantee a larger margin for $\mathbf{w}_{\text{perc}}$ motivates the marginalised Perceptron (Duda et al. [2000]); if ρ_k is a fixed positive constant then the final margin produced by running Algorithm 1 until separation of the training sequence in a batch setting, can be guaranteed to be $\geq \rho_k$. The motivation for setting $\rho_k = \rho > 0$ in order to guarantee $\gamma_{\mathbf{z}}(\mathbf{w}) > \rho$ for the final solution $\mathbf{w}$ is that a larger margin hyperplane has a better guarantee of generalisation performance than a smaller margin one. Note that if we set ρ too large, i.e. $\rho > \gamma_z$, then Algorithm 1 will not converge.

We now give some intuition to why maximising the margin might be a good thing to do, especially in terms of improving generalisation performance of a linear classifier. Consider the linear classifier $h(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b$ in the batch learning setting, where we include the bias term b to maintain the standard form used by other authors when discussing maximising the margin. In fact before we can express a bound on the expected risk of (1.3), or in other words a bound on the generalisation error, we need to make the connection between the VC dimension and the margin. It has been shown by Vapnik

[1995] that training examples within a ball of radius R satisfy the inequalities

$$\|\mathbf{w}\|_2 \leq \rho, \tag{1.6}$$

and

$$d_{VC} \leq \rho^2 R^2 + 1. \tag{1.7}$$

Therefore, the expected risk for a hyperplane satisfying $y_t\left((\mathbf{w}\cdot\mathbf{x})+b\right) \geq 1$ for $t = 1,\ldots,T$, and inequalities (1.6) and (1.7), is bounded from above with probability $1-\delta$ by

$$R(h,\mathrm{P_z}) \leq R_{\mathrm{emp}}(h,\mathbf{z}) + O\left(\sqrt{\frac{R^2\rho^2}{T}\left(\log_2\left(\frac{T}{R^2\rho^2}\right)+1\right)+\frac{\log_2(1/\delta)}{T}}\right). \tag{1.8}$$

One way to minimise the bound of (1.8), therefore guaranteeing a lower expected risk is to perform the optimisation

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2 \tag{1.9}$$

subject to $y_t\left((\mathbf{w}\cdot\mathbf{x})+b\right) \geq 1$ for $t = 1,\ldots,T$. By minimising $\|\mathbf{w}\|_2^2$ (using quadratic programming) we ensure that the complexity term of (1.9) is minimised, and the constraints of (1.9) ensure zero empirical risk. The optimisation outlined in (1.9) is referred to as maximising the margin. The introduction of *slack variables* to the optimisation of (1.9) by Cortes and Vapnik [1995] resulted in what is termed the *support vector machine* (SVM). The SVM optimisation is

$$\min_{\mathbf{w},b,\zeta} \|\mathbf{w}\|_2^2 + C\sum_{j=1}^{T}\zeta_j \tag{1.10}$$

subject to $y_t\left((\mathbf{w}\cdot\mathbf{x})+b\right) \geq 1-\zeta_t$ for $t = 1,\ldots,T$, where $C > 0$ is the regularisation constant used to provide a trade-off between the complexity term and the empirical error.

There have been a number of other interesting batch learning approaches to produce a hyperplane with a large margin. Most notable is the MINOVER trick of Krauth and Mézard [1987], where they proposed training the Perceptron algorithm with examples selected according to $(\mathbf{x}_t, y_t) = \mathrm{argmin}_{(\mathbf{x},y)\in\mathbf{z}} y\left(\mathbf{w}_{t-1}\cdot\mathbf{x}\right)$ in an endeavour to maximise the margin. Several subsequent methods with approaches similar to MINOVER have been proposed (see for example Watkin and Rau [1992], Park and Hu [1996]), but they either require the algorithm to store all of the examples seen so far, or require the algorithm to have seen all the examples beforehand, and thus are essentially batch learning algorithms.

Whilst such analyses and guarantees of a large margin hyperplane are generally only shown in the batch setting, the idea of seeking a large margin hyperplane with an online algorithm still makes sense for two reasons. First, one can always use an

online algorithm to learn in the batch setting by repeatedly iterating over the sample $\boldsymbol{z}$. Second, even in a truly online setting, a large margin solution provides some immunity to attribute noise and concept drift.

There have been a few recent attempts to further develop online algorithms that achieve an approximation to the maximum margin. Kivinen et al. [2002] studied the marginalised Perceptron (and issues arising when it is kernelised). Li and Long [2002] studied an algorithm they called *relaxed online maximum margin algorithm* (ROMMA) where if there is a mistake at the tth trial then $\mathbf{w}_t$ is the smallest element of the constrained set of $\left\{\mathbf{w} : \mathbf{w}_{t-1} \cdot \mathbf{w} \geq \|\mathbf{w}_{t-1}\|_2^2\right\} \cap \{\mathbf{w} : y_t(\mathbf{w} \cdot \mathbf{x}_t \geq 1)\}$, else $\mathbf{w}_{t+1} = \mathbf{w}_t$. It has a similar mistake bound to the Perceptron and is computationally only slightly more costly than the Perceptron. Gentile [2001] has presented an *approximately large margin algorithm* (ALMA)[2] which he analysed in a batch setting showing that for any $\delta \in (0, 1)$ it can achieve a margin of at least $(1 - \delta)\gamma_{\boldsymbol{z}}$ (in the linearly separable case), requiring $O(1/(\delta^2 {\gamma_{\boldsymbol{z}}}^2))$ updates. ALMA is obtained from the generalised Perceptron by setting $\rho_k = (1 - \delta)B/\sqrt{k}$ and $\eta_k = C/\sqrt{k}$ and, if necessary, re-normalising $\mathbf{w}_k$ so that $\|\mathbf{w}_k\|_2 \leq 1$.

1.7 The Bayes point machine

Whilst the maximum margin hyperplane provides a good guarantee of generalisation performance, it is not unique in this regard; indeed, it is generally not the hyperplane with the best generalisation performance, the so-called Bayes point (Ruján [1997]). Here we discuss what the Bayes point is and some methods to estimate it.

1.7.1 The Bayes point

Consider a fixed class $\mathcal{H}$ of classifiers and a sequence of T training examples $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)))$. We would like to find a classifier from $\mathcal{H}$ which correctly classifies future examples drawn i.i.d. from the same distribution as $(\mathbf{x}_i, y_i)$. The Bayes optimal classifier chooses the label that minimises the probability of error, given the data $\mathbf{z}$. In general, the Bayes optimal classifier itself is not in $\mathcal{H}$ and may be very difficult to evaluate even if all the probability distributions are known. The Bayes point is the hypothesis from $\mathcal{H}$ which minimises the probability of error (Ruján [1997]).

Definition 1.7.1. (Bayes point) *Let $\mathcal{H}$ be a fixed class of classifiers, $l : \mathcal{Y} \times \mathcal{Y} \to \Re$ a loss function, c a function chosen from the class $\mathcal{H}$ according to the posteriori $\mathrm{P}_{\mathcal{H}}$, then the Bayes point is*

$$\mathrm{argmin}_{h \in \mathcal{H}} \mathrm{E}_{\boldsymbol{x}} \left[\mathrm{E}_{c \sim \mathrm{P}_{\mathcal{H}}} \left[l\left(h(\boldsymbol{x}), c(\boldsymbol{x})\right) \right] \right].$$

[2] We only consider the 2-norm case of ALMA. ALMA is a more general algorithm for p-norm classifiers.

Formally, the Bayes point is defined by Definition 1.7.1, noting that the loss function l is in general the zero-one loss function, as given by (1.1). For linear classification the Bayes point is thus given by the weight vector that minimises the probability of a classification error. This is still very difficult to find in general, thus motivating the use of approximations.

1.7.2 Estimating the Bayes point

Algorithms which estimate the Bayes point $\mathbf{w}_{\mathrm{BP}}$ are referred to as *Bayes Point Machines* (BPMs)(Herbrich et al. [2001]). The Bayes Point is approximated by $\mathbf{w}_{\mathrm{cm}}$ the centre of mass of $\mathbf{w}$ drawn from $\mathcal{H}$ according to the posterior distribution. The BPM estimate of $\mathbf{w}_{\mathrm{cm}}$, for the linear classifier is

$$\tilde{\mathbf{w}}_t := \frac{1}{N}\sum_{i=1}^{N}\mathbf{w}_{t_i}, \tag{1.11}$$

where $\mathbf{w}_1, \ldots, \mathbf{w}_N$ are N different linear classifier solutions to $\mathbf{z}$.

As an illustration of the concepts related to estimating the Bayes point, consider the example of Figure 1.3 (a modification of the example presented in Herbrich [2002] and Herbrich et al. [2001]). We imagine that we are looking from above (see Figure 1.3) at the surface of a sphere in weight space, such that $\mathbf{w} \in \mathbb{R}^3$ and $\|\mathbf{w}\| = 1$, which we have squashed into two dimensions. The training set $\mathbf{z}$ consists of five examples, $t = 5$, each example $(\mathbf{x}, y)$ defines a plane which cuts the sphere, which is represented in two dimensions by the half space $\{\mathbf{w} : y\mathbf{w}\cdot\mathbf{x} \geq 0\}$. The weight space defined by the intersection of half spaces is the version space $V(\mathbf{z})$, hence $V(\mathbf{z})$ defines the space of all weight vectors which correctly classify the five examples.

Now consider the significance of the margin ρ with respect to unseen examples which are drawn according to the distribution of examples $\mathrm{P}_{\mathbf{z}}$. If the half space for the unseen examples is now defined in terms of ρ, i.e. $\{\mathbf{w} : y\mathbf{w}\cdot\mathbf{x} > \rho\}$ (as is discussed in Section 1.6, though this discussion is in context the of general linear classifiers and not restricted to the Perceptron algorithm) then we get the dotted region of Figure 1.3. The $\mathbf{w}_{\mathrm{cm}}$ is the centre of mass of this dotted region [3], and has a margin greater than ρ, providing an ability for $\mathbf{w}_{\mathrm{cm}}$ to handle noise in the training examples and unseen examples. As the number of training examples $T \to \infty$, and if the instances $\mathbf{x}$ are Gaussian distributed according to $1/\pi^{\frac{d}{2}} \exp\left(-\|\mathbf{x}\|^2\right)$ then $\mathbf{w}_{\mathrm{cm}}$ approaches $\mathbf{w}_{\mathrm{BP}}$ (Herbrich et al. [2001]), where one can imagine the version space is defined by the curved line [4].

There have been a number of Bayes point machine algorithms developed (in the batch setting): the Perceptron based algorithm of Watkin [1993] which ran on different permutations of the training set, the billiard algorithm of Ruján [1997], and later the

[3] Note that this may not correspond to the geometric centre as illustrated in Figure 1.3 unless the posterior of classifier weights given the training sample are uniform.

[4] Strictly this is true if $\mathbf{w}_{\mathrm{cm}}$ is formed according to the distribution of classifier weights in $V(\mathbf{z})$.

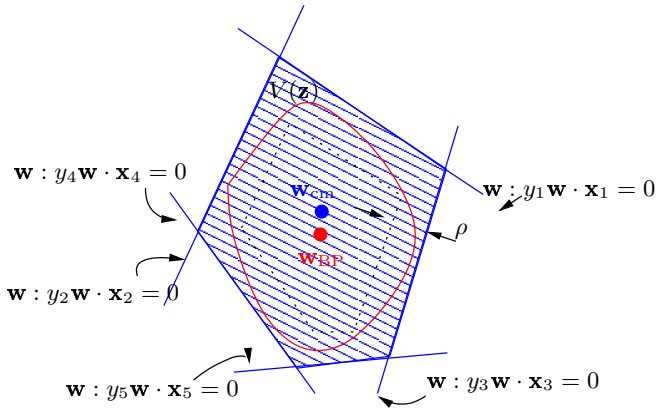

Figure 1.3: Illustration of the Bayes point for linear classifiers $\mathbf{w} \in \mathbb{R}^3$ trained on five examples (defining the planes which cut the surface of a sphere). We are looking at the surface squashed onto a two dimensional space.

kernelised billiard algorithm of Herbrich et al. [2001].

1.8 Aggregate, ensemble and voting methods

We have discussed several methods for improving the generalisation performance of classifiers i.e. large margin classifiers and Bayes point machines. In this section we discuss various methods which improve generalisation by forming a single hypothesis which is a weighted combination of several classifier hypotheses. These methods can be referred to as *aggregate*, *ensemble*, and *voting* methods, though the names seem to be quite interchangeable. We make a clear distinction between the methods here, which combine hypotheses, and the method of the Bayes point machine of the last chapter, that combine the classifier weights and not their associated hypotheses.

1.8.1 Bagging and the voted-Perceptron

One of the most famous aggregating method is Bagging by Breiman [1996], an acronym for *B*ootstrap *agg*regating. As the name suggests, Bagging performs bootstrapping on N different classifiers by training each classifier on different subsets of the training data set and producing an output hypothesis which is an average of them. Bootstrapping is a well known statistical technique (see Efron and Tibshirani [1993], Shao and Tu [1995]) where given a training sample of size T one creates new training sample by resampling the original training sequence by drawing with replacement multiple times (usually T). The performance gain of the average classifier's hypothesis, produced by Bagging, is greatest when the differences in performance amongst the individual boostrapped classifiers is greatest. Several authors have demonstrated this variance reduction property of Bagging: Bauer and Kohavi [1999], Derbeko et al. [2002]. Most recently, Valentini and Dietterich [2003] discussed some pertinent issues regarding Bagging: they stated that

Algorithm 2 voted-Perceptron algorithm (Freund and Schapire [1999])

Require: A training sequence $\mathbf{z} = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_T, y_T))$.

1: Initialise $k := 0$, $\mathbf{w}_1 := 0$, $v_1 := 0$,
2: **for** $t = 1$ to T **do**
3: $\quad \hat{y} := \text{sgn}(\mathbf{w}_k \cdot \mathbf{x}_t)$
4: $\quad$ **if** $\hat{y} = y_t$ **then**
5: $\quad\quad v_k := v_k + 1$
6: $\quad$ **else**
7: $\quad\quad \mathbf{w}_{k+1} := \mathbf{w}_k + y_t \mathbf{x}_t$
8: $\quad\quad v_{k+1} := 1$
9: $\quad\quad k := k + 1$
10: $\quad$ **end if**
11: **end for**
12: **return** $H(\mathbf{x}) := \text{sgn}\left(\sum_{i=1}^{k} v_i \text{sgn}(\mathbf{w}_i \cdot \mathbf{x})\right)$

whilst variance reduction is important, ensuring the final hypothesis produced is not biased is equally as important. There have been several online variants of Bagging (Fern and Givan [2000] and Oza [2001]) the details of which are in Chapter 3.

Several other learning strategies have been suggested in combining the hypotheses other than allocating an equal weighting to each, as is the case for Bagging. One such strategy is the *pocket* algorithm (Gallant [1986]), where the hypothesis during learning which performs the best is kept aside in "one's pocket" and is used as the current output hypothesis. In the spirit of this algorithm is the voted Perceptron algorithm (see Algorithm 2) of Freund and Schapire [1999] where each time the Perceptron is updated (lines 7-9) we store both the hypothesis defined by $\mathbf{w}_k$ and the number of training examples it classified correctly v_k. The final hypothesis of the voted Perceptron $H(\mathbf{x})$ (line 12) is a weighted sum of all the hypothesis, with each hypotheses being scaled by the number of correctly classified examples.

1.8.2 Boosting

Other than Bagging, and the voted-Perceptron, another method used in the combining hypotheses to improve generalisation is *Boosting* (Freund and Schapire [1996]). The most well known version of Boosting is the batch method of Adaboost (Freund and Schapire [1996]). Each iteration i of Adaboost trains a new hypothesis h_i (sometimes called the base model) using the entire training set. At each iteration i in Adaboost after training the h_i a weight is assigned to each example in the training set which is used construct a distribution of examples. When training a new hypothesis with a particular example at the the next iteration $i + 1$ we use this distribution, giving more weight to those examples which the last hypothesis misclassified during the last iteration and less to those it classified correctly. Given that each hypothesis $i = 1, \ldots, N$ has an error

$\epsilon_i < \frac{1}{2}$ over the entire training set, the final hypothesis of Adaboost has been shown to have error bounded from above by $2^N \prod_{i=1}^{N} \sqrt{\epsilon_i(1-\epsilon_i)}$. There are some online versions of Boosting: online Arc-x4 Breiman [1998] and the algorithm of Oza [2001]. Both these online boosting algorithms do not have the same performance guarantees of Adaboost but have been shown empirically to have good generalisation performance.

Whilst Boosting increases the generalisation performance it has some limitations when the data set is "noisy" because it can "boost" the noisy examples. There have been several boosting methods, in the batch learning setting, which try to address this issue of boosting noisy examples (see Meir and Rätsch [2003]).

1.9 Tracking experts

In Boosting (from the previous section) each classifier may perform poorly, but the ensemble of classifiers performs well. In that sense the ensemble has better generalisation than the individual classifier on its own. The method we describe in this section refers to each classifier as an *expert*, and we make no assumptions about whether each expert has good generalisation performance already. In fact we consider the regression setting where we want a find the *mixture of experts* which minimises our prediction error. In the linear regression case of the mixture of experts each expert i forms a prediction $\hat{y}_k(i) := (\mathbf{w}_i \cdot \mathbf{x}_k)$ when given the training example $\mathrm{z}_k = (\mathbf{x}_k, y_k)$. For each example z_k we wish to determine the weight $v_k^m(i)$ for each expert $i = 1, \ldots, M$ such that we give more weight to those experts which perform well (lowest prediction loss). We ensure that $\sum_{i=1}^{M} v_k^m(i) = 1$. The output prediction produced by the mixture of experts is

$$\hat{y}_k^m := \sum_{i=1}^{M} v_k^m(i)\hat{y}_k(i).$$

The mixture of experts algorithm called the *weighted average* algorithm (Haussler et al. [1998]), updates the mixture weight for expert i, $v_t^m(i)$, in such a way that it guarantees that the total prediction loss of the algorithm is not much worse than that of the best expert over the entire training sequence. We are primarily interested in the variants of *weighted average* algorithm in this thesis (see Chapter 6) where it is considered that there may not be a single expert which is best over the entire training sequence, but rather the best expert, or mixture of experts, changes over time. In particular we are interested in the *fixed-share* algorithm of Herbster and Warmuth [1998] which tries to track the mixture of experts with the lowest prediction loss when the best expert, or mixture of experts, changes over time.

1.10 Thesis outline

So far we have discussed existing methods for improving generalisation performance in classification and regression, for both batch and online learning algorithms. While most of the background material required for the rest of the thesis has been given in this chapter, some of this material will be repeated later, so that each chapter is more self-contained, making it possible to read each chapter independently.

The rest of this section gives the outline of the remainder of the thesis.

In Chapter 2 an online variant of the Bayes point machine is presented, the online Bayes point machine (OBPM). We analyse this new algorithm, its speed of convergence and how various parameter settings affect it.

In Chapter 3 we apply the voted-Perceptron algorithm to the OBPM algorithm from Chapter 2, which we call the OBPM-voted algorithm. An empirical comparison is made between the OBPM-voted algorithm using the Perceptron and various online voting algorithms, i.e. the online Bagging algorithm.

In Chapter 4 we focus on the application of the online methods of Chapters 2 and 3 to the problem of ranking instances. We incorporate the ideas behind the OBPM into the Perceptron ranking algorithm, PRank (Crammer and Singer [2002]).

In Chapter 5 we investigate online learning implementations of Bagging algorithm and the OBPM algorithm applied to the signal processing problem of channel equalization.

In Chapter 6 we look at the mixture of experts algorithms which are able to achieve a total loss close to the total loss of the best expert over a sequence of examples. We introduce a new algorithm which allows the rate of switching between experts to vary. We present a number of application scenarios in the area of channel equalization.

Chapter 7 concludes and contains a summary of the key contributions of this thesis.

Online Bayes Point Machine

2.1 Introduction

An online classifier is formed using the example sequence seen up to a given point in time. In contrast a batch classifier is formed based on the entire sequence of T examples. There have been a number of recent online learning classification algorithms (Gentile [2001], Li and Long [2002], Kivinen et al. [2002]) which attempt to replicate the performance of batch learning methods like the support vector machine (SVM) which explicitly maximise the margin. This chapter presents an online algorithm for classification which achieves a large margin by estimating the centre-of-mass (the so-called Bayes point) by a randomisation trick. The concept of the Bayes point was first developed for the Perceptron by Watkin [1993]. More recently Herbrich et al. [2001] presented a kernel method to approximate the Bayes point, which was called the Bayes Point Machine (BPM). In Herbrich et al. [2001] it was shown empirically that the BPM had better generalisation performance compared to the hard margin optimisation techniques of SVMs. The algorithm presented here is an online variant of the BPM, and is called the Online Bayes Point Machine (OBPM).

2.2 The Algorithm

Whilst the variants on the classical Perceptron discussed in Section 1.6 can guarantee convergence to a hyperplane with a large margin, there is a price to pay. The closer one desires the algorithm's hypothesis margin to be to the maximal possible margin, the slower the convergence. In addition, there are several extra parameters one has to choose *a priori*. This suggests it is worthwhile exploring alternate variants on classical Perceptrons which do not maximise the margin directly.

Our starting point for such variants is the observation that (1.4) defines a *convex* set $V(\mathbf{z})$. Thus if one found a number of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_N$ in $V(\mathbf{z})$, one could optimise over the set of convex combinations $C^N := C^N(\mathbf{w}_1, \ldots, \mathbf{w}_N) := \{\sum_i \alpha_i \mathbf{w}_i \colon \|\boldsymbol{\alpha}\|_1 = 1, \boldsymbol{\alpha} \geq \mathbf{0}\}$ of them. Intuitively one would expect that the more diverse $\mathbf{w}_1, \ldots, \mathbf{w}_N$ are, the greater the proportion of $V(\mathbf{z})$ could be covered by C^N. The centre of C^N provides an approximation of the Bayes point and thus potentially

Algorithm 3 OBPM algorithm

Require: A training sample $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: An online learning algorithm with update rule $\mathcal{L}$ for linear discrimination and associated step-size update rule $\mathcal{S}$.

Require: A subroutine Bernoulli(p) which returns independent Bernoulli random variables with probability p of taking the value 1.

Require: Parameters $N \in \mathbb{N}$ and $\tau \in [0, 1]$.

Initialise step sizes $\eta_{j,1}$, for all $j \in \{1, \ldots, N\}$.
Initialise weights $\mathbf{w}_{j,0} := \mathbf{0}$, for all $j \in \{1, \ldots, N\}$.
for $t = 1$ to T **do**
  for $j = 1$ to N **do**
    $b_{j,t} := \text{Bernoulli}(\tau)$
    if $b_{j,k} = 1$ **then**
      $\mathbf{w}_{j,t} := \mathcal{L}(\mathbf{w}_{j,t-1}, \eta_{j,t}, \mathbf{x}_t, y_t)$
    else
      $\mathbf{w}_{j,t} := \mathbf{w}_{j,t-1}$
    end if
    $\eta_{j,t+1} := \mathcal{S}(t, \eta_{j,t}, \ldots)$
  end for
  $\tilde{\mathbf{w}}'_t := \frac{1}{N} \sum_{j=1}^{N} \mathbf{w}_{j,t}$
  $\tilde{\mathbf{w}}_t := \tilde{\mathbf{w}}'_t / \max\{1, \|\tilde{\mathbf{w}}'_t\|_2\}$
end for
return $\tilde{\mathbf{w}}_T$

better performance (dependent on the closeness to the true Bayes point). This convex combination of *weight vectors* $\mathbf{w}_j$ is different to hypothesis aggregation methods such as Bagging (Breiman [1996]) and boosting (Freund and Schapire [1996]) which forms convex combinations of classifier *hypotheses* $\mathbf{x} \mapsto \text{sgn}(\mathbf{w}_j \cdot \mathbf{x})$. Several variants of online hypothesis aggregation methods, such as Bagging, are discussed in more detail in the next chapter.

If $\mathbf{w}_1, \ldots, \mathbf{w}_N$ are all identical, C^N is a singleton and nothing is gained. A well known technique for achieving diversity is to generate $\mathbf{w}_1, \ldots, \mathbf{w}_N$ by running (a suitable variant of) the Perceptron algorithm on different permutations $\pi(\mathbf{z}) := (\mathrm{z}_{\pi(1)}, \ldots, \mathrm{z}_{\pi(T)})$ of the training sample $\mathbf{z}$ (Watkin [1993] and Herbrich et al. [2001]), where $\pi : \{1, \ldots, T\} \to \{1, \ldots, T\}$ is a permutation. Although an elegant and effective trick, it is not an online algorithm — one needs the entire training sample $\mathbf{z}$ before starting. This motivates the algorithm we study here: the *online Bayes point machine* (OBPM) (Algorithm 3). Given a training sequence $\mathbf{z}$, we run N Perceptrons in parallel and ensure diversity of their final solutions by randomly choosing to present a given sample z_t to Perceptron j only if $b_{j,t} = 1$, where $b_{j,t}$, $j = 1, \ldots, N$, $t = 1, 2, \ldots$ are independent Bernoulli random variables with $\Pr\{b_{j,t} = 1\} = \tau$.

There are some theoretical reasons for expecting that the refined optimisation of α_i would lead to better solutions than naively setting them all equal to $1/N$. However optimisation of α_i can only occur in the batch setting: one cannot determine the margin of a candidate weight vector without the whole training sample. One method that was tried to improve the choice of α_i (in the batch setting) was to maximise the following lower bound of the margin of $\tilde{\mathbf{w}}$ (see Appendix A.1 for proof):

$$\gamma_{\mathbf{z}}(\tilde{\mathbf{w}}) \geq \frac{\sum_{j=1}^{N} \alpha_j \gamma_{\mathbf{z}}(\mathbf{w}_j)}{\sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \left(\mathbf{w}_i \cdot \mathbf{w}_j\right)}}. \tag{2.1}$$

It was found by experimentation that maximising the bound of (2.1) had no significant performance gain compared to setting $\alpha_i = 1/N$. In the next chapter we consider online voting methods which use settings other than $\alpha_i = 1/N$, but for the rest of this chapter we will only consider the situation where $\alpha_i = 1/N$ for all i.

On average each Perceptron j sees only a fraction τ of all the examples in $\boldsymbol{z}$. Seeing a subset can be an advantage when the data set is large and is possibly noisy. There is a natural trade-off in selecting τ: the smaller we set τ the more diverse the solutions $\mathbf{w}_j$ become, but the algorithm takes longer to reach these solutions.

In general we take $\mathcal{L}$, the base learner, to be the Perceptron update rule, fixing $\eta_{j,t} = 1$ and $\rho = 0$. This leaves the OBPM algorithm two parameters required for tuning: τ, the probability of seeing a training example at iteration t; and N, the number of Perceptrons. When the update rule $\mathcal{L}$ of Algorithm 3 is the standard Perceptron ($\rho = 0$), each Perceptron $j = 1, \ldots, N$ makes an mistake whenever $y_t \left(\mathbf{w}_{j,t-1} \cdot \mathbf{x}_t\right) \leq 0$, the indicator of this event is denoted $\sigma(\mathbf{w}_{j,t-1}, \mathbf{x}_t, y_t)$. The function $\sigma(\mathbf{w}_{j,t-1}, \mathbf{x}_t, y_t)$ is one when Perceptron j makes an mistake at iteration t, and zero when it does not make an mistake. For ease of notation $\sigma_{j,t}$ is used to represent $\sigma(\mathbf{w}_{j,t-1}, \mathbf{x}_t, y_t)$ for the rest of the chapter.

We see that the number of arithmetic operations for the OBPM algorithm on average is $O(\tau N dt)$; for both the ALMA algorithm and the Perceptron algorithm it is $O(dt)$. The OBPM algorithm is a factor τN more expensive computationally than ALMA algorithm. Each Perceptron used in the OBPM algorithm is trained independently; therefore the OBPM algorithm can be readily implemented using parallel computing.

We further note that step 14 of Algorithm 3 is optional as it only bounds $\|\tilde{\mathbf{w}}\|$ by one and given the instances are normalised, making the choice of ρ in the range $0 \leq \rho \leq 1$ in the generalised Perceptron Algorithm 1.

2.2.1 OBPM kernel implementation

Using a linear classifier is in general not the best choice when the classification problem is not linearly separable. We use the standard trick of extending the dimension of the feature space in such a way as to try to convert the non-separable classification problem to a separable one. An extended feature space can be achieved with the Perceptron by

applying the kernel trick, where the kernel is defined by $K(\mathbf{x}_j, \mathbf{x}_k) = (\phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}_k))$ and $\phi(\mathbf{x})$ is the feature mapping. Applying a kernel will not guarantee separability in all cases, but is used in general as an approach to creating separability.

A kernel implementation of the OBPM algorithm can be derived by first expressing each Perceptron $j = 1, \ldots, N$, using the representer Theorem (Schölkopf et al. [2001]), as $f_j(\cdot) = \sum_{t=1}^{T} \Upsilon_{j,t} K(\mathbf{x}_t, \cdot)$, where $\Upsilon_{j,t} = b_{j,t}\sigma_{j,t}y_t$. Combining with the reproducing property results in $\tilde{f}(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N} f_j(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T} \Upsilon_{j,t} K(\mathbf{x}_t, \mathbf{x})$. The kernel implementation of the OBPM algorithm may be made more efficient by noting that the kernel values of the Gram matrix $K(\mathbf{x}_j, \mathbf{x}_t)$ are the same for all N Perceptrons. Given that the Gram matrix is the same for all the Perceptrons the final solution of Algorithm 3 can be written as

$$\tilde{f}_t(\mathbf{x}) = \sum_{t=1}^{T} \tilde{\Upsilon}_t K(\mathbf{x}_t, \mathbf{x}), \tag{2.2}$$

where $\tilde{\Upsilon}_t = \frac{1}{N}\sum_{j=1}^{N} \Upsilon_{j,}$.

In Herbrich et al. [2001] the concept of a version space with a *soft* boundary was introduced. A soft boundary can be incorporated with the OBPM algorithm (run in a batch mode) for all $(\mathbf{x}_l, y_l) \in \mathbf{z}$ by making the update condition for each Perceptron $j = 1, \ldots, N$

$$y_l \left(\sum_{t=1}^{T} \Upsilon_{j,t} \left(1 + \lambda\delta_{lt}\right) K(\mathbf{x}_l, \mathbf{x}_t) \right) \leq 0, \tag{2.3}$$

where λ is the soft boundary parameter and δ_{lt} is the Kronecker delta function. A setting of $\lambda = 0$ is referred to as a *hard* boundary and corresponds to the original version space.

If the maximum number of updates M_e required for separability is finite, then one only needs to store up to the maximum number of updates for each Perceptron, and thus the storage requirement for Υ_t is $O(M_e)$. During the training phase of Algorithm 3 one needs to store $\Upsilon_{j,t}$ for all $j = 1, \ldots, N$, therefore requiring storage of up to $O(NM_e)$ of the values, though in practice representing the resulting classifier $\tilde{\Upsilon}_t$, requires considerably less storage.

For the kernel implementation of Algorithm 3 to be truly online, one can bound the number of instances $\mathbf{x}_t$ needed to be stored for the kernel expansion by using the methods described in Kivinen et al. [2002] and Crammer et al. [2004].

2.3 Analysis of OBPM

Several key questions regarding the performance of Algorithm 3 are worthy of further exploration. How does τ affect the correlation between the various Perceptron solutions? How is the speed of convergence affected by the setting of τ?

We first introduce a number of definitions and assumptions required for the rest of the chapter (in some cases re-iterating them). The first assumption is that each example is drawn i.i.d from the probability distribution of examples $\mathrm{P_z}$, that is the online learning sequence of examples up to time T of $\mathbf{z} = (\mathrm{z}_1, \mathrm{z}_2, \ldots \mathrm{z}_T)$ is produced by drawing one example at a time from $\mathrm{P_z}$ (where each example is $\mathrm{z}_t = (\mathbf{x}_t, y_t)$). At iteration t each Perceptron j sees example z_t according to the Bernoulli random variable $b_{j,t}$, the example being seen only when $b_{j,k} = 1$. The subsequence of T examples seen by each Perceptron $j = 1, \ldots, N$ is determined by the random variable $S_j^T(\tau) = b_{j,1}, b_{j,2}, \ldots, b_{j,T}$. The random variable $S_j^T(\tau)$ is distributed according to the Bernoulli distribution $\mathcal{B}(\tau)$ with each $b_{j,t}$ drawn independently with $\Pr(b_{j,t} = 1) = \tau$. Therefore for all $j = 1, \ldots, N$ and all $\mathbf{z}$, the Perceptrons weight $\mathbf{w}_{j,T}(\tau)$ is a random variable on the probability space generated by $S_j^T(\tau)$.

2.3.1 Correlation between Perceptrons

Firstly, we study the effect diversity can have on margin of Algorithm 3. By using the subsampling technique of Algorithm 3 with the Perceptron ($\rho = 0$) we try to produce solutions $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N$ which are diverse. One method to measure how diverse the solutions are is to determine their correlation to each other. The higher the correlation the smaller the diversity. The diversity amongst the various weights $\mathbf{w}_1, \ldots, \mathbf{w}_N$ can be expressed in terms of the correlation between $\mathbf{w}_i$ and $\mathbf{w}_j$, where the correlation is defined by $\vartheta_{ij} = \frac{(\mathbf{w}_i \cdot \mathbf{w}_j)}{\|\mathbf{w}_i\| \|\mathbf{w}_j\|}$. From (2.1) for a fixed set of Perceptron weights[1] $\mathbf{w}_1, \ldots, \mathbf{w}_N$ we get

$$\gamma_{\mathbf{z}}(\tilde{\mathbf{w}}) \geq \frac{\sum_{j=1}^{N} \gamma_{\mathbf{z}}(\mathbf{w}_j)}{\sqrt{N^2 + \sum_{i \neq j} \vartheta_{ij}}}. \tag{2.4}$$

We see from (2.4) that the margin produced by the OBPM algorithm is dependent on the correlation ϑ_{ij} for all $i \neq j$. A smaller correlation between Perceptron weights ensures a larger low bound for (2.4).

We now investigate the subsampling of Algorithm 3 and its effects on the correlation of any of the two Perceptron weights produced. We fix $\mathbf{z}$, and look at the correlation with respect to the distribution over $S_j^T(\tau)$. We are interested in ϑ_{ij} but in any run of the Perceptron algorithm $\mathbf{w}_j$ is a random variable so we need to take expectations. This motivates the investigation of the correlation, or normalised expected scalar product defined as

$$\hat{\vartheta}_{S_1^T(\tau) S_2^T(\tau)}(\mathbf{w}_{1,T}(\tau), \mathbf{w}_{2,T}(\tau)) := \frac{\mathrm{E}_{S_1^T(\tau) S_2^T(\tau)} \left(\mathbf{w}_{1,T}(\tau) \cdot \mathbf{w}_{2,T}(\tau)\right)}{\|\mathrm{E}_{S_1^T(\tau)} \mathbf{w}_{1,T}(\tau)\| \|\mathrm{E}_{S_2^T(\tau)} \mathbf{w}_{2,T}(\tau)\|}. \tag{2.5}$$

It is easy to verify that (2.5) is a normalised correlation since $\hat{\vartheta}_{S_1^T(\tau) S_2^T(\tau)}(\mathbf{w}_{1,T}(\tau), \mathbf{w}_{2,T}(\tau)) \leq 1$ by applying the Cauchy-Schwartz inequality.

[1] The Perceptron weights in (2.1) are assumed normalised by $\|\mathbf{w}_1\|$.

Theorem 2.3.1. (Subsampled Perceptrons correlation) *Let* $\boldsymbol{z} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T))$ *be a sequence of examples seen up to time* T. *Perceptron* $j = 1, \ldots, N$ *sees a sequence of examples subsampled according to* $\mathcal{B}(\tau)$. *The probability that Perceptron* j *sees example* t *is given by* $\Pr(b_{j,t} = 1) = \tau$, *and* $S_j^T(\tau)$ *is a random variable which defines a sequence of sampling decisions up to time* T, *that is* $b_{j,1}, b_{j,2}, \ldots, b_{j,T}$. *The correlation between any two Perceptrons is*

$$\hat{\vartheta}_{S_1^T(\tau) S_2^T(\tau)}(\boldsymbol{w}_{1,T}(\tau), \boldsymbol{w}_{2,T}(\tau)) = \frac{\tau\left(\tau G(\tau) - Q(\tau) + F(\tau)\right)}{\|\mathrm{E}_{S_1^t(\tau)} \boldsymbol{w}_{1,t}(\tau)\| \|\mathrm{E}_{S_2^t(\tau)} \boldsymbol{w}_{2,t}(\tau)\|}.$$

where $F(\tau) = 2\sum_{i=1}^{T} \mathrm{E}_{S_1^i(\tau) S_2^i(\tau)}\left[I(\sigma_{1,i} = 0, \sigma_{2,i} = 1) | \left(\boldsymbol{w}_{1,i-1}(\tau) \cdot \boldsymbol{x}_i\right) |\right]$,

$$Q(\tau) = 2\sum_{i=1}^{T} \mathrm{E}_{S_1^i(\tau) S_2^i(\tau)}\left[I(\sigma_{1,i} = 1, \sigma_{2,i} = 1) | \left(\boldsymbol{w}_{1,i-1}(\tau) \cdot \boldsymbol{x}_i\right) |\right],$$

and $G(\tau) = \sum_{i=1}^{T} \Pr(\sigma_{1,i} = 1, \sigma_{2,i} = 1) \|\boldsymbol{x}_i\|^2$.

Proof. The Proof of Theorem 2.3.1 can be found in Appendix A.2. □

From Theorem 2.3.1 we have an expression for the correlation of the OBPM algorithm. We see that $G(\tau)$ from its definition depends on both Perceptrons making the same mistakes at the same time. It is easy to show that $G(\tau) \leq M/\tau$ given that $\Pr(\sigma_{1,i} = 1, \sigma_{2,i} = 1) \leq \Pr(\sigma_{1,i} = 1)$ (see Theorem 2.3.2) and the instances are normalised ($\|\mathbf{x}\| = 1$), where M is the classic Perceptron update bound. Therefore we see that the term $\tau^2 G(\tau)$ is $\mathrm{O}(\tau)$ and will get smaller as τ becomes smaller. Whilst the term $\tau^2 G(\tau)$ can be characterised, the behaviour of the other terms $\tau F(\tau)$ and $\tau Q(\tau)$ is not easy to understand. We can definitely say that for $\tau = 1$ that $F(\tau) = 0$, but is unclear what the exact values of $\tau Q(\tau)$ and $\tau F(\tau)$ are. Whilst Theorem 2.3.1 gives us an expression for the determining the correlation between any two Perceptrons empirically it does not guarantee that making τ smaller will result in smaller correlation. Theorem 2.3.1 does illustrate the difficulties in analysing the correlation, explicitly showing how the correlation depends on the past history of all the Perceptrons.

2.3.2 Mistake bounds

In this section we study the relationship between τ and the bound on the total of number of expected mistakes for T examples. We distinguish between a mistake and an update as follows: a mistake is independent of b_t (hence whether the example is seen or not) and is dependent only on whether $\sigma_t = 1$; an update is when the Perceptron both sees the example at time t and makes a mistake (therefore $b_t = 1$ and $\sigma_t = 1$). The update bound of Algorithm 3 for a linearly separable classification problem is given by Novikoff's Theorem (Novikoff [1962]).

The bound of Theorem 2.3.2 gives us the upper limit on the number of mistakes of the expected Perceptron taking into account the effect of the subsampling of Algorithm 3. To illustrate the effect that the subsampling has on expected number of

mistakes of the Perceptrons of Algorithm 3 consider the following example for a sequence of six examples:

$z_1, \ldots, z_6, \ldots$	$(\mathbf{x}_1, y_1)$	$(\mathbf{x}_2, y_2)$	$(\mathbf{x}_3, y_3)$	$(\mathbf{x}_4, y_4)$	$(\mathbf{x}_5, y_5)$	$(\mathbf{x}_6, y_6)$	$\ldots$
$b_{1,k} \sim \mathcal{B}(0.5)$	1	0	0	1	0	1	$\ldots$
$\sigma_{1,k}$	1	0	0	1	0	1	$\ldots$
$b_{2,k} \sim \mathcal{B}(0.5)$	1	1	0	0	0	1	$\ldots$
$\sigma_{2,k}$	1	1	0	1	0	1	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$b_{N,k} \sim \mathcal{B}(0.5)$	1	0	1	0	1	1	$\ldots$
$\sigma_{N,k}$	1	1	1	1	1	1	$\ldots$

We see for the second Perceptron that the number of updates is three but the number of mistakes is higher with four.

Theorem 2.3.2. (Mistake bound) *Let* $\boldsymbol{z} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T))$ *be the sequence used and* $\boldsymbol{w}^*$ *be a linear classifier that correctly classifies all examples of* $\boldsymbol{z}$ *i.e.* $\gamma_{\boldsymbol{z}} = \min_{(x,y)\in\boldsymbol{z}}(y\boldsymbol{w}^* \cdot x) > 0$. *Each Perceptron* $j = 1, \ldots, N$ *sees a sequence of examples sampled according to* $\mathcal{B}(\tau)$. *The probability that Perceptron* j *sees example* t *is given by* $\Pr(b_{j,t} = 1) = \tau$. $S_j^t(\tau)$ *is the random variable which defines a sequence of sampling decisions up to time* t, *that is* $b_{j,1}, b_{j,2}, \ldots, b_{j,t}$. *For* $i = (1, \ldots, T)$ *the instance vector* $\boldsymbol{x}_i$ *is normalised, such that* $\|\boldsymbol{x}_i\| = 1$. *In the limit as* $N \to \infty$ *the upper bound on the average number of mistakes for* $\boldsymbol{z}$ *is*

$$\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{T} \sigma_{j,i} = \sum_{i=1}^{T} \mathrm{E}_{S_j^i(\tau)}(\sigma_{j,i}) \leq \frac{\|\boldsymbol{w}^*\|^2}{\tau \gamma_{\boldsymbol{z}}^2}.$$

Proof. The Proof of Theorem 2.3.2 can be found in Appendix A.3. □

The bound of Theorem 2.3.2 implies that for smaller τ the convergence of Algorithm 3 is slower compared to a single Perceptron. In Section 2.3.1 we saw there was a trade-off for smaller τ and having larger diversity amongst the Perceptron weights. Hence there is some price to pay for achieving greater diversity that is the convergence will be slower. We can verify for $\tau = 1$ that Theorem 2.3.2 becomes the Perceptron mistake bound of Novikoff, this is easy to believe as all the Perceptrons see the entire sequence of examples $\mathbf{z}$.

Theorem 2.3.2 states the mistake bound in terms of expectation although in practice we only have a finite number of Perceptrons N. To consider the case where N is finite we can consider a probability bound but the intention of Theorem 2.3.2 is to simply illustrate the effect τ has on the convergence of the OBPM algorithm and not to provide an accurate mistake bound for finite N.

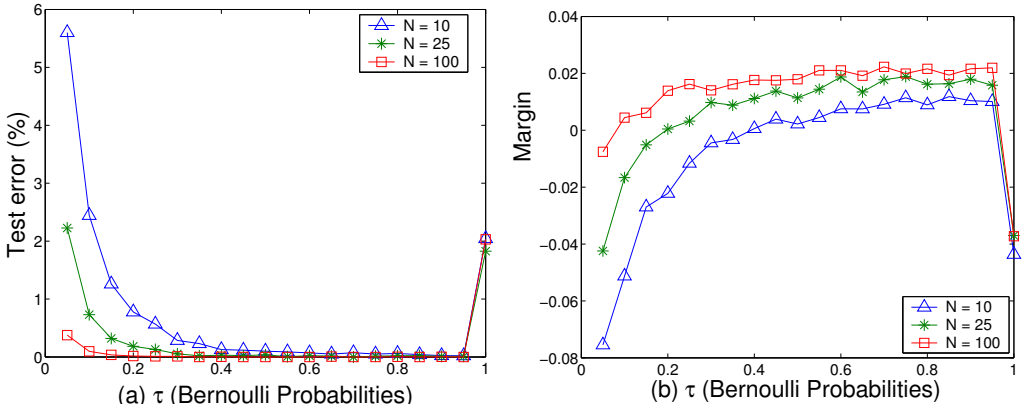

Figure 2.1: Synthetic data set averages from 30 Monte Carlo simulations training OBPM with no label noise ($e = 0.0$) for a single epoch: (a) test error versus τ, (b) margin $\gamma_{\mathbf{z}}(\tilde{\mathbf{w}}_T)$ versus τ. Note setting $\tau = 1$ corresponds to the standard Perceptron.

2.4 Experiment results

In this section we used a toy problem to analyse the effect of τ and N on the test error performance. We show that for this example we can do better than other online methods. Lastly, we show the performance of OBPM on some real-world data sets in both linear and non-linear settings, using the online prediction error analysis and test errors.

The tuning parameters used in the experiments for OBPM were τ (ranging from 0.01 to 0.5 in non-uniform steps) and $N \in \{100, 200\}$. For ALMA[2], B was adjusted from 0.5 up to 20 in non-uniform steps, and $C \in \{\sqrt{2}/4, \sqrt{2}/2, \sqrt{2}, 2\sqrt{2}\}$. Throughout the experiments the final reported parameters were chosen based on the best training error for that experiment. All reported results were for the algorithms using the last weights as the hypothesis.

2.4.1 Synthetic data set

A target $\mathbf{u} \in \{-1, 0, +1\}^{100}$ and T instances $\mathbf{x}_t \in \{-1, +1\}^{100}$ were both generated from uniform random selections and keeping the instances which satisfy a margin, $\gamma_{\mathbf{z}}(\mathbf{u}) \geq \kappa = 0.05$. The associated label y_t was allocated $+1$ if $(\mathbf{u} \cdot \mathbf{x}_t) / \|\mathbf{u}\|_2 \geq \kappa$ and -1 if $(\mathbf{u} \cdot \mathbf{x}_t) / \|\mathbf{u}\|_2 \leq -\kappa$. This process ensured that the training sample has a margin greater than κ; we generated the test sample identically to the training sample. This problem is very similar to the artificial data experiment of Gentile [2001]. Here we use a different data set size T, and we use the same margin in the training and test samples. We also reduced T from 10000 to 1000, and reduced κ from 1 to 0.05. This

[2] This was after initial experimentation to determine the best B and C for all the problems considered here.

Table 2.1: Table of test error percentages produced by averaging over 30 Monte-Carlo simulations of a single epoch of the parameter selection experiment, with 95 percent confidence interval for Student's t-distribution.

NOISE	PERCEPTRON	ALMA	OBPM
0.0	2.03 $\pm$0.32	0.07$\pm$0.04(B=1.11,C=0.71)	0.00$\pm$0.00($\tau = 0.35$,N=100)
0.01	3.35$\pm$0.44	0.14$\pm$0.07(B=0.83,C=0.71)	0.10$\pm$0.06($\tau = 0.50$,N=100)
0.1	12.96$\pm$1.0	0.76$\pm$0.13(B=1.25,C=0.71)	0.96$\pm$0.13($\tau = 0.15$,N=100)

gives a mistake bound of 400 for the standard Perceptron, making it highly unlikely that the Perceptron could learn $\mathbf{u}$ (the target) in a single epoch. Simulations with label noise on the training sample (not the test sample) were done by flipping each label y_t with probability e.

There is a trade-off between computational cost and $\tilde{\mathbf{w}}$ achieving an accurate estimate of the maximum margin classifier. It is indicated in Figure 2.1 (b) that increasing N caused margin to increase. As the margin increased, the total number of Perceptron updates was increased. For example when $\tau = 0.3$ in the noise free case, the total number of updates of all the Perceptrons went from 1334 to 5306, as a result of the total number of Perceptrons going from 25 to 100. The relationship between τ and N is shown by Figure 2.1: if τ is too small then the number of N must be increased to ensure $\tilde{\mathbf{w}}$ is in the version space. From Figure 2.1 (b) OBPM achieved a margin of 0.02, whereas ALMA achieved 0.006. Table 2.1 shows that with label noise the performance of ALMA and OBPM were similar, except OBPM was performed better in the case where there was no label noise. We also noted that ALMA's test error performance was more sensitive to the choice of parameter settings compared to OBPM.

2.4.2 Learning with large data sets

We consider three different data sets. The first data set was derived from a genome of the nematode Caenorhabditis Elegans (C. Elegans). The classification task was to locate the specific splice sites (for more details see Sonnenburg [2002]). The C. Elegans training sample had 6125 dimensions and 100 000 examples, with a separate test set of 10 000 examples. The other two data sets were the UCI Adult and the Web data sets [3]. Each input of the Adult data set consists of 14 mixed categorical and continuous attributes. We increased the input dimensionality to 123 by quantising the continuous attributes and by representing the categorical inputs with a 1-of-N code. The training sample size of the Adult data set is 32562 and the test set size is 16282. The prediction task of the Adult data set is to determine whether a person earns over \$50k a year. The Web data set task consists of predicting whether a web page belongs to a topic or not, based on the presence of 300 words. The Web task has a training sample size of 49 749

[3] Available at `http://www.research.microsoft.com/~jplatt/smo.html`

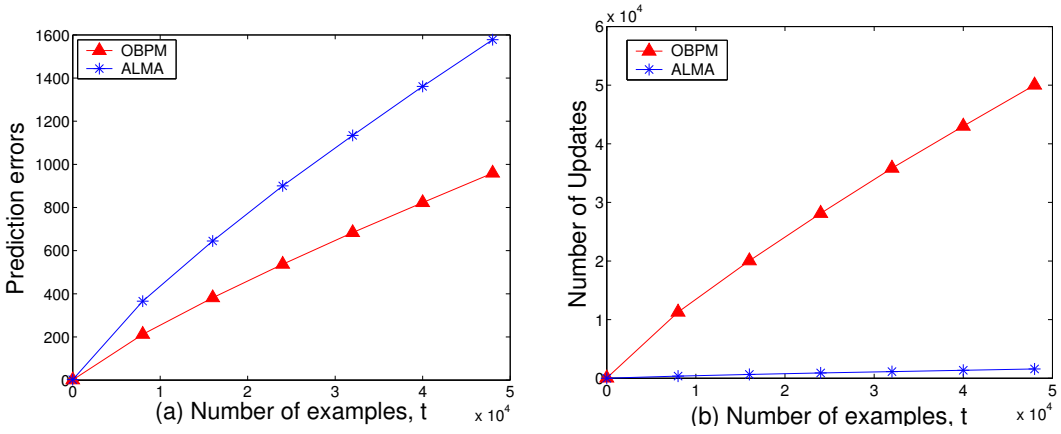

Figure 2.2: (a) Prediction errors for OBPMs $\tilde{\mathbf{w}}_t$ (τ=0.3, N=100) and ALMAs weight solutions and (b) total number updates of both ALMA and OBPM for the Web data after training t examples.

and a test set size of 21 489.

To study the online behaviour of this algorithm we measured the number of prediction errors made at t examples and compared this to ALMA. The prediction errors made on the Web data in Figure 2.2(a) demonstrate a lower prediction error compared to ALMA. The prediction error was consistently lower than ALMA with all the experiments in this paper. The lower prediction error comes at a cost, as shown in Figure 2.2(b): the total number of updates made by OBPM is larger than the number made by ALMA.

The results benchmarking OBPM with other algorithms are given in Table 2.2 for up to 3 epochs. The UCI data results of Table 2.2 report the average test error for 30 trials of permutations of the training sequence with their 95 percent confidence interval using the Student's t-distribution (see Appendix B.1). The maximum margin benchmark, the linear SVM with a linear kernel gave test errors of 15.05% and 1.25% for the Adult and Web respectively. The C. Elegans data set OBPM with $\tau = 0.05, N = 100$, was able to achieve a test error of 1.59% after 20 epochs which was comparable to the SVM (soft margin) result 1.58%, ALMA achieved 1.72%. The Cochran test statistic (see Appendix B.2) for C. Elegans after 20 epochs comparing ALMA, OBPM and SVM was 3.77 which indicates no difference at a significance level of 95 percent (corresponds to the 0.95 quantile of chi-squared distribution). We notice that for the Adult data set after 1 epoch that OBPM was significantly better to ALMA using the t-test. As a general comment after 3 epochs of the results of Table 2.2 OBPM performed better than the standard Perceptron whereas ALMA and OBPM were comparable. Tuning ALMA's parameters was dependent on the data set, the "harder to learn" data set need a larger setting of B which corresponds to a smaller maximum margin. Only the best training error results are shown in Table 2.2 though the OBPM algorithm results were consistent over a range τ settings, seemly less sensitive to parameter selection than the

ALMA algorithm.

The time taken to process the C. Elegans results using C code on a 1GHz DEC alpha were: OBPM with CPU time 19 minutes (real time 21 minutes) and ALMA CPU time of 3 minutes (real time 13 minutes). The CPU time is not prohibitive, especially given that the SVM[4] optimisation took approximately 300 hours of real time. Of course, on a parallel machine, the OBPM algorithm could be sped up further.

Table 2.2: Real world data set test errors in percent run for 3 epochs. Result in bold indicates significantly different.

ALGORITHM	**EPOCHs**	**C. Elegans**	**UCI Adult**	**UCI Web**
Perceptron	1	3.33	19.59±1.54	1.84±0.12
	2	2.99	18.81±0.96	2.12±1.03
	3	2.93	21.21±2.21	1.72±0.18
ALMA(B,C)		(B=3.33,C=0.70)	(B=20,C=0.70)	(B=0.625,C=1.41)
	1	1.95	15.88±0.19	**1.36±0.03**
	2	1.92	15.35±0.13	**1.29±0.03**
	3	1.78	15.26±0.13	1.26±0.03
OBPM(τ,N)		(τ=0.2,N=200)	(τ=0.01,N=200)	(τ=0.3,N=200)
	1	1.94	**15.29±0.15**	1.42±0.04
	2	1.86	15.24±0.19	1.36±0.04
	3	1.79	15.17±0.14	1.31±0.03

2.4.3 The kernel trick and UCI data set

In this section we investigated the kernel trick and the OBPM (see Section 2.2.1). We performed our experiments on five benchmark data sets from the UCI repository[5]. Four of the data sets: Heart, Sonar, Thyroid and Ionosphere were used in the experiments of Herbrich et al. [2001] and Minka [2001]. As some data sets required normalisation to ensure consistency we normalised all data sets instances. The classification experiments were performed using the Radial Basis Function (RBF) kernel,

$$K(\mathbf{x}_j, \mathbf{x}_k) = \exp\left(-\frac{\|\mathbf{x}_j - \mathbf{x}_k\|^2}{2\varsigma^2}\right), \tag{2.6}$$

where the choice of ς was made from the set $\{0.5, 1, 2, 4, 8, 20\}$. The final choice corresponded to the one with the best Perceptron training error for a single epoch. In all the tuning experiments it was found to be $\varsigma = 0.5$.

The same set of parameters were explored as in the previous sections 2.4.1 and 2.4.2. Only the parameters which gave the best training error after 1 epoch were used

[4]We used the fast interior point optimisation package SVlab of Alex Smola which has been shown to have comparable speed to SVM light.

[5]`http://www.ics.uci.edu/~mlearn/MLRepository.html`

to produce the test error results of Table 2.3. The test error results and their 95 percent confidence interval for the Student's t-distribution were produced by 100 random trials, where the training and test data sets having a split of 60/40. The fifth and sixth rows of Table 2.3 are the results reported in Herbrich et al. [2001] of the Bayes Point Machine (BPM) using the *billiard* algorithm with a hard boundary and the SVM with a hard margin respectively.

Table 2.3: UCI data set test errors in percent produced by 100 random trials, for up to 10 epochs. Results in bold indicate that they are the smallest and statistically significant using the 95 percent confidence interval of the Student's t-distribution.

Algorithm	**Heart**	**Sonar**	**Thyroid**	**Pima**	**Ionosphere**
Perceptron					
1 epoch	24.87±1.06	18.99±1.01	13.71±3.45	29.56±1.55	12.94±0.87
3 epochs	22.28±0.78	16.04±0.87	9.44±2.10	29.26±1.13	11.72±0.80
10 epochs	21.80±0.77	15.99±0.87	7.94±1.71	29.63±1.41	10.99±0.60
ALMA(B,C=1.414)	B=0.625	B=0.909	B=1.67	B=4	B=1.25
1 epoch	20.24±0.71	14.87±0.91	**7.43±0.80**	23.39±0.50	10.84±0.56
3 epochs	20.70±0.78	14.66±0.88	**5.49±0.66**	23.96±0.48	10.65±0.61
10 epochs	20.82±0.73	14.66±0.87	5.02±0.58	25.14±0.53	10.69±0.58
OBPM(τ,N=100)	τ=0.45	τ=0.3	τ=0.55	τ=0.25	τ=0.5
(hard boundary)					
1 epoch	21.08±0.84	14.85±0.84	9.09±0.91	23.85±0.54	10.39±0.54
3 epochs	19.97±0.78	13.90±0.74	6.92±0.86	23.87±0.55	9.69±0.55
10 epochs	20.29±0.68	13.73±0.75	5.23±0.64	24.40±0.47	10.01±0.57
OBPM(τ,N=100)	τ=0.45	τ=0.35	τ=0.45	τ=0.25	τ=0.55
(soft boundary)	$\lambda = 0.2$	$\lambda = 0.4$	$\lambda = 0.4$	$\lambda = 0.5$	$\lambda = 0.9$
1 epoch	20.95±0.78	14.70±0.83	9.07±0.81	23.59±0.53	10.35±0.64
3 epochs	**19.63±0.76**	**13.89±0.77**	7.27±0.69	23.56±0.48	**9.62±0.56**
10 epochs	**19.60±0.68**	**13.70±0.70**	5.78±0.60	**23.58±0.43**	**9.68±0.54**
BPM					
(hard boundary)	22.8±0.34	15.9±0.38	**4.4±0.21**	N\A	11.5±0.25
SVM					
(hard margin)	25.4±0.40	15.4±0.37	5.3±0.24	N\A	11.9±0.25

From Table 2.3 we see after 10 epochs in all but the Thyroid data set the OBPM test error was the lowest, with statistical significance. For the Thyroid data set the BPM has the best test error. We see that in general, OBPM performs well on the harder to classify data sets like: Heart, Sonar and Pima. However in the Thyroid data set for which the lowest test error was obtained for all algorithms, ALMA was shown to have a faster convergence. This general improvement by OBPM when compared to ALMA comes with an additional computational cost. As an indication of this cost, the largest difference in updates was for the Sonar data set: OBPM after 10 epochs made 4339

updates for 100 Perceptrons, whereas ALMA made just 95. The extra computational cost is not prohibitive: the entire OBPM experiment on the sonar data set, with soft boundary tuning, took 13 minutes on a 2.4GHz i686 processor, compared to 8 minutes for ALMA's experiment.

Based on the empirical evidence, especially for harder to learn data sets, there appears to be an advantage in using the subsampling approach when compared to both the hard boundary BPM and hard margin SVM. The BPM and SVM results of Table 2.3 can be improved by using a soft boundary with BPM and a soft margin with SVM. For the Heart data set it was shown in Herbrich et al. [2001] that a soft boundary with BPM and a soft margin with SVM both produced a test error of approximately 16%. The subsampling technique of OBPM already gives some immunity to noise as illustrated by the results of Table 2.3. Adding a soft boundary with the OBPM gave improved results compared to the hard boundary, though the gains were modest. Note that the soft boundary parameter was tuned over the range of 0.1 to 0.9 using the training set.

2.4.4 USPS hand digit recognition

The experiments conducted in this section were on the well known USPS (US Postal Service) images of hand written numerals. The data set consists of 7291 training and 2000 test examples. Each example $(\mathbf{x}_i, l_i)$ consisted of a 16×16 image converted to an instance vector $\mathbf{x}_i \in \mathbb{R}^{256}$ and label $l_i \in \{0, 1, \ldots, 9\}$ pair. Each label l was allocated a classifier $\mathbf{w}_l$ which was trained as a binary problem by allocating $(\mathbf{x}, +1)$ to that label's classifier and $(\mathbf{x}, -1)$ to the remaining classifiers, the so called *one versus rest* scheme for m-class problems. The final prediction $\hat{l}$ was chosen according to $\arg\max_l (\mathbf{w}_l \cdot \mathbf{x})$. A similar procedure to the previous section was used to tune σ for RBF. The parameter settings $\sigma = [0.5, 1, \sqrt{2}, 2, 3, 4]$ were tried with the $\sigma = 3$ being found to give the best training error.

The results of Table 2.4 show the final test errors of OBPM with 100 Perceptrons ($N = 100$). Reported also in Table 2.4 are results with various settings of the soft boundary parameter λ and the Bernoulli probability of seeing an example τ. The test error of the OBPM-with-soft-boundary result of 4.63% compares well with the soft margin SVM result of 4.5% reported by Herbrich et al. [2001]. Several other SVM test error results are available for this data set: Schölkopf et al. [1999] with 4.4%, Platt [1998] reported 4.7%. In Gentile [2001] reported a test error of 4.68% after 2 epochs using ALMA($B = 1.053$,$C = \sqrt{2}$) in conjunction with the voting method "average" of Freund and Schapire [1999]. Though they noted that on the third epoch the test error went up slightly to 4.73%.

The time taken to train the Perceptron for 10 epochs using C code on a 2.4 GHz i686 processor was 37 secs excluding the time to calculate the kernel's Gram matrix. It took 10 minutes and 39 secs to train OBPM with $\tau = 0.2$ and $N = 100$. The kernel Gram matrix took only an additional 1 minute and 22 secs to calculate.

Table 2.4: USPS data set test errors in percent run for 10 epochs for a RBF kernel with $\sigma = 3$.

Algorithm	1 Epoch	3 Epochs	10 Epochs
Perceptron	8.77	6.58	6.78
OBPM($\lambda = 0.0$,$\tau = 0.20$,N=100)	6.33	5.23	4.88
OBPM($\lambda = 0.0$,$\tau = 0.30$,N=100)	6.33	5.08	4.98
OBPM($\lambda = 0.0$,$\tau = 0.40$,N=100)	6.02	4.98	4.88
OBPM($\lambda = 0.3$,$\tau = 0.20$,N=100)	6.38	5.18	4.98
OBPM($\lambda = 0.4$,$\tau = 0.20$,N=100)	6.38	5.18	4.88
OBPM($\lambda = 0.5$,$\tau = 0.20$,N=100)	6.22	5.33	4.63

2.4.5 Drifting concept

In order to demonstrate OBPM's abilities to learn a drifting concept we designed a drifting experiment using the well known MNIST OCR data set[6]. To simulate the drift we set the positive class (i.e. $y = 1$) to a single label which varied over time. The negative class (i.e. $y = -1$) was set to be the remaining 9 labels. We took 1000 examples. The labels where swapped in two phases gradually using a linear relationship in the number of examples seen so far. The mixing of the labels in each trial was random and we averaged results over 10 trials, consequently the transition boundaries in Figure 2.3 are not obvious. The following pseudo code shows how we allocated the original examples $(\mathbf{x}_t, c_t)$ where the labels $c_t \in \{0, 1, \ldots, 9\}$ are converted into a binary problem $(\mathbf{x}_t, y_t)$ where $y_t \in \{-1, +1\}$, according to the uniform random variable $r_t \in [0, 1]$ at time t:

if $c_t \in \{1, 2, 3\}$ then
  $(\mathbf{x}_t, 1)$ becomes $(\mathbf{x}_t, +1)$
  if $r_t > (1 - \frac{t}{T/2})$ then
    $(\mathbf{x}_t, 2)$ becomes $(\mathbf{x}_t, +1)$
  endif
  if $r_t > (1 - \frac{t}{1.11T})$ then
    $(\mathbf{x}_t, 3)$ becomes $(\mathbf{x}_t, +1)$
  endif
else $(\mathbf{x}_t, c_t)$ allocated $(\mathbf{x}_t, -1)$
endif

Selecting the labels in this way ensures that by $t = T/2$ all the examples with label 2 are in the positive class and by 900 all the examples with label 3 are in the positive class. Replacing the labels randomly in the above way also gives a smoother and more gradual transition between labels.

[6] Available at `http://yann.lecun.com/exdb/mnist/index.html`.

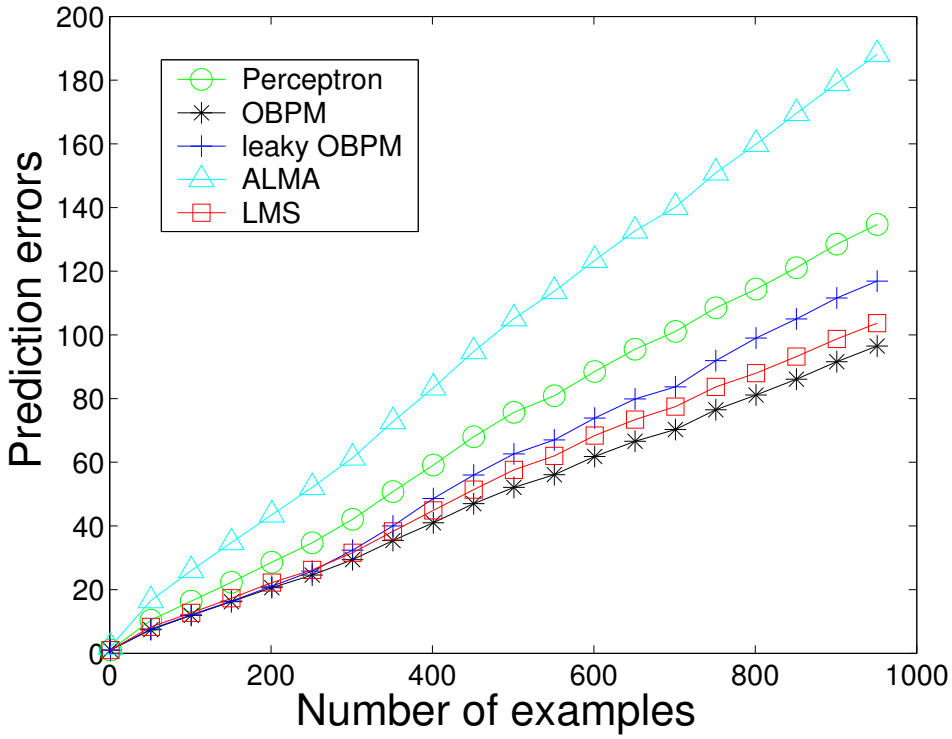

Figure 2.3: Prediction errors from the drifting concept experiment with the MNIST OCR data set comparing LMS (step size 0.5), ALMA (B=2, C=$\sqrt{2}$), Perceptron ($\rho = 0$) and OBPM (τ=0.1, N=100).

Figure 2.3 presents the results of the MNIST drift experiment for 10 trials with permutations. We see that OBPM made fewer prediction errors compared to several other online algorithms. One of the online algorithms compared with OBPM was LMS (Least Mean Squared (Widrow and Hoff [1960])). We converted this to the classification setting by taking the sign of the resulting prediction. LMS is well known for its abilities for adaption, so it is not that surprising it worked well here. One explanation for the OBPM performing better relates to the one versus the rest scheme and due to the imbalance in the number of examples for each binary class. This imbalance limits a method like LMS which directly minimises distance between the two classes. One reason why ALMA performed so poorly is that ALMA assumes stationarity in the way it adjusts the learning rate η_t and margin parameter ρ_k.

In endeavouring to study the *plasticity* of OBPM, and therefore its ability to adapt by continuing to learn from new data, even when the concept is drifting we consider a *leaky* version of OBPM. The leaky OBPM has the ability to forget the past which is achieved by modifying line 7 of Algorithm 3 to

$$\mathbf{w}_{j,t} := \mathcal{L}((1-\beta)\mathbf{w}_{j,t-1}, \eta_{j,t}, \mathbf{x}_t, y_t),$$

where β is the forgetting factor. Figure 2.3 shows the leaky OBPM performance when $\beta = 0.1$. We noticed that the drift is relatively slow, because when setting $\beta = 0.03$ (approximated by the ratio of 1000 examples over 3 different targets,) there was no noticeable difference between the leaky OBPM and the OBPM.

Whilst this is not an exhaustive study on drifting concepts it does illustrate that the large margin of OBPM demonstrates plasticity where the drift is slowly changing.

2.5 Summary

In this chapter we presented a meta-algorithm for online training of a pattern classifier with a large margin. The algorithm randomly selects subsets of the incoming stream of examples, and presents the chosen subsets to each classifier for training. An average of the resulting classifier weights is produced after training, endeavouring to approximate the Bayes point, hence the algorithm is referred to as Online Bayes Point Machine (OBPM). The classifier considered in this chapter with the OBPM algorithm was the standard Perceptron algorithm. We showed that making τ too small will result in slower convergence. The empirical evidence indicates the OBPM algorithm is comparable in both online (i.e. single epoch) and batch learning settings with other large margin algorithms. In some of the cases where there was noisy data the OBPM algorithm had improved results compared to the large margin algorithm ALMA. Due to the subsampling technique, the single processor computation times of the OBPM were not prohibitive. The potential of a parallel architecture implementation of the OBPM algorithm represents a possible saving compared to a single Perceptron method like, ALMA. The simplicity of implementation and versatility of this algorithm should make it appealing to many applications in machine learning where the size of the data sets is large.

Online Voting Methods

3.1 Introduction

We introduced in the last chapter an online version of the Bayes point machine, which we showed improved the generalisation of the Perceptron. The online Bayes point machine produced a classifier which was an average of N classifier weights. One other online approach worth considering is averaging the final classifiers hypotheses rather than their weights. Methods which combine the hypotheses in an endeavour to improve generalisation performance are usually referred to as either ensemble or aggregate methods. One of the most famous of the aggregating methods is Bagging by Breiman [1996]. There have been several online variants for Bagging. The first technique for online Bagging was by Fern and Givan [2000] where they used the same sampling technique used in Algorithm 3, though they only considered Bagging. The second technique for online Bagging was presented by Oza [2001] where the sampling proposed is according to the Poisson distribution with mean of one. The probability of each classifier seeing example k at least once (multiple times are possible) is $1-\exp(-1) \approx 0.63$ according to the method for online Bagging of Oza [2001]. Sampling this way ensures that each base classifier's hypothesis sees on average each example once. This is roughly equivalent to Algorithm 3 with the probability of seeing an example of $\tau = 0.63$. By each classifier on average seeing each sample at least once, the hope is that the online version is lossless compared to Bagging in the batch setting. Online Bagging of Oza [2001] may not achieve losslessness at all, since the potential diversity between the classifier's hypotheses is limited, reducing the effect of Bagging. This is especially true when using the Perceptron as the base hypotheses.

In this chapter the online voting methods of voted Perceptron (Freund and Schapire [1999]) and Bagging are compared with the method of the last chapter, the online Bayes point machine (OBPM). Through the empirical study of this chapter we will compare the various online voting methods when applied to the Perceptron, and to determine any differences between averaging classifier weights versus their hypotheses.

Algorithm 4 GOAM algorithm

Require: A training sample $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: A online learning algorithm with update rule $\mathcal{L}$ for the classifier and associated step-size update rule $\mathcal{S}$.

Require: A subroutine Bernoulli(p) which returns independent Bernoulli random variables with probability p of taking the value 1.

Require: Parameters $N \in \mathbb{N}$ and $\tau \in [0, 1]$.

Initialise step sizes $\eta_{j,1}$, for all $j \in \{1, \ldots, N\}$.
For all $j \in \{1, \ldots, N\}$, initialise weights $\mathbf{w}_{j,0} = \mathbf{0}$ and the number of correctly classified examples $r_{j,0} = 0$.
for $t = 1$ to T **do**
  for $j = 1$ to N **do**
    $b_{j,t} :=$ Bernoulli(τ)
    if $b_{j,t} = 1$ **then**
      $(\mathbf{w}_{j,t}, r_{j,t}) := \mathcal{L}(\mathbf{w}_{j,t-1}, \eta_{j,t}, \mathbf{x}_t, y_t, r_{j,t-1})$
    else
      $\mathbf{w}_{j,t} := \mathbf{w}_{j,t-1}$, $r_{j,t} := r_{j,t-1}$
    end if
    $\eta_{j,t+1} := \mathcal{S}(t, \eta_{j,t}, \ldots)$
  end for
end for
if Bagging **then**
  $q_{j,T} := 1/N$ for $j = 1, \ldots, N$
end if
if Majority vote **then**
  $q_{j,T} := r_{j,T}$ for $j = 1, \ldots, N$
end if
return $H(\mathbf{x}) := \operatorname{sgn}\left(\sum_{j=1}^{N} q_{j,T} \operatorname{sgn}(\mathbf{w}_{j,T} \cdot \mathbf{x})\right)$

3.2 Algorithms for online voting

Fern and Givan [2000] and Oza [2001] restricted their study of their online Bagging algorithms to decision trees. It is well known that Bagging has its greatest effect when the classifier solutions vary a lot, e.g. decision trees. We have seen that the Perceptron can exhibit variation when the order in which examples are presented to it varies. This motivating the investigation of online Bagging with the Perceptron.

The majority vote and Bagging algorithms can be combined into a single meta-learning algorithm for aggregating classifiers: the *generalised online aggregation method* (GOAM) of Algorithm 4. The goal of GOAM is to generate diversity amongst N different classifiers hypotheses by using the same subsampling scheme as in Algorithm 3 (see last chapter). Each classifier sees an example at iteration t when $b_{j,t} = 1$, which

Algorithm 5 OBPM-voted algorithm

Require: A training sample $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: A online learning algorithm with update rule $\mathcal{L}$ for the classifier and associated step-size update rule $\mathcal{S}$.

Require: A subroutine Bernoulli(p) which returns independent Bernoulli random variables with probability p of taking the value 1.

Require: Parameters $N \in \mathbb{N}$ and $\tau \in [0, 1]$.

1: Initialise step sizes $\eta_{j,1}$, for all $j \in \{1, \ldots, N\}$.
2: For all $j \in \{1, \ldots, N\}$, initialise weights $\mathbf{w}_{j,0} = \mathbf{0}$ and the number of correctly classified examples $r_{j,0} = 0$.
3: Initialise $\bar{\mathbf{w}}_0 = \mathbf{0}$, $i = 0$ $v_i = 0$;
4: **for** $t = 1$ to T **do**
5: $(\mathbf{w}_{1,t}, ..., \mathbf{w}_{N,t}, \eta_{1,t+1}, ..., \eta_{N,t+1}) :=$OBPM$(\mathbf{w}_{1,t-1}, ..., \mathbf{w}_{N,t-1}, \eta_{1,t}, ..., \eta_{N,t}, \mathbf{x}_t, y_t, \tau)$
6: **if** $y_t (\bar{\mathbf{w}}_i \cdot \mathbf{x}_t) \leq 0$ **then**
7: $\bar{\mathbf{w}}_{i+1} = \tilde{\mathbf{w}}_t = \sum_{j=1}^{N} \mathbf{w}_{j,t}/N$
8: $i = i + 1$
9: $v_i = 0$
10: **else**
11: $v_i = v_i + 1$
12: **end if**
13: **end for**
14: **return** $H(\mathbf{x}) := \operatorname{sgn}\left(\sum_{i=1}^{m} v_i \operatorname{sgn}\left(\bar{\mathbf{w}}_i \cdot \mathbf{x}\right)\right)$

occurs with probability τ. At line 8 of GOAM, each classifier $j = 1, \ldots, N$ is updated according to the rule $\mathcal{L}$, which returns an updated weight $\mathbf{w}_j$, and the number of correctly classified examples up to and including t, $r_{j,t}$. The generality of the Algorithm 4 comes in the choice of aggregate method, two alternatives are given on lines 15 and 18: Bagging (GOAM-Bagging), or the majority vote (GOAM-majority). GOAM-Bagging is essentially the method proposed by Fern and Givan [2000]. The GOAM-majority scales each classifier's weight $\mathbf{w}_{j,T}$ by $r_{j,T}$, producing a final hypothesis $H(\mathbf{x}) = \sum_{j=1}^{N} r_{j,T} \operatorname{sgn}\left(\mathbf{w}_{j,T} \cdot \mathbf{x}\right)$. The intention of scaling each classifier's weights by $r_{j,T}$ is more weight is given to the better performing classifier, akin to majority vote technique, hence the extension majority.

Although Algorithm 4 gives a meta-learning algorithm for combining arbitrary hypotheses, our focus is on the Perceptron. One online voting method of interest is the voted Perceptron of Freund and Schapire [1999] (see Chapter 1). Freund and Schapire [1999] claimed that the voted Perceptron will asymptotically give the best Perceptron weight solution. For finite sequences, the final voted Perceptron is still dependent on the sequence order. This suggests that the voted Perceptron would benefit from the application of Bagging. One could look at Algorithm 4 to implement Bagging with the voted Perceptron by running N independently run voted Perceptrons and averaging

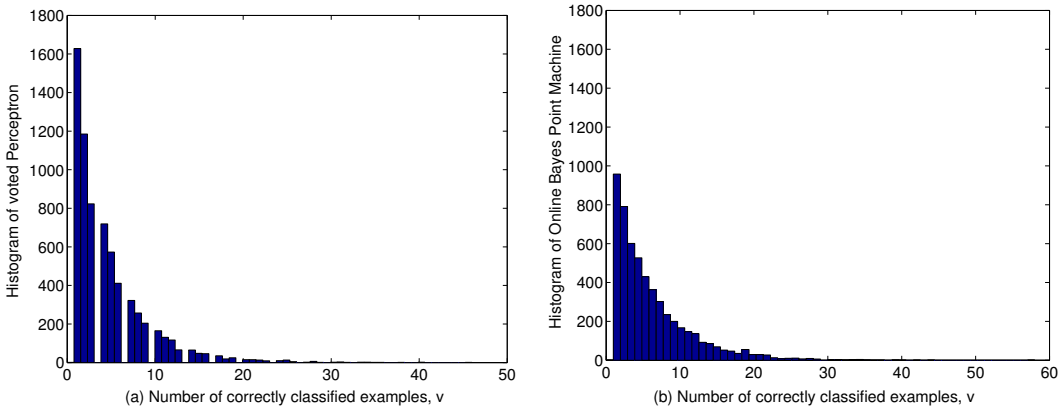

Figure 3.1: Histograms showing the frequency of occurrence of particular numbers of correctly classified training examples (v) made by each stored weight $\mathbf{w}_i$ for $i = 1, \ldots, m$. Each algorithm (a) voted Perceptron and (b) OBPM was trained for a single epoch and trial of the Adult data set. The total number stored Perceptron weights was $m = 6945$ for the voted Perceptron and for OBPM $m = 5469$.

their final hypotheses. The main disadvantage of applying Bagging to the voted Perceptron is that to implement the final hypothesis requires enough memory capacity to store $O(mdN)$ floating point numbers, where m is the number of updates and d the dimension of the weight vectors. Alternatively, rather than using N independently run voted Perceptrons, one can apply the same principles of the voted Perceptron to the online Bayes point machine (OBPM)(Algorithm 3). This way one can take advantage of the improved performance of the OBPM algorithm through the randomisation trick and the ability of the voted Perceptron to converge to the best solution. We refer to this algorithm as the OBPM-voted and its implementation is in Algorithm 5. For each training example $(\mathbf{x}_t, y_t)$ as given by line 5 of Algorithm 5 we perform an update in accordance with the OBPM (Algorithm 3) producing weights $\mathbf{w}_{1,t}, \ldots, \mathbf{w}_{N,t}$. The rest of Algorithm 5, lines 6-14 are an implementation of the voted-Perceptron (Algorithm 2 of Chapter 1). The final hypothesis of OBPM-voted is

$$H(\mathbf{x}) := \operatorname{sgn}\left(\sum_{i=1}^{m} v_i \operatorname{sgn}\left(\bar{\mathbf{w}}_i \cdot \mathbf{x}\right)\right). \tag{3.1}$$

The main advantages of OBPM-voted are that the number of updates, m, should be smaller for OBPM compared to the Perceptron (see last chapter), and we only need to store $O(md)$ floating point numbers to implement (3.1).

3.3 Empirical results

We study here the performance of the proposed online voting methods of Section 3.2 compared with the voted Perceptron and the online Bagging of Oza [2001] (Oza-

Table 3.1: Large data sets N=100 Test errors in percent. The 95 percent confidence intervals of the Student's t distribution for 100 random trials are given. The lowest test errors which are statistically significant are highlighted by bold face.The OBPM result is with a hard boundary.

Algorithm	**Synthetic**	**Adult**	**Web**	**Hypothyroid**	**kr-vs-kp**
Perceptron					
1 epoch	4.03±0.12	19.59±1.54	1.84±0.12	3.87±2.20	13.76±1.62
2 epochs	2.56±0.09	18.81±0.96	2.12±1.03	3.03±0.69	11.11±1.40
3 epochs	1.93±0.07	21.21±2.21	1.72±0.18	3.02±0.69	9.87±1.15
voted Perceptron					
1 epoch	1.27±0.05	15.00±0.02	1.37±0.02	2.26±0.10	6.59±0.15
2 epochs	0.48±0.03	15.01±0.02	1.28±0.01	2.12±0.09	5.39±0.13
3 epochs	0.24±0.02	15.01±0.02	1.25±0.02	2.07±0.08	5.07±0.12
GOAM-Majority	$\tau = 0.3$	$\tau = 0.05$	$\tau = 0.4$	$\tau = 0.45$	$\tau = 0.5$
1 epoch	1.01±0.05	15.43±0.09	1.46±0.03	2.16±0.10	9.48±1.02
2 epochs	0.41±0.02	15.36±0.09	1.38±0.02	2.17±0.09	7.69±0.78
3 epochs	0.18±0.02	15.35±0.09	1.34±0.02	2.08±0.09	7.14±0.71
GOAM-Bagging	$\tau = 0.3$	$\tau = 0.05$	$\tau = 0.4$	$\tau = 0.45$	$\tau = 0.5$
1 epoch	1.02±0.05	15.43±0.09	1.47±0.03	2.16±0.10	9.49±1.01
2 epochs	0.41±0.02	15.36±0.09	1.38±0.02	2.16±0.09	7.70±0.78
3 epochs	0.18±0.02	15.34±0.08	1.34±0.02	2.08±0.09	7.15±0.71
Oza Bagging					
1 epoch	1.02±0.05	18.04±0.66	1.61±0.23	2.27±0.12	10.14±1.25
2 epochs	0.36±0.02	17.97±0.62	1.64±0.32	2.19±0.12	8.09±0.98
3 epochs	0.20±0.02	18.14±0.73	1.60±0.30	2.17±0.10	7.40±0.76
OBPM	$\tau = 0.3$	$\tau = 0.05$	$\tau = 0.3$	$\tau = 0.55$	$\tau = 0.5$
1 epoch	0.93±0.05	15.36±0.09	1.46±0.03	2.19±0.10	9.51±1.00
2 epochs	0.37±0.02	15.31±0.09	1.38±0.02	2.13±0.10	7.68±0.76
3 epochs	0.16±0.02	15.31±0.09	1.31±0.02	2.09±0.09	7.00±0.69
OBPM-voted	$\tau = 0.3$	$\tau = 0.3$	$\tau = 0.9$	$\tau = 0.95$	$\tau = 0.95$
1 epoch	**0.66±0.04**	**14.87±0.01**	1.37±0.00	2.24±0.10	**6.19±0.14**
2 epochs	**0.22±0.01**	**14.94±0.01**	**1.26±0.00**	**2.06±0.08**	**5.20±0.12**
3 epochs	**0.11±0.01**	**14.96±0.01**	**1.23±0.00**	**2.02±0.08**	**4.88±0.12**

Bagging). Most of the data sets of Chapter 2 were used again here. The only exception was the C. Elegans data set which was left out as the voted Perceptron required too much memory to be practical. The experiments are split into two parts: linear classification on large data sets, and radial basis kernel implementations on data sets within the UCI repository.

All of the experimental results are a result of 100 Monte-Carlo trials. In all of the online voting methods the total number of Perceptrons combined was $N = 100$. The Bernoulli probability threshold τ chosen for GOAM and OBPM was the one which produced the best training error.

Table 3.2: Test errors in percent performed on a number of real world data sets, for up to 10 epochs. The 95 percent confidence intervals of the Student's t distribution for 100 random trials are given.The lowest test errors which are statistically significant are highlighted by bold face. The OBPM result is with a hard boundary.

Algorithm	Heart	Sonar	Thyroid	Pima	Ionosphere
Perceptron					
1 epoch	24.87±1.06	18.99±1.01	13.71±3.45	29.56±1.55	12.94±0.87
3 epochs	22.28±0.78	16.04±0.87	9.44±2.10	29.26±1.13	11.72±0.80
10 epochs	21.80±0.77	15.99±0.87	7.94±1.71	29.63±1.41	10.99±0.60
voted Perceptron					
1 epoch	22.07±0.78	22.90±1.12	9.93±0.96	23.50±0.48	13.28±0.68
3 epochs	21.42±0.75	16.76±0.91	6.84±0.70	23.70±0.49	10.99±0.61
10 epochs	21.55±0.69	16.76±0.91	5.47±0.64	24.34±0.48	11.01±0.61
GOAM-Majority	$\tau = 0.9$	$\tau = 0.9$	$\tau = 0.9$	$\tau = 0.9$	$\tau = 0.9$
1 epoch	24.99±1.08	20.12±0.94	12.91±2.92	**20.55±1.00**	13.98±1.03
3 epochs	22.94±0.97	16.17±0.89	9.86±2.79	**16.29±0.90**	11.86±0.71
10 epochs	21.99±0.81	15.87±0.88	7.19±1.65	**16.38±0.89**	11.10±0.62
GOAM-Bagging	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.5$	$\tau = 0.2$	$\tau = 0.7$
1 epoch	21.54±0.80	16.07±0.77	8.92±0.78	23.84±1.55	11.08±0.74
3 epochs	20.18±0.75	14.61±0.76	6.48±0.70	23.53±1.13	10.14±0.58
10 epochs	20.41±0.71	14.59±0.74	5.20±0.62	24.06±1.41	10.23±0.60
Oza Bagging					
1 epoch	21.95±0.85	16.15±0.79	10.31±2.00	26.39±0.99	10.99±0.73
3 epochs	20.13±0.73	14.28±0.73	7.01±1.05	26.34±0.97	9.99±0.56
10 epochs	20.47±0.76	14.45±0.75	5.13±0.61	26.93±1.98	10.24±0.58
OBPM	$\tau = 0.45$	$\tau = 0.3$	$\tau = 0.55$	$\tau = 0.25$	$\tau = 0.5$
1 epoch	21.08±0.84	**14.85±0.84**	9.09±0.91	23.85±0.54	10.39±0.54
3 epochs	19.97±0.78	**13.90±0.74**	6.92±0.86	23.87±0.55	9.69±0.55
10 epochs	20.29±0.68	**13.73±0.75**	5.23±0.64	24.40±0.47	10.01±0.57
OBPM-voted	$\tau = 0.5$	$\tau = 0.5$	$\tau = 0.5$	$\tau = 0.3$	$\tau = 0.7$
1 epoch	**20.14±0.79**	18.12±1.02	**7.78±0.83**	22.87±0.45	11.76±0.67
3 epochs	19.79±0.75	14.70±0.82	**5.44±0.61**	22.73±0.47	10.22±0.55
10 epochs	20.14±0.72	14.63±0.85	**4.70±0.51**	23.42±0.46	10.36±0.59

3.3.1 Online voting on large data sets

The Adult and Web data sets used here are as described in Chapter 2. The Synthetic data set was modified to have a margin of $\kappa = 0.01$ rather than $\kappa = 0.05$, and the size of the data set was increased from 1000 to 10000. Two new data sets from the UCI repository were included: Hypothyroid and kr-vs-kp. The Hypothyroid data set consisted of two classes with 3163 feature vectors of dimension 25. The majority class was the negative with 95 percent of the total samples (Web is similar). The majority

class of the kr-vs-kp data set contained 52 percent of the total sample, the total sample with 3196 features vectors of 36 dimensions.

We notice from the results of Table 3.1 that for these large data sets there is no real difference between the two strategies of GOAM: Bagging and Majority vote. This is indicative of the fact that the individual Perceptrons have converged to where they are performing equally well on the data sets. Considering the online performance (a single epoch) of the various methods, the OBPM-voted gave the lowest test errors. By comparing the results of the two different online Bagging methods GOAM-Bagging and OZA-Bagging, GOAM-Bagging had superior test error results. This lends weight to the benefits of allowing the tuning of τ. The single epoch results of GOAM-Bagging were similar to OBPM, showing no clear advantage in aggregating the weights compared to the hypotheses (in the linear case) and the data sets are large. It was interesting to note that the voted Perceptron was one of the best performing methods. We see that the confidence interval were smaller in the case of the OBPM-voted. This was most noticeable when the data set was rather "noisy', as was the case in the Synthetic and Adult data sets.

Whilst the voted Perceptron did give a performance gain, it came at some cost of increased memory required. One potential solution is to prune the stored weights. For instance, in some cases storing weights where $v = 1$ may harm the final combined hypothesis. This is because after the initial update in the voted Perceptron the last weight is assumed to correctly classifier the updated example, therefore v is set to one, but there is no guarantee that after the update that the weight correctly classifies that example the next time. This suggest that forgetting the weights for $v = 1$ will not penalise the final solution of the voted Perceptron. To make the voted Perceptron practical in an online setting requires a way of pruning/discarding weights. For example pruning the weights where the number of consecutive correctly classified examples was less than 10 (hence survived less than 10 iterations) for the adult data set gave test errors of 14.88 ± 0.01 and 15.16 ± 0.05, for the OBPM-voted and the voted Perceptron respectively. Hence, the OBPM-voted was more resilient to pruning for this data set. From Figure 3.1 pruning in this way is equivalent to discarding up to the 77th percentile for OBPM-voted and 81st percentile for voted Perceptron. Note normalising the histogram of Figure 3.1 gives a distribution of v, which could be used as a tool for online pruning.

3.3.2 Kernel and real world data sets

The data sets were the same as those used in Section 2.4.3 of Chapter 2. The RBF kernel of (2.6) was used with $\varsigma = 0.5$.

Studying the single epoch results of Table 3.2 the OBPM and OBPM-voted had the smallest test errors in all but the Pima data set. The fact that GOAM-Majority had the best result for Pima data set was a bit of an anomaly which we are unable to explain. We see from the results of Table 3.2 that the voted Perceptron did not

perform well, in contrast to the linear setting. The OBPM-voted however achieved good results in both linear and non-linear settings. Comparing the two online Bagging techniques of GOAM-Bagging (Fern and Givan [2000]) and OZA-Bagging (Oza [2001]), there was a statistically significant difference in performance for Thyroid and Pima data sets, though as expected, for a setting of approximately $\tau = 0.6$, GOAM-Bagging had comparable performance to OZA-Bagging. Comparing the single epoch results of GOAM-Bagging and OBPM, there was lower test errors using OBPM against Heart, Sonar and Ionosphere data sets. This empirical comparison indicates there was some improvement in generalisation when using OBPM rather than online Bagging, especially when applying the kernel to the Perceptron. Comparing the OBPM soft boundary results of Table 2.3 (see Chapter 2) with results of OBPM-voted, we see that the OBPM with a soft boundary was better in all except the case of the Thyroid data set. This implies that the soft boundary case of the OBPM-voted would similarly perform better.

3.4 Summary

A number of online voting methods were considered in this chapter. We discussed two existing online learning methods for Bagging: one method, by Fern and Givan [2000], used the same random sampling trick of the OBPM of the last Chapter; the second method used by Oza [2001] had a different random sampling which was distributed according to the Poisson distribution with mean of one.

An online algorithm the Generalised Online Aggregate Method (GOAM), was presented which combined two methods for aggregating the hypotheses: Bagging and Majority-vote, which used the same random sampling technique of Fern and Givan [2000] and OBPM. We introduced a new extension of OBPM which incorporates the voted Perceptron, called the OBPM-voted. We showed that empirically OBPM-voted had improved generalisation performance compared to the other online voting methods considered here. The empirical evidence for a single epoch (online performance) showed that OBPM had favourable test error performance compared to GOAM-Bagging, especially when applying the kernel. One added advantage is that OBPM requires comparatively less memory than Bagging. The OBPM algorithm only has to store a single weight vector for implementation, whereas Bagging must store all N.

We showed in the case of the Adult data set that pruning applied to the OBPM-voted can be done without detrimental effects to the test errors. We showed that empirically in the case of the Perceptron the online Bagging proposed by Oza [2001] could be improved on by using the technique of Fern and Givan [2000]. We proposed a modification to the existing voted Perceptron: rather than setting the count of the correctly classified examples to one initially at every Perceptron update we propose to set it to zero. For very large data sets we propose that a regime of online pruning via a histogram (density of number of corrections) should be used with the voted Perceptron.

The Perceptron, Ranking and Collaborative Filtering

4.1 Introduction

In the last few chapters we have presented and discussed various alternative aggregate methods for online learning which are known to produce a linear classifier with a large margin. Investigated in this chapter are the same large margin classifier methods used in conjunction with the task of information retrieval and collaborative filtering. In applications like information retrieval and collaborative filtering we want to order or *rank* documents rather than (simply) classifying them into *relevant* and *non-relevant* documents, that is, we aim at finding mappings of instances to their correct ranks chosen from an *ordered* set of ranks $\{1, \ldots, k\}$. For example consider the information retrieval task of rating a particular query-document pair into relevant, possible relevant, and not relevant, which we model as an ordered set of 3. This example requires investigating the content of both the document and the query and using some similarity metric between the two, resulting in a ranked list of documents. An alternative approach to ranking the query-document pair is collaborative filtering or recommender systems (Resnick and Varian [1997]; Breese et al. [1998]) where the predicted rankings for query-document pairs are made by using the rankings (or judgements) made by people with similar interests or some expertise in ranking the pairs. We chose to focus on collaborative filtering, as this seems a natural way to rank items and avoid the issue of studying the content of the items.

Two different ways traditionally used in tackling rank learning are: either as a regression problem or a classification problem. However, in the regression setting a metric is required which converts the ranks to real values. Determining this metric is in general very difficult and makes regression algorithms very sensitive to the representation of the ranks rather than their pairwise ordering. On the other hand, classification algorithms completely ignore the ordering of the ranks by treating them as classes and thus require in general more training data. The main difficulty with regression, specifically SVM ordinal regression is the quadratic growth in the number of constraints as the training set size increases.

A recent rank learning algorithm motivated by the Perceptron is the PRank algorithm (Crammer and Singer [2002]). In contrast to other algorithms (see Herbrich et al. [2000]), which usually square the training set size by working on pairs of training examples, PRank requires a much smaller training set. For large data sets memory constraints make ranking methods which perform multiple runs through the data unmanageable. One example of such a ranking method, is the boosting method of RankBoost of Freund et al. [2003] which has good generalisation performance but requires multiple runs through the data set. We are therefore interested in online ranking learning algorithms that can be used for truly online problems, e.g. ranking web documents on the Internet.

Classification algorithms involving the Perceptron have been studied extensively, and recently there has been increased interest in using them to produce large margin solutions (as discussed in Chapter 1, e.g. the variants of the marginalised Perceptron (Duda et al. [2000]), Maximal margin Perceptron (Kowalczyk [2000]), ROMMA (Li and Long [2002]), ALMA (Gentile [2001]), and MIRA (Crammer and Singer [2003])). For example, ALMA guarantees a margin of $(1-\alpha)\gamma_{\mathbf{z}}$ after $O(\frac{1}{\alpha^2\gamma_{\mathbf{z}}^2})$ updates[1], where $\gamma_{\mathbf{z}}$ is the maximum achievable margin and $\alpha \in (0,1]$. Ensuring a large margin solution is desirable in providing the ability to handle concept drift as well as to control generalisation error. However, Herbrich et al. [2000] have shown that this carries over into the ranking learning task. Recently, Shashua and Levin [2003] showed two different Support Vector Machine (SVM) algorithms which maximised the margin resulting in a ranking learning algorithm with better generalisation performance compared to PRank. The first method of Shashua and Levin [2003] is the *fixed margin* approach, where the margin of the two closest neighbouring ranks (classes) is maximised. It turns out that the fixed margin approach is a direct generalisation of a SVM to rank learning. The second approach of Shashua and Levin [2003] is to maximise the sum of margins by defining the margin as the difference between two ranking thresholds, where there they define two thresholds for each of the $k-1$ ranks. The experiments of Shashua and Levin [2003] showed that both approaches outperformed existing ordinal regression for rank learning by maximising the margin using a SVM approach. Having a rank learning solution with a large margin is therefore desirable, but while remaining in the online setting. This motivates the search for a large margin variant of the online algorithm PRank.

In the next section, we formally define the ranking learning setting and recall the Perceptron ranking algorithm (PRank) of Crammer and Singer [2002]. Section 4.3 presents a large margin version of PRank, the Online Aggregate PRank-Bayes Point Machine (OAP-BPM). Based on the online voting methods of the last Chapter, online variants of the batch voting methods of Bagging (Breiman [1998]) and the voted Perceptron (Freund and Schapire [1999]) are also presented. Experimental results are given in Section 4.4 for a number of real world collaborative filtering data sets and a

[1]Note that this is assuming the instances to be normalised.

Algorithm 6 PRank (Crammer and Singer [2002])

Require: A training example at round t of $(\mathbf{x}_t, y_t^r)$ consisting of a rank $y_t^r \in \{1, \ldots, k\}$ and instance $\mathbf{x}_t \in \mathbb{R}^d$.
Require: Perceptron weights $\mathbf{w}_t \in \mathbb{R}^d$.
Require: The threshold set at round t, $\boldsymbol{c}_t = (c_t(1), \ldots, c_t(k-1), c_t(k) = \infty)'$.

1: predict $\hat{y}_t^r = \min_{r \in \{1,\ldots,k\}} \{r : \mathbf{w}_t \cdot \mathbf{x}_t - c_t(r) < 0\}$
2: **if** $\hat{y}_t^r \neq y_t^r$ **then**
3: **for** $r = 1$ to $k-1$ **do**
4: **if** $(y_t^r \leq r)$ **then** $l_t(r) := -1$ **else** $l_t(r) := 1$
5: **end for**
6: **for** $r = 1$ to $k-1$ **do**
7: **if** $(\mathbf{w}_t \cdot \mathbf{x}_t - c_t(r)) l_t(r) \leq 0$ **then**
8: $a_t(r) := l_t(r)$ **else** $a_t(r) := 0$
9: **end for**
10: **update** $\mathbf{w}_{t+1} := \mathbf{w}_t + \sum_{r=1}^{k-1} a_t(r) \mathbf{x}_t$
11: $\boldsymbol{c}_{t+1} := \boldsymbol{c}_t - \boldsymbol{a}_t$
12: **else**
13: $\mathbf{w}_{t+1} := \mathbf{w}_t$
14: $\boldsymbol{c}_{t+1} := \boldsymbol{c}_t$
15: **end if**
16: **return** Updated weights $\boldsymbol{c}_{t+1}$ and thresholds $\mathbf{w}_{t+1}$

synthetic ranking data set.

4.2 PRank in a nutshell

As background we provide the ranking definitions of Crammer and Singer [2002] relating to the PRank algorithm. We have a finite set of ranks $\mathcal{R} = \{1, \ldots, k\}$ from which a rank $y_t^r \in \mathcal{R}$ is assigned to an instance $\mathbf{x}_t \in \mathbb{R}^d$. We are concerned with the supervised learning setting where we have a training sequence of instance rank pairs $\mathbf{z} = ((\mathbf{x}_1, y_1^r), \ldots, (\mathbf{x}_T, y_T^r))$ and the goal is to find a ranking rule $\mathrm{H} : \mathbb{R}^d \to \mathcal{R}$.

The PRank algorithm is defined by rounds (iterations) of the PRank update (Algorithm 6). The ranking rule H of the PRank algorithm consists of the combination of Perceptron weights $\mathbf{w} \in \mathbb{R}^d$ and a threshold vector $\boldsymbol{c} = (c(1), \ldots, c(k-1))$, where it is assumed that $c(k) = \infty$. The objective of the PRank algorithm is to find a Perceptron weight vector $\mathbf{w}$ which successfully projects all the instances in $\mathbf{z}$ into the k subintervals defined by the thresholds $\boldsymbol{c}$, i.e. for the rank of r the subinterval is $c(r-1) < \mathbf{w} \cdot \mathbf{x} < c(r)$. This ranking procedure of the respective instances $\mathbf{x}_t$ is given by Algorithm 6. At round t of PRank the first step is to predict the rank y_t^r (line 1 of Algorithm 6) for a given instance $\mathbf{x}_t$ by selecting the smallest rank r such that $\mathbf{w}_t \cdot \mathbf{x}_t < c(r)$. If the prediction $\hat{y}_t$ is not the correct rank then a label of $l_t(r) = +1$ is allocated to those subintervals above

the target rank y_t^r and $l_t(r) = -1$ to those below[2] (lines 3, 4 and 5 of Algorithm 6). For each subinterval, if the ranking rule H consisting of $\mathbf{w}_t$ and $c_t(r)$ misclassifies the label $l_t(r)$ then the label is subtracted from the threshold $c_t(r)$, and the Perceptron is updated $\mathbf{w}_{t+1} = \mathbf{w}_t + l_t(r)\mathbf{x}_t$. Our intuition tells us that updating $\mathbf{w}$ and $\boldsymbol{c}$ in this way has the effect of trying to move the desired rank r and the updated predicted rank $\mathbf{w}_{t+1} \cdot \mathbf{x}_{t+1}$ closer together. This procedure is repeated for all the subintervals $r = 1, \ldots, k-1$ for round t. As an illustration of this intuition consider the example of Figure 4.1. In this example we have a choice of five possible ranks to apply to our instances, each rank corresponding to intervals on the real line, as shown in Figure 4.1. At iteration t, the correct rank y_t^r is 4 (indicated by the dot in the shaded region titled correct interval (blue dot for the colour version) in Figure 4.1), though our prediction $\hat{y}_t$ is 2 (indicated by the dot titled prediction (green dot for the colour version) in Figure 4.1). Lines 3,4,5 of Algorithm 6 convert the ranking problem to a binary classification problem by mapping those intervals below the target rank of 4 to a label of one, those equal and above to negative one. The thresholds $c_t(2)$ and $c_t(3)$ are between target rank and the predicted rank are reduced by one each. The Perceptron weights are modified according to the update rule of line 7, $\sum_{r=1}^{k-1} a_t(r) = 2$ moving the prediction closer towards the correct interval of 4, according to the number of rank intervals violated, in this case two.

Two important results which are significant when considering the benefits of the PRank algorithm are:

1. The ranking order is preserved (Crammer and Singer [2002]) between rounds, i.e. $c_{t+1}(r+1) \geq c_{t+1}(r)$ given this is true at round t.

2. The number of mistakes (updates) made by this algorithm is at most $(k-1)(\beta^2+1)/\gamma_r^2$, where $\beta = \max_t \|\mathbf{x}_t\|$ and margin $\gamma_r = \min_{r,t}\{(\mathbf{w}^* \cdot \mathbf{x} - c^*(r))l_t(r)\}$ if there exists a rank rule pair $\mathbf{w}^*$ and $\boldsymbol{c}^*$ such that $\gamma_r > 0$.

In other words, we know that the learning algorithm is exploiting the order of the ranks (property 1) and converges as fast as a multi-class Perceptron learning algorithm (property 2).

4.3 Improved generalisation (large margin) versions

Although for the PRank algorithm the number of updates required to correctly rank the training sequence is bounded, there is no guarantee on the size of the margin of PRank as the Perceptron (of PRank) only guarantees $\gamma_r > 0$. At first glance a large margin could be produced with a variant of the Perceptron mentioned in the introduction. However, there is the added complication of the thresholds of $\boldsymbol{c}$ which are central to ranking the instances.

[2]Note that we exclude the kth subinterval because $c_t(k) = \infty$ for all t, so clearly $l_t(k) = 1$.

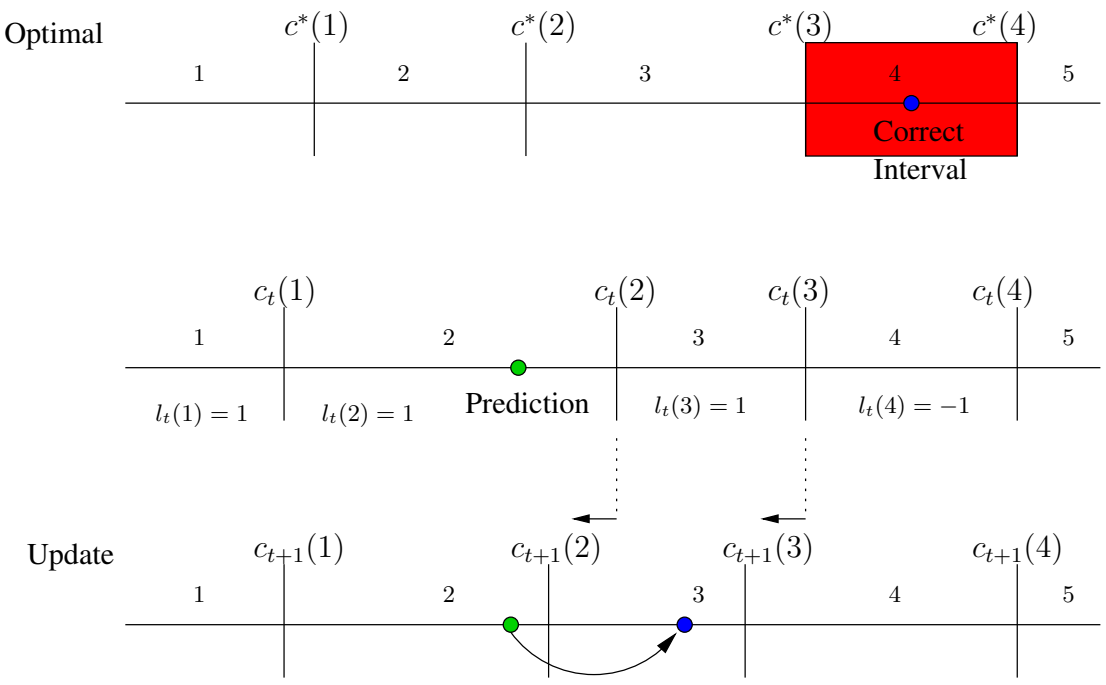

Figure 4.1: Illustrative example of an update at iteration t of the PRank algorithm.

Two approaches to develop a large margin version of PRank are considered: in Section 4.3.1 we look a Bayes Point approach, in Section 4.3.2 we consider online voting methods.

4.3.1 Bayes point approach

As discussed in Chapters 1 and 2, methods which achieve the maximum margin solution on the training examples do not guarantee the same generalisation performance as the Bayes point (Herbrich, 2000). Recall from Chapter 1 that unlike the optimal Bayes classifier the Bayes point classifier is chosen from the fixed class of classifiers $\mathcal{H}$ which is dependent of the training sequence $\mathbf{z}$. In an effort to approximate the Bayes point for the rank learning problem in a truly online way we implement the same randomisation trick of Algorithms 3 and 4 in conjunction with the PRank algorithm. Similar to Algorithms 3 and 4 we train on a sequence $\boldsymbol{z}$ (in parallel) N Perceptrons using PRank. We create diversity between the different PRank algorithms final solutions by presenting randomly each sample $\mathrm{z}_t = (\mathbf{x}_t, y_t^r)$ to each PRank Perceptron j according to whether $b_{j,t} = 1$, where $b_{j,t}$, $j = 1, \ldots, N$, $t = 1, 2, \ldots$ are independent Bernoulli random variables with $\Pr\{b_{j,t} = 1\} = \tau$.

In line 12 of Algorithm 7 we take the equal weighted combination of the weight vectors of the respective PRank algorithms to estimate the Bayes point. It is natural that in order to achieve a good generalised solution and truly estimate the Bayes point

Algorithm 7 OAP-BPM algorithm

Require: A training sample
$\boldsymbol{z} = ((\mathbf{x}_1, y_1^r), \ldots, (\mathbf{x}_T, y_T^r))$.
Require: A online learning algorithm PRank$(\boldsymbol{c}_{j,t}, \mathbf{w}_{j,t}, \mathbf{x}_t, y_t^r)$.
Require: A subroutine Bernoulli(τ) which returns independent Bernoulli random variables with probability τ of taking the value 1.
Require: Parameters $N \in \mathbb{N}$ and $\tau \in (0, 1]$.

Initialise weights $\mathbf{w}_{j,1} := \mathbf{0}$ and thresholds $\boldsymbol{c}_{j,1} := \mathbf{0}$.
for $t = 1$ to T **do**
  for $j = 1$ to N **do**
    $b_{j,t} :=$ Bernoulli(τ)
    if $b_{j,t} = 1$ **then**
      $\mathbf{w}_{j,t+1}, \boldsymbol{c}_{j,t+1} :=$ PRank$(\boldsymbol{c}_{j,t}, \mathbf{w}_{j,t}, \mathbf{x}_t, y_t^r)$
    else
      $\mathbf{w}_{j,t+1} := \mathbf{w}_{j,t}$
      $\boldsymbol{c}_{j,t+1} := \boldsymbol{c}_{j,t}$
    end if
  end for
  $\tilde{\mathbf{w}}_{t+1} := \frac{1}{N} \sum_{j=1}^{N} \mathbf{w}_{j,t+1}$
  $\tilde{\boldsymbol{c}}_{t+1} := \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{c}_{j,t+1}$
end for
return $\mathrm{H}(\mathbf{x}) := \min_{r \in \{1,\ldots,m\}} \{r : \tilde{\mathbf{w}}_{T+1} \cdot \mathbf{x} - \tilde{c}_{T+1}(r) < 0\}$

for ranking, we also have to take an equally weighted combination of threshold vectors of the N PRank algorithms. Interestingly, combining the threshold this way still preserves the order of the thresholds between updates, as each PRank algorithm maintains $c_j(1) \leq \ldots \leq c_{j,t}(k-1) \leq c_{j,t}(k)$ for all $j = 1, \ldots, N$, which implies that $\sum_{j=1}^{N} c_j(1) \leq \ldots \leq \sum_{j=1}^{N} c_{j,t}(k-1) \leq \sum_{j=1}^{N} c_{j,t}(k)$.

In order to see why $\tilde{\boldsymbol{c}}_t = \sum_{j=1}^{N} \boldsymbol{c}_{j,t}/N$ makes sense, taking the same example of Figure 4.2 but now in the context of Algorithm 7 OAP-BPM. After t iterations we consider two PRank algorithms outputs $\mathbf{w}_{j,t+1}$ and $\boldsymbol{c}_{j,t+1}$, for $j = 1, 2$ and that t is large enough such that both PRank algorithms solutions correctly rank all the training sequence $\mathbf{z}$. As shown in Figure 4.2 both PRank solutions predict the rank correctly. One could imagine, as shown in Figure 4.2, that the threshold $\boldsymbol{c}_{1,t+1}$ differs from $\boldsymbol{c}_{2,t+1}$ and their predictions (dots in the subintervals $c_{j,t+1}(3)$ and $c_{j,t+1}(4)$ for $j = 1$ and 2 respectively, red and green dots respectively in the colour version) lie close to the thresholds. In this example both the PRank solutions are likely to make more mistakes to unseen examples compared to being in the centre of the optimal intervals defined by the optimal thresholds $\boldsymbol{c}^* = (c^*(1), c^*(2), c^*(3), c^*(4))$. We can see from Figure 4.2 that the average $\tilde{\boldsymbol{c}}_t$ produces thresholds which are closer to the optimal thresholds. One can imagine that producing predictions which are in the centre of the rank intervals ensures

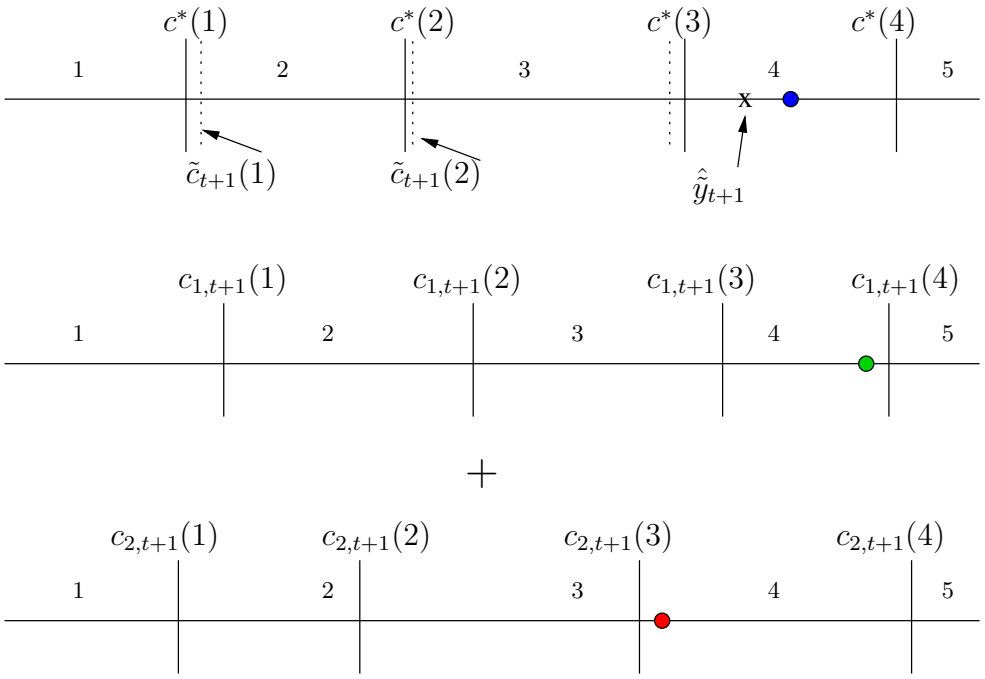

Figure 4.2: Illustration as to why OAP-BPM (Online Aggregate PRank-Bayes Point Machine) gives improved generalisation.

robustness to unseen examples, as seen by the combined estimate $\hat{\tilde{y}}_{t+1}$ of Figure 4.2.

A kernel implementation of OAP-BPM is an easy modification of the result of the OBPM of Section 2.2.1 of Chapter 2. Each Perceptron f_j, $j = 1, \ldots, N$ by the representer theorem is defined as $f_j(\cdot) = \sum_{t=1}^{T} \Upsilon_{j,t} K(\mathbf{x}_t, \cdot)$ where $\Upsilon_{j,t} = b_{j,t} \sum_{r=1}^{k-1} a_{j,r,t}$. Combining this with the reproducing property gives $\tilde{f}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} f_j(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \Upsilon_{j,t} K(\mathbf{x}_t, \mathbf{x})$. Similar to the OBPM, for a truly online algorithm one needs to bound the number of instances $\mathbf{x}_t$ required to store the kernel expansion.

We expect OAP-BPM to take longer to converge since the mistake bound of OAP-BPM is driven by the mistake bound of worst solution (smallest γ_r) amongst the N PRank algorithms. This is not a bad thing as OAP-BPM will continue to learn after the Perceptron based PRank stops. This method of creating diversity by τ therefore has a trade-off of achieving a better generalisation performance (large margin) and with the convergence of the algorithm to the largest margin possible. The smaller τ the greater the diversity but the slower the convergence to the final target margin. This method can be applied in an online parallel fashion making it well suited to large scale ranking with a parallel implementation.

4.3.2 Online voting methods

In the previous section we considered using the online large margin method of Chapter 2 with PRank. In a similar vein, we will now consider the use of the online voting of the

last Chapter's algorithm GOAM to form an aggregation of PRank produced classifiers.

As mentioned previously, the OAP-BPM is different to ensemble methods, like Bagging. Whilst Bagging in general is considered a batch learning method, both Fern and Givan [2000] and Oza [2001] presented online learning versions. The online Bagging method proposed here is essentially the same as GOAM-Bagging (originally by Fern and Givan [2000] though their study was primarily interested in decision trees). The only difference also to OAP-BPM of the last section is in the lines 12,13. In our case, the Online Aggregate PRank-Bagging (OAP-Bagging) the final hypothesis (ranking rule) is

$$\mathrm{H}(\mathbf{x}) = \sum_{j=1}^{N} h_j(\mathbf{x})/N, \tag{4.1}$$

where $h_{j,T+1}(x) := \min_{r\in\{1,\ldots,m\}}\{r : \mathbf{w}_{j,T+1}\cdot\mathbf{x} - c_{j,T+1}(r) < 0\}$. We did not consider the online Bagging method of Oza [2001] given its weaker empirical performance in the last Chapter.

Whilst Bagging is one voting method worth investigating the another is the Majority vote. Like GOAM-Majority of Chapter 3 we combine/aggregate N independent PRank algorithms, sampled again using the same technique used in OAP-BPM. We store the number of correct ranks made by each PRank algorithm, v_i, and aggregating them such that the final hypothesis (ranking rule) is

$$\mathrm{H}(x) = \frac{\sum_{i=1}^{N} v_i h_i(x)}{|\sum_{i}^{N} v_i|}, \tag{4.2}$$

where $h_{j,T+1}(x) := \min_{r\in\{1,\ldots,m\}}\{r : \mathbf{w}_{j,T+1}\cdot\mathbf{x} - c_{j,T+1}(r) < 0\}$. Note that we normalise the Majority vote by v, since rank learning in general is sensitive to the scale, which is not true in the case of binary classification. We refer to this algorithm as OAP-Majority, since we combine the Majority vote with the online sampling of OAP-BPM.

It is well accepted that Bagging and the Majority vote perform well in the classification problem setting: Bagging especially when the base classifier is unstable, meaning it experiences large variations in its final hypothesis. The question is how does OAP-Bagging and OAP-Majority compare with OAP-BPM in the rank learning setting? This question motivates the empirical study in the experimental section.

4.4 Experiments and results

Experiments comparing PRank, the regression algorithm of Widrow and Hoff [1960] (WH) with the OAP-BPM (Bayes Point Machine) and the ensemble variants OAP-Bagging (Bagging) and OAP-Majority (Majority vote), were performed on a synthetic ranking problem and using collaborative filtering on several real-world data sets.

Table 4.1: Synthetic data set experimental results produced by test sample (not used in training), showing the averaged rank loss, $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t^r|$, where T is the test set size and their corresponding 95 percent confidence intervals with the Student's t-distribution for OAP-Majority (Majority vote), OAP-Bagging (Bagging) and OAP-BPM (Bayes Point Machine) for various settings of τ.

τ	OAP-Majority	OAP-Bagging	OAP-BPM
0.3	0.32±0.01	0.33±0.01	0.23±0.01
0.6	0.31±0.02	0.31±0.02	0.24±0.03
0.9	0.31±0.03	0.32±0.03	0.26±0.03

4.4.1 Ranking

The synthetic data experiment of Herbrich et al. [2000], and Crammer and Singer [2002], was performed in the batch setting with a non-homogeneous polynomial kernel of degree two. To generate the data each instance at round t, $\mathbf{x}_t = (x_t(1), x_t(2))$ was chosen according to the uniform distribution on the unit square i.e. $\mathbf{x} \in [0,1]\times[0,1] \subset \mathbb{R}^2$. The ranks $1,\ldots,5$ were assigned by $y^r = \max_{r\in\{1,\ldots,5\}}\{r : 10((x(1)-0.5)(x(2)-0.5))+n > c_r\}$ with the rank thresholds $\boldsymbol{c} = (-\infty, -1, -0.1, 0.25, 1)$ with $c(5) = \infty$ and n normally distributed with zero mean and standard deviation of 0.125. The experiment used a polynomial kernel $K(x_1, x_2) = ((x_1 \cdot x_2) + 1)^2$ with 20 Monte-Carlo trials with 50000 training examples and a separate test set of 1000 examples.

To study the online nature of the ranking algorithms we used the same measure of performance as Crammer and Singer [2002], the average rank losses of $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t^r|$, where T is the number of rounds performed so far. Figure 4.3 shows the average rank losses from 20 trials for PRank, OAP-BPM ($N = 100$ and $\tau = (0.3, 0.6, 0.9)$), Widrow-Hoff (plotting the lowest losses with learning rates/step sizes of $\eta = (0.2, 0.1, 0.05, 0.01)$) and OAP-Bagging ($N = 100$ and $\tau = (0.3, 0.6, 0.9)$) and OAP-Majority ($N = 100$ and $\tau = (0.3, 0.6, 0.9)$). The prediction of the true rank y_t^r for WH at each iteration t was determined according to $\hat{y}_t = \min_{r\in\{1,\ldots,5\}}\{r : |r - (\mathbf{w}_t \cdot \mathbf{x}_t)|\}$. We achieved significantly better results for WH than reported in Crammer and Singer [2002], as they only tried $\eta = 1$. WH result with $\eta = 0.1$ is slightly better average rank loss than PRank after 5000 training examples. It is not that surprising that WH is better than a mistake driven Perceptron, as the squared loss function for WH can produce a large margin like solution. Comparing the five different algorithms the OAP-BPM had the lowest averaged rank loss after the 5000 examples, though OAP-Majority was the lowest until 1000 examples.

We then took the test set of 1000 examples and calculated the averaged rank losses for the same five algorithms PRank, WH, OAP-BPM, OAP-Bagging and OAP-Majority, plus we tried PRank with the Majority vote. The averaged rank loss results of the algorithms with their 95% confidence intervals for a Student's t-distribution were: PRank 0.37 ± 0.07, PRank with voted Perceptron 0.31 ± 0.00, with $\eta = 0.1$ WH 0.30 ± 0.2 and

Table 4.1 shows the rest for $\tau = (0.3, 0.6, 0.9)$. The results from the test examples show that OAP-BPM had the lowest averaged rank loss over the three choices of τ. From Table 4.1 we see that as τ was made smaller so was its confidence interval. It can be seen that there is a difference in the final rank loss results in Figure 4.3 and Table 4.1. This was due to the fact that the test results of Table 4.1 were produced by the final weight solutions.

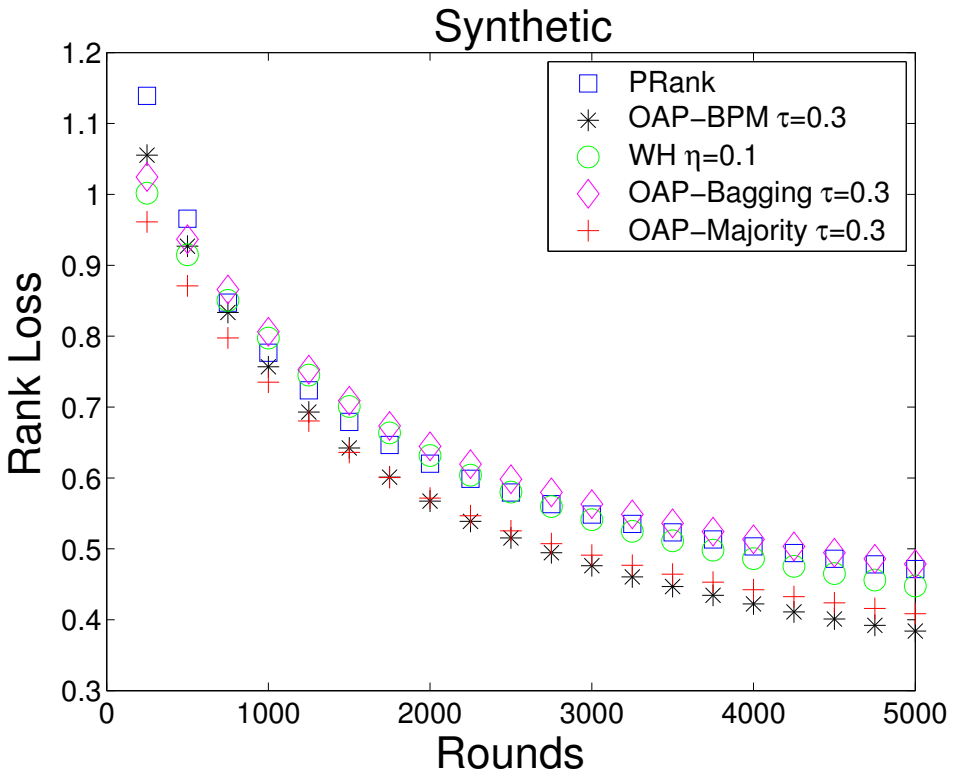

Figure 4.3: Experimental results for synthetic data set comparing the rank learning algorithms PRank, WH with step size $\eta = 0.1$, OAP-BPM with Bernoulli probability $\tau = 0.3$, OAP-BPM ensemble variants OAP-Bagging (Bagging) and OAP-Majority (Majority vote).

4.4.2 Collaborative filtering

To allow for a fair comparison, the general framework used for the collaborative filtering experiments was consistent for all three data sets: OHSUMED [3], cystic fibrosis [4] and EachMovie [5]. We constructed a training set and test set, where given an item (i.e. query-document pair) to be ranked, at first the target rank y_t^r was drawn randomly from

[3] Available at `http://ftp.ics.uci.edu/pub/machine-learning-databases/ohsumed`
[4] Available at `http://www.sims.berkeley.edu/~hearst/irbook/cfc.html`
[5] Available at `http://www.research.compaq.com/SRC/eachmovie/`

Table 4.2: Collaborative filtering experimental results produced by test sample (not used in training), showing the averaged rank loss, $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t^r|$ where T is the test set size and their corresponding 95 percent confidence intervals with the Student's t-distribution.

ALGORITHM USED	OHSUMED	CYSTIC FIBROSIS	EACH-MOVIE
WH			
$(\eta = 0.2)$	0.43±0.01	0.75±0.02	2.19±0.06
$(\eta = 0.05)$	0.38±0.01	0.48±0.02	1.92±0.07
$(\eta = 0.01)$	0.38±0.01	0.43±0.02	2.25±0.08
$(\eta = 0.001)$	0.38±0.01	0.41±0.02	2.74±0.10
PRANK	0.53±0.02	0.50±0.02	1.06±0.03
PRANK-VP	0.49±0.00	0.52±0.01	1.13±0.03
OAP-MAJORITY			
$(\tau = 0.1)$	0.47±0.01	0.47±0.01	0.95±0.03
$(\tau = 0.2)$	0.46±0.01	0.45±0.01	0.95±0.03
$(\tau = 0.3)$	0.46±0.01	0.45±0.01	0.95±0.03
OAP-BPM			
$(\tau = 0.1)$	0.41±0.01	0.40±0.01	0.89±0.03
$(\tau = 0.2)$	0.41±0.01	0.39±0.01	0.89±0.03
$(\tau = 0.3)$	0.41±0.01	0.40±0.02	0.90±0.03
OAP-BAGGING.			
$(\tau = 0.1)$	0.53±0.01	0.50±0.01	0.96±0.03
$(\tau = 0.2)$	0.50±0.01	0.48±0.01	0.95±0.03
$(\tau = 0.3)$	0.49±0.01	0.47±0.01	0.96±0.03

the ratings made on that item and then the remaining ratings were used as dimensions of the instance vector $\mathbf{x}_t$. All the results in this section are averages produced by 500 Monte-Carlo trials.

The details of the experimental setup for each data set used are as follows:

- The OHSUMED data set consists of 106 medical queries constructed by novice doctors. An information retrieval software package given each query returned a list of perceived relevant documents which were extracted from the MEDLINE medical database. Each of the query-document pairs was judged either definitely relevant, possibly relevant, or not relevant by three different sets of people which we converted to ranks 3,2,1 respectively. Note that not all of the three sets of people judged every query-document pair. We selected from the query-document pairs only those where at least two of the possible three sets of people had ranked the pair. To construct the training set, at each round we drew at random a different query-document pair, drew at random one of the three peoples ranks as the target rank and used the other two to construct the instance (noting zero was allocated when there was no rating provided). The size of the training set used was 1000 examples and the test set size was 100 examples.

- The cystic fibrosis data set is similar to OHSUMED in that it consists of query-document pairs with three possible ratings of highly relevant, marginally relevant and not relevant, again assigning ranks of 3,2,1 respectively. The difference to OHSUMED is that we have four ratings for each query-document pair, hence the instances are of three dimensions. The training and test set sizes were 4247 and 582 respectively.

- The EachMovie data set consists of ratings from a possible 1628 movies (films and videos) where 72916 people were involved in assigned scores (this is larger than 61265 people used by Crammer and Singer [2002]): $0, 0.2, 0.4, 0.6, 0.8, 1$ to subsets of the possible movies. A similar experiment to Crammer and Singer (2002) was performed, which considered only people who rated over 300 movies, totalling 547 people. To allow for a score of zero for unseen movies 0.5 was subtracted from the peoples scores. For each Monte-Carlo trial a target rank y was assigned by using the rank $\{1, \ldots, 6\}$ of a person drawn randomly from the possible 547. Each round is represented by the first 300 movies scored by the target person. The instances (features) $\mathbf{x}$ were formed by going through the remaining 546 people for each movie seen by the target person. For a given movie each person's score represents a different dimension of the instance (dimension is zero if that person didn't see the movie this is equivalent to an indifferent rating). Forming the instances and target rank this way we can predict the score of a random person from the scores of the rest of the people. A training set of size 210 was selected at random and the remaining 90 movies were used as the test set. Note that the collaborative filtering formulation of the Each Movie data set is different to the previous two data sets where they predict the ratings of a different target at each round.

From the results of Table 4.2 in two of the collaborative filtering experiments OAP-BPM had the lowest average rank loss for the test sets. In OHSUMED result OAP-BPM was close to the better WH, though this would be considered the easier of the three experiments with only a dimension of two.

Figures 4.4 (a), (b) and (c) give some insight into the convergence behaviour of the five different rank learning methods on the training examples. We see that the OAP-Majority has a faster convergence than both OAP-Bagging and OAP-BPM. We also notice that for the smaller rank set of three and lower dimensional data sets of OHSUMED and cystic fibrosis, the WH regression algorithm had the fastest convergence and comparable performance to the OAP-BPM on the test examples. Yet even though the convergence is slow the results of Table 4.2 show that the Perceptron based OAP-BPM had the best performance overall. It is not surprising that the test set results for OAP-BPM of Table 4.2 are better than the training set results in Figures 4.4 (a), (b) and (c), for two reasons: the test set results are using the final trained weights and the OAP-BPM has larger rank losses at the start making the average rank loss higher, since each rank loss at time t has equal weight over the entire training set.

4.5 Summary

We proposed a simple OAP-BPM rank learning algorithm which improves the generalisation performance of the PRank algorithm. The improved generalisation performance is achieved by endeavouring to have the rank prediction $\mathbf{w}_{t+1} \cdot \mathbf{x}_{t+1}$ in the centre of the subinterval true rank y_t^r. To estimate the centre of the subinterval we enable diversity amongst the ranking rules produced by N different PRank algorithms run in parallel by a simple online randomisation trick. It was demonstrated that averaging the N rank rules (producing the final ranking rule of the OAP-BPM algorithm) had a lower averaged rank loss for a separate test set of examples compared to the algorithms of Widrow-Hoff, PRank and ensemble versions OAP-Bagging and OAP-Majority for a number of experiments. The experiments involved rank learning on a synthetic and the collaborative filtering on the real world data sets of EachMovie, OHSUMED and cystic fibrosis. The advantage of OAP-BPM is improved generalisation performance compared to PRank whilst remaining in an online learning setting. There is an added computational cost in achieving a large margin, which on average is an order of $O(\tau N)$ more than the original PRank algorithm. The method of the OAP-BPM algorithm has two main advantages over ordinal regression in improving the generalisation of the ranking problem: 1) does not require a complex optimisation routine, 2) is memory efficient, since the number of constraints for ordinal regression grows quadratically with the size of the training set.

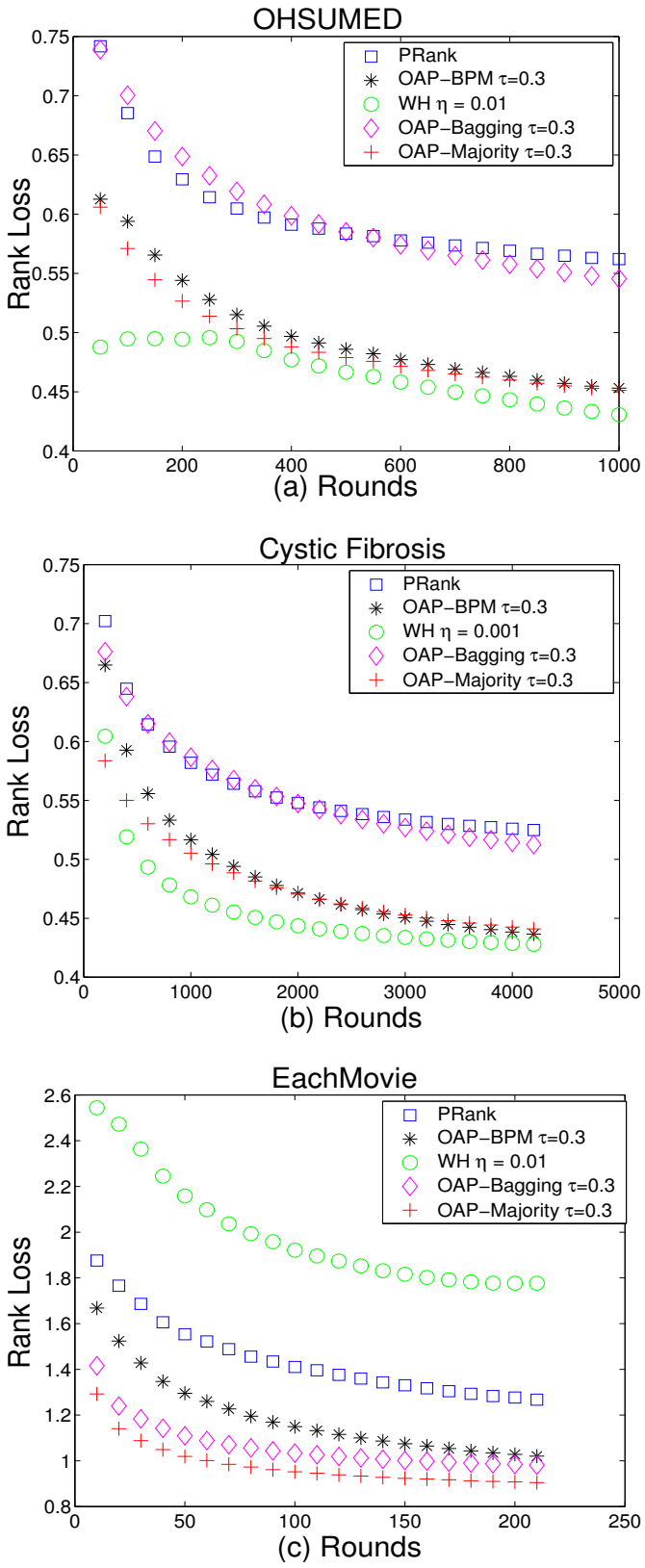

Figure 4.4: Collaborative filtering experimental results, comparing the averaged rank loss $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t^r|$, where T is the number of rounds of training performed, for the PRank, WH, OAP-BPM, OAP-Bagging, and OAP-Majority algorithms, against the three data sets (a) OHSUMED, (b) cystic fibrosis, and (c) EachMovie.

Adaptive Equalizer Aggregation

5.1 Introduction

A standard technique for correcting Inter-Symbol Interference (ISI) caused by a communications channel is to apply an equalizer at the receiver (Proakis [2001]). The motivation for applying an equalizer prior to symbol decisions in a communications receiver is that it should lead to lower error rates. In this chapter an equalizer is trained with a known sequence where the received signal is corrupted by channel noise. There are two distinct ways of formulating equalization algorithms to reduce ISI: regression where an equalizer is used to predict the transmitted symbol, and classification where an equalizer is used to correctly classify the transmitted symbol.

The main goal of the equalizer techniques presented in this chapter is to reduce the ISI levels of traditional adaptive equalizers; making the required training sequence shorter. Reducing ISI with a shorter training sequence and making it more robust to things like symbol decision errors: equating to quicker signal acquisition. The quicker the signal acquisition the better the bandwidth utilisation.

The most frequently used approach to equalization takes the form of a regression problem. The aim of the regressor is to form a prediction of the true symbol. A lot of research has been performed into different regression methods for equalization; we focus on a subset of them referred to as stochastic gradient methods. Stochastic gradient methods, like Least Mean Squared (LMS), exhibit randomness in the equalizer weights $\mathbf{w}_t$ due to the presence of noise. The randomness of the weights caused by the presence of noise makes an estimate of the gradient based on a single sample prone to updating $\mathbf{w}_t$ in the wrong direction. The error resulting from $\mathbf{w}_t$ being updated in the wrong direction is reduced by making the step-size (η) small, but the convergence is slow. Gardner [1984] showed that averaging the gradients over B data points reduced the effect of the noise, but there was still a trade off with convergence rate. Another method known to reduce the effect of noise is to apply a low pass filter to the gradient estimate (Proakis [2001]); reducing the effect of noise by adding momentum to the weights.

The last three chapters have demonstrated that the Perceptron with aggregation produces a large margin solution giving the Perceptron added immunity to noise. Up to

now we have considered aggregation in the classification setting; we now extend the use of aggregation to the regression setting. A number of aggregate methods for regression are investigated in this chapter: in particular Bagging (as discussed in Chapters 1) and its effectiveness when applied to channel equalization. We propose an equalizer in the regression setting which uses a buffered approach to Bagging.

There have been several pattern classification approaches to equalization: Chen et al. [1993b], Chen and Harris [2000], Chen et al. [1993a], Pérez-Cruz and Artés-Rodríguez [2002], Sebald and Bucklew [2000] and Santamaría et al. [2003]. We know that in general a classifier with a large margin results in good test error performance. Several questions arise: How well does a large margin in a pattern classification setting translate to the problem of equalization? How do pattern classifier equalizers compare to their regression counterparts? Several recent papers have considered these questions: Pérez-Cruz and Artés-Rodríguez [2002], Sebald and Bucklew [2000], Santamaría et al. [2003]. Although they were primarily interested with non-linear solutions which require optimisations which are both computationally complex and offline. In this chapter we are concerned with linear equalizer solutions which allow for better adaptive behaviour. For this reason we consider the online linear classification methods of previous chapters: OBPM of Chapter 2, and online voting methods of Chapter 3.

5.2 System model

The basic signal model consists of a binary signal (BPSK), $y_t \in \{-1, +1\}$, transmitted at time t (we only consider the binary case in this thesis; this could be extended to complex or multi-class, suitable for quadrature modulation types). As shown in Figure 5.1, y_t is transmitted over a communications channel $\mathbf{a} = [a_0, \ldots, a_{L-1}]'$, resulting in the signal at the receiver r_t,

$$r_t := \sum_{l=0}^{L-1} y_{t-l} a_l + n_t, \tag{5.1}$$

where n_t is the noise, which is typically additive white Gaussian noise (AWGN) with zero mean. We consider a linear equalizer of length $d = 2K + 1$ at time step t is given as $\mathbf{w}_t = [w_{t,1}, w_{t,2}, \ldots, w_{t,d}]'$. The equalizer produces an estimate $\hat{y}_t$ of the transmitted signal y_t via

$$\hat{y}_t := \sum_{i=1}^{d} w_{t,i} x_{t,i}, \tag{5.2}$$

where the instances are $\mathbf{x}_t = [r_{t-K}, \ldots, r_t, \ldots, r_{t+K}]'$. We assume that the transmitted labels $y_1, \ldots, y_T$ are known at the receiver, since in this chapter we presume that the equalization is non-blind therefore utilises a training sequence.

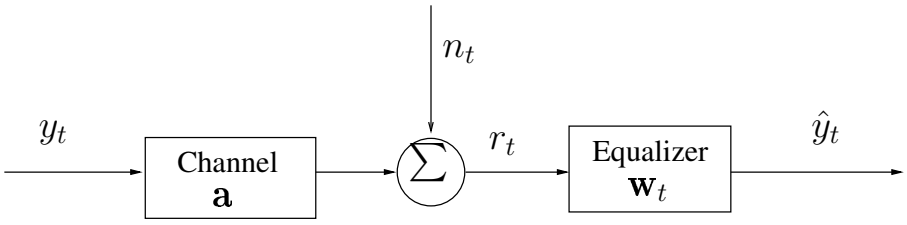

Figure 5.1: Communications system discrete time model with channel equalization applied at the receiver.

5.3 Stochastic gradient methods

The standard stochastic gradient descent method which approaches the optimal equalizer makes steps scaled by $\eta \in \mathbb{R}^+$ in the direction of the negative gradient of the cost function J,

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \frac{\partial \mathrm{J}(\mathbf{w}_t)}{\partial \mathbf{w}_t}. \tag{5.3}$$

We consider two different families of cost functions. The first family consists of the regression methods using the cost functions of *least mean squared* (LMS) (Widrow and Hoff [1960]) and *least mean absolute* (LMA) which is known to be more robust to outliers than LMS (Bishop [1995]). The LMS algorithm's cost function approximation is given by

$$\mathrm{J}_{\mathrm{LMS}}(\xi_t) := \xi_t^2. \tag{5.4}$$

where residual $\xi_t = y_t - \mathbf{w}_t \cdot \mathbf{x}_t$.

A large body of research has been devoted to numerous variations of the LMS algorithm (Haykin and Widrow [2003] and Proakis [2001]). One recent modification to LMS the *normalised* LMS (NLMS) algorithm (Haykin and Widrow [2003]) is known for its improved convergence compared to LMS. In NLMS the gradient descent update (5.3) is normalised by the instance vector making

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \frac{\partial \mathrm{J}(\xi_t)}{\partial \mathbf{w}_t} \frac{1}{\mathbf{x}_t{}' \mathbf{x}_t}, \tag{5.5}$$

where $\mathbf{x}'$ indicates transpose of $\mathbf{x}$. The LMA approximate cost function is given by

$$\mathrm{J}_{\mathrm{LMA}}(\xi_t) := |\xi_t|. \tag{5.6}$$

Comparing the cost functions of (5.4) and (5.6) we see that (5.6) is much less sensitive to large estimate errors which result from outlier training examples compared to LMS. However, LMS has better convergence properties when the noise effects are small. In an effort to combine the positive features of the two approaches, and avoid their negative ones, one can consider using the Huber loss function instead. The Huber loss function

for some $\epsilon > 0$ is

$$\mathrm{J}_{\mathrm{Huber}}(\xi_t) := \begin{cases} \frac{\xi_t^2}{2} & \text{if } |\xi_t| \leq \epsilon, \\ \epsilon|\xi_t| - \epsilon^2 & \text{if } |\xi_t| > \epsilon. \end{cases} \tag{5.7}$$

The Huber loss has a lower final MSE compared to LMA when a regime of diminishing step sizes is used. The main aim of equalization is to reduce the ISI such that our symbol error rate is zero (this is sometimes referred to as opening the eye). As we aim to only open the eye and is not necessary to achieve the lowest possible MSE; we restrict our study to the simpler LMA and LMS.

One limiting factor of the above gradient descent methods in a regression setting is achieving a small MSE is dependent of the choice of step-size. There is a trade-off between the speed of convergence and the final MSE. A small step-size results in a small final MSE, but has a slower convergence, and the converse is true for a larger step-size. Convergence of gradient descent in the deterministic setting is guaranteed provided the step-size is $0 < \eta < \Lambda_{\max}$, where $\Lambda_{\max}$ is the largest eigenvalue of the autocorrelation matrix of the received signal, Γ. The autocorrelation matrix Γ given the system model of Section 5.2 is (Proakis [2001])

$$\Gamma_{jl} := \begin{cases} \sum_{n=0}^{d} a_n a_{n+j-l} + N_0\delta_{jl} & \text{if } |l-j| \leq d, \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

where $N_0\delta_{jl}$ is the covariance of the noise sequence $\{n_t\}$ and is equal to $\frac{1}{2}\mathrm{E}(n_j^* n_l)$ (δ_{jl} is the Kronecker delta function).

In practice a choice of η close to $\Lambda_{\max}$ in the stochastic case can lead to instability. The instability of stochastic gradient methods in the regression setting is one motivating factor for the use of gradient methods like the Perceptron, where instability is not an issue. The only restriction on the learning rate of the Perceptron is $\eta > 0$. This brings us to consider the second family of cost functions (classification): one member of that being the marginalised Perceptron (as discussed in Chapter 1). The marginalised Perceptron's approximate cost function is

$$\mathrm{J}_{\mathrm{Perc}}(\xi_t) := -\sigma(\xi_t)\,\xi_t, \tag{5.9}$$

with residual $\xi_t = \rho - y_t \mathbf{w}_t \cdot \mathbf{x}_t$, $\sigma(\xi_t) = 1$ when $\xi_t \leq 0$ and zero otherwise, and ρ is the induced margin ($\rho = 0$ is the mistake driven standard Perceptron).

5.4 Aggregation methods for equalization

We will now consider aggregation methods for equalization. We assume that we have a sequence of T training examples $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)))$. Our discussion of aggregation methods is split into two different paradigms: regression and classification.

5.4.1 Regression

In Chapter 3 the aggregating technique of Bagging was applied to the problem of pattern classification. Bagging for regression (and for classification) was first presented by Breiman [1996]. In the regression setting the final Bagging predictor is an aggregation of N predictors. Each predictor $j = 1, \ldots, N$ is formed from a different re-sampling with replacement of the training sequence $\mathbf{z}_j$. This technique is referred to as bootstrapping (see Chapter 1). The success of bootstrapping relies on how good the training sequence's distribution matches the true distribution of examples (Breiman [1996]). Any difference can result in bias in different bootstrapped equalizers. We know that a communications system is a special case where the distribution of examples is intentionally made close to uniform to give some added robustness to bit errors caused by noise by applying randomisation to the bit stream prior to transmission. Consider a variation on the usual approach of bootstrapping by forming B examples drawing without replacement rather than with replacement. Using without replacement to create the training examples for each predictor results in a single training example for each received symbol, therefore removing any chance of altering the effective sampling rate. The jth predictor's bootstrapped sequence $\pi_j(B)$ is formed from random permutations of B examples. The resulting Bagging predictor formed from N bootstrapped sequences is given by $f_{\text{Bagging}}(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N} f_{\pi_j(B)}(\mathbf{x})$. This is an approximation of the expected predictor over all possible bootstrapped sequences denoted by $E_{\pi_j(B)} f_{\pi_j(B)}(\mathbf{x})$.

What motivates the approach of Bagging? The analysis of Breiman [1996] goes some way to explaining the benefits of this approach. Considering the expected squared loss for each bootstrapped predictor $j = 1, \ldots, N$ we have

$$\begin{aligned} \mathrm{E}_{\pi_j(B)}\mathrm{E}_{\mathbf{x},y}((y - f_{\pi_j(B)}(\mathbf{x}))^2) &= \mathrm{E}_y(y^2) + \mathrm{E}_{\pi_j(B)}\mathrm{E}_{\mathbf{x}}(f_{\pi_j(B)}(\mathbf{x}))^2 \\ &- 2\mathrm{E}_y(y)\mathrm{E}_{\pi_j(B)}\mathrm{E}_{\mathbf{x}}(f_{\pi_j(B)}(\mathbf{x})). \end{aligned} \tag{5.10}$$

For Bagging we have an expected squared loss of

$$\begin{aligned} \mathrm{E}_{\pi_j(B)}\mathrm{E}_{\mathbf{x},y}((y - f_{\text{Bagging}}(\mathbf{x}))^2) = \mathrm{E}_y(y^2) + \mathrm{E}_{\mathbf{x}}(f_{\text{Bagging}}(\mathbf{x}))^2 \\ -2\mathrm{E}_y(y)\mathrm{E}_{\mathbf{x}}(f_{\text{Bagging}}(\mathbf{x})). \end{aligned} \tag{5.11}$$

Jensen's inequality implies that $\mathrm{E}_{\mathbf{zp}_j}\mathrm{E}_{\mathbf{x}}(f_{\mathbf{zp}_j}(\mathbf{x}))^2 \geq \mathrm{E}_{\mathbf{x}}(f_{\text{Bagging}}(\mathbf{x}))^2$ and provided that $E_{\mathbf{x}}(f_{\text{Bagging}}(\mathbf{x}))$ is equal to $E_{\mathbf{zp}_j}\mathrm{E}_{\mathbf{x}}(f_{\mathbf{zp}_j}(\mathbf{x}))$ the loss of (5.10) will be approximately greater than (5.11). In fact $E_{\mathbf{x}}(f_{\text{Bagging}}(\mathbf{x}))$ is guaranteed to be equal to $E_{\mathbf{zp}_j}\mathrm{E}_{\mathbf{x}}(f_{\mathbf{zp}_j}(\mathbf{x}))$ provided that $\mathbf{x}$ is i.i.d. This result also implies that the larger the variation/diversity amongst the N predictors the greater the disparity between ((5.10) and (5.11), therefore more is to be gained by using Bagging. As we have seen in the classification setting, solution based on gradient methods like the Perceptron, are sensitive to altering the order of the training sequence, and this leads to diversity between the individual classifiers solutions. Even though we have the added complication with

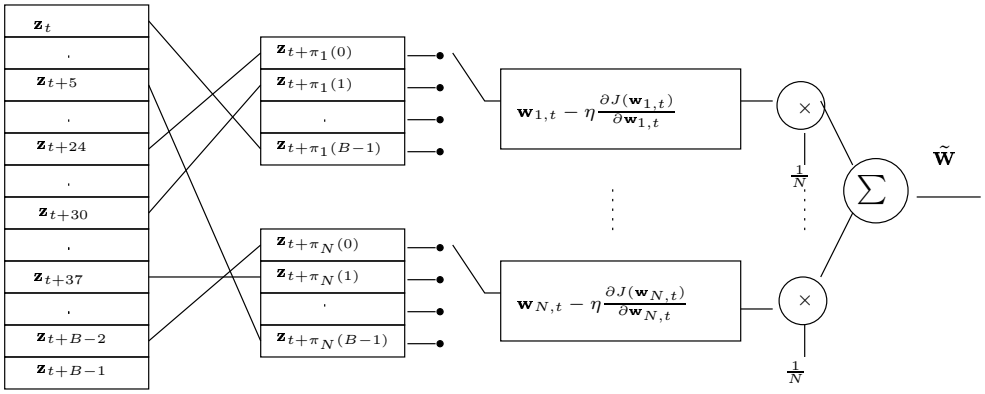

Figure 5.2: Architecture of the buffer method.

equalization that the $\mathbf{x}$'s are not i.i.d. this analysis does give some motivation for using Bagging with equalization, especially where the term $\mathrm{E}_{\mathbf{zp}_j}\mathrm{E}_{\mathbf{x}}(f_{\mathbf{zp}_j}(\mathbf{x}))^2$ dominates the loss of (5.10).

5.4.2 Classification

One perceived difficulty in using pattern classification to perform equalization is the fact that the elements of the instance vector are correlated temporally, i.e. $\mathrm{E}_{\mathbf{x}}\mathbf{x}_t \cdot \mathbf{x}_{t+j} > 0$ for $j < d$. The temporal correlation between the instance vectors makes incorrect the general assumption usually used in pattern classification that the instances are i.i.d.. This makes some of the theoretical results in pattern classification harder to apply. For example theoretical results for generalisation error bounds assume that the instances are i.i.d.. An awareness that the data set may not be linearly separable has motivated the several nonlinear methods of Pérez-Cruz and Artés-Rodríguez [2002], Sebald and Bucklew [2000], Santamaría et al. [2003]. Whilst the Perceptron methods prescribed in this section have nonlinear counterparts (see Chapter 2 and 3) we restrict our study to the linear case because in an online, or adaptive setting, has a simpler implementation.

We have seen in the classification setting that the generalisation performance can be improved by using the techniques of Chapters 2 and Chapters 3. A natural question is whether the good performance of techniques like the voted Perceptron, or online Bayes point machine (OBPM), translate to the problem of equalization? Even when there is the presence of noise and the instance vectors are correlated?

5.5 Online implementations

We present two different online approaches to create the diversity amongst linear discriminants, and therefore implement both of the aggregate methods Bagging and OBPM (see Section 5.4). The buffered approach can be used with all the loss functions discussed in Section 5.3. The subsampling approach is restricted to the classification setting only

Algorithm 8 Buffered implementation

Require: A training sample $\boldsymbol{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: For $j = 1, \ldots, N$ have $\pi_j : \{0, \ldots, B-1\} \rightarrow \{0, \ldots, B-1\}$ used to permute buffered samples (at iteration t) $\boldsymbol{z}_B := (z_t, \ldots, z_{t+B-1})$ into $\boldsymbol{zp}_j := (z_{t+\pi_j(0)}, \ldots, z_{t+\pi_j(B-1)})$

1: Initialise weights $\mathbf{w}_{j,1} := \mathbf{0}$, for all $j \in \{1, \ldots, N\}$.
2: **for** $t = 1$ to $T - B + 1$ **do**
3: update buffer $\boldsymbol{z}_B := (z_t, z_{t+1}, \ldots, z_{t+B-1})$
4: **for** $j = 1$ to N **do**
5: $\mathbf{w}_{j,t+1} := \mathbf{w}_{j,t} - \eta \sum_{i=0}^{B-1} \frac{\partial \mathrm{J}(\mathbf{w}_{j,t}, z_{t+\pi_j(i)})}{\partial \mathbf{w}_{j,t}}$
6: **end for**
7: $\tilde{\mathbf{w}}_{t+1} := \frac{1}{N} \sum_{j=1}^{N} \mathbf{w}_{j,t+1}$
8: **end for**

(see Section 5.5.2).

5.5.1 Buffered approach

Consider running N equalizers in parallel where each equalizer $j = 1 \ldots, N$ sees a sequence formed from a permutation of B examples from $\mathbf{z}$. Hence for each equalizer j we associate a sequence $\pi_j(1), \ldots, \pi_j(B)$ of integers in $\{0, \ldots, B-1\}$ such that $\forall r, s \in \{0, \ldots, B-1\}$ with $r \neq s$ we have $\pi_j(r) \neq \pi_j(s)$. The sequence of permuted examples seen by equalizer j is then $\boldsymbol{zp}_j := z_{t+\pi_j(0)}, \ldots, z_{t+\pi_j(B-1)}$; see Figure 5.2 and Algorithm 8. For each new example t in Algorithm 8: we update the buffer (line 3); for each equalizer j we update the equalizer according to $\boldsymbol{zp}_j$, their permuted version of the buffer $\mathbf{z}_B$, for example line 5 in case of LMS would be $\mathbf{w}_{j,t+1} = \mathbf{w}_{j,t} + \sum_{i=0}^{B-1} \eta \left(y_{t+\pi_j(i)} - \left(\mathbf{w}_{j,t} \cdot \mathbf{x}_{t+\pi_j(i)} \right) \right) \mathbf{x}_{t+\pi_j(i)}$.

The computational cost varies with B, where the best approximation to the Bayes point (classification) and Bagging (regression) is when $B = T$. The number of arithmetic operations required for the buffer approach is $O(NKB)$ compared to $O(KT)$ for LMS.

5.5.2 Subsampling approach

The buffer approach suffers a latency of B samples. An alternative method which avoids latency is the Online Bayes Point Machine (OBPM) (see Chapter 2). We use Algorithm 3 for our subsampling approach substituting the gradient update of (5.3) for our learning update rule (line 7). We now re-iterate the steps of Algorithm 3. Given a training example $z_t = (\mathbf{x}_t, y_t)$, we run N equalizers "in parallel" and ensure diversity of their solutions by randomly choosing to present z_t to each equalizer j only if $b_{j,t} = 1$, where $b_{j,t}$, $j = 1, \ldots, N$, are independent Bernoulli random variables with $\Pr(b_{j,t} = 1) = \tau$; see Figure 5.3. This process results in a subsample which has an

average sample size τT and requires $O(\tau NKT)$ arithmetic operations. This approach exploits the fact that in general some examples are redundant to learning the best hypothesis.

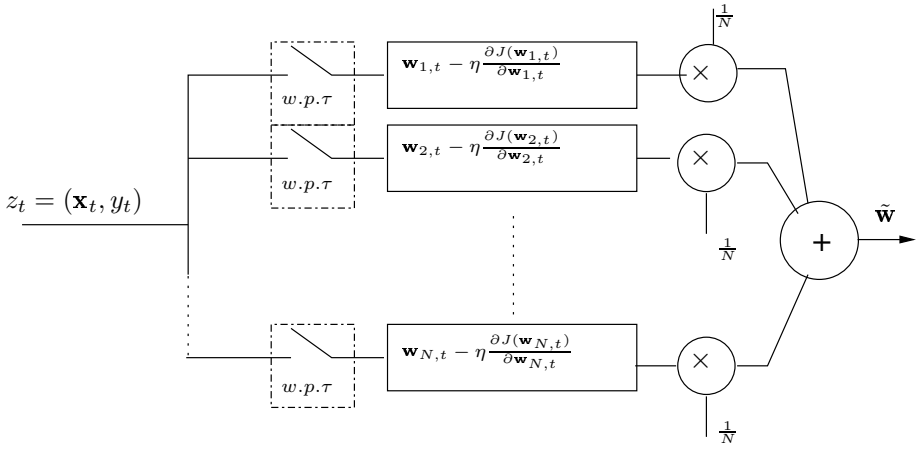

Figure 5.3: Architecture of the subsampling method.

One approximate cost function considered with OPBM is that of the Perceptron; i.e. the classifier (5.9). There is no gain by applying the subsampling of OBPM to regression methods. To see this, consider the example of LMS which is updated according to the cost function (5.4) in (5.3). If we set $\mathbf{w}_{i,0} = 0$ for all $i = 1, \ldots, N$ then equation (5.3) and (1.11) imply

$$\begin{aligned} \tilde{\mathbf{w}}_{t+1} &:= \tilde{\mathbf{w}}_t + \eta y_t \mathbf{x}_t \sum_{i=1}^{N} \frac{b_{i,t}}{N} \\ &- \eta \left(\left(\frac{1}{N} \sum_{i=1}^{N} b_{i,t} \mathbf{w}_{i,t} \right) \cdot \mathbf{x}_t \right) \mathbf{x}_t. \end{aligned} \tag{5.12}$$

From (5.12) in the limit as $N \to \infty$ the second term $\sum_{i=1}^{N} \frac{b_{i,t}}{N} \to \tau$ and the third term $\frac{1}{N} \sum_{i=1}^{N} b_{i,t} \mathbf{w}_{i,t} \to \tau \tilde{\mathbf{w}}$, therefore the subsampling for LMS simply scales η by τ.

5.6 Experiments

To demonstrate the effectiveness of the online implementations proposed in Section 5.5 we considered several simulations using two channels from [Proakis, 2001, pages 631 and 686]: channel A, $\mathbf{a}_A = [0.04, -0.05, \ 0.07, -0.21, 0.72, 0.36, 0, 0.21, 0.03, 0.07]'$ and channel B, $\mathbf{a}_B = [0.26, 0.96, 0.26]'$. The experimental results using the various aggregation methods were produced with the number of equalizers $N = 100$. The equalizer lengths used on channels A and B were 31 and 11 respectively. The test set was drawn independently to the training set and consisted of 3000 binary labelled examples. All the results in this section were produced from 200 Monte-Carlo trials. Note that the

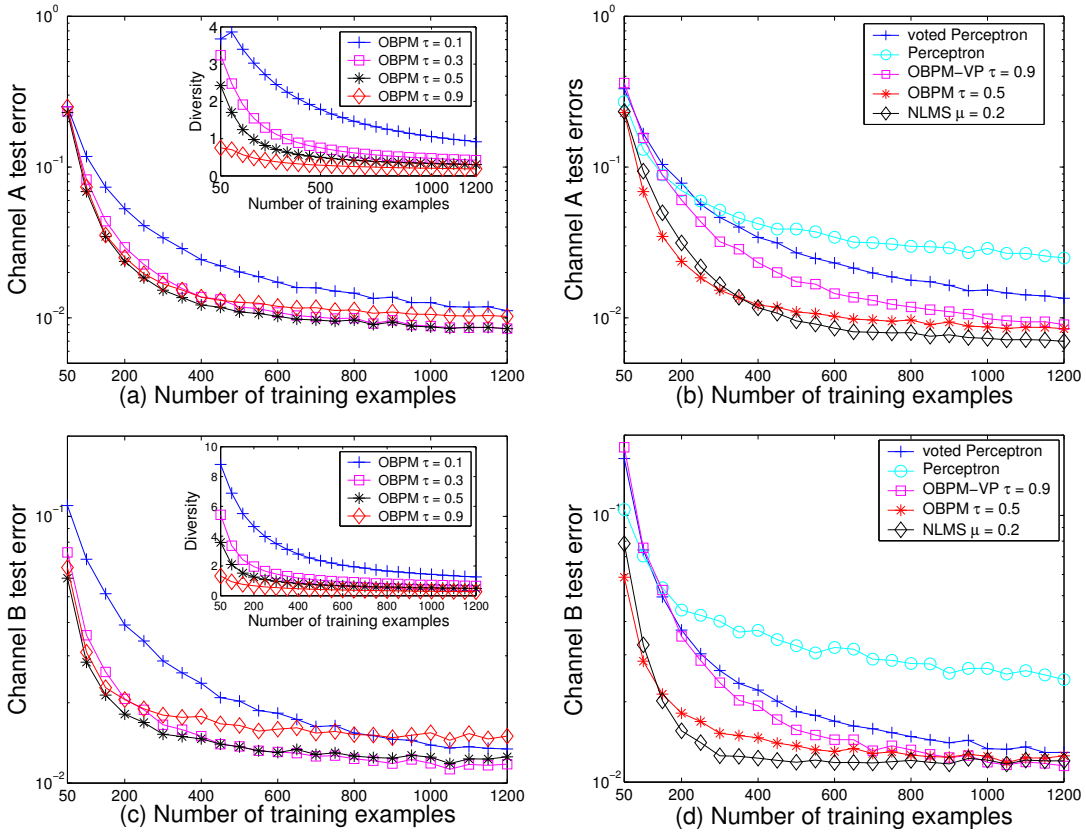

Figure 5.4: Subsampling experiment comparing the test error performance on two different channels with AWGN: (a) OBPM for Channel A with different τ settings,(b) comparison of different classification aggregate methods with NLMS on Channel A, (c) OBPM for Channel B with different τ settings and (d) comparison of different classification aggregate methods with NLMS on Channel B.

parameters for the various stochastic gradient methods where chosen in such a way as to show each algorithm in it's best light, in both convergence speed and test error performance. We define the test error as the proportion of symbols correctly decoded (for BPSK this corresponds to the raw bit rate). In all the experiments we normalised the instances $\mathbf{x}$ for both the training and test sets.

5.6.1 Subsampled experiments

The first experiment investigated the performance of the various aggregation methods in the presence of additive noise, specifically AWGN (as in discussed in Section 5.2). To verify that there is some level of consistency in the test error results for different channels we performed the same experiment for two channels $\mathbf{a}_A$ and $\mathbf{a}_B$. The *signal to noise ratio* (SNR) was set at 10 dB, which is a relatively high amount of noise. Figure 5.4 shows the test error results, both the NLMS and OBPM had similar performance. Both NLMS and

OBPM had faster convergence compared to Perceptron, voted Perceptron and OBPM-VP. Figure 5.4 (a) and (c) show the test error results of OBPM for different settings of τ. To measure the diversity of the linear equalizers $\mathbf{w}_i$ produced by Algorithm 8 we used $\frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}_i - \tilde{\mathbf{w}})^2$. From Figure 5.4 we see that the diversity diminished asymptotically as the training of the equalizer proceeded. The lowest final test error achieved by OBPM once converged was for $\tau = 0.3$. There was small differences in the final test error results once converged over the settings of τ; it appears that only the speed of convergence was affected by τ. The smaller τ the slower the convergence, the converse was true for larger values of τ. The fact that the setting of τ had little effect on the final test error performance at convergence is in contrast to the setting of the step-size for LMS and NLMS. Figure 5.5 (a) shows how sensitive the test error performance of NLMS can be to the setting of its step-size.

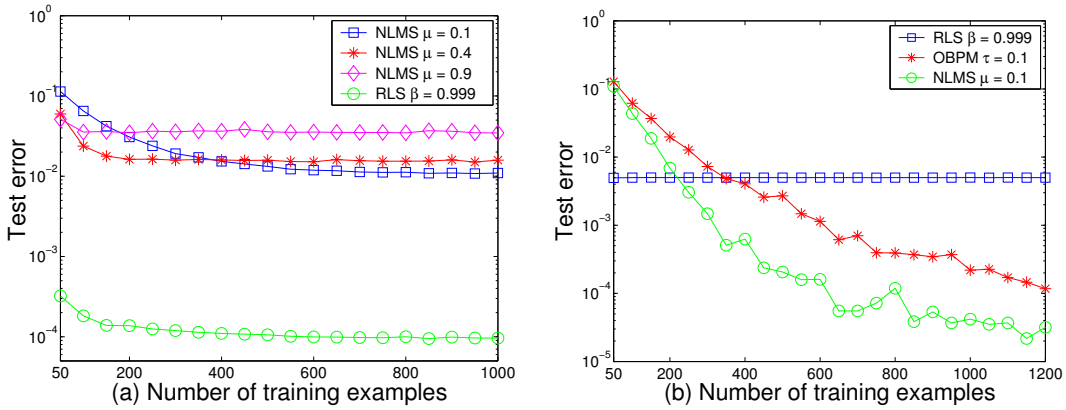

Figure 5.5: Test error performance for channel B: (a) NLMS with various step sizes versus RLS with forgetting factor $\beta = 0.999$ in the presence of AWGN of 10dB (no label flipping), and (b) NLMS, OBPM and RLS with 10 percent label flipping and 20dB AWGN.

It is well known that Recursive Least-Squares (RLS) algorithm has superior performance compared to gradient methods when the noise source is AWGN, as RLS is able to significantly reduce the effect of AWGN. The RLS procedure (as presented in Proakis [2001] page 686, noting that RLS included feedback) was used to produce the results of Figure 5.5 (a) where RLS had better convergence and test error results compared to NLMS. Whilst RLS had better test error performance compared to gradient methods like NLMS they are computationally more expensive. The second experiment hopefully shows one other example of why adaptive methods can be superior compared to RLS. In the second experiment we consider the scenario of phase noise or errors, which usually occur due to fact the phase tracking takes some time to converge, resulting in incorrect symbol decisions. We model this by flipping labels at random (uniformly) in the training set. Figure 5.5 (b) shows the test error results comparing NLMS, OBPM, and RLS when 10 percent of the labels were flipped uniformly, with SNR of 20dB. Noting that the test set did not have label flipping. From Figure 5.5 (b) clearly NLMS and OBPM

are both able to handle label flipping, whereas RLS does not.

5.6.2 Buffered experiments

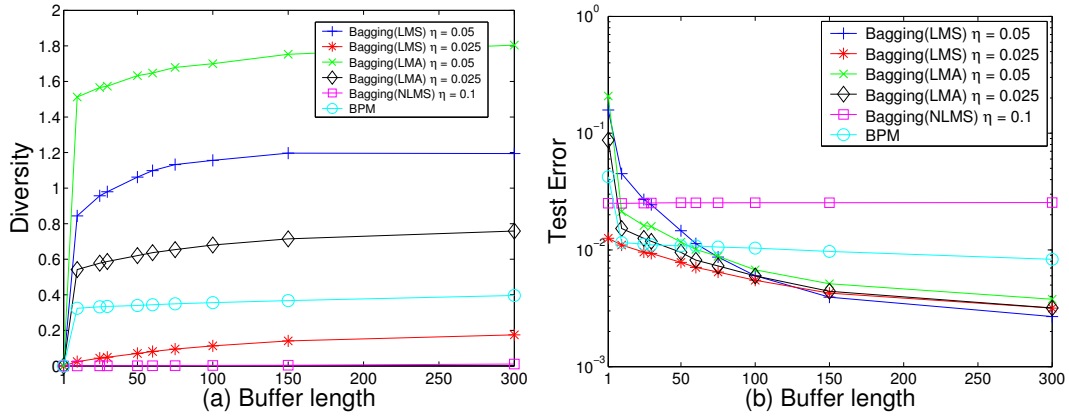

Figure 5.6: Test error (a), and diversity (b) results for Channel A with AWGN, comparing Bagging(LMS), Bagging(LMA) Bagging(NLMS), and BPM algorithms.

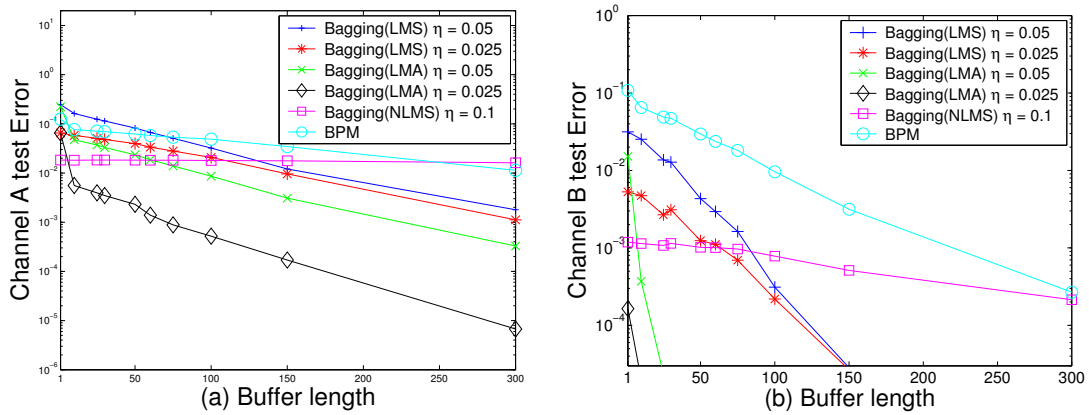

Figure 5.7: Label noise results for (a) Channel A and (b) Channel B using BPM equalizer, where only the training examples had label flipping, test examples had no label flipping.

Both experiments of the last section were repeated using the buffer implementation of Section 5.5.1. The training sequence was reduced to 300 examples. To get some idea of the "real" time required for a training sequence of 300 examples, for a typical BPSK signal (binary) with a typical throughput of 1Mbits/sec this equates to 30msecs delay. This means for every 100 examples we do not use for training we save 10msecs.

In the AWGN (additive noise) experiment the implementations investigated were: Bagging of Section 5.4.1 with regression with (5.4) and (5.6), also the BPM classification method based on the Perceptron's cost function (5.9). Figure 5.6 demonstrates the effectiveness of Bagging combined with LMA (5.6) for an SNR of 10dB. We see that

Bagging has significant test error improvement compared to the standard regression gradient methods of LMS, NLMS and LMA. Note that the test error results of Figure 5.6 for the standard gradient methods of LMS, NLMS, LMA and the Perceptron are given by a buffer of length 1. We see in Figure 5.6 (a) and (b) that the methods and buffer sizes with the largest final diversity had corresponding lower test errors. Comparing the best buffer method test error results of Figure 5.6 (LMS with $\eta = 0.05$, buffer length=300) with the best subsampling technique of the last section (see Figure 5.4). There was a reduction in the test error after 300 examples of 8.2dB (dB used here is equivalent to $10^{-8.2/10}$ times the subsampled test error) when using the buffered method compared to the best subsampling method. The final test error for Channel A of Bagging(LMS) after 300 examples was 2×10^{-3}, this was 5dB (dB used here is equivalent to $10^{-5/10}$ times the subsampled test error) better than the NLMS result of the last section after 1200 examples. We see that Bagging (LMS) achieved a lower test error with the same number of training examples as used by NLMS. This implies that Bagging(LMS) requires shorter training sets than the NLMS algorithm.

The label flipping experiments using the buffer method are given in Figure 5.7. Again, like the subsampling experiments the absolute loss function was the better method in terms of test errors. We notice that NLMS did not appear to benefit from the buffer technique.

5.7 Summary

Two alternative methods of aggregating equalizers were explored in connection with adaptive equalization: one using subsampling technique, the other was a buffering technique. The buffered method was applied to both regression and classification algorithms. The subsampling method was shown to be only effective in the classification setting. Comparison with traditional stochastic gradient methods of NLMS with AWGN and label flipping (decision errors) showed the buffered approach (Bagging) had the superior test error performance. The subsampling OBPM classifier did not outperform the NLMS algorithm. Whilst the subsampling approach had no improvement over NLMS it still warrants consideration given it does not have the instability issues of step-size choice, like in NLMS based equalizers. The use of aggregation methods suggested here with linear equalization allows for shorter training sequences to be used. Shorter training sequences are desirable as they increase channel throughput. Aggregation applied to equalization has shown robustness to label flipping (decision errors) which should provide for better equalizers in the presence of phase errors and noise.

Tracking the Best Equalizer

6.1 Introduction

As mentioned in the last chapter, the goal of the equalizer is to reduce the distortion (inter-symbol interference, ISI) caused by the channel to the transmitted signal in a communications system. The faster we reduce the distortion caused by the channel, the more expedient the recovery of the transmitted signal is at the receiver; leading to shorter training sequences, and more efficient use of bandwidth. Whilst one way to achieve better equalization is by improved generalisation, (as shown in the last chapter,) the other is selecting the most appropriate equalization algorithm and its parameter settings. A number of examples of where the algorithm and parameter selection are important in determining the success of the equalizer: choosing the order of the equalizer filter to match the channel model, when either the channel or noise are not known *a priori*; choosing the appropriate equalization algorithm with the properties to suit whether the channel is sparse or not ; choosing the adaptive equalizer step-size, trading-off between convergence speed and the optimal mean square error (MSE).

One possible approach to parameter and algorithm selection would be to try a pool of different equalizers algorithms and parameter settings; taking the one with the lowest loss at the end of training. However, in many cases no individual equalizer and its associated parameter setting are the best over its entire training. This is even true when the channel is stationary and is especially true when it is non-stationary. For these reasons a mixture of equalizers could be a better approach, where the best mix can be tracked over time. Recently, there has been research in machine learning devoted to such mixture learning algorithms, the so called *mixture of experts* algorithms (Herbster and Warmuth [1998], Vovk [1998], Littlestone and Warmuth [1994], Kivinen and Warmuth [1997, 1999], Gramacy et al. [2003], Harrington [2003], Herbster and Warmuth [1998, 2001]), as introduced in Section 1.9 of Chapter 1. In this chapter the mixture of experts algorithms are used to mix equalization algorithms and their parameters by giving higher weight to the predictions of the best performing equalizers. Specifically, we are interested in the use of the mixture of experts algorithms of Herbster and Warmuth [1998] which cater for the situation where the correct equalizer and its parameters can change over time, or the channel model and/or noise is non-stationary.

In the *fixed-share* algorithm of Herbster and Warmuth [1998] the assumption is made that the rate of switching between experts is constant. Several potential limitations arise from this assumption, such as how to determine the ideal switching rate online? That any variation in the ideal switching rate over time will result in an increase in the prediction error. This motivates a new mixture of experts algorithm (called the fixed-share hierarchy algorithm), a two tier hierarchy of the mixture of experts algorithm of Herbster and Warmuth [1998] with the ability to track the best expert when the share between experts is either varying over time or is constant. We compare the ability of the hierarchy of mixture of experts with the original algorithm of Herbster and Warmuth [1998], and with the method of Singer and Feder [1999] the *universal equalization algorithm* (UEA). We draw attention to the many potential uses of our mixture of experts algorithm to adaptive channel equalization through several usage example scenarios.

6.2 More adaptive equalization methods

We use here the same problem setting and system model as in Section 5.2, we also consider (Section 6.6.5) the more difficult problem of blind equalization (where the labels are not known). In this section we present a number of adaptive equalizers additional to those discussed in Section 5.3.

Given the residual $\xi_t = y_t - \hat{y}_t$, consider gradient method updates of the form

$$f(w_{t+1,i}) := f(w_{t,i}) - \eta_t \frac{\partial L(\xi_t)}{\partial w_{t,i}}, \tag{6.1}$$

where $\mathbf{w}_t$ is the vector used to produce $\hat{y}_t$ as in (5.2), f is a linear or possibly nonlinear function used to modify the weights, $\eta_t (> 0)$ is the learning rate, and L is a loss function. In general (6.1) can be used with a number of different gradient based equalizers; but we only consider in this chapter gradient based equalizers which have a squared loss function $L(\xi_t) = \frac{1}{2}\xi_t^2$.

We will now consider a number of different examples of $f(w_{t,i})$. The simplest case is the Stochastic Gradient Descent (SGD) algorithm where $f(w_{t,i}) = w_{t,i}$ and

$$w_{t+1,i} := w_{t,i} - \eta_t \frac{\partial L(\xi_t)}{\partial w_{t,i}}. \tag{6.2}$$

When the target weight vector is known to be sparse it is well known that the Exponentiated Gradient (EG) algorithm (Kivinen and Warmuth [1997]) has better convergence properties compared to the SGD algorithm. The existence of communications channels which have corresponding sparse solutions (target weights) motivates the inclusion of the EG algorithm in this discussion on adaptive equalizers. The EG update is derived

from the general update by substituting $f(w_{t,i}) = \ln(w_{t,i})$ in (6.1) giving,

$$w_{t+1,i} := w_{t,i} \exp\left(-\eta_t \frac{\partial L(\xi_t)}{\partial w_{t,i}}\right). \tag{6.3}$$

Note that for $f(w_{t,i}) = \ln(w_{t,i})$ we must guarantee that $w_{t,i}$ is positive; warranting the consideration of alternative approaches to EG.

Both the SGD and EG algorithms can be generalised by replacing the gradient with the natural gradient (NG,Amari [1998]). The NG algorithm utilises a general Riemannian metric in the weight space. For brevity we will not go into more detail concerning the NG algorithm. Rather we are interested in the simpler and easier to implement first order approximation, the Approximate Natural Gradient algorithm (ANG) (Martin et al. [2002]). Mahony and Williamson introduce a different interpretation of the NG algorithm, that the true weight vector $\mathbf{w}^*$ is drawn at random from some prior distribution $w_{t,i} \neq z_{t,i}$ hence replacing $w_{t,i}$ with $z_{t,i}$ in with density $\phi(w_{t,i})$ (note that $\phi(w)$ need not satisfy $\int \phi(w)dw = 1$). The ANG algorithm is given by,

$$w_{t+1,i} := w_{t,i} - \eta_t \frac{\partial L(\xi_t)}{\partial w_{t,i}} \frac{1}{\phi(w_{t,i})^2}. \tag{6.4}$$

In order to develop intuition about (6.4) consider the case where the prior is $\phi(w_{t,i}) = 1/w_{t,i}$ and $w_{t,i} > 0$. In this case if the $w_{t,i}$ is small in value then the prior will be high, and the opposite is true when $w_{t,i}$ is large. Hence in the ANG algorithm there is a warping of the weight space to suit the fact that the target weight is sparse, since the prior is high when $w_{t,i}$ is small.

A similar paradigm to the ANG and NG algorithms, are the *proportionate adaptation* algorithms (see Haykin and Widrow [2003]). The proportionate adaptation algorithms, like ANG algorithm, perform well when the channel is sparse by applying learning rates to different weights according to the sparseness of the channel. The gradient methods of ANG and SGD have the advantage over batch methods of being able to adapt when the problem is non-stationary, as they are adaptive (online) methods. In the case where being able to adapt is of no advantage we consider optimal solutions to least-squares criterion, i.e. recursive least squares (RLS) algorithm. The reason that the least squares criterion, $\min_{\mathbf{w}} \frac{1}{T}\sum_{t=1}^{T}(y_t - \mathbf{w}'\mathbf{x}_t)^2$, is more suited to the batch setting than the online one is because it is only optimal if target solution is stationary. In this chapter we consider the computational efficient lattice version of RLS, the least squares adaptive lattice equalizer (LSALE) of Satorius and Pack [1981].

6.3 Preliminaries: mixture of experts

We will consider mixtures of experts, in particular the online fixed-share algorithm from Herbster and Warmuth [1998]. The simplest algorithm is referred to as the *static expert* algorithm (Haussler et al. [1998], Herbster and Warmuth [1998]). In the static expert

case each expert $i = 1, \ldots M$ makes a prediction $\hat{y}_t(i)$ at trial t (where an expert could be a neural network, decision tree, or as in our case, an equalizer) and is assigned a weight $v_t(i)$. The overall prediction of the mixture of experts is

$$\hat{y}_t^m := \sum_{i=1}^{M} v_t^m(i)\hat{y}_t(i), \tag{6.5}$$

where $v_t^m(i) := v_t(i)/\sum_{c=1}^{M} v_t(c)$ is the normalised weight applied to expert i. The key to performing close to the best expert's prediction is in the loss-based weight update. The loss-based weight update for the static-expert is

$$v_{t+1}(i) := v_t(i)\exp(-\nu L(\xi_t(i))), \tag{6.6}$$

where $\nu > 0$ is the learning rate, and $\xi_t(i) = y_t - \hat{y}_t(i)$ is the residual of expert i at trial t. We know from the loss-based weight update that if an expert has large losses then the weight $v_t(i)$ associated with that expert will become exponentially small.

We define the total loss for the weighted average $\hat{y}_t^m$ for the sequence of examples $\mathbf{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$ as $L_{WA}(\mathbf{z}) := \sum_{t=1}^{T} L(\xi_t^m)$ where $\xi_t^m = y_t - \hat{y}_t^m$. Similarly the total loss for expert i over the same sequence $\mathbf{z}$ is $L_{\mathbf{z},i} := \sum_{t=1}^{T} L(\xi_t(i))$ where $\xi_t(i) = y_t - \hat{y}_t(i)$.

The static-expert algorithm[1] for the weighted average has been shown in Kivinen and Warmuth [1999] to have for a large class of loss functions L and range of learning rates ν, the loss bound

$$L_{WA}(\mathbf{z}) \leq \min_{1 \leq i \leq M}(L_{\mathbf{z},i}) + \frac{1}{\nu}\ln M. \tag{6.7}$$

This bound tells us that the total loss of the weighted average $L_{WA}(\mathbf{z})$ will be no more than the total loss of the best expert plus the term $\frac{1}{\nu}\ln M$ (a term that does not increase with T, only in the number of experts M). It is worth exploring the range of ν for (6.7) holds, when using the loss function for LMS, so as to better understand the expected performance. For $|\xi_t(i)| \leq \xi_{\max}$ and using the squared loss function $L(\xi_t(i)) := \frac{1}{2}\xi_t(i)^2$ the bound (6.7) holds for all ν such that $\frac{1}{\nu} \geq \xi_{\max}^2$.

The assumption of the static expert algorithm is that one expert will be the best over all T examples. In fact we will give a number of equalization scenarios in Section 6.6 where one expert is not the best over T examples. The *fixed-share* algorithm (see Algorithm 9) is able to track the best expert, or mixture of experts, when the target concept (optimal solution) or environment (environment refers to both the noise and channel) is changing. The fixed-share algorithm prevents the weight of any expert getting too low as a consequence of the loss-based weight update (6.6) by distributing

[1] We must note that the bound of (6.7) was first done by Vovk [1998] with a more complicated prediction than in (6.5) though achieved smaller loss bounds.

Algorithm 9 fixed-share algorithm (Herbster and Warmuth [1998])

Require: A example sequence $\boldsymbol{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: Each expert $i = 1, \ldots, M$ having update rule $\mathcal{L}_i$ and associated parameters $\mathbf{w}_i$.

Require: Loss function $L(\cdot)$ where $L : \mathbb{R} \to \mathbb{R}$.

1: Initialise: $v_1(i) := 1/M$ for $i = 1, \ldots, M$
2: Initialise: $w_{1,i}(j) := 0$ for $j = 1, \ldots, M$ and $i = 1, \ldots, d$
3: **for** $t = 1$ to T **do**
4: **for** $i = 1$ to M **do**
5: $(\mathbf{w}_{t+1,i}, \xi_t(i), \hat{y}_t(i)) := \mathcal{L}_i(\mathbf{w}_{t,i}, \mathbf{x}_t, y_t)$
6: **end for**
7: **for** $i = 1$ to M **do**
8: $v_t^m(i) := v_t(i) / \sum_{c=1}^{M} v_t(c)$
9: $v_t'(i) := v_t(i) \exp(-\nu L(\xi_t(i)))$
10: **end for**
11: $\xi_t^m := y_t - \hat{y}_t^m$ where $\hat{y}_t^m := \sum_{c=1}^{M} v_t^m(c) \hat{y}_t(c)$
12: **for** $i = 1$ to M **do**
13: $v_{t+1}(i) := (1 - \alpha) v_t'(i) + \frac{\alpha}{(M-1)} (\sum_{c=1}^{M} v_t'(c) - v_t'(i))$
14: **end for**
15: return $\hat{y}_t^m$
16: **end for**

a small fraction α of the total weight uniformly over all the experts. Thus, if an expert that previously performed poorly starts to perform well its weight can recover faster. Line 13 of the fixed-share algorithm

$$v_{t+1}(i) := (1 - \alpha) v_t(i) \exp(-\nu L(\xi_t(i))) + \frac{\alpha}{(M-1)} (V_t - v_t(i) \exp(-\nu L(\xi_t(i)))), \quad (6.8)$$

is the weight update for some $\alpha : 0 \leq \alpha < 1$, where $V_t := \sum_{i=1}^{M} v_t(i) \exp(-\nu L(\xi_t(i))$. Note that the fixed-share update is constructed such that $\sum_{i=1}^{M} v_{t+1}(i) := \sum_{i=1}^{M} v_t(i) \exp(-\nu L(\xi_t))$.

One of the methods presented[2] by Bousquet and Warmuth [2002] is essentially equivalent to the fixed-share algorithm, and gives us some added intuition into how it works. In the method of Bousquet and Warmuth [2002] the fixed-share algorithm replaces the loss-based weight update of (6.6) with $v_t^m(i) := v_t(i) \exp(-\nu L(\xi_t(i))) / \sum_{i=1}^{M} v_t(i) \exp(-\nu L(\xi_t(i)))$; and (6.8) with $v_{t+1}(i) := (1 - \alpha) v_t^m(i) + \alpha v_1^m(i)$, where $v_1^m(i)$ is set equal to $1/M$. The term $v_1^m(i) = 1/M$ captures the intuition that into the future there is an equal chance of any of the M experts

[2]We must clarify that Bousquet and Warmuth [2002] developed methods which track a small set of experts from within a larger pool of experts.

(equalizers) becoming the best expert.

In an effort to analysis Algorithm 9 we consider, for some fixed k, all possible partitionings of the trial sequence $1, 2, \ldots, T$ into $k+1$ segments $(t_0, \ldots, t_1 - 1)$ $, (t_1, \ldots, t_2 - 1), \ldots, (t_k, \ldots, t_{k+1} - 1)$ with corresponding trial examples $\mathbf{z}_1 = ((\mathbf{x}_{t_0}, y_{t_0}), \ldots, (\mathbf{x}_{t_1-1}, y_{t_1-1}))$, $\mathbf{z}_2 = ((\mathbf{x}_{t_1}, y_{t_1}), \ldots, (\mathbf{x}_{t_2-1}, y_{t_2-1})), \ldots, \mathbf{z}_{k+1} = ((\mathbf{x}_{t_k}, y_{t_k}), \ldots, (\mathbf{x}_{t_{k+1}-1}, y_{t_{k+1}-1}))$, where $t_0 = 1$ and $t_{k+1} = T$. The accumulated loss over segment j for expert i is denoted by $L_{\mathbf{z}_j,i} := \sum_{s=t_{j-1}}^{t_j-1} L(\xi_s(i))$ where $\xi_s(i) = y_s - \hat{y}_s(i)$.

Theorem 6.3.1 (Herbster and Warmuth [1998]). *For any partitioning of the example sequence* $\mathbf{z}$ *of length* T *into* k *segments* $\mathbf{z}_j$ *and with the assignments of experts* i_j *to each segment, the fixed-share algorithm satisfies*

$$L_{WA}(\mathbf{z}) \leq \sum_{j=1}^{k+1} L_{\mathbf{z}_j,i_j} + \frac{\ln M}{\nu} + \frac{(T-k-1)}{\nu} \ln \frac{1}{1-\alpha} + k/\nu \left[\ln \frac{1}{\alpha} + \ln(M-1) \right]. \tag{6.9}$$

The assumptions about L *and* ν *are as for (6.7).*

The first thing to notice is the fixed-share loss bound of (6.9) grows with the number of experts M, but like the static case the growth is logarithmic. We also see that the bound is dependent on our choice in the number of segments k and on the number of examples T. By taking the derivative of (6.9) we find that the choice of the share, α, that minimises the error term (third and fourth terms of (6.9)) is $\alpha = k/(T-1)$. The rate of switching between each expert determines α, sometimes referred to as the *switching rate*. The optimal choice of the share/switching rate is dependent on the problem we are looking at, it may require fast switching or slow switching (or variable switching, see next section). The choice of α, therefore k, is made to minimise the accumulated loss for the assignment of experts, $\sum_{j=1}^{k+1} L_{\mathbf{z}_j,i_j}$. We note that the size of α dictates the ability of the fixed-share algorithm to switch between experts effectively: making α larger allows for faster switching but once it has changed to the best expert the losses will larger due to the allocating of $v_t^m(i) = \alpha/M$ to the other experts predictions which are not the best performing. The opposite is true for setting α small since in this case the ability to react to a change will be reduced but when it has changed to the correct expert the losses will be smaller.

As shown in Figure 6.1, the error term $H(\alpha, \nu, M)$ (third and fourth terms of (6.9)) increases with an increase in the choice of the switching rate α. The smaller the switching rate the tighter the bound of (6.9). From Figure 6.1 we see if the switching rate α is less than 0.01 then the effect of the error term $H(\alpha, \nu, M)$ is comparable [3] to an average cumulative loss $(\frac{1}{T}\sum_{j=1}^{k+1} L_{\mathbf{z}_j,i_j})$ of 0.01 $((H(\alpha, \nu, M) = 10)/T)$ in the case where

[3] Note that the choice of parameters in Figure 6.1 match those used in the experiments section.

$T = 1000$. It appears that the bound of (6.9) is more appropriate for when T and α are small.

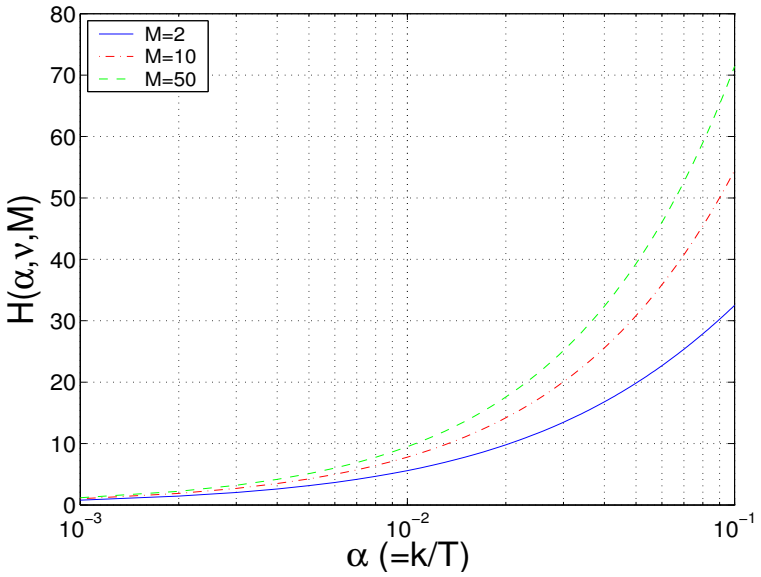

Figure 6.1: Plot of the error term of bound (6.9), $H(\alpha, \nu, M) = \frac{(T-k-1)}{\nu} \ln \frac{1}{1-\alpha} + \frac{k}{\nu}\left[\ln \frac{1}{\alpha} + \ln(M-1)\right]$ versus the switching rate $\alpha = k/T$. The number of experts was varied from $M = 2, 10, 50$, with the learning rate set at $\nu = 10$ and number of examples $T = 1000$.

6.4 The fixed-share hierarchy

In some online learning problems the optimal share may not be constant over all T examples. To assume the optimal share is constant it must first be assumed that the best equalizer (expert) mixture switching rate is constant. This assumption of a constant switching seems limiting! Even if the optimal choice of the share (α) is constant there are still the issue of determining it in an online setting.

From the discussion of the previous section we know that a larger share for the fixed-share algorithm has the trade-off of faster speed of transition between experts but with cost of distributing more weight amongst all the experts. This trade-off is one advantage of being able to vary the value of the share (α). Having a higher choice of α when the ideal mixture of experts changes enables a faster transition to approximately the correct mixture. During the periods where the ideal mixture is constant having a lower share (α) leads to less weight allocated uniformly to all the experts, and more to the ideal mixture of experts. In equalization it seems clear that if either the channel or noise are time variant then being able to tune the share would enable better tracking of the correct equalizer model. One reason that tuning the share is better is because we do not know a priori which share is most appropriate at a given point in time. Even when the channel is stationary the ideal mixture of equalizers will more than likely

Algorithm 10 fixed-share hierarchy

Require: A example sequence $\boldsymbol{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.

Require: Each expert $i = 1, \ldots, M$ having update rule $\mathcal{L}_i$ and associated parameters $\mathbf{w}_i$.

Require: Loss function $L(\cdot)$ where $L : \mathbb{R} \to \mathbb{R}$.

Require: Fixed-share algorithms $j = 1, \ldots, K$ with expert weights $i = 1, \ldots, M$ at iteration t of $v^m_{t,j}(i)$ and fraction $0 \leq \alpha_j < 1$ of total weight.

Initialise: $v_{1,j}(i) := 1/M$ for $i = 1, \ldots, M$ and $j = 1, \ldots, K$
Initialise: $h_{1,j} := 1/K$ for $j = 1, \ldots, K$
Initialise: $w_{1,i}(j) := 0$ for $i = 1, \ldots, M$ and $j = 1, \ldots, d$
for $t = 1$ to T **do**
  for $i = 1$ to M **do**
    $(\mathbf{w}_{t+1,i}, \xi_t(i), \hat{y}_t(i)) := \mathcal{L}_i(\mathbf{w}_{t,i}, \mathbf{x}_t, y_t)$
  end for
  for $j = 1$ to K **do**
    for $i = 1$ to M **do**
      $v^m_{t,j}(i) := v_{t,j}(i) / \sum_{c=1}^{M} v_{t,j}(c)$
      $v'_{t,j}(i) := v_{t,j}(i) \exp(-\nu L(\xi_t(i)))$
    end for
    $\xi^m_{t,j} := y_t - \hat{y}^m_{t,j}$ where $\hat{y}^m_{t,j} := \sum_{c=1}^{M} v^m_{t,j}(c) \hat{y}_t(c)$
    for $i = 1$ to M **do**
      $v_{t+1,j}(i) := (1 - \alpha_j) v'_{t,j}(i) + \frac{\alpha_j}{(M-1)} (\sum_{c=1}^{M} v'_{t,j}(c) - v'_{t,j}(i))$
    end for
  end for
  for $j = 1$ to K **do**
    $h^m_{t,j} := h_{t,j} / \sum_{c=1}^{K} h_{t,c}$
    $h'_{t,j} := h_{t,j} \exp(-\nu L(\xi^m_{t,j}))$
  end for
  for $j = 1$ to K **do**
    $h_{t+1,j} := (1 - \alpha_h) h'_{t,j} + \frac{\alpha_h}{(K-1)} (\sum_{c=1}^{K} h'_{t,c} - h'_{t,j})$
  end for
  return $\hat{y}^h_t := \sum_{j=1}^{K} h^m_{t,j} \hat{y}^m_{t,j}$
end for

change when considering equalizers with different parameter settings. It is unlikely that the switching between these equalizers will be constant for the entire training sequence. In fact the switching may stop at some point in time. In the equalization scenarios presented in Section 6.6 the share which minimises the cumulated loss of the fixed-share algorithm in practise is not constant.

The possibility of a varying share motivates the *fixed-share hierarchy* of Algorithm 10. Algorithm 10 consists of K fixed-share algorithms, each with a different

share α_j where $j = 1, \ldots, K$. A two level hierarchy is formed where at the top level an additional fixed-share algorithm tracks the fixed-share algorithm with the best performing share. The fixed-share hierarchy of Algorithm 10 performs the following steps at iteration t: each expert (equalizer) $i = 1, \ldots, M$ is presented with an instance $\mathbf{x}_t$, updates its parameters (updating of parameters may not be necessary as the expert models can be fixed) and makes a prediction $\hat{y}_t(i)$ (line 6); for each fixed-share algorithm j with share α_j, the weights and the loss-based weight update are determined for each expert (lines 10 and 11, respectively); the residual and prediction are then calculated for each fixed-share algorithm j (line 13); weights for all M experts are updated according to the fixed-share update of (6.8) (line 15); at the top level the weights $h^m_{t,j}$ associated with shares $j = 1, \ldots, K$ are updated according to the fixed-share algorithm (lines 18 to 24); the hierarchy's prediction $\hat{y}^h_t$ is then returned. There is a single line alternative to the steps of lines 13 and 25 of Algorithm 10 in determining the prediction $\hat{y}^h_t$. Lines 13 and 25 of Algorithm 10 can be reduced to

$$\hat{y}^h_t := \sum_{c=1}^{M} v^h_t(c)\hat{y}_t(c), \tag{6.10}$$

where the expert weight of expert $c \in \{1, \ldots, M\}$ is

$$v^h_t(c) := \sum_{j=1}^{K} h^m_{t,j} v^m_{t,j}(c), \tag{6.11}$$

and $\hat{y}_t(c)$ is the prediction of expert (equalizer) c.

The static case of Algorithm 10, that is a single switching rate is α_j is the best for all T can be found by setting $\alpha_h = 0$. Learning a single optimal share ($\alpha_h = 0$) was investigated in Monteleoni and Jaakkola [2004] but, as we have discussed, a single share may not be optimal over the entire trial sequence (this is demonstrated in the experiments of Section 6.7).

The additional computations required to perform each fixed-share algorithm, the loss-based weight update (line 10 and 11 of Algorithm 10), and fixed-share update (line 15 of Algorithm 10) is $O(M)$ per iteration. The number of arithmetic operations for the proposed equalizer system of Algorithm 10 is $O(KMd)$ per iteration. This number of operations was determined using the adaptive equalizers of LMS and ANG, and counting the exponentiations as just one. The low computational cost of most of the methods for adaptive equalization makes the additional computation of Algorithm 10 is not too prohibitive.

6.5 Fixed-share hierarchy bound

In this section a Corollary to Theorem 6.3.1 is presented that gives the accumulated loss bound for the fixed-share hierarchy of Algorithm 10. Before presenting Corollary 6.5.1

we re-iterate the setting of Algorithm 10. The fixed-share hierarchy algorithm consists of M experts and K fixed-share algorithms. Each fixed-share algorithm $j = 1, \ldots, K$ has a fixed k_j (or $\alpha_j = k_j/T$) with the trial sequence having an associated segmentation of $(t_0^j, \ldots, t_1^j - 1)$, $(t_1^j, \ldots, t_2^j - 1), \ldots, (t_{k_j}^j, \ldots, t_{k_j+1}^j - 1)$. At the top level of the fixed-share hierarchy, the trial sequence is again partitioned into segments, in this case $k_h + 1$ segments of $(t_0^h, \ldots, t_1^h - 1)$, $(t_1^h, \ldots, t_2^h - 1), \ldots, (t_{k_h}^h, \ldots, t_{k_h+1}^h - 1)$. The corresponding trial sequence for each segment is $\mathbf{z}_s^h = ((\mathbf{x}_{t_{s-1}^h}, y_{t_{s-1}^h}), \ldots, (\mathbf{x}_{t_s^h - 1}, y_{t_s^h - 1}))$. The accumulated loss over each segment $\mathbf{z}_s$ for the fixed-share algorithm with share α_{j_s} is denoted by $L_{\mathbf{z}_s^h, \alpha_{j_s}}$.

Corollary 6.5.1. *For any partitioning of the trial sequence $\boldsymbol{z}$ into $k_h + 1$ segments $\boldsymbol{z}_s^h$, for $s = 1, \ldots, k_h + 1$, and assignments of shares α_{j_s} to each segment, the fixed-share hierarchy algorithm satisfies*

$$L_{WA}(\boldsymbol{z}) \leq \sum_{s=1}^{k_h+1} L_{\boldsymbol{z}_s^h, \alpha_{j_s}} + \frac{\ln K}{\nu} + \frac{(T - k_h - 1)}{\nu} \ln \frac{1}{1 - \alpha_h} + \frac{k_h}{\nu} \left[\ln \frac{1}{\alpha_h} + \ln(K - 1) \right], \tag{6.12}$$

where, with the hierarchy's residual being $\xi_t^h = y_t - \hat{y}_t^h$ at time t we have $L_{WA}(\boldsymbol{z}) = \sum_{t=1}^{T} L(\xi_t^h)$. The assumptions about L and ν are as for (6.7).

Corollary 6.5.1 can be shown to be true by simply applying Theorem 6.3.1 to a mixture of fixed-share algorithms.

6.6 Example scenarios

A number of different scenarios of using Algorithm 10 for the problem of adaptive channel equalization are presented in this section. This by no means covers all the possible uses of this algorithm; the intention being only to illustrate the potential uses of a hierarchy of fixed-share algorithms.

Determining the most appropriate equalization algorithm and its associated parameters is difficult, especially when either the channel characteristic or the noise model are not known beforehand. Even during the convergence of a particular equalization algorithm the most appropriate parameters choice can change over time. One solution is to consider a mixture of parameters, or algorithms, allowing for a more adaptive equalizer. Listed below are five different situations where a hierarchy of fixed-share algorithms can be used in tracking the equalization algorithm and its associated parameters with the lowest accumulated loss.

6.6.1 Model tracking (multipath)

In wireless communications the transmitted signal may simultaneously travel multiple paths in its journey to the receiver. This is referred to as multipath propagation. The

effect of multipath propagation is to create fluctuations in the received signals amplitude and phase. When the time-delay of the multipath components is larger then the symbol timing, they create ISI similar to that produced by a electronic filter in the receiver of a communications system. For this reason this type of multipath is sometimes referred to as *channel-induced* ISI. Consider the situation where we can model the channel fluctuations as several different channels. We propose using the FSH algorithm to track the relative changes in the channel model due to changes in the multipath, given the channel model and its matching equalizer. Note that it is the intention to use this scenario to explore the capabilities of the FSH algorithm in tracking a switching model. This is not intended as a completely realistic communications application. Actually it is an approach more relevant to time series modelling for example a time series with a time dependence could be modelled with a mixture of *autoregressive* (AR) models. One reason why the FSH algorithm is not completely realistic is that it will be limited to only tracking the multipath during the training phase of the equalizer.

6.6.2 Determining model order

One problem in channel equalization is choosing the most appropriate model order of the equalizer to match the channel model. In a broader sense, determining the model order of a filter is a common problem in most adaptive filtering settings. Several methods have been proposed to determine the minimum order required of a linear predictor: *Akaike's information criterion* (AIC) of Akaike [1974], and its Bayesian counterpart BIC. Unfortunately, all these methods may give multiple minima; they assume that the data is Gaussian distributed and are in general determined off-line.

Singer and Feder [1999] presented an online learning algorithm for equalization the Universal Equalization Algorithm (UEA) which combines the Least Squares Lattice (LSALE) algorithm of Satorius and Pack [1981] with essentially the static expert algorithm of (6.6). The UEA forms a prediction of the best LSALE order by a weighted average of all M order predictions of LSALE (as given in (6.6)). Hence, UEA has essentially the same loss bound performance as the static expert of (6.7).

One potential limitation of UEA is its use of the static expert update to find the best order. The best model order of the LSALE more than likely changes during the convergence. The static expert algorithm has also the tendency early in the convergence to make the weight assigned to model orders other than the best expert too small. This is where the fixed-share algorithm has the advantage over the static case as it stops any weight getting too small, hence has the ability to track sudden changes in the best model order. Further robustness can be added by using the hierarchy of fixed-share algorithms, since this enables the tracking of the best share (α). Section 6.7.2 demonstrates the effectiveness of Algorithm 10 in tracking the LSALE model order with the best MSE.

6.6.3 Tracking LMS step-sizes

A well understood property of gradient methods in the presence of noise is their sensitivity to the choice of step size (learning rate) η. For constant η the final MSE of such gradient methods is given by $J_{min} + J_\eta$. The first component is the *minimum mean squared error* (MMSE), and the second component is the measurement noise (excess MSE) which is dependent on the value of η. In the case of LMS the measurement noise is given by $J_\eta \sim \eta d J_{min}$, where d is the equalizer's filter length. In determining the best η there is a trade-off between rate of convergence and the final MSE. The larger we make η the faster the initial convergence; though the final MSE may be higher. This motivates the second application of Algorithm 10 using a mixture of M LMS algorithms with different step sizes. Note the same idea could be quite easily used with the normalised LMS (NLMS).

A large body of research (Proakis [2001], Mathews and Xie [1993], Benveniste et al. [1990], Ang and Farhang-Boroujeny [2001]) has been devoted to finding the most appropriate step size to accelerate the convergence of LMS. The simplest and most commonly proposed method for fast convergence of LMS is to start with a large step size and decrease it over time, i.e. the Variable Step-size LMS (VSLMS). Most methods require some adaptive feedback to determine how to decrease η. In general for VSLMS (see Mathews and Xie [1993], Benveniste et al. [1990], Ang and Farhang-Boroujeny [2001]) the step-size can be updated by,

$$\eta_t := \eta_{t-1} - \rho \frac{\partial}{\partial \eta_{t-1}} L(\xi_t), \tag{6.13}$$

where the loss is $L(\xi_t) = \xi_t^2/2$, the residual is $\xi_t = y_t - (\mathbf{w}_t \cdot \mathbf{x}_t)$, and the constant ρ is some positive real number which controls the how the step-size changes. There are a number of different ways of constructing VSLMS, for instance the method of (6.13) also has a multiplication update version (see Ang and Farhang-Boroujeny [2001]). Potential problems arise with the approach of VSLMS since there is the possibility of instability because of the feedback process used by it. Further more, it is difficult to analyse the effective MSE with a varying step-size. One advantage of the mixture of LMS algorithms proposed is it does not have these potential problems, since each LMS algorithm has a constant step-size and does not have any feedback. The use of a hierarchy of fixed-share algorithms also allows the ideal share to be tracked as opposed to being predetermined (as is the case for a single fixed-share algorithm). It is also highly unlikely that we can pick M step-sizes such that the switching between them is at a constant rate.

6.6.4 ANG prior parameters selection

As discussed in Section 6.2, if the channel is sparse then ANG can have faster convergence than traditional LMS. We also know that the convergence performance of the ANG algorithm of equation (6.4) is dependent on the selection of a suitable prior $\phi(w_{t,i})$.

If the channel is not sparse then we might want to try $\phi(w_{t,i}) = 1$, as LMS generally has better convergence in this case. Here we focus on the choice of parameters for a given prior. The methods proposed in Martin et al. [2002] to select prior parameters involve running different ANG algorithms with different parameter settings and taking the one with best average MSE over a number of Monte-Carlo trials. This is not particularly practical because it is not online, and it assumes that the prior parameters are constant over the entire signal transmission.

We propose using the fixed-share algorithm to track the best mix of parameter settings for the ANG algorithm. Specifically we are interested in the parameter settings of the inverse power law prior $\phi(w_{t,i}) = (1/(w_{t,i}^{\beta} + \zeta))$. Though a mixture of different priors could also be considered, like the exponential prior $\phi(w_{t,i}) = \exp(\beta w_{t,i})$. The advantages of using the fixed-share hierarchy algorithm in selecting the prior parameters are that it is online (adaptively selects the most appropriate prior parameters at each iteration), and it can deal with situation where the share between the different parameters is not constant.

6.6.5 Switching from blind to decision-directed equalization

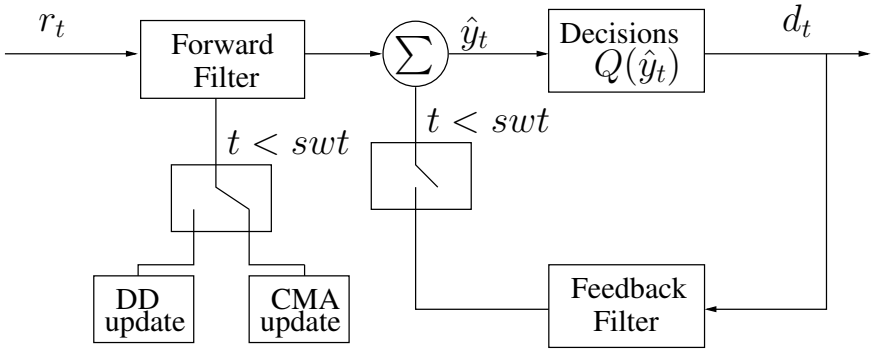

Figure 6.2: Model of the proposed system for switching from CMA to DD-DFE equalization. When the number of iterations is less than swt the forward filter coefficients are updated using CMA, otherwise they are updated according to LMS in decision-directed (DD) mode.

If it is not possible to send a training sequence of known symbols (labels) at the start of transmission, then the inter-symbol interference (ISI) can be reduced by using blind channel equalization, as pioneered by Sato [1975]. One common method (Treichler et al. [1998]) of blind equalization is to initially use the Constant Modulus Algorithm (CMA) and once the ISI is sufficiently reduced switch to the Decision Directed (DD) mode (see Appendix C.1 for more details on CMA). The point at which one switches to DD mode can make a significant difference, as the cost surface of a poorly initialised DD equalizer can result in ill convergence (Mazo [1980]). We will consider blind equalization where the DD mode is the Decision Feedback Equalizer (DFE) or DD-DFE (see Appendix C.2 for more details on DD-DFE). The DD-DFE has also been shown to not converge to

the optimal MSE given inappropriate initialisation (Casas et al. [1998]).

A number of switching strategies have been proposed to give better transition from CMA to DD-DFE (Treichler et al. [1998], McGinty [2002]). The method proposed here is similar to that shown in Treichler et al. [1998]. We start with CMA switching after a set number of iterations swt to DD-DFE, initialising the forward filter of the DD-DFE with the CMA filter (though one could also incorporate the detuning technique of McGinty [2002]), the proposed system is shown in Figure 6.2. Determining the symbol error rate as proposed in Treichler et al. [1998] may be difficult. Instead, given the relatively low computations per update of DD-DFE and CMA, we run M CMA and DD-DFE equalizers with different values of swt in order to track the best performing choice of time swt to transition from CMA to DD-DFE using the mixing Algorithm 10. One advantage of using the FSH algorithm in this way is that if one choice of swt or step-size leads to the DD-DFE i having a larger loss then the associated fixed-share weight $v_t^h(i)$ will diminish to a small value close to $\sum_{j=1}^{K} h_{t,j}^m \alpha_j / M$. This provides the FSH algorithm with some resistance to becoming unstable if one corresponding DD-DFE does. By setting $\alpha_j = 0$, and using a single fixed-share algorithm (the static expert algorithm) one can ensure complete immunity to an unstable DD-DFE, by making the weight $v_t^m(i)$ associated with expert i approach zero asymptotically.

There are other possible scenarios for using the fixed-share algorithm in switching between CMA and DD-DFE: one was explained by Harrington [2003] where a CMA and DD-DFE were mixed using a single fixed-share algorithm and the predictions from the fixed-share algorithm were used for feedback decisions.

6.7 Experiments

The aim of this section is to demonstrate and analyse the proposed scenarios of Section 6.6. Unless otherwise stated, the results in this section were produced with 500 Monte-Carlo trials. In all experiments we used the FSH algorithm with parameters: learning rate $\nu = 10$, shares $\boldsymbol{\alpha} = [0.1, 0.01, 0.001, 0.0001]'$ and $K = 4$.

6.7.1 Model tracking (multipath)

The experiment performed in this section investigates the use of equalization to compensate for the distortion caused by multipath (see Section 6.6.1). The main intention of this experiment was to study the capabilities of the FSH algorithm in tracking a changing model, and was not intended as a realistic application of it.

We consider the particular model of multipath which comprises of the unaffected line of sight signal, and a single multipath interferer which is a time-delayed version of the unaffected signal (where time delay is larger than symbol time). This model is essentially the same as that used in Ricean fading (see Sklar [1997]). We consider the situation where the time-delay and power of the interfering signal are both varying over time. To simulate this time variation in the multipath model we switch between three differ-

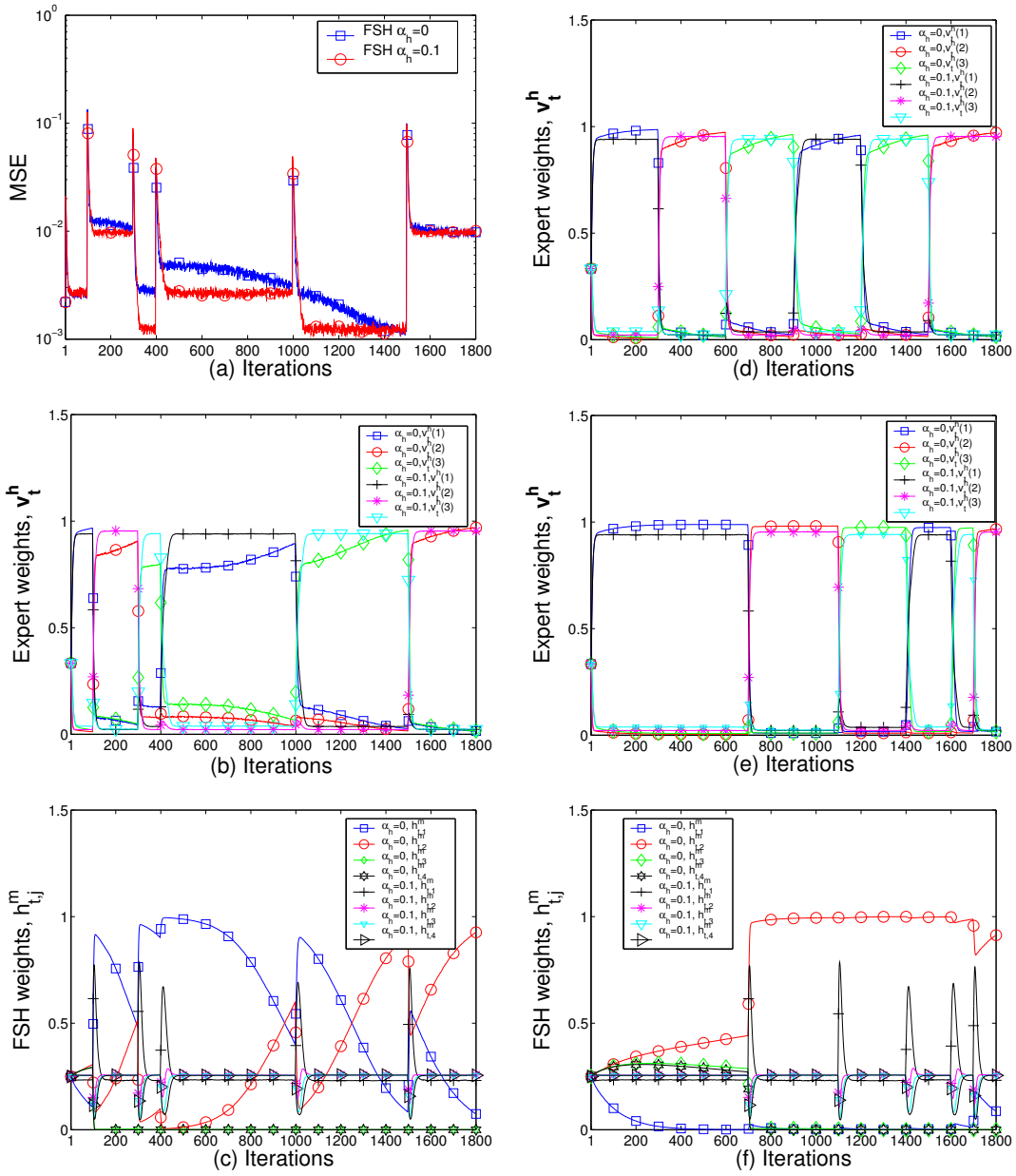

Figure 6.3: Results for FSH algorithm tracking the equalizer model which matches the given (time-dependent) multipath. We compare the performance of FSH algorithm with $\alpha_h = 0$ and 0.1. For $\mathbf{a}_s$ changing at 1,100,300,400,1000,1500 iterations: (a) mean squared error (MSE), (b) expert weights $\boldsymbol{v}_t^h$ (weight applied to each equalizer's prediction) and (c) FSH weights $h_{t,j}^m$ (weight applied to fixed-share algorithm j) for $j = 1, \ldots, K = 4$. For $\mathbf{a}_s$ with a constant share, changing at iterations of 1,300,600,900,1200,1500: (d) expert weights $\boldsymbol{v}_t^h$. For $\mathbf{a}_s$ changing at 1,700,1100,1400,1600,1700 iterations: (e) expert weights $\boldsymbol{v}_t^h$ and (f) FSH weights $h_{t,j}^m$ for $j = 1, \ldots, K = 4$.

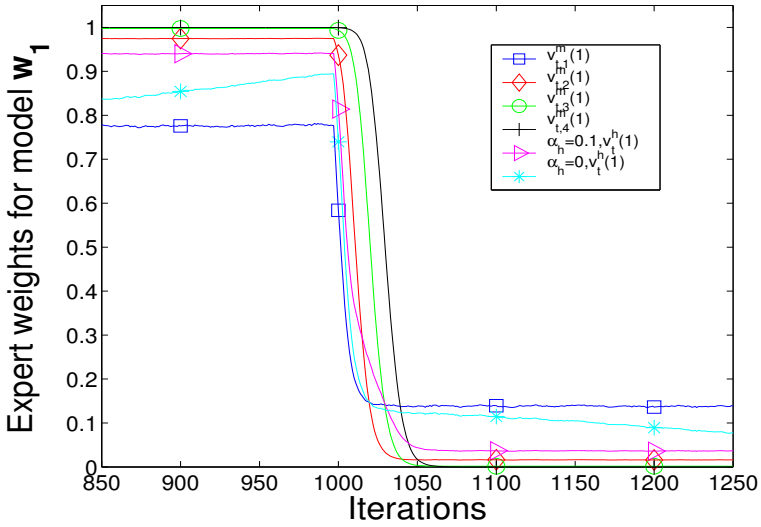

Figure 6.4: Various weights generated by FSH algorithm showing the transition at iteration 1000 from expert 1 (equalizer $\mathbf{w}_1$) to expert 3 for the multipath experiment with $\mathbf{a}_s$ changing at 1,100,300,400,1000,1500 iterations.

ent channels according to the sequence $\mathbf{a}_s = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_1, \mathbf{a}_3, \mathbf{a}_2)$, where the channels are: $\mathbf{a}_1 = [0, 0, 0, 1, 0, 0.2, 0]'$, $\mathbf{a}_2 = [0, 0, 0, 1, 0.3, 0, 0]'$, and $\mathbf{a}_3 = [0, 0, -0.1, 1, 0, 0, 0]'$. For each multipath channel $\mathbf{a}_i$, $i = 1, \ldots, M$, there was a corresponding equalizer $\mathbf{w}_i$ (expert model), where $\mathbf{w}_1 = [0, -0.2, 0, 1, 0, 0, 0]'$, $\mathbf{w}_2 = [0, 0, -0.3, 1, 0, 0, 0]'$, and $\mathbf{w}_3 = [0, 0, 0, 1, 0.1, 0, 0]'$. We focus on studying the ability of the FSH algorithm to track the appropriate equalizer, therefore match the channel changing sequence $\mathbf{a}_s$. We also note the inclusion of AGWN in our model with an SNR of 30dB.

We explore three different scenarios of switching between the three different multipath channels in $\mathbf{a}_s$.

The first scenario considered the switched between channels defined in $\mathbf{a}_s$ at the following iteration times 1,100,300,400,1000,1500. In this scenario the switching times between experts were small initially, and progressively got larger. We see from Figures 6.3 (a), (b) and (c) that the FSH algorithm with $\alpha_h = 0.1$ was more successful at tracking this switching scenario than with $\alpha_h = 0$. From Figure 6.3 (c) we can see that for $\alpha_h = 0$ that most of the hierarchy's weight ($h^m_{t,j}$) was placed on the fixed-share update with $\alpha = 0.1$ ($h^m_{t,1}$). Distributing the weight to the fixed-share update with a higher share (α) facilitates faster transition between different equalizer models (experts), as shown in Figure 6.4. We see from Figure 6.4 that while the transitions are close to ideal, the final weight's value assigned to equalizer (expert) 1 was not; as it was a lot smaller than one. On the other hand, for $\alpha_h = 0.1$ (see Figure 6.4) the FSH algorithm was able to strike a compromise between fast switching between the "best" equalizer models and having a weight value ($v^h_t(1)$) which is closer to the ideal value, i.e. 1. We see from Figure 6.4 that although the fixed-share algorithm with (share)

$\alpha = 0.1$ ($v_t^m(1)$) switches almost exactly at the right point (iteration 1000) the value of the weight is not very close to 1. This demonstrates higher we set the share (α) of the fixed-share algorithm the more weight that is uniformly given to each expert irrespective of their performance. Resulting in potentially larger prediction errors, but allowing for faster transitions between the experts. For this reason we want to find a mixture of settings of α, larger at transition then smaller when not in transition.

In the second scenario a constant switch rate was considered, where the channels in $\mathbf{a}_s$ were switched at iterations 1,300,600,900,1200,1500. We see from Figure 6.3 (d) that the FSH algorithm with $\alpha_h = 0$ was slower at learning the correct equalizer when compared with the FSH algorithm with $\alpha_h = 0.1$. One explanation as to why $\alpha_h = 0$ was less effective was the bound of (6.9) which is minimised when α equals the optimal share (see Figure 6.1). In this scenario where the share was constant the optimal switching rate was 1/300. For $\alpha_h = 0$ the final mixture of fixed-share algorithms was $\boldsymbol{v}_T^m = [0.045, 0.955, 0, 0]'$, and this essentially corresponds to the fixed-share algorithm with a suboptimal share of 0.01.

In the third scenario the transitions for sequence $\mathbf{a}_s$ occurred at iterations: 1,700,1100,1400,1600,1700. In this instance the FSH algorithm with $\alpha_h = 0$ was better able to track the "best" equalizer model than in the previous two scenarios. Setting $\alpha_h = 0$ in the FSH algorithm resulted in it tracking a smaller value of $\alpha = 0.01$ than in the first scenario. In the first scenario the FSH algorithm with $\alpha_h = 0$ got stuck on the larger share $\alpha = 0.1$ and was unable to change to $\alpha = 0.01$ until about 1500 iterations (see Figure 6.3 (c)).

6.7.2 Determining model order

We conducted the same adaptive equalization experiment as Singer and Feder [1999]. The channel used in the experiment was $\mathbf{a} = (1, 0.75, 0.5, 0.2, 0.1)$ with additive Gaussian noise with zero mean and standard deviation of 0.025 (SNR of approximately 32dB). The simulated signal was BPSK, such that $y \in \{+1, -1\}$. We used a single implementation of LSALE, as described in Satorius and Pack [1981], with model orders ranging up to $M = 11$.

We study the three model orders with the largest disparity in the accuracy of their predictions. These three model orders are displayed in Figure 6.5 where the model orders were $m = 2, 4$ and 8. We found that the best MSE performance of LSALE was when the model order was $m = 8$, as shown in Figure 6.5 (a) and (c). Note that the lowest excess MSE out of the 11 orders was found when the model order was 8.

Figure 6.5 (a) shows that the fixed-share hierarchy (FSH) algorithm during convergence has a lower MSE compared to the universal equalization algorithm (UEA) of Singer and Feder [1999]. From Figure 6.5 (b) it is apparent that the UEA algorithm has not been able to reduce the weight allocated to model orders 2 and 4. This is due to UEA being essentially a static expert update. Unlike UEA the FSH algorithm which is able to track the best performing LSALE order of 8. Figures 6.5 (c) and (d) give

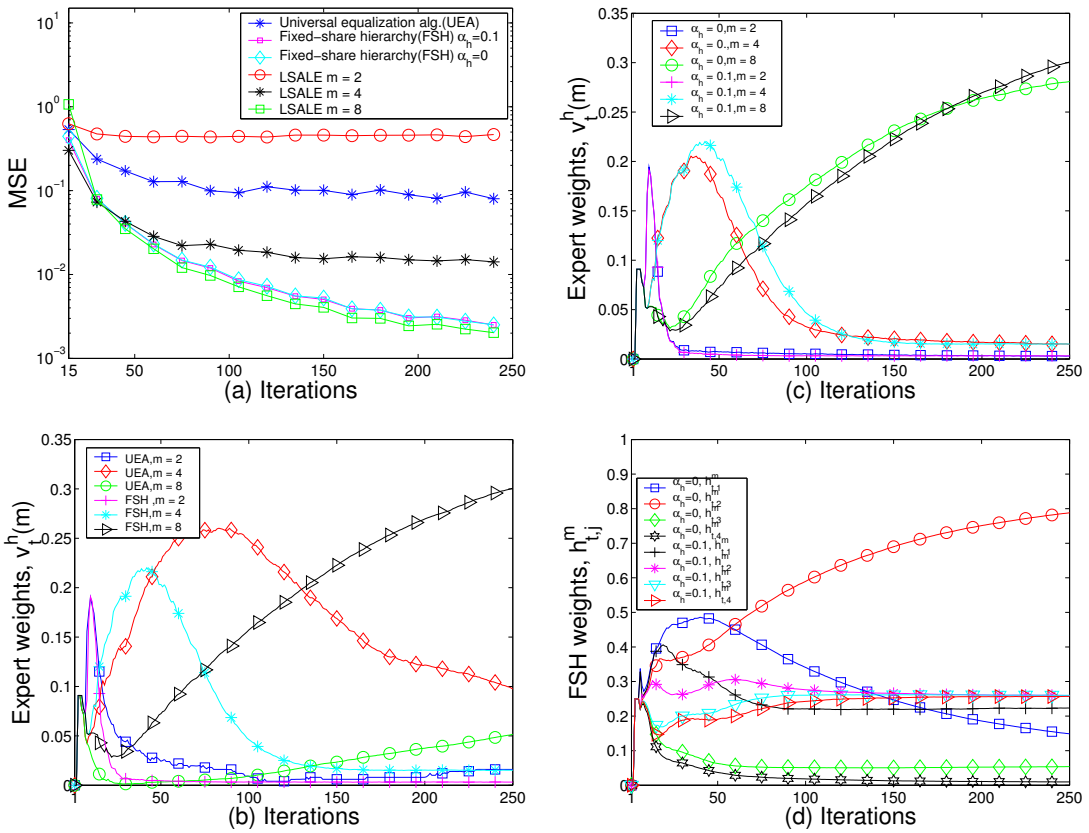

Figure 6.5: Results of tracking the best model order of LSALE. We compare the algorithms UEA and FSH, with the individual LSALE algorithm. For graphical comparison we plot the three model orders $m = 2, 4, 8$ of the eleven models considered in UEA and FSH algorithms. We show: (a) MSE performance of UEA, FSH, and LSALE for three chosen model orders $m = 2, 4, 8$; (b) comparing expert weights (the weight assigned to each LSALE model's prediction for $m = 1, \ldots, 11$,) $v_t^h(m)$ of UEA and FSH ($\alpha_h = 0.1$) algorithms; (c) expert weights $v_t^h(m)$, for FSH algorithm comparing $\alpha_h = 0.1$ with $\alpha_h = 0$; (d) FSH weight (weight applied to each fixed-share algorithm with share α_j for $j = 1, \ldots, 4$,) $h_{t,j}^m$, for FSH algorithm comparing $\alpha_h = 0.1$ with $\alpha_h = 0$.

some insight into whether allowing the share to vary ($\alpha_h = 0.1$) has any advantages compared to a single share ($\alpha_h = 0$). We see from Figures 6.5 (c) that there was little difference between setting $\alpha_h = 0.1$ and $\alpha_h = 0$. In this example it is sufficient to set $\alpha_h = 0$.

This experiment demonstrates the advantage of the FSH algorithm compared to UEA which assumes that the best performing model order of the LSALE does not change over time. The computational complexity of UEA is $O(M)$, whereas the FSH algorithm is of $O(KM)$, therefore FSH algorithm represents an increase in computational complexity.

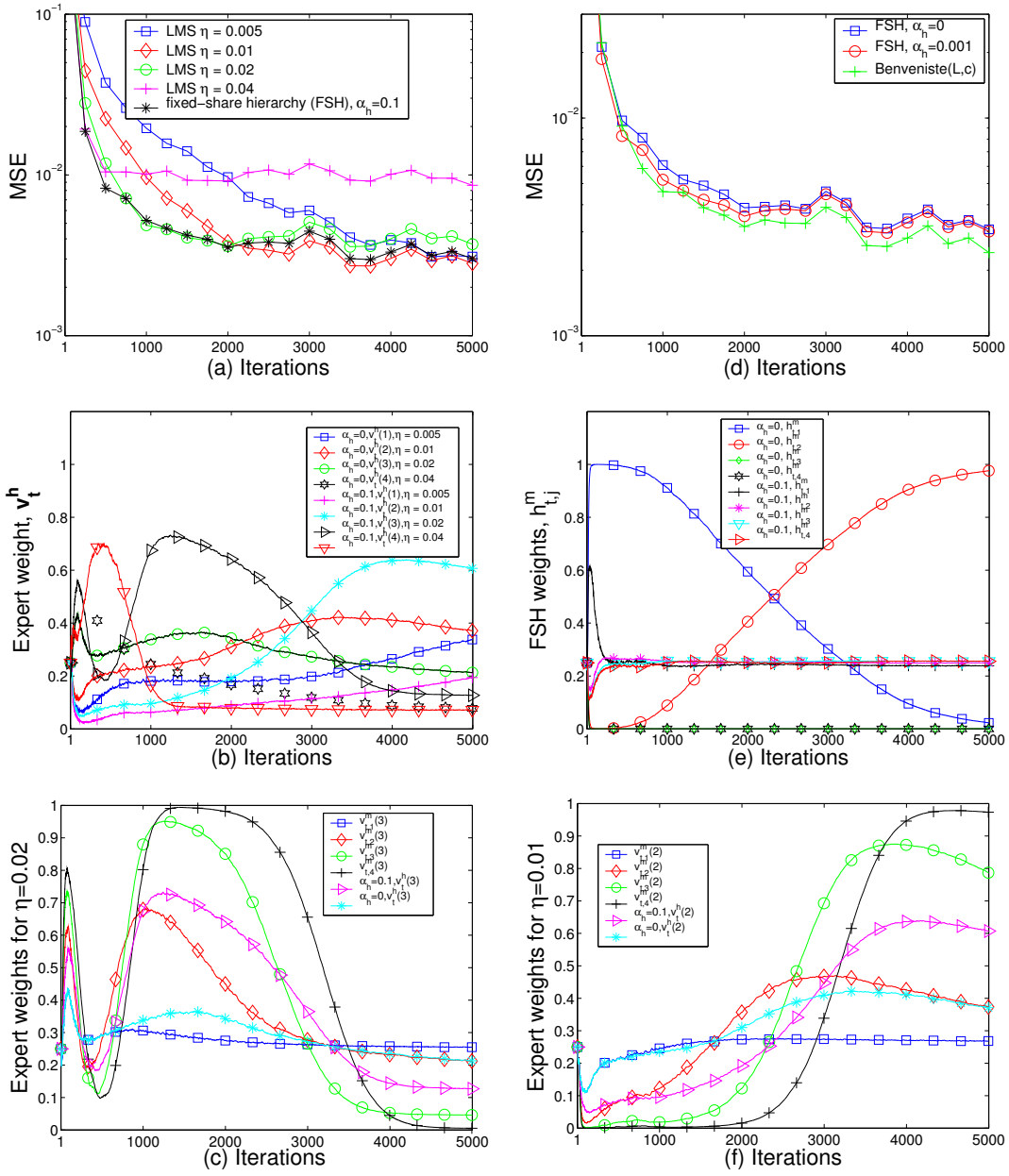

Figure 6.6: Tracking the step-size/learning rate which produces the smallest cumulative loss (and MSE) during the convergence of LMS. Plotted here are: (a) MSE performance comparing FSH algorithm ($\alpha_h = 0.1$) with LMS algorithm with various step-sizes η; (b) expert weights $v_t^h(m)$ (applied to each LMS prediction $\hat{y}_t(m)$ for $m = 1, \ldots, M = 4$); (c) FSH weights associated with LMS equalizer with $\eta = 0.02$;(d) MSE performance of FSH with $\alpha_h = 0.1$, FSH with $\alpha_h = 0$, and Benveniste(L,c); (e) FSH weight $h_{t,j}^m$ for $j = 1, \ldots, K = 4$; (f) FSH weights associated with LMS equalizer with $\eta = 0.01$.

6.7.3 Tracking LMS step-sizes

The next experiment tests the idea of choosing the most appropriate step-size with a mixture of LMS algorithms for a stationary channel in the presence of additive noise. The stationary channel used was from [Proakis, 2001, page 631] $\mathbf{a} = (0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0, 0.21, \quad 0.03, 0.07)$. The training data was generated from a binary sequence (BPSK) filtered by $\mathbf{a}$, with zero mean Gaussian noise and variance $\sigma_n = \sqrt{10^{-3}}$. The fixed-share hierarchy of Algorithm 10 was used with four independently run LMS algorithms with step sizes, $\eta = 0.005, 0.01, 0.02, 0.04$. Each LMS algorithm had a weight vector or equalizer of length $d = 31$.

A comparison was made between the FSH algorithms with the various Variable Step-size LMS (VSLMS) algorithms discussed in Ang and Farhang-Boroujeny [2001], with the exception of the Mathews algorithm, since Ang and Farhang-Boroujeny [2001] reported better test error results with the Benveniste algorithms. In endeavouring to find best of the VSLMS algorithms various settings of the parameter $\rho = 1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}$ were tried (ρ as defined in (6.13)). The initial step-sizes for the various VSLMS algorithms were set at the largest step-size of the LMS algorithms (0.04). All the VSLMS algorithms required hard limiting of their step-sizes at each iteration to maintain stability of the algorithm. It was also found for $\rho \geq 1 \times 10^{-4}$ that the step-sizes could become negative, requiring the step-size to be set to zero. After comparing all the VSLMS algorithms it was found that the linear update Benveniste algorithm (for short Benveniste(L,c)) had the best final MSE with $\rho = 1 \times 10^{-4}$.

The results of Figures 6.6 (a) and (b) especially illustrate the effectiveness of the FSH algorithm in tracking the value of η for the LMS algorithm with the lowest MSE. We see in Figure 6.6 (b) that FSH gradually tracks as it converges the smaller step-sizes from $\eta = 0.04$, to $\eta = 0.02$, and finally to $\eta = 0.01$, by making the expert weight associated with that learning rate closer to one. From Figure 6.6 (d) we see the effect α_h (the share between individual fixed-share algorithms) has on the MSE performance of the FSH algorithm. For $\alpha_h = 0.1$ the MSE performance was better up to 2000 iterations compared to setting $\alpha_h = 0$. This is supported by the Figure 6.6 (b) which shows that with $\alpha_h = 0.1$ there are clearer transitions between the different learning rates compared to $\alpha_h = 0$. Figures 6.6 (c) and (f) show that the FSH algorithm like in the multipath experiment makes a compromise between the speed of transition between the experts and the weight assigned to the "best" expert. Looking at Figure 6.6 (e) the performance in this experiment with $\alpha_h = 0$ is similar to the multipath experiment of Section 6.7.1. The FSH algorithm with $\alpha_h = 0$ locked onto to the largest share due to its static expert update, and took a long time to change to the better share of 0.01. We also note that the FSH algorithm performance was not quite as good as the Benveniste(L,c) algorithm, although it had the advantage of not requiring the tuning of an additional parameter ρ.

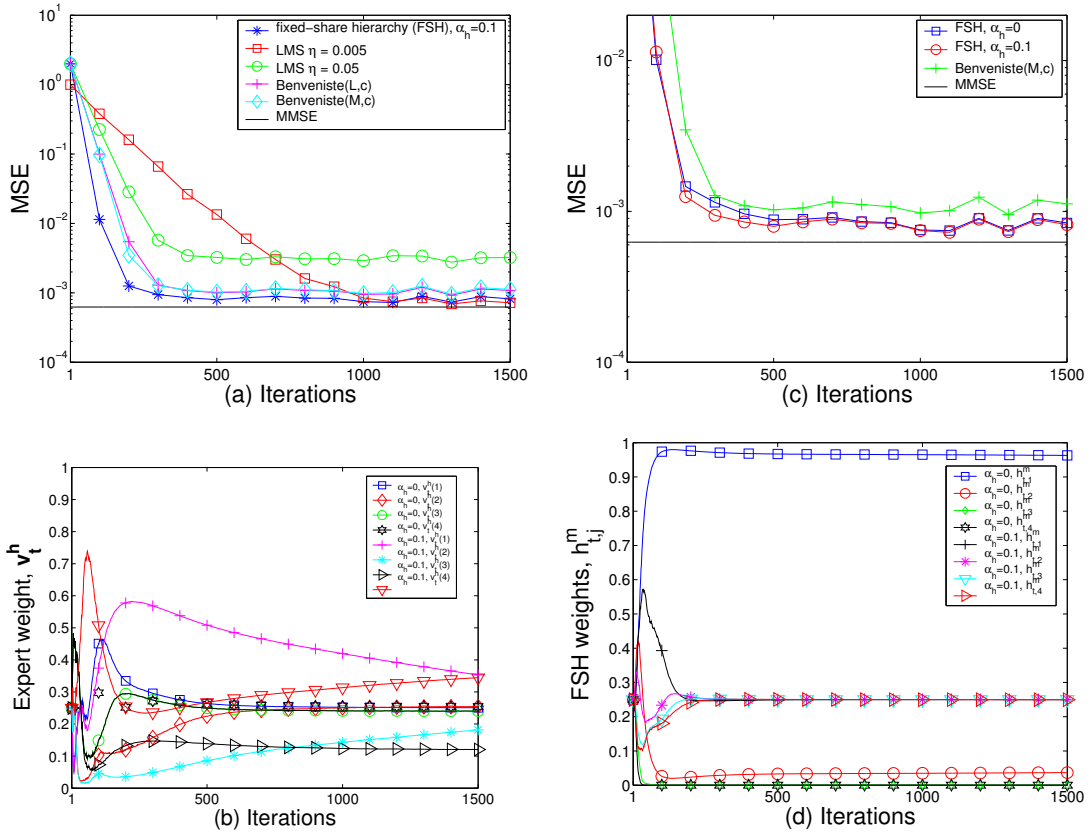

Figure 6.7: Tracking the prior parameters of the ANG algorithm which produce the smallest cumulative loss (and MSE). Plotted are :(a) MSE performance comparing FSH algorithm ($\alpha_h = 0.1$) with LMS algorithm with various step-sizes η and VLMS algorithms Benveniste(M,c) and Benveniste(L,c); (b) expert weights $v_t^h(m)$ (applied to each LMS prediction $\hat{y}_t(m)$ for $m = 1, \ldots, M = 4$); (c) MSE performance of FSH with $\alpha_h = 0.1$, FSH with $\alpha_h = 0$, and Benveniste(L,c); (d) FSH weight $h_{t,j}^m$ for $j = 1, \ldots, K = 4$.

6.7.4 ANG prior parameters selection

In this section we demonstrate the use of Algorithm 10 in tracking the prior parameters of the ANG algorithm when the channel is sparse, as discussed in Section 6.6.4. We consider the specific case of the ANG algorithm using the inverse power law prior $\phi(w_{t,i}) = (1/(w_{t,i}^{\beta} + \zeta))$ with $\zeta = 0.1, 0.3$ and $\beta = 0.125, 0.5$. A sparse channel was created with a length of $L = 15$. The channel had the value zero in all but positions $[1, 5, 10]$, which had the values $[0.1, 1, -0.1]$ respectively. The model included additive Gaussian noise with a variance of 0.025^2 (SNR of approximately 32dB) giving a corresponding theoretical MMSE of 6.25×10^{-4} (see Figures 6.7 (a) and (c)).

From the results in Figure 6.7 (a) the FSH algorithm has a faster convergence when compared to LMS with $\eta = 0.005$ (to ensure a fair comparison their steady state MSE were made similar). LMS was also plotted with the same $\eta = 0.05$ as used by the

ANG mixture, though it converged to a higher MSE. Figure 6.7 (b) shows the weight, $v_t^h(i)$, given to each parameter setting of the prior per iteration, by the FSH algorithm when forming the weighted average of (6.6). Figures 6.7 (a), (b) and (c) show the performance of the fixed-share hierarchy for two different settings of α_h: 0 and 0.1. From Figure 6.7 (a) we see that the FSH algorithm $\alpha_h = 0.1$ had a faster convergence compared to $\alpha_h = 0$. It is shown by Figure 6.7 (a) that the FSH algorithm outperformed the VSLMS algorithms: Benveniste(L,c) and Benveniste(M,c). We see that Benveniste(M,c) had faster convergence compared to the linear update of Benveniste(L,c). This is as expected given that the multiplying update of Benveniste(M,c) is similar to the ANG algorithm's update, hence works well when the channel is sparse. Note that the prior which produced the lowest MSE in Figure 6.7 (a) was not a single parameter setting, but a mixture of parameter settings, as shown by Figure 6.7 (b). Whilst initially the assumption of a varying share between experts was important after some time the mixture becomes relatively constant, as indicated in Figure 6.7 (b). Again like several of the previous experiments the assumption of a constant "best" share ($\alpha_h = 0$) results in poorer performance. Figure 6.7 (d) demonstrates how $\alpha_h = 0.1$ is able to adapt from a initially fast switching between experts to a more appropriate mixture of shares than compared to $\alpha_h = 0$. As already stated, the faster the convergence of our equalizer the smaller the training sequence required, therefore allowing for more effective utilisation of bandwidth.

6.7.5 Switching from blind to decision-directed equalization

The simulation in this section investigates the use of the FSH algorithm in selecting the most appropriate number of iterations at which to switch from blind equalization (CMA) to decision directed feedback equalization (DD-DFE), as proposed in Section 6.6.5. We considered the same stationary channel **a** as in Section 6.7.3, though with additive noise of variance 0.01^2. We ran three independent CMA and DD-DFE equalizers with switching iterations of 1000, 1500, and 3000. The CMA used was the second order Godard algorithm with $R_2 = 1$, step-size $\eta = 0.01$ and length of 31. At switching the CMA equalizer taps were used to define the forward taps of the DD-DFE equalizer. The feedback filter had a length of 31 and used a step-size $\eta = 0.005$. The loss function used by the fixed-share hierarchy was $L(\xi) = (\xi)^2/2$, where $\xi = |\hat{y}|^2 - R_2$.

Figure 6.8 shows the comparison of the second order dispersion ($D^{(2)} = \mathrm{E}(|\hat{y}|^2 - R_2)^2/2$) for the three choices of *swt* of transition from CMA to DD-DFE, and for the fixed-share hierarchy algorithm's prediction. As we see from Figure 6.8 (a) the fixed-share hierarchy was successfully able to track the *swt* with the lowest dispersion. From Figure 6.8 (b) it is evident that switching between the experts was not equally spaced, therefore vindicating the assumption that the share was not constant in this case. The slower transitions between the different CMA to DD-DFE schemes allowed the setting of $\alpha_h = 0.001$, which was smaller than the previous experiments using $\alpha_h = 0.1$.

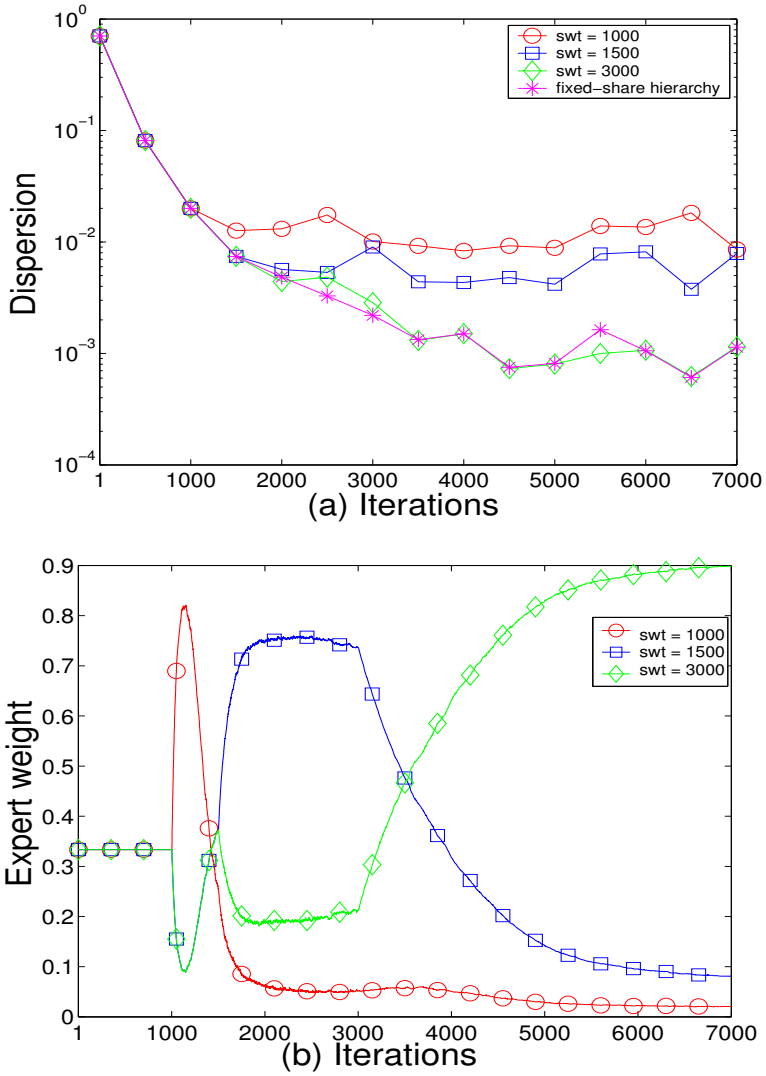

Figure 6.8: Results of using the fixed-share hierarchy algorithm to switch from blind (CMA) to decision directed feedback equalization (DD-DFE) with three independently run equalizers with switching iterations 1000, 1500 and 1500: (a) Dispersion and (b) hierarchy's expert weights $v_t^h(i)$ for $i = 1, \ldots, M$ and $\alpha_h = 0.001$.

6.8 Summary

We introduced an algorithm for tracking the expert (equalizer) with the lowest cumulative loss where the share between experts is not constant. The algorithm is able to track a variable share by being constructed with a two tier hierarchy of fixed-share algorithms. We demonstrated the versatility of the fixed-share hierarchy (FSH) algorithm by its successful application to several scenarios of adaptive equalization. We showed that the FSH algorithm has a wide range of uses, from model tracking for multipath to parameter tracking for time invariant channels. Experiments demonstrated the gen-

erality of the FSH algorithm applied to adaptive equalization, showing improved performance compared to the universal equalization algorithm (UEA) of Singer and Feder [1999], and comparable performance to variable step-size LMS (VSLMS), without its additional stability issues and constraints.

Conclusions and Further Work

This chapter summaries the main contributions of this thesis, the conclusions drawn from them, and some new interesting problems arising which warrant further investigation.

7.1 Main contributions

The main contributions of this thesis are:

- The development and analysis of an online implementation of the Bayes point machine (OBPM) algorithm for linear classifiers. We demonstrate that it can be keneralised to deal with problems that are not linearly separable, though without considering methods for bounding the kernels representation, i.e. methods prescribed in Kivinen et al. [2002], Crammer et al. [2004].
- An empirical comparison of different online learning variants of the Perceptron algorithm: voted-Perceptron algorithm, Bagging algorithm and OBPM algorithm.
- The presentation of a new online variant of the Perceptron algorithm for ranking instances (PRank), called the online aggregate PRank Bayes point machine (OAP-BPM), which was shown empirically to improve generalisation while maintaining the ranking order.
- Novel online implementations of Bagging and the Bayes point machine algorithms for regression and classification settings, applied to the communications problem of adaptive equalization.
- The introduction of a new mixture of experts algorithm allowing for the optimal switching rate between experts to change over time, called the fixed-share hierarchy (FSH) algorithm.
- Demonstration of the successful application of the FSH algorithm to the problem of adaptive equalization in a number of scenarios.

7.2 Conclusions and further work

The primary appeal of the Online Bayes Point Machine (OBPM) algorithm of Chapter 2 is its simplicity: one merely needs to run a number of Perceptrons in parallel on different (random) subsamples of the training sequence. Hence, it is well suited to very large data sets. We illustrated the effectiveness of the algorithm with experiments conducted on a synthetic data set and a number of different real world data sets. We demonstrated that this method can be used in both the online and batch setting with comparable, and in some cases with improved performance compared to methods which maximise the margin explicitly. Given that the OBPM algorithm was a subsampling technique it was interesting to note that some of the experiments had smaller test error results when compared with the billiard algorithm of BPM, which had the entire training set to learn from, allowing for repetitions during training of examples contained in that set. Several proposed extensions to this work are the investigation into an online variant of the EP algorithm of Minka [2001] using a similar approach to the OBPM algorithm, and further study of the use of online kernels methods of Kivinen et al. [2002], Crammer et al. [2004] with the OBPM algorithm.

From Chapter 3 we found that there was little difference in terms of test error performance between combining the linear weights as opposed to the hypotheses when using the Perceptron. This is quite surprising since the former gives a linear classifier and the latter a classifier from the convex hull which can be much more complex. The empirical results do indicate that there is some gain to be made when using kernels. There is an advantage in combining the weights, i.e. BPM, as it is more efficient at implementation of the classifier, because we only need to store a single weight vector instead of N hypotheses. One interesting outcome was that the voted-Perceptron showed a significant improvement compared to most methods for the linear case of the Perceptron. The combination of the OBPM algorithm and the voted Perceptron algorithm, the OBPM-voted algorithm, had favourable results in both the linear and non-linear settings indicating this is one of the best voting strategies to use with the Perceptron. One limiting factor of the voted Perceptron for large data sets is its storage requirements which grow linearly with the number of mistakes. A problem for further research would be to develop a method of pruning the voted Perceptron weights whilst maintaining its good generalisation performance and remaining online.

In Chapter 4 we introduced an online method which improves the generalisation performance of the Perceptron ranking algorithm (PRank of Crammer and Singer [2002]), whilst preserving the ranking order of the instances, called the OAP-BPM algorithm. One advantage of the OAP-BPM algorithm if implemented in a parallel architecture is it represents a "real" time solution making it quite a practical method to use in real world applications of information retrieval. One further area of research is to incorporate the methods for online kernels with the OAP-BPM algorithm. Further study could be undertaken in applying the OAP-BPM algorithm to some real world problems,

such as large scale document retrieval via collaborative filtering. Another idea worth researching is whether it is possible to design a large margin Perceptron with the two methods of Shashua and Levin [2003] using ALMA or ROMMA.

One of the main conclusions to be drawn from Chapter 5 is that the Bagging algorithm applied to traditional equalizers using a buffered approach leads to improved test error results in the presence of additive Gaussian white noise and label flipping. The buffered result is significant, since the test results were considerably better than the NLMS algorithm. We can conclude that the subsampling technique combined with the Perceptron classifier (the OBPM algorithm) provided comparable test error performance to the NLMS algorithm. Unlike the LMS algorithm, and the NLMS algorithm, the Perceptron algorithm does not require careful tuning of the step-size to ensure stable performance. As we observed in Chapter 2, the convergence of the subsampling method (OBPM algorithm) was dependent on τ; the smaller τ the slower the convergence. For a wide range of $\tau < 1$ the final the subsampling method's test error results were close at convergence. The same could not be said about the step-size of the NLMS algorithm: there was a trade-off between convergence speed and the final test error performance. Further research could be done to incorporate phase tracking with the various aggregate equalization techniques to investigate a more complete system.

From Chapter 6 we conclude that in the domain of adaptive equalization there are a number of applications where a hierarchy of fixed-share algorithms proves useful. The experiments demonstrated that choosing a pool of the equalizers (experts) and their parameters such that the optimal switching rate between them is a constant was not practical. In several of the experiments the fixed-share hierarchy algorithm had favourable MSE compared to methods which assume a constant switching rate, and to methods which assume a constant expert over the entire training sequence. Further research could be done into the use of this algorithm on non-stationary channels. A specific example is the tracking of the forgetting factor of the *leaky* LMS algorithm when the channel is time varying. While our focus was on the problem of adaptive equalization, it is believed the fixed-share hierarchy algorithm has wider implications for time series prediction in general, as it is naturally online. Most statistical models are relatively cheap to compute, therefore considering several models at once is entirely practicable.

Thesis proofs

A.1 Lower bound proof of (2.1)

Recall that the convex hull of weights $\mathbf{w}_1, \ldots, \mathbf{w}_N$ is defined by $C^N := C^N(\mathbf{w}_1, \ldots, \mathbf{w}_N) := \{\sum_i \alpha_i \mathbf{w}_i \colon \|\boldsymbol{\alpha}\|_1 = 1, \boldsymbol{\alpha} \geq \mathbf{0}\}$. From the definition of C^N, and $\tilde{\mathbf{w}} = \sum_{i=1}^N \alpha_i \mathbf{w}_i$, we find that $\|\tilde{\mathbf{w}}\|_2$ is equal to

$$\sqrt{\left(\sum_{i=1}^N \alpha_i \mathbf{w}_i \cdot \sum_{j=1}^N \alpha_j \mathbf{w}_j\right)} = \sqrt{\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \left(\mathbf{w}_i \cdot \mathbf{w}_j\right)}. \tag{A.1}$$

For any example $z_i := (\mathbf{x}_i, y_i)$ from $\mathbf{z} = (z_1, \ldots, z_t)$ and exploiting the normalization $\|\mathbf{w}_i\|_2 = 1$ for $i = 1, \ldots, N$, we conclude that

$$y_i \left(\tilde{\mathbf{w}} \cdot \mathbf{x}_i\right) = y_i \left(\sum_{j=1}^N \alpha_j \mathbf{w}_i \cdot \mathbf{x}_i\right) = \sum_{j=1}^N \alpha_j y_i \frac{(\mathbf{w}_i \cdot \mathbf{x}_i)}{\|\mathbf{w}_i\|_2}. \tag{A.2}$$

Using the definition for the margin for a particular weight solution $\mathbf{w}$ over the sequence $\mathbf{z}$, $\gamma_{\mathbf{z}}(\mathbf{w}) := \min\{y_i \left(\mathbf{w} \cdot \mathbf{x}_i\right) / \|\mathbf{w}\|_2 : i \in \{1, \ldots, T\}\}$ with the result of (A.2), we obtain

$$y_i \left(\tilde{\mathbf{w}} \cdot \mathbf{x}_i\right) \geq \sum_{j=1}^N \alpha_j \gamma_{\mathbf{z}}(\mathbf{w}_j). \tag{A.3}$$

Combining (A.1) and (A.3) completes the proof that

$$\gamma_{\mathbf{z}}(\tilde{\mathbf{w}}) \geq \frac{\sum_{j=1}^N \alpha_j \gamma_{\mathbf{z}}(\mathbf{w}_j)}{\sqrt{\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j \left(\mathbf{w}_i \cdot \mathbf{w}_j\right)}}.$$

A.2 Proof of Theorem 2.3.1

The proof is started with the following scalar product,

$$(\mathbf{w}_{1,t}(\tau)\cdot\mathbf{w}_{2,t}(\tau)) = ((\mathbf{w}_{1,t-1}(\tau)+b_{1,t}\sigma_{1,t}y_t\mathbf{x}_t)\cdot(\mathbf{w}_{2,t-1}(\tau)+b_{2,t}\sigma_{2,t}y_t\mathbf{x}_t)), \quad (A.4)$$

where $b_{i,t}$ is the indicator function for Bernoulli trial t, Perceptron i; $\sigma_{i,t}$ is one if Perceptron i makes a mistake at iteration t and zero otherwise. Telescoping (A.4) gives,

$$\begin{aligned}(\mathbf{w}_{1,T}(\tau)\cdot\mathbf{w}_{2,T}(\tau)) = \sum_{i=1}^{T}\big[&(b_{1,i}\sigma_{1,i}y_i\,(\mathbf{w}_{2,i-1}(\tau)\cdot\mathbf{x}_i)+b_{2,i}\sigma_{2,i}y_i\,(\mathbf{w}_{1,i-1}(\tau)\cdot\mathbf{x}_i))\\ &+b_{1,i}b_{2,i}\sigma_{1,i}\sigma_{2,i}\|\mathbf{x}_i\|^2\big].\end{aligned} \quad (A.5)$$

Using the indicator function $I(a,b)$ with (A.5) it can be expressed as,

$$\begin{aligned}(\mathbf{w}_{1,T}(\tau)\cdot\mathbf{w}_{2,T}(\tau)) = \sum_{i=1}^{T}\big[&I(\sigma_{1,i}=1,\sigma_{2,i}=1)\left(b_{1,i}b_{2,i}\|\mathbf{x}_i\|^2-b_{1,i}|\,(\mathbf{w}_{2,i-1}(\tau)\cdot\mathbf{x}_i)\,|\right.\\ &\left.-\,b_{2,i}|\,(\mathbf{w}_{1,i-1}(\tau)\cdot\mathbf{x}_i)\,|\right)+I(\sigma_{1,i}=1,\sigma_{2,i}=0)b_{1,i}|\,(\mathbf{w}_{2,i-1}(\tau)\cdot\mathbf{x}_i)\,|\\ &+\,I(\sigma_{1,i}=0,\sigma_{2,i}=1)b_{2,i}|\,(\mathbf{w}_{1,i-1}(\tau)\cdot\mathbf{x}_i)\,|\big].\end{aligned} \quad (A.6)$$

Taking expectation w.r.t $S_1^T(\tau)$ and $S_2^T(\tau)$ of (A.6) gives,

$$\begin{aligned}\mathrm{E}_{S_1^T(\tau)S_2^T(\tau)}(\mathbf{w}_{1,T}(\tau)\cdot\mathbf{w}_{2,T}(\tau)) &= \sum_{i=1}^{T}\mathrm{E}_{S_1^i(\tau)S_2^i(\tau)}\big[I(\sigma_{1,i}=1,\sigma_{2,i}=1)b_{1,i}b_{2,i}\|\mathbf{x}_i\|^2\\ &\quad+b_{1,i}(C_i(\tau)-A_i(\tau))+b_{2,i}(D_i(\tau)-B_i(\tau))\big]\\ &=\sum_{i=1}^{T}\mathrm{E}_{b_{1,i}b_{2,i}}[b_{1,i}b_{2,i}]\,\mathrm{E}_{S_1^i(\tau)S_2^i(\tau)}\big[I(\sigma_{1,i}=1,\sigma_{2,i}=1)\|\mathbf{x}_i\|^2\big]\\ &\quad+\mathrm{E}_{b_{1,i}b_{2,i}}[b_{1,i}]\,\mathrm{E}_{S_1^i(\tau)S_2^i(\tau)}[C_i(\tau)-A_i(\tau)]\\ &=\sum_{i=1}^{T}\tau^2\mathrm{Pr}(\sigma_{1,i}=1,\sigma_{2,i}=1)\|\mathbf{x}_i\|^2\\ &\quad-\tau\mathrm{E}_{S_1^i(\tau)S_2^i(\tau)}[A_i(\tau)+B_i(\tau)]\\ &\quad+\tau\mathrm{E}_{S_1^i(\tau)S_2^i(\tau)}[C_i(\tau)+D_i(\tau)].\end{aligned} \quad (A.7)$$

where

$$\begin{aligned}A_i(\tau) &= I(\sigma_{1,i}=1,\sigma_{2,i}=1)|\mathbf{w}_{2,i-1}(\tau)\cdot\mathbf{x}_i|, B_i(\tau)=I(\sigma_{1,i}=1,\sigma_{2,i}=1)|\mathbf{w}_{1,i-1}(\tau)\cdot\mathbf{x}_i|,\\ C_i(\tau) &= I(\sigma_{1,i}=1,\sigma_{2,i}=0)|\mathbf{w}_{2,i-1}(\tau)\cdot\mathbf{x}_i|, D_i(\tau)=I(\sigma_{1,i}=0,\sigma_{2,i}=1)|\mathbf{w}_{1,i-1}(\tau)\cdot\mathbf{x}_i|.\end{aligned}$$

A.3 Proof of Theorem 2.3.2

The plan of the proof is to determine the upper bound of $\sum_{i=1}^{T} \mathrm{E}_{S_j^i(\tau)}(\sigma_{j,i})$ from first combining the lower and upper bounds of $\|\tilde{\mathbf{w}}_t\|^2$. First we determine a lower bound on $\|\tilde{\mathbf{w}}_T\|^2$ by considering $(\mathbf{w}^* \cdot \tilde{\mathbf{w}}_t)$,

$$(\mathbf{w}^* \cdot \tilde{\mathbf{w}}_t) = (\mathbf{w}^* \cdot \tilde{\mathbf{w}}_{t-1}) + \frac{1}{N} \sum_{j=1}^{N} b_{j,t}\sigma_{j,t} \left(\mathbf{w}^* \cdot \mathbf{x}_t y_t\right). \tag{A.8}$$

From (A.8), we get $(\mathbf{w}^* \cdot \tilde{\mathbf{w}}_t) \geq (\mathbf{w}^* \cdot \tilde{\mathbf{w}}_{t-1}) + \gamma_{\mathbf{z}} \frac{1}{N} \sum_{j=1}^{N} b_{j,t}\sigma_{j,t}$, and after T rounds of algorithm 3

$$(\mathbf{w}^* \cdot \tilde{\mathbf{w}}_T) \geq \frac{\gamma_{\mathbf{z}}}{N} \sum_{i=1}^{T} \sum_{j=1}^{N} b_{j,i}\sigma_{j,i}. \tag{A.9}$$

By using the Cauchy-Schwartz inequality with (A.9) gives $\|\mathbf{w}^*\|^2 \|\tilde{\mathbf{w}}_T\|^2 \geq ((\mathbf{w}^* \cdot \tilde{\mathbf{w}}_T))^2 \geq \frac{\gamma_{\mathbf{z}}^2}{N^2} \left(\sum_{i=1}^{T} \sum_{j=1}^{N} b_{j,i}\sigma_{j,i}\right)^2$ and resulting in the following lower bound

$$\|\tilde{\mathbf{w}}_T\|^2 \geq \left(\frac{\gamma_{\mathbf{z}}}{N} \sum_{i=1}^{T} \sum_{j=1}^{N} b_{j,i}\sigma_{j,i}\right)^2 / \|\mathbf{w}^*\|^2. \tag{A.10}$$

In order to find the upper bound of $\|\tilde{\mathbf{w}}_T\|^2$ we look at the dispersion measure $\frac{1}{N} \sum_{j=1}^{N} (\mathbf{w}_{j,T} - \tilde{\mathbf{w}}_T)^2$ which gives a measure of the deviation of each Perceptron j from the average weight of N Perceptrons. Using $\tilde{\mathbf{w}}_T = \frac{1}{N} \sum_{j=1}^{N} \mathbf{w}_{j,T}$ we can write

$$\begin{aligned} \frac{1}{N} \sum_{j=1}^{N} (\mathbf{w}_{j,T} - \tilde{\mathbf{w}}_T)^2 &= \frac{1}{N} \sum_{j=1}^{N} (\mathbf{w}_{j,T} \cdot \mathbf{w}_{j,T}) \\ &- \frac{2}{N^2} \sum_{j=1}^{N} \sum_{i=1}^{N} \left(\frac{\mathbf{w}_{i,T}}{N} \cdot \mathbf{w}_{j,T}\right) + \left(\sum_{i=1}^{N} \frac{\mathbf{w}_{i,T}}{N}\right)^2 \\ &= \frac{1}{N} \sum_{j=1}^{N} \|\mathbf{w}_{j,T}\|^2 - \|\tilde{\mathbf{w}}_T\|^2. \end{aligned} \tag{A.11}$$

Modifying (A.11) we get

$$\|\tilde{\mathbf{w}}_T\|^2 \leq \frac{1}{N} \sum_{j=1}^{N} \|\mathbf{w}_{j,T}\|^2. \tag{A.12}$$

For each independent Perceptron j evaluating $\|\mathbf{w}_{j,t}\|^2$ as

$$\begin{aligned} (\mathbf{w}_{j,t} \cdot \mathbf{w}_{j,t}) &= (\mathbf{w}_{j,t-1} \cdot \mathbf{w}_{j,t-1}) + 2\left(\mathbf{w}_{j,t} \cdot b_{j,t}\sigma_{j,t}\mathbf{x}_t y_t\right) \\ &+ (b_{j,t}\sigma_{j,t})^2 \|\mathbf{x}_t\|^2. \end{aligned} \tag{A.13}$$

The second term of (A.13) we know from the update rule for the Perceptron $2\left(\mathbf{w}_{j,t} \cdot b_{j,t}\sigma_{j,t}\mathbf{x}_t y_t\right) \leq 0$, the third term simplifies using $\left(b_{j,t}\sigma_{j,t}\right)^2 = b_{j,t}\sigma_{j,t}$ and that $\|\mathbf{x}_t\| = 1$, so (A.13) becomes

$$\|\mathbf{w}_{j,t}\|^2 \leq \|\mathbf{w}_{j,t-1}\|^2 + b_{j,t}\sigma_{j,t}. \tag{A.14}$$

After T rounds of (A.13) put into (A.12)

$$\|\tilde{\mathbf{w}}_T\|^2 \leq \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{T} b_{j,i}\sigma_{j,i}. \tag{A.15}$$

Combining equations (A.10) and (A.15) we have

$$\left(\frac{\gamma_{\mathbf{z}}}{N}\sum_{i=1}^{T}\sum_{j=1}^{N} b_{j,i}\sigma_{j,i}\right)^2 /\|\mathbf{w}^*\|^2 \leq \|\tilde{\mathbf{w}}_T\|^2 \leq \sum_{i=1}^{T}\sum_{j=1}^{N} b_{j,i}\sigma_{j,i} \tag{A.16}$$

Hence the bound on the estimated average number of updates across the N Perceptrons after t rounds of Algorithm 3 is given by

$$\frac{1}{N}\sum_{i=1}^{T}\sum_{j=1}^{N} b_{j,i}\sigma_{j,i} \leq \frac{\|\mathbf{w}^*\|^2}{\gamma_{\mathbf{z}}^2}. \tag{A.17}$$

The result above is as expected from Theorem Novikoff [1962]. However there is a difference in that (A.17) is for the combination of Perceptrons. This part of the proof can be replaced by simply stating Novikoff's bound for each Perceptron, and that the average $\tilde{\mathbf{w}}$ will therefore have the same bound. The above part of proof which is essentially Novikoff's result is included for completeness.

We must make clear that (A.17) gives the bound on the number of updates, not the number of mistakes; see the definitions and example at the start of Section 2.3.2 for clarification. We re-interate the difference here, a mistake is dependent only on whether $\sigma_t = 1$, whereas an update is made when $b_t = 1$ and $\sigma_t = 1$. It is possible that a mistake can be made on examples not seen for updating by that particular Perceptron.

To separate $b_{j,t}$ and $\sigma_{j,t}$ we look at (A.17) in the limit as $N \to \infty$. We know that the examples seen by Perceptron j is defined by the sequence of binary random variables, $S_j^t = (b_{j,1}, \ldots, b_{j,t})$, drawn i.i.d from $\mathcal{B}(\tau)$. Given that the sampling of the Perceptrons is done independently

$$\begin{aligned} \mathrm{E}_{S_j^t(\tau)}(b_{j,t}\sigma_{j,t}) &= \mathrm{E}_{b_{j,t}}(b_{j,t})\mathrm{E}_{S_j^t(\tau)}(\sigma_{j,t}) \\ &= \Pr(b_{j,t} = 1)\mathrm{E}_{S_j^t(\tau)}(\sigma_{j,t}) = \tau\mathrm{E}_{S_j^t(\tau)}(\sigma_{j,k}). \end{aligned} \tag{A.18}$$

The final desired result is achieved by combining (A.18) with (A.17)

$$\sum_{i=1}^{T} \mathrm{E}_{S_j^i(\tau)}(\sigma_{j,i}) \leq \frac{\|\mathbf{w}^*\|^2}{\tau\gamma_{\mathbf{z}}^2}.$$

Statistics

B.1 Student's t-distribution and associated statistics

Firstly, we assume that $X_1, \ldots, X_T$ be independent normal random variables with the same mean μ and variance σ^2. Each X_t in this thesis represents the percentage test error produced by a random and independently chosen test set example. The resulting random variable

$$Q = \sqrt{T}\frac{\bar{X} - \mu}{S} \tag{B.1}$$

has a Student's t-distribution with $T-1$ degrees of freedom, where $\bar{X} = \sum_{i=1}^{T} X_i$ and $S^2 = \frac{1}{T-1}\sum_{i=1}^{T}(X_i - \bar{X})^2$.

The confidence interval of μ is given by $\bar{X} - eS/\sqrt{T} \leq \mu \leq \bar{X} + eS/\sqrt{T}$ where e is determined by Student's t-distribution's table for $T-1$ degrees of freedom. The probabilty associated with the confidence interval of μ is given by δ. For example, widely used in this thesis is a $\delta = 0.95$, corresponding to a 95% confidence interval with $e = 1.65$ for ∞ degrees of freedom.

B.2 Cochran test

The Cochran test (Conover [1999]) is applied when there is a single test set, and we wish to compare the test error performance of several different classifier algorithms using a single test measure. Each of the m algorithms is applied independently to each of the test examples, and the result of each application is either a 1 or 0, representing respectively a correctly classified test example or not. The results form a table with T rows and m columns, T being the total number of test examples. For each algorithm i and test example j, an entry of $X_{ij} \in \{1, 0\}$ is made in Table B.1. Also, Table B.1 consists of row totals, $R_i = \sum_{j=1}^{m} X_{ij}$, and column totals, $C_j = \sum_{i=1}^{T} X_{ij}$.

From Table B.1 the test statistic for the Cochran test is determined by

$$K = m(m-1)\frac{\sum_{j=1}^{m}\left(C_j - \frac{\sum_{i=1}^{T} R_i}{m}\right)^2}{\sum_{i=1}^{T} R_i\,(m - R_i)}. \tag{B.2}$$

Table B.1: Cochran test table

	Algorithms				
Test Examples	1	2		m	Row Totals
1	X_{11}	X_{12}	...	X_{1m}	R_1
2	X_{21}	X_{22}	...	X_{2m}	R_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T	X_{T1}	X_{T2}	...	X_{Tm}	R_m
Column Totals	C_1	C_2	...	C_m	$\sum_{i=1}^{T} R_i$

We now present the hypothesis test used in this thesis, where H_0 is the hypothesis that each algorithm is equally effective, and H_1 is the hypothesis that there is a difference between the performance of the algorithms. To reject the hypothesis H_0 at the level δ, K must be greater than the $1 - \delta$ quantile of the chi-squared distribution with $m - 1$ degrees of freedom, for example for a critical region of $\delta = 0.05$ and $m = 3$, K must be greater than 5.99 to reject H_0.

Several Blind Equalization Methods

C.1 Constant Modulus Algorithm

We provide details on the Constant Modulus Algorithm (CMA) attributed to Godard [1980]. CMA is a stochastic gradient method which attempts to minimise the dispersion, $D^{(2)}$. The dispersion measures the squared residual between absolute prediction produced by the CMA equalizer $\|\hat{y}\|$ and the second order dispersion constant R_2, i.e. $D^{(2)} = \mathrm{E}(|\hat{y}|^2 - R_2)^2/2$. Noting that in the BPSK (binary case) $R_2 = 1$, and that only the dispersion of the second order is considered here. The stochastic gradient update for CMA is

$$\mathbf{w}_{\mathrm{CMA},t+1} = \mathbf{w}_{\mathrm{CMA},t} + \eta \hat{y}_t (R_2 - |\hat{y}_t|^2) \tag{C.1}$$

where $\eta \leq 1$ is the step size, $\mathbf{w}_{\mathrm{CMA}} \in \mathbb{R}^{K_1+1}$ is the CMA equalizer's weight, $\hat{y}_t = (\mathbf{w}_{\mathrm{CMA},t} \cdot \mathbf{x}_t)$ is the equalizer's prediction for instance vector consisting of received symbols r_t as its dimensions, i.e. $\mathbf{x}_t = [r_{t+K_1}, r_{t+K_1-1}, \ldots, r_t]'$.

C.2 Decision Directed Decision Feedback Equalizer

In decision directed mode there is no known training sequence instead the symbol decision (label decision in the binary case) are used to train the equalizer. As the name suggests the Decision Feedback Equalizer (DFE) is the equalizer which has been shown to have smaller final *mean squared error* (MSE) performance compared with linear equalizers, like CMA of the previous Section C.1 (Proakis [2001]). One disadvantage is that DFE requires some initialisation otherwise it may become unstable or not converge to a good solution.

The weight update for DD-DFE for the LMS update is

$$\mathbf{w}_{\mathrm{DD-DFE},t+1} = \mathbf{w}_{\mathrm{DD-DFE},t} + \eta(\mathrm{sgn}(\hat{y}_t) - \hat{y}_t)\mathbf{x}_t, \tag{C.2}$$

where $\eta < 1$ is the step size and $\hat{y}_t = (\mathbf{w}_{\mathrm{DD-DFE},t} \cdot \mathbf{x}_t)$.

The instance vector $\mathbf{x}_t$ used in (C.2) is different to that used in Chapter 5 for linear equalizers. The instance vector consists of two parts: a $K_1 + 1$ sequence received symbols $r_{t+K_1}, r_{t+K_1-1}, \ldots, r_t$; and K_2 past symbol predictions $\hat{y}_{t-1}, \hat{y}_{t-2}, \ldots, \hat{y}_{t-K_2}$,

i.e. $\mathbf{x}_t = [r_{t+K_1}, r_{t+K_1-1}, \ldots, r_t, \hat{y}_{t-1}, \hat{y}_{t-2}, \ldots, \hat{y}_{t-K_2}]'$.

Bibliography

H. Akaike. A New Look at Statistical Model Identification. *IEEE Trans. Automat. Contr.*, AC-19:716–723, 1974.

S. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2): 251–276, 1998.

W. P. Ang and B. Farhang-Boroujeny. A New Class of Gradient Adaptive Step-Size LMS Algorithms. *IEEE Trans. Signal Processing*, 49(4):805–810, 2001.

E. Bauer and R. Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36:105–139, 1999.

A. Benveniste, M. Metivier, and P. Prioret. *Adaptive Algorithms and Stochastic Approximations*. New York: Springer-Verlag, 1990.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

H. D. Block. The Perceptron: A Model for Brain Functioning. *Reviews of Modern Physics*, 34:123–135, 1962.

L. Bottou and Y. Le Cun. On-Line Learning for Very Large Datasets. Technical report, NEC Labs America, 2003.

L. Bottou and Y. Le Cun. Large Scale Online Learning. In *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press, 2004. To appear.

O. Bousquet and M. K. Warmuth. Tracking a Small Set of Experts by Mixing Past Posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

J. S. Breese, D. Heckerman D., and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52. Madison, WI: Morgan Kaufmann, 1998.

L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):120–140, 1996.

L. Breiman. Arcing Classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

R. A. Casas, C. R. Johnson Jr., R. A. Kennedy, Z. Ding, and R. Malamut. Blind Adaptive Decision Directed Feedback Equalization: a Class of Channels Resulting in Ill Convergence from a Zero Initialization. *Intl. J. on Adaptive Control and Signal Processing*, 12(2):173–268, 1998.

S. Chen and C. J. Harris. Design of the Optimal Separating Hyperplane for Decision Feedback Equalizer Using Support Vector Machines. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP00)*, volume V, pages 2701–2704, June 2000.

S. Chen, B. Mulgrew, and P. M. Grant. A clustering technique for digital communications channel equalization using radial basis function networks. *IEEE Trans. Neural Networks*, 4(4):570–578, 1993a.

S. Chen, B. Mulgrew, and L. Hanzo. Asymptotic Bayesian Decision Feedback Equalizer using a Set of Hyperplanes. *IEEE Trans. Signal. Process.*, 48(12):3493–3500, 1993b.

W.J. Conover. *Practical Nonparametric Statistics*. John Wiley, 3 edition, 1999.

C. Cortes and V. N. Vapnik. Support Vector Network. *Machine Learning*, 20:273–297, 1995.

K. Crammer, J. Kandola, and Y. Singer. Online Classification on a Budget. In *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press, 2004. To appear.

K. Crammer and Y. Singer. Pranking with Ranking. In *Advances in Neural Information Processing Systems 15*, pages 641–647. Cambridge, MA: MIT Press, 2002.

K. Crammer and Y. Singer. Ultraconservative Online Algorithms for Multiclass Problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

P. Derbeko, R. El-Yaniv, and R. Meir. Online Passive-Aggressive Algorithms. In *ECML*, pages 60–71. Springer-Verlag, 2002.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2nd edition, 2000.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. San Francisco: Chapman & Hall, 1993.

A. Fern and R. Givan. Online Ensemble Learning: An Empirical Study. In *Proceedings of the Seventeenth International Conference of Machine Learning (ICML-2000)*, pages 279–286. Morgan Kaufmann, 2000.

Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

Y. Freund and R. Schapire. Game Theory, On-line Prediction and Boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

Y. Freund and R. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, 1999.

S. I. Gallant. Optimal Linear Discriminants. In *Eighth International Conference on Pattern Recognition*, pages 849–852. IEEE, 1986.

W. A. Gardner. Learning Characteristics of Stochastic-Gradient Descent Algorithms: A General Study, Analysis, and Critique. *Signal Processing*, 6:113–133, 1984.

C. Gentile. A New Approximate Maximal Margin Classification Algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

D. N. Godard. Self Recovering and Carrier Tracking in Two-dimensional Data Communications. *IEEE Trans. Commun.*, COMM-28(11):1867–1875, 1980.

R. B. Gramacy, M. K. Warmuth, S. A. Brandt, and I. Ari. Adaptive Caching by Refetching. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 16*, pages 1465–1472. Cambridge, MA: MIT Press, 2003.

E. Harrington. Expert Mixture Methods for Adaptive Channel Equalization. In O. Kaynak, E. Alpaydin, E. Oja, and L. Xi, editors, *Proceedings of ICANN/ICONIP, LNCS 2714*, pages 538–545. Berlin Heidelberg: Springer Verlag, 2003.

D. Haussler, J. Kivinen, and M. K. Warmuth. Sequential Prediction of Individual Sequences Under General Loss functions. *IEEE Trans. on Information Theory*, 44 (5):1906–1925, 1998.

S. Haykin and B. Widrow. *Least-Mean-Square Adaptive Filters*. Wiley, 2003.

R. Herbrich. *Learning Kernel Classifiers*. Cambridge MA: MIT Press, 2002.

R. Herbrich, T. Graepel, and C. Campbell. Bayes Point Machines. *Journal of Machine Learning Research*, 1:245–279, 2001.

R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, 2000.

M. Herbster and M. K. Warmuth. Tracking the Best Expert. *Machine Learning*, 32(2): 151–178, 1998.

M. Herbster and M. K. Warmuth. Tracking the Best Linear Predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.

J. Kivinen, A. Smola, and R. C. Williamson. Online Learning with Kernels. In *Advances in Neural Information Processing Systems 15*, pages 785–792. Cambridge, MA: MIT Press, 2002.

J. Kivinen and M. K. Warmuth. Additive Versus Exponentiated Gradient Updates for Linear Prediction. *Information and Computation*, 132(1):1–64, 1997.

J. Kivinen and M. K. Warmuth. Averaging Expert Predictions. In *In P.Fischer and H.U.Simon, editors, Proc. 4th European Conference on Computational Learning Theory*, pages 153–167. Springer LNAI 1572, Berlin, 1999.

A. Kowalczyk. Maximal margin perceptron. In *Advances in Large Margin Classifiers*, pages 75–115. MIT Press, 2000.

W. Krauth and M. Mézard. Learning Algorithms with Optimal Stability in Neural Networks. *J. Phys. A: Math. Gen.*, 20:745–752, 1987.

Y. Li and P. Long. The Relaxed Online Maximum Margin Algorithm. *Machine Learning*, 46(1–3):361–387, 2002.

N. Littlestone and M. K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261, 1994.

R. K. Martin, W. A. Sethares, R. C. Williamson, and C. R. Johnson. Exploiting Sparsity in Adaptive Filters. *IEEE Trans. Signal Processing*, 50:1883–1894, Aug. 2002.

V. J. Mathews and Z. Xie. Stochastic Gradient Adaptive Filters with Gradient Adaptive Step Size. *IEEE Trans. Signal Processing*, 41:2075–2087, June 1993.

J. Mazo. Analysis of Decision-Directed Equalizer Convergence. *Bell System Tech. Journal*, 59(10):1857–1876, 1980.

N. McGinty. Strategy to Transistion from the Constant Modulus Algorithm to a Decision Feedback Blind Equalizer. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP02)*, volume III, pages 2661–2664, May 2002.

R. Meir and G. Rätsch. An introduction to boosting and leveraging. In S. Mendelson and A. Smola, editors, *Advanced Lectures on Machine Learning LNCS 2600*, pages 119–184. Springer-Verlag, 2003.

T. P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, January 2001.

C. Monteleoni and T. Jaakkola. Online Learning of Non-stationary Sequences. In *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press, 2004. To appear.

A.B.J. Novikoff. On Convergence Proofs for Perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

N. C. Oza. *Online Ensemble Learning*. PhD thesis, University of California, Berkeley, Fall 2001.

J.-M. Park and Y.-H. Hu. On-Line Learning for Active Pattern Recognition. *IEEE Signal Processing Letters*, pages 301–303, Nov 1996.

F. Pérez-Cruz and A. Artés-Rodríguez. Adaptive SVC for Nonlinear Channel Equalization. In *Proceedings of EUSIPCO2002, vol. II*, pages 45–48, 2002.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Machines*, pages 185–208. MIT Press, 1998.

J. G. Proakis. *Digital Communications*. New York: McGraw-Hill, 4th edition, 2001.

P. Resnick and H. Varian. Recommender Systems. *Communications of the ACM*, 40 (3):56–58, 1997.

F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Orginsation in the Brain. *Psychological Review*, 65:386–407, 1958.

P. Ruján. Playing Billards in Version Space. *Neural Computation*, 9:99–122, 1997.

I. Santamaría, R. Gonzalez, C. Pantaleon, and J. C. Principe. Maximum Margin Equalizers Trained with the Adatron Algorithm. *Signal Processing*, 83:593–602, 2003.

Y. Sato. A Method of Self-Recovering Equalization for Multilevel Amplitude Modulation Systems. *IEEE Trans. Commun.*, COM-23:679–682, 1975.

E. Satorius and J. Pack. Application of Least Squares Lattice Algorithms to Adaptive Equalization. *IEEE Trans. on Communications*, vol. COM-29:136–142, 1981.

B. Schölkopf, R. Herbrich, and A. J. Smola. A Generalized Representer Theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

B. Schölkopf, S. Mika, C. J. C., Burges, K. Muller P. Knirsch, G. Rätsch, and A. Smola. Input Space vs. Feature Space in Kernel-Methods. *IEEE Trans. on Neural Networks*, 10(5):1000–1017, 1999.

D. J. Sebald and J. A. Bucklew. Support Vector Machines Techniques for Nonlinear Equalization. *IEEE Trans. Signal Process.*, 48(11):3217–3226, 2000.

J. Shao and D. Tu. *The Jackknife and Bootstrap*. New York: Springer-Verlag, 1995.

A. Shashua and A. Levin. Taxonomy of Large Margin Principle Algorithms for Ordinal Regression Problems. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 16*, pages 937–944. Cambridge, MA: MIT Press, 2003.

A. C. Singer and M. Feder. Universal Linear Prediction by Model Order Weighting. *IEEE Trans. Signal Processing*, 47(10):2685–2699, Oct. 1999.

B. Sklar. Rayleigh Fading Channels in Mobile Digital Communications Systems Part I: Characterization. *IEEE Communications Magazine*, pages 90–100, July 1997.

S. Sonnenburg. New Methods for Splice Site Recognition. Master's thesis, Humbold University, 2002.

J. R. Treichler, M. G. Larimore, and J. C. Harp. Practical Blind Demodulators for High-Order QAM Signals. *IEEE Proc.*, 86(10):1907–1926, 1998.

G. Valentini and T. G. Dietterich. Low Bias Bagged Support Vector Machines. In *Proceedings of the Twentieth International Conference of Machine Learning (ICML-2003)*, pages 752–759. Morgan Kaufmann, 2003.

V. N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, New York, 1995.

V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2): 264–280, 1971.

V. Vovk. A Game of Prediction with Expert Advice. *Journal of Computer and System Sciences*, 56:153–173, 1998.

T. Watkin. Optimal Learning with a Neural Network. *Europhysics Letters*, 21(8): 871–876, 1993.

T. Watkin and A. Rau. Selecting Examples for Perceptrons. *J. Phys. A: Math. Gen.*, 25:113–121, 1992.

B. Widrow and M. E. Hoff. Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, 1960.

Glossary of Symbols

The following are the symbols used in this thesis and their corresponding page of first appearance. Symbols in bold indicate vectors or sequences which contain vectors.