Online Bayes Point Machines

Edward Harrington[1], Ralf Herbrich[2], Jyrki Kivinen[1], John C. Platt[3], and Robert C. Williamson[1]

[1] Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT 0200
{edward.harrington,jyrki.kivinen,bob.williamson}@anu.edu.au
[2] Microsoft Research
7 J J Thomson Avenue,
Cambridge, CB3 0FB, UK
rherb@microsoft.com
[3] Microsoft Research
1 Microsoft Way,
Redmond, WA 98052, USA
jplatt@microsoft.com

Abstract. We present a new and simple algorithm for learning large margin classifiers that works in a truly online manner. The algorithm generates a linear classifier by averaging the weights associated with several perceptron-like algorithms run in parallel in order to approximate the Bayes point. A random subsample of the incoming data stream is used to ensure diversity in the perceptron solutions. We experimentally study the algorithm's performance on online and batch learning settings. The online experiments showed that our algorithm produces a low prediction error on the training sequence and tracks the presence of concept drift. On the batch problems its performance is comparable to the maximum margin algorithm which explicitly maximises the margin.

1 Introduction

An online classifier tries to give the best prediction based on the example sequence seen at time t in contrast to a batch classifier which waits for the whole sequence of T examples. There have been a number of recent attempts [1–3] in the online setting to achieve a approximation to the maximum margin solution of batch learners such as SVMs. In this paper we present a truly online algorithm for linear classifiers which achieves a large margin by estimating the centre-of-mass (the so-called Bayes point) by a randomisation trick.

It is well accepted that a (fixed) large margin classifier provides a degree of immunity to attribute noise and concept drift. One advantage of the algorithm presented here is the ability to also track concept drift.

The primary appeal of the algorithm 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 illustrate the effectiveness of the algorithm to track concept drift with an experiment based on the MNIST OCR database.

We define a sequence of training examples seen by the online learning algorithm in the form of a sequence of T training examples, $\boldsymbol{z} = (z_1, \ldots, z_T) := ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$, comprising an instance $\mathbf{x}$ in the instance space $\mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \{-1, +1\}$ the corresponding binary label. The weight vector $\mathbf{w} \in \mathbb{R}^d$ defines the d-dimensional hyperplane[4] $\{\mathbf{x} \colon (\mathbf{w} \cdot \mathbf{x}) = 0\}$ which is the decision boundary associated with the hypothesis $\operatorname{sgn}(\mathbf{w} \cdot \mathbf{x})$ of the perceptron algorithm [4], where $\operatorname{sgn}(v) = 1$ if $v \geq 0$ and $\operatorname{sgn}(v) = -1$ if $v < 0$. We often assume that the target labels y_i are defined by $y_i = \operatorname{sgn}(\mathbf{u} \cdot \mathbf{x}_i)$; in other words, there exists a linear classifier which can correctly classify the data. The *margin* $\gamma_{\boldsymbol{z}}(\mathbf{w})$ plays a key role in understanding the performance and behaviour of linear classifier learning algorithms. It 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 z_i is $\gamma_{z_i}(\mathbf{w}) := y_i (\mathbf{w} \cdot \mathbf{x}_i)/\|\mathbf{w}\|_2$, and the margin with respect to the whole training sequence is $\gamma_{\boldsymbol{z}}(\mathbf{w}) := \min\{\gamma_{z_i}(\mathbf{w}) \colon i \in \{1, \ldots, T\}\}$. The *version space* $V(\boldsymbol{z})$ is the set of hypotheses consistent with the training sample. Since we only consider linear classifiers, we consider $V(\boldsymbol{z})$ as a subset of the weight space:

$$V(\boldsymbol{z}) := \left\{\mathbf{w} \in \mathbb{R}^d : \operatorname{sgn}(\mathbf{w} \cdot x_i) = y_i \text{ for all } (\mathbf{x}_i, y_i) \in \boldsymbol{z}\right\}. \tag{1}$$

Clearly, the target function $\mathbf{u}$ is in $V(\boldsymbol{z})$ for any $\boldsymbol{z}$. A learning algorithm that attains zero training error finds an hypothesis in the version space.

2 (Large Margin) Perceptron Learning Algorithms

We first consider 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 to 1. When $\rho_k = 0$, and η_k is fixed it is known by a result of Novikoff [5] that the algorithm makes no more than $(\beta/\gamma_{\boldsymbol{z}})^2$ mistakes where $\beta := \max_{i=1,\ldots,T} \|\mathbf{x}_i\|_2$ (we assume that the instances throughout the paper are normalised so $\beta = 1$) and $\gamma_{\boldsymbol{z}} := \max\{\gamma_{\boldsymbol{z}}(\mathbf{w}) : \mathbf{w} \in \mathbb{R}^d\}$ is the margin attained by the maximum margin hyperplane. This perceptron convergence theorem only guarantees that the perceptron's hypothesis $\mathbf{w}$ is in $V(\boldsymbol{z})$; *i.e.* only that $\gamma_{\boldsymbol{z}}(\mathbf{w}_{\text{perc}}) > 0$ where $\mathbf{w}_{\text{perc}}$ is the weight vector the perceptron algorithm converges to. In the marginalised perceptron [4], ρ_k is a positive constant and the final margin, if run until separation of the training sequence in the batch setting, can be guaranteed to be $\geq \rho_k$. The perceptron algorithm has several key advantages: it is extremely simple, on-line, and it can be readily kernelised since it only uses inner products between input vectors [6].

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

Algorithm 1 Generalised Perceptron algorithm

```
Require: A linear separable training sequence $\boldsymbol{z} = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots)$.
Require: Margin parameter sequence $\boldsymbol{\rho}_k$ and learning rate $\boldsymbol{\eta}_k$.
1: $t = 1,\ k = 1.$
2: repeat
3:   if $y_t\left(\mathbf{w}_k \cdot \mathbf{x}_t\right)/\|\mathbf{w}_k\|_2 \le \rho_k$ then
4:     $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta_k y_t \mathbf{x}_t$
5:     $k = k + 1$
6:   end if
7: until No more updates made
```

The motivation for setting $\rho_k = \rho > 0$ in order to guarantee $\gamma_{\boldsymbol{z}}(\mathbf{w}) > \rho$ for the final solution $\mathbf{w}$ is that a larger margin hyperplane has a better guarantee of generalisation performance than a smaller margin one [6]. It should be noted that the maximum margin hyperplane is not unique in this regard; indeed, generically it is not the hyperplane with the best generalisation performance (the so-called Bayes point) [7], but historically much effort has been expended on maximising the margin.

Whilst such analyses and guarantees are proven 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 develop further online algorithms that achieve an approximation to the maximum margin. Kivinen *et al.* [3] studied the marginalised perceptron (and issues arising when it is kernelised). Li and Long [2] studied an algorithm they called ROMMA where if there is a mistake at the tth trial then $\mathbf{w}_{t+1}$ is the smallest element of the constrained set of $\left\{\mathbf{w} : \mathbf{w}_t \cdot \mathbf{w} \ge \|\mathbf{w}_t\|_2^2\right\} \cap \{\mathbf{w} : y_t(\mathbf{w} \cdot \mathbf{x}_t \ge 1)\}$, else $\mathbf{w}_{t+1} = \mathbf{w}_t$. It has a similar mistake bound to the perceptron and is computationally slightly more costly than the perceptron. Gentile [1] has presented an Approximately Large Margin Algorithm (ALMA)[5] 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. Thus, 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 $\|\mathbf{w}_k\|_2 \le 1$.

3 Online Bayes Point Machines

Whilst the variants on the classical perceptron discussed above can guarantee convergence to a hyperplane with a large margin, there is a price to pay. The

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

Algorithm 2 OBPM algorithm

```
Require: A training sample $\boldsymbol{z} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T))$.
Require: A 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 vari-
  ables 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: Initialise weights $\mathbf{w}_{j,1} = \mathbf{0}$, for all $j \in \{1, \ldots, N\}$.
 3: for $t = 1$ to $T$ do
 4:   $\tilde{\mathbf{w}}'_t = \mathbf{0}$
 5:   for $j = 1$ to $N$ do
 6:     $b_{j,t} = \text{Bernoulli}(\tau)$
 7:     if $b_{j,t} = 1$ then
 8:       $\mathbf{w}_{j,t+1} = \mathcal{L}(\mathbf{w}_{j,t}, \eta_{j,t}, \mathbf{x}_t, y_t)$
 9:     else
10:       $\mathbf{w}_{j,t+1} = \mathbf{w}_{j,t}$
11:     end if
12:     $\eta_{j,t+1} = \mathcal{S}(t, \eta_{j,t}, \ldots)$
13:     $\tilde{\mathbf{w}}'_t = \tilde{\mathbf{w}}'_t + \mathbf{w}_{j,t+1}/N$
14:   end for
15:   $\tilde{\mathbf{w}}_t = \tilde{\mathbf{w}}'_t / \max\{1, \|\tilde{\mathbf{w}}'_t\|_2\}$
16: end for
17: return $\tilde{\mathbf{w}}_T$
```

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. This suggests it is worthwhile exploring alternate variants on classical perceptrons.

Our starting point for such variants is the observation that (1) defines a *convex* set $V(\boldsymbol{z})$. Thus if one found a number of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_N$ in $V(\boldsymbol{z})$, one could optimise over the set of convex combinations $C^N := C^N(\mathbf{w}_1, \ldots, \mathbf{w}_N) := \{\sum_i \alpha_i \mathbf{w}_i : \|\boldsymbol{\alpha}\|_1 = 1, \boldsymbol{\alpha} \geq \mathbf{0}\}$ of them. Intuitively one would expect that the more "diverse" $\mathbf{w}_1, \ldots, \mathbf{w}_N$ are, the greater the proportion of $V(\boldsymbol{z})$ could be covered by C^N. The centre of C^N provides an rough approximation of the Bayes point and thus potentially 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 boosting which form convex combinations of classifier *hypotheses* $\mathbf{x} \mapsto \text{sgn}\,(\mathbf{w}_j \cdot \mathbf{x})$. The sampling presented in this paper can be mistaken for an online version of Bagging [8] since the sample is drawn randomly with replacement from a fixed set. The difference with bagging (and boosting [9]) is that we combine the classifier weights not the hypotheses.

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 $\pi(\boldsymbol{z}) :=$

$(z_{\pi(1)}, \ldots, z_{\pi(T)})$ of the training sample $\boldsymbol{z}$ [7], where $\pi : \{1, \ldots, T\} \rightarrow \{1, \ldots, T\}$ is a permutation. Although an elegant and effective trick, it is not an online algorithm — one needs the entire training sample $\boldsymbol{z}$ before starting.

This motivates the algorithm we study in this paper: the Online Bayes Point Machine (OBPM) (Algorithm 2). Given a training sequence $\boldsymbol{z}$, we run N Perceptrons “in parallel” and ensure diversity of their final solutions by randomly choosing to present a given sample z_k to perceptron j only if $b_{jk} = 1$, where b_{jk}, $j = 1, \ldots, N$, $k = 1, 2, \ldots$ are independent Bernoulli random variables with $\Pr\{b_{jk} = 1\} = \tau$.

Although there are theoretical reasons for expecting that refined optimisation of the α_i would lead to better solutions than naively setting them all equal to $1/N$, we have found experimentally this expectation not to hold. Furthermore, such optimisation can only occur in the batch setting: one cannot determine the margin of a candidate weight vector without the whole training sample. One method tried in optimizing the choice of α_i was to maximize the following lower bound in the batch setting (see Appendix A for proof):

$$\gamma_{\mathbf{z}}(\tilde{\mathbf{w}}) \geq \frac{\sum_{j=1}^{N} \alpha_j \gamma_{\mathbf{z}}(\mathbf{w}_j)}{\sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \left(\mathbf{w}_i \cdot \mathbf{w}_j\right)}}. \tag{2}$$

It was found by experimentation that maximizing the bound of (2) had no performance gain compared to setting $\alpha_i = 1/N$. Hence in the rest of the paper we will only consider the situation where $\alpha_i = 1/N$ for all i. Step 15 of algorithm 2 is optional as it only bounds $||\tilde{\mathbf{w}}||$ by one and given the instances are normalized, this results in $\rho \leq 1$ in the generalised Perceptron algorithm 1.

On average each perceptron j sees only a fraction τ of all the examples in $\boldsymbol{z}$. The smaller τ, the more diverse the solutions $\mathbf{w}_j$ become, although the algorithm takes longer to reach these solutions.

In this paper we take $\mathcal{L}$ the base learner to be the perceptron update rule and fix $\eta_{j,t} = 1$. This leaves OBPM two parameters required for tuning: τ and N.

We see that the number of arithmetic operations for OBPM on average is $O(\tau N d T)$, for both ALMA and the perceptron it is $O(dT)$. OBPM is a factor τN more expensive computationally than ALMA. A point worth noting is that each perceptron used in OBPM is independent and so OBPM can be readily implemented using parallel computing.

In the case of OBPM, the implementation may be made more efficient by noting that the kernel values $K(\mathbf{x}, \mathbf{x}_t)$ are the same for all N perceptrons. For a truly online algorithm we need to bound the number of instances $\mathbf{x}_t$ needed to be stored for the kernel expansion [3].

4 Experiments

We used a toy problem to analyse the effect of τ and N on approximating the maximum margin. We show that for this example we can do better than other

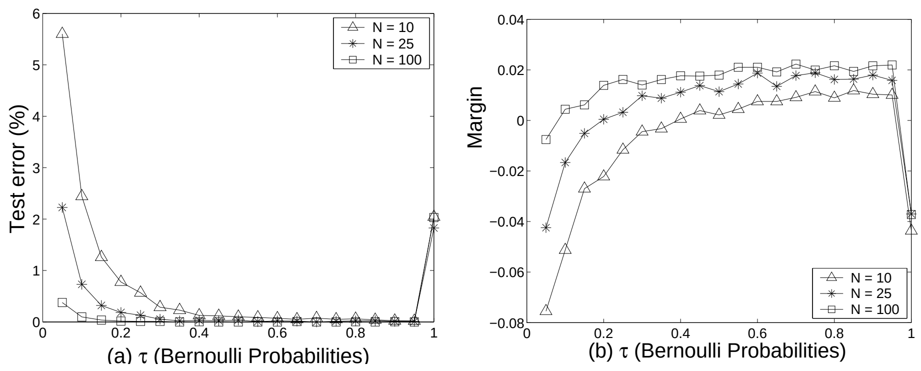

Fig. 1. Artificial data set averages from 30 Monte Carlo simulations training OBPM with no label noise ($e = 0.0$) for a single epoch: (a) test error versus τ, (b) margin $\gamma_{\boldsymbol{z}}(\tilde{\mathbf{w}}_T)$ versus τ.

online methods but there is a trade-off with computational cost. We demonstrate that OBPM has the ability to handle concept drift by applying it to a problem generated using the MNIST OCR data set. Last, we show the performance of OBPM on some real-world data sets using the online prediction error analysis and batch 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, B was adjusted from 0.5 up to 20 in non-uniform steps, and $C \in \{\sqrt{2}/4, \sqrt{2}/2, \sqrt{2}, 2\sqrt{2}\}$[6]. Thoughout the experiments the final reported parameters were chosen based on the best training error for that experiment. The SVM results were produced with a linear kernel and soft margin using an exact optimisation method. Training and test errors and margins for the algorithms were measured using the last weights as the hypothesis. The results were produced using C code on a 1GHz DEC alpha.

4.1 Parameter Selection

A target $\mathbf{u} \in \{-1, 0, +1\}^{100}$ and T instances $\mathbf{x}_t$ were generated randomly with a margin, $\gamma_{\boldsymbol{z}}(\mathbf{u}) \geq m = 0.05$. The associated label y_t was allocated $+1$ if $(\mathbf{u} \cdot \mathbf{x}_t)/\|\mathbf{u}\|_2 \geq m$ and -1 if $(\mathbf{u} \cdot \mathbf{x}_t)/\|\mathbf{u}\|_2 \leq -m$. This process ensured that the training sample has a margin greater than m; we generated the test sample identically to the training sample. This problem is very similar to the artificial data experiment of Gentile [1]. We use a different data set size of T, and we use the same margin in the training and test samples. We reduced T from 10000 to 1000, m from 1 to 0.05. This gives a mistake bound of 400 for the classic

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

Table 1. Table of test error averages from 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 ±0.32	0.07±0.04(B=1.11,C=0.71)	0.00±0.00($\tau = 0.35$,N=100)
0.01	3.35±0.44	0.14±0.07(B=0.83,C=0.71)	0.10±0.06($\tau = 0.50$,N=100)
0.1	12.96±1.0	0.76±0.13(B=1.25,C=0.71)	0.96±0.13($\tau = 0.15$,N=100)

Perceptron, making it highly unlikely that the perceptron could learn this 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. It is indicated in Figure 1 (b) that by increasing N, the margin was increased. 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 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 1 (b) OBPM achieved a margin of 0.02, whereas ALMA achieved 0.006. Table 1 shows with label noise that the performance of ALMA and OBPM were similar, except in the noise-free case. We also noted that ALMA's test error performance was more sensitive to the choice of parameter settings compared to OBPM.

4.2 Drifting concept

In order to demonstrate OBPM's ability to learn a drifting concept we designed a drifting experiment using the well known MNIST OCR data set[7]. To simulate the drift we set the positive class to a single label which varied over time. The negative class 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, therefore the transition boundaries in Figure 2 are not obvious. The following psuedo code shows how we allocated labels $l = 1, \ldots, 3$ to the positive class of $z_t = z_t^+$ (i.e. $y = 1$) according to the uniform random variable $r_t \in [0, 1]$ at time t, where $z_t^l = (\mathbf{x}_t^l, y^l = 1)$:

```
$z_t^+ = z_t^1$
if $r_t > (1 - \frac{t}{T/2})$ then $z_t^+ = z_t^2$
if $r_t > (1 - \frac{t}{1.11T})$ then $z_t^+ = z_t^3$
```

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

Selected the labels this way we ensure by $t > T/2$ all the positive class examples are with label 2 and by 900 they are all label 3. Replacing the labels randomly in the above way also gives a smoother and more gradual transition between labels.

Figure 2 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 [12]), which is a regression algorithm where we take the sign of the predictions to convert to the classification setting. One reason why ALMA did not perform as well as OBPM is that ALMA assuming a stationary target in order to adjust the learning rate η_t and margin parameter ρ_k. Note that this is not an exhaustive study on drifting concepts but it appears the large margin of OBPM allows for the improvement over the perceptron (the generalised Perceptron $\rho = 0$).

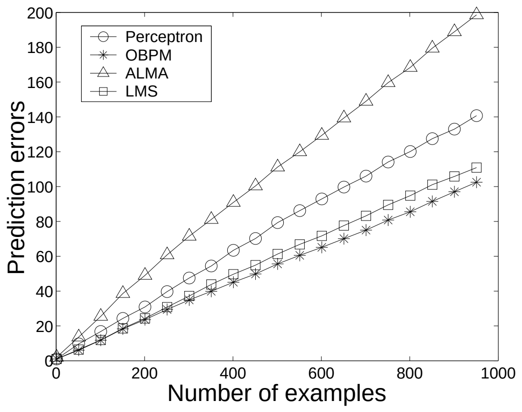

Fig. 2. Prediction errors from drifting concept experiment with the MNIST OCR data set comparing LMS (step size 0.5), ALMA (B=2, C=$\sqrt{2}$), Perceptron ($\rho = 0$) and OBPM (τ=0.4, N=100).

4.3 Real-world data sets

We consider three different data sets. The first data set was derived from a genome of the nematode Ceanorhabditis Elegans (C. Elegans). The classification task was to locate the specific splice sites (for more details see [10]). 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 the Web data sets [8]. Each input consists of 14 mixed categorical and

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

continuous attributes. We increase 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 Adult 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 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 and a test set size of 21 489. To study the

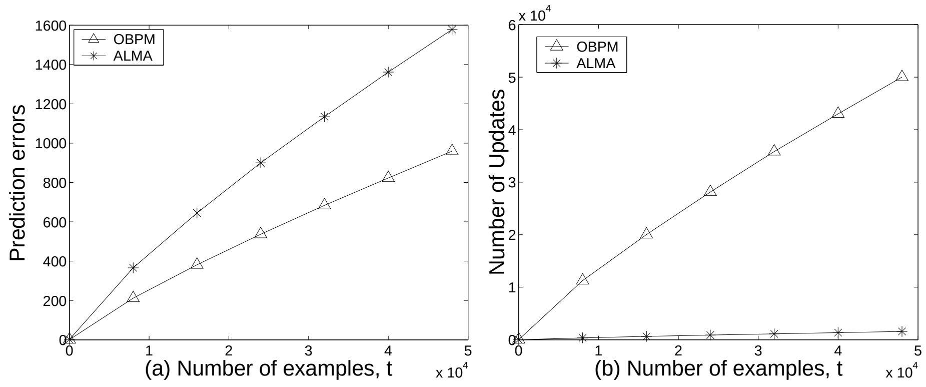

Fig. 3. (a) Prediction errors for OBPMs $\tilde{\mathbf{w}}_t$ (τ=0.3, N=100) and ALMAs weight solutions and (b) total number updates for the Web data after training example t.

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 3(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 3(b): the total number of updates made by OBPM is larger than the number made by ALMA.

The batch setting results benchmarking OBPM with other algorithms are given in Table 2. The UCI data results of Table 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. The maximum margin benchmark, the soft margin linear SVM 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 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. A general comment of the results of Table 2 was that in the batch setting OBPM performed better than the standard perceptron whereas ALMA and OBPM were comparable. For these experiments it was not obvious how to tune ALMA's parameters easily, whereas with OBPM good results were achieved by simply adjusting τ over a small range 0.1 to 0.4.

The time taken to process the C. Elegans results 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[9] optimisation took approximately 300 hours of real time. Of course, on a parallel machine, OBPM could be sped up further.

Table 2. Real world data set test errors in percent run for 3 epochs.

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

5 Conclusions

We have presented OBPM, which is a simple meta-algorithm for the online training of linear classifiers with large margin. OBPM trains N linear classifiers in parallel on random subsets of the data. We have shown experimentally that compared to online algorithms such as the standard perceptron and ALMA, OBPM was able to achieve a lower prediction error and track concept drift. We were able to demonstrate that OBPM is a truly online algorithm with a large margin which is simple to implement with the potential for a parallel architecture. OBPM's CPU times were not prohibitive and its performance compared favourably with SVM and ALMA when trained in the batch setting on three real datasets.

Acknowledgements This work was supported by the Australian Research Council. We thank Gunnar Rätsch, Alex Smola and Sören Sonnenburg for their

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

help with the C. Elegans data and for providing their SVM results for C. Elegans. Edward's research was funded by the Defence Science and Technology Organisation, Australia.

A Lower bound proof of (2)

Proof. 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

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

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

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

References

1. C. Gentile, (2001) A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, **2**:213-242.
2. Y. Li. & P. Long, (2002) The relaxed online maximum margin algorithm. *Machine Learning*, **46(1-3)**:361-387.
3. J. Kivinen, A. Smola, and R. C. Williamson, (2002) Online Learning with kernels. *Advances in Neural Information Processing Systems 14*, Cambridge, MA: MIT Press (pp. 785-793).
4. R.O. Duda & P.E. Hart & D.G. Stork, (2000) *Pattern Classification And Scene Analysis 2nd Edition.* John Wiley.
5. A.B.J. Novikoff, (1962) On convergence proofs on perceptrons. *In Proceedings of the Symposium on Mathematical Theory of Automata*, vol. XII, (pp. 615-622).

6. R. Herbrich, (2002) *Learning Kernel Classifiers*, Cambridge, MA: MIT Press.
7. R. Herbrich, T. Graepel & C. Campbell, (2001) Bayes Point Machines. *Journal of Machine Learning Research*, **1**:245-279.
8. L. Breiman, (1996) Bagging predictors. *Machine Learning*, **24(2)**:120-140.
9. R.E. Schapire, (1990) The strength of weak learnability. *Machine Learning*, **5**:197-227.
10. S. Sonnenburg, (2002) New Methods for Splice Site Recognition. Master's thesis, Humbold University.
11. W.J. Conover, (1980) *Practical nonparametric statistics, 2nd Edition.* John Wiley.
12. B. Widrow and M. E. Hoff, (1960) Adaptive switching circuits. *1960 IRE WESCON Convention Record*, pt. 4, pp. 96-104.