Online Ranking/Collaborative filtering using the Perceptron Algorithm

Edward F. Harrington EDWARD.HARRINGTON@DSTO.DEFENCE.GOV.AU
Intelligence, Surveillance and Reconnaissance Division, Defence Science and Technology Organisation,
Lock Bag 5076, Kingston, Canberra, ACT 2604, Australia

Abstract

In this paper we present a simple to implement truly online large margin version of the Perceptron ranking (PRank) algorithm, called the OAP-BPM (Online Aggregate Prank-Bayes Point Machine) algorithm, which finds a rule that correctly ranks a given training sequence of instance and target rank pairs. PRank maintains a weight vector and a *set* of thresholds to define a ranking rule that maps each instance to its respective rank. The OAP-BPM algorithm is an extension of this algorithm by approximating the Bayes point, thus giving a good generalization performance. The Bayes point is approximated by averaging the weights and thresholds associated with several PRank algorithms run in parallel. In order to ensure diversity amongst the solutions of the PRank algorithms we randomly subsample the stream of incoming training examples. We also introduce two new online versions of Bagging and the voted Perceptron using the same randomization trick as OAP-BPM, hence are referred to as OAP with extension -Bagg and -VP respectively. A rank learning experiment was conducted on a synthetic data set and collaborative filtering experiments on a number of real world data sets were conducted, showing that OAP-BPM has a better performance compared to PRank and a pure online regression algorithm, albeit with a higher computational cost, though is not too prohibitive.

1. Introduction

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 & Varia, 1997; Breese, Heckerman, & Kadie 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.

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

A recent rank learning algorithm motivated by the Perceptron is the PRank algorithm (Crammer & Singer, 2001). In contrast to other algorithms (see Herbrich, 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. 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 (see, e.g. the variants of the marginalized Perceptron (Duda, Hart, & Stork, 2000), Maximal margin perceptron (Kowalczyk, 2000), ROMMA (Li & Long, 2002), ALMA (Gentile, 2001), and MIRA (Crammer & Singer, 2003)). For example, ALMA guarantees a margin of $(1-\alpha)\gamma$ after $O(\frac{1}{\alpha^2\gamma^2})$ updates[1], where γ 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 generalization error. However, Herbrich, Graepel, and Obermayer (2000) have shown that this carries over into the ranking learning task. Recently, Shashua and Levin (2002) showed two different Support Vector Machine (SVM) algorithms which maximized the margin resulting in a ranking learning algorithm with better generalization performance compared to PRank. Having a rank learning solution with a large margin is desirable whilst 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 (2001). Section 3 presents a large margin version of PRank, the Online Aggregate Prank-Bayes Point Machine (OAP-BPM). Online variants of the batch voting methods of Bagging (Breiman, 1996) and the voted Perceptron (Freund & Schapire, 1999) are also presented. Experimental results are given in Section 4 for a number of real world collaborative filtering data sets and a synthetic ranking data set.

2. PRank in a nutshell

As background we provide the ranking definitions [4] relating to the PRank algorithm. We have a finite set of ranks $\mathcal{Y} = \{1, \ldots, k\}$ from which a rank $y_t \in \mathcal{Y}$ is assigned to an instance $\boldsymbol{x}_t \in \mathbb{R}^d$. We are concerned with the supervised learning setting where we have a training sequence of instance rank pairs $\boldsymbol{z} = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T))$ and the goal is to find a ranking rule $\mathrm{H} : \mathbb{R}^d \to \mathcal{Y}$.

Algorithm 1 PRank (Crammer & Singer, 2002)

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

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

The PRank algorithm is defined by rounds (iterations) of the PRank update (Algorithm 1). The ranking rule H of the PRank algorithm consists of the combination of Perceptron weights $\boldsymbol{w} \in \mathbb{R}^d$ 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 $\boldsymbol{w}$ which successfully projects all the instances in $\boldsymbol{z}$ into the k subintervals defined by the thresholds $\boldsymbol{c}$, i.e. for the rank of r the subinterval is $c(r-1) < \boldsymbol{w} \cdot \boldsymbol{x} < c(r)$. This ranking procedure of the respective instances $\boldsymbol{x}_t$ is given by Algorithm 1. At round t of PRank the first step is to predict the rank y_t (line 1 of Algorithm 1) for a given instance $\boldsymbol{x}_t$ by selecting the smallest rank r such that $\boldsymbol{w}_t \cdot \boldsymbol{x}_t < c(r)$. 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 and $l_t(r) = -1$ to those below[2] (lines 3, 4 and 5 of Algorithm 1). For each subinterval, if the ranking rule H consisting of

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

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

$\boldsymbol{w}_t$ 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 $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + l_t(r)\boldsymbol{x}_t$. Our intuition tells us that updating $\boldsymbol{w}$ and $\boldsymbol{c}$ in this way has the effect of moving the threshold of the desired rank $c_{t+1}(r)$ and the updated predicted rank $\boldsymbol{w}_{t+1} \cdot \boldsymbol{x}_{t+1}$ closer together. This procedure is repeated for all the subintervals $r = 1, \ldots, k-1$ for round t.

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

1. The ranking order is preserved [4] 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 $(k-1)(R^2+1)/\gamma^2$, where $R = \max_t \|\boldsymbol{x}_t\|^2$ and margin $\gamma = \min_{r,t}\{(\boldsymbol{w}^* \cdot \boldsymbol{x} - c^*(r))l_t(r)\}$ if there exist a rank rule pair $\boldsymbol{w}^*$ and $\boldsymbol{c}^*$ such that $\gamma > 0$.

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).

3. Improved generalization (large margin) version

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 $\gamma > 0$. 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.

Methods which achieve the maximum margin solution on the training examples do not guarantee the same generalization performance as the Bayes point (Herbrich, 2000). The Bayes point is the single hypothesis chosen from a fixed class of classifiers $\mathcal{H}$ that achieves the minimum probability of error. Hence the Bayes point differs to the Bayes optimal classifier in that the latter may not be in $\mathcal{H}$. The Bayes optimal classifier in general is difficult to evaluate even if all the probability distributions are known, which motivates the use of the Bayes point instead.

A suitable estimate of the Bayes point for linear classifiers is to generate N diverse solutions $\boldsymbol{w}_i$, by training each classifier with a different permutation, i.e. $\pi(\boldsymbol{z}) := (z_{\pi(1)}, \ldots, z_{\pi(T)})$ and have an equally weighted sum $\tilde{\boldsymbol{w}} = \sum_{i=1}^{N} \boldsymbol{w}_i/N$ (Herbrich, Graepel & Campbell, 2001). This is an elegant trick but in the form presented requires that we see all training examples before we can permute them. Our aim is in finding an online method to estimate the Bayes point for the ranking learning problem. Our idea is as follows: 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_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 $\Pr\{b_{j,t} = 1\} = \tau$. This method should not be confused with voting methods like Bagging (Breiman, 1996) where, instead of the weights $\boldsymbol{w}_j$, the *hypotheses* $\boldsymbol{x} \mapsto \operatorname{sgn}(\boldsymbol{w}_j \cdot \boldsymbol{x})$ are averaged. Conceptually, this algorithm (see Algorithm 2) is similar to the OBPM algorithm presented in Harrington *et al.* (2003) for classification. In Harrington *et al.* (2003) it was shown that OBPM achieved comparable performance in producing a linear solution to the exact large margin (SVM) (using a fast SVM optimization, SVMlight) on a number of real world data sets. On the largest data set of 100 000 training examples and instance dimension of 6125, OBPM was several orders of magnitude faster than the SVM.

Algorithm 2 OAP-BPM algorithm

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

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

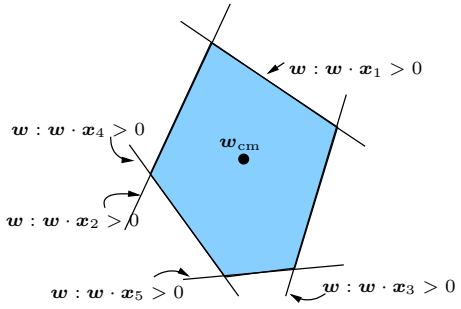

Figure 1. Illustration of approximating the Bayes point for a two dimensional feature space.

In line 12 of Algorithm 2 we take the equal weighted combination of the weight vectors of the respective PRank algorithms to estimate the Bayes point. It seems natural that in order to achieve a good generalized solution and truly estimate the Bayes point 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 $\sum_{j=1}^{N} c_j(1) \leq \ldots \leq \sum_{j=1}^{N} c_{j,t}(k-1) \leq \sum_{j=1}^{N} c_{j,t}(k)$.

To see why $\tilde{\boldsymbol{c}}_t = \sum_{j=1}^{N} \boldsymbol{c}_{j,t}/N$ makes sense, consider a two dimensional space where all the instances with the same rank are grouped together. We justify considering the ranks separately due to the idea that PRank represents a rank $y = r$ by each subinterval's threshold $c(r)$ i.e. $\boldsymbol{w} \cdot \boldsymbol{x} \leq c(r)$, and there is order preserved amongst the thresholds. We add a dimension to the weights for each rank, putting the threshold at $w(1) = c(r)$ for subinterval r and make the instance's first dimension $x(1) = 1$. For the example consider that there are five instances with a rank of three in $\boldsymbol{z}$ which defines half spaces $\{\boldsymbol{w} : \boldsymbol{w} \cdot \boldsymbol{x} > 0\}$ (see Figure 1). The shaded region of Figure 1 defines the intersection of half spaces—a region which represents all the solution weights which correctly classify the five instances. We can see from Figure 1 that the center of this region (indicated by the $\boldsymbol{w}_{\text{cm}}$) would give some immunity to unseen examples.

A kernel implementation of OAP-BPM can be easily derived by first expressing each Perceptron f_j, $j = 1, \ldots, N$ by the representer theorem (Herbrich, 2002) as $f_j(\cdot) = \sum_{t=1}^{T} \alpha_{j,t} K(\boldsymbol{x}_t, \cdot)$ where $\alpha_{j,t} = b_{j,t} y_t \sum_{r=1}^{k-1} a_{j,r,t}$, combined with the reproducing property giving $\tilde{f}(\boldsymbol{x}) = \frac{1}{N} \sum_{j=1}^{N} f_j(\boldsymbol{x}) = \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \alpha_{j,t} K(\boldsymbol{x}_t, \boldsymbol{x})$. For a truly online algorithm we need to bound the number of instances $\boldsymbol{x}_t$ needed to be stored for the kernel expansion (Kivinen, Smola & Williamson 2001).

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) 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 generalization 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 data sets with parallel application.

3.1. Ensemble variants of OAP

As mentioned in the previous section the OAP-BPM is different to ensemble methods, like Bagging. Whilst Bagging in general is considered a batch learning method, Oza (2001) presented an online learning version. The online Bagging method of Oza is different to the sampling proposed for the OAP-BPM, in Oza (2001) each training example is presented to the online base model learning algorithm q times, where q is chosen from the Poisson distribution with a mean of one. Aside from the sampling technique being different, the other difference is our interest in the PRank as the base model learning algorithm. The Bagging method proposed in this paper uses the same Bernoulli sampling used in OAP-BPM, hence we refer to this as the OAP-Bagg algorithm.

Whilst Bagging is one ensemble method worth investigating another is the voted Perceptron (Freund & Schapire, 1999). In the voted Perceptron, applied to the PRank algorithm at each update $i = 1, \ldots, u$ the weight $\boldsymbol{w}_i$ is stored and the number of correct ranks between updates, v_i. At the end of training the final hypothesis of the voted Perceptron is given by $H(x) = \sum_{i=1}^{u} v_i h_i(x)$, where $h_i(x) := \min_{r \in \{1,\ldots,m\}} \{r : \boldsymbol{w}_i \cdot \boldsymbol{x} - c_i(r) < 0\}$. We propose, rather than a single Perceptron/PRank algorithm, combining N independent PRank algorithms sampled using the same technique used in OAP-BPM. We store the number of correct ranks made by each PRank algorithm, v_i, and combine them such that the final hypothesis is $H(x) = \sum_{i=1}^{N} v_i h_i(x) / |\sum_{i}^{N} v_i|$ (in the Bagging case, OAP-Bagg the final hypothesis is $H(x) = \sum_{i=1}^{N} h_i(x)/N$) . Note that we normalize the voted Perceptron by v, since rank learning in general

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

τ	OAP-VP	OAP-BAGG	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

is sensitive to the scale, which is not true in the case of binary classification. We refer to this algorithm as OAP-VP, since we combine the voted Perceptron (VP) with the online sampling of OAP-BPM.

It is well accepted that Bagging and the voted Perceptron perform well in the classification problem setting, Bagging especially in the presence of noise. The question is how does OAP-Bagg and OAP-VP compare with OAP-BPM in the rank learning setting, which motivates the empirical study in the experimental section.

4. Experiments

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

4.1. Ranking with a synthetic data set

We performed the synthetic data experiment of Herbrich, Graepel and Obermayer (2000), and Crammer and Singer (2002) in the batch setting with a non-homogeneous kernel of degree two. To generate the data each instance at round t, $\boldsymbol{x}_t = (x_t(1), x_t(2))$ was chosen according to the uniform distribution on the unit square i.e. $\boldsymbol{x} \in [0,1] \times [0,1] \subset \mathbb{R}^2$. The ranks $1, \ldots, 5$ were assigned by $y = \max_{r\in\{1,\ldots,5\}}\{r : 10((x(1)-0.5)(x(2)-0.5)) + n > c_r\}$ with the rank thresholds $\boldsymbol{c} = (-\infty, -1, -0.1, 0.25, 1)$ give that $c(5) = \infty$ and n normally distributed with zero mean and standard deviation of 0.125. The experiment consisted 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 $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t|$ where T is the number of rounds performed so far. Figure 2 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-Bagg ($N = 100$ and $\tau = (0.3, 0.6, 0.9)$) and OAP-VP ($N = 100$ and $\tau = (0.3, 0.6, 0.9)$). 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-VP 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-Bagg and OAP-VP, plus we tried PRank with the voted Perceptron. 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 1 show 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 1 we see as τ is made smaller so is the confidence interval.

4.2. Collaborative filtering

To allow for a fair comparison a general frame work 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 was drawn randomly from the ratings made on that item and then the remaining ratings were used as dimensions of the instance vector $\boldsymbol{x}_t$. All the results in this section are averages produced by 500 Monte-Carlo trials.

[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/`

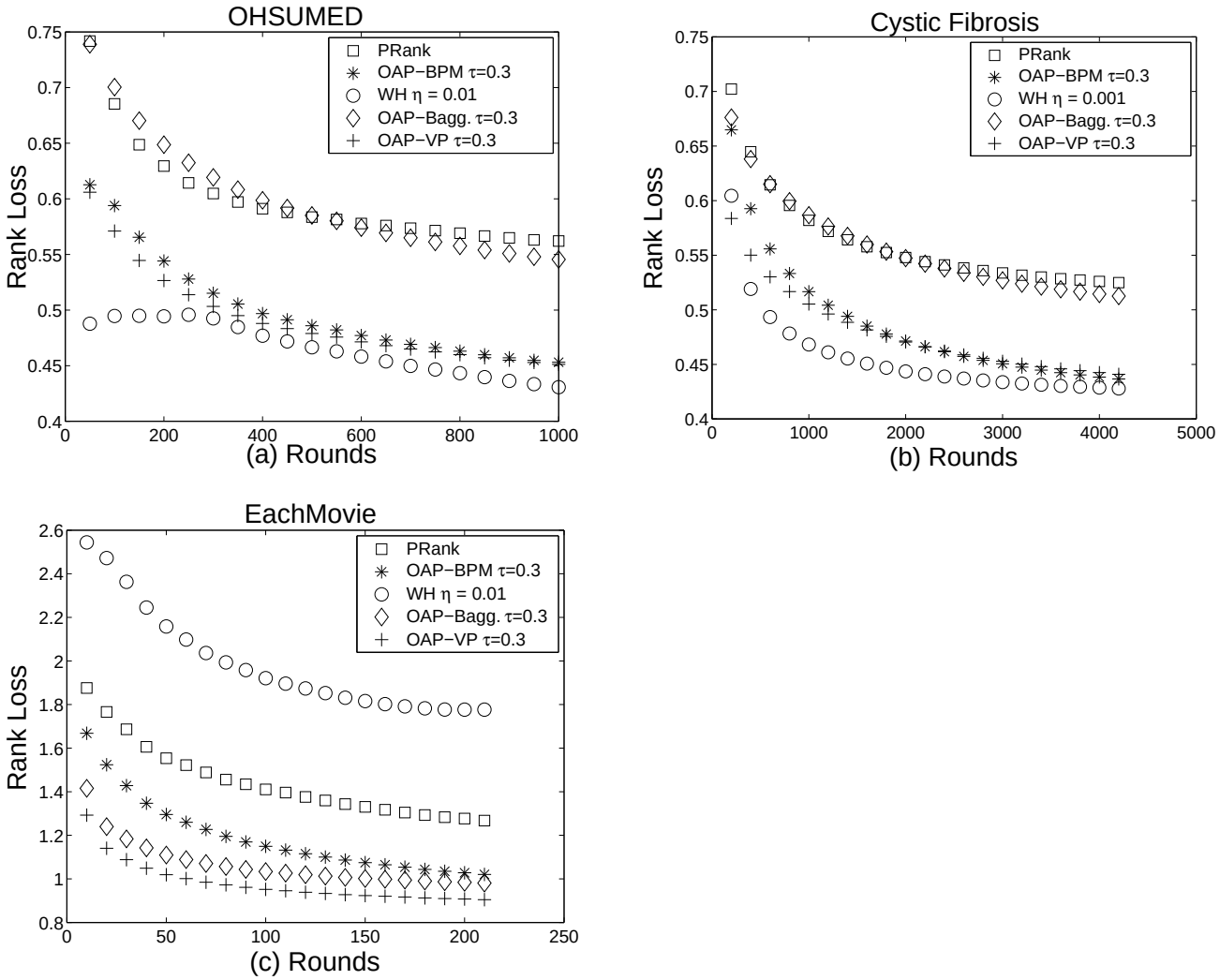

Figure 3. Collaborative filtering experimental results, comparing the averaged rank loss $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t|$, where T is the number of rounds of training performed, for the PRank, WH, OAP-BPM, OAP-Bagg, and OAP-VP algorithms, against the three data sets (a) OHSUMED, (b) cystic fibrosis, and (c) EachMovie.

The detail 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. Noting 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 differ-

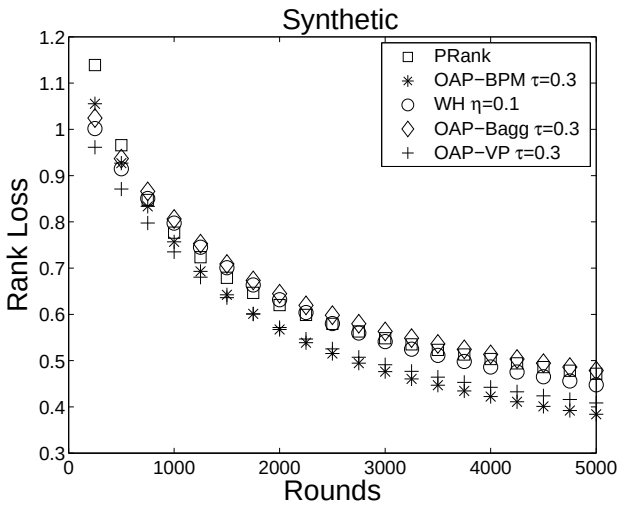

Figure 2. Experimental results for synthetic data set comparing the rank learning algorithms PRank, WH with step size $\eta = 0.1$, OAP-BPM with Bernoulli probability $\tau = 0.3$, OAP-BPM ensemble variants OAP-Bagg (Bagging) and OAP-VP (voted Perceptron).

ence 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, totaling 547 people. To allow for a score of zero for unseen movies 0.5 was subtracted from the peoples scores. The 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) $\boldsymbol{x}$ 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. The training set was selected at random to be 210 and the remaining 90 movies the test set.

Table 2. Collaborative filtering experimental results produced by test sample (not used in training), showing the averaged rank loss, $\frac{1}{T}\sum_{t=1}^{T}|\hat{y}_t - y_t|$ 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-VP			
($\tau = 0.1$)	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-Bagg.			
($\tau = 0.1$)	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

From the results of Table 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 3 (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-VP has a faster convergence than both OAP-Bagg 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 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 2 are better than the training set results in Figures 3 (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.

5. Summary and conclusions

We propose a simple OAP-BPM rank learning algorithm which improves the generalization performance of the PRank algorithm. The improved generalization performance is achieved by having the rank prediction $\boldsymbol{w}_{t+1} \cdot \boldsymbol{x}_{t+1}$ in the centre of the subinterval true rank y_t. 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 randomization trick. It was demonstrated that averaging the N rank rules (producing the final ranking rule of OAP-BPM) 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-Bagg and OAP-VP 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 generalization 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.

For future work, an interesting extension to the OAP-BPM would be the incorporation of ROMMA rather than the Perceptron in the update rule of PRank.

6. Acknowledgements

This work was conducted by the author whilst at The Australian National University. The author would like to thank Evan Greensmith, Ralf Herbrich, Jyrki Kivinen, John Platt and Bob Williamson, for there assistance in producing this manuscript and to the anonymous reviewers for their helpful comments.

References

[1] Breese, J. S., Heckerman D., & Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI:Morgan Kaufmann.

[2] Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24(2)*, 120-140.

[3] Crammer K. & Singer Y. (2003). A Family of Additive Online Algorithms for Category Ranking. *Journal of Machine Learning Research*, *3*, 1025-1058.

[4] Crammer K., & Singer Y. (2002). Pranking with Ranking. *Advances in Neural Information Processing Systems 14*, (pp. 641-647). Cambridge, MA: MIT Press.

[5] Duda R. O., Hart P. E., & Stork D. G. (2000). *Pattern Classification And Scene Analysis 2nd Edition.* John Wiley.

[6] Fruend Y., & Schapire R. (1999). Large Margin Classification Using the Perceptron Algorithm.*Machine Learning*, *37(3)*, 277-296.

[7] Gentile C. (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, *2*, 213-242.

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

[9] Herbrich, R. (2002). *Learning Kernel Classifiers*, Cambridge, MA: MIT Press.

[10] Herbrich, R., Graepel T., & Campbell C. (2001). Bayes Point Machines. *Journal of Machine Learning Research*, *1*, 245-279.

[11] Herbrich, R., Graepel, T. & Obermayer K. (2000). Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, MIT Press, pp. 115-132.

[12] Kowalczyk, A. (2000). Maximal margin perceptron. *Advances in Large Margin Classifiers*, MIT Press, pp. 75-115.

[13] Kivinen, J., Smola, A., & Williamson, R. C. (2002). Online Learning with kernels. *Advances in Neural Information Processing Systems 14* (pp. 785-792). Cambridge, MA: MIT Press.

[14] Y. Li. & P. Long, (2002). The relaxed online maximum margin algorithm. *Machine Learning*, *46(1-3)*, 361-387.

[15] Oza, N. C. (2001). *Online Ensemble Learning*. Doctoral dissertation, University of California, Berkeley.

[16] Resnick, P., & Varian, H. (1997). Recommender systems. *Communications of the ACM*, *40(3)*:56-58.

[17] Shashua A. & Levin A. (2002). Taxonomy of Large Margin Principle Algorithms for Ordinal regression Problems. *Advances in Neural Information Processing Systems 15* (to Appear). Cambridge, MA: MIT Press.

[18] Widrow B., & Hoff M. E. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record*, pt. 4, pp. 96-104.