Technische Universität Darmstadt
Knowledge Engineering Group
Hochschulstrasse 10, D-64289 Darmstadt, Germany

http://www.ke.informatik.tu-darmstadt.de

Technical Report TUD–KE–2007–05

Eneldo Loza Mencía, Johannes Fürnkranz

Pairwise Learning of Multilabel Classifications with Perceptrons

Pairwise Learning of Multilabel Classifications with Perceptrons

Eneldo Loza Mencía ENELDO@KE.INFORMATIK.TU-DARMSTADT.DE
Johannes Fürnkranz JUFFI@KE.INFORMATIK.TU-DARMSTADT.DE
Knowledge Engineering Group
Department of Computer Science
Technische Universität Darmstadt, Germany

Abstract

Multiclass Multilabel Perceptrons (MMP) are an efficient incremental algorithm for training a team of perceptrons for a multilabel prediction task. The key idea is to train one binary classifier per label, as is typically done for addressing multilabel problems, but to make the training signal dependent on the performance of the whole ensemble. In this paper, we propose an alternative approach that is based on a pairwise approach, i.e., we incrementally train a perceptron for each pair of classes. An evaluation on the Reuters 2000 (RCV1) data shows that our multilabel pairwise perceptron (MLPP) algorithm yields substantial improvements over MMP in terms of ranking quality and overfitting resistance, while maintaining its efficiency. Despite the quadratic increase in the number of perceptrons that have to be trained, the increase in computational complexity is bounded by the average number of labels per training example.

1. Introduction

An important field in machine learning are algorithms that learn an association of objects to classes. Often, exactly one out of a set of more than two classes is to be assigned, a case called multiclass classification. While this problem setting applies to a wide range of real life problems, there are several types of problems that are not covered by multiclass algorithms. One example is the association of text documents to genres, where more than one class can be appropriate for a single document. This type of classification, where an object is associated with a set of classes, is called *multilabel classification*. Only a small number of algorithms are able to naturally learn this type of problems.

A common approach for circumventing this restriction is the use of *class binarization* methods, i.e. the decomposition of a classification problem with more than two classes into several binary subproblems that can then be solved using a *binary base learner*. The simplest strategy is *one-against-all*, in the multilabel setting also referred to as the *binary relevance* method. It tackles a multilabel problem by learning one classifier for each class, using all objects of this class as positive examples and all other objects as negative examples. At query time, each binary classifier predicts whether its class is relevant for the query example or not, resulting in a set of relevant labels.

Another option is to transform the multilabel classification problem into a topic ranking task, where the goal is to compute prediction values indicating how relevant each class is for a particular example. Though this does not immediately result in a set of classes, it is possible to obtain the desired output in an additional step that selects classes which exceed a determined relevance value. Different methods exist for determining the threshold, a good overview is provided by Sebastiani [2002]. Recently, Brinker et al. [2006] introduced the idea of using an artificial label that encodes the boundary between relevant and irrelevant labels for each example. We note that all these approaches can be applied to the algorithm proposed in this paper, and in the following we restrict ourselves to a topic ranking scenario, which is also relevant for many practical applications.

Crammer and Singer [2003] combined the mentioned one-against-all method and the topic ranking idea in their multilabel multiclass perceptron algorithm (MMP). Instead of learning the relevance of each class individually and independently, MMP incrementally trains the entire classifier ensemble as a whole so that it predicts a real-valued relevance value for each class. This is done by always evaluating the performance of the entire ensemble, and only producing training examples for the individual classifiers when their corresponding classes are misplaced in the ranking. It uses perceptrons as base classifiers. In a way, the used learning technique may also be viewed as reinforcement learning from immediate rewards.

In this paper, we propose the use of pairwise decomposition as an alternative training method for an effective ensemble of perceptrons. In this method, one classifier is trained for each possible class pair, using the examples belonging to the two classes as positive or negative examples respectively. During prediction, an overall ranking of the classess determined by combining the predictions of the individual classifiers, e.g. by voting. One of the advantages of the approach is its efficiency: it can indeed be shown that pairwise ensembles can be more efficiently trained than the one-against-all ensemble [Fürnkranz, 2002]. Another advantage, of particular importance for perceptrons, is that decomposing the problem into smaller subproblems will yield simpler, often linear decision boundaries. For example, Knerr et al. [1992] observed that the classes of a digit recognition task were pairwise linearly separable, while the corresponding one-against-all task was not solvable with perceptrons.

Although the superiority of pairwise classification over one-against-all classification has been shown in several applications [Hsu and Lin, 2002, Fürnkranz, 2002], the study presented in this paper makes still important contributions for two reasons: First, previous works have exclusively concentrated on classification tasks. While it is natural to assume that the performance of the pairwise approach will also extend to the multilabel or topic ranking task, this has so far not been experimentally confirmed. Second, and more important, the comparison to MMPs is of particular interest because of the two alternative approaches for tackling the topic ranking problem: While MMPs propose to include information about the ranking task into the training signals, the pairwise approach addresses the ranking problem by breaking the ranking signal down into elementary binary preferences that induce the final ranking [Fürnkranz and Hüllermeier, 2003].

2. Preliminaries

We represent an instance or object as a vector $\bar{\mathrm{x}} = (x_1, \ldots, x_N)$ in a feature space $\mathcal{X} \subseteq \mathbb{R}^N$. Each instance $\bar{\mathrm{x}}_i$ is assigned to a set of relevant labels $\mathcal{y}_i$, a subset of the K possible classes $\mathcal{Y} = \{c_1, \ldots, c_K\}$. For multilabel problems, the cardinality $|\mathcal{y}_i|$ of the label sets is not restricted, whereas for binary problems $|\mathcal{y}_i| = 1$. For the sake of simplicity we use the following notation for the binary case: we define $\mathcal{Y} = \{1, -1\}$ as the set of classes so that each object $\bar{x}_i$ is assigned to a $y_i \in \{1, -1\}$, $\mathcal{y}_i = \{y_i\}$.

2.1 Perceptrons

A perceptron is a binary classifier initially developed as a model of the biological neuron [Rosenblatt, 1958]. Internally, it computes a linear combination of a real-valued input vector and predicts the positive class if the result is positive, and the negative class otherwise. More precisely, given an input vector $\bar{x}$, the predicted class of a perceptron is computed as

$$o'(\bar{x}) = sgn(\bar{x} \cdot \bar{w} + \omega) \tag{1}$$

with the weight vector $\bar{w}$, threshold ω and $sgn(t) = 1$ for $t \geq 0$ and -1 otherwise. We can interpret a perceptron as a hyperplane with the formula $\bar{x} \cdot \bar{w} = -\omega$ that divides the N-dimensional space into two halves. An instance is a point in this space and its position determines whether it is considered as belonging to one class or the other. If the two sets of positive and negative points, respectively, can be separated by a hyperplane, they are called *linearly separable*. As a consequence, irrespective of the training algorithm used, linear classifiers like the perceptron cannot arrive at correct predictions for all potential instances unless the negative and positive instances are linearly separable. In order to find a possibly existing *separating hyperplane*, the weights are adapted according to the following perceptron training rule:

$$\theta_i = (y_i - o'(\bar{x}_i)) \qquad \bar{w}_{i+1} = \bar{w}_i + \eta\theta_i\bar{x}_i \qquad \omega_{i+1} = \omega_i + \eta\theta_i\delta \tag{2}$$

with δ usually being set to 1 and the initial weights set to zero without loss of generality. The learning rate η can be ignored if set to be constant [Bishop, 1995], as it will be the case in this work. When a N-dimensional point is misclassified, the hyperplane is moved towards this point (indicated by θ). If the training examples can be seen iteratively and the data is linearly separable, the algorithm provably finds a dividing hyperplane. This is called the perceptron convergence criterion [cf., e.g., Bishop, 1995]. Irrespective of training until convergence not always being desirable, this property does not reveal anything about the performance on unseen data.

Note that the number of errors until convergence depends on the margin between the positive and negative points, i.e. the maximum diameter a separating hyperplane could have. The hyperplane that maximizes the margin to the closest positive and negative point is called the *optimal hyperplane*. Contrary to support vector machines, perceptrons will not necessarily find an optimal hyperplane. However, the size of the margin is an indicator for the hardness of the learning problem: the smaller the margin the harder it is for the perceptron algorithm to find a good solution. On the other hand, perceptrons can be trained efficiently in an incremental setting, which makes them particularly well-suited for large-scale classification problems such as the Reuters 2000 (RCV1) benchmark [Lewis et al., 2004]. For this reason, the perceptron has recently received increased attention [Freund and Schapire, 1999, Li et al., 2002, Shalev-Shwartz and Singer, 2005, Dekel et al., 2005].

Certainly, the δ value becomes important when the perceptron is trained in only one epoch: it is easily shown that $|\bar{w}| \leq |\mathcal{W}| \cdot \max_{\bar{x} \in \mathcal{W}} |\bar{x}|$ and $|\omega| \leq |\mathcal{W}| \cdot \delta$ holds for misclassified training examples $\mathcal{W} = \{\bar{x} \mid o'(\bar{x}) \neq y\}$. A disproportion between $\max_{\bar{x}} |\bar{x}|$ and δ can obviously lead to an excessive predominance of the threshold and make the scalar product even obsolete. To circumvent the problem of determining the right value for δ, we can set it to zero sacrificing one dimension in the hypothesis space (thus ω=0). Graphically this means that only separating hyperplanes through the origin are considered, reducing the number of potentially solvable problems. In practice, especially in high dimensional spaces as for text documents, this is usually not a very significant restriction, and it additionally renders on-line learning possible.

2.2 Binary Relevance Ranking

In the binary relevance or one-against-all (OAA) method, a multilabel training set with K possible classes is decomposed into K binary training sets of the same size that are then used to train K binary classifiers. So for each pair $(\bar{x}_i, \mathcal{y}_i)$ in the original training set K different pairs of instances and binary class assignments $(\bar{x}_i, y_{i_j})$ with $j = 1 \dots K$ are generated as follows:

$$y_{i_j} = \begin{cases} 1 & c_j \in \mathcal{y}_i \\ -1 & otherwise \end{cases} \tag{3}$$

Supposing we use perceptrons as base learners, K different o'_j classifier are trained in order to recognize if an instance is included in their respective class c_j. In consequence, the combined prediction would be the set $\{c_j \mid o'_j(\bar{x}) = 1\}$. If, in contrast, we want to obtain a ranking of classes according to their relevance, we can simply use the result of the internal computation of the perceptrons as a measure of relevance. According to Equation 1 the desired linear combination is the inner product $o_j(\bar{x}) = \bar{x} \cdot \bar{w}_j$ (ignoring ω as mentioned above). So the result of the prediction is a vector $\bar{o}(\bar{x}) = (\bar{x}\bar{w}_1, \dots, \bar{x}\bar{w}_K)$ where component j corresponds to the relevance of class c_j. We will denote the ranking function that returns the position of class c in the ranking with $r(c) \in \{1 \dots K\}$. Ties are broken randomly to not favor any particular class.

2.3 Ranking Loss Functions

In order to evaluate the predicted ranking we use different *ranking losses*. The losses are computed comparing the ranking with the true set of relevant classes, each of them focusing on different aspects. For a given instance $\bar{x}$, a relevant label set $\mathcal{y}$, a negative label set $\overline{\mathcal{y}} = \mathcal{Y} \backslash \mathcal{y}$ and a given predicted ranking $r(\bar{x})$ the different loss functions are computed as follows:

IsErr The is-error loss determines whether $r(c) < r(c')$ for all relevant classes $c \in \mathcal{y}$ and all irrelevant classes $c' \in \overline{\mathcal{y}}$. It returns 0 for a completely correct, *perfect ranking*, and 1 for an incorrect ranking, irrespective of 'how wrong' the ranking is.

ErrSetSize The error set size loss returns the number of pairs of labels which are not correctly ordered. Like IsErr, it is 0 for a perfect ranking, but it additionally differentiates between different degrees of errors.

$$E \overset{\text{def}}{=} \{(c, c') \mid r(c) > r(c')\} \subseteq \mathcal{y} \times \overline{\mathcal{y}} \tag{4}$$

$$\delta_{\text{ErrSetSize}} \overset{\text{def}}{=} |E| \tag{5}$$

```
Require: Training example pair $(\bar{x}, \mathcal{y})$, perceptrons $\bar{w}_1, \ldots, \bar{w}_K$
 1: calculate $\bar{x}\bar{w}_1, \ldots, \bar{x}\bar{w}_K$, loss $\delta$
 2: if $\delta > 0$ then                                  ▷ only if ranking is not perfect
 3:     calculate error sets $E$, $F$
 4:     for each $c \in F$ do $\tau_c \leftarrow 0$                    ▷ initialize $\tau$'s
 5:     for each $(c, c') \in E$ do
 6:         $p \leftarrow$ Penalty$(\bar{x}\bar{w}_1, \ldots, \bar{x}\bar{w}_K)$
 7:         $\tau_c \leftarrow \tau_c + p$                          ▷ push up positive classes
 8:         $\tau_{c'} \leftarrow \tau_{c'} - p$                      ▷ push down negative classes
 9:         $\sigma \leftarrow \sigma + p$                            ▷ for normalization
10:     normalize $\tau$'s
11:     for each $c \in F$ do
12:         $\bar{w}_c \leftarrow \bar{w}_c + \delta \frac{\tau_c}{\sigma} \cdot \bar{x}$                  ▷ update perceptrons
13: return $\bar{w}_1 \ldots \bar{w}_K$                       ▷ return updated perceptrons
```

Figure 1: Pseudocode of the training method of the MMP algorithm

Margin The margin loss returns the number of positions between the worst ranked positive and the best ranked negative classes. This is directly related to the number of wrongly ranked classes, i.e. the positive classes that are ordered below a negative class, or vice versa. We denote this set by F.

$$F \overset{\text{def}}{=} \{c \in \mathcal{y} \mid r(c) > r(c'), c' \in \overline{\mathcal{y}}\} \cup \{c' \in \overline{\mathcal{y}} \mid r(c) > r(c'), c \in \mathcal{y}\} \tag{6}$$

$$\delta_{\text{Margin}} \overset{\text{def}}{=} \max(0, \max\{r(c) \mid c \in \mathcal{y}\} - \min\{r(c') \mid c' \notin \mathcal{y}\}) \tag{7}$$

AvgP Average Precision is commonly used in Information Retrieval and computes for each relevant label the percentage of relevant labels among all labels that are ranked before it, and averages these percentages over all relevant labels. In order to bring this loss in line with the others so that an optimal ranking is 0, we revert the measure.

$$\delta_{\text{AvgP}} \overset{\text{def}}{=} 1 - \frac{1}{\mathcal{y}} \sum_{c \in \mathcal{y}} \frac{|\{c^* \in \mathcal{y} \mid r(c^*) \leq r(c)\}|}{r(c)} \tag{8}$$

3. Multiclass Multilabel Perceptrons

MMPs were proposed as an extension of the one-against-all algorithm with perceptrons as base learners [Crammer and Singer, 2003]. Just as in one-against-all, one perceptron is trained for each class, and the prediction is calculated via the inner products. The difference lies in the update method: while in the one-against-all method all perceptrons are trained independently to return a value greater or smaller than zero, depending on the relevance of the classes for a certain instance, MMPs are trained to produce a good ranking so that the relevant classes are all ranked above the irrelevant classes. The perceptrons therefore cannot be trained independently, considering that the target value for each perceptron depends strongly on the values returned by the other perceptrons.

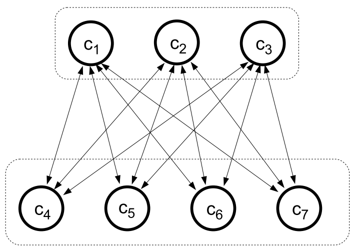

Figure 2: MLPP training: training example $\bar{x}$ belongs to $y = \{c_1, c_2, c_3\}$, $\overline{y} = \{c_4, c_5, c_6, c_7\}$ are the irrelevant classes, the arrows represent the trained perceptrons.

The pseudocode in Figure 1 describes the MMP training algorithm. When the MMP algorithm receives a training instance $\bar{x}$, it calculates the inner products, the ranking and the loss on this ranking in order to determine whether the current model needs an update. For determining the ranking loss, any of the methods of Section 2.3 is appropriate, since they all return a low value on good rankings. This allows to optimize the ranking in accordance with the used ranking loss. If the ranking is perfect, the algorithm is done, otherwise it calculates the error set of wrongly ordered class pairs E. The wrongly ranked classes are also represented in F. In the next step, each class that is present in a pair of E receives a penalty score. This is done according to a selectable penalty function. Crammer and Singer [2003] propose several methods, including a function that returns a value proportional to the difference of the scalar products of both classes. The most successful one, however, seemed to be the uniform update method, where each pair in E receives the same score. In the next step, the update weights τ are normalized and each perceptron whose class was wrongly ordered is updated.

An example will illustrate the peculiarities of the MMP update method: Suppose that all classes are correctly ordered except for one relevant and three irrelevant classes. The three negative classes are ranked immediately over the positive.The error set contains three wrongly ordered pairs and according to the uniform update method the positive class will receive in the sum a penalty of 3 and the negatives each 1. Thus the perceptron of the positive class will be updated to a degree three times as great compared with the other three, in accordance with the degree to which it contributed to the wrong ranking. Note that regardless of the used penalty function the positive and the negative classes receive in total the same penalty scores and these are afterwards normalized, so that the degree of the overall model update only depends on δ, i.e. on the quality of the ranking. More precisely, the hyperplanes of the perceptrons of the relevant classes are translated by a total amount of $\delta\,\bar{x}$, and the remaining classes by $-\,\delta\,\bar{x}$. In summary, the degree of the update for a particular perceptron depends 1) on the used penalty method, 2) on how much it contributed to the wrong ranking, and 3) on the general ranking performance.

```
Require: Training example pair $(\bar{x}, y)$,
    perceptrons $\{\bar{w}_{u,v} \mid u < v, c_u, c_v \in \mathcal{Y}\}$
1: for each $(c_u, c_v) \in y \times \overline{y}$ do
2:     if $u < v$ then
3:         $\bar{w}_{u,v} \leftarrow$ TrainPerceptron($\bar{w}_{u,v}, (\bar{x}, 1)$)      ▷ train as positive example
4:     else
5:         $\bar{w}_{v,u} \leftarrow$ TrainPerceptron($\bar{w}_{v,u}, (\bar{x}, -1)$)     ▷ train as negative example
6: return $\{\bar{w}_{u,v} \mid u < v, c_u, c_v \in \mathcal{Y}\}$                ▷ updated perceptrons
```

Figure 3: Pseudocode of the training method of the MLPP algorithm.

4. Multilabel Pairwise Perceptrons

In the pairwise binarization method, one classifier is trained for each pair of classes, i.e., a problem with K different classes is decomposed into $\frac{K(K-1)}{2}$ smaller subproblems. For each pair of classes (c_u, c_v), only examples belonging to either c_u or c_v are used to train the corresponding classifier $o'_{u,v}$. All other examples are ignored. In the multilabel case, an example is added to the training set for classifier $o'_{u,v}$ if u is a relevant class and v is an irrelevant class, i.e., $(u, v) \in y \times \overline{y}$ (cf. Figure 2). We will typically assume $u < v$, and training examples of class u will receive a training signal of $+1$, whereas training examples of class v will be classified with -1. Figure 3 shows the training algorithm in pseudocode. Of couse MLPPs can also be trained incrementally.

In order to return a class ranking we use a simple voting strategy, known as *max-wins*. Given a test instance, each perceptron delivers a prediction for one of its two classes. This prediction is decoded into a vote for this particular class. After the evaluation of all $\frac{K(K-1)}{2}$ perceptrons the classes are ordered according to their sum of votes.[1] At first sight, it may be disturbing that many 'unqualified' perceptrons are involved in the voting process: suppose that an unseen example $\bar{x}$ belongs to a label set y, then a perceptron trained on two classes of $\overline{y}$ cannot know anything relevant in order to separate y from $\overline{y}$ because it has not seen examples from y. In the worst case the noisy votes concentrate on single negative classes, which would lead to misclassifications. But note that any class can at most receive $K - 1$ votes, so that in the extreme case when the qualified perceptrons all classify correctly and the unqualified ones concentrate on a single class, a positive class will still receive at least $K - |y|$ and a negative at most $K - |y| - 1$ votes.

The pairwise binarization method is often regarded as superior to one-against-all because it profits from simpler decision boundaries in the subproblems [Fürnkranz, 2002, Hsu and Lin, 2002]. In the case of an equal class distribution, the subproblems have $\frac{2}{K}$ times the original size while one-against-all maintains the size. Typically, this goes hand in hand with an increase of the space where a separating hyperplane can be found. A simple example illustrates this: imagine two points a and b on a line representing the center of the positive and negative points. We now insert points according to an arbitrary distribution around a and b. Let $\mu(n)$ denote the margin between the negative and positive points depending on the number of inserted points n. This function is monotonically decreasing. Thus it is

1. Ties are broken randomly in our case.

very likely for a subproblem to have a larger margin than the full problem. We have seen in Section 2.1 that the performance strongly depends on the available margin between the points of the binary classes. Thus, it can be expected that the MLPP algorithm will also benefit from the pairwise approach. MMP, on the other hand, is based on an one-against-all binarization, which, as we have noted in Section 1, will typically have more complex decision boundaries.

5. Computational Complexity

The notation used in this section is the following: K denotes the number of possible classes, L the average number of relevant classes per instance in the training set, N the number of attributes, δ and $\hat{\delta}$ denote the loss and the accumulated average loss respectively. For each complexity we will give a lower Ω and upper bound O in Landau notation and an expected value. Since all three algorithms are incrementally trained we will present the computational complexity per instance. We will also represent the runtime dependencies in terms of perceptron prediction and update operations. Note that a scalar product operation $\bar{\mathrm{w}}\bar{\mathrm{x}}$ requires nearly the same amount of floating point additions and multiplications as an update operation $\bar{\mathrm{w}} + \tau\bar{\mathrm{x}}$, so we do not need to discriminate between these two. As we are interested in the difference between different binarization approaches, we can also ignore operations that have to be performed by both MMP and MLPP, such as sorting or internal real value operations.The complexity of a basic perceptron operation depends on the average number of non-zero attributes. This is particularly important for applications like text classification, where sparse feature vectors are common.

In terms of memory, a single perceptron model has a complexity of $O(N)$. Since the MMP algorithm uses one perceptron for each class and the MLPP algorithm one for each pair of classes, the memory complexities are $O(KN)$ and $O(K^2N)$ respectively.

Obviously, a perceptron prediction costs one basic perceptron operation. In each training step the perceptron must predict a class. If it was wrong the update will cost another operation. Let δ_{per} be 0 if the prediction was correct, otherwise 1, and let $\hat{\delta}_{per}$ be the expected average error, then we can represent the complexity as $1 + \hat{\delta}_{per} = O(1)$ for each instance. The first step in the MMP training is to produce a ranking, hence at least K operations are necessary even for a perfect ranking. If the ranking is incorrect, for each wrongly ranked class in F one perceptron is updated (assuming uniform penalties). This corresponds to $\delta_{\text{Margin}} + 1$, but only if there was an error, otherwise 0. We can hence write $|F|$ as $\delta_{\text{Margin}} + \delta_{\text{IsErr}}$. The time complexity for the MMP algorithm is therefore $K + \hat{\delta}_{\text{Margin}} + \hat{\delta}_{\text{IsErr}} = O(K)$. The MLPP algorithm evaluates each training example with $|y \times \overline{y}|$ perceptrons, i.e., $|y|(K - |y|)$ operations. One additional operation is required for each perceptron that makes an incorrect prediction. Using the average values $\hat{\delta}_{per}$ and L, we can denote the expected runtime as $L(K - L)(1 + \hat{\delta}_{per}) = O(LK)$. In the worst case L is $\frac{K}{2}$, resulting in quadratic time.[2] The bounds for the complexity relationship of MLPP to MMP are the following:

$$\tfrac{1}{4}L = L\frac{\frac{1}{2}K}{2K} \leq L\frac{K-L}{2K} \leq \frac{L(K-L)(1+\hat{\delta}_{per})}{K+\hat{\delta}_{\text{Margin}}+\hat{\delta}_{\text{IsErr}}} \leq L\frac{2(K-L)}{K} < L\frac{2K}{K} < 2L \tag{9}$$

2. We assume that $L \leq \frac{K}{2}$ holds, otherwise the problem can simply be inverted.

Table 1: Computational complexity given in expected number of perceptron operations per instance. K: *#classes*, L: *avg. #labels per instance*, N: *#attributes*, $\hat{\delta}$: *avg. Loss*, $\hat{\delta}_{per}, \hat{\delta}_{\textsc{IsErr}} \leq 1, \hat{\delta}_{\textsc{Margin}} < K$.

	training time	testing time	memory requirement
perceptron	$1 + \hat{\delta}_{per}$	1	N
MMP	$K + \hat{\delta}_{\textsc{Margin}} + \hat{\delta}_{\textsc{IsErr}}$	K	$O(KN)$
MLPP	$L(K-L)(1+\hat{\delta}_{per})$	$\frac{K(K-1)}{2}$	$O(K^2N)$
$\frac{MLPP}{MMP}$	$O(L)$	$\frac{K-1}{2}$	$O(K)$

Thus, on average, the MLPP algorithm will be L times slower than the MMP algorithm. An overview over the complexities can be found in Table 1.

6. Evaluation

The *Reuters Corpus Volume I* (RCV1) is currently one of the most widely used test collections for text categorization research. We used the preprocessed version by Lewis et al. [2004] that contains 804,414 newswire documents belonging to 103 different categories. The amount of relevant topics per example ranges from 1 to 17 and is on average 3.24.

6.1 Experimental Setup

The corpus was split into 535,987 training documents (all documents before and including April 26th, 1999) and 268,427 test documents (all documents after April 26th, 1999) and processed in chronological order. We used the token files from Lewis et al. [2004], which are already word-stemmed and stop-word reduced. However, we repeated the last step, as we experienced that there were still a few occurrences of stop words. Several tests with different values for the number of attributes and different methods for term weighting and feature selection were done in a systematic way in order to determine the most appropriate settings for both algorithms. Typically, we used MMPs to reduce the number of candidates and, among the remaining candidates, we picked a setting that worked well for both. The following settings proved to generally provide good results and to allow a fair and representative comparison: we used the common TF-IDF term weighting method [Sebastiani, 2002] and used the first 25,000 features ordered by their document frequency. All parameters of the pre-processing methods were only computed on the training set to ensure that no information from the test set enters the training phase. For the MMP algorithm we used the IsErr loss function and the uniform penalty function. This setting showed the best results in the work of Crammer and Singer [2003] on the RCV1 data set and our experiments confirm this. All perceptrons were initialized with random values.

Table 2: Comparison on the test set. Δ indicates the difference between MMP and MLPP in percentage.

	MMP	Δ [%]	MLPP	OAA
IsErr $\times 100$	29.35	-4.11	28.14	35.87
ErrSetSize	2.801	-31.45	1.920	7.614
Margin	2.120	-31.47	1.453	5.833
AvgP	92.82	0.92	93.67	90.00

6.2 Direct Comparison

The results of a direct comparison of MMPs and MLPPs are presented in Table 2. The values for IsErr and AvgP are presented $\times 100$ for better readability, AvgP is also presented in the conventional way (with 100% as the optimal value) and not as a loss function. The results clearly show that the MLPP algorithm outperforms the MMP algorithm (all differences are statistically significant). Especially on the losses that directly evaluate the ranking performance the improvement is quite pronounced. On average, MLPPs increase the number of correctly ranked relevant and irrelevant class pairs by almost one pair per example. Similarly, the margin between the positive and negative classes is increased by more than half a class. For the IsErr, the advantage is less pronounced. Typically, a perfect classification is more likely to occur on documents that have a small number of labels, whereas on documents with an increasing number of labels the IsErr performance decreases rapidly. Thus, IsErr focuses more on the performance on cases where there is not much to rank. The AvgP measure yields a similar gain.

It is particularly important to note that MLPPs outperform MMPs in terms of IsErr, although MMPs were trained to directly optimize this loss function, whereas MLPPs are independent of a particular loss function. This holds also if MMPs are trained to optimize a different loss. For example, the best MMPs trained to optimize Margin yield an average Margin-Loss of 1.95.

In order to evaluate the algorithms on problem classes other than text classification, we ran a few quick tests on the *yeast* (2417 examples, 103 features, 14 classes) and *scene* (2000 examples, 294 features, 5 classes) datasets[3]. The results for 100 epochs over the training data and 10 fold cross-validation are shown in Table 3. The results for yeast are mixed, and for scene MLPPs are somewhat better according to all measures, but in general the differences are inconclusive on these datasets (only IsErr, ErrSetSize and Margin differences on *yeast* being statistically significant). In contrast to text classification, these problems cannot be expected to be linearly separable. The small number of features hardly allows to construct a separating hyperplane, even for pairs of classes, as the small differences between both aproaches suggest. In summary, MMP and MLPC seem not to apply quite well to hardly linearly separable problems, they are rather designed to solve high-dimensional (and also huge) problems. However kernel functions can be implemented and might solve this restriction [Freund and Schapire, 1999, Crammer and Singer, 2003].

3. both retrieved at http://mlkd.csd.auth.gr/multilabel.html

Table 3: Comparison for the *yeast* and *scene* data sets.

	yeast		*scene*	
	MMP	MLPP	MMP	MLPP
IsErr ×100	89.04	90.69	48.40	48.36
ErrSetSize	15.14	14.51	0.988	0.958
Margin	8.09	7.99	0.972	0.946
AvgP	57.38	57.27	71.52	71.75

6.3 Learning Curve

A learning curve shows how quickly a learner is able to adapt its model to the data presented. For incremental learners, it is often used to show the learning progress. Before a new example is added to the training set, it is first tested. The learning curve then shows the accumulated loss over the processed training instances. The result for the IsErr loss can be seen in Figure 4 and 6. Figure 6 shows that with an increasing number of examples, MLPPs accumulate a clear advantage. However, in the beginning, as can be seen from Figure 4, the differences are less pronounced and the MMP algorithm also seems to have a somewhat better performance in this region. For the ranking losses ErrSetSize (cf. Figure 5) and Margin (not shown here) the graphs look very similar, except that here MLPPs clearly have a better performance from the start. This result and even the comparison in terms of IsErr is remarkable because the perceptrons of the MLPP are trained on fewer examples, which may be particularly problematic in the beginning of the training phase.

6.4 Overfitting Analysis

In order to evaluate the overfitting property, both algorithms were trained in several epochs over the training set. Crammer and Singer [2003] observed that the performance of the MMP algorithm became worse with an increasing number of epochs. Our results confirm this observation: While the evaluation on the training data indicates a better adaptation, the performance on the test data decreases (Figure 7). For the MLPP algorithm the better adaptation to the training data is also clearly observable, it quickly reaches losses near 0, but in contrast to MMP, the results on the test data remain stable. We interpret this as evidence that pairwise decomposition of the problem does in fact fit the problem structure, i.e., that the classes here are in fact pairwise linearly separable. MLPPs learn these linear decision boundaries after the first epoch through the training examples, so that further training is not necessary (but can also not lead to more overfitting).

6.5 Computational Costs

We measured the required run-time in order to compare it to our analysis of the computational complexity in Section 5. We found it most convenient to measure the amount of processed operations instead of the amount of (CPU-)time, since in this way it is guaranteed to be independent from external factors such as logging activities. Therefore we used perceptron operations (as discussed above) as measure for our experiments. The MMP algorithm required 56,680,708 operations for training and 27,647,981 to process the test

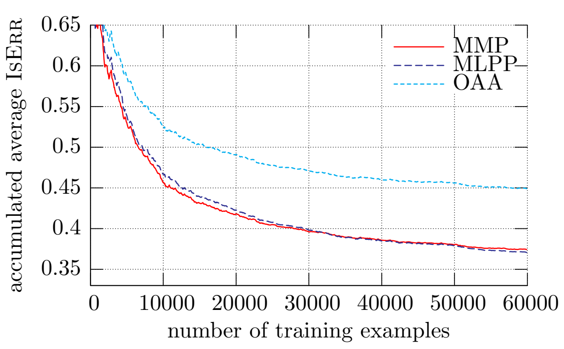

Figure 4: Learning curve for the first 60000 examples (IsErr).

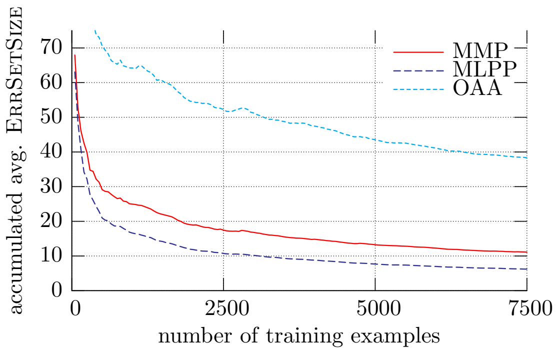

Figure 5: Learning curve for the first 7500 examples (ErrSetSize).

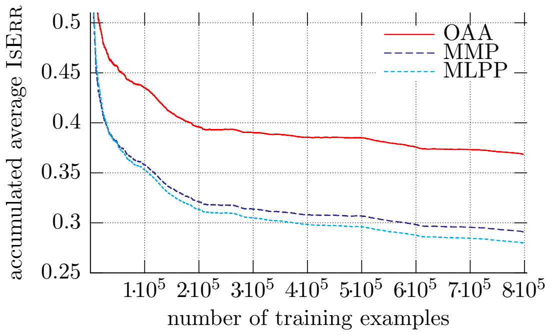

Figure 6: Learning curve for the full data set (IsErr).

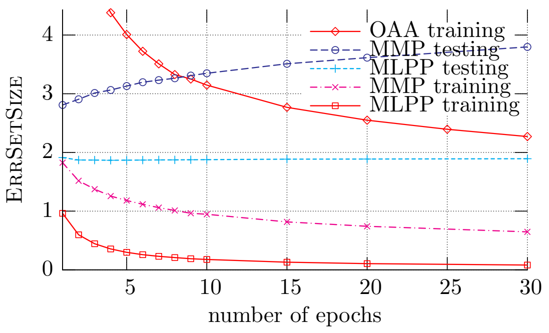

Figure 7: Error on training and test data depending on the number of epochs.

set. Analogously, the MLPP spent 174,184,241 and 1,410,047,031 operations. The ratios between MLPP and MMP for these values come to 3.07 and 51, respectively. This confirms our analysis, since the ratio for training is approximately the average number of labels and the ratio in testing is $\frac{103-1}{2}$.

7. Conclusions

In this paper, we evaluated the use of a pairwise ensemble of perceptrons for multilabel classification. Our results showed that the resulting incremental learning algorithm can be efficiently applied to large-scale text categorization problems such as the Reuters 2000 benchmark dataset, which are too complex for popular text categorization techniques such as standard support vector machines due to their super-linear time complexity. Recently, however, efficient alternatives have been proposed [Joachims, 2006].

In terms of accuracy, the pairwise approach compares favorably to multiclass multilabel perceptrons, a recent algorithm for training an one-per-class ensemble of perceptrons in a co-ordinated way by making the training signal of each perceptron dependent on a loss function that depends on the entire ranking, and thus dependent on the predictions of the other perceptrons in the ensemble. With the pairwise approach, we go an alternative way and try to break up the problem into independent subproblems by not trying to directly minimize any particular ranking loss, but by trying to learn the ordering relation that induces the ranking. This comparison of principles for addressing ranking problems is, in our opinion, the most important contribution of this work. In addition, we also believe that pairwise classification has not yet been tried with a problem of this size (both in number of training examples and in number of labels).

However, the increase in predictive performance has to be paid with a small increase in computational complexity, namely by a factor that depends on the average number of labels per example. As for most multilabel problems (in particular in text classification), this factor is rather small, so we consider this not to be a significant problem.

The reason for the good performance of MLPPs seems to be the adequacy of this pairwise problem decomposition. Contrary to MMPs, they are able to find perfect classifications on the training data, which are not due to overfitting but also carry over to improved performance on the test set. Moreover, this performance does not degrade if more training epochs are used, as seems to be the case for MMPs. Thus, the problem seems to be pairwise linearly separable. We believe that this will be the case for many text categorization problems.

For future research, we see space for improvement especially in two areas for the pairwise approach. First, the used decoding is suboptimal because the prediction weights of the base classifiers could be taken into account. Second, several variants of the perceptron were developed that, like SVMs, try to maximize the margin of the separating hyperplane in order to produce more accurate models. Preliminary tests with the voting technique by Price et al. [1995] and ballseptrons [Shalev-Shwartz and Singer, 2005] showed promising results.

References

Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995. ISBN 0198538642.

Klaus Brinker, Johannes Fürnkranz, and Eyke Hüllermeier. A unified model for multilabel classification and ranking. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06)*, 2006. URL http://www.ke.informatik.tu-darmstadt.de/~juffi/publications/ecai-06.pdf.

Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3(6):1025–1058, 2003. URL http://www.jmlr.org/papers/v3/crammer03b.html.

Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems 18*, 2005.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999. URL http://www.cse.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf.

Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002. URL http://www.jmlr.org/papers/v2/fuernkranz02a.html.

Johannes Fürnkranz and Eyke Hüllermeier. Pairwise preference learning and ranking. In N. Lavrač, D. Gamberger, H. Blockeel, and L. Todorovski, editors, *Proceedings of the 14th European Conference on Machine Learning (ECML-03)*, volume 2837 of *Lecture Notes in Artificial Intelligence*, pages 145–156, Cavtat, Croatia, 2003. Springer-Verlag. URL http://www.ke.informatik.tu-darmstadt.de/~juffi/publications/ecml-03.pdf.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002. URL http://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.pdf.

Thorsten Joachims. Training linear svms in linear time. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 217–226. ACM, 2006. ISBN 1-59593-339-5.

Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. Handwritten digit recognition by neural networks with single-layer training. *IEEE Transactions on Neural Networks*, 3(6): 962–968, 1992.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004. URL http://jmlr.csail.mit.edu/papers/v5/lewis04a.html.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola. The perceptron algorithm with uneven margins. In Claude Sammut and Achim G. Hoffmann, editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 379–386. Morgan Kaufmann, 2002. ISBN 1-55860-873-7.

David Price, Stefan Knerr, Leon Personnaz, and Gerard Dreyfus. Pairwise neural network classifiers with probabilistic outputs. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 1109–1116. The MIT Press, 1995. URL http://citeseer.ist.psu.edu/price94pairwise.html.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002. URL http://citeseer.ist.psu.edu/article/sebastiani99machine.html.

Shai Shalev-Shwartz and Yoram Singer. A new perspective on an old perceptron algorithm. In *Learning Theory, 18th Annual Conference on Learning Theory (COLT 2005)*, pages 264–278. Springer, 2005. ISBN 3-540-26556-2. URL http://www.cs.huji.ac.il/~shais/papers/ShalevSi05.pdf.