Pairwise Learning of Multilabel Classifications with Perceptrons

Eneldo Loza Mencía and Johannes Fürnkranz

Abstract— **Multiclass multilabel perceptrons (MMP) have been proposed as an efficient incremental training algorithm for addressing a multilabel prediction task with a team of perceptrons. 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 technique that is based on a pairwise approach, i.e., we incrementally train a perceptron for each pair of classes. Our evaluation on four multilabel datasets shows that the 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.**

I. Introduction

MULTILABEL CLASSIFICATION problems have gained increasing attention in recent times. In contrast to the well-known multiclass setting, the target classes are not exclusive: an object may belong to an unrestricted set of classes instead of exactly one. This applies to a wide range of real life problems, the mapping of texts to genres being the best known in the field of machine learning. Only a small number of algorithms are naturally able to learn this type of problem.

A common approach to address this problem is the use of *class binarization* methods, i.e. the decomposition of the problem 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 label 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. Several methods exist for determining the threshold [Sebastiani, 2002], the method of Elisseeff and Weston [2001] being one of the most cited. 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 label ranking scenario, which is also relevant for many practical applications.

Crammer and Singer [2003] combined the mentioned one-against-all method and the label ranking idea in their multiclass multilabel 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 score 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 incorrectly ordered in the ranking. Perceptrons are used as base classifiers.

In this paper, we propose the use of pairwise decomposition as an alternative training method for an effective ensemble of perceptrons. We train one classifier 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 classes is 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 still makes 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

The authors are with the Knowledge Engineering Group, TU Darmstadt, Germany. {eneldo,juffi}@ke.informatik.tu-darmstadt.de.

This is the authors' version of the paper, which appeared in the *Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IJCNN-08)*, DOI: 10.1109/IJCNN.2008.4634206, and for which the following applies:

also extend to the multilabel or label ranking task, this has so far not been experimentally confirmed. Second, and more importantly, the comparison to MMPs provides another datapoints in evaluating the two alternative approaches for tackling the label 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].

II. Preliminaries

We represent an instance or object as a vector $\bar{\mathrm{x}} = (x_1, \dots, x_N)$ 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 $\mathcal{Y} = \{c_1, \dots, c_K\}$. For multilabel problems, the cardinality $|\mathcal{y}_i|$ of the label sets is not restricted, whereas for binary problems $|\mathcal{y}_i| = 1$ holds. 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 $\bar{\mathrm{x}}_i$ is assigned to a $y_i \in \{1, -1\}$, $\mathcal{y}_i = \{y_i\}$.

A. Perceptrons

The perceptron is a well-studied 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 a class depending on whether the result is positive or negative. We use a version that works without learning rate and threshold.[1] The output of this perceptron is given by

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

with the weight vector $\bar{\mathrm{w}}$ and $sgn(t) = 1$ for $t \geq 0$ and -1 otherwise. We can interpret a perceptron as a hyperplane that divides the N-dimensional space into two halves. If it is possible to find a *separating hyperplane* that separates negative from positive instances, the two set of points are called *linearly separable*. In order to find such a hyperplane, the weights are adapted according to the following rule:

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

If the training examples are seen iteratively and the data is linearly separable, the algorithm provably finds a separating hyperplane [cf., e.g., Bishop, 1995]. In this case, the number of errors until convergence depends on the margin between the positive and negative points. The size of the margin is thus an indicator for the hardness of the learning problem: the smaller the margin the harder it is for the algorithm to find a good solution. Contrary to Support Vector Machines, which find an (optimal) maximum margin hyperplane, perceptrons can be trained efficiently in an incremental setting, which makes them particularly well-suited for large-scale classification problems such as the RCV1 benchmark [Lewis et al., 2004], without forfeiting too much accuracy. For this reason, the perceptron has recently received increased attention [e.g. Freund and Schapire, 1999, Li et al., 2002, Shalev-Shwartz and Singer, 2005, Crammer et al., 2006, Khardon and Wachman, 2007, Tsampouka and Shawe-Taylor, 2007]. Alternatively, some researchers have proposed efficient training algorithms for approximating the maximum margin hyperplane [Joachims, 2006].

[1]Note that the learning rate becomes superfluous when it is set to be constant [Bishop, 1995] and using a threshold, i.e. extending each vector $\bar{\mathrm{x}}_i$ by one dimension with a constant value ρ, implies having to tune an additional parameter [Tsampouka and Shawe-Taylor, 2007]. To circumvent this problem, we set it to zero sacrificing one dimension in the hypothesis space. 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.

B. Binary Relevance Ranking

In the binary relevance (BR) 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 $(\bar{\mathrm{x}}_i, \mathcal{y}_i)$ in the original training set K different pairs of instances and binary class assignments $(\bar{\mathrm{x}}_i, y_{i_j})$ 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 $\{c_j \mid o'_j(\bar{\mathrm{x}}) = 1\}$. 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 $o_j(\bar{\mathrm{x}}) = \bar{\mathrm{x}} \cdot \bar{\mathrm{w}}_j$ (ignoring ω as mentioned above). So the result of the prediction is a vector $\bar{\mathrm{o}}(\bar{\mathrm{x}}) = (\bar{\mathrm{x}}\bar{\mathrm{w}}_1, \dots, \bar{\mathrm{x}}\bar{\mathrm{w}}_K)$ 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.

C. 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 $\bar{\mathrm{x}}$, a relevant label set $\mathcal{y}$, a negative label set $\overline{\mathcal{y}} = \mathcal{Y} \backslash \mathcal{y}$ and a given predicted ranking function r 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. Such as IsErr, it is 0 for a perfect ranking, but it additionally differentiates between different degrees of errors.

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

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

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.

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

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

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.

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

III. MULTICLASS MULTILABEL PERCEPTRONS

MMPs were proposed as an extension of the binary relevance algorithm with perceptrons as base learners [Crammer and Singer, 2003]. Just as in binary relevance, 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 binary relevance 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.

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 II-C 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

```
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
```

Fig. 1. Pseudocode of the training method of the MMP algorithm

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 to 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.

IV. 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., $(u, v) \in \mathcal{y} \times \overline{\mathcal{y}}$ (cf. Figure 2). We will typically assume $u < v$, and training examples of class u will receive a

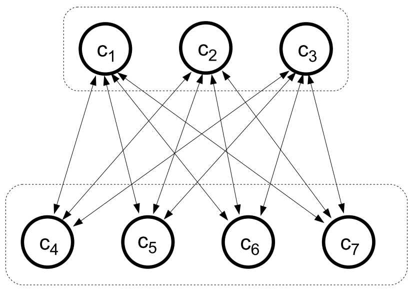

Fig. 2. MLPP training: training example $\bar{x}$ belongs to $y = \{c_1, c_2, c_3\}$, $\overline{y} = \{c_4, c_5, c_6, c_7\}$ are the irrelevant classes, the arrows represent the trained perceptrons.

```
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
```

Fig. 3. Pseudocode of the training method of the MLPP algorithm.

training signal of $+1$, whereas training examples of class v will be classified with -1. Figure 3 shows the training algorithm in pseudocode. Of course 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 (ties are broken randomly). 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 binary relevance 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 binary relevance 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 very likely for a subproblem to have a larger margin than the full problem. We have seen in Section II-A 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 binary relevance binarization, which, as we have noted in Section I, will typically have more complex decision boundaries.

Note that pairwise classification can be seen as a special case of the generalized Error Correcting Output Codes (ECOC) framework [Allwein et al., 2000] with a fixed encoding matrix. We believe that this is a case that deserves special attention for several reasons. First and foremost, it has clearly defined semantics. Each binary classifier determines which of two labels is to be preferred for a given example. This is the smallest piece of information that is needed for establishing an order between all labels. Second, it is a fixed, domain-independent and non-stochastic decomposition method that has a good overall performance. In several experimental studies (including Allwein et al. [2000]), it performed en par or better than competing decoding matrices. The chief reason for the good performance is that two-class problems are often linearly separable, even in low-dimensional spaces. Finally, it is also among the most efficient decoding schemes. In some sense, our philosophy is orthogonal to ECOC: While ECOC puts its efforts into choosing a good encoding matrix, we fix the encoding to the all-pairs approach and concentrate on the decoding phase. We believe that many practical problems can be reduced to estimating pairwise probabilities. These can then be combined in various ways, optimizing different performance criteria with a single, fixed pairwise ensemble.

V. 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 $\bar{w}\bar{x}$ requires nearly the same amount of floating point additions and multiplications as an update operation $\bar{w} + \tau\bar{x}$, 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

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

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 such as 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 $\delta_{\textsc{Margin}} + 1$, but only if there was an error, otherwise 0. We can hence write $|F|$ as $\delta_{\textsc{Margin}} + \delta_{\textsc{IsErr}}$. The time complexity for the MMP algorithm is therefore $K + \hat{\delta}_{\textsc{Margin}} + \hat{\delta}_{\textsc{IsErr}} = O(K)$. 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:

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

Thus, assuming similar loss rates, the MLPP algorithm will be on average L times slower than the MMP algorithm. An overview over the complexities can be found in Table I.

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

VI. 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.

A. 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.

We performed also tests with the binary relevance method and a multilabel variant of the multinomial Naive Bayes (NB) algorithm in order to provide a baseline. In one of our first experiments we counted the TF-IDF instead of the term frequency values for the Naive Bayes. We found out that by using this additional information about the overall relevance of each term the accuracy even doubled for some losses. We report therefore these improved results.[3]

B. Direct Comparison

The results of a direct comparison of MMPs and MLPPs are presented in Table II. 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 according to the Wilcoxon Signed-Rank Test [Demšar, 2006]). Especially on

[3]Note also that using the Naive Bayes as base classifier for a pairwise binarization is pointless as it results in the normal Naive Bayes [Sulzmann et al., 2007].

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 improved 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 other data sets, we performed a few quick tests on the older *reuters-21578* data set (11367 examples, 10000 features, 120 classes)[4] and, to represent other application settings, the *yeast* (2417 examples, 103 features, 14 classes, in average 4.24 labels per example) and *scene* (2000 examples, 294 features, 5 classes, in average 1.24 labels per example) data sets.[5]

Previous experiments showed that both non-text problems seem to be hardly linearly separable even using pairwise decomposition and that both algorithms appear to not apply well to this sort of problems [Loza Mencía and Fürnkranz, 2007]. As the results were inconclusive, we simulated the use of a polynomial kernel of second degree by adding all possible pairwise products of the original features to the feature vector. This allows us to translate the problem into a higher dimensional space where it is more likely linearly separable without having to modify our implementation and analysis. Note however that it is generally possible to use any kernel function in combination with perceptrons [Freund and Schapire, 1999, Crammer and Singer, 2003]. We plan to investigate how to take advantage especially of the pairwise decomposition in order to use kernel functions more efficiently.

The results for 10 epochs (i.e. 10 iterations over the training data) in the case of *reuters-21578* and 100 epochs for *yeast* and *scene* over the training data and ten-fold cross-validation are shown in Table III. The results on the *reuters21578* data set confirm the results on the newer Reuters data set, however the rather small difference in IsErr is not statistically significant (one-side Wilcoxon signed-rank test with $p = 0.05$ was performed on the three data sets). The pairwise decomposition also shows to improve the classification results on the *yeast* data set (all results are statistically significant). Nevertheless the problem seems to be generally hard to handle even with the usage of kernels since the losses are quite high. In contrast the results on the *scene* data set are better in general. The differences between MMP and MLPP are clearly visible, though the results of ErrSetSize and Margin are not statistically significant. Note the remarkable performance of BR in this case, even outperforming the MMPs on IsErr (no comparison with BR is statistically significant). Although we see the main application field of the MLPP algorithm in the efficient solving of large scale (in number of examples as well as features) problems such as text classification, where feature vectors are additionally very sparse, the reported results are competitive with other published results on the *yeast* and *scene* data set [Zhang and Zhou, 2006, Veloso et al., 2007].

[4]retrieved at http://www.daviddlewis.com/resources/testcollections/reuters21578 and preprocessed similar to RCV1

[5]both retrieved at http://mlkd.csd.auth.gr/multilabel.html

TABLE II
COMPARISON ON THE RCV1 TEST SET. Δ INDICATES THE DIFFERENCE BETWEEN MMP AND MLPP IN PERCENTAGE.

	MMP	Δ [%]	MLPP	BR	NB
IsErr $\times 100$	29.35	-4.11	28.14	35.87	51.79
ErrSetSize	2.801	-31.45	1.920	7.614	6.285
Margin	2.120	-31.47	1.453	5.833	4.513
AvgP	92.82	0.92	93.67	90.00	82.26

TABLE III
COMPARISON FOR THE *reuters21578*, *yeast* AND *scene* DATA SETS.

	reuters21578			*yeast*			*scene*		
	MLPP	MMP	BR	MLPP	MMP	BR	MLPP	MMP	BR
IsErr $\times 100$	17.46	17.83	22.18	74.43	77.33	79.07	25.22	26.30	26.21
ErrSetSize	1.691	2.974	12.38	6.456	7.887	8.581	0.392	0.412	0.430
Margin	1.399	2.565	9.602	4.396	5.188	5.553	0.391	0.410	0,430
AvgP	90.77	90.20	84.45	75.15	71.39	70.41	86.43	85.82	85.64

C. 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 on the RCV1 data 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.

D. Overfitting Analysis

In order to evaluate the overfitting property, both algorithms were trained in several epochs over the training set.

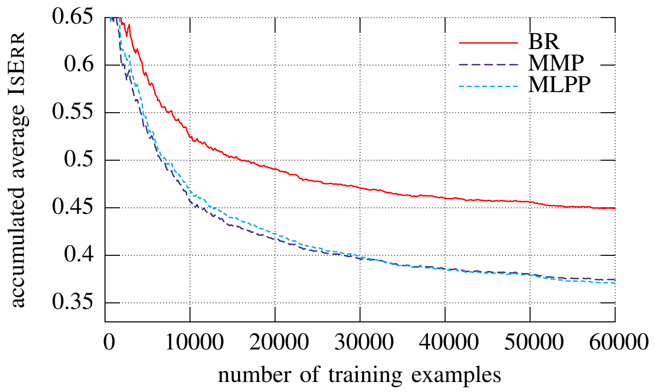

Fig. 4. Learning curve for the first 60000 examples (IsErr).

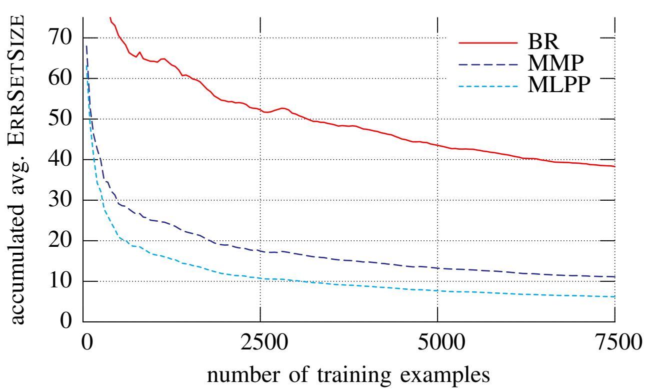

Fig. 5. Learning curve for the first 7500 examples (ErrSetSize).

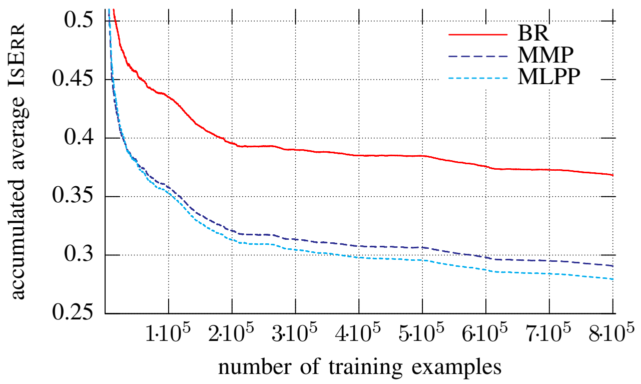

Fig. 6. Learning curve for the full data set (IsErr).

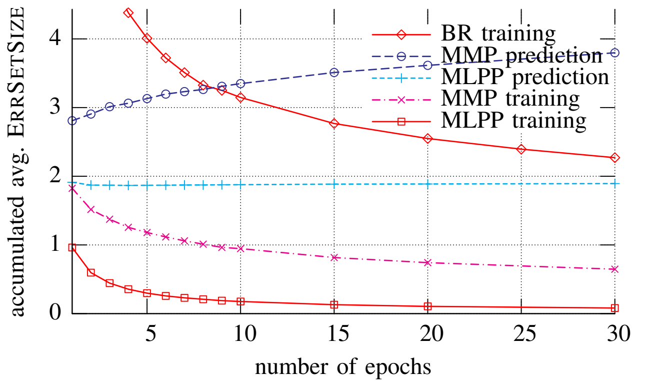

Fig. 7. Error on training and test data depending on the number of epochs.

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).

E. Computational Costs

To validate our analysis of the computational complexity in Section V, we measured the amount of processed perceptron operations to guarantee to be independent from external factors such as logging activities and others. The MMP algorithm required 56,680,708 operations for training and 27,647,981 to process the test set. Analogously, the MLPP spent 174,184,241 and 1,410,047,031 operations (cf. Table IV). The ratios between MLPP and MMP for these values come to 3.07 and exactly 51, respectively. This confirms our analysis, since the ratio for training is approximately the average number of labels and the ratio in testing is $\frac{103-1}{2}$. The CPU-time for predicting the labels of 268,427 documents was about 7 minutes for MMPs and 84 minutes for MLPPs (on a 2 GHz Dual Core Opteron).

We expect to be able to considerably reduce the prediction time by adapting the efficient voting method proposed in [Park and Fürnkranz, 2007] to multilabel problems. The key idea of the method is to ignore predictions for which it can be guaranteed that they only affect the ranking of irrelevant labels. It can easily be adapted to the multilabel case in conjunction with a thresholding method such as the artificial boundary label of Brinker et al. [2006].

VII. Conclusions

In this paper, we evaluated the use of a pairwise ensemble of perceptrons for multilabel classification. Our results showed that despite the need for training a quadratic number of classifiers, the resulting incremental learning algorithm can be efficiently applied to large-scale text categorization problems. Moreover, 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 coordinated 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

TABLE IV

NUMBER OF PERCEPTRON OPERATIONS FOR THE DIFFERENT ALGORITHMS ON THE TRAINING AND TEST SET

	BR	MMP	MLPC
training	55,896,832	56,680,708	174,184,241
prediction	27,647,981	27,647,981	1,410,047,031

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 (about 3.24 in the Reuters 2000 data set), so we consider this not to be a significant problem. The prediction time remains a problem, however we expect to considerably reduce the prediction costs to the level of training time by using Quick Weighted Voting [Park and Fürnkranz, 2007] in conjunction with the artificial boundary label of Brinker et al. [2006] (cf. Section VI-E). We are currently working on this issue.

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 it does 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 room 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 algorithm were developed that, similar to SVMs, try to maximize the margin of the separating hyperplane in order to produce more accurate models. Preliminary tests with the weighted voting technique by Price et al. [1995] on the one hand and perceptrons with margins [Li et al., 2002, Crammer et al., 2006, Khardon and Wachman, 2007, Tsampouka and Shawe-Taylor, 2007] on the other hand showed promising results.

REFERENCES

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 9–16, 2000.

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

Klaus Brinker, Johannes Fürnkranz, and Eyke Hüllermeier. A Unified Model for Multilabel Classification and Ranking. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06)*, 2006.

Koby Crammer and Yoram Singer. A Family of Additive Online Algorithms for Category Ranking. *Journal of Machine Learning Research*, 3(6):1025–1058, 2003.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Andre Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, volume 14, pages 681–687, 2001.

Yoav Freund and Robert E. Schapire. Large Margin Classification using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, 1999.

Johannes Fürnkranz. Round Robin Classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

Johannes Fürnkranz and Eyke Hüllermeier. Pairwise Preference Learning and Ranking. In *Proceedings of the 14th European Conference on Machine Learning (ECML-03)*, pages 145–156, 2003.

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.

Thorsten Joachims. Training Linear SVMs in Linear Time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006.

Roni Khardon and Gabriel Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8: 227–248, 2007.

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.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola. The Perceptron Algorithm with Uneven Margins. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 379–386, 2002.

Eneldo Loza Mencía and Johannes Fürnkranz. Pairwise learning of multilabel classifications with perceptrons. Technical Report TUD-KE-2007-05, Knowledge Engineering Group, TU Darmstadt, 2007.

Sang-Hyeun Park and Johannes Fürnkranz. Efficient pairwise classifciation. In *Proceedings of 18th European Conference on Machine Learning (ECML-07)*, pages 658–665, Warsaw, Poland, 2007.

David Price, Stefan Knerr, Leon Personnaz, and Gerard Dreyfus. Pairwise Neural Network Classifiers with Probabilistic Outputs. In *Advances in Neural Information Processing Systems*, volume 7, pages 1109–1116. The MIT Press, 1995.

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.

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.

Jan-Nikolas Sulzmann, Johannes Fürnkranz, and Eyke Hüllermeier. On pairwise naive bayes classifiers. In *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, pages 371–381, Warsaw, Poland, 2007.

Petroula Tsampouka and John Shawe-Taylor. Approximate maximum margin algorithms with rules controlled by the number of mistakes. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference on Machine Learning(ICML 2007)*, volume 227, pages 903–910, 2007.

Adriano Veloso, Wagner Meira Jr., Marcos André Gonçalves, and Mohammed Zaki. Multi-label lazy associative classification. In *Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 4702, pages 605–612, 2007.

Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10): 1338–1351, 2006.