Advances in Efficient Pairwise Multilabel Classification

Technical Report TUD-KE-2008-06

Eneldo Loza Mencía,
Sang-Hyeun Park,
Johannes Fürnkranz

Knowledge Engineering Group,
Technische Universität Darmstadt
{eneldo,park,juffi}@ke.tu-darmstadt.de

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Knowledge
Engineering

Contents

1 Introduction

Multilabel classification refers to the task of learning a function that maps instances $\bar{x} \in \mathcal{X}$ to label subsets $\lambda_{\bar{x}} \subset \mathcal{L}$, where $\mathcal{L} = \{\lambda_1, \ldots, \lambda_n\}$ is a finite set of predefined labels, typically with a small to moderate number of alternatives. Thus, in contrast to multiclass learning, alternatives are not assumed to be mutually exclusive, such that multiple labels may be associated with a single instance.

A prototypical application scenario for multilabel classification is the assignment of a set of keywords to a document, a frequently encountered problem in the text classification domain. With upcoming Web 2.0 technologies this domain is extended by a wide range of tag suggestion tasks (e.g. Tsoumakas et al., 2008; Katakis et al., 2008). This kind of problems are often associated with a large number of instances or classes which demand for an efficient processing. The Reuters-2000 dataset for instance is composed of over 800,000 documents and 103 classes (cf. Section 6.3), a benchmark extracted from the *del.icio.us* platform contains almost 1000 classes (Tsoumakas et al., 2008) and the EUR-Lex database consists of almost 4000 classes (cf. Section 6.3). Other tasks include protein classification and semantic multimedia annotation.

The predominant approach to multilabel classification is *binary relevance learning* (BR). In BR the problem is decomposed into several binary problems in the following way: for each class a binary classifier is trained to discriminate between examples of the class and the examples of the remaining classes. A different approach is to have a classifier for each possible pair of classes that is trained to distinguish only between these two classes. This approach is usually denominated *one-vs-one*, *round robin* or *pairwise classification* and has shown to achieve better predictive performance in the multiclass (Fürnkranz, 2002; Hsu and Lin, 2002) as well as in the multilabel case (Loza Mencía and Fürnkranz, 2008a; Fürnkranz et al., 2008).

While it has been shown that training a pairwise ensemble of classifiers is similarly efficient than training a BR ensemble (Fürnkranz, 2002; Loza Mencía and Fürnkranz, 2008a), the problem of evaluating a quadratic number of classifiers to produce a prediction remained. Our first attempts in efficient multilabel pairwise classification lead to the algorithm *MLPP* which uses the fast perceptron algorithm as base classifier and was succesfull in efficiently processing an already mentioned large Reuters benchmark, despite evaluating all base classifiers (Loza Mencía and Fürnkranz, 2008a). Although we were able to beat the competing fast MMP algorithm (Crammer and Singer, 2003) in terms of ranking performance and were competitive in training, the costs for testing were not similarly satisfactory.

Park and Fürnkranz (2007a) recently introduced a method named *QWeighted* for multiclass problems that intelligently selects only the base classifiers that are actually necessary to predict the top class. This reduced the evaluations needed from $n(n-1)/2$ to only $n \log(n)$ in practice, which is near the n evaluations processed by BR.

In this paper we introduce a novel algorithm which adapts the *QWeighted* method to the MLPP algorithm. In a nutshell, the adaption works as follows: instead of stopping when the top class is determined, we repeatedly apply *QWeighted* to the remaining classes until the final label set is predicted. In order to determine at which position to stop, we introduce an artificial label that indicates the boundary between positive and negative classes (Fürnkranz et al., 2008). Our experiments on a wide selection of multilabel datasets in terms of problem domain, number of classes and label density demonstrate that by applying these extensions to MLPP we are able to process such data in comparable time to the one-per-class approaches, while producing more accurate predictions.

Nevertheless, this novel algorithm still uses a quadratic number of base classifiers, i.e. the memory requirements grow quadratically to the number of classes. In (Loza Mencía and Fürnkranz, 2007) we analyzed for the first time a multilabel task with almost 4000 classes. Training MLPP for this problem would mean to maintain nearly 8,000,000 perceptrons in memory, which is almost impossible even for present-day computer systems. In (Loza Mencía and Fürnkranz, 2008c,b) we presented a modification of MLPP which represents the perceptrons as a linear combination of training examples and is therefore able to virtually store the 8 mio. perceptrons in memory. It even decreases the computational costs for some tasks. In this paper we compare the Dual MLPP algorithm to the previously mentioned *QWeighted* variant.

A related work on the issue of multilabel classification is the *HOMER* algorithm by Tsoumakas et al. (2008). The label set is organized through clustering into a hierarchy of labels. A multilabel classifier is then trained at each inner node. This reformulating leads to less complex problems at each inner node and hence allows to train the classifier ensemble more efficiently in terms of computations and memory.

This work is organized as follows: Section 2 defines the problem and describes the basic algorithms such as perceptrons, binary relevance and MLPP. Section 3 introduces *QWeighted* and the adaptation to multilabel, whereas DMLPP is presented in Section 4. In Section 5 we compare the time and space complexity of the different algorithms. Section 6 is dedicated to the experimental setup along with the used datasets and evaluation measures, the results are presented in Section 7. Section 8 provides a final discussion and concludes this paper.

2 Multilabel Classification

We represent an instance or object as a vector $\bar{x} = (x_1, \ldots, x_m)$ in a feature space $\mathcal{X} \subseteq \mathbb{R}^a$. Each instance $\bar{x}_i$ is assigned to a set of relevant labels P_i, a subset of the n possible classes $\mathcal{L} = \{\lambda_1, \ldots, \lambda_n\}$. For multilabel problems, the cardinality $|P_i|$ of the label sets is not restricted, whereas for binary problems $|P_i| = 1$. For the sake of simplicity we use the following notation for the binary case: we define $\mathcal{L} = \{1, -1\}$ as the set of classes so that each object $\bar{x}_i$ is assigned to a class $\lambda_i \in \{1, -1\}$, $P_i = \{\lambda_i\}$.

2.1 Perceptrons

We use the simple but fast perceptrons as base classifiers (Rosenblatt, 1958). As Support Vector Machines (SVM), their decision function describes a hyperplane that divides the a-dimensional space into two halves corresponding to positive and negative examples. We use a version that works without learning rate and threshold:

$$o(\bar{x}) = sgn(\bar{x} \cdot \bar{w}) \tag{2.1}$$

with the internal weight vector $\bar{w}$ and $sgn(t) = 1$ for $t \geq 0$ and -1 otherwise. If there exists a *separating hyperplane* between the two set of points, i.e. they are linearly separable, the following update rule provably finds it (cf., e.g., (Bishop, 1995)).

$$\alpha_i = (\lambda_i - o(\bar{x}_i)) \qquad \bar{w}_{i+1} = \bar{w}_i + \alpha_i \bar{x}_i \tag{2.2}$$

It is important to see that the final weight vector can also be represented as linear combination of the training examples:

$$\bar{w} = \sum_{i=1}^{m} \alpha_i \bar{x}_i \qquad o(\bar{x}) = sgn(\sum_{i=1}^{m} \alpha_i \cdot \bar{x}_i \bar{x}) \tag{2.3}$$

assuming m to be the number of seen training examples and $\alpha_i \in \{-1, 0, 1\}$. The perceptron can hence be coded implicitly as a vector of instance weights $\alpha = (\alpha_1, \ldots, \alpha_m)$ instead of explicitly as a vector of feature weights. This representation is denominated the dual form and is crucial for developing the memory efficient variant in Section 4. The main reason for choosing the perceptrons as our base classifier is because, contrary to SVMs, they can be trained efficiently in an incremental setting, which makes them particularly well-suited for large-scale classification problems such as the Reuters-RCV1 benchmark (Lewis et al., 2004), without forfeiting too much accuracy though SVMs find the *maximum-margin hyperplane* (Freund and Schapire, 1999; Crammer and Singer, 2003; Shalev-Shwartz and Singer, 2005).

In addition, important advancements were achieved in recent times trying to adapt the perceptron algorithm in order to maximize the margin of the separating hyperplane, without losing the advantages of simplicity and efficiency that characterize the perceptron algorithm (Li et al., 2002; Crammer et al., 2006; Khardon and Wachman, 2007; Tsampouka and Shawe-Taylor, 2007). The presented algorithms can easily be adapted in order to use these variants if desired.

2.2 Binary Relevance Ranking

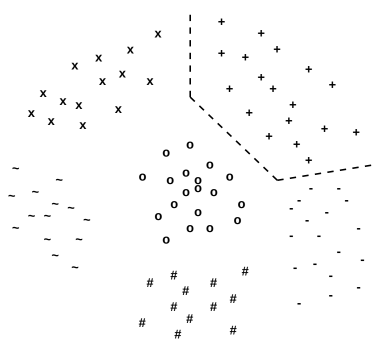
In order to provide a baseline and to show the efficiency of the pairwise approach, we compare our algorithms to the binary relevance (BR) or one-against-all (OAA) variant with perceptrons as base classifier.

In the binary relevance method, a multilabel training set with n possible classes is decomposed into n binary training sets of the same size that are then used to train n binary classifiers. So for each pair $(\bar{x}_i, P_i)$ in the original training set n different pairs $(\bar{x}_i, \lambda_{i_j})$ with $j = 1 \ldots n$ are generated as follows:

$$\lambda_{i_j} = \begin{cases} 1 & \lambda_j \in P_i \\ -1 & otherwise \end{cases} \tag{2.4}$$

A brief visual description of this technique is available in Figure 2.2.

Supposing we use perceptrons as base learners, n different o_j classifiers are trained in order to determine the relevance of λ_j. In consequence, the combined prediction of the binary relevance classifier for an instance $\bar{x}$ would be the set $\{\lambda_j \mid o_j(\bar{x}) = 1\}$. 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 2.1 the desired linear combination is the inner product $o'_j(\bar{x}) = \bar{x} \cdot \bar{w}_j$. So the result of the prediction is a vector $\bar{o}'(\bar{x}) = (\bar{x}\bar{w}_1, \ldots, \bar{x}\bar{w}_n)$ where component j corresponds to the relevance of class λ_j. Ties are broken randomly to not favor any particular class.


(a) *binary-relevance classification*
n classifiers, each separates one class from all other classes. Here: + against all other classes.

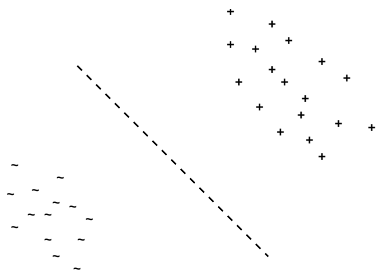

(b) *pairwise classification*
$\frac{n(n-1)}{2}$ classifiers, one for each pair of classes. Here: $+$ against $\sim$.

Figure 2.1: One-against-all and pairwise binarization.

2.3 Multiclass Multilabel Perceptrons

The Multiclass Multilabel Perceptron algorithm (MMP) by (Crammer and Singer, 2003) represents an extension to the simple BR approach. The difference is that the perceptrons ensemble training is done together, i.e. the perceptrons are interconnected. The aim of this is to achieve an ordering of relevant over irrelevant classes instead of the simple approach of a correct absolute ordering, i.e. relevant classes over a zero point and irrelevant classes below. However, the output is not longer a set of labels but a class relevance ranking. Several thresholding and calibration techniques to transform rankings into relevant label sets exist (Sebastiani, 2002), the method of Elisseeff and Weston (2001) being one of the most cited. We will see an effective calibration technique in Section 2.5.

The MMP algorithm was successfully applied to the large Reuters-RCV1 benchmark (Lewis et al., 2004), outperforming the simple BR approach. Refer to (Lewis et al., 2004) or (Loza Mencía and Fürnkranz, 2008a) for a more detailed description of the algorithm and performance comparison.

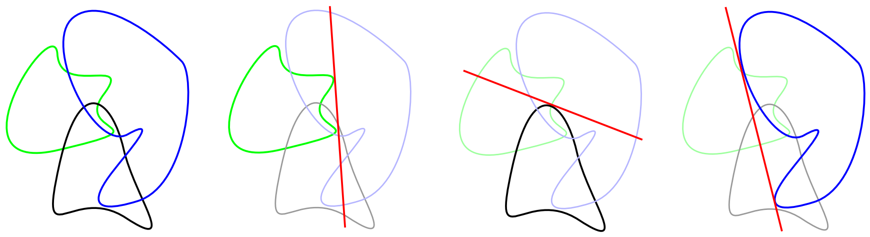


Figure 2.2: Subproblems in *binary relevance* for multilabel classification: original three-class problem (green, blue and black classes, shown as overlapping clouds in left picture) is divided into green vs. rest (second picture), black vs. rest (third) and blue vs. rest two-class subproblems. Separating hyperplanes, denoted by red lines, have to respect all examples (inside the clouds).

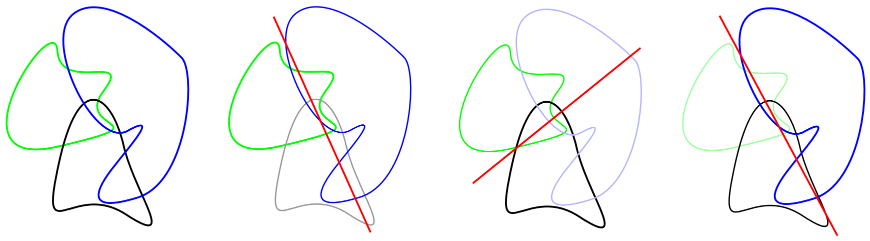


Figure 2.3: Subproblems in *pairwise multilabel classification*: original three-class problem is divided into green vs. blue (second picture, black examples are ignored), green vs. black (blue is ignored) and blue vs. black two-class subproblems. Separating hyperplanes have to respect only examples from two classes in contrast to BR in Figure 2.2.

```
**Require:** Training example pair $(\bar{x}, P)$, perceptrons $\{\bar{w}_{u,v} \mid u < v, \lambda_u, \lambda_v \in \mathcal{L}\}$
1: **for each** $(\lambda_u, \lambda_v) \in P \times N$ **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** $\{\bar{w}_{u,v} \mid u < v, \lambda_u, \lambda_v \in \mathcal{L}\}$          ▷ updated perceptrons
```

Figure 2.4: Pseudocode of the training method of the MLPP algorithm.

2.4 Multilabel Pairwise Perceptrons

In the pairwise binarization method, one classifier is trained for each pair of classes, i.e., a problem with n different classes is decomposed into $\frac{n(n-1)}{2}$ smaller subproblems. For each pair of classes (λ_u, λ_v), only examples belonging to either λ_u or λ_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., $(u, v) \in P \times N$ (cf. Figure 2.3). 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 2.4 shows the training algorithm in pseudocode. Of course MLPPs can also be trained incrementally because it inherits this property from the perceptron.

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 $\frac{n(n-1)}{2}$ perceptrons the classes are ordered according to their sum of votes. Ties are broken randomly in our case.

$o_{1,2} =$ 1	$o_{2,1} =$ -1	$o_{3,1} =$ -1	$o_{4,1} =$ -1	$o_{5,1} =$ -1
$o_{1,3} =$ 1	$o_{2,3} =$ 1	$o_{3,2} =$ -1	$o_{4,2} =$ -1	$o_{5,2} =$ -1
$o_{1,4} =$ 1	$o_{2,4} =$ 1	$o_{3,4} =$ 1	$o_{4,3} =$ -1	$o_{5,3} =$ -1
$o_{1,5} =$ 1	$o_{2,5} =$ 1	$o_{3,5} =$ 1	$o_{4,5} =$ 1	$o_{5,4} =$ -1
$v_1 =$ 4	$v_2 =$ 3	$v_3 =$ 2	$v_4 =$ 1	$v_5 =$ 0

Figure 2.5: MLPP voting: an example $\bar{x}$ is classified by all 10 base perceptrons $o_{i,j}, i \neq j$, $\lambda_i, \lambda_j \in \mathcal{L}$. Note the redundancy given by $o_{i,j} = -o'_{j,i}$. The last line counts the positive outcomes for each class.

Figure 2.5 shows a possible result of classifying the sample instance of Figure 2.6. Perceptron $o_{1,5}$ predicts (correctly) the first class, consequently λ_1 receives one vote and class λ_5 zero (denoted by $o_{1,5} = 1$ in the first and $o_{5,1} = -1$ in the last row). All 10 perceptrons (the values in the upper right corner can be deduced due to the symmetry property of the perceptrons) are evaluated though only six are 'qualified' since they were trained with the original example.

This may be disturbing at first sight since many 'unqualified' perceptrons are involved in the voting process: $o_{1,2}$ is asked though it cannot know anything relevant in order to determine if $\bar{x}$ belongs to λ_1 or λ_2 since it was neither trained on this example nor on other examples belonging simultaneously to both classes λ_1 and λ_2 (or to none of both). In the worst case the noisy votes concentrate on a single negative class, which would lead to misclassifications. But note that any class can at most receive $n - 1$ 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 would still receive at least $n - |P|$ and a negative at most $n - |P| - 1$ votes. Class λ_3 in Figure 2.5 is an example for this: It receives all possible noisy votes but still loses against the positive classes λ_1 and λ_2.

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 $\frac{2}{n}$ times the original size whereas binary relevance maintains the size. Typically, this goes hand in hand with an increase of the space where a separating hyperplane can be found. An intuitive visualization of this aspect can be found in Figure 2.1 for the multiclass case and in Figure 2.3 for the multilabel case, in contrast to the BR binarization depicted in Figure 2.2. A simple example also illustrates this: imagine you repeatedly insert points around two points on a line. The distance between the two sets will inevitably monotonically decrease with increasing number of points. Thus it is very likely for a subproblem to have a larger margin than the full problem.

Particularly in the case of text classification the obtained benefit clearly exists. An evaluation of the pairwise approach on the Reuters-RCV1 corpus (cf. Section 6.3), which contains over 100 classes and 800,000 documents, showed a significant and substantial improvement over the MMP method (Loza Mencía and Fürnkranz, 2008a).

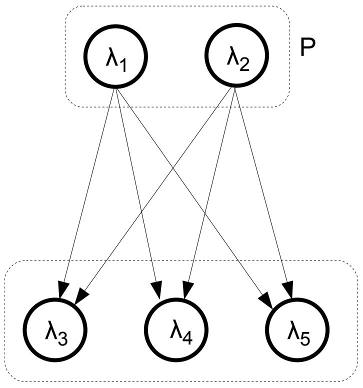


Figure 2.6: MLPP training: training example $\bar{x}$ belongs to $P_{\bar{x}} = \{\lambda_1, \lambda_2\}$, $N_{\bar{x}} = \{\lambda_3, \lambda_4, \lambda_5\}$ are the irrelevant classes, the arrows represent the trained perceptrons $\bar{w}_{1,3}, \bar{w}_{1,4}, \bar{w}_{1,5}, \bar{w}_{2,3}, \bar{w}_{2,4}, \bar{w}_{2,5}$.

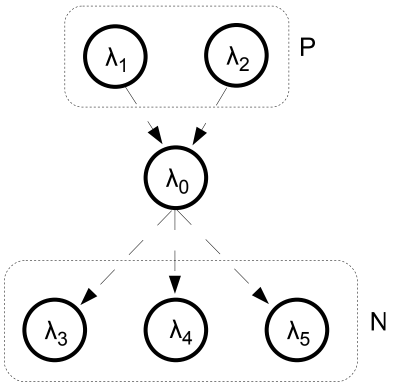


Figure 2.7: calibration: introducing virtual label λ_0 that separates P an N. Perceptrons $\bar{w}_{1,0}, \bar{w}_{2,0}, \bar{w}_{0,3}, \bar{w}_{0,4}, \bar{w}_{0,5}$ are additionally trained.

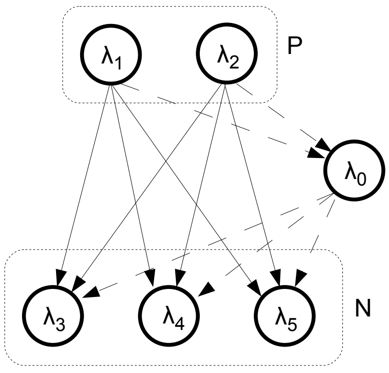


Figure 2.8: CMLPP training: the complete set of trained perceptrons.

2.5 Calibrated Label Ranking

To convert the resulting ranking of labels into a multilabel prediction, we use the *calibrated label ranking* approach (Brinker et al., 2006; Fürnkranz et al., 2008). This technique avoids the need for learning a threshold function for separating relevant from irrelevant labels, which is often performed as a post-processing phase after computing a ranking of all possible classes. The key idea is to introduce an artificial *calibration label* λ_0, which represents the split-point between relevant and irrelevant labels. Thus, it is assumed to be preferred over all irrelevant labels, but all relevant labels are preferred over λ_0. This introduction of an additional label during training is depicted in Figure 2.7, the combination with the normal pairwise base classifiers is shown in Figure 2.8.

As it turns out, the resulting n additional binary classifiers $\{o_{i,0} \mid i = 1 \ldots n\}$ are identical to the classifiers that are trained by the binary relevance approach. Thus, each classifier $o_{i,0}$ is trained in a one-against-all fashion by using the whole dataset with $\{\bar{x} \mid i \in P_{\bar{x}}\} \subseteq \mathcal{X}$ as positive examples and $\{\bar{x} \mid i \in N_{\bar{x}}\} \subseteq \mathcal{X}$ as negative examples. At prediction time, we will thus get a ranking over $n+1$ labels (the n original labels plus the calibration label). Then, the projection of voting aggregation of pairwise perceptrons with a calibrated label to a multilabel output is quite straight-forward:

$$\hat{P} = \{\lambda \in \mathcal{L} \mid v(\lambda) > v(\lambda_0)\}$$

where $v(c)$ is the amount of votes class c has received.

We denote the MLPP algorithm adapted in order to support the calibration technique as CMLPP. This algorithm was again applied to the large Reuters-RCV1 corpus and to other smaller datasets presented also in Section 6.3, outperforming the binary relevance and MMP approach. For further details please refer to Fürnkranz et al. (2008).

3 Quick Weighted Voting

As already seen, the quadratic number of base classifier does not seem to be a serious drawback for training MLPP and also CMLPP. However, at prediction time it is still necessary to evaluate a quadratic number of base classifier. Two approaches to overcome this problem for multiclass and for multilabel task are presented in the following.

3.1 QWeighted for Multiclass Classification

For the multiclass case, the simple voting strategy, which is applied often to combine the predictions of pairwise classifiers to one multiclass classification result, can be computed efficiently with the Quick Weighted Voting algorithm (*QWeighted*) (Park and Fürnkranz, 2007a). Instead of the evaluation of the quadratic number of all pairwise perceptrons, it is possible to evaluate a smaller subset of it in order to compute the class with the highest accumulated voting mass.

During a voting procedure there exist many situations where particular classes can be excluded from the set of possible top rank classes, even if they reach the maximal voting mass in the remaining evaluations. Its main idea can be described in a simple example: Given n classes with $n > j$, if class λ_a has received more than $n-j$ votes and class λ_b lost j votings, it is impossible for λ_b to achieve a higher total voting mass than λ_a. Thus further evaluations with λ_b can be safely ignored.

Pairwise classifiers will be selected depending on a *loss* value, which is the amount of potential voting mass that a class has *not* received. More precisely, the loss l_i of a class λ_i is defined as $l_i := p_i - v_i$, where p_i is the number of evaluated incident classifiers of λ_i and v_i is the current vote amount of λ_i. Obviously, the loss will begin with a value of zero and is monotonically increasing. The class with the current minimal loss is one of the top candidates for the top rank class.

```
**Require:** Testing example $\bar{x}$, perceptrons $\{\bar{w}_{u,v} \mid u < v, \lambda_u, \lambda_v \in \mathcal{L}\}$
1: **while** $\lambda_{top}$ not determined **do**
2:     $\lambda_a \leftarrow \lambda_i \in \mathcal{L}$ with minimal $l_i$
3:     $\lambda_b \leftarrow \lambda_j \in \mathcal{L} \backslash \{\lambda_a\}$ with minimal $l_j$ & $\bar{w}_{a,b}$ not yet evaluated
4:     **if** no $\lambda_b$ exists **then**
5:         $\lambda_{top} \leftarrow \lambda_a$
6:     **else**
7:         $v_{ab} \leftarrow \text{EVALUATE}(\bar{w}_{a,b})$
8:         $l_a \leftarrow l_a + (1 - v_{ab})$
9:     $l_b \leftarrow l_b + v_{ab}$
```

Figure 3.1: Pseudocode of the Quick Weighted Voting algorithm (Multiclass classification prediction).

First the pairwise classifier $o_{a,b}$, in our case the perceptron $\bar{w}_{a,b}$, will be selected for which the losses l_a and l_b of the relevant classes λ_a and λ_b are minimal, provided that the classifier $o_{a,b}$ has not yet been evaluated. In the case of multiple classes that have the same minimal loss, there exists no further distinction, and we select a class randomly from this set. Then, the losses l_a and l_b will be updated based on the evaluation returned by $o_{a,b}$ (recall that v_{ab} is interpreted as the amount of the voting mass of the classifier $o_{a,b}$ that goes to class λ_a and $1 - v_{a,b}$ is the amount that goes to class λ_b). These two steps will be repeated until all classifiers for the class λ_m with the minimal loss has been evaluated. Thus the current loss l_m is the correct loss for this class. As all other classes already have a greater loss, λ_m is the correct *top rank* class.

Theoretically, a minimal number of comparisons of $n - 1$ is possible (*best case*). The *worst case*, on the other hand, is still $n(n-1)/2$ comparisons, which can, e.g., occur if all pairwise classifiers classify randomly with a probability of 0.5. In practice, the number of comparisons will be somewhere between these two extremes, depending on the nature of the problem.

3.2 QWeighted for Multilabel Classification

A simple adaptation of *QWeighted* to multilabel classification is to repeat the process. We can compute the top class λ_{top} using *QWeighted* and remove this class from $\mathcal{L}$ and repeat this step, until the returned class is the artificial label λ_0. This adaptation uses two simple extensions of the original algorithm: (1) The information about which pairwise perceptrons have been evaluated and their results are carried through the iterations so that no pairwise perceptron is evaluated more than once. (2) By using the calibrated label ranking approach we know beforehand that at some point the vote amount of the artificial label has to be computed. So, in hope for a *better* starting distribution of votes, all incident classifiers $o_{i,0}$ respectively $\bar{w}_{i,0}$ of the artificial label are evaluated explicitly before employing iterated *QWeighted*.

Note that the effectiveness of this testing procedure is highly influenced by the relation of average number of relevant labels to total number of labels. We can expect a high reduction of pairwise comparisons, if the above relation is relatively small, which holds for the most real-world multilabel datasets. We will refer to this procedure in the following text as QCMLPP1.

In addition to this straight-forward adaptation, we considered also an slightly improved variant (QCMLPP2). In retrospect, QCMLPP1 computes a partial ranking of classes down to the calibrated label. That means, that for all relevant labels all their incident classifiers are evaluated. It neglects the fact that for multilabel classification the information that a particular class *is ranked above* the calibrated label is sufficient, rather than *to which amount*. QCMLPP2 works in the same way as QCMLPP1 except that it stops the evaluation of the current top rank λ_t if it already received a higher voting mass than the calibrated label. The class λ_t is not automatically removed from the set of labels as in QCMLPP1, since further evaluations for the computation of other classes can occur, but it can not be selected as a new top rank candidate.

We are currently investigating further variants for improving the performance. For example, different search heuristics based on other losses than the number of “lost games“ are imaginable. Furthermore, the selection of the two next classes for evaluation can also be varied, i.e. by pairing the “best“ and the “worst“ class in the next iteration instead of the two currently best classes. In addition, we are working on the derivation of formal complexity bounds to strengthen the *QWeighted* approach.

4 Dual Multilabel Pairwise Perceptrons

Perhaps the hardest problem in the context of pairwise multilabel classification is that even if training and testing can be performed efficiently, one still has to store a number of classifiers that is quadratic in the number of potential labels. Even on modern computers with a large memory this problem becomes unsolvable for a high number of classes. Consider for instance the EUR-Lex database introduced in Section 6.4. For its EUROVOC setting with almost 4000 classes the use of MLPP would mean maintaining approximately 8,000,000 perceptrons in memory.

In order to circumvent this obstacle we reformulate the MLPP ensemble of perceptrons in dual form as we did with one single perceptron in Equation 2.3. In contrast to MLPP, the training examples are thus required and have to be kept in memory in addition to the associated weights, as a base perceptron is now represented as $\bar{w}_{u,v} = \sum_{i=1}^{m} \alpha_{u,v}^{t} \bar{x}_i$. This makes an additional loop over the training examples inevitable every time a prediction is demanded. But fortunately it is not necessary to recompute all $\bar{x}_i \bar{x}$ for each base perceptron since we can reuse them by iterating over the training examples in the outer loop, as can be seen in the following equations:

$$\begin{aligned}
\bar{w}_{1,2}\bar{x} &= \alpha_{1,2}^{1}\bar{x}_1\bar{x} + \alpha_{1,2}^{2}\bar{x}_2\bar{x} + \ldots + \alpha_{1,2}^{m}\bar{x}_m\bar{x} \\
\bar{w}_{1,3}\bar{x} &= \alpha_{1,3}^{1}\bar{x}_1\bar{x} + \alpha_{1,3}^{2}\bar{x}_2\bar{x} + \ldots + \alpha_{1,3}^{m}\bar{x}_m\bar{x} \\
&\vdots \\
\bar{w}_{1,n}\bar{x} &= \alpha_{1,n}^{1}\bar{x}_1\bar{x} + \alpha_{1,n}^{2}\bar{x}_2\bar{x} + \ldots + \alpha_{1,n}^{m}\bar{x}_m\bar{x} \\
\bar{w}_{2,3}\bar{x} &= \alpha_{2,3}^{1}\bar{x}_1\bar{x} + \alpha_{2,3}^{2}\bar{x}_2\bar{x} + \ldots + \alpha_{2,3}^{m}\bar{x}_m\bar{x} \\
&\vdots
\end{aligned} \tag{4.1}$$

By advancing column by column it is not necessary to repeat the dot products computations, however it is necessary to store the intermediate values, as can also be seen in the pseudocode of the training and prediction phases in Figures 5.1 and 5.2. Note also that the algorithm preserves the property of being incrementally trainable. We denote this variant of training the pairwise perceptrons the *dual multilabel pairwise perceptrons* algorithm (DMLPP).

In addition to the savings in memory and run-time, analyzed in detail in Section 5, the dual representation allows for using the kernel trick, i.e. to replace the dot product by a kernel function, in order to be able to solve originally not linearly separable problems. However, this is not necessary in our case since text problems are in general linearly separable.

Note also that the pseudocode needs to be slightly adapted when the DMLPP algorithm is trained in more than one epoch, i.e. the training set is presented to the learning algorithm more than once. It is sufficient to modify the assignment in line 8 in Figure 5.1 to an additive update $\alpha_{u,v}^{t} = \alpha_{u,v}^{t} + 1$ for a revisited example $\bar{x}_t$. This setting is particularly interesting for the dual variant since, when the training set is not too big, memorizing the inner products can boost the subsequent epochs in a substantial way, making the algorithm interesting even if the number of classes is small.

We are currently investigating hybrid variants to further reduce the computational complexity. The idea is to use a different formulation in training than in the prediction phase depending on the specific memory and runtime requirements of the classification task. In order e.g. to combine the advantage of MLPP during training and DMLPP during predicting on the subject matter subproblem, we could train the classifier as in the MLPP (with the difference of iterating over the base classifier first and than over the instances instead of reversely so that only one perceptron has to remain in memory) and than convert it by means of the collected information during training the perceptrons to the dual representation.

Other further steps include to adapt the algorithm in order to use the calibration technique described in Section 2.5 and to investigate the usage of *QWeighted* to further reduce computational costs.

5 Computational Complexity

The notation used in this section is the following: n denotes the number of possible classes, d the average number of relevant classes per instance in the training set, a the number of attributes and a' the average number of attributes not zero (size of the sparse representation of an instance), and m denotes the size of the training set. For each complexity we will give an upper bound O in Landau notation. We will indicate the runtime complexity in terms of real value additions and multiplications ignoring operations that have to be performed by all algorithms such as sorting or internal real value operations. Additionally, we will present the complexities per instance since all algorithms are incrementally trainable.

- **Space Requirements**
  BR and MMP follow an one model per class approach, so they have to keep the same amount of perceptrons in memory, leading to $O(n \cdot a)$ memory space. In contrast the non-dual pairwise approaches require one perceptron for each of the $\frac{n(n-1)}{2}$ pairs of classes, hence we need $O(n^2 a)$ memory. In addition, the calibrated versions require an overhead of n perceptrons for the comparisons with the artificial label. The DMLPP algorithms keeps the whole training set in memory, and additionally requires for each training example $\bar{x}$ access to the weights of all class pairs $P \times N$. Furthermore, it has to intermediately store the resulting scores for each base perceptron during prediction, hence the complexity is $O(mdn + ma' + n^2) = O(m(dn + a') + n^2)$.[1] We can see that (QC)MLPP is applicable especially if the number of classes is low and the number of examples high, whereas DMLPP is suitable when the number of classes is high, however it does not handle huge training sets very well.

- **Training**
  For processing one training example, n dot product have to be computed by BR, plus at most the same amount if there was a prediction error. The non-dual MLPPs require $O(dn)$ dot products, one for each associated perceptron. Assuming that a dot product computation costs $O(a')$, we obtain a complexity of $O(dna')$ per training example. Thus, assuming similar loss rates, the pairwise training will be only on average d resp. $d + 1$ for the calibrated version slower than the BR or MMP algorithm although training a quadratic number of base classifier.

  DMLPP spends m dot product computations. In addition the summation of the scores costs $O(dn)$ per training instance, leading to $O(m(dn + a'))$ operations. It is obvious that (QC)MLPP has a clear advantage over DMLPP in terms of training time, unless n is of the order of magnitude of m or the model is trained over several epochs, as already outlined in the previous Section 4

- **Prediction**
  During prediction the one-per-class approaches achieve $O(na')$ computations for one instance. For the pairwise approach without the usage of *QWeighted* all perceptrons have to be evaluated, leading to $O(n^2 a')$ computations. The same upper bound holds analytically for QCMLPP, but as previous experiments have shown for the multiclass case, *QWeighted* (QW) reduces the amount of required base classifier evaluations from $\frac{n(n-1)}{2}$ to $n \log(n)$ in practice (Park and Fürnkranz, 2007a). It is easy to see that the multilabel adaptations of *QWeighted* are upper bounded by $n + d \cdot O(\text{QW})$, since n classifiers involving the calibrated class are always evaluated and *QWeighted* is applied d times to compute the relevant labels in average. Assuming that the average runtime of *QWeighted* is $n \log(n)$, we can expect an average runtime of $n + dn \log(n)$ for the adaption to the multilabel case. Our experimental evaluations in Section 7 will confirm this observation.

  The dual variant again iterates over all training examples and associated weights, hence the complexity is $O(m(dn + a'))$. At this phase DMLPP benefits from the linear dependence of the number of classes in contrast to the quadratic relationship of the MLPP. Roughly speaking the breaking point when DMLPP is faster in prediction is approximately when the square of the number of classes is clearly greater than the number of training documents. Of course this does not hold in the same degree for the comparison with the *QWeighted* variant. We can find a similar trade-off for the memory requirements that holds in general for the comparison between dual and non-dual variants, with the difference that the factor between sparse and total number of attributes becomes more important, leading earlier to the breaking point when the sparseness is high.

[1] Note that we do not estimate d as $O(n)$ since both values are not of the same order of magnitude in practice. For the same reason we distinguish between a and a' since particularly in text classification both values are not linked: a text document often turns out to employ around 100 different words whereas the size of the vocabulary of a the whole corpus can easily reach 100,000 words (although this number is normally reduced by feature selection).

Table 5.1: Computational complexity given as upper bounds of number of addition and multiplication operations, for each instance. n: *#classes*, d: *avg. #labels per instance*, m: *#training examples*, a: *#attributes*, a': *#attributes*$\neq 0$.

	training time	prediction time	memory requirement
MMP/ BR	$O(na')$	$O(na')$	$O(na)$
MLPP	$O(dna')$	$O(n^2a')$	$O(n^2a)$
QCMLPP	$O(dna')$	$\sim na' + dn\log(n)\, a'$	$O(n^2a)$
DMLPP	$O(m(dn+a'))$	$O(m(dn+a'))$	$O(m(dn+a')+n^2)$

```
Require: New training example pair $(\bar{x}_m, P)$,
    training examples $\bar{x}_1 \ldots \bar{x}_{m-1}$,
    weights $\{\alpha^i_{u,v} \mid \lambda_u, \lambda_v \in \mathcal{L}, 0 < i < m\}$
1: for each $\bar{x}_i \in \bar{x}_1 \ldots \bar{x}_{m-1}$ do
2:     $p_i \leftarrow \bar{x}_i \cdot \bar{x}_m$
3:     for each $(\lambda_u, \lambda_v) \in P \times N$ do
4:         if $\alpha^i_{u,v} \neq 0$ then
5:             $s_{u,v} \leftarrow s_{u,v} + \alpha^i_{u,v} \cdot p_t$        ▷ note that $s_{u,v} = -s_{v,u}$
6: for each $(\lambda_u, \lambda_v) \in P \times N$ do
7:     if $s_{u,v} < 0$ then
8:         $\alpha^m_{u,v} \leftarrow 1$        ▷ note that $\alpha_{u,v} = -\alpha_{v,u}$
9: return $\{\alpha^m_{u,v} \mid (\lambda_u, \lambda_v) \in P \times N\}$        ▷ return new weights
```

Figure 5.1: Pseudocode of the training method of the DMLPP algorithm.

```
Require: example $\bar{x}$ for classification,
    training examples $\bar{x}_1 \ldots \bar{x}_{m-1}$,
    weights $\{\alpha^i_{u,v} \mid \lambda_u, \lambda_v \in \mathcal{L}, 0 < i < m\}$
1: for each $\bar{x}_i \in \bar{x}_1 \ldots \bar{x}_m$ do
2:     $p \leftarrow \bar{x}_i \cdot \bar{x}$
3:     for each $(\lambda_u, \lambda_v) \in P_i \times N_t$ do
4:         if $\alpha^i_{u,v} \neq 0$ then
5:             $s_{u,v} \leftarrow s_{u,v} + \alpha^i_{u,v} \cdot p$
6: for each $(\lambda_u, \lambda_v) \in \mathcal{L} \times \mathcal{L}$ do
7:     if $u \neq v \vee s_{u,v} > 0$ then
8:         $v_u \leftarrow v_u + 1$
9: return voting $\bar{v} = (v_1, \ldots, v_{|\mathcal{L}|})$        ▷ return voting
```

Figure 5.2: Pseudocode of the prediction phase of the DMLPP algorithm.

A compilation of the analysis can be found in Table 5.1, together with the complexities of MMP and BR. A more detailed comparison between MMP and MLPP is available from Loza Mencía and Fürnkranz (2008a). Note that the stated prediction time for QCMLPP in the table is not an analytical complexity bound like the others, it is an empirically estimated value.

In summary, it can be stated that the dual form of the MLPP balances the relationship between training and prediction time by increasing training and decreasing prediction costs, and especially benefits from a decreased prediction time and memory savings when the number of classes is large, which was the main obstacle to applying the pairwise approach to large scale problems in terms of classes. In contrast, at first view QCMLPP does not benefit analytically from the *QWeighted* voting, but there is empirical evidence for a clear improvement compared to the full voting. There is no disadvantage of using QCMLPP instead of CMLPP unless a more fine-grained distinction between classes than relevant-irrelevant is required.

6 Experimental Setup

6.1 Multilabel Evaluation Measures

There is no generally accepted procedure for evaluating multilabel classifications. Our approach is to consider a multilabel classification problem as a meta-classification problem where the task is to separate the set of possible labels into relevant labels and irrelevant labels. Let $\hat{P}_{\bar{x}}$ denote the set of labels predicted by the multilabel classifier and $\hat{N}_{\bar{x}} = \mathcal{L} \setminus \hat{P}_{\bar{x}}$ the set of labels that are not predicted by the classifier. Thus, we can, for each individual instance $\bar{x}$, compute a two-by-two confusion matrix $C_{\bar{x}}$ of relevant/irrelevant vs. predicted/not predicted labels:

$C_{\bar{x}}$	predicted	not predicted	
relevant	$\lvert P_{\bar{x}} \cap \hat{P}_{\bar{x}} \rvert$	$\lvert P_{\bar{x}} \cap \hat{N}_{\bar{x}} \rvert$	$\lvert P_{\bar{x}} \rvert$
irrelevant	$\lvert N_{\bar{x}} \cap \hat{P}_{\bar{x}} \rvert$	$\lvert N_{\bar{x}} \cap \hat{N}_{\bar{x}} \rvert$	$\lvert N_{\bar{x}} \rvert$
	$\lvert \hat{P}_{\bar{x}} \rvert$	$\lvert \hat{N}_{\bar{x}} \rvert$	$\lvert \mathcal{L} \rvert$

From such a confusion matrix $C_{\bar{x}}$, we can compute several well-known measures:

- The *Hamming loss* (HAMLOSS) computes the percentage of labels that are misclassified, i.e., relevant labels that are not predicted or irrelevant labels that are predicted. This basically corresponds to the error in the confusion matrix. In order to be consistent with the following measures, we report 1 minus the Hamming loss, which corresponds to the accuracy on the predicted labels.

$$\textsc{HamLoss}(C_{\bar{x}}) \overset{\text{def}}{=} 1 - \frac{1}{|\mathcal{L}|}\left|\hat{P}_{\bar{x}} \triangle P_{\bar{x}}\right| \tag{6.1}$$

  The operator $\triangle$ denotes the symmetric difference between two sets and is defined as $A \triangle B \overset{\text{def}}{=} (A \setminus B) \cup (B \setminus A)$, i.e. $\hat{P}_{\bar{x}} \triangle P_{\bar{x}}$ has all labels that only appear in one of the two sets.

- *Precision* (PREC) computes the percentage of predicted labels that are relevant, *recall* (REC) computes the percentage of relevant labels that are predicted, and the *F1*-measure is the harmonic mean between the two.

$$\textsc{Prec}(C_{\bar{x}}) \overset{\text{def}}{=} \frac{|\hat{P}_{\bar{x}} \cap P_{\bar{x}}|}{|\hat{P}_{\bar{x}}|} \qquad \textsc{Rec}(C_{\bar{x}}) \overset{\text{def}}{=} \frac{|\hat{P}_{\bar{x}} \cap P_{\bar{x}}|}{|P_{\bar{x}}|} \tag{6.2}$$

$$\text{F1}(C_{\bar{x}}) \overset{\text{def}}{=} \frac{2}{\frac{1}{\textsc{Rec}(C_{\bar{x}})} + \frac{1}{\textsc{Prec}(C_{\bar{x}})}} = \frac{2\,\textsc{Rec}(C_{\bar{x}})\,\textsc{Prec}(C_{\bar{x}})}{\textsc{Rec}(C_{\bar{x}}) + \textsc{Prec}(C_{\bar{x}})} \tag{6.3}$$

To average these values, we compute a micro-average over all values in a test set, i.e., we add up the confusion matrices $C_{\bar{x}}$ for examples in the test set and compute the measure from the resulting confusion matrix. Thus, for any given measure f, the average is computed as:

$$f_{\text{avg}} = f(\sum_{i=1}^{n} C_{x_i}) \tag{6.4}$$

If we use a cross-validation, the measures $f_{\text{avg}_j}, j = 1 \dots q$ are averaged over all q folds.

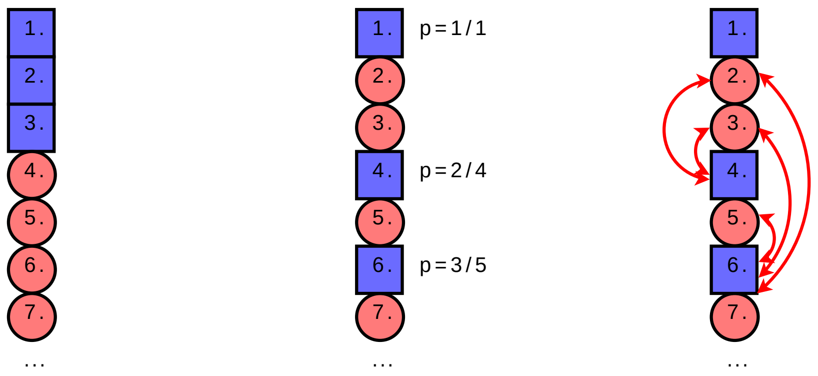


Figure 6.1: Diagrams of predicted label rankings. Blue rectangles denote real positive classes, red ones negatives. First ranking: perfect classification, all relevant classes are ranked over irrelevant ones, RankLoss = 0, AvgP = 1. Second and third ranking: classes on position 4 and 6 are misplaced, thus 5 of 12 possible pairs of labels are not correctly ordered, top class is correct, RankLoss = 5/12, AvgP = 21/30 = 0.7.

6.2 Ranking Loss Functions

Some previous works on multilabel classification, in particular the work on MMPs to which we compare, evaluated the ranking performance and neglected the calibration. For this reason, we also employ two previously used ranking measures (Crammer and Singer, 2003). Using these measures allows us to compare the ranking performance of our calibrated methods to previous methods that do not use calibration and can not be evaluated with the above multilabel loss functions. The different losses we used are computed comparing the ranking with the true set of relevant classes, each of them focusing on different aspects.

We use the following notational conventions: For a given instance x, let $r(\lambda_i)$ denote the position of λ_i in the predicted ranking (with the calibrating label λ_0 being removed from the ranking) and $r^{-1}(i)$ the label λ that is assigned to the position i.

- *average precision* (AvgP) 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.

$$\textsc{AvgP}(P_{\bar{x}}, r) \overset{\text{def}}{=} \frac{1}{|P_{\bar{x}}|} \sum_{\lambda \in P_{\bar{x}}} \frac{|\{\lambda' \in P_{\bar{x}} | r(\lambda') \leq r(\lambda)\}|}{r(\lambda)} \tag{6.5}$$

- The *ranking loss* (RankLoss) computes the average fraction of pairs of labels which are not correctly ordered:

$$\textsc{RankLoss}(P_{\bar{x}}, r) \overset{\text{def}}{=} \frac{\left|\{(\lambda, \lambda') \in P_{\bar{x}} \times N_{\bar{x}} \mid r(\lambda) > r(\lambda')\}\right|}{|P_{\bar{x}}||N_{\bar{x}}|} \tag{6.6}$$

These measures are computed for each example and then averaged over all examples. A visualization of some example computations of the losses is shown in Figure 6.1.

6.3 Standard Benchmark Datasets

The datasets that were included in the experimental setup cover three application areas in which multilabeled data are frequently observed: *text categorization* (the *Reuters-RCV1* and *Reuters-21578* datasets), *image classification* (the *scene* dataset) and *bioinformatics* (the *yeast* dataset).[1] The *Reuters Corpus Volume I (Reuters-RCV1)* is one of the most widely used test collection for text categorization research. It contains 804,414 newswire documents, which we 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). We used the token files of Lewis et al. (2004), which are already word-stemmed and stop word reduced. However we repeated the stop word reduction as we experienced that there were still a few

[1] The *Reuters-RCV1* dataset is available from `http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm` (Lewis et al., 2004), the *Reuters-21578* dataset from `http://www.daviddlewis.com/resources/testcollections/reuters21578/`, and the *yeast* and *scene* datasets from `http://mlkd.csd.auth.gr/multilabel.html`.

Table 6.1: Statistics of datasets. The attribute number in parenthesis denotes the actual used number of features, i.e. for *scene* and *yeast* the number of features after adding the pairwise products and for the text collections the amount after feature selection. *Label density* indicates the average number of labels per instance d relative to the total number of classes n, and *distinct* counts the distinct label-sets found in the dataset $|\{P_i \mid i = 0 \ldots m\}|$.

dataset name	domain	#instances m	#numeric attributes a	#labels n	avg. label-set size d	density $\frac{d}{n}$	distinct
scene	multimedia	2407	294 (86732)	6	1.074	17.9 %	15
yeast	biology	2417	103 (10712)	14	4.237	30.3 %	198
reuters21578	text	11367	21474 (10000)	120	1.258	1.0 %	533
rcv1-v2	text	804414	231188 (25000)	101	2.880	2.9 %	1028
EUR-Lex *subject matter*	text	19596	166448 (5000)	201	2.210	1.1 %	2540
EUR-Lex *directory code*	text	19596	166448 (5000)	412	1.292	0.3 %	1648
EUR-Lex EUROVOC	text	19596	166448 (5000)	3993	5.317	0.1 %	16871

occurrences. The 25,000 most frequent features on the training set were selected and weighted with TF-IDF weights (Salton and Buckley, 1988). We did not restrict the set of 103 categories although one class does not contain any examples in the training set.

We also experimented with the older *Reuters-21578* corpus (Lewis, 1997), which has 11,367 examples and 120 possible labels. Through similar pre-processing as in the *Reuters-RCV1* dataset, we obtained 10,000 features for this dataset.

The learning task in the *yeast* gene functional multiclass classification problem is to associate genes with a subset of 14 functional classes from the Comprehensive Yeast Genome Database of the Munich Information Center for Protein Sequences[2]. Each of 2417 genes is represented with 103 features. In previous experiments (Loza Mencía and Fürnkranz, 2008a), we found that even the pairwise problems are hard to separate with a linear classifier (much more so in the binary relevance setting). Thus, in this set of experiments, we added all pairwise feature products to the original feature representation, in order to simulate a quadratic kernel function.

The task in the *scene* dataset (Boutell et al., 2004) is to recognize which of six possible scenes (*beach, sunset, field, fall foliage, mountain, urban*) can be found in a 2407 pictures. Many pictures contain more than one scene. For each image, spatial color moments are used as features. Each picture is divided into 49 blocks using a 7×7 grid. A picture is then represented using the mean and the variance of each color band of each block, i.e., using a total of $2 \times 3 \times 7 \times 7 = 294$ features. Like in the *Yeast* dataset, we enriched the feature set with all pairwise feature products.

6.4 The EUR-Lex Repository

The DMLPP algorithm arose from the need to be able to process the EUR-Lex repository (Loza Mencía and Fürnkranz, 2007, 2008c,b). The EUR-Lex text collection is a collection of documents about European Union law. It contains many several different types of documents, including treaties, legislation, case-law and legislative proposals, which are indexed according to several orthogonal categorization schemes to allow for multiple search facilities. The most important categorization is provided by the EUROVOC descriptors, which is a topic hierarchy with almost 4000 categories regarding different aspects of European law.

In addition to the challenging task of handling such great amount of classes, this document collection provides an excellent opportunity to study text classification techniques for several reasons:

- it contains multiple classifications of the same documents, making it possible to analyze the effects of different classification properties using the same underlying reference data without resorting to artificial or manipulated classifications,
- the overwhelming number of produced documents make the legal domain a very attractive field for employing supportive automated solutions and therefore a machine learning scenario in step with actual practice,
- the documents are available in several European languages and are hence very interesting e.g. for the wide field of multi- and cross-lingual text classification,
- and, finally, the data is freely accessible (at `http://eur-lex.europa.eu/`)

[2] `http://mips.gsf.de/genre/proj/yeast/`

Title and reference

Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs

Classifications

EUROVOC descriptor

- *data-processing law, computer piracy, copyright, software, approximation of laws*

Directory code

- 17.20.00.00 *Law relating to undertakings /* Intellectual property law

Subject matter

- *Internal market, Industrial and commercial property*

Text

COUNCIL DIRECTIVE of 14 May 1991 on the legal protection of computer programs (91/250/EEC)

THE COUNCIL OF THE EUROPEAN COMMUNITIES,

Having regard to the Treaty establishing the European Economic Community and in particular Article 100a thereof, . . .

Figure 6.2: Excerpt of a EUR-Lex sample document with the CELEX ID 31991L0250. The original document contains more meta-information. We trained our classifiers to predict the EUROVOC descriptors, the directory code and the subject matters based on the text of the document.

We retrieved the HTML versions with bibliographic notes recursively from all (non empty) documents in the English version of the *Directory of Community legislation in force*[3], in total 19,596 documents. Only documents related to secondary law (in contrast to primary law, the constitutional treaties of the European Union) and international agreements were included. The legal form of the included acts are mostly *decisions* (8,917 documents), *regulations* (5,706), *directives* (1,898) and *agreements* (1,597). The bibliographic notes of the documents contain information such as dates of effect and validity, authors, relationships to other documents and classifications. The classifications include the assignment to several EUROVOC descriptors, directory codes and subject matters, hence all classifications are multilabel ones. EUROVOC is a multilingual thesaurus providing a controlled vocabulary[4]. Documents in the documentation systems of the EU are indexed using this thesaurus. The directory codes are classes of the official classification hierarchy of the *Directory of Community legislation in force*. It contains 20 chapter headings with up to four sub-division levels. The high number of 3,993 different EUROVOC descriptors were identified in the retrieved documents, each document is associated to 5.37 descriptors on average. In contrast there are only 201 different subject matters appearing in the dataset, with a mean of 2.23 labels per document, and 412 different directory codes, with a label set size of on average 1.29. Note that for the directory codes we used only the assignment to the leaf category as the parent nodes can be deduced from the leaf node assignment. For the document in Figure 6.2 this would mean a set of labels of $\{17.20\}$ instead of $\{17, 17.20\}$.

Figure 6.2 shows an excerpt of a sample document with all information that has not been used removed. The full document can be viewed at `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31991L0250:EN:NOT`. We extracted the text body from the HTML documents, excluding HTML tags, bibliographic notes or other additional information that could distort the results. We conducted a similar text preprocessing as for the Reuters datasets, this time selecting the 5,000 most frequent features.

6.5 Algorithmic Setup

All algorithms are trained incrementally. For the *RCV1* dataset, a single, chronological pass through the data was used (one epoch) because our previous results have shown that multiple iterations are not necessary (Loza Mencía and Fürnkranz, 2008a). For the EUR-Lex datasets we report the results for one epoch, more results can be found in (Loza Mencía and Fürnkranz, 2008b). On the other datasets, we trained for multiple epochs. We report the results for training after 100 epochs for *yeast* ans *scene* and 10 epochs for the small *Reuters-21578*. However, in terms of the relative order of the tested methods, we found that the results are quite insensitive to the exact numbers of epochs.

Except for the large Reuters RCV1 data, where we used the dedicated test set, all reported results are estimated from 10-fold cross-validation. In order to ensure that no information from the test set enters the training phase for the

[3] `http://eur-lex.europa.eu/en/legis/index.htm`

[4] `http://europa.eu/eurovoc/`

text datasets, the TF-IDF transformation and the feature selection were conducted only on the training sets of the ten cross-validation splits.

For the MMP algorithm we used the IsErr loss function and the uniform penalty function. This setting showed the best results in (Crammer and Singer, 2003) on the RCV1 data set. All the perceptrons of the different algorithms were initialized with random values, except for DMLPP where the result of training a base classifier for the first time was randomized. We used the first variant of the QCMLPP algorithm for the prediction performance results.

7 Evaluation

The following sections analyze, in short, the prediction performance and in a more extensive way the computational and memory efficiency of the presented algorithms.

7.1 Prediction Performance

Table 7.1 shows the ranking prediction performance according to the evaluation measures presented in Section 6.2 and Table 7.2 shows the label set predictions performance according to Section 6.1. The tables summarizes results from previous experiments (Fürnkranz et al., 2008; Loza Mencía and Fürnkranz, 2008b,a) as well as new experiments with QCMLPP on the several multilabel datasets. Note that no results were reported on the *EUROVOC* dataset for the non-dual variants due to the overwhelming memory consumption.

The first remarkable observation is that the pairwise approaches dominate the one-per-class approaches for each dataset and measure. The only exception is the QCMLPP1 algorithm for the ranking losses, since QCMLPP is not interested in providing a good ranking: after it is clear which labels are at top of the calibrated label, no additional computations are done in order to determine the remaining ordering. On the other hand note that the multilabel losses of the QCMLPP are exactly equal to that of CMLPP since both compute for every instance the same partitioning into relevant and irrelevant labels.

7.2 Computational Efficiency

We analyzed the computational efficiency in two settings. The first comparison concentrates on the savings in base classifier evaluations using the *QWeighted* method on the different multilabel datasets. The second setting compares the costs when the number of classes is becoming very high and using the dual variant becomes more important and essential. For both settings we ignored minor operations that have to be performed by all algorithms, such as sorting or internal operations in order to allow a comparison independent from external factors such as logging activities and the run-time environment.

7.2.1 QWeighted

Table 7.3 depicts the gained reduction of prediction complexity of the *QWeighted* approach with respect to the classifier evaluations for CMLPP. For each of the four listed methods the average number of base classifier evaluations is stated. In addition, for QCMLPP1 and 2 the ratio of classifier evaluations to the complete set of pairwise classifiers, which are typically evaluated in the CMLPP approach, are denoted within brackets, to emphasize the achieved reduction. Note that the costs of DMLPP cannot be expressed by base classifier evaluations so that we left it out for this analysis.

The first remarkable observation is the clear improvement when using the *QWeighted* approach. Except for the *yeast* and *scene* dataset, both variants of the QCMLPP use less than the tenth part of the classifier evaluations for CMLPP.

Table 7.1: Ranking performance of the different algorithms. For AvgP the higher values the better, for RankLoss low values near zero are good. Bold values represent the best value for each dataset and measure combination.

	AvgP					RankLoss				
dataset	BR	MMP	(D)MLPP	CMLPP	QCMLPP	BR	MMP	(D)MLPP	CMLPP	QCMLPP
scene	85.64	79.48	86.43	**86.70**	81.88	8.17	11.81	7.47	**7.29**	12.07
yeast	70.41	71.39	**75.15**	75.05	67.70	22.73	21.03	**17.47**	17.56	24.45
rcv1-v2	88.23	92.82	93.70	**93.83**	81.19	2.53	0.69	0.48	**0.47**	5.47
reuters21578	84.45	90.20	90.77	**91.14**	79.11	6.00	1.50	0.79	**0.78**	7.37
EUR-Lex *subject matter*	62.90	74.71	**78.15**	78.92	54.98	16.36	2.96	1.17	**1.15**	16.82
EUR-Lex *directory code*	57.10	70.00	76.42	**76.95**	41.19	19.30	2.75	**1.14**	**1.14**	23.49
EUR-Lex EUROVOC	26.90	29.28	**46.67**	—	—	40.35	3.906	**2.779**	—	—

Table 7.2: Multilabel performance of the different algorithms. For HamLoss low values are good, for F1 the higher the better. Bold values represent the best value for each dataset and measure combination.

	HamLoss		$F1$	
	BR	(Q)CMLPP	BR	(Q)CMLPP
scene	10.42	**10.00**	71.19	**72.76**
yeast	24.09	**22.67**	59.76	**62.83**
rcv1-v2	1.26	**1.03**	79.93	**83.22**
reuters21578	0.78	**0.55**	67.92	**74.63**
EUR-Lex *subject matter*	1.05	**0.69**	57.93	**63.20**
EUR-Lex *directory code*	0.42	**0.23**	46.91	**50.81**

Table 7.3: Computational costs at prediction in average number of predictions per instance. The italic values next to the two multilabel adaptations of *QWeighted* show the ratio of predictions to CMLPP and the second rightmost column describes the average number of relevant labels.

dataset	n	MMP, BR	CMLPP	QCMLPP1	QCMLPP2	$n\log(n)$	$n+dn\log(n)$	d	density $\frac{d}{n}$
scene	6	6	21	11.51 *(54.8%)*	11.46 *(54.58%)*	10.75	17.50	1.07	17.9 %
yeast	14	14	105	67.57 *(64.4%)*	64.99 *(61.9%)*	36.94	170.65	4.24	30.3 %
rcv1-v2	103	103	5356	485.23 *(9.06%)*	456.23 *(8.52%)*	477.38	1649.70	3.24	2.9 %
reuters21578	120	120	7260	378.45 *(5.21%)*	325.94 *(4.49%)*	574.50	843.87	1.26	1.0 %
EUR-Lex *sj*	201	201	20301	1144.2 *(5.64%)*	825.07 *(4.06%)*	1065.96	2556.78	2.21	1.1 %
EUR-Lex *dc*	412	412	85078	2610.76 *(3.07%)*	1288.22 *(1.51%)*	2480.66	3612.05	1.29	0.3 %

Another appreciable point, especially regarding the mentioned deviation, is the clearly visible correlation between the gained reduction and the label density of the problem, i.e. the ratio of the average number of labels per instance to the total number of labels. The dataset with the highest density, *yeast*, achieved the lowest reduction. Similarly both QCMLPP variants evaluated the lowest ratio of classifiers for the dataset with the lowest density, the EUR-Lex *directory code* dataset. This observation confirms the previously stated expectation that the reduction is highly influenced by the density. This effect is not surprising, since roughly speaking QCMLPP employs iteratively *QWeighted* until the calibrated label is found, and the number of iterations is obviously related to the density. Furthermore the results show that QCMLPP2 slightly outperforms QCMLPP1, except for the last dataset, where QCMLPP2 performs half the operations than the naive variant. Here again especially QCMLPP2 seems to benefit from the low label density of this dataset.

For estimating the average runtime in practice, two columns were included, which state the $n\log(n)$ and $n+dn\log(n)$ values for the corresponding datasets. We can clearly confirm that the number of classifier evaluations is for all considered datasets smaller than the previously estimated upper bound of $n+dn\log(n)$. Note that the value for *yeast* 170.65 is actually greater than the number of existing classifiers (105). This is due to the fact, that the values lie yet in a range where lower order terms have still an impact in the equation.

Figure 7.1 visualizes the above results and allows again a comparison to different complexity values such as n, $n\log(n)$ and n^2. The upper figure is a recapitulation of the results from Park and Fürnkranz (2007a) extended with multiclass classification performance results of the multilabel datasets considered in this paper: instead of evaluating until finding the calibrating label, *QWeighted* was only applied once as if it was a multiclass problem. These results for the simulated multiclass classification performance support additionally the statement that *QWeighted* achieves an $n\log(n)$ runtime in practice. For better readability, a logarithmic scale for both axis is used. The lower figure is more interesting in this context, where multilabel classification prediction complexity of QCMLPP is presented. Note that the y-axis now describes the number of comparisons respectively classifier evaluations divided by the number of labels, which is graphically motivated and allows a finer distinction of the different curves. Note that for the black curve ($n+dn\log(n)$), the actual average number of labels from data was used for computing the values and are identical to the ones from table 7.3. Though the figure may indicate that a runtime of $n\log(n)$ is still achievable for multilabel classification, especially for QCMLPP2, we interpret the results more cautiously and conclude that the more sound $n+dn\log(n)$ runtime can be expected in practice.

7.2.2 Dual Representation

As already analyzed in Section 5, the dual variant DMLPP not only allows to reduce the memory consumption on problem with high class cardinality, but in the same manner allows to reduce computational costs for this type of large-scale problems. We analyzed therefore the computational costs on the three different EUR-Lex datasets since they allow us to observe the direct impact of a varying label set size. Table 7.2.2 is based on previous experiment (Loza Mencía and Fürnkranz, 2008c,b) and was enriched by the results of the calibrated and *QWeighted* variants. It shows the amount of real value addition and multiplication computations (measured on the first cross validation split, trained for one epoch).

We can observe a clear advantage of the non-pairwise approaches on the *subject matter* data especially for the prediction phase, however the training costs are in the same order of magnitude.[1] Between MLPP and DMLPP we can see an antisymmetric behavior: while MLPP requires only almost half of the amount of the DMLPP operations for training, DMLPP reduces the amount of prediction operations by a factor of more than 4. Nevertheless this value is beaten by far by the *QWeighted* variants, repeating the previous observations on the number of base classifier evaluations. For the *directory code* the rate for MMP and BR more than doubles in correspondence with the increase in number of classes, additionally the MLPP testing time substantially increases due to the quadratic dependency, while DMLPP profits from the decrease in the average number of classes per instance. It even causes less computations in the training phase than MMP/BR. Again QCMLPP is faster in testing than DMLPP, but the distance is quite smaller. For training DMLPP is clearly faster. Note that the calibrating variants need the sum of BR's and MLPP's computations for training. The reason for the low number of computations for DMLPP is not only the reduced maximum amount of weights per instance (cf. Section 5), but particularly the decreased probability that a training example is relevant for a new training example (and consequently that dot products and scores have to be computed) since it is less probable that both class assignments match, i.e. that both examples have the same pair of positive and negative classes.

This becomes particularly clear if we observe the number of non-zero weights and actually used weights during training for each new example. The classifier for subject matter has on average 21 weights set per instance out of 443 ($= d(n-d)$) in the worst case (a ratio of 4.47%), and on average 5.1% of them are required when a new training example arrives. In other terms, this means that a training instance is only relevant to on average 4.47% of the base classifiers, and of these base classifiers only 5.1% are affected when updating with a new training instance. For the directory code with a smaller fraction d/n 35.5 weights are stored (3.96%), of which only 1.11% are used when updating. This also explains the relatively small number of operations for training on EUROVOC, since from the 1,802 weights per instance (8.41%), only 0.55% are relevant to a new training instance.

However DMLPP cannot benefit in the same manner from this point during the prediction phase on the *EUROVOC* as on the *directory code* subset. Nevertheless it is still more efficient during training than the one-per-class variants. No results could be retrieved for the non-dual pairwise variants on the EUR-Lex dataset due to the high memory requirements (cf. Section 7.3), but we can try to estimate the expected number of computations.

Based on the estimation that MLPP requires d times computations than BR, we could expect around 175.000 M op. for training on *EUROVOC*. (Q)CMLPP would require additionally the computations of BR. For testing, we estimate the number of base classifier evaluations with $n + dn \log n$ (cf. Section 5) which seems to provide reliable results (cf. previous setting). Under this assumption and using an average of 0.955 M op. per base classifier evaluation (for the whole test set), we obtain approx. 88.000 M op. for QCMLPP, although computing the calibrated . This would mean again an advantage over DMLPP, but a relatively small one compared to the previous datasets. However for training, DMLPP's costs are almost ten times smaller than the estimated MLPP costs. Note also that such experiment setting are (currently) not possible due to the exponential memory requirements of the non-dual pairwise approaches (see next Section).

7.3 Memory Requirements

In order to compare the memory requirements, we measured the consumed memory on the EUR-Lex datasets. These datasets allows us to compare directly the memory consumption in dependency of the number of classes since they share the same instance set and only differ in the concrete labelling. In addition this dataset collection was the initial reason for the development of a memory saving variant of the pairwise approach, as it was not possible to process in particular the *EUROVOC* view with its almost 4000 classes.

The memory consumption provided by the Java Virtual Machine after training the several classifiers is depicted in Table 7.3. Note that these sizes include the overhead caused by the virtual machine and the machine learning framework.[2]

[1] Note that MMP and BR compute the same amount of dot products, the computational costs only differ in the number of vector additions, i.e. perceptron updates. A deeper analysis of the contrary behavior of both algorithms when the number of classes increases can be found in (Loza Mencía and Fürnkranz, 2007).

[2] We used the WEKA framework (`http://www.cs.waikato.ac.nz/~ml/weka/`), but we adapted it so that it only maintains a copy of an instance in memory when necessary for the incremental updating.

Table 7.4: Computational costs in millions of real value operations (M op.) for the EUR-Lex datasets.

subject matter	training	testing
BR	1,675 M op.	192 M op.
MMP	1,789 M op.	192 M op.
MLPP	3,870 M op.	19,210 M op.
CMLPP	5,545 M op.	19,402 M op.
QCMLPP1	5,545 M op.	977 M op.
QCMLPP2	5,545 M op.	729 M op.
DMLPP	6,089 M op.	4,628 M op.

directory code	training	testing
BR	3,410 M op.	394 M op.
MMP	3,579 M op.	394 M op.
MLPP	4,712 M op.	80,938 M op.
CMLPP	8,123 M op.	81,331 M op.
QCMLPP1	8,123 M op.	2,331 M op.
QCMLPP2	8,123 M op.	1,123 M op.
DMLPP	2,986 M op.	5,438 M op.

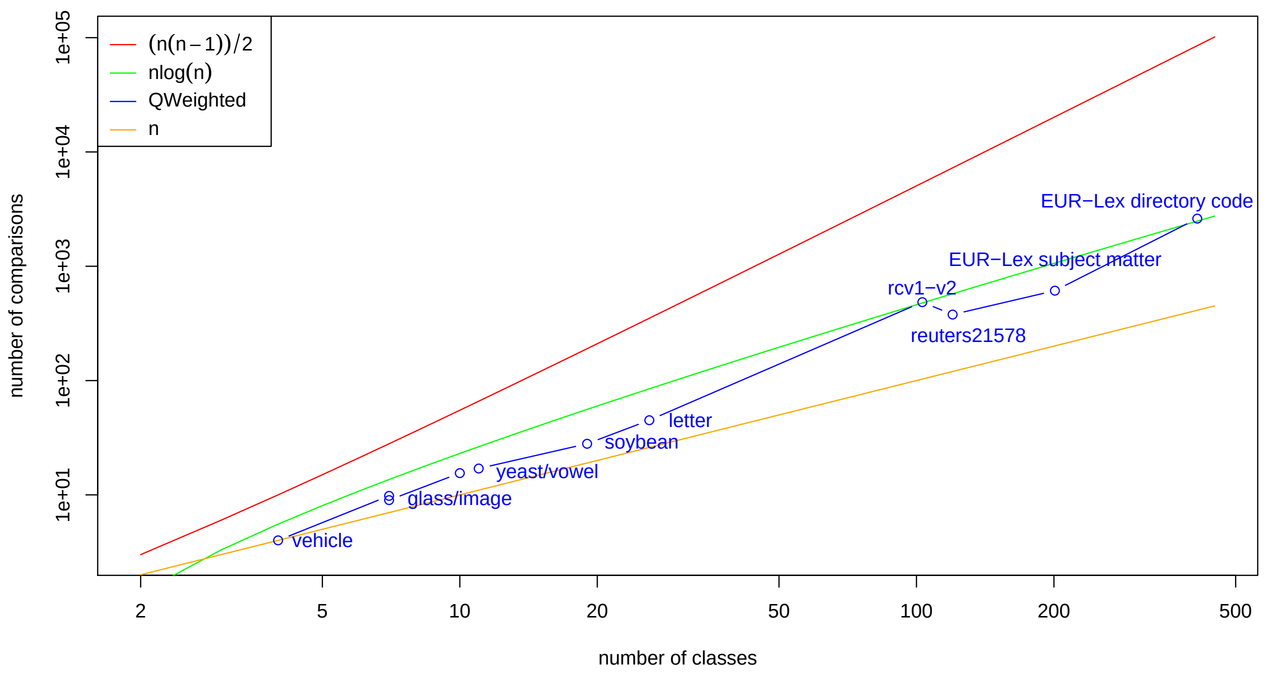
EUROVOC	training	testing
BR	32,975 M op.	3,817 M op.
MMP	40,510 M op.	3,817 M op.
DMLPP	17,719 M op.	127,912 M op.

Table 7.5: Memory requirements of the different classifiers for the EUR-Lex datasets.

dataset	BR/MMP	DMLPP	MLPP
subject matter	151 MB	203 MB	539 MB
directory code	165 MB	217 MB	1825 MB
EUROVOC	1143 MB	2246 MB	–

MLPP already consumes more memory than the dual variant for the first dataset with 200 classes. For the 400 classes of the *directory code* view the algorithm requires almost 2 GB, while DMLPP is able to compress the same information into slightly more than 200 MB. As expected and already mentioned in Section 7.1, MLPP is not applicable to *EUROVOC*. A simple estimation based on the number of base classifier (almost eight mio.), number of features and bytes per float results in 152 GB of memory.

Another remarkable fact is that the memory requirement of DMLPP is comparable to the one of the one-per-class algorithms: for the smaller datasets we obtain an overhead of only 50 MB and for the bigger *EUROVOC* view it requires only double of the memory, although representing a quadratic number of base classifiers.



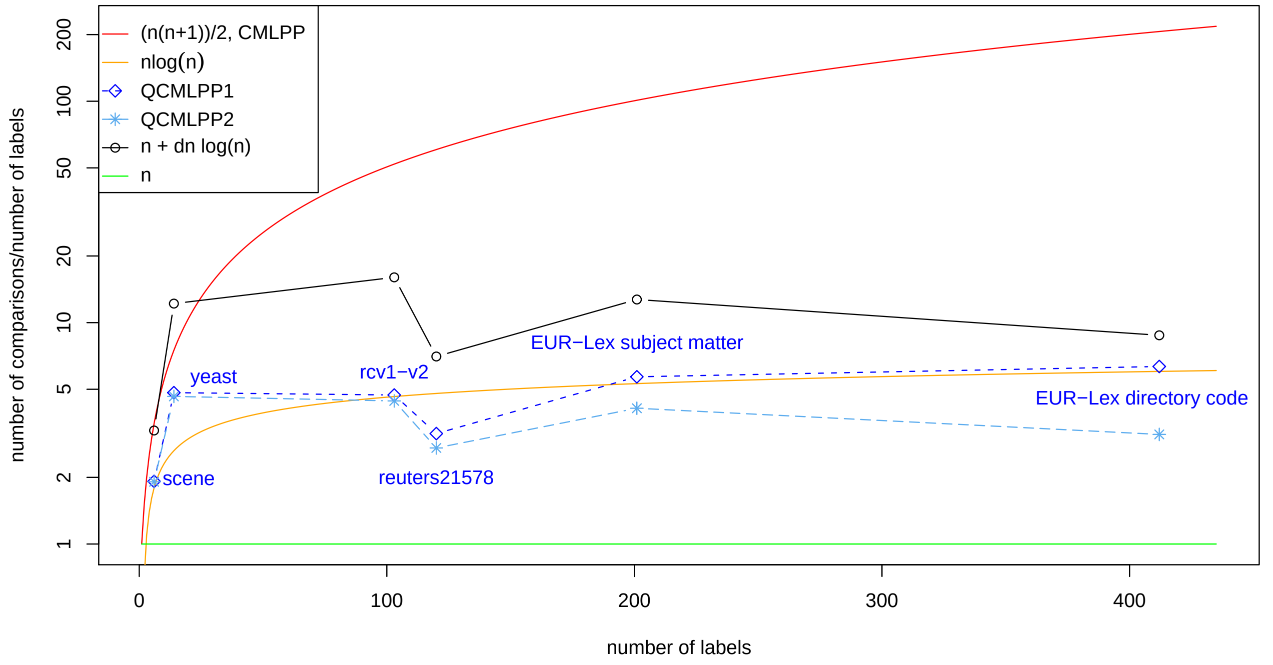


Figure 7.1: Prediction complexity of *QWeighted* and QCMLPP: number of comparisons needed in dependancy of the number of classes n for different multiclass and multilabel problems. *Upper figure*: Problems *vehicle* to *letter* in the first figure are multiclass problems already analyzed by Park and Fürnkranz (2007a), while multiclass versions of the multilabel datasets *rcv1-v2, reuters21578, EUR-Lex subject matter* and *EUR-Lex directory code* were evaluated within this study. *Lower figure*: QCMLPP1/2 is compared to $n(n+1)/2$ as in CMLPP, n as in BR and $n\log(n)$ on 6 multilabel datasets.

8 Conclusions

Multilabel classification is becoming a more and more important task in machine learning due to the increasing amount of application scenarios where it is necessary to not only predict one top class as in multiclass classification, but a set of relevant classes. The common approach of training one classifier for each class that determines a binary relevance is clearly outperformed by the approach of learning pairwise preferences between pairs of classes. The main disadvantage of this approach was, until now, the quadratic number of base classifiers needed and hence the increased computational costs for prediction and the increased memory requirements. We have presented in this paper one time efficient and one memory space efficient algorithm based on the pairwise approach.

The first variant combines a technique that transforms a class ranking into a bipartite prediction by introducing an artificial thresholding class, called calibration (Fürnkranz et al., 2008), with the *QWeighted* voting that stops the computation of the ranking when the bipartite separation is already determined (Park and Fürnkranz, 2007a). For the combined *QWeighted* multilabel method the computational costs savings compared to the normal voting are especially important with increasing number of classes. Though not analytically proven, our empirical results show that the complexity is upper bounded by $n + dn\log(n)$, in comparison to the evaluation of n in the case of one-per-class approaches and $O(n^2)$ for the unmodified pairwise approach. For the *QWeighted* multilabel approach, we see improvements in a more appropriate integration of the *QWeighted* concept, namely to identify and exploit unnecessary classifier evaluations, to the multilabel setting. In this context, QCMLPP2 was already a step forward. Additionally, we are currently investigating approaches in order to reduce the number of computations needed for converge to the final ranking such as the swiss system used in tournaments (Park and Fürnkranz, 2007b).

The second method emerged from the need to process a dataset with almost 4000 classes (Loza Mencía and Fürnkranz, 2008b). DMLPP takes advantage of the dual representation of the perceptrons in order to shift the main dependency from the number of classes to the number of training instances. This enables to use the pairwise approach with a high number of classes and, additionally reduces the computational costs for some datasets. The next step will be to combine *QWeighted* and the dual approach, both in the sense of applying the intelligent mechanism to voting in DMLPP, as well as in the sense of using a hybrid variant that switches between dual or *QWeighted* training or testing depending on the characteristics of the dataset (cf. Section 4).

Acknowledgements

This work was supported by the EC 6th framework project *ALIS (Automated Legal Information System)* and by the *German Science Foundation (DFG)*.

Bibliography

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995. 4

M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9): 1757–1771, 2004. 15

K. Brinker, J. Fürnkranz, and E. Hüllermeier. A Unified Model for Multilabel Classification and Ranking. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06)*, 2006. 7

K. Crammer and Y. Singer. A Family of Additive Online Algorithms for Category Ranking. *Journal of Machine Learning Research*, 3(6):1025–1058, 2003. 3, 4, 5, 14, 17

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006. 4

A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, volume 14, pages 681–687, 2001. 5

Y. Freund and R. E. Schapire. Large Margin Classification using the Perceptron Algorithm. *Machine Learning*, 37(3): 277–296, 1999. 4

J. Fürnkranz. Round Robin Classification. *Journal of Machine Learning Research*, 2:721–747, 2002. 3, 6

J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 2008. In Press. 3, 7, 18, 23

C.-W. Hsu and C.-J. Lin. A Comparison of Methods for Multi-class Support Vector Machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002. 3, 6

I. Katakis, G. Tsoumakas, and I. Vlahavas. Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD-08 Workshop on Discovery Challenge*, Antwerp, Belgium, 2008. 3

R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007. 4

D. D. Lewis. Reuters-21578 text categorization test collection. README file (V 1.2), available from `http://www.research.att.com/~lewis/reuters21578/README.txt`, September 1997. 15

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004. 4, 5, 14

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. S. Kandola. The Perceptron Algorithm with Uneven Margins. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 379–386, 2002. 4

E. Loza Mencía and J. Fürnkranz. An evaluation of efficient multilabel classification algorithms for large-scale problems in the legal domain. In *LWA 2007: Lernen - Wissen - Adaption, Workshop Proceedings*, pages 126–132, 2007. 3, 15, 20

E. Loza Mencía and J. Fürnkranz. Pairwise learning of multilabel classifications with perceptrons. In *Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IJCNN 08)*, pages 2900–2907, Hong Kong, 2008a. 3, 5, 6, 12, 15, 16, 18

E. Loza Mencía and J. Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In W. Daelemans, B. Goethals, and K. Morik, editors, *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Disocvery in Databases (ECML-PKDD-2008), Part II*, pages 50–65, Antwerp, Belgium, 2008b. Springer-Verlag. 3, 15, 16, 18, 20, 23

E. Loza Mencía and J. Fürnkranz. Efficient multilabel classification algorithms for large-scale problems in the legal domain. In *Proceedings of the Language Resources and Evaluation Conference (LREC) Workshop on Semantic Processing of Legal Texts*, pages 23–32, Marrakech, Morocco, 2008c. 3, 15, 20

S.-H. Park and J. Fürnkranz. Efficient pairwise classification. In J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, editors, *Proceedings of 18th European Conference on Machine Learning (ECML-07)*, pages 658–665, Warsaw, Poland, 2007a. Springer-Verlag. 3, 8, 11, 19, 22, 23

S.-H. Park and J. Fürnkranz. Efficient pairwise classification and ranking. Technical Report TUD-KE-2007-03, TU Darmstadt, Knowledge Engineering Group, 2007b. 23

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. 4

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988. 15

F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002. 5

S. Shalev-Shwartz and Y. 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. 4

P. Tsampouka and J. 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. 4

G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of the ECML/PKDD-08 Workshop on Mining Multidimensional Data (MMD-08)*, Antwerp, Belgium, 2008. 3