Efficient Decomposition-Based Multiclass and Multilabel Classification

TECHNISCHE UNIVERSITÄT DARMSTADT

Vom Fachbereich Informatik der
Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

von

Dipl.-Inform. Sang-Hyeun Park
(geboren in Frankfurt am Main)

Erstreferent:	Prof. Dr. Johannes Fürnkranz
Korreferent:	Prof. Dr. Eyke Hüllermeier (Philipps-Universität Marburg)
Tag der Einreichung:	29.03.12
Tag der Disputation:	24.05.12

D17
Darmstadt 2012

Abstract

Decomposition-based methods are widely used for multiclass and multilabel classification. These approaches transform or reduce the original task to a set of smaller possibly simpler problems and allow thereby often to utilize many established learning algorithms, which are not amenable to the original task. Even for directly applicable learning algorithms, the combination with a decomposition-scheme may outperform the direct approach, e.g., if the resulting subproblems are simpler (in the sense of learnability). This thesis addresses mainly the efficiency of decomposition-based methods and provides several contributions improving the scalability with respect to the number of classes or labels, number of classifiers and number of instances.

Initially, we present two approaches improving the efficiency of the training phase of multiclass classification. The first of them shows that by minimizing redundant learning processes, which can occur in decomposition-based approaches for multiclass problems, the number of operations in the training phase can be significantly reduced. The second approach is tailored to Naïve Bayes as base learner. By a tight coupling of Naïve Bayes and arbitrary decompositions, it allows an even higher reduction of the training complexity with respect to the number of classifiers. Moreover, an approach improving the efficiency of the testing phase is also presented. It is capable of reducing testing effort with respect to the number of classes independently of the base learner.

Furthermore, efficient decomposition-based methods for multilabel classification are also addressed in this thesis. Besides proposing an efficient prediction method, an approach rebalancing predictive performance, time and memory complexity is presented. Aside from the efficiency-focused methods, this thesis contains also a study about a special case of the multilabel classification setting, which is elaborated, formalized and tackled by a prototypical decomposition-based approach.

Kurzfassung

Multiklassen- und Multilabel-Klassifikationsprobleme werden häufig durch zerlegungsbasierte Ansätze gelöst. Zerlegungsbasierte Ansätze haben gemeinsam, dass sie das ursprüngliche Problem auf eine Menge von kleineren potentiell einfacheren Problemen abbilden. Oft ermöglichen solche Ansätze die Wiederverwendung von vielen bewährten Lernalgorithmen, die nicht direkt auf das ursprüngliche Problem anwendbar sind. Darüber hinaus können auch für direkt anwendbare Lernalgorithmen die zerlegten Teilprobleme einfacher (im Sinne der Lernbarkeit) sein, so dass ein zerlegungsbasierter Ansatz insgesamt eine höhere Vorhersagequalität besitzen kann als die direkte Lösung des Problems. Diese Dissertation beschäftigt sich hauptsächlich mit der Effizienz der zerlegungsbasierten Methoden und erarbeitet mehrere Ansätze mit einer besseren Skalierbarkeit bezüglich Anzahl der Klassen bzw. Labels, Klassifizierer und Instanzen der Daten.

Es werden zunächst zwei Ansätze vorgestellt, welche die Trainingsphase für Multiklassenprobleme beschleunigen. In dem ersten Ansatz wird gezeigt, dass durch Minimierung von redundanten Lernvorgängen, die oft in zerlegungsbasierten Multiklassen-Klassifikationsansätzen vorkommen können, Einsparungen in der Trainingsphase möglich sind. Der zweite Ansatz ist speziell auf Naïve Bayes als Basislerner ausgerichtet und ermöglicht durch die Ausnutzung spezieller Eigenschaften in diesem Fall eine noch größere Reduktion der Lernkomplexität bezüglich der Klassifiziereranzahl. Es wird zusätzlich ein Ansatz präsentiert, welches die Klassifikationsphase für Multiklassenprobleme beschleunigt. Dieses Verfahren ist unabhängig vom verwendeten Basislerner und reduziert die Klassifikationskomplexität bezüglich der Klassenanzahl.

Darüber hinaus werden in dieser Dissertation auch Multilabelprobleme behandelt und dafür neben einer effizienten Klassifikationsmethode auch ein Ansatz vorgestellt, welches die Vorhersagequalität, den Zeitaufwand und die Speicherkomplexität neu abwägt. Neben den effizienzfokussierten Ansätzen beinhaltet diese Dissertation auch eine Studie, die einen Spezialfall von Multilabel-Klassifikationsproblemen vorstellt, formalisiert und mittels einem prototypischen zerlegungsbasierten Ansatz zu lösen versucht.

Contents

List of Figures

List of Algorithms

List of Tables

Notation

$\vec{x}, x \in X$	instance (see text), set of instances
t	number of instances
D	dataset
$a \in A$	feature, set of features
g	number of features
$c \in K$	class, set of classes
$\lambda \in L$	label, set of labels
k	number of classes/labels
P	relevant labels or positive classes
N	irrelevant labels or negative classes
d	average number of relevant labels
y	output variable
$f(x) \in C$	classifier, set of classifiers
n	number of classifiers
$c\vec{w} = (b_1, \ldots, b_n)$	code word, code bits
$\vec{w}$	weight vector
r_{zp}	random zero probability (random codes)
$\tau, \tau(\lambda)$	ranking (of labels/classes), rank position of label λ
β	number of partitions (HOMER)
$z \in Z$	label constraint, set of label constraints
$pr \in PR$	preference, set of preferences
q	number of label constraints
$\mathbb{N}, \mathbb{R}$	natural numbers, real numbers
$\mathbb{S}$	permutation space
$\mathbb{Z}$	label constraints space

1 Introduction

The classification task is one of the most elementary problems in machine learning. The automatic learning of a predictor function from a given set of *training* instances, which maps unseen *test* instances to its corresponding true class was studied extensively since the beginning of machine learning. The distinct formulation and easy comprehensibility of the underlying setting have certainly contributed to this high attention. It is a testbed from which many successful learning algorithms evolved, e.g., RIPPER/SLIPPER (Cohen, 1995; Cohen and Singer, 1999), C4.5 (Quinlan, 1993) and SVM (Vapnik, 1998).

It is apparent that automatic classification also has a high direct practical value. The possibility to learn an accurate predictor in domains such as speech recognition, optical character recognition or spam classification improves the convenience for humans. Though these tasks are in principle trivial for humans, in light of the increasing mass of digital information, manual classification becomes infeasible for large scale datasets and efficient automatic classification methods grow in importance. Another aspect is the automatic analysis of data, which may complement the human analysis and provide an *alternative* view – the rules of a learned classifier may expose previously unknown insights about the data. The aspired discovery of these so-called "nuggets", i.e. new patterns or rules inherent in data, provides a strong incentive for research in machine learning or data mining in general respectively.

Multiclass and multilabel classification are special cases of the above-mentioned classification task and impose different restrictions on the target output variable. For multiclass classification the target variable has to be one of a given set of identifiers, called *classes*. An example is the task of optical character recognition, where the target variable is one of the alphabetic or numeric characters. For the multilabel case, the target variable is a subset of a given set of classes. This means more than one class (in this context typically called labels) can be associated with the instance. A *German* movie m categorized as an *action*-film with *comedy*-elements may be seen as the multilabel instance $(m, \{\text{german}, \text{comedy}, \text{action}\})$.

These tasks are commonly tackled by so-called *decomposition-based* approaches, which transform the problem into a set of smaller possibly simpler problems. For instance, a k-class classification problem can be transformed to a set of *binary* (2-class) problems. This makes many classification algorithms which are only applicable for 2-class problems amenable for this task. Decompositions allow to reuse learn algorithms readily in more complex tasks by applying an appropriate transformation scheme. Besides this practical convenience, the decompositions to simpler problems can also lead to a better predictive and efficiency performance compared to the direct approach (Ghani, 2000; Hsu and Lin, 2002; Fürnkranz, 2002).

1.1 Motivation

In contrast to classical computer science algorithms such as sorting algorithms, where the only acceptable correctness is *fully sorted* and algorithm research was focused in minimizing time and memory complexity, in machine learning a perfect (predictive) result is seldom feasible. It is apparent to prioritize the predictive performance over time and memory complexity. Thus, predictive performance is the predominant measure considered in multiclass and multilabel methods.

However, in light of the growing amount of data which needs to be processed, the time and memory complexity increased in its importance. At the latest, in extreme cases, where the memory or time complexity presents an infeasible bottleneck for the whole task, a reprioritization of these factors is necessary.

Furthermore, as previously described, classification is one of the basic components, which is often employed in a modular way for more complex machine learning tasks. For example, throughout this thesis, multiclass and multilabel classification will be tackled by an *ensemble* of binary classifiers. In general for multiclass classification, we will be concerned with so-called *error-correcting output codes (ECOC)* which provide a unified framework for common decomposition-based ensemble types. Efficiency bottlenecks of base classifiers can drastically accumulate throughout the ensemble or the whole framework. This means also that efficiency improvements on the base classifier level may have a drastic overall reduction to the whole task.

Besides the illustrated extreme case, improving the efficiency can also implicitly improve the predictive performance. The sheer capability of processing more data in a fixed time means obviously to increase the *sample*-size and, therefore, to strengthen the *statistical power* of the underlying statistical learning algorithms or at least of the various employed statistical measures.

1.2 Contributions

In this thesis, we present mainly efficiency and partly efficacy improvements for decomposition-based multiclass and multilabel classification methods.

Multiclass Classification

Efficient ECOC Training by Exploiting Code-Redundancies
ECOC-based classifier ensembles can have overlapping training processes. We develop a training schedule, which tries to minimize these redundant learning processes. This schedule based learning is directly applicable for genuine incremental learners, but we also develop modifications for the genuine batch learner SVM to be applicable for this approach.

Efficient ECOC Training with Naïve Bayes
A simple tight combination of Naïve Bayes and ECOC is presented, which reduces the training effort significantly compared to the straight-forward combination. This is mainly done by an alternative equivalent computation scheme exploiting some special relations in this setting.

Efficient ECOC Prediction
Here, we present an efficient prediction/decoding scheme for ECOC classifiers. It is based on the fact, that it is not always necessary to consider all classifier evaluations for the decoding phase. Although we do not give a theoretical average-case analysis, extensive empirical evaluations indicate the significance of this approach in practice.

Multilabel Classification

Efficient Pairwise Multilabel Prediction
Two efficient prediction algorithms for the calibrated label ranking approach for multilabel classification are presented. The prediction phase of the underlying ensemble of *pairwise* classifiers (special type of binary classifiers) is similarly improved as for the case of ECOC classifiers in multiclass prediction.

Combination of Multilabel Classification Decompositions
Here, the efficiency and also the predictive performance of the above approach is further improved in combination with the HOMER approach, which transforms the original multilabel problem into a set of smaller multilabel problems.

Multilabel Classification with Label Constraints
We consider a variant of the multilabel setting and elaborate on the existence of constraints, or at least, dependencies among labels in real data. Besides using association-rule learning to find such constraints, we experiment with two methods on incorporating them into the learning process.

1.3 Structure of this thesis

This thesis is divided into two parts: Part I is dedicated to multiclass and Part II to multilabel classification. Each part begins with its own preliminaries chapter (Chapter 3 and Chapter 7), which briefly recapitulates the setting, measures and common decomposition-based methods. Except for the introduction (Chapter 1), the introductory chapter on binary classification (Chapter 2) and the summary (Chapter 11), each of the remaining chapters is dedicated to one contribution, which in turn is based on a publication. The corresponding references are shown in the following list. Please note, that the basics chapters are also mainly based on these publications.

Part I Multiclass Classification

Chapter 4, Efficient ECOC Training by Exploiting Code-Redundancies
(Park, Weizsäcker, and Fürnkranz, 2010)

Chapter 5, Efficient ECOC Training with Naïve Bayes
(Park and Fürnkranz, 2011)

Chapter 6, Efficient ECOC Prediction
(Park and Fürnkranz, 2012)

Part II Multilabel Classification

Chapter 8, Efficient Pairwise Multilabel Prediction
(Loza Mencía, Park, and Fürnkranz, 2010)

Chapter 9, Combination of Multilabel Classification Decompositions
(Tsoumakas, Loza Mencía, Katakis, Park, and Fürnkranz, 2009)

Chapter 10, Multilabel Classification with Label Constraints
(Park and Fürnkranz, 2008)

2 Binary Classification

Contents

Binary classification refers to the task of automatically mapping input instances to their most probable class or category, where the number of classes is restricted to two classes. A multitude of approaches based on different learning *concepts* originated for these *binary* problems.

This chapter contains a brief recapitulation of basic knowledge about binary classification and standard learning algorithms for this task.

2.1 Binary Setting

In the binary classification setting, we consider a set of instances/examples $X = \{\vec{x}_i \mid i = 1 \dots t\}$, where each instance is associated to one of two classes. One often speaks of the *positive* and the *negative* class in this context, similarly also for instances, i.e. positive instances are instances which true class is the positive class and vice versa. Thus, we consider instance-class pairs $(\vec{x}, y)$, where $y \in K = \{c_{pos}, c_{neg}\}$ or $\{+, -\}$. Furthermore, each instance is represented as a vector $\vec{x} = (a_1, \dots, a_g)$ in a feature space $\mathbb{R}^g$.

Typically, a subset of instances $\bar{X} \subseteq X$ along with their corresponding true class associations is given, i.e. we have given a set of pairs $(\vec{x}, y)_i$, which are used to learn a classifier $f(.) : X \to K$. This classifier is supposed to predict for previously unseen instances $\vec{x}_i$ the correct class, such that some performance criterion (e.g., classification accuracy) is maximized.

In the following, we will often neglect the vector notation of instances for convenience reasons, i.e. we will use x synonymously with $\vec{x}$, except for cases, where we particularly focus on the features a_i of an instance. Furthermore, if it is clear from the context, that the same instance x_i is addressed, we will often also omit the index.

2.2 Classification Evaluation and Variants

2.2.1 Standard Measures and Methods

We will provide a short description of standard performance measures and methods used in binary classification.

Accuracy

The following measure describes the *empirical accuracy* of classifier f for a given set of instances $\bar{X}$, which is the predominant predictive evaluation measure in binary classification.

$$\textsc{Acc}(f, \bar{X}) := \frac{1}{t} \sum_{i=1}^{t} I(f(x_i) = y_i)$$

where $I(.)$ denotes the indicator function, which returns 1 if the statement is true and 0 otherwise. It represents the fraction of correctly predicted instances. Usually, one is interested in obtaining a classifier, which optimizes this measure for unseen related instances. This task is typically viewed in the following way: the given (training-) set of instances $\bar{X}$ represents *only* a subset of an unknown complete, possibly infinite set of instances X and we are rather interested in a classifier which optimizes $\textsc{Acc}(f, X)$. Thus, instances $x \in X$, which are not necessarily members of $\bar{X}$ can appear during the prediction phase. In probabilistic terms, the general objective is to maximize

$$\mathrm{E}_{(x,y)\sim X} I(f(x) = y)$$

The expected value of the complementary event, i.e. $I(f(x) \neq y)$, is also called the *true risk* of a classifier f.

True Negative Rate and True Positive Rate

The true negative rate (TNR) and true positive rate (TPR) describe the accuracy only for one of the two classes, i.e. it describes the fraction of correctly predicted negative or positive instances respectively. The computation is similar to accuracy except that one iterates only over the negative or positive instances respectively.

False Negative Rate and False Positive Rate

The false negative rate (FNR) describes the fraction of positive instances which were incorrectly classified as negative ones, or shorter, the fraction of incorrectly predicted

positive instances. This equals $1 - \text{TPR}$ or can be computed similarly as the TPR by considering only positive instances and determining the fraction of mispredictions, i.e. $I(f(x_i) \neq y_i)$.

Similar to the false negative rate, the false positive rate (TPR) describes the ratio of incorrectly predicted negative instances, i.e. which were predicted as positive. Again, it can be computed by $1 - \text{TNR}$ or if TNR is not given in a straight-forward manner.

Cross-Validation

Since, in practice, the complete set of instances is not known or not finite, one can only approximate the true risk or other similar measures of a classifier. One well-known approach for evaluating the *generalization* property of a classifier is *cross-validation*. This method repeatedly trains and tests on disjoint subsets of the given data such that each instance serves at least once as training instance as well as testing instance. First, the dataset is equally divided into n folds. Then, the classifier is typically evaluated by using $n - 1$ folds for training and the remaining fold for testing. This process is repeated n-times, such that each fold is used exactly once for testing. Finally, the multiple evaluation results are combined by averaging. This process is called n-fold cross validation and if the folds are additionally sampled according to the class distribution of the set of training instances, one speaks of *stratified* cross-validation. Furthermore, the special case of $n = t$, i.e. considering each single instance as a fold, is called *leave-one-out* validation.

2.2.2 Cost-Sensitive Classification

Cost-sensitive classification is a variant of the basic setting. Until now, accuracy treated each class as equally important. This is sometimes not the case in real-world problems. For instance, the prediction of *not cancer* of a patient, who has cancer is obviously a more severe error than the reverse case. So, in this setting different importance among classes and the severity of their mispredictions are tackled by associating different cost-values $\gamma_{a,b}$ for each pair of classes (true class a and predicted class b). Thus, the objective in this case is to minimize a *cost-weighted* error measure:

$$\textsc{Cost}(f, \bar{X}) = \frac{1}{t} \sum_{i=1}^{t} I(\hat{y}_i \neq y_i) \cdot \gamma_{y_i, \hat{y}_i} \quad \text{where } \hat{y}_i = f(x_i)$$

For the multiclass classification setting, which will be introduced in the next chapter and involves more than two classes, the cost matrix $\gamma_{a,b} \in \mathbb{R}^{K \times K}$ allows not only to associate different cost values for the misprediction *between classes*, e.g., that a misprediction of the diagnosis *not cancer* is more costly than the misprediction of *cancer*. It is also possible to differentiate between different mispredictions for each class, for instance in weather forecasting (posed as a 3-class problem) that for the true event *hot*, the misprediction of *cold* is associated with a higher cost than *mild*.

Note that the cost-sensitive classification setting is actually a generalization of the standard classification setting. By setting all elements of the cost matrix to 1, the objective is reduced to accuracy. Actually, it is sufficient when the non-diagonal elements are set to 1. Here, the diagonal elements are essentially ignored, since the indicator function returns in these cases (correct predictions) a value of zero.

2.2.3 Area under the ROC Curve (AUC)

Besides accuracy and cost-weighted errors, in some related tasks, one is interested in the ranking capability of classifiers. Here, classifiers are assumed to return a *score* $s \in \mathbb{R}$ which quantifies the degree of class-membership of an instance. Classification can be achieved by using an appropriate thresholding value, which divides the space of score values into positive and negative class spaces (in the two-class case). To assess such ranking capabilities of a classifier/ranker f the *area under the ROC curve* (AUC) is commonly used. The receiver operating characteristic (ROC) curve has its origin from signal theory and was adapted to the machine learning field. Foremost, it is a graphical 2-d plot, which shows the false positive rate (x-axis) and the true positive rate (y-axis) of a binary classifier for varying thresholding values. If the classifier f is a good ranker, i.e. if positive examples are mostly ranked before negative examples with respect to their score, the ROC curve should ideally be convex, lie clearly above the diagonal (which represents a random ranker) and close as possible to the axis parallels of the unit square. The area under the ROC curve, i.e. the integral of the ROC curve, summarizes these criteria roughly in a single value.

2.3 Learning Algorithms

Before we provide superficial descriptions of some common learning algorithms, we will recapitulate in the following text a brief general view of learning algorithms based on (Mitchell, 1997; Witten et al., 2011).

It is not an easy task to give a general unified view over the multitude of various learning algorithms for classification. Some textbooks regard the learning process as a search process to provide a simple unified view for introductory purposes. A large portion of learning algorithms are covered by this perspective (but not all), particularly if one considers that optimization problems can be viewed as a search problem similarly. So, we make use of the same perspective here. Assuming an enumerable space of all learnable classifier functions (e.g., linear or axis-parallel decision boundaries in the feature space) for a given learning algorithm, the task is to find a classifier function which maximizes some criterion on the training data. A simple algorithm which enumerates over all possible classifier functions and determines thereby the maximizer might represent a sufficient learning algorithm, if the search space is finite and small enough. But, this is seldom the case, such that typical learning algorithms employ different more efficient but also not necessarily globally optimal methods to be practical.

One important concept related to the generalization property of classifiers is the *inductive bias*, which consists of a set of assumptions or restrictions related to the learning process. To generalize, we have to make some assumptions about the classifier function and the data. Amongst others, we have to precisely define *in which form* we generalize. An overly simplified but illustrative example which is often used for describing inductive reasoning is the following:

$$\left.\begin{array}{l}\textit{Socrates is a human.}\\ \textit{Socrates is mortal.}\end{array}\right\} \xrightarrow{\text{Induction}} \quad \textit{All humans are mortal.}$$

Now, there is no real justification for generalizing the mortal property to *all* humans. It is equally valid to infer that, e.g., 2 human beings are mortal or *only the next* encountered human is mortal. As one might have noticed, the generalization is obviously influenced by the choice of *operators* and/or *quantifiers* we allow and in general by the *language* in which statements are formed.

The previous simple example shows that the form of generalization can differ, and that there is no general right choice. But it is clear that these kind of decisions have to be made eventually during learning. Another example is to fit a line to a set of observed 2-dimensional points. Here, we make the assumption that the x, y coordinates have a linear relationship and have restricted the model to lines, excluding arbitrary functions. In general, we have no guarantee that the "true" relationship which generated these points might not have been a very spiky curve. Of course, by using the assumption that the examples are independent and identically distributed, we get a different situation. But, also here, in the extreme case, where we do not restrict the classifier function, we may end up with a function which represents the total memorization of the points, if we are primarily searching for a classifier function which matches the *most* with the training data. This kind of overly adapting the model to the training data and therefore losing its generalization capabilities is called *overfitting*. In total, it is essential to the learning process to impose some restrictions and make assumptions on the classifier function as well as on the data.

Consider the set of arbitrary classifier functions, then, different actual learning algorithms can be superficially characterized by imposing different explicit and implicit biases/restrictions distinguished in three categories:

- **description language or model**: What is the underlying representation language used by the classifier? For rule learning, it might be arbitrary propositional logic formulas. For perceptrons, the search space consists of all possible hyperplanes of the feature space.

- **search-method, optimization and details**: In which order, if any, is the search space examined. Is a greedy method, e.g. gradient descent, applied? Which additional criteria are used during search?

- **overfitting-avoidance method**: Are there separate mechanics for this purpose? In which form are they integrated?

In the following, we will provide only a brief overview of some common learning algorithms in its most basic form. Detailed knowledge about these algorithms are in general not necessary for this thesis, since the contributions in the following chapters are mostly independent of the used base learner. However, if the details of a particular learning algorithm are relevant, a more detailed description will be provided in the corresponding chapter (SVMs in Chapter 4, Naïve Bayes in Chapter 5 and perceptrons in Chapter 8).

2.3.1 Rule Learning

In this context, the objective is to learn a set of rules in the following form:

IF attribute-value condition(s) THEN class y

The conditions which are formed by conjunctions of propositions using the given feature representation are typically learned within the *separate-and-conquer* framework. Starting from the empty rule, i.e. the set of attribute-value conditions is empty, one tries to determine the best attribute-test according to some measure on the resulting partitioning and is added to the set of conditions. This process is repeated until some stopping criterion is met. Then, the implication of the rule is associated with the majority class of the covered instances, i.e. the instances which satisfy the conditions. Afterwards, all instances which are covered by this rule are removed from the training set (SEPARATE). The process repeats by learning the next rule (CONQUER), until, again, some stopping criteria is met. Some rule learning algorithms try to learn only rules for one class and conclude the set of rules with a so-called default-rule, which classifies all remaining instances with the other class.

Within this thesis, our choice of a rule learner in various experiments is Ripper (Cohen, 1995) in its implementation of the Weka-framework (Hall et al., 2009), called JRip.

2.3.2 Decision Trees

The generation of decision trees can be viewed as a learning process, which generates non-overlapping rules arranged in a tree. Starting with all training instances, an attribute is determined according to some criterion which generates a partitioning of the training data corresponding to the values of this attribute. The attribute is represented as a node in a graph, from which directed edges are connected to child nodes, one for each different attribute value. These child nodes represent the partitions, which satisfy the corresponding attribute-value test. This process is recursively repeated on the child nodes until all remaining instances are members of one class or met some stopping criterion. Now, if you consider each path from the root node to the leaf, it represents a rule consisting of the conjunction of attribute-value tests and an implication to a particular class. The non-overlapping property follows by considering that for each pair of distinct root-leaf paths, there is exactly one attribute-test, in which they exclude each other. This kind of recursive explicit

excluding process is commonly referred to as the *divide-and-conquer* approach in contrast to the separate-and-conquer approach from rule learning, where different rules can cover the same instance.

The above-mentioned criteria to select the next attribute is typically minimizing the *spread* of instances of each class to different partitions. Examples of measures which address this issue are Information Gain and the Gini-Index (Breiman et al., 1984). In the following chapters, we will mainly use the implementation of C4.5 (Quinlan, 1993) in Weka (Hall et al., 2009), called J48, as a representative for decision tree learners.

Hoeffding Trees

Hoeffding Trees (Domingos and Hulten, 2000) are a special kind of decision trees, which make use of the so-called *Hoeffding-bound*. It is a result known from statistics, which provides an upper bound on the probability, that the empirical mean of random variables deviates from its expected value by some value t. Consider the previously mentioned attribute-selection criterion for decision trees evaluating each attribute by a numeric value v_c in an incremental manner. Using the bound it is possible to make the statement after observing a portion of the instances, that the current best attribute is with a specifiable probability really the best attribute, if the mean of its corresponding value v_{best} has a larger distance to the mean of the second best attribute value v_{second} than t. Roughly speaking, the mean of the *random variable* v_{best} *stabilizes* with increasing number of observations. The Hoeffding bound allows to compute the number of necessary observations or instances, after which it is very unlikely that the deviation is higher than the distance to the second best attribute value. So, it is possible to select attributes and therefore build a decision tree in an incremental manner instead after observing all instances. In total, this results in a very fast incremental decision tree learner with anytime properties, i.e. it is possible to classify during the learning process. One of its main features is that its prediction is guaranteed to be asymptotically nearly identical to that of a corresponding batch learned tree.

2.3.3 Perceptrons

Perceptrons or artificial neurons (McCulloch and Pitts, 1943; Rosenblatt, 1958) are based on the human neurons in the brain. Human neurons are inter-connected and influence each other. The *activation* or *excitation* of a neuron, which can be caused for instance by seeing something, is distributed in different strengths among its connected neurons, where each neuron may react differently to the input signals. The actual functionality of a neuron is modeled in perceptrons as a simple linear combination of its input signals accompanied with a thresholding function. The weighted sum of the input signals have to satisfy some threshold to forward an activation signal and the learning task is essentially to learn the weights.

For the learning of complex functions, typically a network of artificial neurons are utilized, which are called *artificial neural networks*. However, a single artificial neuron is also often used as a classifier model, i.e. a linear function is learned which (by thresholding) discriminates one class against another one, often visualized as a hyperplane of the feature/input space, which separates instances of different classes.

2.3.4 Support Vector Machines (SVM)

Roughly speaking, support vector machines (Vapnik, 1998) also learn a linear separator but include two significant techniques. First, it is possible to alter the working feature space in an efficient way by using so-called *kernel functions*, i.e. it is possible to learn a linear separating function in some projected feature space in feasible time. Since it is nearly always possible to learn a linear separable function in a sufficient high-dimensional feature space for a given set of training instances, this technique makes a powerful tool. However, because of the resulting massive flexibility of the model, the probability of overfitting increases. One integral approach of SVMs which tries to address this issue, is to select the one separator from the possibly infinite set of linear separators, which maximizes the *margin*, i.e. the distance between the hyperplane to the nearest positive example plus the distance from it to the nearest negative example. This margin-maximization principle has to be shown to have desireable properties with respect to generalization (Shawe-Taylor et al., 1998; Smola et al., 2000).

In this thesis, we will utilize three different implementations/variants of SVMs: an SVM implementation in Weka called SMO, the implementation of LibLinear (Fan et al., 2008) and the one from LibSVM (Chang and Lin, 2011).

2.3.5 Naïve Bayes

Naïve Bayes is one of the probabilistic approaches to classification. It is assumed, that there exists a joint probability distribution of instance-class pair events and the classification objective is to return the most probable class, after observing a test instance. Thus, basically, the learning of such a model is primarily focused with estimating relevant probabilities from training data. Since the estimation of the probability distribution requires a very large number of instances, the direct approach is in general not practical. Naïve Bayes makes a probabilistic model amenable by imposing a strong assumption about the distribution. It uses a "naive" assumption, namely that the attributes are conditionally independent given the class. In combination with the Bayes Theorem, this makes the training process tremendously efficient and feasible.

Again, we used in this thesis the Naïve Bayes implementation of Weka, denoted by NB.

Part I

Multiclass Classification

3 Multiclass Preliminaries

Contents

Multiclass classification extends binary classification by considering more than two classes. For multiclass classification problems, besides direct multiclass-capable learning algorithms, the common approaches are so-called decomposition-based approaches, which reuse binary learners from the previous chapter.

This chapter contains a brief recapitulation of basic knowledge about multiclass classification and decomposition-based approaches for this task. We will focus particularly only on necessary information for this thesis.

3.1 Multiclass Setting

The multiclass classification setting is identical to the binary classification setting except that now the set of classes $K = \{c_i \mid i = 1 \dots k\}$ is not anymore restricted to two classes, i.e. $k \geq 2$ instead of $k = 2$. It is assumed, that there exists at least one instance for each class. Moreover, the number of instances t is typically significantly greater than the number of classes k in order to make any reasonable induction.

One typical example of multiclass classification problems is optical character recognition. Here, the task is to recover a text, which is given in an image-representation. Seen as a classification task, a natural approach is to consider each alphabetic and numeric character as a class. This task becomes non-trivial for hand-written texts and also for machine-generated texts, e.g., by considering the multitude of different font styles.

3.2 Multiclass Evaluation and Variants

Nearly all previously introduced measures and setting variants for binary classification apply straight-forwardly also for multiclass classification. We will briefly describe the few exceptions in the following text.

The previously introduced measures true positive rate and true negative rate, which were essentially class-based accuracy values apply also for the multiclass setting. However, there is no more the distinction between *the* positive and *the* negative class, such that both notions lose the association to a particular class. The underlying semantic becomes essentially redundant if we neglect this. Thus, often only one term, the true positive rate *regarding* a particular class c_a is used to denote the class-based accuracy of c_a.

Similarly, the semantics of the notions of false negative and false positive rates have to be slightly altered. Again, the semantic for both notions changes. Not the misprediction rate of positive or negative instances are determined respectively, but *all* mispredictions predicting a particular class. Here also, one usually agrees on one term, the false positive rate regarding a particular class, to denote the measure in this sense.

Regarding the AUC, there is no consensus for the multiclass case, but various approaches which tackle the multiclass ROC analysis by projecting them to binary ones were proposed in the literature (Hand and Till, 2001; Provost and Domingos, 2003).

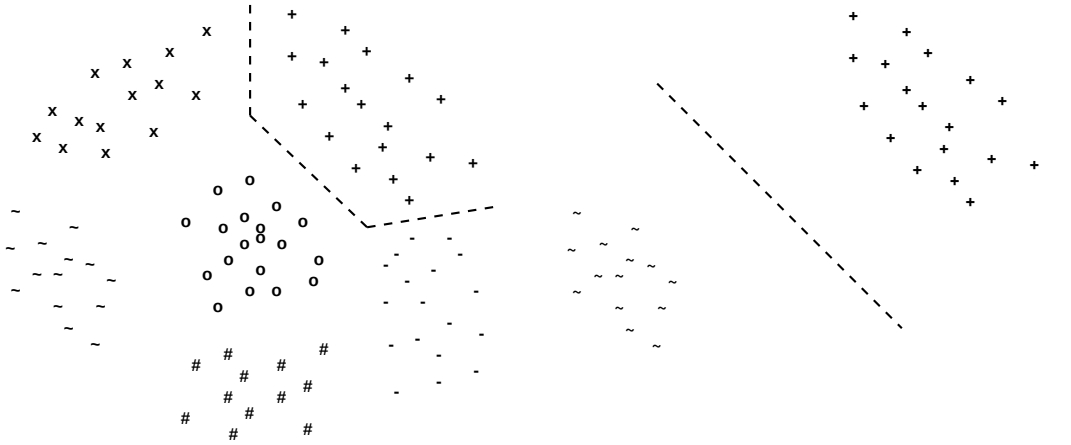
3.3 Common Decomposition-Based Approaches

Many learning algorithms can only deal with two-class problems. For multiclass problems, they have to rely on *binary decomposition* (or *binarization)* procedures that transform the original learning problem into a series of binary learning problems. In the following, we will recapitulate the well-known *one-against-all* and *one-against-one* decompositions, which will be used throughout this thesis. Afterwards, in the next section, we will describe a general framework for decomposition-based approaches.

3.3.1 One-Against-All (OAA)

A standard solution for this problem is the *one-against-all* approach, also known as *one-against-rest*, which constructs one binary classifier f_i for each class c_i, where the positive training examples are those belonging to this class and the negative training examples are formed by the union of all other classes, i.e. all remaining examples. An illustration of this decomposition scheme is shown in Figure 3.1a on the facing page.

At prediction time, all classifiers are queried on the given test instance. If exactly one classifier f_i predicts the instance as positive, the corresponding class c_i is returned as the overall prediction. For the other cases, i.e., if more than one or none classifier predicts the instance as positive, usually two solutions are possible. If the classifiers return score, confidence or probability values instead of a binary value (negative or



(a) One-Against-All Classification: k classifiers, each separates one class from all other classes. Here: + against all other classes

(b) Pairwise Classification: $\frac{k(k-1)}{2}$ classifiers, one for each pair of classes. Here: + against $\sim$.

Figure 3.1: One-against-all and pairwise binarization (Fürnkranz, 2002)

positive), one can return the class, which corresponding classifier maximizes this measure. In the case, that the maximal value is not unique, or the classifiers return only binary values, one usually applies a random tie-breaking, i.e. randomly selecting one class among the set of predicted classes.

3.3.2 Pairwise Classification (OAO)

Pairwise classification also known as *round-robin classification* (Fürnkranz, 2002; Wu et al., 2004) and *one-against-one* is besides one-against-all one of the well-known decomposition schemes. This approach has been shown to produce more accurate results than the one-against-all approach for a wide variety of learning algorithms such as support vector machines (Hsu and Lin, 2002) or rule learning algorithms (Fürnkranz, 2002). Further support is given by a recent extensive experimental study of Galar et al. (2011).

The key idea of pairwise classification is to learn one classifier for each pair of classes. At classification time, the prediction of these classifiers are then combined into an overall prediction.

Training Phase

A pairwise or round robin classifier trains a set of $k(k-1)/2$ *binary classifiers* $f_{i,j}$, one for each pair of classes (c_i, c_j), $i < j$. Each binary classifier is only trained on the subset of training examples belonging to classes c_i and c_j, all other examples are ignored[1] for the training of $f_{i,j}$. An example of this procedure is shown in Figure 3.1b.

1 Several extensions of the pairwise approach, such as Tri-Class SVMs (Angulo et al., 2006) and Pairwise Correcting Classifiers (Moreira and Mayoraz, 1998), also integrate the remaining examples

We will refer to the learning algorithm that is used to train these classifiers $f_{i,j}$ as the *base learner*. We will also say that classifier $f_{i,j}$ is *incident* to classes c_i and c_j.

It is important to note that the total effort required to train the entire ensemble of the $k(k-1)/2$ classifiers is only linear in the number of classes k, and, in fact, cheaper than the training of a one-against-all ensemble. It is easy to see this, if one considers that in the one-against-all case each training example is used k times (namely in each of the k binary problems), while in the round robin approach each example is only used $k-1$ times, namely only in those binary problems, where its own class is paired against one of the other $k-1$ classes (cf. also Fürnkranz, 2002).

Typically, the binary classifiers are *class-symmetric*, i.e., the classifiers $f_{i,j}$ and $f_{j,i}$ are identical. However, for some types of classifiers this does not hold. For example, standard rule learning algorithms will always learn rules for the positive class, and classify all uncovered examples as negative. Thus, the predictions may depend on whether class c_i or class c_j has been used as the positive class. As has been noted in (Fürnkranz, 2002), a simple method for solving this problem is to average the predictions of $f_{i,j}$ and $f_{j,i}$, which basically amounts to the use of a so-called *double round robin* procedure, where we have two classifiers for each pair of classes.

Prediction Phase

At classification time, each binary classifier $f_{i,j}$ is queried and issues a vote (a prediction for either c_i or c_j) for the given example. This can be compared with sports and games tournaments, in which each player plays each other player once. In each game, the winner receives a point, and the player with the maximum number of points is the winner of the tournament.

In this thesis, we will assume binary classifiers $f_{i,j}$ that return class probabilities $p(c_i \mid c_i \vee c_j)$ and $p(c_j \mid c_i \vee c_j)$ if not stated otherwise. These can be used for *weighted voting* (also called max-wins), i.e., we predict the class c^* that receives the maximum weighted number of votes:

$$c^* = \operatorname*{argmax}_{c \in K} \sum_{c' \in K \setminus c} p\left(c \mid c \vee c'\right)$$

Other choices for decoding pairwise classifiers are possible (cf., e.g., Wu et al., 2004; Hastie and Tibshirani, 1997), but voting is surprisingly stable. For example, one can show that weighted voting, where each binary vote is split according to the probability distribution estimated by the binary classifier, minimizes the Spearman's rank correlation coefficient with the correct ranking of classes, provided that the classifier provides good probability estimates (Hüllermeier et al., 2008). Also, empirically and theoretically, weighted voting seems to be a fairly robust method that is hard to beat with other, more complex methods (Hüllermeier and Fürnkranz, 2004; Hüllermeier and Vanderlooy, 2010).

into the training process. In several experiments, this has lead to an improved performance, which has to be paid with a considerable increase in training time, and more complex decision boundaries for the involved classifiers.

3.3.3 Efficient Pairwise Prediction (QWeighted)

Although the training effort for the entire ensemble of pairwise classifiers is only linear in the number of examples, at prediction time we still have to query a quadratic number of classifiers, which can be very inefficient for a high number of classes k. In this section, we discuss a recently proposed algorithm (Park, 2006; Park and Fürnkranz, 2007) that allows to significantly reduce the number of classifier evaluations in practice without changing the prediction of the ensemble.

Key Idea

Weighted or unweighted voting predicts the top rank class c^* by returning the class with the highest accumulated voting mass after evaluation of all pairwise classifiers. During such a 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. Consider the following simple example: Given k classes and an arbitrary number $j \in \mathbb{N}$ with $j < k$, if, currently, class c_b has lost j votings and class c_a has received more than $k - j$ votes, it is impossible for c_b to achieve a higher total voting mass than c_a. Thus further evaluations involving c_b can be safely ignored for the comparison of these two classes.

To increase the reduction of evaluations we are interested in obtaining such exploitable situations frequently. 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 specifically, the loss l_i of a class c_i is defined as $l_i := p_i - v_i$, where p_i is the number of evaluated incident classifiers of c_i and v_i is the current vote amount of c_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.

The QWeighted Algorithm

Algorithm 1 on the next page shows the QWeighted algorithm, which implements this idea. First, the pairwise classifier $f_{a,b}$ will be selected for which the losses l_a and l_b of the relevant classes c_a and c_b are minimal, provided that the classifier $f_{a,b}$ 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 $f_{a,b}$ (recall that v_{ab} is interpreted as the amount of the voting mass of the classifier $f_{a,b}$ that goes to class c_a and $1 - v_{ab}$ is the amount that goes to class c_b). These two steps will be repeated until all classifiers for the class c_m with the minimal loss has been evaluated. Thus the current/estimated loss l_m is the correct loss for this class. As all other classes already have a greater or equal loss and considering that the losses are monotonically increasing, c_m is the correct *top rank* class or among the set of equal classes with minimal loss.

Algorithm 1 QWeighted

Require: pairwise classifiers $f_{i,j}$ with $1 \leq i < j \leq k$, testing instance $x \in X$

1: $\vec{l} \in \mathbb{R}^k \leftarrow 0$ # loss values vector
2: $c^* \leftarrow$ NULL
3: $G \leftarrow \emptyset$ # keep track of evaluated classifiers
4: **while** $c^* =$ NULL **do**
5: $c_a \leftarrow \underset{c_i \in K}{\text{argmin}}\ l_i$ # select top candidate class
6: $c_b \leftarrow \underset{c_j \in K \setminus \{c_a\},\ f_{a,j} \notin G}{\text{argmin}}\ l_j$ # select second
7: **if** no c_b exists **then**
8: $c^* \leftarrow c_a$ # top rank class determined
9: **else** # evaluate
10: $v_{ab} \leftarrow f_{a,b}(x)$ # one vote for c_a ($v_{ab} = 1$) or c_b ($v_{ab} = 0$)
11: $l_a \leftarrow l_a + (1 - v_{ab})$ # update voting loss for c_a
12: $l_b \leftarrow l_b + v_{ab}$ # update voting loss for c_b
13: $G \leftarrow G \cup f_{a,b}$ # update already evaluated classifiers
14: **return** c^*

Theoretically, a minimal number of comparisons of $k - 1$ is possible (*best case*). Assuming that the incident classifiers of the correct top rank c^* always return the maximum voting amount ($l^* = 0$), c^* is always in the set $\{c_j \in K \mid l_j = \min_{c_i \in K} l_i\}$. In addition, c^* should be selected as the first class in step 1 of the algorithm among the classes with the minimal loss value. It follows that exactly $k - 1$ comparisons will be evaluated, more precisely all incident classifiers of c^*. The algorithm terminates and returns c^* as the correct top rank.

The *worst case*, on the other hand, is still $k(k-1)/2$ 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.

QWeighted will always predict the same class as the full pairwise classifier except for ambiguous predictions, but the actual runtime complexity of the algorithm is close to linear in the number of classes, in particular for large numbers of classes, where the problem is most stringent. For very hard problems, where the performance of the binary classifiers reduces to random guessing, its worst-case performance is still quadratic in the number of classes as mentioned before, but even there practical gains can be expected. These properties of QWeighted are based on empirical evaluations done in (Park and Fürnkranz, 2007) and will be re-validated later in Section 6.2.1 on page 63 on extended experiments.

Alternative Approaches

The loss l_i, which we use for selecting the next classifier, is essentially identical to the voting-against principle introduced by Cutzu (2003a,b), who also observed that it

allows to reliably conclude a class when not all of the pairwise classifiers are present. For example, Cutzu claims that using the voting-against rule one could correctly predict class c_i even if none of the incident pairwise classifiers $f_{i,j}$ $(j = 1 \dots k, j \neq i)$ are used. However, this argument is based on the assumption that all base classifiers classify correctly. Moreover, if there is a second class c_j that should ideally receive $k-2$ votes, voting-against could only conclude a tie between classes c_i and c_j, as long as the vote of classifier $f_{i,j}$ is not known. The main contribution of his work, however, is a method for computing posterior class probabilities in the voting-against scenario.

The voting-against principle was already used by Platt et al. (1999) earlier in the form of DDAGs (Decision Directed Acyclic Graphs), which organize the binary base classifiers in a decision graph. Each node represents a binary decision that rules out the class that is not predicted by the corresponding binary classifier. At classification time, only the classifiers on the path from the root to a leaf of the tree (at most $k-1$ classifiers) are consulted. While the authors empirically show that the method does not lose accuracy on three benchmark problems, it does not have the guarantee of QWeighted, which will always predict the same class as the full pairwise classifier. Intuitively, one would also presume that a static evaluation routine that uses only $k-1$ of the $k(k-1)/2$ base classifiers will sacrifice one of the main strengths of the pairwise approach, namely that the influence of a single incorrectly trained binary classifier is diminished in a large ensemble of classifiers (Fürnkranz, 2003). Our empirical results (presented in Section 6.2.1 on page 63) will confirm that DDAGs are only slightly more efficient but less accurate than the QWeighted approach.

3.4 Error-Correcting Output Codes (ECOCs)

Error-correcting output codes (ECOCs) (Dietterich and Bakiri, 1995) are a general framework for decomposition-based multiclass classification methods. This framework unifies common approaches such as the previously described one-against-one and one-against-all. ECOCs have its origin in coding and Information Theory (MacWilliams and Sloane, 1983; Gallager, 1968), where it is used for detecting and correcting errors in suitably encoded signals. In the context of classification, we *encode* the class variable with an n-dimensional binary *code word*, whose entries specify whether the example in question is a positive or a negative example in the corresponding binary classifier.

3.4.1 Binary ECOCs

Formally, each class c_i $(i = 1 \dots k)$ is associated with a so-called *code word* $\vec{cw}_i \in \{-1, 1\}^n$ of length n. We denote the j-th bit of $\vec{cw}_i$ as $b_{i,j}$. In the context of ECOC, all relevant information is summarized in a so-called *coding matrix* $(m_{i,j}) = M \in \{-1, 1\}^{k \times n}$, whose i-th row describes code word $\vec{cw}_i$, whereas the j-th column represents a classifier f_j. The set of all such classifiers is denoted as $C = \{f_1, \dots, f_n\}$.

Furthermore, the coding matrix implicitly describes a decomposition scheme of the original multiclass problem. In each column j the rows contain a (1) for all classes whose training examples are used as positive examples, and (-1) for all negative examples for the corresponding classifier f_j.

$$M = \begin{pmatrix} 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 & 1 & 1 \end{pmatrix}$$

The previous example shows a coding matrix for 4 classes (rows), which are encoded with 6 classifiers (columns). The first classifier uses the examples of classes 1 and 2 as positive examples, and the examples of classes 3 and 4 as negative examples.

At prediction time, all binary classifiers are queried, and collectively predict an n-dimensional vector, which must be *decoded* into one of the original class values, e.g., by assigning it to the class of the closest code word. More precisely, for the classification of a test instance x, all binary classifiers are evaluated and their predictions, which form a *prediction vector* $\vec{p} = [f_1(x), f_2(x), \ldots, f_n(x)]$, are compared to the code words. The class c^* whose associated code word $c\vec{w}_{c^*}$ is "nearest" to $\vec{p}$ according to some distance measure $d(.)$ is returned as the overall prediction, i.e.

$$c^* = \underset{c}{\operatorname{argmin}} \; d\left(c\vec{w}_c, \vec{p}\right)$$

For computing the similarity between the prediction vector and the code word, the most common choice is the Hamming Distance, which measures the number of bit positions in which the prediction vector $\vec{p}$ differs from a code word $c\vec{w}_i$.

$$d_H\left(c\vec{w}_i, \vec{p}\right) = \sum_{j=1}^{n} \frac{|m_{i,j} - p_j|}{2} \tag{3.1}$$

Typically, the number of classifiers exceeds the number of classes, i.e., $n > k$. This allows for longer code words, so that the mapping to the closest code word is not compromised by individual mistakes of a few binary classifiers. Thus, ECOCs not only make multiclass problems amenable to binary classifiers, but may also yield a better predictive performance than conventional multiclass classifiers.

The good performance of ECOCs has been confirmed in subsequent theoretical and practical work. For example, it has been shown that ECOCs can to some extent correct variance and even bias of the underlying learning algorithm (Kong and Dietterich, 1995). An exhaustive survey of this area can be found in (Windeatt and Ghaderi, 2003).

3.4.2 Ternary ECOCs

Conventional ECOCs as described in the previous section always use all classes and all training examples for training each binary classifier. Thus, binary decompositions

which use only parts of the data (such as pairwise classification) can not be modeled in this framework.

Allwein et al. (2000) extended the ECOC approach to the ternary case, where code words are now of the form $c\vec{w}_i \in \{-1, 0, 1\}^n$. The additional code $m_{i,j} = 0$ denotes that examples of class c_i are ignored for training classifier f_j. We will sometimes also denote a classifier f_j as f_{P_j,N_j}, where P_j is the set of classes that are used as positive examples, and N_j is the set of all classes that are used as negative examples. We will adapt the notion of incidency (from pairwise classifiers) and say that a ECOC classifier $f_j = f_{P_j,N_j}$ is *incident* to a class c_i, if the examples of c_i are either positive or negative examples for f_j, i.e., if $c_i \in P_j$ or $c_i \in N_j$, which implies that $m_{i,j} \neq 0$.

This extension increases the expressive power of ECOCs, so that now nearly all common multiclass binarization methods can be modeled. For example, pairwise classification (Section 3.3.2 on page 17), where one classifier is trained for each pair of classes, could not be modeled in the original framework, but can be modeled with ternary ECOCs. Its coding matrix has $n = k(k-1)/2$ columns, each consisting of exactly one positive value $(+1)$, exactly one negative value (-1), and $k-2$ zero values (0). Below, we show the coding matrix of a pairwise classifier for a 4-class problem.

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix} \tag{3.2}$$

The conventionally used Hamming decoding can be adapted to this scenario straight-forwardly. Note that while the code word can now contain 0-values, the prediction vector is considered as a set of binary predictions which can only predict either -1 or 1. Thus, a zero symbol in the code word ($m_{i,j} = 0$) will always increase the distance by $\frac{1}{2}$ (independent of the actual prediction).

Many alternative decoding strategies have been proposed in the literature. Along with the generalization of ECOCs to the ternary case, Allwein et al. (2000) proposed a loss-based strategy. Escalera et al. (2006) discussed the shortcomings of traditional Hamming distance for ternary ECOCs and presented two novel decoding strategies, which should be more appropriate for dealing with the zero symbol. We will address them in Section 6.1.5 on page 60.

3.4.3 Code Design for (Ternary) ECOCs

A well-known theorem from coding theory states that if the minimal Hamming Distance between two arbitrary code words is h, the error detection and correction framework is capable of correcting up to $\lfloor \frac{h}{2} \rfloor$ bits. This is easy to see, since every code word $c\vec{w}_i$ has a $\lfloor \frac{h}{2} \rfloor$ neighborhood, for which every code in this neighborhood is nearer to $c\vec{w}_i$ than to any other code word. Thus, it is obvious that good error correction crucially depends on the choice of a suitable coding matrix.

Unfortunately, some of the results in coding theory are not fully applicable to the machine learning setting. For example, the above result assumes that the bit-wise error is independent, which leads to the conclusion that the minimal Hamming Distance is the main criterion for a good code. But this assumption does not necessarily hold in machine learning. Classifiers are learned with similar training examples and therefore their predictions tend to correlate (as noted in Dietterich and Bakiri, 1995). Thus, a good ECOC code also has to consider, e.g., column distances, which may be taken as a rough measure for the independence of the involved classifiers, since they resemble to some extent training set overlaps.

In the machine-learning literature, a considerable amount of research has been devoted to code design for ternary ECOCs (see, e.g., Crammer and Singer, 2002b; Pimenta et al., 2008), but without reaching a clear conclusion. We want to emphasize that this thesis does not contribute to this discussion, because we will mainly not be concerned with comparing the predictive quality of different coding schemes.

Nevertheless, we will briefly review common coding schemes, which will be used in experimental evaluations in some of the following chapters.

Exhaustive Ternary Codes

These codes cover *all* possible classifiers involving a given number of classes $l \leq k$. More formally, a (k, l)-exhaustive ternary code defines a ternary coding matrix M, for which every column j contains exactly l non-zero values, i.e., $\forall j \in \{1, \ldots, n\}. \sum_{i \in K} |m_{i,j}| = l$. Obviously, in the context of multiclass classification, only columns with at least one positive $(+1)$ and one negative (-1) class are useful. The following example shows a $(4, 3)$-exhaustive code.

$$M = \begin{pmatrix} 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 1 & -1 \\ -1 & 1 & 1 & 0 & 0 & 0 & 1 & -1 & 1 & 1 & -1 & 1 \\ 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 \end{pmatrix} \tag{3.3}$$

The number of classifiers for a (k, l) exhaustive ternary code is $\binom{k}{l}(2^{l-1} - 1)$, since the number of *binary* exhaustive codes is $2^{l-1} - 1$ and the number of combinations to select l row positions from k rows is $\binom{k}{l}$. These codes are a straight-forward generalization of the exhaustive binary codes, which were considered in the first works on ECOC (Dietterich and Bakiri, 1995), to the ternary case. Note that $(k, 2)$-exhaustive codes correspond to pairwise classification.

In addition, we define a *cumulative* version of exhaustive ternary codes, which subsumes all (k, i)-exhaustive codes with $i = 2, 3, \ldots, l$ up to a specific level l. In this case, we speak of (k, l)-cumulative exhaustive codes, which generate a total of $\sum_{i=2}^{l} \binom{k}{i}(2^{i-1} - 1)$ columns. For a dataset with k classes, (k, k)-cumulative exhaustive codes represent the set of all possible binary classifiers.

Random Codes

We consider two types of randomly generated codes. The first variant allows to control the probability distribution of the set of possible symbols $\{-1, 0, 1\}$ from which random columns are drawn. By specifying a parameter $r_{zp} \in [0, 1]$, the probability for the zero symbol is set to $p(\{0\}) = r_{zp}$, whereas the remainder is equally subdivided to the other symbols: $p(\{1\}) = p(\{-1\}) = (1 - r_{zp})/2$. This type of code allows to control the *sparsity*, i.e. the fraction of the zero symbol in the coding matrix, which will be useful for evaluating which factors determine the performance of our ECOC-based algorithms.

The second random code generation method selects randomly a subset from the set of all possible classifiers C. This set of classifiers C equals the cumulative ternary code matrix where the used level l equals the number of classes k. Obviously, this variant guarantees that no duplicate classifiers are generated, whereas it can occur in the other variant. We do not enforce this, because we wanted to model and evaluate two interpretations of randomly generated codes: randomly filled matrices and randomly selected classifiers.

Coding Theory, BCH Codes

Many different code types were developed within coding theory. We pick the so-called BCH Codes (Bose and Ray-Chaudhuri, 1960) as a representative, because they have been studied in depth and have properties which are favorable in practical applications. For example, the desired minimum Hamming distance of M can be specified, and fast decoding methods are available. Note, however, that efficient decoding in coding theory has the goal to minimize the complexity of finding the nearest code word given the received *full* code word. In the following chapters, we are mainly interested in minimizing the classifier evaluations, and this relates to using the minimum number of *bits* of the receiving code word to estimate the nearest code word respectively class. Although some concepts of efficient decoding in coding theory seem to be transferable to our setting, they lack the capability to be a general purpose decoding method for arbitrary coding matrices.

BCH codes represent a special class of *cyclic* codes, i.e. codes (a mapping from one alphabet to another one, here called code words) for which cyclic shifts of any code word is again a specified code word. So, if $\vec{cw} = (cw_1, cw_2, \ldots, cw_n)$ is a specified code word, then $\mathsf{shift}(\vec{cw}) = (cw_n, cw_1, \ldots, cw_{n-1})$ is also a specified code word. Furthermore, every cyclic code has a special code word $\vec{cw}_G$, called *generator polynomial* $g(x)$, such that each code word is a multiple (in finite field arithmetic) of $\vec{cw}_G$ and that for every multiple of $\vec{cw}_G$ there exists a corresponding code word. The polynomial term is due to the commonly used representation form for elements of finite fields, i.e. the corresponding polynomial for code word $\vec{cw} = (cw_1, cw_2, \ldots, cw_n)$ is denoted by $cw(x) = cw_1 + cw_2 x + \cdots + cw_n x^{n-1}$. Now, if the generator polynomial $g(x)$ satisfies certain properties then one can show that the resulting code words have a prespecified minimum distance.

Definition (Binary BCH code)
A cyclic code of length n over $GF(2)$ is a binary BCH code of designed distance δ if, for some integer $b \geq 0$,

$$g(x) = \text{lcm}\big(M^b(x), M^{b+1}(x), \ldots, M^{b+\delta-2}(x)\big)$$

where lcm(.) denotes the least common multiple of its arguments and $M^i(x)$ denotes the *minimal polynomial* of x^i, i.e. the least-degree polynomial with $M^i(x^i) = 0$.

This brief description of BCH codes is based on (MacWilliams and Sloane, 1983), to which we also refer for a detailed description of this code family and further information regarding error-correcting codes.

Domain-Dependent Codes

The previous code-types have in common, that the actual data is neglected. Only the number of classes k from the data at hand is used in the code generation process.

In contrast, domain-dependent codes project data-specific relationships or expert knowledge explicitly to the coding matrix. For example, the knowledge of an inherent hierarchy or order among the classes can be used to model classifiers which exploit this information (e.g., Melvin et al., 2007; Cardoso and da Costa, 2007). Another interesting direction of generating a data-based code is considered by Pujol et al. (2006). Their proposed algorithm DECOC tries to generate a coding matrix, whose columns consist of the best discriminating classifiers on the considered dataset. By applying only classifiers with the maximum discriminative ability, they expect to maximize the overall prediction accuracy. Also, it seems to be rather efficient, since they restrict the length of the coding matrix.

4 Efficient ECOC Training by Exploiting Code-Redundancies

Contents

In this chapter, we will present our first contribution. We focus on the training phase of ECOCs, where overlaps of training instances in highly *redundant* codes are reduced, and therefore its overall training complexity, without altering the models. This is done by identifying shared subproblems in the ensemble, which need to be learned only once, and by rescheduling the binary classification problems so that these subproblems can be reused as often as possible. This approach is directly feasible in conjunction with incremental base learners, but its main idea is still applicable for the more interesting case when SVMs are used as base learners, by reusing computed weights of support vectors from related subproblems and applying an adapted ensemble caching strategy.

We will discuss the redundancies within ECOCs in Section 4.1 and present an algorithm to exploit them in Section 4.2. The performance of this algorithm is then evaluated for Hoeffding Trees and for SVMs as base classifiers (Section 4.3). Finally, we will discuss the results and elaborate on the limitations of this approach.

4.1 Code Redundancy in ECOC

Many code types specify classifiers which share a common code configuration. For instance, in the case of (k,l)-cumulative exhaustive codes with $k \geq l \geq 3$, we can

construct a *subclassifier* by setting some $+1$ bits and/or some -1 bits of a specified classifier to zero. Clearly, the resulting classifier is itself a valid classifier that occurs in the ECOC matrix of this cumulative code. Furthermore, every classifier f of length $l' < l$, is subclassifier of exactly $2 \cdot (k - l')$ classifiers with length $l' + 1$, since there are $k - l'$ remaining classes and each class can be specified as positive or negative[1]. Such redundancies also occur frequently in random codes with a probability of the zero-symbol smaller than 0.5, and therefore also in the special case of random dense codes, where the codes consists only of $+1$ and -1 symbols. On the other hand, the widely used one-against-one code has no code redundancy, and the redundancy of the one-against-all code is very low.

In general, the learning of a binary classifier is independent of the explicit specification, which class of instances is regarded as positive and which one as negative (cf. *class-symmetric* property in Section 3.3.2 on page 17). So, from a learning point of view, the classifier specified by a column $\vec{m}_i = (m_{1i}, \ldots, m_{ki})$ is equivalent to $-\vec{m}_i$.

Definition (Code Redundancy)
Let f_i and f_j be two classifiers and $(m_{1i}, \ldots, m_{ki})$ and $(m_{1j}, \ldots, m_{kj})$ their corresponding (ternary) ECOC columns. We say f_i and f_j are $p-$*redundant*, if for $a \in \{1 \ldots k\}$,

$$p = \max\left(\left|\{a \mid m_{ai} = m_{aj}, m_{ai} \neq 0\}\right|, \left|\{a \mid m_{ai} = -m_{aj}, m_{ai} \neq 0\}\right|\right)$$

To elaborate, let $d = \max(d_H(\vec{m}_i, \vec{m}_j), d_H(-\vec{m}_i, \vec{m}_j))$, where d_H is the Hamming distance. Two classifiers f_i and f_j are p-redundant, if and only if $k - p = d - \left|\left\{a \in \{1 \ldots k\} \mid m_{ai} = 0 \wedge m_{aj} = 0\right\}\right|$. Thus, in essence, classifier redundancy is the opposite of Hamming distance except that bit positions with equal zero values are ignored. For convenience, similarly to the symmetric difference operator $\triangle$ of sets, we denote for two classifiers f_i and f_j the set of classes which are only involved in one of their code configurations m_i and m_j as $f_i \bigtriangledown f_j$. More precisely, $f_i \bigtriangledown f_j = \left\{c_a \mid a \in \{1 \ldots k\} \wedge |m_{ai}| + |m_{aj}| = 1\right\}$. In addition, we speak of a *specified* classifier, if there exists a corresponding code-column in the given ECOC matrix.

4.2 Exploitation of Code Redundancies

Code redundancies can be directly exploited by incremental base learners, which are capable of extending an already learned model on additional training instances. Then, repeated iterations over the same instances can be avoided, since shared subclassifiers only have to be learned once. The key issue is to find a *training protocol* that maximizes the use of such shared subclassifiers, and therefore minimizes the redundant computations. Note that the subclassifiers do not need to be specified classifiers, i.e., they do not need to correspond to a class code in the coding matrix.

1 Here, the length of a classifier is understood as the number of classes, which are used for learning the classifier, i.e. the number of *incident* classes.

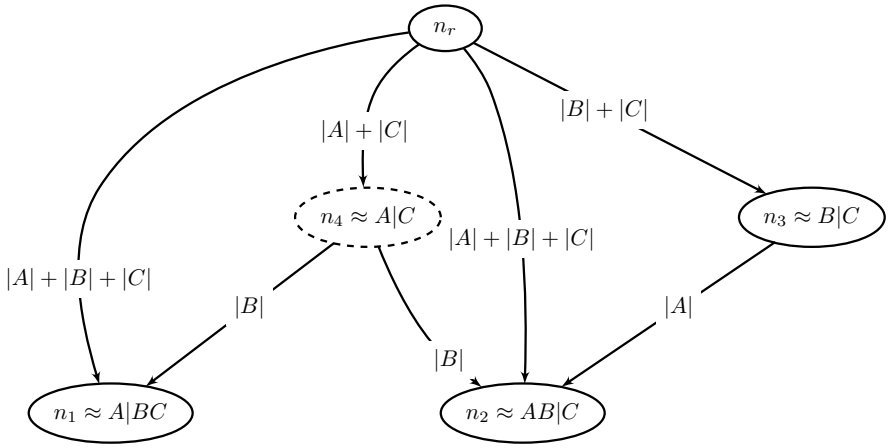


Figure 4.1: Training graph example: Three classifiers $f_1 = A|BC$, $f_2 = AB|C$ and $f_3 = B|C$ are specified. The non-specified classifier $f_4 = A|C$ is added because it is the maximal common subclassifier of f_1 and f_2. For each edge $e_{ij} = (n_i, n_j)$ the weights depict the training effort for learning classifier f_j based on classifier f_i ($|A|$ is the no. of training instances of class A).

This task may be viewed as a graph-theoretic problem. Let $G = (V, E)$ be a weighted directed graph with $V = \{n_r\} \cup \{f_i\} \cup \{f_s\}$, i.e., each classifier f_i and each possible subclassifier f_s are in the set of nodes V. Furthermore, the special *root* node n_r is connected to every other node $n_i \in V$ with the directed edge (n_r, n_i). Besides, for each two non-root nodes n_i and n_j, there exists a directed edge (n_i, n_j), if and only if n_i is subclassifier of n_j. The weight of these edges is $f_j \bigtriangledown f_i$. For all edges (n_r, n_i), which are incident to the root node, the weight is the number of training instances involved in classifier f_i.

To elaborate, incident edges to the root node depict classifiers which are learned by batch learning. All other edges (n_i, n_j), which are edges between two (sub)-classifiers, represent incremental learning steps. Based on the learned model of classifier f_i, the remaining training instances of $f_j \bigtriangledown f_i$ are used to learn classifier f_j. The multiple possible paths to one particular classifier represents the possible ways to learn it. Each of these paths describe a different partitioning of training costs, represented by the number of edges (number of partitions) and edge weights (size of the partitions). Considering only one classifier, the cost for all paths are identical. But, by considering that paths of different redundant classifiers can overlap, and that shared subpaths are trained only once, the total training cost can be reduced. Another view at this graph is the following: every subgraph of G which is an arborescence consisting of all specified classifiers is a *valid* scheme for learning the ensemble, in the sense that it produces exactly the specified set of classifiers.

In this context, our optimization problem is to find a minimum-weight subgraph of G including all classifier nodes f_i, which relates to minimizing the processed training instances for the set of specified classifiers and therefore total training complexity of the ECOC ensemble. Note, this problem is known in graph theory as *Steiner problem in a directed graph*, which is NP-hard (Wong, 1984).

Figure 4.1 shows an example of such a *training graph* for a 3-class problem, where three classifiers $f_1 = A|BC$, $f_2 = AB|C$ and $f_3 = B|C$ are specified by a given ECOC matrix. A, B, C are symbol representatives for classes and $A|B$ describes the binary classifier which discriminates instances of class A against B. The standard training scheme, which learns each classifier separately, can be represented as a subgraph $G_1 \subseteq G$ consisting of $V_1 = \{n_r, n_1, n_2, n_3\}$ and $E_1 = \{e_{r1}, e_{r2}, e_{r3}\}$. This scheme uses $2|A| + 3|B| + 3|C|$ training instances in total. An example where fewer training instances are needed is $G_2 = (V_1, E_2)$ with $E_2 = \{e_{r1}, e_{r3}, e_{32}\}$, which exploits that classifier f_2 can be incrementally trained from f_3, resulting in training costs $2|A| + 2|B| + 2|C|$. Another alternative is to add a non-specified classifier $f_4 = A|C$ to the graph, resulting in $G_3 = (V, E_3)$ with $E_3 = \{e_{r4}, e_{41}, e_{42}, e_{r3}\}$ with training costs $|A| + 3|B| + 2|C|$. It is easy to see that either G_2 or G_3 is the optimal Steiner tree in this example and that both process fewer training examples than the standard scheme. Whether G_2 or G_3 is optimal, depends on whether $|A| > |B|$.

Since the optimal solution is in general hard to compute, we use a greedy approach. We first have to generate the training graph. Then, we iteratively remove local non-optimal edges, starting from the leaf nodes (specified classifiers) up to the root. Both methods are described in detail in the following subsections.

4.2.1 Generation of Training Graph

We consider an algorithm which is particularly tailored for exhaustive and cumulative exhaustive codes. Let C be the set of all classifiers f of a specific length l, which is successively decreased from k down to 2. For each pair $(f_i, f_j) \in C \times C$ the maximal common subclassifier f_s is determined and eventually integrated into the graph. Then, these classifiers are marked as processed ($\mathsf{seen}(f) = 1$) and are not considered in the following steps of the generation algorithm. Level l is decreased and the algorithm repeats. The processed classifiers can be ignored, because for the systematic codes (exh. and cumulative exh.) all potential subclassifiers can be constructed using its immediate subclassifiers. This algorithm does not find all edges for random codes or general codes, but only for their inherent systematic code structures. For the sake of efficiency and also considering that we employ a greedy Steiner tree Solving procedure afterwards, we neglect this fact.

A pseudo code is given in Algorithm 2 on the next page. Note that there, the set C is populated with classifiers of length greater equal than l instead of exactly l, considering the special case that there can be multiple levels with zero classifiers or only one classifier. Also, classifiers should only be flagged as *processed* if they were actually checked at least once. The complexity of this version is exponential, but it will be later reduced to quadratic in combination with the greedy Steiner tree algorithm.

In the beginning, for each specified classifier f_i a corresponding node n_i is generated in the graph and connected with the root node by the directed edge (n_r, n_i). In the main loop, which iterates over $l = n$ down to 2, for each pair (f_i, f_j) of classifiers of length l the maximum common subclassifier f_s is determined. If it is valid (i.e., it is

Algorithm 2 Training Graph Generation

Require: ECOC Matrix $\vec{M} = (m_{i,j}) \in \{-1, 0, 1\}^{k \times n}$, binary classifiers $f_1, \ldots, f_n$

$V \leftarrow \{n_r\}$
$E \leftarrow \emptyset$

for each f_i **do**
    $V \leftarrow V \cup \{n_i\}$ # Integration of all specified classifiers
    $e_{ri} \leftarrow (n_r, n_i)$
    $w(e_{ri}) \leftarrow I(f_i)$
    $E \leftarrow E \cup \{e_{ri}\}$

for $l \leftarrow k$ **downto** 2 **do** # level-wise subclassifier generation
    $F \leftarrow \{n \in V \setminus \{n_r\} \mid \mathsf{length}(n) \geq l\,,\, \mathsf{seen}(n) = 0\}$

    for each pair $(n_i, n_j) \in F \times F$ with $i \neq j$ **do**
        $n_s \leftarrow \mathsf{intersection}(n_i, n_j)$ # generate shared subclassifier of f_i and f_j
        if n_s is valid **then**
            if $n_s \notin V$ **then**
                $V \leftarrow V \cup \{n_s\}$ # classifier is new
                $e_{rs} \leftarrow (n_r, n_s)$
                $w(e_{rs}) \leftarrow I(f_s)$
                $E \leftarrow E \cup \{e_{rs}\}$
            $e_{si} \leftarrow (n_s, n_i)$, $e_{sj} \leftarrow (n_s, n_j)$
            $w(e_{si}) \leftarrow I(f_s \bigtriangledown f_i)$
            $w(e_{sj}) \leftarrow I(f_s \bigtriangledown f_j)$
            $E \leftarrow E \cup \{e_{si}, e_{sj}\}$
    $\forall n \in F.\mathsf{seen}(n) = 1$ # mark as processed, see also note in text

return $G = (V, E, w)$

non-zero and contains at least one positive and one negative class), two cases are possible:

- a corresponding node to f_s already exists in the tree: f_i and f_j are included to the set of childs of f_s, that means, two directed edges e_{si} and e_{js} with weights $I(f_i \bigtriangledown f_s)$, $I(f_j \bigtriangledown f_s)$ respectively are created, where $I(.)$ denotes the total number of training instances for a given code configuration.
- There exists no corresponding node to the subclassifier f_s: f_s is integrated into the tree by creating a corresponding node and by linking it to the root node with edge e_{rs} of weight $I(f_s)$. In addition, the same steps as in the first case are applied.

Algorithm 3 Greedy Steiner Tree Computation

Require: Training Graph $G = (V, E, w)$, binary classifiers $f_1, \ldots, f_n$

let Q be an empty FIFO-queue
$\hat{V} \leftarrow \emptyset$, $\hat{E} \leftarrow \emptyset$

for each f_i **do**
    Q.push(n_i) , $\hat{V} \leftarrow \hat{V} \cup \{n_i\}$

while !Q.isEmpty() **do**
    $n_i \leftarrow Q$.pop()
    $(n_x, n_i) \leftarrow \text{argmin}_{(n_a, n_i) \in E}\, w((n_a, n_i))$
    $\hat{E} \leftarrow \hat{E} \cup (n_x, n_i)$, $\hat{V} \leftarrow \hat{V} \cup \{n_x\}$

    if $n_x \neq n_r$ **then**
        Q.push(n_x)

return $\hat{G} = (\hat{V}, \hat{E}, w)$

4.2.2 Greedy Computing of Steiner Trees

A Steiner tree is, essentially, a minimum spanning tree of a graph, but it may contain additional nodes (which, in our case, correspond to unspecified classifiers). Minimizing the costs is equivalent to minimizing the total number of training examples that are needed to train all classifiers at the leaf of the tree from its root. As mentioned previously, we tackle this problem in a greedy way.

Let f_i be a specified classifier and E_i the set of incident incoming edges. We compute the minimum-weight edge and remove all other incoming edges. The outgoing node of this minimum edge is stored to repeat the process on this node afterwards, e.g., by adding it into a FIFO-queue. This is done until all classifiers and connected subclassifiers have been processed. Note that some subclassifiers are never processed, since all outgoing edges may have been removed. A pseudocode of this simple greedy approach is depicted in Algorithm 3. In the following, we will refer to it as the *min-redundant training scheme* and to the calculated approximate Steiner tree as $\hat{G}$.

This greedy approach can be combined with the generation method of the training graph, such that the resulting Steiner Graph is identical and such that the overall complexity is reduced to polynomial time. Recall the first step of the generation method: all pairs of classifiers of length k are checked for common subclassifiers and eventually integrated into G. After generating $O(n^2)$ subclassifiers, for each classifier f_i (of length k) the minimal incoming edge[2] is marked. All unmarked edges and

2 The weights of the edges are identical to the corresponding ones in the fully generated training graph, since it only depends on the total number of training instances, computable by the code configuration of the subclassifier, and not on the actual partitioning.

also the corresponding outgoing nodes, if they have no other child, are removed. In the next step of the iteration, $l = k - 1$, the number of nodes with length $l - 1$ are now at most n, since only maximally n new subclassifiers were included into the graph G. This means, for each level, $O(n^2)$ subclassifiers are generated, where the generation/checking of a subclassifier has cost of $\Theta(k)$, since we have to check k bits. So, in total, each level costs $O(n^2 \cdot k)$ operations. And, since we have k levels, the total complexity is $O(n^2 \cdot k^2)$. The implementation of the combined greedy method is straight-forward, so we omit a pseudocode and we will refer to it as GSteiner.

4.2.3 Incremental Learning with Training Graph

Given a Steiner tree of the training graph, learning with an incremental base learner is straight-forward. The specific training scheme is traversed in preorder depth-first-manner, i.e. at each node, the node is first evaluated and then its subtrees are traversed in left-to-right order. Starting from the root node, the first classifier f_1 is learned in batch mode. In the next step, if f_1 has a child, i.e. f_1 is subclassifier of another classifier f_2, f_1 is copied and incrementally learned with instances of $f_2 \bigtriangledown f_1$, yielding classifier f_2 and so on.

After the learning process, all temporary learned classifiers, which served as subclassifiers and are not specified in the ECOC matrix, are removed, and the prediction phase of the ECOC ensemble remains the same.

In this work, we use Hoeffding Trees (cf. Section 2.3.2 on page 11) as an example for an incremental learner and used the implementation in the Massive Online Analysis Framework (Bifet et al., 2010).

4.2.4 SVM Learning with Training Graph

While incremental learners are obvious candidates for our approach to save training time, the problem actually does not demand full incrementality because we always add batches of examples corresponding to different classes to the training set. Thus, the incremental design of a training algorithm might retard the training compared to an algorithm that can naturally incorporate larger groups of additional instances. Therefore, we decided to study the applicability of this approach to a genuine batch learner, and selected the Java-implementation of LibSVM (Chang and Lin, 2011). The adaption of this base learner consists of two parts: First, the previous model (subclassifier) is used as a starting point for the successor model in the training graph, and second, the caching strategy is adapted to this scenario.

Reuse of Weights

A binary SVM model consists of a weight vector $\vec{w}$ containing the weights w_i for each training instance $(\vec{x}_i, y_i)$ and a real-valued threshold b. The latter is derived from $\vec{w}$ and the instances without significant costs. The weights w are obtained as the solution of a quadratic optimization problem with a quadratic form $\vec{w}^T(\vec{y}^T H \vec{y})\vec{w}$ that incorporates the inputs through pairwise evaluations $H_{ij} = \kappa(\vec{x}_i, \vec{x}_j)$ of the kernel

function κ. The first component to speed up the training is to use the weights w of the parent model as start values for optimizing the child weights $\bar{w}$. That is, we set $\bar{w}_i = w_i$, if instance i belongs to the parent model and $\bar{w}_i = 0$ otherwise.

The mutual influence of different instances on their respective weights is twofold. There is a local mutual influence due to the fact that an instance can stand in the shadow of another instance closer to the decision boundary. And there is a weaker, global mutual influence that also takes effect on more unrelated instances communicating through the error versus regularization trade-off in the objective.

If we add additional instances to the training set we might expect that there is only a modest alternation of the old weights, because many of the new instances will have little direct effect on the local influence among previous instances. On the other hand, if the new instances do interfere with some subsets of the previous instances, the global influence can strongly increase as well. In any case, we are more interested in the question whether the parent initialization of the weights does speed up the optimization step.

Cache Strategy

It is well-known that caching of kernel evaluations provides significant speed-up for the learning with SVMs (Joachims, 1999). LibSVM uses a *least-recently-used* (LRU) Cache, which stores columns of the matrix H respective its signed variant $Q = \vec{y}^T H \vec{y}$. Since we use an ensemble of classifiers which potentially overlap in terms of their training instances and therefore also in their matrices H, it is beneficial to replace their local caches, which only keep information for each individual classifier, with an *ensemble cache*, which allows to transfer information from one classifier to the next one.

Typically, each classifier receives a different subset of training instances $T_l \subset T$, specified by its code configuration. In order to transfer common kernel evaluations H_{ab} from classifier f_i to another classifier f_j, the cached columns have to be transformed, since they can contain evaluations of irrelevant instances. Each H_{ab} has to be removed, if instances a or b are not contained in the new training set and also the possible change in the ordering of instances has to be considered in the columns. The main difficulty is the implementation of an efficient mapping of locally used instance ids to the entire training set and its related transformation steps, otherwise, the expected speed-up of an ensemble caching strategy is undone.

Two ensemble cache strategies were evaluated, which are based on the local cache implementation of LibSVM. The first one reuses *nearly all* reusable cached kernel evaluations from one classifier to another. For each classifier, two mapping tables $m_a(.)$ and $m_o(.)$ are maintained, where m_a associates each local instance number with its corresponding global instance number in order to have a unique addressing used in the transformation step. The table m_o is the mapping table from the previous learning phase. Before using the old cache for the learning of a new classifier, all cached entries are marked (as to be converted). During querying of the cache two cases can occur:

- **a cached column is queried:** If the entry is marked, the conversion procedure is applied. Using the previous and actual mapping table m_o, m_a, the column is transformed to contain only kernel evaluations for relevant instances, which can be done in $O(|m_o| \cdot \log |m_o|)$. Missing kernel evaluations are marked with a special symbol, which are computed afterwards. In addition, the mark is removed.
- **an uncached column is queried:** If the free size of the cache is sufficient, the column is computed and normally stored. Otherwise, beforehand, the least recently used entry is repeatedly removed until the cache has sufficient free space.

Since the columns are converted only on demand, unnecessary conversions are avoided and their corresponding entries are naturally replaced by new incoming kernel evaluations due to the LRU strategy. But, this tradeoff has the disadvantage that kernel evaluations that have been computed and cached at some point earlier may have to be computed again if they are requested later. The marked entries are carried maximally only over two iterations, otherwise it would be necessary for each additional iteration to carry another mapping table. We denote this ensemble caching method as *Short-Term Memory* (STM). One beneficial feature is the compatibility to any training scheme, in particular to the standard and the min-redundant training scheme.

The second ensemble caching method is particularly tailored to the use with a min-redundant training scheme. It differs from the previous one only in its transformation step. Recall that the learning phase traverses the subgraph in preorder depth-first manner. That means that during the learning procedure only the following two cases can occur: either the current classifier f_i is the child of a subclassifier f_j, or the current classifier is directly connected with the root node.

This information can be used for a more efficient caching scheme. For the first case, the set of training instances of f_i is superset of f_j, i.e. $T_j \subseteq T_i$. That means, $|T_j|$ rows and columns can be reused and also importantly without any costly transforming method. The columns and rows have to be simply trimmed to size $|T_j|$ for the reuse in the current classifier. Trimming is sometimes necessary, since they can contain further kernel evaluations from previously learned sibling nodes, i.e. nodes which share the same subclassifier f_j. So, the cache for the current classifier is prepared by removing Q_{ab} with $a > |T_j| \vee b > |T_j|$. In the second case, we know beforehand that no single kernel evaluation can be reused in the actual classifier. So, the cache is simply cleared. We denote this ensemble cache method as *Semi-Local* (SL) cache.

4.3 Experimental Evaluation

4.3.1 Experimental Setup

As we are primarily concerned with computational costs and not with predictive accuracy, we applied pre-processing based on all available instances instead of building

a pre-processing model on the training data only. First, missing values were replaced by the average or majority value for numeric or ordinal attributes respectively. Second, all numeric values were normalized, such that the values lie in the unit interval.

Our experiment consisted of following parameters and parameter ranges:

- **6+2 multiclass classification datasets**, where 6 relatively small datasets in terms of instances (up to ca. 4000) were used in conjunction with LibSVM and two large-scale datasets, *pokerhand* and *covtype* consisting of 581,012 and 1,025,010 instances, were used with Hoeffding Trees. The number of classes lie in the range between 4 and 11. All datasets are available from the UCI repository (Asuncion and Newman, 2010).
- **3 code types**: (k,l)-exhaustive and -cumulative exhaustive codes, random codes of up to length 500 with $l = 3, 4$ and $r_{zp} = 0.2, 0.4$
- **2 learn methods**: min-redundant and standard training scheme
- **2 base learners**: incremental learner Hoeffding Trees and batch learner LibSVM (no parameter tuning, RBF-kernel) for which following parameters were evaluated:
  - **3 cache methods**: two ensemble cache methods, namely STM and SL, and the standard local cache of LibSVM
  - **4 cache sizes**: 25 %, 50 %, 75 %, 100 % of the number of total kernel evaluations

All experiments with LibSVM were conducted with 5-fold cross-validation and for Hoeffding Trees a training-test split of 66 % to 33 % was used. The parameters of the base learners were not tuned, because we were primarily interested in their computational complexity.[3]

4.3.2 Hoeffding Trees

Table 4.1 on the next page shows a comparison between the standard training scheme and the greedy computed min-redundant scheme with respect to the total amount of training instances. It shows that even with the suboptimal greedy procedure a significant amount of training instances can be saved. In this evaluation, the worst case can be observed for dataset *covtype* with 3-level exhaustive codes, for which the ratio to the standard training scheme is 22 %. In absolute numbers, this relates to processing 3.8 million training instances instead of 17.2 million. In summary, the improvements range from 78 % to 98 % or in other words, 4 to 45 times less training instances are processed.

3 Tuning of the SVM parameters of the base learners can be relevant here because it may affect the effectiveness of reusing and caching of models. However, this would add additional complexity to the analysis of total cost and was therefore omitted to keep the analysis simple.

Table 4.1: Total number of processed training instances of standard and min-redundant training scheme. The italic values show the ratio of both. The datasets *pokerhand* and *covtype* consist of 581,012 and 1,025,010 instances respectively, from which 66 % was used as training instances.

dataset	standard	min-redundant	standard	min-redundant
	CUMULATIVE EXHAUSTIVE CODES			
	$l = 3$		$l = 4$	
pokerhand	79,151,319	9,429,611 (*0.119*)	476,937,435	10,479,451 (*0.022*)
covtype	19,556,868	3,807,748 (*0.195*)	73,242,388	5,354,720 (*0.073*)
	EXHAUSTIVE CODES			
	$l = 3$		$l = 4$	
pokerhand	73,062,756	9,429,591 (*0.129*)	397,786,116	10,478,523 (*0.026*)
covtype	17,256,060	3,796,818 (*0.220*)	53,685,520	5,191,055 (*0.097*)
	RANDOM CODES			
	$r_{zp} = 0.4$		$r_{zp} = 0.2$	
pokerhand	258,035,711	10,205,330 (*0.040*)	311,051,271	8,990,547 (*0.029*)
covtype	153,519,616	6,744,692 (*0.044*)	95,483,532	5,300,005 (*0.056*)

Table 4.2 on the following page shows the corresponding total training time. It shows that the previous savings with respect to the number of training instances do not transfer directly to the training time. One reason is that the constant factor in the linear complexity of Hoeffding Trees regarding the number of training instances decreases for increasing number of training instances. Furthermore, some overhead is incurred for copying the subclassifiers before each incremental learning step. In total, exploiting the redundancies yields a run-time reduction of about $44.6\,\% - 85.8\,\%$.

The running-time for GSteiner (constructing the graph and greedily finding the Steiner tree, without evaluation of the classifiers) is depicted in Table 4.3 on the next page. For the *systematic* code types, exhaustive and its cumulative version, the used time is in general negligible compared to the total training time. The only exception is for dataset *pokerhand* with random codes and $r_{zp} = 0.2$: About 106 seconds were used and contributes therefore one-fifth to the total training time in this case.

4.3.3 LibSVM

Table 4.4 on page 39 shows a comparison of training times between LibSVM and its adaptions with weight reusing and ensemble caching strategies. M1 and M2 use the standard training scheme, where M1 is standard LibSVM with local cache and M2 uses the ensemble caching strategy STM. M3 and M4 utilize a min-redundant training scheme with STM and SL respectively. The underlined values depict the best value for each dataset and code-type combination. The results confirm that the weight reuse and ensemble caching techniques can be used to exploit code redundancies for

Table 4.2: Training time in seconds. This table shows training performances for the standard and the min-redundant learning scheme. The italic values shows the ratio of both.

dataset	standard	min-redundant	standard	min-redundant
	CUMULATIVE EXHAUSTIVE CODES			
	$l = 3$		$l = 4$	
pokerhand	261.27	127.33 (*0.487*)	1530.06	542.57 (*0.355*)
covtype	118.70	40.89 (*0.344*)	463.09	93.71 (*0.202*)
	EXHAUSTIVE CODES			
	$l = 3$		$l = 4$	
pokerhand	236.52	131.12 (*0.554*)	1337.00	522.18 (*0.391*)
covtype	101.50	34.65 (*0.341*)	330.97	83.58 (*0.253*)
	RANDOM CODES			
	$r_{zp} = 0.4$		$r_{zp} = 0.2$	
pokerhand	896.41	356.43 (*0.398*)	1089.99	537.11 (*0.493*)
covtype	1106.48	157.61 (*0.142*)	695.84	107.12 (*0.154*)

Table 4.3: GSteiner running time in seconds

	CUMULATIVE EXH.		EXHAUSTIVE		RANDOM	
	$l = 3$	$l = 4$	$l = 3$	$l = 4$	$r_{zp} = 0.4$	$r_{zp} = 0.2$
pokerhand	0.82	4.63	4.56	3.57	22.09	105.97
covtype	0.24	3.01	0.14	0.17	0.67	0.52

LibSVM. For exhaustive codes and its cumulative variant, M4 dominates all other approaches and achieves an improvement of 31.4 % – 78.4 % of the training time. However, the results for random codes are not so clear.

For the datasets *vowel* and *yeast* both methods employing the min-redundant training schemes (M3 and M4) use significantly more time. This can be explained with the relative expensive cost for generating and solving the Steiner tree in these cases, as depicted in Table 4.5 on the next page (89 and 52 sec for *vowel* and *yeast*). Contrary to the results on *optdigits*, for these datasets the tree generation and solving has a big impact on the total training time. Nevertheless, this factor is decreasing for increasing number of instances, since the complexity of GSteiner only depends on k and n. Besides, based on the results with various cache sizes (cf. Tables A.2, A.3 and A.4 in the Appendix) the cache size has a greater impact on the training time for random codes than for the systematic ones. Table 4.6 on page 40 shows as an example the performance for random codes with a cache size of 75 %. Notice the reduction of the training time for the different methods in comparison to Table 4.4 on the next page, where a cache size of 25 % was used. M4 achieves the best efficiency increase and by subtracting the time for generating and solving the tree, M4 dominates again all other methods.

Table 4.4: Training time in seconds (cache size: 25 %)

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
			CUMULATIVE EXHAUSTIVE CODES			
			$l = 3$			
M1	92.28 ± 0.36	8.73 ± 0.19	6.56 ± 0.05	3.47 ± 0.07	5.80 ± 0.02	5.43 ± 0.03
M2	80.70 ± 0.37	8.32 ± 0.37	6.00 ± 0.03	4.30 ± 0.08	4.90 ± 0.02	5.62 ± 0.02
M3	76.93 ± 0.60	6.90 ± 0.18	6.94 ± 0.05	3.13 ± 0.16	6.28 ± 0.04	5.77 ± 0.03
M4	53.37 ± 0.40	2.93 ± 0.27	4.19 ± 0.05	1.70 ± 0.25	3.51 ± 0.01	2.98 ± 0.02
			$l = 4$			
M1	833.12 ± 14.98	24.66 ± 0.43	33.98 ± 0.21	18.61 ± 0.35	47.61 ± 0.08	40.42 ± 0.09
M2	666.02 ± 1.54	21.19 ± 0.80	28.69 ± 0.14	22.94 ± 0.52	36.72 ± 0.08	41.19 ± 0.11
M3	680.75 ± 8.23	18.30 ± 0.51	36.91 ± 0.39	15.08 ± 1.71	51.61 ± 0.15	41.79 ± 0.10
M4	410.44 ± 6.08	5.32 ± 0.53	17.18 ± 0.13	8.59 ± 1.27	25.26 ± 0.06	22.01 ± 0.10
			EXHAUSTIVE CODES			
			$l = 3$			
M1	87.42 ± 0.35	7.63 ± 0.39	6.02 ± 0.03	3.17 ± 0.05	5.51 ± 0.03	5.11 ± 0.02
M2	75.28 ± 0.29	6.76 ± 0.12	5.48 ± 0.03	3.95 ± 0.07	4.58 ± 0.03	5.28 ± 0.01
M3	75.61 ± 1.04	7.09 ± 0.27	6.91 ± 0.04	3.13 ± 0.14	6.25 ± 0.03	5.83 ± 0.05
M4	53.13 ± 0.39	2.90 ± 0.21	4.13 ± 0.03	1.71 ± 0.25	3.48 ± 0.02	3.00 ± 0.02
			$l = 4$			
M1	735.76 ± 9.63	15.31 ± 0.49	27.13 ± 0.31	15.14 ± 0.28	41.78 ± 0.09	34.99 ± 0.08
M2	570.69 ± 1.93	12.72 ± 0.45	22.76 ± 0.13	18.72 ± 0.42	31.92 ± 0.06	35.73 ± 0.06
M3	646.6 ± 11.98	16.39 ± 0.44	34.24 ± 0.36	14.69 ± 1.59	49.75 ± 0.10	41.09 ± 0.10
M4	397.79 ± 5.07	4.76 ± 0.46	15.88 ± 0.09	8.45 ± 1.17	24.55 ± 0.10	21.71 ± 0.06
			RANDOM CODES			
			$r_{zp} = 0.4$			
M1	1654.0 ± 22.6	25.7 ± 1.1	156.5 ± 1.7	34.7 ± 1.5	37.5 ± 0.6	46.9 ± 1.2
M2	1424.4 ± 32.8	24.3 ± 0.5	162.9 ± 0.8	46.1 ± 1.9	39.7 ± 0.7	52.1 ± 1.3
M3	1609.2 ± 44.3	22.6 ± 0.3	190.6 ± 3.8	39.9 ± 5.4	65.8 ± 2.3	79.1 ± 2.0
M4	1378.8 ± 34.4	5.7 ± 0.3	140.6 ± 3.0	25.9 ± 3.7	57.1 ± 2.5	64.5 ± 2.4
			$r_{zp} = 0.2$			
M1	2634.6 ± 59.5	10.2 ± 0.3	123.0 ± 0.9	48.2 ± 2.0	49.6 ± 0.4	67.2 ± 1.2
M2	2281.7 ± 29.6	8.6 ± 0.5	129.7 ± 1.4	63.2 ± 3.1	53.0 ± 0.4	74.1 ± 1.3
M3	3049.0 ± 48.3	12.7 ± 0.2	157.9 ± 1.4	57.6 ± 13.3	153.0 ± 2.0	157.5 ± 2.1
M4	2594.0 ± 64.8	3.6 ± 0.2	128.5 ± 2.4	39.1 ± 9.4	144.6 ± 1.7	144.0 ± 2.2

Table 4.5: GSteiner running time for random codes in seconds

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
$r_{zp} = 0.4$	8.93	< 0.01	0.12	0.50	15.10	8.56
$r_{zp} = 0.2$	53.60	< 0.01	0.12	1.26	89.34	52.80

Table 4.6: Training time in seconds of random codes (cache size: 75 %)

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
			RANDOM CODES, CACHE=75 %			
			$r_{zp} = 0.4$			
M1	1603.4 ± 22.2	25.8 ± 0.5	153.7 ± 1.5	34.3 ± 1.5	36.0 ± 0.6	45.6 ± 1.1
M2	1317.4 ± 16.3	23.0 ± 0.4	136.9 ± 1.0	45.1 ± 1.9	35.9 ± 0.6	51.2 ± 1.3
M3	1364.6 ± 53.6	22.4 ± 0.2	148.7 ± 1.3	35.8 ± 4.5	60.9 ± 2.0	82.6 ± 2.2
M4	1162.6 ± 27.6	5.5 ± 0.3	70.3 ± 0.4	7.9 ± 0.8	42.8 ± 2.0	27.4 ± 1.2
			$r_{zp} = 0.2$			
M1	2507.0 ± 33.8	10.3 ± 0.3	119.8 ± 1.2	47.6 ± 2.0	47.6 ± 0.4	65.3 ± 1.2
M2	1826.2 ± 21.3	8.5 ± 0.6	98.7 ± 0.6	61.3 ± 3.1	44.3 ± 0.4	70.6 ± 1.2
M3	2093.7 ± 38.6	12.4 ± 0.2	116.9 ± 0.8	51.0 ± 11.0	139.9 ± 1.8	163.7 ± 2.6
M4	1632.5 ± 40.2	3.9 ± 0.1	56.8 ± 0.3	10.0 ± 1.6	118.6 ± 1.6	87.7 ± 2.2

Table 4.7: Comparison of LibSVM optimization iterations. The values show the ratio of optimization iterations of a min-redundant training scheme with weight reusing to standard learning.

CUMULATIVE EXH.		EXHAUSTIVE		RANDOM	
$l = 3$	$l = 4$	$l = 3$	$l = 4$	$r_{zp} = 0.4$	$r_{zp} = 0.2$
0.673	0.576	0.768	0.745	0.701	0.773

Table 4.7 shows the number of optimization iterations of LibSVM, which can be seen as an indicator of training complexity. The ratio values are averaged over all datasets and show that the reuse of weights in the pseudo-incremental learning steps lead to a reduction of optimization iterations.

Once again, the effect on the ensemble caching strategy can be seen in Table 4.8 on the facing page, showing a selection of the results, here for cache sizes 25 % and 75 %. The first column of each block describes the number of kernel evaluation calls. The consistent reduction for min-redundant schemes M3 and M4 is accredited to the weight-reusing strategy. Except for random codes with $r_{zp} = 0.2$ and cache size=25 % all methods using an ensemble cache strategy (M2, M3 and M4) outperform the baseline of LibSVM with a local cache. Among these three methods, M3 and M4 both outperform M2 in absolute terms, but not relative to the number of calls. For the special case (random codes, $r_{zp} = 0.2$, M3, M4), one can again see the increased gain of a bigger cache size for the min-redundant training schemes.

Even though all ensemble caching strategies almost always outperform the baseline in terms of hit-miss measures, the corresponding time complexities of Table 4.4 on the previous page show that only M4, which uses a min-redundant training scheme and the SL caching strategy, is reliably reducing the total training time. The rather costly transformation cost of STM is the cause for the poor performance of M2 and M3.

Table 4.8: Cache efficiency and min-redundant training scheme impact: averaged mean ratio values of kernel evaluation calls (first column) and actual computed kernel evaluations (second column) to the baseline: standard LibSVM (M1). The values of M1 are set to 1 and the following values describe the ratio of corresponding values of M2, M3 and M4 to M1.

	CUMULATIVE EXH.				EXHAUSTIVE				RANDOM			
	$l = 3$		$l = 4$		$l = 3$		$l = 4$		$r_{zp} = 0.4$		$r_{zp} = 0.2$	
	CACHE = 25 %											
M2	1.00	0.68	1.00	0.61	1.00	0.66	1.00	0.60	1.00	0.83	1.00	0.84
M3	0.78	0.56	0.71	0.52	0.87	0.63	0.88	0.67	0.84	0.83	0.95	1.01
M4	0.78	0.56	0.71	0.51	0.87	0.63	0.88	0.65	0.84	0.83	0.95	1.00
	CACHE = 75 %											
M2	1.00	0.59	1.00	0.48	1.00	0.56	1.00	0.44	1.00	0.64	1.00	0.56
M3	0.78	0.43	0.71	0.34	0.87	0.47	0.88	0.42	0.84	0.41	0.95	0.47
M4	0.78	0.44	0.71	0.32	0.87	0.48	0.88	0.41	0.84	0.42	0.95	0.49

4.4 Related Work

In (Blockeel and Struyf, 2003), an efficient algorithm for cross-validation with decision trees is proposed, which also exploits training set overlaps, but focuses on a different effect, namely that in this case the generated models tend to be similar, such that often identical test nodes are generated in the decision tree during the learning process. This approach is not applicable here, since during the incremental learning steps, the inclusion of new classes may lead to significant model changes. Here, a genuine incremental learner or in the case of LibSVM different approaches are necessary. However, the main idea, to reduce redundant computations is followed also here.

Pimenta et al. (2007) consider the task of optimizing the size of the coding matrix so that it balances effectivity and efficiency. Our approach is meant to optimize efficiency for a given coding matrix. Thus, it can also be combined with their approach if the resulting *balanced* coding matrix is code-redundant.

4.5 Conclusions

We studied in this chapter the possibility of reducing the training complexity of ECOC ensembles with highly redundant codes such as cumulative exhaustive, exhaustive and random codes. We proposed an algorithm for generating a so-called training graph, in which edges are labeled with training cost and nodes represent (sub-)classifiers. By finding an approximate Steiner tree of this graph in a greedy manner, the training complexity can be reduced without changing the prediction quality. An initial evaluation with Hoeffding Trees, as an example for an incremental learner, yielded time savings in the range of 44.6 % to 85.8 %. Subsequently, we also demonstrated how SVMs can be adapted for this scenario by reusing weights and by employing an

ensemble caching strategy. With this approach, the time savings for LibSVM ranged from 31.4 % to 78.4 %. In general, we can expect higher gains for incremental base learners whose complexity grows more steeply with the number of training instances. The presented approach is useful for all considered high-redundant code types, and also for random codes, for which the impact of the GSteiner algorithm decreases with increasing training instances. In addition, the generation of a min-redundant training scheme could be seen as a pre-processing step, such that it is not counted or only counted once for the total training time of an ECOC ensemble, because it is reusable and independent of the base learner.

However, this approach has its limitations. GSteiner can be a bottleneck for problems with a high class count, since its complexity is $O(n^2 \cdot k^2)$ and the length n for common code types such as exhaustive codes grow exponentially in the number of classes k. And, this work considers only highly redundant code types, which are not unproblematic. First, usually in conjunction with ECOC ensembles, one prefers *diverse* classifiers, which are contrasting the redundant codes in our sense. The more shared code configurations exist in an ensemble, the less independent are its classifiers. Secondly, these codes are not as commonly used as the low-redundant decompositions schemes one-against-all and one-against-one.

Another point is, that we implicitly assumed that the incremental learners are independent with respect to the order of examples. This is usually not the case, also for the used Hoeffding Tree algorithm. Though we observed only negligible deviations regarding the predictive performance in our experiments, an appropriate base learner should satisfy this property.

5 Efficient ECOC Training with Naïve Bayes

Contents

While the original motivation for the development of the ECOC framework was to make multiclass classification problems amenable to binary base classifiers, it is also known that the resulting multiclass classifier may benefit from the error-correcting properties of this framework. In particular, several authors have shown that the Naïve Bayes algorithm can benefit from the ECOC framework, especially for text-classification (Berger, 1999; Ghani, 2000).

In this chapter, we will show that a simple tight combination of these two methods allows a significantly more efficient learning procedure for a ternary ECOC-based classifier with Naïve Bayes, without any change in predictive performance in comparison to the straight-forward approach. The key idea is the realization that binary decompositions of a Naïve Bayes classifier can be computed very efficiently from the estimated conditional probabilities of the original Naïve Bayes procedure.

Though Naïve Bayes as a natural incremental learner is compatible to the approach from the previous chapter, we will show that in this case an exploitation of redundant operations is applicable on a substantially deeper and more effective level, namely in some sense on the feature level instead of the level of shared subclassifiers.

First, we briefly recapitulate Naïve Bayes in Section 5.1 and derive the efficient computation of ECOC ensembles with Naïve Bayes base classifiers in Section 5.2. Then, in Section 5.2.3, we present a suitable *precalculation* method for discrete, normal and kernel density estimation methods. Finally, we provide empirical support for the method in Section 5.3, and end with the conclusion in Section 5.4.

5.1 Naïve Bayes

Though Naïve Bayes (NB) is capable of directly learning multiclass predictors, results in the literature indicate that its predictive performance can be increased in combination with ECOC methods. Especially for text classification it seems to be promising (Berger, 1999; Ghani, 2000).

Naïve Bayes is essentially an application of the Bayes Theorem with the so-called naïve-independency assumption. In the following, we recapitulate the derivation, which, although commonly known, is helpful for the presentation of the alternative computation scheme. Let $a_1, \ldots, a_g$ denote features or attributes and c be the class-variable which has k values. Using Bayes Theorem, we can compute the class probability as

$$P(c \mid a_1, \ldots, a_g) = \frac{P(c) \cdot P(a_1, \ldots, a_g \mid c)}{P(a_1, \ldots, a_g)}$$

Since the denominator of the right hand side is constant for a given test instance $x = (a_1, \ldots, a_g)$, we can ignore this term, and focus on the numerator. More precisely, for the case of classification, the following holds:

$$\operatorname*{argmax}_c P(c \mid a_1, \ldots, a_g) = \operatorname*{argmax}_c P(c) \cdot P(a_1, \ldots, a_g \mid c)$$

Using the *class-conditional independence assumption* $P(a_i \mid c, a_j) = P(a_i \mid c)$ for two arbitrary attributes a_i, a_j and $i \neq j$, we can estimate the class-conditional probability $P(a_1, \ldots, a_n \mid c)$ with

$$P(a_1, \ldots, a_n \mid c) = \prod_{i=1 \ldots g} P(a_i \mid c)$$

$P(a_i \mid c)$ and $P(c)$ are estimated from the training data.

5.2 Computation of ECOC for Naïve Bayes in a single pass

In this section, we describe the key idea of this approach. We first show that all probability estimates that are conditioned on a mutually exclusive group of classes are additive (Section 5.2.1), and that this can be used for faster probability estimation in ECOC codes (Section 5.2.2). Finally, we discuss how the idea can be implemented for nominal and numeric data (Section 5.2.3).

5.2.1 Reduction to Base Probabilities

The key idea behind the efficient computation of arbitrary class-based decomposition schemes such as ECOC is that all constituent classifiers of a class-based decomposition can be reduced to the estimation of the parameters of the Naïve Bayes classifier, $P(a_i \mid c)$ and $P(c)$.

Let $c_D^+ = \{c_1, \ldots, c_l\}$ be the set of classes defined as positive given by a decomposition D, e.g., from a column of a ECOC matrix. Then it holds that

$$P(c_D^+) = \sum_{c \in c_D^+} P(c) \tag{5.1}$$

and

$$\begin{aligned} P(a_i \mid c_D^+) &= P(a_i \mid c_1 \vee \cdots \vee c_l) \\ &= \frac{P(a_i \wedge (c_1 \vee \cdots \vee c_l))}{P(c_1 \vee \cdots \vee c_l)} \\ &= \frac{P(a_i \wedge c_1) + \cdots + P(a_i \wedge c_l)}{\sum_{j=1}^{l} P(c_j)} \\ &= \frac{\sum_{j=1}^{l} P(a_i \wedge c_j)}{\sum_{j=1}^{l} P(c_j)} \end{aligned} \tag{5.2}$$

since the events of $P(c)$ are mutually exclusive. The probabilities $P(c_D^-)$ and $P(a_i \mid c_D^-)$ for the negative class can be reduced analogously.

Equations 5.1 and 5.2 simply show, that all necessary values $P(c_D^+)$ and $P(a_i \mid c_D^+)$ can be computed using $P(c)$ and $P(a_i \mid c)$, as computed by the standard Naïve Bayes. Therefore, different decompositions within the ECOC-Framework can be applied with Naïve Bayes without employing further probability estimation steps from training data, since they would involve redundant computations.

This tight combination of Naïve Bayes and ECOC, i.e. applying standard Naïve Bayes learning and computing the appropriate probabilities using Equations 5.1 and 5.2, will be called ECOC-NB in the following text, for convenience. Note, there exist code types, for which the binary decomposition in conjunction with voting aggregation (which is in principle identical to Hamming Decoding, addressed in the next chapter in Section 6.1.1) is equivalent to standard Naïve Bayes. For example, in (Sulzmann et al., 2007) it was shown that a one-against-one decomposition of Naïve Bayes is equivalent to Naïve Bayes.

5.2.2 Complexity

Instead of n iterations over the dataset for estimating the corresponding estimations of $P(a_i \mid c_D^+)$ and $P(c_D^+)$ for an n-bit ECOC scheme, only one pass is necessary. The usual training complexity of $O(n \cdot t \cdot g)$ can thus be reduced to $O(t \cdot g)$, where n is the number of classifiers, t the number of training instances and g the number of features. This is possible by applying standard Naïve Bayes to estimate $P(a_i \mid c)$ and $P(c)$ followed by utilizing Equations 5.1 and 5.2 at classification time for each decomposed classifier and test instance.

Note, however, that although the training complexity is significantly decreased in comparison to the straight-forward application of the ECOC framework, the cost is moved to the prediction phase, because we now have to perform more calculations for estimating the class-probabilities of an example. In particular, for a problem with a large number of attributes and classifiers, this can lead to a significant increase of testing complexity.

5.2.3 Precalculation

We will show in this section that the above-mentioned drawback can be solved by *precalculating* the probability distributions needed by the classifiers, i.e precalculating the combined probability distribution $P(a_i \mid c_D^+)$, instead of always aggregating over a series of part-probabilities according to Equation 5.2 for each test instance. This approach results in a *training* complexity of $O((n+t) \cdot g)$ and the same *testing* complexity as the standard approach.

First, we take a closer look into probability estimation methods for common attribute types which are used in conjunction with Naïve Bayes. Then, we show how to precalculate the needed probability distributions.

Discrete / Nominal Attribute

For discrete attributes, i.e. $a_i \in A_i$, where A_i is a finite set of distinct values, the following frequency based model is usually used:

$$P(a_i \mid c_j) = \frac{|a_i \wedge c_j|}{|c_j|}$$

where $|x|$ denotes here the number of observed instances which satisfy statement x. Also: $P(a_i \wedge c_j) = \frac{|a_i \wedge c_j|}{n}$ and $P(c_j) = \frac{|c_j|}{n}$, where n is the number of observed instances, so far. So for Equation 5.2,

$$P(a_i \mid c_D^+) = \frac{\sum_{j=1}^{l} |a_i \wedge c_j|}{\sum_{j=1}^{l} |c_j|}$$

This leads to $(|A_i|+1)k$ additions and $|A_i|$ divisions for generating the pseudo probability estimator. Note, this complexity is not dependent on the number of training instances.

Numeric Attribute

For numeric attributes, the following two estimation procedures are commonly applied:

- **Normal Density Estimation** The conditional probability $P(a_i \mid c_j)$ is in this case usually modeled as a normal distribution:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

whereas the mean value $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ and corresponding standard deviation in following particular calculation formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2 / n}{n}}$$

is updated for each incoming training instance by maintaining the number of observations n, the sum of observed attribute values $\sum_{i=1}^{n} x_i$ and the sum of squared values $\sum_{i=1}^{n} x_i^2$. These values can be analogously summed up for representing the pseudo probability distribution, which computational cost is also independent of the number of instances and dependent of the number of attributes.

- **Kernel Density Estimation** Here, the probability density model is in contrast to the previous two models not represented by a rather small number of model parameters. Simply said, kernel density estimators maintain all observed data values and, depending on their distance to a requested value, contribute to its probability estimate, which results in a somewhat smooth and not necessary unimodal probability density function. The definition is

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where $K(.)$ is some kernel (often a standard Gaussian function with mean zero and variance one) and h is a smoothing parameter, called the *bandwidth*.

In our context, the straight-forward method to combine these probability distributions is to merge the observations, which can be done in $O(t)$. In contrast to the previous estimation techniques, the overall worst-case is therefore equivalent to the straight-forward ECOC method.

But, there is still an advantage of this precalculation of kernel density estimators compared to the straight-forward ECOC. For this, we have to recapitulate a bit more the implementation of $f(x)$, as it is, e.g., realized in the WEKA software (Hall et al., 2009). Each *distinct* observed attribute value is stored in a sorted array along with its number of occurrences (or weight). The merging complexity of an arbitrary partition of z values in this representation (sorted arrays of distinct values with their weight) is $O(w)$, where w is the number of actual distinct observed values of z. If this w is very small compared to the number of instances t (which is an upper bound for the maximal number of distinct values), the precalculation method is superior to the straight-forward method. Carried on to the whole dataset, we denote the ratio w/z as *diversity*-value of a dataset, where w is summed over all attributes and $z = t \cdot g$.

Algorithm 4 ECOC-NB training scheme

Require: ECOC Matrix $(m_{ij}) \in \{-1, 0, 1\}^{k \times n}$, training set $T = (x, y)_r$

1: **for each** training instance (x, y_i) **do**
2: $\quad ePrior$.OBSERVE(i) # $P(c_i)$
3: $\quad$ **for each** attribute a_j of x **do**
4: $\quad\quad eCond_{i,j}$.OBSERVE(valueOf(a_j)) # $P(a_j \mid c_i)$
5:
6: **for each** classifier f_j **do**
7: $\quad P(c^+_{D_j}) \leftarrow \sum_{i=1, m_{ij}>0}^{k} P(c_i)$
8: $\quad P(c^-_{D_j}) \leftarrow \sum_{i=1, m_{ij}<0}^{k} P(c_i)$
9: $\quad$ **for each** attribute a_k **do**
10: $\quad\quad$ precalculate $P(a_k \mid c^+_{D_j})$ and $P(a_k \mid c^-_{D_j})$ according to attribute type and
11: $\quad\quad$ estimation technique

Algorithm 5 ECOC-NB testing scheme

Require: $m_{ij}, P(c^+_{D_j}), P(c^-_{D_j}), P(a_k \mid c^+_{D_j}), P(a_k \mid c^-_{D_j})$, instance $x = (a_1, \ldots, a_g)$

1: **for each** classifier/column f_j **do**
2: $\quad b_j \leftarrow$ MAKETERNARY$(P(c^+_{D_j}) \cdot \prod_k P(a_k \mid c^+_{D_j}))$ # compute bit-prediction
3: **return** $\text{argmax}_i \sum_{j=1}^{n} m_{ij} \cdot b_j$ # weighted decoding

5.2.4 ECOC-NB Algorithm

Algorithms 4 and 5 show in pseudocode the simple combined algorithm. The first four lines of Algorithm 4 correspond to standard Naïve Bayes training, whereas the remaining lines represent the precalculation scheme using Equations 5.1 and 5.2. The testing phase depicted in Algorithm 5 is in principle identical to the one of standard ECOC. The function MAKETERNARY maps the prediction of each classifier into the interval $[-1, 1]$ to be in line with the ternary ECOC framework using an appropriate mapping function, for instance $f(x) = (x - 0.5) \cdot 2$.

5.3 Experimental Evaluation

5.3.1 Experimental Setup

ECOC-NB was implemented within the WEKA framework (Hall et al., 2009). For the evaluation, we mainly focused on text-classification problems and used a freely available package of 19 text-classification datasets (19MclassTextWc[1]), from which we selected 17 datasets. The remaining two datasets were excluded because one yielded a very low accuracy ($< 1\,\%$) and the other had a relatively high time complexity, which would unnecessarily increase the overall time for the experiments without

1 http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html

gaining any increased insight (both with standard Naïve Bayes). This collection is composed of well-known benchmark datasets such as TREC and OHSUMED. For a detailed description, we refer to (Forman, 2003). Table 5.5 on page 54 shows the dataset characteristics and the last column shows the diversity of each dataset.

In our experiments, we follow prior work in Naïve Bayes text classification (Ghani, 2000), and use BCH codes (cf. Section 3.4.3 on page 25). Each of the k classes is initialized with a randomly selected vector which is then multiplied with the generator polynomial to yield the code word for this class, similar to (Dietterich and Bakiri, 1995). In our evaluation, we used the bchpoly routine of Gnu Octave[2] to generate binary BCH codes of lengths $15, 31, 63, 127, 255, 511$ and 1023 with maximal designed minimum Hamming distance respectively. In the usual notation, we used BCH codes $(15, 5, 7)$, $(31, 6, 15)$, $(63, 7, 31)$, $(127, 8, 63)$, $(255, 9, 127)$, $(511, 10, 255)$ and $(1023, 11, 511)$, where the parameters describe (in this order) the code word bit-length, the bit-length for coded information, and the minimal Hamming distance between any pair of code words.

For all experiments, 10-fold Cross-Validation was applied and they were conducted on a 2.4 GHz AMD Opteron 250 system with 8GB RAM. For kernel density estimation, a Gaussian kernel with mean zero and variance one was used. No feature selection was applied, since we are mainly interested on the training complexity, which is more interesting with a high number of features. But this comes with the disadvantage, that the accuracy performances may not represent the optimal values. So, in this regard, the following accuracy results should be viewed with reservation.

In the following performance tables, some cells are empty, because for these particular combinations of dataset and BCH bit-length, the BCH code generation process could not generate a valid ECOC matrix, which satisfies some machine learning relevant properties: The code generation process randomly picks k BCH code words of the specified length as the ECOC matrix and checks for every column, if there is at least one $(+1)$ and one (-1) symbol, respectively. Furthermore, no two columns must be identical. The code generation process is aborted after 100,000 iterations. It is clear, that the lower the number of classes, the lower the possibility to generate a suitable ECOC-matrix with high bit-length.

Note, that we used weighted decoding (Dietterich and Bakiri, 1995) instead of Hamming decoding, because it performed slightly better with respect to accuracy in our setting in some preliminary tests.

5.3.2 Accuracy Evaluation

Tables 5.1 and 5.2 show the accuracy performance for Naïve Bayes compared to ECOC-NB with various bit-lengths, using normal density estimation (Table 5.1 on the following page) and Kernel density estimation (Table 5.2 on page 51). In general, Naïve Bayes and ECOC-NB without kernel density estimators perform better on these datasets. Furthermore, as can be seen in Table 5.1, ECOC-NB yields superior

2 Octave is a free alternative to MatLab available from http://www.gnu.org/software/octave/.

Table 5.1: Accuracy of Naïve Bayes and ECOC-NB with different BCH code lengths. Both use normal density estimators. Bold values depict the best performance for the dataset in row and the underlined values among Tables 5.1 and 5.2.

data	NB	15	31	63	127	255	511	1023
fbis	62.61	54.77	58.95	61.35	64.15	64.60	64.80	**65.33**
la1	75.06	75.25	**76.03**	–	–	–	–	–
la2	74.89	73.79	**75.22**	–	–	–	–	–
oh0	79.66	79.06	80.65	80.95	**81.05**	80.95	80.75	–
oh5	77.88	76.68	78.97	**80.06**	79.73	**80.06**	79.62	–
oh10	72.67	71.90	72.86	74.10	73.90	73.90	**74.38**	–
oh15	75.24	76.56	76.88	78.09	**79.41**	78.53	78.75	–
re0	57.51	59.71	62.70	64.70	67.56	65.29	**69.55**	68.55
re1	66.33	68.92	71.94	72.30	**74.29**	73.87	73.63	74.17
tr11	54.83	48.34	56.54	55.57	**57.98**	57.74	–	–
tr12	54.67	55.95	57.25	59.10	**61.98**	–	–	–
tr21	**46.39**	37.50	39.87	–	–	–	–	–
tr23	**55.79**	34.19	37.19	–	–	–	–	–
tr31	80.69	82.95	83.61	**84.57**	–	–	–	–
tr41	85.65	85.32	86.91	87.14	87.37	**87.82**	87.59	–
tr45	65.51	58.26	62.32	66.38	66.09	**67.39**	**67.39**	–
wap	**72.76**	61.35	66.54	65.51	67.50	68.27	67.76	68.40

results than standard Naïve Bayes, except for the worse performance on datasets *tr21*, *tr23* and *wap*. Using kernel density estimators (Table 5.2 on the facing page), we can observe a different result. Both methods seem to be competitive, with some deviations in favor of both methods.

We can also view the choice of the density estimator as an additional parameter in the parameter tuning phase. So, if we focus on the best performance for each datasets across both tables (depicted by underlined values), only in 4 of 17 datasets, namely *la1*, *tr21*, *tr23* and *wap*, the traditional Naïve Bayes outperforms ECOC-NB.

5.3.3 Run-time Evaluation

The corresponding training times are shown in Tables 5.3 and 5.4. For bit-lengths 15, 31, 63 and 127 the second column in both tables show the training time of the straight-forward ECOC implementation, which should serve as a sanity check and for exposition purposes. The training time increase for the straight-forward ECOC method compared to Naive Bayes corresponds very closely to the number of used ECOC bits respectively classifiers. Furthermore, one can clearly observe the very mild increase of the training time for ECOC-NB for increasing bit-length.

Also, using kernel density estimators, we can observe only a relative slight increase for increasing number of classifiers (Table 5.4 on page 53). As previously mentioned, the worst-case training complexity is still the same as the baseline in this case. But, if the dataset has a very small ratio of distinct values compared to the number of

Table 5.2: Accuracy of Naïve Bayes and ECOC-NB with different BCH code lengths. Both use kernel density estimators. Bold values depict the best performance for the dataset in row and the underlined values among Tables 5.1 and 5.2.

data	NB	15	31	63	127	255	511	1023
fbis	**56.27**	43.36	49.49	50.14	51.23	51.56	52.45	51.93
la1	**78.59**	74.63	77.75	–	–	–	–	–
la2	75.02	74.31	**75.35**	–	–	–	–	–
oh0	79.95	79.06	80.06	79.76	80.26	80.16	**80.36**	–
oh5	74.29	72.55	73.30	74.50	74.17	**75.05**	74.39	–
oh10	69.43	67.05	68.10	69.24	68.86	69.43	**69.62**	–
oh15	**70.32**	69.22	70.10	69.98	70.31	70.20	69.98	–
re0	**66.42**	63.90	64.69	64.76	64.96	64.83	65.03	64.90
re1	69.16	70.91	71.52	72.84	72.54	**72.96**	72.36	72.72
tr11	**49.30**	42.29	49.06	46.88	48.08	48.33	–	–
tr12	47.62	48.59	48.34	52.42	**52.44**	–	–	–
tr21	**44.33**	39.30	41.36	–	–	–	–	–
tr23	**53.33**	30.31	32.81	–	–	–	–	–
tr31	66.66	70.12	70.87	**71.73**	–	–	–	–
tr41	76.66	77.46	79.62	80.87	80.75	**80.98**	**80.98**	–
tr45	54.93	49.71	54.78	53.91	54.64	**55.65**	55.22	–
wap	**74.17**	69.23	70.13	70.58	71.28	71.47	71.67	71.47

instances, it is significantly smaller. This is also the case here, the last column of Table 5.5 on page 54 shows the ratio of the sum of distinct values over all attributes to the number of instances times the number of features, which are all far away from the worst-case scenario. In addition, the tight combination of ECOC and Naïve Bayes may benefit also from the reduced overhead on the programming language level, e.g., less function calls and I/O operations.

Note for discrete and normal density estimation, the difference of training time between ECOC-NB to NB is independent of the number of instances t. For instance, if dataset *fbis* had far more instances, the training time of ECOC-NB with 31-bit BCH codes will still only last about 1 sec longer than standard Naïve Bayes.

5.4 Conclusions

In this chapter, we presented a simple combined computation of ECOC ensembles with Naïve Bayes as base learner. Compared to the straight-forward method with a training complexity of $O(n \cdot t \cdot g)$ its complexity using normal and discrete density estimation methods is reduced to $O((n + t) \cdot g)$.

In conjunction with kernel density estimators the worst-case complexity remains the same, but, in contrast, it can benefit from a low number of distinct feature values. We show some empirical evaluations supporting this statement and expect similar training complexity reduction also on the majority of real-world datasets, which, in our experience, typically exhibit such a low diversity.

Table 5.3: Training time comparison of Naïve Bayes and ECOC-NB with different BCH code lengths (with precalculation and normal density estimators, in seconds). For bit-lengths $15, 31, 63$ and 127 the second column shows the corresponding training time for the straight-forward ECOC implementation.

data	NB	15-Bit BCH		31-Bit BCH		63-Bit BCH		127-Bit BCH		255-BCH	511-BCH	1023-BCH
fbis	11.16	11.34	165.24	12.03	340.76	11.84	630.94	12.58	1355.48	12.78	14.04	20.39
la1	101.74	101.20	1596.21	110.98	3219.63	–	–	–	–	–	–	–
la2	91.02	92.41	1400.38	98.21	2865.61	–	–	–	–	–	–	–
oh0	4.23	4.39	57.30	4.54	122.39	4.57	235.56	4.81	489.79	5.27	7.07	–
oh5	3.80	3.79	48.54	3.86	104.57	3.92	201.35	4.12	414.65	4.66	6.75	–
oh10	4.57	4.73	61.87	4.87	132.27	4.92	253.59	5.15	524.22	5.69	7.84	–
oh15	3.98	3.94	52.04	4.06	109.39	4.18	210.48	4.39	430.19	4.84	6.96	–
re0	6.07	6.14	76.51	6.16	156.57	6.24	309.98	6.48	657.10	7.03	8.54	15.11
re1	9.08	9.00	118.84	8.97	244.10	9.25	484.02	9.85	1013.32	10.98	12.93	21.55
tr11	4.37	4.45	61.24	4.51	124.40	4.59	249.29	5.03	506.52	6.05	–	–
tr12	2.83	2.85	39.66	2.92	81.02	3.03	161.02	3.39	338.22	–	–	–
tr21	4.35	4.46	62.16	4.51	128.97	–	–	–	–	–	–	–
tr23	1.78	1.82	24.92	1.85	50.94	–	–	–	–	–	–	–
tr31	16.10	16.49	229.83	16.35	467.13	16.32	940.35	–	–	–	–	–
tr41	11.23	11.31	153.79	11.35	313.71	11.46	629.27	12.15	1314.06	13.06	15.62	–
tr45	9.82	9.94	134.89	9.80	279.80	10.16	558.91	10.87	1174.95	11.63	14.58	–
wap	23.78	23.19	309.03	22.93	651.38	23.68	1294.39	24.63	2714.96	26.04	29.62	40.95

Table 5.4: Training time comparison of Naïve Bayes and ECOC-NB with different BCH code lengths (with precalculation and kernel density estimators, in seconds). For bit-lengths $15, 31, 63$ and 127 the second column shows the corresponding training time for the straight-forward ECOC implementation.

data	NB	15-Bit BCH		31-Bit BCH		63-Bit BCH		127-Bit BCH		255-BCH	511-BCH	1023-BCH
fbis	11.85	13.53	181.11	13.51	376.83	13.62	750.80	15.88	1559.50	17.58	23.74	38.46
la1	107.12	120.18	1700.63	119.79	3474.00	–	–	–	–	–	–	–
la2	105.69	106.56	1517.53	107.69	3080.02	–	–	–	–	–	–	–
oh0	4.95	5.26	67.72	5.60	140.77	5.95	286.17	7.30	584.12	9.10	15.83	–
oh5	4.31	4.55	58.76	4.86	121.46	5.17	248.96	6.49	505.73	8.88	15.01	–
oh10	5.36	5.72	73.57	6.02	152.98	6.30	311.64	7.78	635.28	10.32	16.82	–
oh15	4.48	4.73	60.54	5.01	124.98	5.36	252.04	6.75	513.94	9.14	15.48	–
re0	6.94	7.25	91.68	7.59	190.76	7.83	388.70	9.58	792.19	12.27	18.31	32.40
re1	10.24	10.94	136.05	11.76	283.06	12.72	575.87	16.61	1175.56	22.14	36.46	63.40
tr11	4.76	5.25	66.61	5.79	138.48	6.62	280.17	9.19	568.41	13.53	–	–
tr12	3.12	3.49	43.84	3.92	90.84	4.69	184.31	6.64	374.89	–	–	–
tr21	4.76	5.26	69.61	5.75	143.73	–	–	–	–	–	–	–
tr23	1.92	2.29	27.81	2.72	57.59	–	–	–	–	–	–	–
tr31	17.75	18.41	256.99	19.18	526.89	19.90	1070.32	–	–	–	–	–
tr41	12.37	12.96	172.23	13.68	357.73	14.48	719.62	18.15	1489.90	23.13	36.71	–
tr45	10.75	11.52	152.41	12.30	313.86	13.30	640.29	17.14	1318.36	22.33	37.56	–
wap	25.87	27.24	354.87	28.60	731.24	30.24	1482.71	37.65	3110.42	46.45	70.85	119.63

Table 5.5: Dataset characteristics. This table shows the number of instances t, the number of features g, the number of classes k and the diversity of a dataset.

data	#instances	#features	#classes	diversity
fbis	2463	2000	17	0.0050
la1	3204	31472	6	0.0013
la2	3075	31472	6	0.0013
oh0	1003	3182	10	0.0038
oh5	918	3012	10	0.0038
oh10	1050	3238	10	0.0043
oh15	913	3100	10	0.0042
re0	1504	2886	13	0.0023
re1	1657	3758	25	0.0022
tr11	414	6429	9	0.0127
tr12	313	5804	8	0.0151
tr21	336	7902	6	0.0184
tr23	204	5832	6	0.0294
tr31	927	10128	7	0.0058
tr41	878	7454	10	0.0049
tr45	690	8261	10	0.0076
wap	1560	8460	20	0.0022

A possible disadvantage of the decomposition approach is the need for tuning parameters such as the bit-length. However, with the efficient computation scheme proposed in this chapter, the cost of such a parameter tuning has become feasible. Furthermore, ECOC-NB can benefit naturally from sophisticated or more specialized code types in the future, which is an active research topic (e.g., Pujol et al., 2006).

In summary, we have shown that the combination of Naïve Bayes with error-correcting output codes is almost as fast as a conventional Naïve Bayes classifier. ECOC are thus a viable technique for improving the predictive performance of Naïve Bayes on large-scale datasets.

6 Efficient ECOC Prediction

Contents

Pairwise classification may be viewed as a special case of ternary error-correcting output codes which are a general framework for describing different decompositions of a multiclass problem into a set of binary problems. Although not strictly necessary, the number of the generated binary classification problems typically exceeds the number of class values ($n > k$), for many common general encoding techniques by several orders of magnitude. For example, for the above-mentioned pairwise classification, the number of binary classifiers is quadratic in the number of classes. Thus, the increase in predictive accuracy comes with a corresponding increase in computational demands at classification time.

For pairwise classification the recently proposed algorithm QWeighted was able to reduce the computational costs at prediction time to some extent. In this chapter, we generalize this algorithm to allow for quick decoding of arbitrary ternary ECOC ensembles. The resulting predictions are, similar to QWeighted, guaranteed to be equivalent to the original decoding strategy except for ambiguous final predictions. In addition, we will show that the algorithm is applicable to various decoding techniques.

6.1 Efficient ECOC Decoding

In this section, we will generalize the QWeighted algorithm to arbitrary ternary ECOC matrices. We will then discuss the three key modifications that have to be made: first, Hamming decoding has to be reduced to a voting process (Section 6.1.1), second, the heuristic for selecting the next classifier has to be adapted to the case where multiple classifiers can be incident with a pair of classes (Section 6.1.2), and

finally the stopping criterion can be improved to take multiple incidences into account (Section 6.1.3). We will then present the resulting QuickECOC algorithm for Hamming decoding in Section 6.1.4. Finally, we will discuss how QuickECOC can be adapted to different decoding techniques (Section 6.1.5).

6.1.1 Reducing Hamming Distances to Voting

Obviously, pairwise classification may be considered as a special case of ternary ECOCs, where each column of the coding matrix contains exactly one positive, one negative, and $k-2$ ignore values, as shown in Example 3.2. Thus, it is natural to ask the question whether the QWeighted algorithm can be generalized to arbitrary ternary ECOCs.

To do so, we first have to consider that ECOCs typically use Hamming distance for decoding, whereas pairwise classification typically uses a simple voting procedure. In voting aggregation, the class that receives the most votes from the binary classifiers is predicted, i.e.,

$$\tilde{c} := \operatorname*{argmax}_{c_i \in K} \sum_{j \neq i,\, c_j \in K} f_{i,j}$$

where $f_{i,j}$ is the prediction of the pairwise classifier that discriminates between classes c_i and c_j.

Traditional ECOC with Hamming decoding predicts the class c^* whose code word $\vec{cw}_{c^*}$ has the minimal Hamming Distance $d_H(\vec{cw}_{c^*}, \vec{p})$ to the prediction vector $\vec{p} = (p_1, \ldots, p_n)$. A certain analogy between both methods can be seen easily and was further examined by Kong and Dietterich (1995) and has a relation to *correlation decoding* from coding theory (Gallager, 1968). However, we briefly repeat with following lemma which shows that the minimization of Hamming distances reduces to voting aggregation:

Lemma 1 *Let* $v_{i,j} := \left(1 - \frac{|m_{i,j} - p_j|}{2}\right)$ *be the vote that classifier* f_j *gives to class* c_i*, then*

$$\operatorname*{argmin}_{i=1\ldots k} d_H(\vec{cw}_i, \vec{p}) = \operatorname*{argmax}_{i=1\ldots k} \sum_{j=1}^{n} v_{i,j}$$

Proof Recall that

$$d_H(\vec{cw}_i, \vec{p}) = \sum_{a=1}^{n} \frac{|cw_{i_a} - p_a|}{2} = \sum_{a=1}^{n} \frac{|m_{i,a} - p_a|}{2}$$

Let $b_{i,a} := \frac{|m_{i,a} - p_a|}{2}$. Since for each codebit $b_{i,a} \in [0,1]$,

$$\operatorname*{argmin}_{i=1\ldots k} \sum_{a=1}^{n} b_{i,a} = \operatorname*{argmax}_{i=1\ldots k} \sum_{a=1}^{n} (1 - b_{i,a}) = \operatorname*{argmax}_{i=1\ldots k} \sum_{a=1}^{n} v_{i,a}$$

holds and we obtain the proposition.

6.1.2 Next Classifier Selection

The QWeighted algorithm always pairs the class with the least amount of voting loss l_i with the class that has the least amount of voting loss among all remaining classes with which it has not yet been paired, and evaluates the resulting classifier. This choice is deterministic because, obviously, there is only one classifier that is incident with any given pair of classes.

General ECOC coding matrices, on the other hand, can have more then two non-zero entries per column. As a result, a pair of classes may be incident to multiple binary classifiers. This has the consequence that the selection of the next classifier to evaluate has gained an additional degree of freedom. For instance, consider a 4-class problem (c_1, c_2, c_3, c_4) using 3-level ternary exhaustive codes, as shown in Example 3.3. If classes c_1 and c_2 are those with the current minimum voting loss, we could select any of four different classifiers that discriminate the classes c_1 and c_2, namely $f_2 = f_{\{1,3\},\{2\}}$, $f_3 = f_{\{1\},\{2,3\}}$, $f_5 = f_{\{1,4\},\{2\}}$, and $f_6 = f_{\{1\},\{2,4\}}$.

QuickECOC uses a selection process conforming to the key idea of QWeighted: Given the current favorite class c_{i_0}, we select all incident classifiers C_{i_0}, i.e.,

$$C_{i_0} = \{f_{P_j,N_j} \in C \mid c_{i_0} \in P_j \vee c_{i_0} \in N_j\} \tag{6.1}$$

Let K_j denote the set of classes, which are involved in the binary classifier $f_j = f_{P_j,N_j}$, but with a different sign than c_{i_0}, i.e.,

$$K_j = \begin{cases} P_j & \text{if } c_{i_0} \in N_j \\ N_j & \text{if } c_{i_0} \in P_j \end{cases}$$

In other words, K_j contains all rows i of column j in the coding matrix M, for which $m_{i,j} \neq m_{i_0,j}$ and $m_{i,j} \neq 0$ hold. We then compute a score

$$s(j) = \sum_{c_i \in K_j} k - \tau(c_i)$$

for every classifier $f_j \in C_{i_0}$, where $\tau(c_i)$ denotes the rank of class c_i when all classes are increasingly ordered by their current votings (or, equivalently, ordered by decreasing distances). Finally, we select the classifier f_{j_0} with the maximal score $s(j_0)$. Roughly speaking, this amounts to selecting the classifier which discriminates c_{i_0} to the greatest number of currently highly ranked classes.

We experienced that this simple score-based selection was superior among other tested methods, whose presentation and evaluation we omit here. One point to note is, that for the special case of pairwise codes, this scheme is identical to the one used by QWeighted.

6.1.3 Stopping Criterion

The key idea of the algorithm is to stop the evaluation of binary classifiers as soon as it is clear which class will be predicted, irrespective of the outcome of all other

classifiers. Thus, the QuickECOC algorithm has to check whether c_{i_0}, the current class with the minimal Hamming distance to $\vec{p}$, can be caught up by other classes at the current state. If not, c_{i_0} can be safely predicted.

A straight-forward adaptation of the QWeighted algorithm for pairwise classification would simply compute the maximal possible Hamming distance for c_{i_0} and compare this distance to the current Hamming distances l_i of all other classes $c_i \in K \setminus \{c_{i_0}\}$. The maximal possible Hamming distance for c_{i_0} can be estimated by assuming that all outstanding evaluations involving c_{i_0} will increase its Hamming distance by 1 and all remaining outstanding (non-incident) classifiers will increase its distance by 0.5 (according to the definition of hamming distance for ternary code words). Thus, we simply add the number of remaining incident classifiers of c_{i_0} and one half of the number of remainder classifiers to its current distance l_{i_0}.

Note, however, that this simple method makes the assumption that all binary classifiers only increase the Hamming distance of c_{i_0}, but not of the other classes. This is unnecessarily pessimistic, because each classifier will always equally increase the Hamming distance for *all* (or none) of the incident classes that have the same sign in the coding matrix (positive or negative). Thus, we can refine the above procedure by computing a separate upper bound of l_{i_0} for each class c_i. This bound does not assume that all remaining incident classifiers will increase the distance for c_{i_0} by 1, but only those where c_i and c_{i_0} are on different sides of the training set. For the cases where either c_i or c_{i_0} was ignored in the training phase, $\frac{1}{2}$ is added to the distance. If there exist no class which can overtake c_{i_0}, the algorithm returns c_{i_0} as the prediction.

Note that the stopping criterion can only test whether no class can surpass the current favorite class. However, there may be other classes with the same Hamming distance. As the QuickECOC algorithm will always return the first class that cannot be surpassed by other classes, this may not be the same class that is returned by the full ECOC ensemble. Thus, in the case, where the decoding is not unique, QuickECOC may return a different prediction. However, in all cases where the code word minimal Hamming distance is unique, QuickECOC will return exactly the same prediction as ECOC.

We also defined a second criterion, which simply stops when all classifiers of the favorite class c_{i_0} have already been evaluated. Strictly speaking, this is a special case of the first stopping criterion and could be removed. However, we found that making this distinction facilitated some of our analyses (presented in Section 6.3 on page 78), so we leave it in the algorithm.

6.1.4 Quick ECOC Algorithm

Algorithm 6 on the facing page shows the pseudocode of the QuickECOC algorithm. The algorithm maintains a vector $\vec{l} = (l_1, l_2, \ldots, l_k) \in \mathbb{R}^k$, where l_i is the current accumulated Hamming distance of the prediction vector $\vec{p}$ to the code word $\vec{cw}_i$ of class c_i. The l_i can be seen as lower bounds of the distances $d_H(\vec{cw}_i, \vec{p})$, which are updated incrementally in a loop which essentially consists of four steps:

Algorithm 6 QuickECOC

Require: ECOC Matrix $\vec{M} = (m_{i,j}) \in \{-1, 0, 1\}^{k \times n}$, binary classifiers $C = \{f_1, \ldots, f_n\}$, testing instance $\vec{x} \in X$

1: $\vec{l} \in \mathbb{R}^k \leftarrow 0$ # Hamming distance vector
2: $c^* \leftarrow$ NULL
3: $C' \leftarrow C$
4: **while** $c^* =$ NULL **do**
5: $f_j \leftarrow$SelectNextClassifier$(\vec{M}, \vec{l})$
6: $p \leftarrow f_j(\vec{x})$ # Evaluate classifier
7: **for each** $i \in K$ **do**
8: $l_i \leftarrow l_i + \frac{|m_{i,j} - p|}{2}$
9: $C' \leftarrow C' \setminus \{f_j\}$
10: $\vec{M} \leftarrow \vec{M} \setminus M_j$
11: $c_{i_0} \leftarrow \underset{c_i \in K}{\operatorname{argmin}} l_i$
12: # First stopping criterion
13: abort $\leftarrow$ **true**
14: **for each** $c_i \in K \setminus \{c_{i_0}\}$ **do**
15: $n_{Full} \leftarrow \left|\{f_j \in C' \mid m_{i,j} \cdot m_{i_0,j} = -1\}\right|$
16: $n_{Half} \leftarrow \left|\{f_j \in C' \mid m_{i,j} \cdot m_{i_0,j} = 0 \textbf{ and } m_{i,j} + m_{i_0,j} \neq 0\}\right|$
17: **if** $l_{i_0} + n_{Full} + \frac{1}{2} n_{Half} > l_i$ **then**
18: abort $\leftarrow$ **false**
19: # Second stopping criterion
20: **if** abort **or** $\forall f_j \in C' : m_{i_0,j} = 0$ **then**
21: $c^* \leftarrow c_{i_0}$
22: **return** c^*

(1) Selection of the Next Classifier:
First, the next classifier is selected. Depending on the current Hamming distance values, the routine SelectNextClassifier returns a classifier that pairs the current favorite $c_{i_0} = \operatorname{argmin}_{c_i \in K} l_i$ with another class that is selected as described in Section 6.1.2. In the beginning all values l_i are zero, so that SelectNextClassifier returns an arbitrary classifier f_j.

(2) Classifier Evaluation and Update of Bounds $\vec{l}$:
After the evaluation of f_j, $\vec{l}$ is updated using the Hamming distance projected to this classifier (as described in Section 6.1.1) and f_j is removed from the set of possible classifiers.

(3) First Stopping Criterion:
Starting with Line 12, the first stopping criterion is checked. It checks whether the current favorite class c_{i_0} can already be safely determined as the class with the maximum number of votes, as described in Section 6.1.3.

(4) Second Stopping Criterion:
Starting with Line 19, the algorithm stops when all incident classifiers of c_{i_0} have been evaluated. In this case, since it holds that $l_{i_0} \leq l_i$ for all classes c_i with l_{i_0} fixed and considering that l_i can only increase monotonically, we can safely ignore all remaining evaluations.

6.1.5 Adaption to Different Decoding Strategies

As briefly discussed in Section 3.4.2 on page 22, various decoding strategies have been proposed as alternatives to Hamming decoding. In this section, we show how QuickECOC can be adapted to a variety of domain-independent encoding strategies via small modifications to the basic algorithm.

In general, there are two locations where adaptations are needed. First, the statistics update step and the first stopping criteria have to be adapted according to the used distance measure. Second, some decoding strategies require a special treatment of the zero symbol, which can, in general, be modeled as a preprocessing step.

In the following, we review some important decoding strategies and show how QuickECOC can be adapted to deal with each strategy.

Euclidean Distance

The Euclidean Distance d_E computes the distance between the code-word and the prediction vector in Euclidean space.

$$d_E\left(\vec{cw}_i, \vec{p}\right) = \left|\left|\vec{cw}_i - \vec{p}\right|\right|_2 = \sqrt{\sum_{j=1}^{n} \left(m_{i,j} - p_j\right)^2} \tag{6.2}$$

For minimizing this distance we can ignore the root operation and, instead, minimize the sum of squared bit-wise differences of both vectors. This can again be computed incrementally, by substituting the update statement of the pseudocode (Line 8) with:

$$l_i \leftarrow l_i + \left(m_{i,j} - p\right)^2$$

Consequently, in the sum in Line 17, the weight for n_{Half} is changed to 1 and the one for n_{Full} to 4.

Attenuated Euclidean/Hamming Distance

These measures, introduced by Escalera et al. (2006), work analogously to the Hamming Distance and the Euclidean distance, but distances to zero symbols in the coding vector are ignored (which is equivalent to weighting the distance with $|m_{i,j}|$). The attenuated Euclidean distance is thus defined as

$$d_{AE}\left(\vec{cw}_i, \vec{p}\right) = \sqrt{\sum_{j=1}^{n} \left|m_{i,j}\right| \left(m_{i,j} - p_j\right)^2} \tag{6.3}$$

The analogue version for the Hamming distance is:

$$d_{AH}\left(\vec{cw}_i, \vec{p}\right) = \sum_{j=1}^{n} |m_{i,j}| \frac{|m_{i,j} - p_j|}{2} \tag{6.4}$$

The modifications to Lines 8 and 17 are analogous to the previous case.

Loss-based Decoding

In loss-based decoding (Allwein et al., 2000) we assume a score-based base classifier, and want to take the returned score $f(.)$ into consideration. The similarity function d_L is then defined as

$$d_L\left(\vec{cw}_i, \vec{p}\right) = \sum_{j=1}^{n} l\left(m_{i,j} \cdot f_j\right) \tag{6.5}$$

where $l(.)$ is a loss function, such as $l(s) = -s$ or the exponential loss $l(s) = e^{-s}$.

For both loss functions, we assume that we have given a normalizing function $w(.)$ which projects $f_j(x)$ into the interval $[-1, 1]$, e.g.,[1]

$$w(z) = \begin{cases} \frac{z}{\max z} & z \geq 0 \\ \frac{z}{|\min z|} & z < 0 \end{cases}$$

For the linear loss, we substitute Line 6 with

$$p \leftarrow w\left(f_j(x)\right)$$

and the update in Line 8 with

$$l_i \leftarrow l_i + \frac{1 - p \cdot m_{i,j}}{2}$$

and remove the occurrences of n_{Half}.

For the exponential loss, we have to change Line 6 as above and the update step with

$$l_i \leftarrow l_i + e^{-p \cdot m_{i,j}}.$$

In addition, the weights in Line 17 are set to e^1 for n_{Full} and to $e^0 = 1$ for n_{Half}.

1 Note that we did not use such a normalizing function in our actual evaluation since we used a decision tree learner which already returns scores in the right range. Although the normalization of score-based functions, such as SVMs, is not a trivial task, the sketched function $w(.)$ could be possibly determined by estimating $\min f(x)$ and $\max f(x)$ during training time (e.g., saving the largest distances between instances to the hyperplane for each classifier).

Laplace Strategy

The Laplace Strategy (Escalera et al., 2006) interprets the zero symbol in a different way: If a code word $c\vec{w}_1$ consists of more zero symbols than $c\vec{w}_2$, the number of "reasonable" predictions is smaller, so every non-zero symbol prediction of $c\vec{w}_1$ should be given a higher weight.

$$d_{LA}\left(c\vec{w}_i, \vec{p}\right) = \frac{E+1}{E+C+T} = \frac{d_{AH}\left(c\vec{w}_i, \vec{p}\right)+1}{\sum_{j=1}^{n}|m_{i,j}|+T} \tag{6.6}$$

where C is the number of bit positions in which they are equal and E in which they differ. So, roughly speaking, depending of the number of zero symbols of $c\vec{w}_i$, every bit agreement contributes more or less to the distance measure. T is the number of involved classes, in our case $T = 2$, since we employ binary classifiers. Thus, the default value of $d_{LA}(.)$ is $\frac{1}{2}$.

This strategy can be used by incorporating a class- respectively row-based incrementer. Note that each error bit between a code word $c\vec{w}$ and the prediction vector $\vec{p}$ contributes $1/(b+T)$ to the total distance $d_{LA}(c\vec{w}, \vec{p})$, where b is the number of non-zero bits of $c\vec{w}$. This incrementer denoted by I_i for class c_i can be computed as a preprocessing step from the given ECOC Matrix. So, the update step in Line 8 has to be changed to

$$l_i \leftarrow l_i + I_i$$

and the weight of n_{Full} in the sum in Line 17 changes to I_i. Besides, n_{Half} can be removed.

Beta Density Distribution Pessimistic Strategy

This measure also proposed by Escalera et al. (2006) assumes that the distance is a Beta-distributed random variable parametrized by C and E of two code words. The Beta distribution is here defined as

$$\psi(z, E, C) = \frac{1}{T} z^E (1-z)^C$$

Its expected value is $\mathrm{E}(\psi_i) = \frac{E}{E+C}$.
Let $Z_i := \mathrm{argmax}_{z\in[0,1]}(\psi_i(z))$ and $a_i \in [0,1]$ such that

$$\int_{Z_i}^{Z_i+a_i} \psi_i(z) = \frac{1}{3}$$

then we define

$$d_{BD}(x, y) = Z_i + a_i \tag{6.7}$$

The value a_i is regarded as a pessimistic value, which incorporates the uncertainty of Z_i into the distance measure.

Here, we use an approximation of the original strategy. First, similar to the Laplace Strategy, an incrementer is used to determine $Z_i = \frac{E}{E+C}$. And second, instead of using

a numerical integration to determine $Z_i + a_i$, its standard deviation is added, which is in compliance with the intended semantic of this overall strategy to incorporate the uncertainty. The incrementer I_i is again set during a preprocessing step and we change the update step (Line 8) to

$$l_i \leftarrow l_i + \min\left(1, (I_i + \sigma_i)\right)$$

The weight for n_{Full} has to be changed to I_i and n_{Half} has to be removed. In practice, this approximation yielded in all our evaluations the same prediction as the original strategy.

Remarks

The presented decoding techniques were just a few examples, which we have empirically tested. Other techniques can be adapted in a similar fashion. In general, a distance measure is compatible to QuickECOC if the distance can be determined bit-wise or incremental, and the iterative estimate of l_i has to be monotonically increasing, but must never over-estimate the true distance.

6.2 Experimental Evaluation

In this section, we show the results of the empirical evaluation of our algorithms. We focus on the number of classifier evaluations that have to be performed in order to compute a prediction. We typically do not compare our results in terms of predictive accuracy, because our algorithms make the same prediction as their respective versions that use all classifiers. Nevertheless, unless mentioned otherwise, the reported results are averages of a 10-fold cross-validation, in order to get more reliable estimates.

Before we present the results on QuickECOC in Section 6.2.2, we will provide supplemental evaluations regarding QWeighted, which extend previous experiments in (Park and Fürnkranz, 2007) by using parameter-tuned classifiers, addressing large scale datasets and providing an analysis on the computational overhead. Since, QWeighted resembles in principle a special case of QuickECOC in conjunction with pairwise codes, this helps to determine the overall performance of QuickECOC including pairwise codes.

6.2.1 Pairwise Classification - Evaluation of QWeighted

UCI Datasets

We start with a comparison of the QWeighted algorithm with the full pairwise classifier and with DDAGs on seven arbitrarily selected multiclass datasets from the UCI database of machine learning databases (Asuncion and Newman, 2010). We used four commonly used learning algorithms as base learners (the rule learner

RIPPER,[2] a Naive Bayes algorithm (NB), the C4.5 decision tree learner (J48), and a support vector machine (SMO) all in their implementations in the WEKA machine learning library (Hall et al., 2009). Each algorithm was used as a base classifier for QWEIGHTED, and the combination was run on each of the datasets. A mild parameter tuning was applied to each base algorithm, which does not necessarily help to answer the question of the choice for best predictive combination on these datasets because of its non-exhaustive conducting, but were rather applied to take the fact into account that the predictive performance has an impact on the efficiency of the QWEIGHTED algorithm. Inner 5-fold cross-validation tuning[3] was applied for following base learners and parameters:

- NB:
  - with or without kernel density estimators (cf. Section 5.2.3 on page 46)
- SMO:
  - complexity $\{0.1, 0.2, \ldots, 1\}$
  - exponent of polynomial kernel $\{0.5, 1, 1.5, 2\}$
- J48:
  - confidence factor $\{0.02, 0.04, \ldots, 0.5\}$
  - minimum number of instances per leaf $\{1, 2, 3, 4\}$
- JRIP:
  - folds for pruning $\{2, 3, 4\}$
  - minimum total weight of instances per rule $\{1, 2, 3\}$
  - number of optimization runs $\{1, 2, 3\}$

All results are obtained via a 10-fold cross-validation except for *letter*, where the supplied testset was used. The same experiments were already performed without parameter tuning and can be found in (Park and Fürnkranz, 2007).

Table 6.1 on the next page shows the results. With respect to accuracy, there are only 5 cases in a total of 28 experiments (4 base classifiers $\times$ 7 datasets) where DDAGs outperformed QWEIGHTED, whereas QWEIGHTED outperformed DDAGs in 20 cases (3 experiments ended in a tie). Even according to the very conservative sign test, this difference is significant with $p = 0.004$. This and the fact that, to the best of our knowledge, it is not known what loss function is optimized by DDAGs, confirm our intuition that QWEIGHTED is a more principled approach than DDAGs.

With respect to the number of comparisons, it can be seen that the average number of comparisons needed by QWEIGHTED is much closer to the best case than to

2 We used a double round robin for Ripper for both, the full pairwise classifier and for QWEIGHTED. In order to be comparable to the other results, we, in this case, divide the observed number of comparisons by two.

3 The *CVParameterSelection* method implemented in WEKA was used for parameter tuning.

Table 6.1: Comparison of QWeighted and DDAG with different base learners on seven multiclass datasets. The right-most column shows the number of comparisons needed by a full pairwise classifier ($k(k-1)/2$). Next to the average numbers of comparisons $\hat{n}$ for QWeighted we show their trade-off $\frac{\hat{n}-(k-1)}{\max-(k-1)}$ between best and worst case (in brackets).

			Accuracy		∅ Comparisons		
dataset	k	learner	QWeighted	DDAG	QWeighted	DDAG	full
vehicle	4	NB	61.24 ± 4.66	61.12 ± 4.75	4.11 (*0.370*)	3	6
		SMO	80.62 ± 4.30	81.09 ± 4.57	3.70 (*0.233*)		
		J48	71.40 ± 3.06	70.70 ± 2.64	3.94 (*0.314*)		
		JRip	69.63 ± 7.30	69.99 ± 6.36	4.00 (*0.335*)		
glass	7	NB	51.88 ± 7.77	51.43 ± 7.11	9.68 (*0.245*)	6	21
		SMO	62.66 ± 5.89	63.59 ± 6.29	10.03 (*0.269*)		
		J48	70.09 ± 7.43	68.25 ± 5.60	9.81 (*0.254*)		
		JRip	68.27 ± 10.39	66.43 ± 10.11	9.77 (*0.251*)		
image	7	NB	85.76 ± 1.20	85.76 ± 1.20	8.95 (*0.197*)	6	21
		SMO	96.58 ± 1.14	96.62 ± 1.13	8.04 (*0.136*)		
		J48	95.84 ± 1.29	96.36 ± 1.08	8.65 (*0.177*)		
		JRip	96.67 ± 1.32	96.45 ± 1.48	8.77 (*0.185*)		
yeast	10	NB	59.16 ± 3.58	58.96 ± 3.46	15.96 (*0.193*)	9	45
		SMO	58.28 ± 4.05	58.15 ± 3.83	15.48 (*0.180*)		
		J48	58.96 ± 3.58	58.29 ± 3.75	15.61 (*0.184*)		
		JRip	58.89 ± 3.59	57.81 ± 3.31	15.68 (*0.185*)		
vowel	11	NB	71.41 ± 5.57	71.31 ± 5.43	17.19 (*0.160*)	10	55
		SMO	98.59 ± 1.28	98.38 ± 1.66	14.88 (*0.108*)		
		J48	82.73 ± 3.48	78.79 ± 4.07	16.99 (*0.155*)		
		JRip	83.64 ± 5.28	77.88 ± 6.01	17.64 (*0.170*)		
soybean	19	NB	92.96 ± 1.83	92.96 ± 1.83	27.70 (*0.063*)	18	171
		SMO	93.40 ± 2.63	93.11 ± 2.70	28.36 (*0.068*)		
		J48	93.55 ± 2.63	91.36 ± 2.55	28.98 (*0.072*)		
		JRip	93.70 ± 1.97	92.67 ± 2.30	29.79 (*0.077*)		
letter	26	NB	73.93	73.88	43.77 (*0.063*)	25	325
		SMO	91.70	91.13	41.49 (*0.055*)		
		J48	91.10	85.90	47.86 (*0.076*)		
		JRip	90.23	85.83	47.33 (*0.068*)		

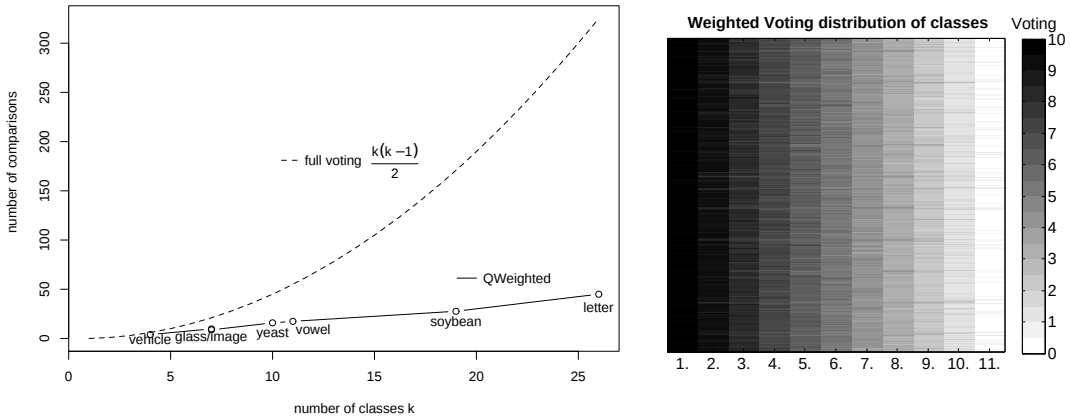


Figure 6.1: a) Efficiency of QWeighted in comparison to a full pairwise classifier, b) Distribution of votes for *vowel* (11-class problem, base learner SMO). The x-axis describes the ranking positions.

the worst case. Next to the absolute numbers, we show the trade-off between best and worst case (in brackets). A value of 0 indicates that the average number of comparisons is $k-1$, a value of 1 indicates that the value is $k(k-1)/2$ (the value in the last column). As we have ordered the datasets by their respective number of classes, we can observe that this value has a clear tendency to decrease with the number of the classes. For example, for the 19-class *soybean* and the 26-class *letter* datasets, only about $6-7\,\%$ of the possible number of additional pairwise classifiers are used, i.e., the total number of comparisons seems to grow only linearly with the number of classes. This can also be seen from Figure 6.1a, which plots the datasets with their respective number of classes together with a curve that indicates the performance of the full pairwise classifier.

Finally, we note that the results are qualitatively the same for all base classifiers. QWeighted does not seem to depend on a choice of base classifiers.

Simulation Experiment

For a more systematic investigation of the complexity of the algorithm, we performed a simulation experiment. We assume classes in the form of numbers from $1 \ldots k$, and, without loss of generality, 1 is always the correct class. We further assume pairwise base pseudo-classifiers $f^{\epsilon}_{i,j}$, which, for $i < j$, return *true* with a probability $1-\epsilon$ and *false* with a probability ϵ. For each example, the QWeighted algorithm is applied to compute a prediction based on these pseudo-classifiers. The setting $\epsilon = 0$ (or $\epsilon = 1$) corresponds to a pairwise classifier where all predictions are consistent with a total order of the possible class labels, and $\epsilon = 0.5$ corresponds to the case where the predictions of the base classifiers are entirely random.

Table 6.2: Average number $\hat{n}$ of pairwise comparisons for various number of classes and different error probabilities ϵ of the pairwise classifiers using QWeighted, and for the full pairwise classifier. Below, we show their trade-off $\frac{\hat{n}-(k-1)}{\max -(k-1)}$ between the best and worst case, and an estimate of the growth ratio $\frac{\log(\hat{n}_2/\hat{n}_1)}{\log(k_2/k_1)}$ of successive values of $\hat{n}$.

	$k = 5$	$k = 10$	$k = 25$	$k = 50$	$k = 100$
$\epsilon = 0.0$	5.43	14.11	42.45	91.04	189.51
	0.238 —	*0.142* **1.378**	*0.067* **1.202**	*0.036* **1.101**	*0.019* **1.058**
$\epsilon = 0.05$	5.72	16.19	60.01	171.53	530.17
	0.287 —	*0.200* **1.501**	*0.130* **1.430**	*0.104* **1.515**	*0.089* **1.628**
$\epsilon = 0.1$	6.07	18.34	76.82	251.18	900.29
	0.345 —	*0.259* **1.595**	*0.191* **1.563**	*0.172* **1.709**	*0.165* **1.842**
$\epsilon = 0.2$	6.45	21.90	113.75	422.58	1,684.21
	0.408 —	*0.358* **1.764**	*0.325* **1.798**	*0.318* **1.893**	*0.327* **1.995**
$\epsilon = 0.3$	6.90	25.39	151.19	606.74	2,504.54
	0.483 —	*0.455* **1.880**	*0.461* **1.974**	*0.474* **2.005**	*0.496* **2.045**
$\epsilon = 0.4$	6.93	27.73	182.58	776.98	3,265.56
	0.488 —	*0.520* **2.000**	*0.575* **2.057**	*0.619* **2.089**	*0.653* **2.071**
$\epsilon = 0.5$	7.12	28.74	198.51	868.25	3,772.45
	0.520 —	*0.548* **2.013**	*0.632* **2.109**	*0.697* **2.129**	*0.757* **2.119**
full	10	45	300	1,225	4,950

Table 6.2 on this page shows the results for various numbers of classes ($k = 5$, 10, 25, 50, 100) and for various settings of the error parameter ($\epsilon = 0.0$, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5). Each data point is the average outcome of 1000 trials with the corresponding parameter settings. We can see that even for entirely random data, our algorithm can still save about 1/4 of the pairwise comparisons that would be needed for the entire ensemble. For cases with a total order and error-free base classifiers, the number of needed comparisons approaches the number of classes, i.e., the growth appears to be linear.

To shed more light on this, we provide two more measures below each average: the lower left number (in italics) shows the trade-off between best and worst case, as defined above. The result confirms that for a reasonable performance of the base classifiers (up to about $\epsilon = 0.2$), the fraction of additional work reduces with the number of classes. Above that, we start to observe a growth. The reason for this is that with a low number of classes, there is still a good chance that the random base classifiers produce a reasonably ordered class structure, while this chance is decreasing with increasing numbers of classes. On the other hand, the influence of each individual false prediction of a base classifier decreases with an increasing number of classes, so that the true class ordering is still clearly visible and can be better exploited by the QWeighted algorithm.

This can also be seen in Figure 6.1b on page 66, which shows the distribution of the votes produced by the SVM base classifier for the dataset *vowel*. As shown on the scale to right, different shades of gray are used for encoding different numbers of received votes. Each horizontal line in the plot represents one example, the left shows the highest number of votes, the right the lowest number of votes. If all classes receive the same number of votes, the area should be colored uniformly. However, here we observe a fairly clear change in the color distribution, the dark areas to the left indicating the top-rank class often receives nine or more votes, and the bright areas to the right indicating that the lowest ranking class typically receives less than one vote (recall that we use weighted voting).

We tried to directly estimate the exponent of the growth function of the number of comparisons of QWeighted, based on the number of classes k. The resulting exponents, based on two successive measure points, are shown in bold font below the absolute numbers. For example, the exponent of the growth function between $k = 5$ and $k = 10$ is estimated (for $\epsilon = 0$) as $\frac{\log(14.11/5.43)}{\log(10/5)} \approx 1.378$. We can see that in the growth rate starts almost linearly (for a high number of classes and no errors in the base classifiers) and approaches a quadratic growth when the error rate increases.[4]

Datasets with a Large Number of Classes

In addition to the small datasets from Table 6.1 on page 65, we evaluated the QWeighted algorithm on three more real-world datasets with a relative high number of classes:

Uni-label RCV1-v2

RCV1-v2 (Lewis et al., 2004) is a dataset consisting of over 800,000 categorized news articles from Reuters, Ltd. For the category *topic* multiple labels from a total of 103 hierarchically organized labels are assigned to the instances. We transformed this original multilabel dataset to a multiclass dataset by selecting the assigned label with the greatest depth in the hierarchical tree as the class label. We applied this procedure on the provided trainset and testset no. 0 by Lewis et al. (2004) resulting to a multiclass dataset with 100 classes, 23,149 train- and 199,328 test-instances, with at least one positive example for each of the 100 classes. We selected 2,000 features according to a χ^2-based feature selection (Yang and Pedersen, 1997). We will refer to this created dataset as *urcv1-v2*.

ASTRAL 2 & 3

These datasets describe protein sequences retrieved from the SCOP 1.71 protein database (Murzin et al., 1995). We used ASTRAL (Brenner et al., 2000) to filter these sequences so that no two sequences share greater than 95 % identity.

4 At first sight, it may be surprising that some of the numbers are greater than 2. This is a result of the fact that $k(k-1)/2 = k^2/2 - k/2$ is quadratic *in the limit*, but for low values of k, the subtraction of the linear term $k/2$ has a more significant effect. Thus, e.g., the estimated growth of the full pairwise classifier from $k = 5$ to $k = 10$ is $\frac{\log(45/10)}{\log(10/5)} \approx 2.17$.

Table 6.3: Results of QWeighted for datasets with a relative high number of classes. In parentheses, we show their trade-off $\frac{\hat{n}-(k-1)}{\max-(k-1)}$ between the best and worst case (base learner J48).

dataset	k	QWeighted	full	Accuracy
urcv1-v2	100	312.62 (*0.0440*)	4,950	62.1
astral2	971	9,490.81 (*0.0018*)	470,935	24.8
astral3	1,588	28,476.20 (*0.0213*)	1,260,078	20.8

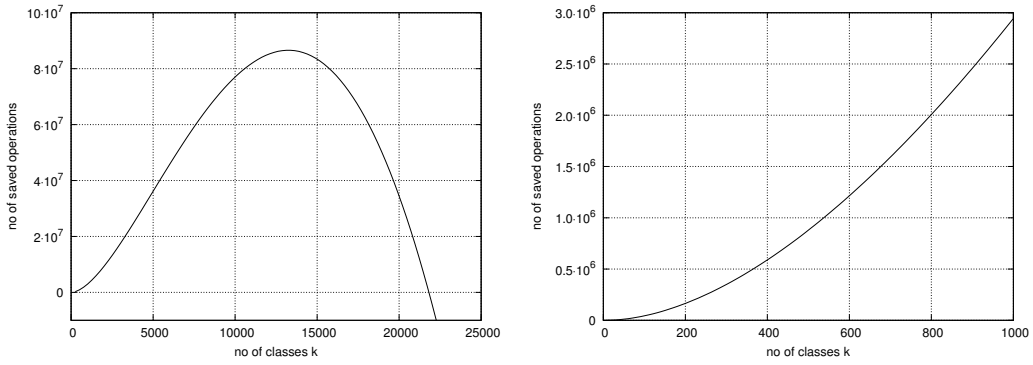
The class labels are organized in a 3-level hierarchy, consisting of protein folds, superfamilies and families (in descending order). *astral3* consists of 1,588 classes and contains the original hierarchy. To fill the gap between datasets *urcv1-v2* and *astral3* in terms of number of classes, we constructed a second dataset *astral2* by limiting the hierarchical depth to 2. So, two instances which previously shared the same superfamily x are now assigned to superfamily x as new class label. By decreasing the depth, the number of classes were reduced to 971. Both datasets have 13,006 instances and 21 numeric attributes (20 amino acids plus selenocysteine).

Table 6.3 shows the results of these experiments. For *astral2* and *astral3*, 66 percent of all instances were used for training and the rest for testing. Once again, trade-off values were estimated for the average number of pairwise comparisons. As these values show, QWeighted uses only a fairly small amount compared to a full voting aggregation and is much closer to the best case than to the worst case ($k(k-1)/2$ comparisons). One can see an increasing growth of the trade-off values between *astral2* and *astral3*. However, this effect can be explained with the general poor classification accuracy of protein sequences. According to the simulation results, there exist a correlation between performance of QWeighted and performance of the underlying base classifiers. The decreased accuracy on *astral3* compared to *astral2* (right-most column) indicates weaker base classifiers, which leads to a increasing number of needed pairwise comparisons.

In summary, our results indicate that the QWeighted algorithm always increases the efficiency of the pairwise classifier: for high error rates in the base classifiers, we can only expect improvements by a constant factor, whereas for the practical case of low error rates we can also expect a significant reduction in the asymptotic algorithmic complexity.

Overall Complexity

Besides the complexities for the comparisons, the overall complexity of the algorithm including the inherent overhead of the algorithm, e.g., estimating the next classifier $f_{a,b}$ and so on, can be stated as $g(k) \cdot (k+p)$ operations, where $g(k)$ denotes the number of comparisons in dependence of k and p describes the cost of one comparison (prediction) in terms of basic operations. The summand k is here understood as the operations for



(a) full range of advantageous k **(b)** focus on $k = 1..1000$

Figure 6.2: Approximate computational savings for $p = 20$ and assumed average complexity of $k \cdot \log k$ of QWeighted

the overhead involving additions, value associations and argmin operations, which can be implemented in an incremental manner with $O(k)$ by maintaining a sorted vector of classes (for the limits l_i) updated after each comparison and using a 2-dimensional boolean array which maintains the status of the classifiers (evaluated/not evaluated).

Given the empirical evidences, lets assume $g(k) = k \cdot \log k$. This results in a total of $k^2 \cdot \log k + p \cdot k \cdot \log k$ operations. Obviously, the overall asymptotic complexity of QWeighted is worse than the complexity for standard voting of $O(k^2)$, such that for very large k the complexity of the standard voting is favored. But, in practice for reasonable assumptions, e.g $p \gg 1$ (keep also in mind that p can increase for some learning schemes, e.g., in dependence of the number of training instances), there exist an upper limit $\hat{k} \in \mathbb{N}$ such that $k^2 \cdot \log k + p \cdot k \cdot \log k$ is significantly smaller than $p \cdot \frac{k \cdot (k-1)}{2}$ for $k < \hat{k}$.

To give a clearer picture, consider Figure 6.2, where the difference of both quantities, i.e. $p \cdot \frac{k \cdot (k-1)}{2} - (k \cdot \log k \cdot (k + p))$ is plotted for $p = 20$ (20 atomar operations needed for one prediction). The graph shows the saved number of operations by using QWeighted in contrast to standard voting procedure in dependence of k. The left figure shows the savings up to the critical class count $\hat{k} \approx 22,000$ and the right figure shows the same plot for the selected range $k = [1, 1000]$, which corresponds approximately to the typical range in real-world datasets.

6.2.2 ECOC Classification - Evaluation of QuickECOC

In this section, we evaluate the performance of QuickECOC for a variety of different codes. In addition, we were interested to see if it works for all decoding methods and whether we can gain insights on which factors determine the performance of QuickECOC. In particular, we investigated the effects of the sparsity and length of the codes.

Experimental Setup

In contrast to the results presented in the previous section, we only used the decision tree learner J48 with default parameters as a base learner to restrict the already large number of the experiments. Besides, the results in Section 6.2.1 on page 63 gave no indication that the performance in terms of the number of needed comparisons depends on the choice of the base classifier. Thus, we are quite confident that the presented results are representative for other base classifiers.

Our setup consisted of

- **5 encoding strategies**: BCH Codes and two versions each of exhaustive and random codes.
- **7 decoding methods**: Hamming, Euclidean, attenuated Euclidean, linear loss-based, exponential loss-based, Laplacian Strategy and Beta Density Probabilistic Pessimistic
- **7 multiclass datasets** selected from the UCI Machine Learning Repository.

For the encoding strategies, we also tried several different parameters. Regarding the exhaustive codes, we evaluated all (k, l) codes ranging from $l = 2$ to $l = k$ per dataset and analogously for the cumulative version. For the generation of the first type of random codes the zero symbol probability was parametrized by $r_{zp} = 0.2, 0.4, 0.6, 0.8$ and the dimension of the coding matrix was fixed to 50 % of the maximum possible dimension with respect to the number of classes. The second type of random codes was generated by randomly selecting 20 %, 40 %, 60 % and 80 % from the set of all valid classifiers respectively columns (all columns of an (k, k) cumulative ternary coding matrix) without repetition. Regarding BCH Codes, we generated $7, 15, 31, 63, 127$ and 255-bit BCH codes and randomly selected n rows matching the class count of the currently evaluated dataset. For the datasets *machine* and *ecoli* where the number of classes is greater than 7, we excluded the evaluation with 7-bit BCH codes.

For the evaluation of QuickECOC, the seven datasets were selected to have a rather low number of different classes. The main reason for this limitation was that for some considered code types the number of classifiers grows exponentially. Especially for the datasets with the maximum number of eight classes (*machine* and *ecoli*), the cumulative ternary exhaustive codes generates up to 3025 classifiers. In addition, we evaluated all possible combinations of decoding methods, code types with various parameters, which we can not present here completely (in total 1246 experiments) because of lack of space. Nevertheless, we performed experiments with a few of more efficient codes on datasets with a larger number of classes as well. These will be shown in Section 6.2.2 on page 75.

Because of the high number of experiments, we cannot present all results in detail, but will try to focus on the most interesting aspects. In addition to assess the general performance of QuickECOC, we will analyze the influence of the sparsity of the code matrix, of the code length, and of different decoding strategies.

Table 6.4: QuickECOC performance using Hamming Decoding and Exhaustive Ternary Codes. The maximal relative standard deviation for all values is 8.65 % with mean 3.88 %.

l	*vehicle*	*derm.*	*auto*	*glass*	*zoo*	*ecoli*	*machine*
2	3.82 *63.7*	7.12 *47.5*	7.95 *37.9*	9.99 *47.6*	9.48 *45.1*	11.75 *42.0*	11.60 *41.4*
3	7.91 *65.9*	26.05 *43.4*	42.86 *40.8*	43.47 *41.4*	41.64 *39.7*	58.85 *35.0*	57.90 *34.5*
4	5.65 *80.8*	46.30 *44.1*	115.22 *47.0*	116.45 *47.5*	107.03 *43.7*	199.31 *40.7*	194.81 *39.8*
5		43.11 *47.9*	163.67 *52.0*	163.98 *52.1*	148.50 *47.1*	369.06 *43.9*	355.23 *42.3*
6		16.54 *53.4*	114.87 *52.9*	116.77 *53.8*	102.41 *47.2*	394.25 *45.4*	369.19 *42.5*
7			34.24 *54.3*	37.84 *60.1*	31.52 *50.0*	234.80 *46.6*	218.09 *43.3*
8						62.17 *49.0*	57.27 *45.1*

Reduction in Number of Evaluations

Table 6.4 shows the reduction in the number of classifier evaluations with QuickECOC on all evaluated datasets with Hamming decoding and ternary exhaustive codes. In every column, the average number of classifier evaluations is stated with its corresponding ratio to the number of generated classifiers in italics (the lower the better). The datasets are ordered from left to right by ascending class-count. As the level parameter l is bounded by the class-count k, some of the cells are empty.

One can clearly see that QuickECOC is able to reduce the number of classifier evaluations for all datasets. The percentage of needed evaluations ranges from about 81 % (*vehicle*, $l = 4$) to only 35 % (*machine*, $l = 3$). At first glance, these improvements may not seem striking, because a saving of a little less than 40 % for the small datasets does not appear to be such a large gain. However, one must put these results in perspective. For example, for the *vehicle* dataset with a $(4, 3)$-exhaustive code, QuickECOC evaluated 65.9 % of all classifiers. A $(4, 3)$-exhaustive code has 12 classifiers, and each individual class is involved in 75 % of these classifiers (cf. the example in Section 3.4.3 on page 24). Thus, on average, QuickECOC did not even evaluate all the classifiers that involve the winning class before this class was predicted.

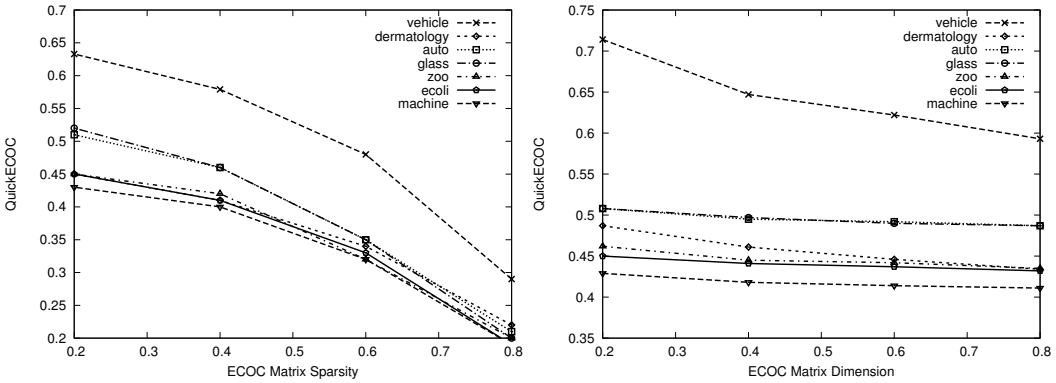
Furthermore, one can observe a general trend of higher reduction by increasing class-count. This is particularly obvious if we compare the reduction on the exhaustive codes (the last line of each column, where $l = k$), but can also be observed for individual code sizes (e.g., for $l = 3$). Although we have not performed a full evaluation on datasets with a larger amount of classes because of the exponential growth in the number of classifiers, a few informal and quick tests supported the trend: the higher the class-count, the higher the reduction.

Another interesting observation is that except for dataset *vehicle* and *auto* the exhaustive ternary codes for level $l = 3$ consistently lead to the best QuickECOC performance over all datasets. We will provide later on a possible explanation based on a "combinatorial trade-off" in Section 6.3 on page 78.

The results for BCH codes are shown in Table 6.5 on the next page. Again, we can observe an improved performance in all cases. This result is particularly interesting

Table 6.5: QuickECOC performance on BCH Codes. The maximal relative standard deviation for all values is 10.2 % with mean 5.64 %.

	vehicle	*derm.*	*auto*	*glass*	*zoo*	*ecoli*	*machine*
7	0.764	0.774	0.851	0.880	0.834	-	-
15	0.646	0.656	0.699	0.717	0.659	0.670	0.648
31	0.571	0.564	0.607	0.662	0.581	0.602	0.558
63	0.519	0.506	0.567	0.616	0.517	0.540	0.509
127	0.489	0.447	0.522	0.565	0.477	0.493	0.459
255	0.410	0.380	0.450	0.467	0.397	0.417	0.388



(a) in dependence of Sparsity **(b)** in dependence of Code Length

Figure 6.3: QuickECOC performance on random codes

because for BCH-codes, all coding matrices are dense, i.e., they do not have any zero entries. Even in this case, we see that there was no situation, where all classifiers were needed for multiclass classification. And again, we observe that for higher dimensions (increasing the length of the BCH bit code) higher reductions can be observed.

For random codes, we obtained qualitatively the same results. We do not show them here, but some of them will appear in the following sections.

Sparsity of Coding Matrices

We define the sparsity of the ECOC matrix as the fraction of zero values it contains. Random codes provide a direct control over the matrix sparsity (as described in Section 3.4.3 on page 25), and are thus suitable for analyzing the influence of the sparsity degree of the ECOC matrix for QuickECOC. Note, however, that the observed influences regarding sparsity and dimension of the matrix on the QuickECOC performance can also be seen in the evaluations of the other code types, but not as clearly as with the random codes presented in this section.

Figure 6.3a shows QuickECOC applied to random codes with varying matrix sparsity. A clear trend can be observed that the higher the sparsity of the coding matrix the better the reduction for all datasets. Keep in mind that the baseline

performance (evaluating all binary classifiers) is a parallel to the x-axis with the y-value of 1.0. Note that the absolute reduction tends to be minimal over all considered datasets at datasets with higher class-counts i.e *machine* at 80 % sparsity, and the lowest reduction can be seen for the dataset *vehicle* with the smallest number of classes $k = 4$ at 20 % sparsity.

The main effect of an increase of sparsity on the coding matrices is that for each class the number of incident classifiers decreases. For sparsity 0, all classes are involved in all classifiers, for sparsity 0.5, each class is (on average) involved in only half of the classifiers. This will clearly affect the performance of the QuickECOC algorithm. In particular, the second stopping criterion essentially specifies that the true class is found if all incident classifiers for the favorite class c_{i_0} have been evaluated. Clearly, the algorithm will terminate faster for higher sparsity levels (ignoring, for the moment, the possibility that the first stopping criterion may lead to even faster termination).

Code Length

The second type of random codes, which were generated by randomly selecting a fixed number from the set of all possible binary classifiers can be seen in Figure 6.3b on the previous page. All coding matrices for a k-class dataset have nearly the same sparsity, which relates to the average sparsity of (k, k) cumulative exhaustive codes and differ only in the length of the coding matrix (in percent of the total number of possible binary classifiers). This allows us to observe the effect of different numbers of classifiers on the QuickECOC performance. Here, we can also see a consistent relationship, that higher dimensions lead to better performance, but the differences are not as remarkable as for sparse matrices.

For a possible explanation, assume a coding matrix with fixed sparsity and we vary the dimension. For a higher dimension the ratio of number of classifiers per class increases. Thus, on average, the number of incident classifiers for each class also increases. If we now assume that this increase is uniform for all classes, this has the effect that the distance vector $\vec{l}$ is multiplied by a positive factor $x > 1$, i.e., $\vec{l^+} = \vec{l} \cdot x$. This alone would not change the QuickECOC performance, but if we consider that classifiers are not always perfect, we can expect that for a higher number of classifiers, the variance of the overall prediction will be smaller. This smaller variance will lead to more reliable voting vectors, which can, in turn, lead to earlier stopping. It also seems reasonable that this effect will not have such a strong impact as the sparsity of the coding matrix, which we discussed in the previous section.

Different Decoding Strategies

As previously stated, because of the large number of experiments, we can not give a complete account of all results. We evaluated all combinations of experiments, that includes also all mentioned decoding methods. All the previously shown results were based on Hamming decoding, since it is still one of the commonly used decoding strategies even for ternary ECOC matrices. However, we emphasize, that all obser-

Table 6.6: QuickECOC performance on the 8-class *ecoli* dataset with all decoding methods and Cumulative Exhaustive Ternary Codes. The first two columns show the number of non-zero code values for each class and the number of resulting classifiers. The maximal relative standard deviation for all values is 3.76 % with mean 2.69 %.

l	$\lvert C\rvert$	Hamming	Euclidean	A. Euclidean	LBL	LBE	Laplace	BDDP
2	28	*0.420*	*0.420*	*0.420*	*0.399*	*0.398*	*0.406*	*0.426*
3	196	*0.331*	*0.331*	*0.331*	*0.335*	*0.350*	*0.332*	*0.333*
4	686	*0.377*	*0.377*	*0.377*	*0.383*	*0.402*	*0.374*	*0.375*
5	1526	*0.400*	*0.400*	*0.400*	*0.414*	*0.439*	*0.399*	*0.401*
6	2394	*0.421*	*0.421*	*0.421*	*0.437*	*0.466*	*0.419*	*0.418*
7	2898	*0.427*	*0.427*	*0.427*	*0.444*	*0.475*	*0.426*	*0.425*
8	3025	*0.428*	*0.428*	*0.428*	*0.446*	*0.477*	*0.427*	*0.426*

vations on this small subset of results can also be found in the experiments on the other decoding strategies. As an exemplary data point, Table 6.6 shows an overview of the QuickECOC performance for all decoding strategies for the dataset *ecoli* using cumulative exhaustive ternary codes. It can be seen that the performance is quite comparable on all datasets. Even the optimal reduction for $l = 3$ can be found in the results of all decoding strategies.

Datasets with a Large Number of Classes

The previous sections evaluated and analyzed QuickECOC on a broad spectrum of various code types and decoding methods. This was only feasible for datasets with a smaller number of classes. In this section, we will evaluate QuickECOC on datasets with a larger number of classes. As the code length of most coding strategies is exponential in the number of classes k, we selected a few codes which generate a comparably low number of classifiers:

1. $(k, 3)$- and $(k, 4)$-exhaustive ternary codes
2. $(k, 4)$-cumulative exhaustive ternary codes
3. random codes of type 1 with fixed sparsity of 66 %
4. random codes of type 2

For both random code types the code length was set to the equivalent of the number of $(k, 4)$-cumulative exhaustive codes, i.e. $n = \sum_{i=2}^{4} \binom{k}{i} \cdot (2^{i-1} - 1)$.

Table 6.7 on the following page shows the results for Hamming decoding. Each cell shows the average number of classifier evaluations of QuickECOC and, in italics, its corresponding ratio to the full number of classifiers. First, we can observe a considerably higher improvement than with the results on the datasets with lower number of classes. The best reduction can be found for the $(19, 3)$-exhaustive code for

Table 6.7: QuickECOC performance on datasets with high number of classes. The maximal relative standard deviation for all values is 2.37 % with mean 1.65 %.

	k	exh. $l=3$		exh. $l=4$		cum. exh. $l=4$		random1		random2	
yeast	10	105.92	*0.294*	524.13	*0.357*	631.02	*0.337*	844.02	*0.296*	1351.94	*0.474*
vowel	11	139.42	*0.282*	797.32	*0.345*	937.43	*0.328*	880.89	*0.306*	1388.74	*0.482*
soybean	19	443.89	*0.153*	5351.90	*0.197*	5804.73	*0.192*	8481.74	*0.282*	12964.83	*0.431*

the soybean dataset, where QuickECOC only performs about 15 % of the evaluation in order to determine the winning class.

Moreover, one can clearly see an increasing reduction for increasing number of classes k, especially for the first three columns respectively code types. For these code types, the sparsity increases with k, since the number of non-zero values per column stays fixed whereas the number of rows (the number of classes k) of the corresponding ECOC matrix is increased. This observation confirms our results of Section 6.2.2 on page 73, which showed that a high sparsity is beneficial for the performance of QuickECOC.

We can also confirm our results regarding the influence of the code length. Both types of random codes, shown in the last two columns, have a fixed sparsity level. In both cases, although a small improvement can be observed (just as in Figure 6.3b on page 73), the improvement is small in comparison to the improvement resulting from increased sparseness. For example, on datasets *yeast* ($k = 10$) and *soybean* ($k = 19$), QuickECOC applies for the second type of random codes in average 47 % and 43 % classifier evaluations, which is a relative reduction/ratio of about 10 %, whereas for $(k, 3)$-exhaustive codes a relative reduction of $0.153/0.294 \approx 48\,\%$ is gained.

Simulation Experiment

We also conducted a simulation experiment for QuickECOC, similar in spirit to the pairwise case (Section 6.2.1 on page 66). Again, we consider classes $1 \dots k$ and always assume that class 1 is the correct class and further assume that pseudo base classifiers f_i^ϵ return the desired prediction with probability ϵ, i.e., with probability ϵ, they predict the same sign in the ECOC matrix as the smallest incident class of f_i^ϵ. We simulate the efficiency of QuickECOC using exhaustive ternary codes of level 3 for various class counts k and error probabilities ϵ. Table 6.8 on the facing page shows the results.

In contrast to pairwise classification, we can observe that the base-classifier accuracy now has a stronger influence on the efficiency. In the case of random classifiers $\epsilon = 0.5$, we can observe almost no reduction, as was the case for pairwise classification (though for greater k, it might converge to the worst-case there too). But, for $\epsilon < 0.5$, focusing on the ratio values, one can see an increasing reduction trend for increasing k, which slowly loses its steepness. The growth values suggest that only in the near optimal case $\epsilon < 0.05$, a super-linear reduction with the number of classes can be expected.

Table 6.8: Average number $\hat{n}$ of comparisons for various number of classes and different error probabilities ϵ of ECOC classifiers using QuickECOC with exhaustive ternary codes of level 3, and for the full ensemble of classifiers. Below, we show the ratio to the full number of comparisons and an estimate of the growth ratio $\frac{\log(\hat{n}_2/\hat{n}_1)}{\log(k_2/k_1)}$ of successive values of $\hat{n}$.

	$k = 5$	$k = 10$	$k = 25$	$k = 50$	$k = 100$
$\epsilon = 0.0$	13.00	93.00	783.00	3,433.00	14,358.00
	0.433 —	*0.258* **2.839**	*0.113* **2.325**	*0.058* **2.132**	*0.030* **2.064**
$\epsilon = 0.05$	14.31	101.18	951.28	7,432.19	61,244.02
	0.477 —	*0.281* **2.822**	*0.138* **2.446**	*0.126* **2.966**	*0.126* **3.043**
$\epsilon = 0.1$	15.95	114.85	1,734.24	14,588.86	119,232.96
	0.532 —	*0.319* **2.848**	*0.251* **2.963**	*0.248* **3.072**	*0.246* **3.031**
$\epsilon = 0.2$	19.78	177.30	3,296.39	27,589.68	223,472.28
	0.659 —	*0.492* **3.164**	*0.478* **3.190**	*0.469* **3.065**	*0.461* **3.018**
$\epsilon = 0.3$	24.62	249.87	4,733.95	39,396.05	319,798.78
	0.821 —	*0.694* **3.343**	*0.686* **3.210**	*0.670* **3.057**	*0.659* **3.021**
$\epsilon = 0.4$	27.71	309.34	5,993.48	50,121.35	407,887.14
	0.924 —	*0.859* **3.481**	*0.869* **3.235**	*0.852* **3.064**	*0.841* **3.025**
$\epsilon = 0.5$	29.08	336.05	6,635.10	57,502.71	478,871.55
	0.969 —	*0.933* **3.530**	*0.962* **3.255**	*0.978* **3.115**	*0.987* **3.058**
full	30	360	6900	58,800	485,100

However, in absolute terms, the reduction can be significant for predictors with high computational complexity. Furthermore, this analysis was based on one of many applicable code types which can be used with ECOC. Other code types, e.g., codes with beneficial error-correcting ability or codes which may not grow exponentially in k like the considered exhaustive ternary code, may perform differently.

Overall Complexity

Since the QuickECOC algorithm is more general than QWeighted, its overhead is significantly greater. The stopping criteria depicted in Algorithm 6 on page 59 can be implemented in an incremental manner such that the complexity is $O(k)$, by maintaining for each class a variable storing the potential worst-case Hamming distance and updating only the relevant values after each comparison (prediction). All in all, the overhead is linear in k except for the *Next Classifier Selection* Scheme (cf. Section 6.1.2 on page 57), the complexity of which is $O(nk)$. Therefore, the computational savings diminishes in this case far more quicker as in the case of QWeighted for pairwise classification.

A reduction of operations is still possible for problems up to about $k = 10$ and depending of the actual prediction complexity and code type. But for greater class counts the overhead starts to dominate the overall complexity such that the efficiency

is worse than standard voting. In these cases, a reasonable choice is to work with alternative selection schemes, which check only a fixed number of classifiers incident of the current best class. Or selecting an unevaluated classifier randomly from the set of incident classifiers to the current best class is an alternative, with a slight decrease in comparisons efficiency but important increase in overall efficiency of the algorithm. Note, this passage holds for code types which grow exponentially in k, e.g., exhaustive ternary codes. In cases of more practicable code types, such as BCH codes, the overhead of the algorithm remains still in the tolerable range.

6.3 Analysis of (k,3)-Exhaustive Ternary Codes

Since we are interested in conditions under which QuickECOC performs well (cf. Section 6.2.2 on page 70), this special case of exhaustive ternary codes was further investigated. In this regard, we examined the reduction effects of the two stopping criteria separately with varying levels on several datasets.

It is easy to see that the performance regarding the second stopping criterion is strongly dependent on the incidency of the ECOC matrix. Considering that the selection process of QuickECOC always selects incident classifiers of the current best class c_0, the number of classifier evaluations can be estimated as $|I_0| + |R_0|$ whereas I_i is the set of incident and $R_i \subseteq C \setminus I_i$ is a subset of non-incident classifiers of c_i.[5] These remaining classifiers R can be caused by initial random pairings until a classifier involving the true class c_0 is evaluated. Even then, depending on the accuracy of the classifiers, still some non-incident classifiers can be falsely selected and evaluated in the next steps. In this context, the second stopping criterion can be seen as a reduction method which tries to minimize the non-incident classifier evaluations. Considering that for increasing level l the incidency of exhaustive ternary codes is constantly increasing, the reduction performance of the second criterion alone is decreasing percentagewise.

On the other hand, the first stopping criterion also tries to reduce the number of incident classifier evaluations. But this comes with a price: in general more evaluations of classes different from c_0 have to be evaluated to enable an earlier cut. But this case comes naturally by increasing the level, since each classifier involves an increasing number of classes, so that already with fewer evaluations, a reasonable amount of votes have been distributed. So in short, by increasing the level l, which increases the incidency, the reduction performance of QuickECOC is more and more due to the first stopping criterion whereas the impact of the second criterion decreases.

These considerations can be confirmed in Figures 6.4 and 6.5. Figure 6.4 on the facing page shows the performances of QuickECOC with both stopping criteria, without the first criterion, and the incidency of the ECOC matrix for a given exhaustive ternary code level using the example of the dataset *ecoli* ($k = 8$). The differences between the two QuickECOC variants (dashed and dotted curves) depict the

5 Actually, for exhaustive ternary codes, it holds $|I_i| = |I_j|$ and $|R_i| = |R_j|$ for arbitrary $c_i, c_j \in K$ and fixed level l. Thus, $|I|$ and $|R|$ are in this case only dependent on l and k.

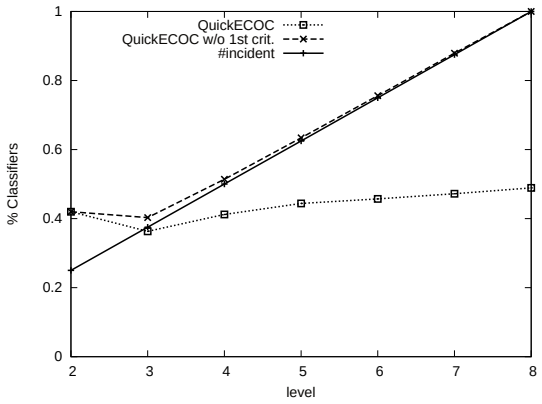
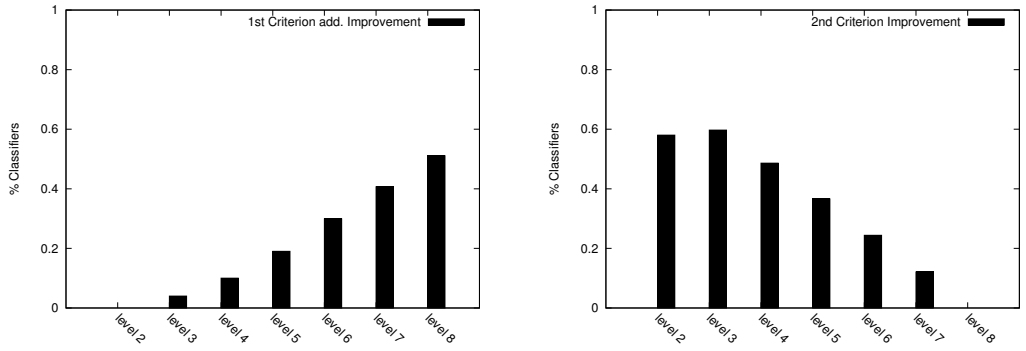


Figure 6.4: Dependency of stopping criteria for different levels respectively incidencies (*ecoli*)



(a) Additional First Criterion Improvement **(b)** Second Criterion Improvement

Figure 6.5: Impact of stopping criteria for reduction under varying level of exhaustive ternary codes (*ecoli*)

additional improvement caused by the first stopping criterion, which can be seen also separately in Figure 6.5a. In line with the above considerations, one can see that the performance of QuickECOC without the first criterion is similar to the number of incident classifiers for a given level, it almost converges to it. Thus, the amount of non-incident classifiers R seems to decrease.

We can observe that the first criterion begins to reduce the evaluations at $l = 3$ and the gain increases with increasing level (see also Figure 6.5a). This additional improvement for $l = 3$ is nevertheless not the only reason for the best performance, since the improvement is too small. However, observing the figure, the right question seems rather why QuickECOC does perform so much worse for $l = 2$ rather than why $l = 3$ yields the best performance. For the sake of simplicity, it seems sufficient to consider only the second stopping criterion for this matter in the following.

For this case we describe a model which approximates the observed effect and therefore could yield a possible explanation. Assume that all classes form a linear

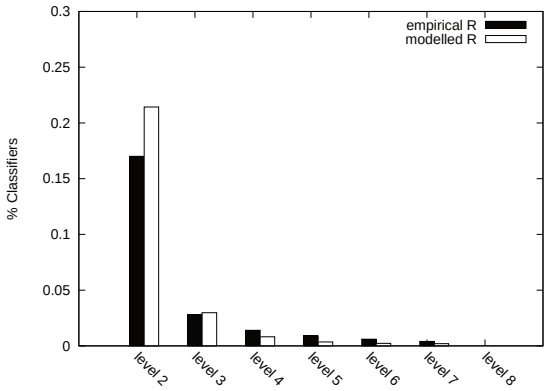


Figure 6.6: Comparison of simplified QuickECOC versus actual performance ignoring the first stopping criterion (*ecoli*)

order with respect to their votes, i.e. $v(c_1) > v(c_2) > \cdots > v(c_k)$ and that every classifier f returns a prediction in favor to the class with the highest (true) votes among the incident classes, i.e. the evaluation of f votes for the classes c_i, whose signs equal the one of class $c^* = \text{argmax}_{c_i \in IN(f)} v(c_i)$ and $IN(f)$ represents the set of incident classes of f. In addition, we reduce QuickECOC to a simple method which follows the only rule: Pick as the next classifier a remaining one which involves the classes with the lowest count of lost games. Now, we consider the worst case of classifier evaluation sequences, the maximal number of evaluations, until the true best class c_0 has been evaluated once as an estimate for R. It turns out that this count is $k-l$ for a given level l of exhaustive ternary codes. Then, because of our assumptions, the following holds: if a classifier involving the true class c_0 is evaluated, all remaining classifiers will be an incident classifier of c_0. So, the worst case complexity of this setting is $k-l+|I_0(l)|$. As stated before, the cardinality of I_0 is given by l and k, more precisely $|I_0(l)| = |I(l)| = \frac{l}{k} n(k,l)$. This simplified model provides a surprisingly close fit to the empirical values, as one can see in Figure 6.6.

One can show that

$$2 = \underset{l \in \{2 \ldots k\}}{\text{argmax}} \frac{k-l}{n(k,l)} = \underset{l \in \{2 \ldots k\}}{\text{argmax}} \frac{k-l}{\binom{k}{l}(2^{l-1}-1)}$$

and in particular

$$\frac{k-2+|I_0(2)|}{n(k,2)} > \frac{k-3+|I_0(3)|}{n(k,3)} < \frac{k-l+|I_0(l)|}{n(k,l)}$$

where $k \geq l > 3$ for $k > 4$.

So, the ratio of non-incident classifiers R has yet a significantly strong influence on the reduction for $l = 2$, in fact, it is the only case where R exceeds the constant increase of I_0 for the next level $l = 3$, i.e. $\frac{k-2}{n(k,2)} > \Delta \frac{|I_0|}{n} = \frac{1}{k}$. This yields a minimum of $(|R| + |I|)/n$ for $l = 3$, since R has an exponential decay, thus its influence for

the overall reduction diminishes very fast (cf. Figure 6.6 on the preceding page or the differences of solid and dashed curves in Figure 6.4 on page 79), whereas $|I|$ is constantly increasing. Here, the quantity of non-incident classifiers R was explained as the possible maximum amount of classifier evaluations avoiding the class c_0.

6.4 Conclusions

In this chapter, we have presented an algorithm that allows to speed up the prediction phase for binary decomposition methods such as pairwise classification and, more generally, ternary ECOC classifiers. Both variants only need to evaluate a fraction of the classifiers, but are guaranteed to make the same prediction as the original version using all classifiers. In general, this gain increases with the complexity of the problem, i.e., with the number of classes, with the sparsity of the coding matrix, and (somewhat less) with the length of the code words of the ECOC classifiers. But even for very hard problems, where the performance of the binary classifiers reduces to random guessing, practical gains can be expected.

For the general case of ternary ECOC matrices, which subsume nearly all possible binary decomposition schemes, we have demonstrated this gain for a wide variety of coding and decoding strategies. Regardless of the used code, QuickECOC improves the overall prediction efficiency, but, depending on the coding strategy, the amount of improvement is not always as striking as for the pairwise case, where we could observe a reduction from k^2 to $k \cdot \log k$. One must keep in mind that in ECOC codings, each class has a much larger number of incident classifiers, and thus a higher number of evaluations must be expected to determine the winning class. Moreover, for code types whose code length grows exponentially with the number of classes, the overhead of QuickECOC (in its presented form) can dominate the gained reduction of classifier evaluations, thus resulting in a worse performance than standard voting. However, we briefly described alternative more overhead-efficient approaches which allow to adjust QuickECOC to the problem at hand such that a beneficial reduction can still be expected. In general, we recommend the practitioner to carefully pre-assess the parameters of the present problem, such as the number of classes k, code type and prediction complexity in terms of base operations, integrate them into the overall complexity model and to adjust the selection scheme to maximize the efficiency performance.

One could argue that typically the training phase is more expensive than the classification phase, and that the gains obtained by QuickECOC are negligible in comparison to what can be gained by more efficient coding techniques. While this is true, we note that QuickECOC can obtain gains independent of the used coding technique, and can thus be combined with any coding technique. In particular in time-critical applications, where classifiers are trained once in batch and then need to classify on-line on a stream of in-coming examples, the obtained savings can be decisive.

Another point to consider is that in applications where the classification time is crucial, a parallel approach could be applied effectively because each classifier defined by a column of the ECOC matrix can be evaluated independently. QuickECOC loses this advantage because the choice of the next classifier to evaluate depends on the results of the previous evaluations. However, QuickECOC can still be parallelized on the instance level instead of the classifier level. Given n processors or n threads we want to utilize, we select n incoming test instances and apply QuickECOC for each of them. Basically by paralleling the decoding process on the instance level, we avoid the problem that QuickECOC can not be directly parallelized on the classifier level for one instance. This method is still very efficient, since every CPU is constantly utilized. Considering that in total, the number of evaluations is decreased by using QuickECOC, a higher speed up can be expected as with a straight-forward parallelization of ECOC.

Recently, Hsu et al. (2009b) presented an efficient ensemble approach for multilabel classification. They exploit the general label sparseness in target vectors of real-world multilabel problems to reduce the number of the labels to $O(\log k)$ using techniques from compressed sensing. Instead of learning k one-against-all regression predictors for generating a multiclass predictor, they only need to learn (and therefore to predict) about $\log k$ regression predictors. Since multiclass classification is a special case of multilabel classification (by limiting the label size to 1) and noting that it poses a maximal label-sparse multilabel problem, their results naturally also apply to multiclass classification. However, it is not entirely clear how their ensemble approach consisting of a one-against-all decomposition with regression base-learners performs in comparison to the commonly studied classification-based ensemble approaches with respect to predictive performance. Moreover, input or output data transformations yield often to a reduction of the comprehensibility of the learned models, which is disadvantageous for the acceptance in real-world applications, where a white-box property of the system is favored. A direct comparison of this work to conventional classification-based decompositions is certainly interesting and overdue, but beyond the scope of this work, where our goal was to improve the classification time of well established ensemble techniques.

Besides pairwise classification and ECOCs, a variety of other decomposition-based approaches have been proposed for the multiclass classification task. It was not the goal of this chapter to contribute to the discussion of their respective virtues—for a recent survey on this subject we refer to (Lorena et al., 2008). Our contribution to this on-going debate was to solve one of the most severe problems with two of the most popular decomposition methods, namely by improving their classification efficiency without changing their predictive quality.

Part II

Multilabel Classification

7 Multilabel Preliminaries

Contents

Multilabel classification refers to the task of learning a function that maps instances $x \in X$ to label subsets $P_x \subseteq L$, where $L = \{\lambda_1, \ldots, \lambda_k\}$ 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 and the EUR-Lex database consists of almost 4000 classes. Other tasks include protein classification and semantic multimedia annotation.[1]

7.1 Multilabel Setting

The multilabel classification setting is similar to multiclass classification with the exception, that each instance can be associated with multiple classes instead of exactly one class. We often speak of *labels* instead of classes in this context. In addition, we will use a different notation for the labels and the set of labels to allow an easy distinction from the multiclass setting.

1 A collection of datasets can be found at http://mlkd.csd.auth.gr/multilabel.html

To be more precise, each instance x_i is associated with a set of *relevant* labels $P_i \subseteq L$, a subset of a given set of k possible classes/labels $L = \{\lambda_1, \ldots, \lambda_k\}$. The remaining labels are regarded as *irrelevant* and are denoted as $N_i = L \backslash P_i$. For multilabel problems, the cardinality $|P_i|$ of these labelsets is not restricted, whereas for multiclass problems it is exactly one. Multilabel learning algorithms are trained on a training set $D = \{(x_i, P_i) \mid i = 1 \ldots t\}$.

Note that we will use two types of indexing conventions for P and N: P_x is the true labelset for instance x and P_i is meant to be the true labelset for instance x_i.

The task of multilabel classification is to find a function $f : X \to 2^L$, which takes as input an instance x and returns a set of labels $\hat{P}_x \subseteq L$ as output. This function f should typically minimize the empirical risk for some *labelset* loss function $l : 2^L \times 2^L \to \mathbb{R}_0^+$

$$\sum_{x_i \in X} l\big(f(x_i),\, P_i\big) = \sum_{x_i \in X} l\big(\hat{P}_i,\, P_i\big)$$

7.2 Multilabel Evaluation

There is no generally accepted single measure for evaluating multilabel classifications. Thus, evaluations consider typically various measures. Furthermore, if the task is divided into a label ranking problem (which returns a sorted ranking of labels from most relevant to most irrelevant) in combination with a thresholding function (which defines the splitting point, between relevant and irrelevant labels) to compute the labelset prediction $\hat{P}$, additional *ranking*-based measures are often also considered. In the following, we will give a brief recapitulation of common *labelset* and *ranking-based* measures.

7.2.1 Labelset Loss Functions

Probably the most direct approach for evaluation 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 $\hat{P}_x$ denote the set of labels predicted by the multilabel classifier and $\hat{N}_x = L \setminus \hat{P}_x$ the set of labels that are not predicted by the classifier. Thus, we can, for each individual instance x, compute a two-by-two confusion matrix CM_x of relevant/irrelevant vs. predicted/not predicted labels:

CM_x	predicted	not predicted	
relevant	$\lvert P_x \cap \hat{P}_x \rvert$	$\lvert P_x \cap \hat{N}_x \rvert$	$\lvert P_x \rvert$
irrelevant	$\lvert N_x \cap \hat{P}_x \rvert$	$\lvert N_x \cap \hat{N}_x \rvert$	$\lvert N_x \rvert$
	$\lvert \hat{P}_x \rvert$	$\lvert \hat{N}_x \rvert$	$\lvert L \rvert = k$

From such a confusion matrix CM_x, we can compute several well-known measures:

Hamming Loss (HammLoss)

The Hamming loss 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.

$$\textsc{HammLoss}\left(CM_x\right) := \frac{1}{k}\left|\hat{P}_x \triangle P_x\right| \tag{7.1}$$

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

Precision (Prec)

Precision computes the percentage of predicted labels that are relevant.

$$\textsc{Prec}\left(CM_x\right) := \frac{|\hat{P}_x \cap P_x|}{|\hat{P}_x|} \tag{7.2}$$

Recall (Rec)

Recall computes the percentage of relevant labels that are predicted.

$$\textsc{Rec}\left(CM_x\right) := \frac{|\hat{P}_x \cap P_x|}{|P_x|} \tag{7.3}$$

F1-Measure (F1)

The measures precision and recall considered in isolation have some shortcomings. Precision considers only which fraction of the predicted labels is actually relevant and ignores the misprediction of the remaining true labels. So, the "false negative" prediction of labels, i.e. predicting a true relevant label as irrelevant, does not contribute to this measure. Recall, on the other hand, takes this into account, but ignores the "false positive" ones, i.e. predicting an irrelevant label as relevant (note that a simple perfect recall predictor is to predict always all labels). Thus, a combination of both measures is commonly used.

The F1-measure is the harmonic mean between precision and recall. For the case of a zero denominator, a common convention is to define the result as zero.

$$\mathrm{F1}\left(CM_x\right) := \frac{2}{\frac{1}{\textsc{Rec}(CM_x)} + \frac{1}{\textsc{Prec}(CM_x)}} = \frac{2\,\textsc{Rec}\left(CM_x\right)\textsc{Prec}\left(CM_x\right)}{\textsc{Rec}\left(CM_x\right) + \textsc{Prec}\left(CM_x\right)} \tag{7.4}$$

7.2.2 Ranking-Based Loss Functions

Some multilabel methods decompose the learning problem into two components:

1. learning of a ranking function f, which returns for a given test instance x a sorted list of all labels from relevant to irrelevant

2. given such a ranking τ, learning of a thresholding function $g(\tau)$, which allows to separate relevant from irrelevant labels

It is possible to evaluate the ranking function f separately from g in a meaningful way. Although, there is typically no information of degree of relevance in the given (multilabel) data, it is clear, that the true relevant labels should be ranked very high. The accurate prediction of relevance degrees can not be evaluated and is also not necessary here, since arbitrary deviations from a supposed true relevance degree have no effect on the multilabel performance, as long as the irrelevant labels still receive a lesser value. This isolated evaluation (ignoring the thresholding function g) is helpful, because it allows a more detailed evaluation of the learning system at hand. Deviations from the ideal case are differently measured in the following measures and captivate therefore different aspects.

We use the following notational conventions: For a given instance x, let $\tau(\lambda_i)$ denote the position of λ_i in the predicted ranking and $\tau^{-1}(i)$ the label λ that is assigned to the position i, $i = 1 \dots k$. Moreover, $\tau(\lambda_i) < \tau(\lambda_j)$ indicates, that λ_i has a higher degree of relevance/is higher ranked than λ_j.

Average Precision (AvgPrec)

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.

$$\textsc{AvgPrec}\,(P_x, \tau) := \frac{1}{|P_x|} \sum_{\lambda \in P_x} \frac{|\{\lambda' \in P_x \mid \tau(\lambda') \leq \tau(\lambda)\}|}{\tau(\lambda)} \tag{7.5}$$

Ranking Loss (RankLoss)

The ranking loss computes the average fraction of pairs of labels which are not correctly ordered:

$$\textsc{RankLoss}\,(P_x, \tau) := \frac{|\{(\lambda, \lambda') \in P_x \times N_x \mid \tau(\lambda) > \tau(\lambda')\}|}{|P_x|\,|N_x|} \tag{7.6}$$

This measure is related to Kendall's rank correlation coefficient, which measures the correlation between two rankings.

An example computation of RankLoss and AvgPrec along with a visualization is shown in Figure 7.1 on the next page.

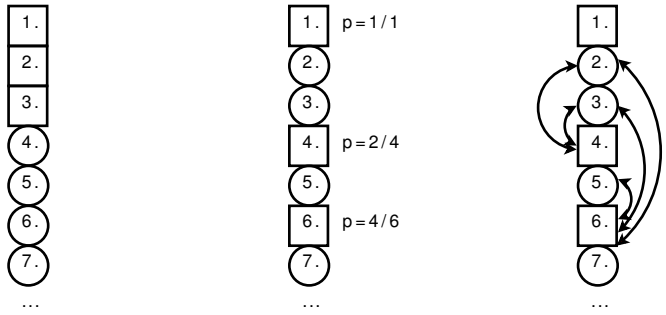


Figure 7.1: Diagrams of predicted label rankings (Loza Mencía, 2006). Rectangles denote true relevant labels and circles irrelevant labels. First ranking: perfect classification, all relevant labels are ranked over irrelevant ones, RankLoss = 0, AvgPrec = 1. Second and third ranking: labels on position 4 and 6 are misplaced, thus 5 of 12 possible pairs of labels are not correctly ordered, top label is correct, RankLoss = 5/12, AvgPrec = 2/3.

One-Error Loss (OneErr)

The one-error loss determines whether the top-ranked label is relevant or not, and ignores the relevancy of all other labels.

$$\textsc{OneErr}\left(P_x, \tau\right) := \begin{cases} 1 & \text{if } \tau^{-1}(1) \notin P_x, \\ 0 & \text{otherwise.} \end{cases} \tag{7.7}$$

Margin Loss (Margin)

The margin loss returns the normalized number of positions between the worst (lowest) ranked relevant and the best (highest) ranked irrelevant label. This is directly related to the number of wrongly ranked labels, i.e. the relevant labels that are ordered below a irrelevant label, or vice versa. We denote this set by F.

$$F := \left\{\lambda \in P_x \mid \exists \lambda' \in N_x . \tau(\lambda') < \tau(\lambda)\right\} \cup \left\{\lambda' \in N_x \mid \exists \lambda \in P_x . \tau(\lambda') < \tau(\lambda)\right\} \tag{7.8}$$

$$\textsc{Margin}\left(P_x, \tau\right) := \frac{\max\left(0, \max\left\{\tau(\lambda) \mid \lambda \in P_x\right\} - \min\left\{\tau(\lambda') \mid \lambda' \in N_x\right\}\right)}{k-1} \tag{7.9}$$

Ranking Error (RankErr)

If a true ranking τ^* is given, the ranking error is commonly used to evaluate the predicted ranking τ. It returns the normalized sum of squared position differences for each label in the predicted and true ranking. It is 0 for a ranking which is identical to the true ranking and 1 for a complete reversed ranking.

$$\textsc{RankErr}\left(\tau_x^*, \tau_x\right) := n(k) \sum_{\lambda \in L} \left|\tau^*(\lambda) - \tau(\lambda)\right|^2 \tag{7.10}$$

where n refers to the normalizing constant in dependence of the number of classes k. This error corresponds to the Spearman's rank correlation coefficient between two rankings.

7.2.3 Averaging Methods

These labelset and ranking-based measures are computed for each example x. For averaging a measure f over a set of instances (x_i, P_i), $i = 1 \dots t$ we have two principal options:

Macro-average is the average value of each measure over all examples

$$f_{\text{mac}} = \frac{1}{t} \sum_{i=1}^{t} f\left(CM_{x_i}\right) \tag{7.11}$$

Micro-average is the value of the measure computed over a confusion matrix that is the sum of all confusion matrices CM_x.

$$f_{\text{mic}} = f\left(\sum_{i=1}^{t} CM_{x_i}\right) \tag{7.12}$$

The difference between these two approaches is that macro-averaging gives equal weight to each instance, while micro-averaging gives equal weight to each relevant label, i.e., instances with more relevant labels have a larger influence on the average.

In this thesis, if not stated otherwise, we will perform micro-averaging over all measures. To combine the results of the individual folds of a cross-validation, we average the estimates f^j_{avg}, $j = 1 \dots q$ over all q folds.

7.3 Decomposition-Based Approaches

In the following, we will recapitulate common and recently proposed decomposition-based multilabel classification schemes.

7.3.1 Binary Relevance (BR)

In the binary relevance method, a multilabel training set with k possible classes is decomposed into k binary training sets that are then used to train k binary classifiers f_j. So for each pair (x_i,P_i) in the original training set k different pairs (x_i, λ_{i_j}) with $j = 1 \dots k$ are generated as follows:

$$\lambda_{i_j} = \begin{cases} +1 & \lambda_j \in P_i \\ -1 & otherwise \end{cases} \tag{7.13}$$

Note, that all of these k decomposed training sets are of the same size as the original training set. A brief visual description of this technique is available in Figure 7.2 on the facing page.

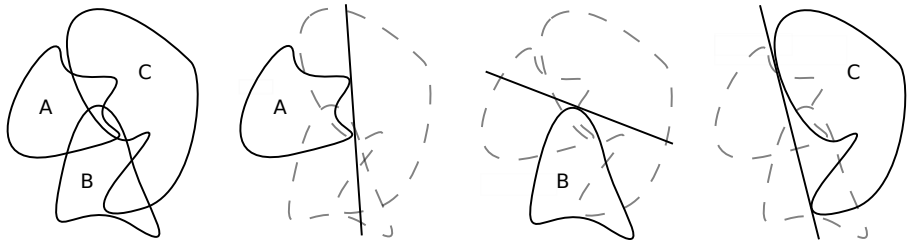


Figure 7.2: Subproblems in *binary relevance* for multilabel classification: original three-class problem (A, B and C, shown as overlapping clouds in the left-most picture) is divided into A vs. rest (second picture), B vs. rest (third) and C vs. rest two-class subproblems. Separating hyperplanes, denoted by a straight line, have to respect all examples (inside the clouds). Clouds of negative examples have dashed lines.

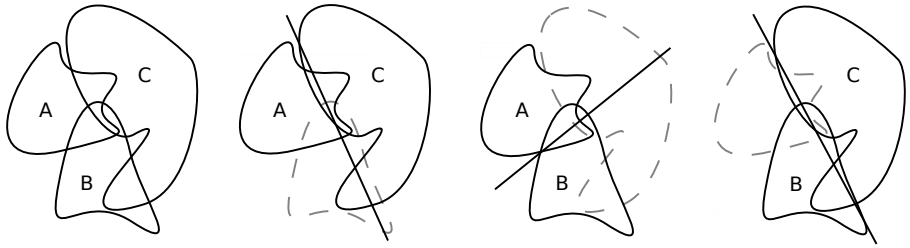


Figure 7.3: Subproblems in *pairwise* multilabel classification: original three-class problem is divided into A vs. C (second picture, dashed examples are ignored), A vs. B (class C is ignored) and B vs. C two-class subproblems. Separating hyperplanes have to respect only examples from two classes in contrast to BR in Figure 7.2. Dashed lines denote the ignored class.

So, each of the k classifiers is trained in order to determine the relevance of one particular label. In consequence, the combined prediction of the BR classifier for an instance x would be the set $\{\lambda_j \mid f_j(x) = 1\}$. Self-evidently, if the classifiers are capable of returning probability estimates or score-based predictions due to the used learning algorithm, one can obtain a ranking of classes according to their relevance.

7.3.2 Multilabel Pairwise Learning (MLP)

In the pairwise binarization method for *multiclass* classification (previously described in Section 3.3.2 on page 17), 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 (λ_u, λ_v), only examples belonging to either λ_u or λ_v are used to train the corresponding classifier $f_{u,v}$. All other examples are ignored.

In the multilabel case, and assuming $u < v$, an example x is added to the training set for classifier $f_{u,v}$ if λ_u is a relevant class and λ_v is an irrelevant class or vice versa, i.e., if $(\lambda_u, \lambda_v) \in P_x \times N_x$ or vice versa $(\lambda_u, \lambda_v) \in N_x \times P_x$ with $N_x = L \backslash P_x$ as negative labelset (cf. Figure 7.3). Thus training examples of class λ_u will receive a training signal of $+1$, whereas training examples of class λ_v will be classified with -1.

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

Figure 7.4: MLP voting (Loza Mencía and Fürnkranz, 2008a): an example x is classified by all 10 base classifiers $f_{i,j}, i \neq j$, $\lambda_i, \lambda_j \in L$. Note the redundancy given by $f_{i,j} = -f'_{j,i}$. The last line counts the positive outcomes for each class.

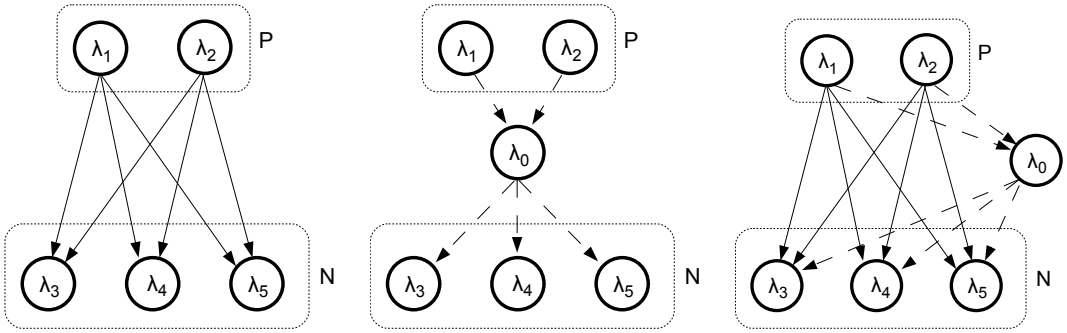
Furthermore, the number of training instances regarding a classifier $f_{u,v}$ is always less than or in the worst case equal to the number of total training instances.

During classification, the predictions or votes of the base classifiers $f_{u,v}$ can be interpreted as *preference statements* that predict for a given example which of the two labels λ_u or λ_v is preferred. In order to convert these binary preferences into a class ranking, we use simple weighted voting (cf. Section 3.3.2 on page 18), which interprets each binary preference as a vote for the preferred class. Classes are then ranked according to the number of received votes after the evaluation of all $\frac{k(k-1)}{2}$ classifiers. Ties are broken randomly in our case.

Figure 7.4 shows a possible result of classifying the sample instance of Figure 7.5a on the facing page. Classifier $f_{1,5}$ predicts (correctly) the first class, consequently λ_1 receives one vote and class λ_5 zero (denoted by $f_{1,5} = 1$ in the first and $f_{5,1} = -1$ in the last row). All 10 classifiers (the values in the upper right corner can be deduced due to the assumed symmetry property of the classifier) are evaluated though only six are 'competent' since only they were trained with the original example.

This may be disturbing at first sight since many non-competent classifiers, i.e. classifiers discriminating λ_u and λ_v for a given instance x with $\lambda_u, \lambda_v \in P_x$ or $\lambda_u, \lambda_v \in N_x$, are involved in the voting process: $f_{1,2}$ is asked though it cannot know anything relevant in order to determine if 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 resulting noisy votes (votes from non-competent classifiers) concentrate on a single negative class, 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 competent classifiers all classify correctly and the non-competent ones concentrate on a single class, a positive class would still receive at least $k - |P|$ and a negative at most $k - |P| - 1$ votes. Class λ_3 in Figure 7.4 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 BR because it profits from simpler decision boundaries in the subproblems (Fürnkranz, 2002; Hsu and Lin, 2002). These subproblems can be significantly smaller in terms of training instances compared to subproblems of BR, which always use the complete training set for learning. Typically, this goes hand in hand with an increase of the space where a separating hyperplane can be found. An intuitive visualization of



(a) MLP training: arrows represent the classifiers $f_{1,3}$, $f_{1,4}$, $f_{1,5}$, $f_{2,3}$, $f_{2,4}$, $f_{2,5}$, for which x is used during training.

(b) Calibration: introducing virtual label λ_0 that separates P and N. Classifiers $f_{1,0}$, $f_{2,0}$, $f_{0,3}$, $f_{0,4}$, $f_{0,5}$ are additionally trained.

(c) CLR training: the complete set of classifiers for which x is used for training.

Figure 7.5: Graphical representation of MLP Training, Calibration and CLR Training (Brinker et al., 2006). Here, let $P_x = \{\lambda_1, \lambda_2\}$ and $N_x = \{\lambda_3, \lambda_4, \lambda_5\}$ for a given example x.

this aspect can be found in Figure 3.1 on page 17 for the multiclass case (where one-against-all is identical to the binary-relevance decomposition) and in Figure 7.3 on page 91 for the multilabel case, in contrast to the BR binarization depicted in Figure 7.2 on page 91. 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.

7.3.3 Calibrated Label Ranking (CLR)

To convert the resulting ranking of labels into a multilabel prediction, the *calibrated label ranking* approach (Brinker et al., 2006; Fürnkranz et al., 2008) can be used. 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 7.5b, the combination with the normal pairwise base classifiers is shown in Figure 7.5c.

As it turns out, the resulting k additional binary classifiers $\{ f_{i,0} \mid i = 1 \dots k\}$ are identical to the classifiers that are trained by the binary relevance approach. Thus, each classifier $f_{i,0}$ is trained in a one-against-all fashion by using the whole dataset with $\{ x \mid \lambda_i \in P_x\} \subseteq X$ as positive examples and $\{ x \mid \lambda_i \in N_x\} \subseteq X$ as negative examples. At prediction time, we will thus get a ranking over $k + 1$ labels (the k original labels plus the calibration label). Then, the projection of voting

$f_{0,1} = -1$	$f_{1,0} = 1$	$f_{2,0} = 1$	$f_{3,0} = -1$	$f_{4,0} = -1$	$f_{5,0} = -1$
$f_{0,2} = -1$	$f_{1,2} = 1$	$f_{2,1} = -1$	$f_{3,1} = -1$	$f_{4,1} = -1$	$f_{5,1} = -1$
$f_{0,3} = 1$	$f_{1,3} = 1$	$f_{2,3} = 1$	$f_{3,2} = -1$	$f_{4,2} = -1$	$f_{5,2} = -1$
$f_{0,4} = 1$	$f_{1,4} = 1$	$f_{2,4} = 1$	$f_{3,4} = 1$	$f_{4,3} = -1$	$f_{5,3} = -1$
$f_{0,5} = 1$	$f_{1,5} = 1$	$f_{2,5} = 1$	$f_{3,5} = 1$	$f_{4,5} = 1$	$f_{5,4} = -1$
$v_0 = 3$	$v_1 = 5$	$v_2 = 4$	$v_3 = 2$	$v_4 = 1$	$v_5 = 0$

Figure 7.6: MLP voting with calibrated label λ_0: an example x is classified by all 15 base classifiers. The last line counts the positive outcomes for each class.

aggregation of pairwise classifiers with a calibrated label to a multilabel output is quite straight-forward:

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

where $v(\lambda)$ is the amount of votes class λ has received.

Figure 7.6 extends the example from Figure 7.4 on page 92 and shows a possible result of classifying with the calibrated label λ_0. It shows the ideal case, where for instance, the relevant classes λ_1 and λ_2 receive a vote respectively in direct comparison with the calibrated label (classifiers $f_{1,0}$ and $f_{2,0}$). After evaluating all classifiers, the number of votes for the calibrated label $v(\lambda_0) = v_0$ is used as the split-point to discriminate relevant classes from irrelevant classes. In this example, λ_1 and λ_2 are returned as the set of relevant classes $\hat{P}$.

We denote the MLP algorithm adapted in order to support the calibration technique as CLR.

7.3.4 Hierarchy of Multilabel Classifiers (HOMER)

HOMER (Tsoumakas et al., 2008) follows the divide-and-conquer paradigm of algorithm design. The main idea is the transformation of a multilabel classification task with a large set of labels L into a tree-shaped hierarchy of simpler multilabel classification tasks, each one dealing with a small number $\beta \ll k$ of labels.

This tree-shaped hierarchy has k leaves, one leaf for each class λ_j of L. Each internal node ν contains the union of the labelsets of its children, $L_\nu = \bigcup L_{\text{CH}}, \text{CH} \in \text{children}(\nu)$. The root contains all labels, $L_{\text{root}} = L$.

Each non-leaf node represents a multilabel prediction problem that assigns a set of meta labels to an example. A *meta label* μ_ν of a node ν is defined as the disjunction of the labels contained in that node, i.e., $\mu_\nu \equiv \bigvee \lambda_j, \lambda_j \in L_\nu$. These meta labels have the following semantics: a training example can be considered to be annotated with meta label μ_ν if it is annotated with at least one of the labels in L_ν.

HOMER associates a multilabel classifier h_ν with each internal node ν of the hierarchy. The task of h_ν is the prediction of one or more of the meta labels of its children. Therefore, the set of labels for h_ν is $M_\nu = \{\mu_{\text{CH}} \mid \text{CH} \in \text{children}(\nu)\}$. Figure 7.7 on the next page shows a sample hierarchy produced for a multilabel classification task with 8 labels $\{\lambda_1, \ldots, \lambda_8\}$.

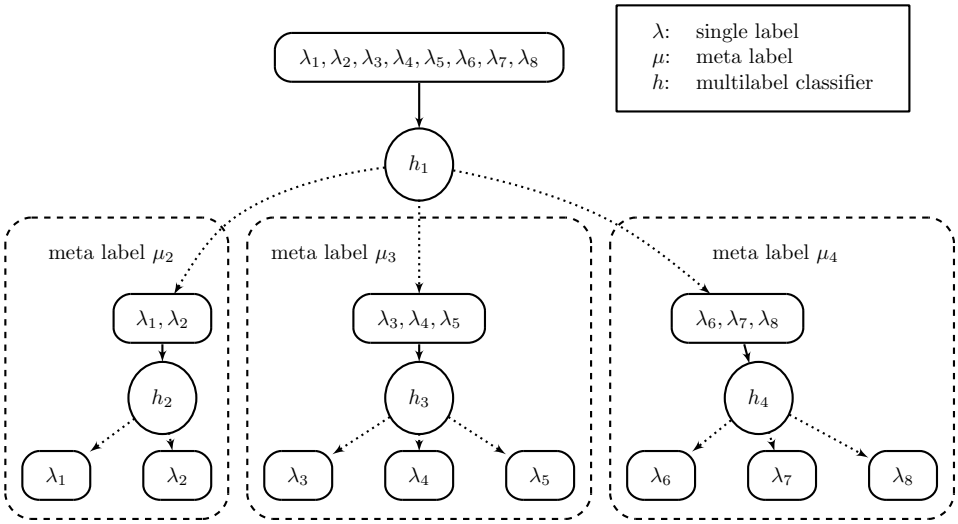


Figure 7.7: Sample hierarchy for a multilabel classification task with 8 labels (based on Tsoumakas et al., 2008). The original multilabel problem with $k = 8$ is transformed to 4 multilabel problems (to learn h_1, h_2, h_3 and h_4) with $k_1, k_3, k_4 = 3$ and $k_2 = 2$. Classifier h_1 predicts meta labels μ_2, μ_3, μ_4 and is learned with meta examples, i.e. examples which label values are adapted according to the partitioning. For instance, an example x with relevant labels λ_1 and λ_3 is relabeled as μ_2 and μ_3.

For the multilabel classification of a new instance x, HOMER starts with h_{root} and follows a recursive process forwarding x to the multilabel classifier h_{CH} of a child node CH only if μ_{CH} is among the predictions of $h_{\text{parent(CH)}}$. Eventually, this process may lead to the prediction of one or more single-labels by the multilabel classifier(s) just above the corresponding leaf(ves). The union of these predicted single-labels is the output in this case, while the empty set is returned otherwise.

In the training phase, HOMER creates the tree recursively in a top-down depth-first fashion starting with the root. At each node ν, β child nodes are first created using a clustering algorithm (see below). In case $|L_\nu| < \beta$, the number of children is set to $|L_\nu|$. Each such child ν filters the data of its parent, keeping only the examples that are annotated with at least one of its own labels: $D_\nu = \{(x_i, P_i) \mid (x_i, P_i) \in D_{\text{parent}(\nu)}, P_i \cap L_\nu \neq \emptyset\}$. The root uses the whole training set, $D_{\text{root}} = D$. The examples in D_ν are then transformed into meta examples (x_i, Z_i), where $Z_i = \{\mu_{\text{CH}} \mid \text{CH} \in \text{children}(\nu), P_i \cap L_{\text{CH}} \neq \emptyset\}$, which are subsequently used for training h_ν.

The main issue in the former process is how to distribute the labels of L_ν to the β children. Tsoumakas et al. (2008) argue that labels should be *evenly* distributed to β subsets in a way such that labels belonging to the same subset are as *similar* as possible. Such a task can be thought of as clustering with the additional constraint of equal cluster size. It has been considered in the past in the literature, under the name *balanced clustering* (Banerjee and Ghosh, 2006). In (Tsoumakas et al., 2008), a new balanced clustering algorithm named balanced k-means has been proposed for HOMER, which guarantees that the clusters will be of exactly the same size.

The justification for preferring a similarity-based distribution is that if similar labels of a node ν are placed in the same subset, then only a few (ideally just one) meta labels of h_ν will be predicted and thus the rest sub-trees will not be activated. This will lead to reduced cost during the operation and testing of HOMER. Another expected benefit is that each child node will probably contain less training examples. The justification for preferring an even distribution is that the multilabel classifiers at each node will deal with a more balanced distribution of positive examples for each meta label. This is expected to lead to improved predictive performance.

In total, HOMER can be considered as the combination of two components: a) an algorithm that constructs a hierarchy on top of the labels of a multilabel dataset, and b) a generalization of the well-known Pachinko-machine hierarchical classification algorithm (Koller and Sahami, 1997) to the multilabel case.

In (Tsoumakas et al., 2008), HOMER, when using the well-known binary relevance classifier (BR) as the base multilabel classifier in each internal node, has shown to outperform BR in terms of quality of prediction and, especially, classification time.

8 Efficient Pairwise Multilabel Prediction

Contents

The predominant approach to multilabel classification is *binary relevance* (BR) learning (cf. Section 7.3.1 on page 90). A different, more *costly* approach is to have a classifier for each possible pair of classes or labels that is trained to distinguish only between these two classes. This pairwise approach (cf. Section 7.3.2 on page 91) has shown to achieve a higher predictive quality in the multiclass (Fürnkranz, 2002; Hsu and Lin, 2002) as well as in the multilabel case (Loza Mencía and Fürnkranz, 2008c; Fürnkranz et al., 2008).

Similar as for the case of ternary error-correcting output codes in Chapter 6 on page 55, we introduce in this chapter a novel algorithm which adapts the QWeighted method to improve the efficiency of the pairwise approach in the multilabel classification setting. 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 labelset is predicted. In order to determine at which position to stop, we use the calibrated label ranking technique (cf. Section 7.3.2 on page 91), which introduces an artificial label for indicating the boundary between relevant and

irrelevant classes. We evaluated this technique on a selection of multilabel datasets that vary in terms of problem domain, number of classes and label density. The results demonstrate that this modification allows the pairwise technique to process such data in comparable time to the one-per-class approaches while producing more accurate predictions.

This chapter is organized as follows: Section 8.1 describes briefly the perceptron algorithm, which was used as base learner, and their combination with MLP and CLR. Then, Section 8.2 describes the adaptation of QWeighted to multilabel classification. In Section 8.3 we compare the time and space complexity of the different algorithms. Section 8.4 is dedicated to the experimental evaluation along with its setup and used datasets and, finally, we provide a discussion and conclusion of this chapter in Section 8.5.

8.1 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 g-dimensional space into two halves corresponding to positive and negative examples. We use a version that works without learning rate and threshold:

$$o(x) = \operatorname{sgn}(x \cdot \bar{\mathrm{w}}) \tag{8.1}$$

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

$$\alpha_i = (\lambda_i - o(x_i)) \qquad \bar{\mathrm{w}}_{i+1} = \bar{\mathrm{w}}_i + \alpha_i x_i \tag{8.2}$$

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.

Nevertheless, we also experimented with SVMs as base classifier. Although we were able to increase prediction accuracy in some cases, many datasets could not be processed despite using the efficient LibLinear library (Fan et al., 2008).

Algorithm 7 Pseudocode of the incremental training method of the MLPP algorithm.

Require: Training example pair (x_i, P_i), perceptrons $\{\bar{w}_{u,v} \mid u < v, \lambda_u, \lambda_v \in L\}$

```
N_i ← L \ P_i
for each (λ_u, λ_v) ∈ P_i × N_i do
    if u < v then
        w̄_{u,v} ← TrainPerceptron(w̄_{u,v}, (x_i, 1))     # train as positive example
    else
        w̄_{v,u} ← TrainPerceptron(w̄_{v,u}, (x_i, −1))    # train as negative example
return {w̄_{u,v} | u < v, λ_u, λ_v ∈ L}                  # updated perceptrons
```

8.1.1 MLPP and CMLPP

As previously mentioned, the incremental training capability of perceptrons are particularly valuable in a large scale setting and it transfers naturally to its combination with the pairwise approach for multilabel classification. This means, it is not longer necessary to maintain all training instances in the memory at training time. Algorithm 7 shows the efficient incremental training procedure for the pairwise approach with perceptrons as base models.

In the following, we will refer to the pairwise approach for multilabel classification (MLP) (cf. Section 7.3.2 on page 91) in conjunction with perceptrons as base model as MLPP. The same applies for the further combination with the calibrated label ranking approach (Section 7.3.3 on page 93), which will be denoted as CMLPP.

8.2 Quick Weighted Voting for Multilabel Classification

The calibrated label ranking approach evaluates at prediction time a rather costly quadratic number of classifiers. Two approaches to overcome this problem based on QWeighted are presented in the following.

8.2.1 QCMLPP1

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 the set of labels L and repeat this step, until the returned class is the artificial label λ_0, which means that all remaining classes will be considered to be irrelevant.

This adaptation uses two simple extensions of the original algorithm. Firstly, 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. And secondly, 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. We denote this method as QCMLPP1.

Algorithm 8 QCMLPP2

Require: example x; pairwise classifiers $\left\{ f_{i,j} \mid i < j \wedge \lambda_i, \lambda_j \in L \cup \{\lambda_0\} \right\}$

$\vec{l} \in \mathbb{R}^{k+1} \leftarrow 0$
$\vec{v} \in \mathbb{R}^{k+1} \leftarrow 0$
$\tilde{P} \leftarrow \emptyset$
$G \leftarrow \emptyset$ # keep track of evaluated classifiers

for $i \leftarrow 1$ to k **do** # evaluate all classifiers of artificial label λ_0
    $l_i \leftarrow f_{0,i}(x)$
    $v_0 \leftarrow v_0 + (1 - l_i)$ # compute votes of calibrated label

repeat
    $\lambda_{top} \leftarrow$ NULL # apply adapted QWeighted
    while $\lambda_{top} =$ NULL **do** # (cf. Algorithm 1 on page 20)
        $\lambda_a \leftarrow \underset{\lambda_i \in L}{\operatorname{argmin}}\ l_i$
        $\lambda_b \leftarrow \underset{\lambda_j \in L \setminus \{\lambda_a\},\ f_{a,j} \notin G}{\operatorname{argmin}}\ l_j$
        if $v_a \geq v_0$ **or** no λ_b exists **then** # adapted stopping criterion
            $\lambda_{top} \leftarrow \lambda_a$
        else # evaluate classifier
            $v_{ab} \leftarrow f_{a,b}(x)$ # update statistics
            $v_a \leftarrow v_a + v_{ab}$
            $v_b \leftarrow v_b + (1 - v_{ab})$
            $l_a \leftarrow l_a + (1 - v_{ab})$
            $l_b \leftarrow l_b + v_{ab}$
            $G \leftarrow G \cup f_{a,b}$ # update already evaluated classifiers

    if $v_{top} \geq v_0$ **then**
        $\tilde{P} \leftarrow \tilde{P} \cup \lambda_{top}$ # relevant label found
        $l_{top} \leftarrow +\infty$ # arrange λ_{top} at the end of possible opponents queue

until $v_{top} \geq v_0$ **and** $|\tilde{P}| < k$ # check if all relevant labels found

return $\tilde{P}$ # return relevant labels

8.2.2 QCMLPP2

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*.

Now, 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. The pseudocode of QCMLPP2 is depicted in Algorithm 8 on the preceding page.

8.2.3 Discussion

Note that the effectiveness of both testing procedures 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.

Other variants of QCMLPP1/2 may possibly further improve 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.

8.3 Computational Complexity

The notation used in this section is the following: k denotes the number of possible classes, d the average number of relevant classes per instance in the training set, g the number of attributes and g' the average number of attributes not zero (size of the sparse representation of an instance), and t 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.

8.3.1 Memory Requirements

BR follows an one model per class approach, so it has to keep one perceptron for each class in memory, leading to $O(k \cdot g)$ memory space. In contrast, the pairwise approaches require one perceptron for each of the $\frac{k(k-1)}{2}$ pairs of classes, hence we

Table 8.1: Computational complexity given as upper bounds of number of addition and multiplication operations, for each instance. k: *#classes*, d: *avg. #labels per instance*, g: *#attributes*, g': *#attributes* $\neq 0$.

	training	prediction	memory
BR	$O(kg')$	$O(kg')$	$O(kg)$
MLPP	$O(dkg')$	$O(k^2g')$	$O(k^2g)$
QCMLPP	$O(dkg')$	$\sim kg' + dk \log(k)\ g'$	$O(k^2g)$

need $O(k^2g)$ memory. In addition, the calibrated versions require an overhead of k perceptrons for the comparisons with the artificial label.

8.3.2 Training

For processing one training example, k dot products have to be computed by BR, plus at most the same amount if there was a prediction error. MLPP requires $O(dk)$ dot products, one for each associated perceptron. Assuming that a dot product computation costs $O(g')$, we obtain a complexity of $O(dkg')$ 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 algorithm despite training a quadratic number of base classifier.

8.3.3 Prediction

During prediction the one-per-class approach achieves $O(kg')$ computations for one instance. For the pairwise approach without the usage of QWeighted all perceptrons have to be evaluated, leading to $O(k^2g')$ 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 $\frac{k(k-1)}{2}$ to $k \log(k)$ in practice (cf. Section 6.2.1 on page 63). Let C_{QW} be the runtime of one iteration of QWeighted. Then, it is easy to see that the number of base classifier evaluations for the multilabel adaptations of QWeighted is bounded from above by $k + d \cdot C_{\mathrm{QW}}$, since we always evaluate the k classifiers involving the calibrated class, and have to do one iteration of QWeighted for each of the (on average) d relevant labels. Assuming that QWeighted on average needs $C_{QW} = k \log(k)$ base classifier evaluations we can expect an average number of $k + dk \log k$ classifier evaluations for the QCMLPP variants, as compared to the $\approx k^2$ evaluations for the regular CMLPP. Thus, the effectiveness of the adaption to the multilabel case crucially depends on the average number d of relevant labels. We can expect a high reduction of pairwise comparisons if d is small compared to k, which holds for most real-world multilabel datasets.

Table 8.2: Dataset characteristics. 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. *Labelset size* d denotes the average number of labels per instance, and *label density* indicates the average number of labels per instance d relative to the total number of classes k.

dataset	k	#instances t	#attributes g	d	density
scene	6	2407	294 (86732)	1.07	17.9 %
emotions	6	593	72	1.87	31.1 %
yeast	14	2417	103 (10712)	4.24	30.3 %
tmc2007	22	28596	49060	2.16	9.8 %
genbase	27	662	1186	1.25	4.6 %
medical	45	978	1449	1.25	2.8 %
enron	53	1702	1001	3.39	6.4 %
mediamill	101	43907	120	4.38	4.3 %
rcv1-v2	101	804414	231188 (25000)	3.24	3.1 %
r21578	120	11367	21474 (10000)	1.26	1.0 %
bibtex	159	7395	1836	2.4	1.5 %
eurlex_sm	201	19348	166448 (5000)	2.21	1.1 %
eurlex_dc	410	19348	166448 (5000)	1.29	0.3 %
delicious	983	16105	500	19.02	1.9 %

A compilation of the analysis can be found in Table 8.1 on the preceding page, together with the complexities of BR. 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.

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.

Note that we have assumed a linear dependence on the number of training instances since we use the perceptron algorithm as our base classifier. For base classifiers with a super-linear relationship the ratio to BR in terms of training complexity may be further reduced due to the smaller subproblems (Fürnkranz, 2002). For instance in the multiclass setting, a perceptron needs the same time for a problem of t examples as for k problems of $\frac{t}{k}$ examples. But this relation does not hold for learning algorithms like SVMs or C4.5 since $t^x > k(\frac{t}{k})^x = (\frac{t^x}{k^{x-1}})$ for $x > 1$.

8.4 Experimental Evaluation

8.4.1 Datasets

The datasets that were included in the experimental setup cover three application areas in which multilabeled data are frequently observed: *text categorization* (among

others, the *Reuters-RCV1*[1] and *Reuters-21578*[2] datasets and the *EUR-Lex*[3] dataset), *multimedia classification* (the *scene*, *mediamill* and *emotions* datasets) and *bioinformatics* (*yeast* and *genbase*).[4] Table 8.2 on the preceding page provides an overview of the different characteristics of the used datasets.

The *Reuters Corpus Volume I (Reuters-RCV1)* is one of the most widely used test collection for text categorization research, which was also used in Section 6.2.1 on page 68. It contains 804,414 newswire documents, which we split here 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 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 *EUR-Lex* is a recent dataset containing 19,348 legislative documents from the European Union. The documents are classified according to three different classification schemes: *subject matter* with 201 classes, *directory code* with 410 classes and *EUROVOC* with 3956 classes. However, we did not conduct experiments on the latter dataset since with almost 4000 classes we would need to maintain nearly 8 mio. perceptrons in memory. A special variant of MLPP was developed in order to be able to process datasets of this size (Loza Mencía and Fürnkranz, 2008b). After a similar pre-processing as for *RCV1* and *Reuters-21578*, we obtained 5,000 features.

Other text classification datasets include *medical* from a competition that aimed at assigning codes from the International Classification of Diseases to clinical free texts, the *enron* dataset of business-related emails from the Enron Corp. management, *bookmarks* and *bibtex*, collections from the social bookmarking platform BibSonomy, the *tmc2007* dataset of aviation safety reports assigned to flight problem types, and the large *delicious* dataset extracted from the *del.icio.us* social bookmarking platform. We used the pre-determined training/test splits.

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[5]. Each of 2417 genes is represented with 103 features. Previous experiments of Loza Mencía and Fürnkranz (2008c) indicate, that even the pairwise problems of this dataset are hard to separate with a linear classifier (much more so in the binary

1 http://www.jmlr.org/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm
2 http://www.daviddlewis.com/resources/testcollections/reuters21578/
3 http://www.ke.tu-darmstadt.de/resources/eurlex/
4 http://mlkd.csd.auth.gr/multilabel.html.
5 http://mips.gsf.de/genre/proj/yeast/

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.

Furthermore, the *genbase* dataset contains a protein classification task. The dataset from the *mediamill* Challenge is dedicated to news video classification, and in *emotions* the task is assign emotions to music.

8.4.2 Experimental Setup

All algorithms are trained incrementally. For the *RCV1* dataset, a single, chronological pass through the data was used (one epoch) because previous results have shown that multiple iterations are not necessary (Loza Mencía and Fürnkranz, 2008c). For the remaining text classification we report the results for ten epochs. The classifiers for the supposedly more difficult non-textual datasets were trained using 100 epochs. 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.

For *yeast*, *scene*, *Reuters-21578* and *EUR-Lex* the 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 text datasets, the TF-IDF transformation and the feature selection were conducted only on the training sets of the cross-validation splits. For datasets for which it was not indicated we used the first two thirds of examples for training and the remaining for testing. Specifically, we used 391 training examples for *emotions*, 21519 for *tmc2007*, 463 for *genbase*, 465 for *medical*, 1123 for *enron*, 30993 for *mediamill*, the aforementioned 535,987 for *rcv1-v2*, 4930 documents for *bibtex* and 12,920 for *delicious*.

All the perceptrons of the different algorithms were initialized with random values.

8.4.3 Computational Efficiency

Our analysis of computational efficiency concentrates on the savings in base classifier evaluations using the QWeighted method on the different multilabel datasets.

Table 8.3 on the following page 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 (BR, CMLPP, QCMLPP1 and QCMLPP2) 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

Table 8.3: Computational costs at prediction in average number of classifier evaluations per instance. The italic values next to the two multilabel adaptations of QWeighted show the ratio of classifier evaluations to CMLPP and the second rightmost column describes the average number of relevant labels.

dataset	k	BR	CMLPP	QCMLPP1	QCMLPP2	$k \log(k)$	$k + dk \log(k)$	d	density $\frac{d}{k}$
scene	6	6	21	11.51 *(54.8 %)*	11.46 *(54.6 %)*	10.75	17.50	1.07	17.9 %
emotions	6	6	21	17.03 *(81.1 %)*	16.59 *(79.0 %)*	10.75	26.10	1.87	31.2 %
yeast	14	14	105	67.57 *(64.4 %)*	64.99 *(61.9 %)*	36.94	170.65	4.24	30.3 %
tmc2007	22	22	253	81.76 *(32.3 %)*	78.01 *(30.8 %)*	68.00	168.89	2.16	9.82 %
genbase	27	27	378	71.53 *(18.9 %)*	62.11 *(16.4 %)*	88.99	138.23	1.25	4.63 %
medical	45	45	1035	112.78 *(10.9 %)*	103.67 *(10.0 %)*	171.30	259.12	1.25	2.78 %
enron	53	53	1431	286.36 *(20.0 %)*	262.30 *(18.3 %)*	210.43	764.24	3.38	6.38 %
mediamill	101	101	5151	489.45 *(9.50 %)*	378.04 *(7.34 %)*	466.13	2142.64	4.38	4.34 %
rcv1-v2	103	103	5356	485.23 *(9.06 %)*	456.23 *(8.52 %)*	477.38	1649.70	3.24	3.15 %
r21578	120	120	7260	378.45 *(5.21 %)*	325.94 *(4.49 %)*	574.50	843.87	1.26	1.05 %
bibtex	159	159	12720	604.37 *(4.75 %)*	492.73 *(3.87 %)*	805.96	2093.29	2.40	1.51 %
eurlex_sm	201	201	20301	926.71 *(4.56 %)*	771.62 *(3.80 %)*	1065.96	2556.78	2.21	1.10 %
eurlex_dc	410	410	83845	1667.16 *(1.98 %)*	1136.85 *(1.35 %)*	2466.62	3591.95	1.29	0.31 %
delicious	983	983	483636	48680.12 *(10.1 %)*	46835.89 *(9.68 %)*	6773.47	129814.40	19.02	1.93 %

classifiers, which are typically evaluated in the CMLPP approach, are denoted within brackets, to emphasize the achieved reduction.

The first remarkable observation is the clear improvement using the QWeighted approach. Except for the four smallest datasets regarding the labelsize, both variants of the QCMLPP use less than 20 percent of the classifier evaluations for CMLPP.

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, *emotions*, achieved the lowest reduction, followed by *yeast* with a similar density and reduction ratio. Similarly both QCMLPP variants evaluated the lowest ratio of classifiers for the dataset with the lowest density, the *eurlex_dc* 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 but constantly outperforms QCMLPP1.

For estimating the average runtime in practice, two columns were included, which state the $k \log(k)$ and $k + dk \log(k)$ 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 $k + dk \log(k)$. 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 8.1 on the next page visualizes the above results and allows again a comparison to different complexity values such as k, $k \log(k)$ and k^2. The upper figure is a recapitulation of the results from Section 6.2.1 on page 63 extended with multiclass classification performance results of the multilabel datasets considered in this chapter: instead of evaluating until finding the calibrating label, QWeighted was only applied once such as if it was a multiclass problem. These results for the simulated multiclass classification performance support additionally the statement that QWeighted achieves an $k \log(k)$ 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 also that for the black curve ($k + dk \log(k)$), the actual average number of labels from data was used for computing the values and are identical to the ones from Table 8.3 on the preceding page. These values are also depicted in the additional Figure 8.2 on page 110, which shows again the comparison of computational costs split into two figures, the first for smaller datasets with $k < 103$ and the second for larger datasets. In comparison to Figure 8.1 on the next page, the x-axis is now linear and we have added the dataset names to the data points.

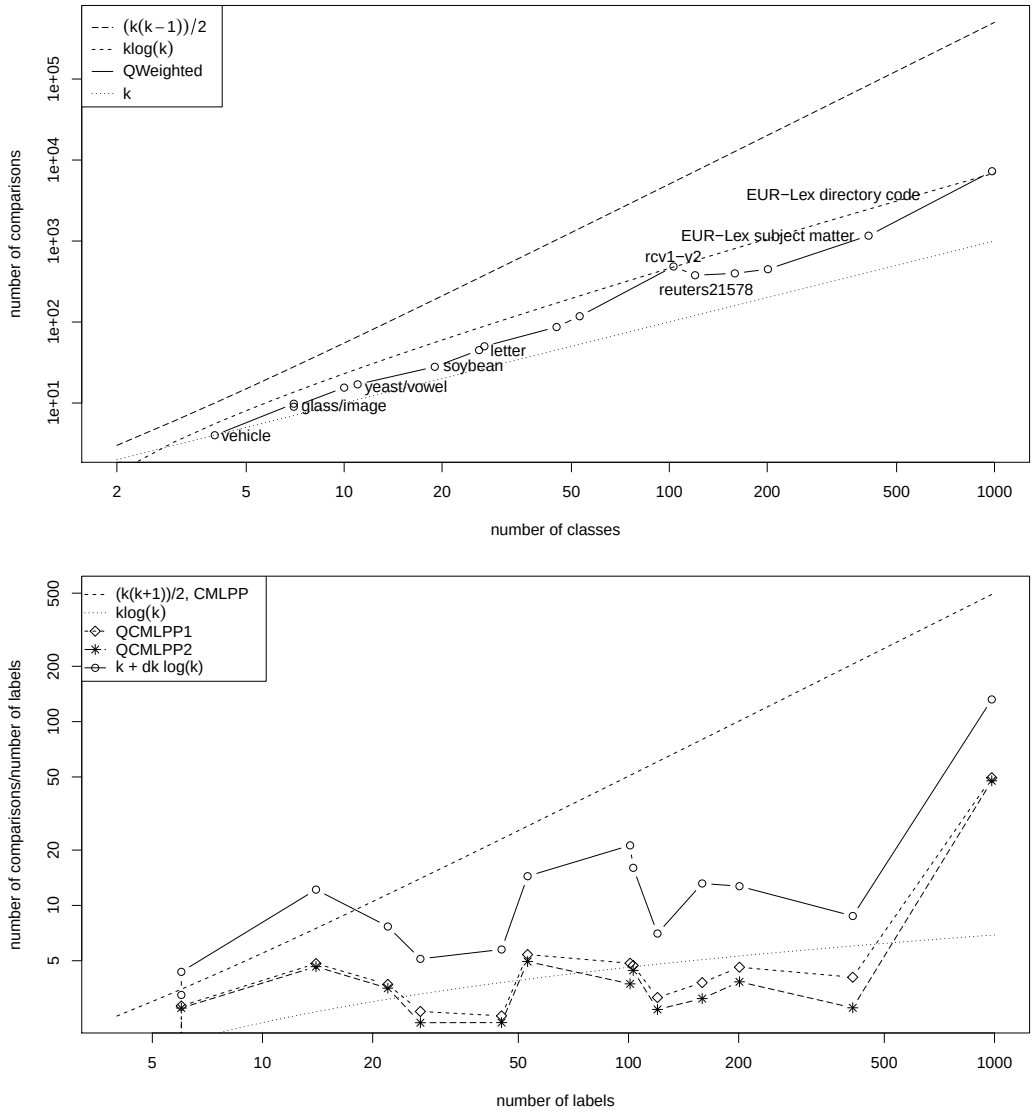


Figure 8.1: Prediction complexity of QWeighted and QCMLPP: number of comparisons needed in dependency of the number of classes k for different multiclass and multilabel problems.
Upper figure (multiclass): Problems *vehicle* to *letter* are multiclass problems already analyzed by Park and Fürnkranz (2007), while multiclass versions of the multilabel datasets described in Table 8.2 on page 103 were evaluated within this study.
Lower figure (multilabel): QCMLPP1/2 is compared to $k(k+1)/2$ as in CMLPP, k as in BR and $k \log(k)$ on 14 multilabel datasets.

Table 8.4: Multilabel performance of the different algorithms (in %, micro-averaged). For HammLoss low values are good, for the other three measures the higher the better. Bold values represent the best value for each dataset and measure combination. Note that the multilabel losses of QCMLPP are exactly equal to those of CMLPP.

		HammLoss		Precision		Recall		F1	
dataset	k	BR	CMLPP	BR	CMLPP	BR	CMLPP	BR	CMLPP
scene	6	10.42	**10.00**	71.80	**71.83**	71.21	**74.20**	71.19	**72.76**
emotions	6	35.64	**34.08**	46.78	**48.62**	60.15	**61.90**	52.63	**54.47**
yeast	14	24.09	**22.67**	60.47	**62.37**	59.07	**63.31**	59.76	**62.83**
tmc2007	22	7.37	**6.78**	62.57	**64.16**	66.47	**73.61**	64.46	**68.56**
genbase	27	**0.26**	0.48	99.22	**99.59**	**95.49**	90.60	**97.32**	94.88
medical	45	**1.51**	1.51	71.72	**76.02**	**75.84**	66.75	**73.72**	71.08
enron	53	7.56	**6.01**	41.56	**52.82**	47.05	**49.51**	44.13	**51.11**
mediamill	101	4.52	**4.16**	42.28	**56.66**	10.05	**19.70**	16.24	**29.23**
rcv1-v2	103	1.26	**1.03**	80.15	**84.89**	79.70	**81.61**	79.93	**83.22**
r21578	120	0.78	**0.55**	59.98	**72.89**	**78.36**	76.68	67.92	**74.63**
bibtex	159	1.57	**1.35**	46.53	**57.97**	**36.30**	34.84	40.78	**43.53**
eurlex_sm	201	0.76	**0.54**	63.39	**77.88**	**74.11**	71.57	68.32	**74.59**
eurlex_dc	410	0.26	**0.17**	56.26	**79.21**	**70.54**	61.98	62.58	**69.54**
delicious	983	5.58	**3.48**	11.88	**19.77**	**29.59**	26.51	16.95	**22.65**

As we can see from these figures, the empirical runtime bound $k + dk \log(k)$ is never exceeded. We conclude that this estimate is a reasonable indicator for the runtime complexity of QCMLPP.

8.4.4 Predictive Quality

Although it is not the focus of this study, we will compare in this section the prediction quality of BR and CMLPP in order to demonstrate the expected advantage of the pairwise approach. Note that the multilabel losses of the QCMLPP are exactly equal to those of CMLPP since both compute for every instance the same partitioning into relevant and irrelevant labels. Table 8.4 shows the labelset predictions performance according to Section 7.2 on page 86.

The first remarkable observation is that for the overall evaluation measures HammLoss and F1 the pairwise approach dominates the one-per-class approach for every dataset except *genbase* and *medical*. BR's Prec is even outperformed for these datasets. On the other hand, QCMLPP achieves a lower Rec for the datasets with slightly more than 100 classes, beginning at *reuters-21578* with 120 classes. This is due to the fact that the calibration tends to underestimate the number of returned labels for each instance, especially for a high number of total classes. A possible explanation for this behavior is the following: when the binary relevance classifiers, that are also included in CMLPP, predict that v classes are positive, then this means for the remaining classes that they have to obtain at least $k - v$ votes of their maximum of k votes in order to be predicted as positive. The probability that this

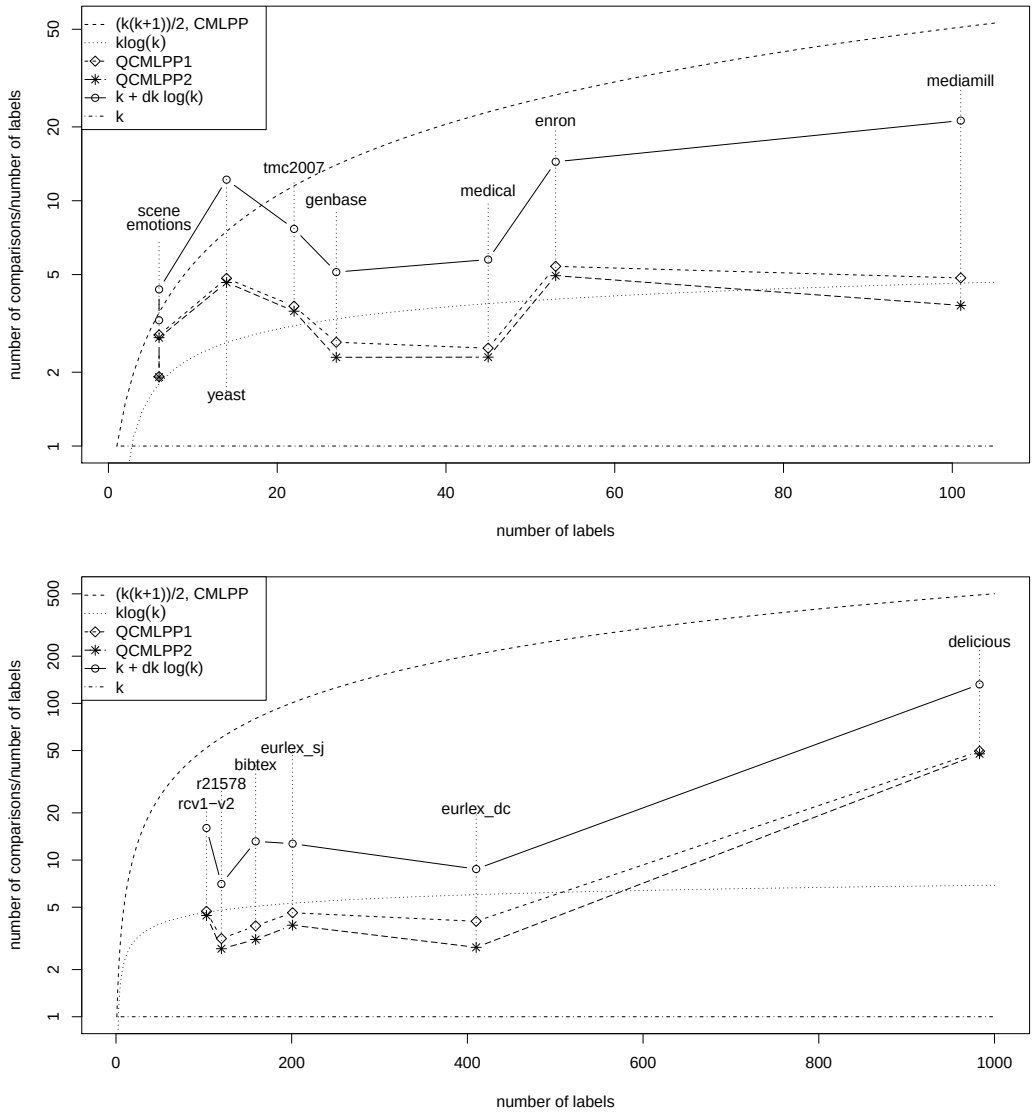


Figure 8.2: Prediction complexity of QCMLPP: the upper figure contains the small datasets (datasets with $k < 103$), the bottom figure the large datasets

Table 8.5: SVM as base learner - Computational costs at prediction in average number of classifier evaluations per instance. The italic values next to the multilabel adaptation of QWeighted (QCMLPP2) shows the ratio of classifier evaluations to CMLPP and the rightmost column describes the average number of relevant labels.

dataset	k	BR	CMLPP	QCMLPP2	$k \log(k)$	$k + dk \log(k)$	d
scene	6	6	21	7.88 *(37.5 %)*	10.75	17.50	1.07
emotions	6	6	21	11.87 *(56.5 %)*	10.75	26.10	1.87
yeast	14	14	105	40.31 *(38.4 %)*	36.94	170.65	4.24
tmc2007	22	22	253	68.92 *(27.2 %)*	68.00	168.89	2.16
medical	45	45	1035	97.40 *(9.41 %)*	171.30	259.12	1.25
enron	53	53	1431	223.42 *(15.6 %)*	210.43	764.24	3.38
r21578	120	120	7260	303.90 *(4.19 %)*	574.50	843.87	1.26
bibtex	159	159	12720	485.97 *(3.82 %)*	805.96	2093.29	2.40

happens for a real positive class decreases with increasing k, since it becomes more probable that at least v base classifiers mistakenly take a wrong decision. However, a look at the avg. predicted labelset size shows that this is only the case for the *EUR-Lex* datasets and not for *Reuters-21578* or *delicious*. For *delicious* QCMLPP even predicts more than 25 instead of 19 labels. On the other hand we can observe that BR always predicted a higher label number than QCMLPP on the dataset where it achieved a higher Rec. One extreme are the 47 predicted labels for *delicious*, but note that in general it cannot be stated that BR overestimates the number of labels.

Note that it is easily possible to bias the recall/precision trade-off of the calibration by simply subtracting or adding a fixed number of votes to the artificial class count.

8.4.5 Support Vector Machines

Such as BR, MLP is potentially able to use any binary classifier as base classifier. Therefore, we conducted experiments with Support Vector Machines as base learners in order to demonstrate that the same positive effects can also be expected from the pairwise approach and the QWeighted optimization when using a different base learner. We used the LIBSVM implementation (Chang and Lin, 2011) with standard settings for the non-textual datasets and the efficient LIBLINEAR implementation (Fan et al., 2008) for textual datasets with the primal L2-loss SVM option, which is supposed to enhance speed (Hsu et al., 2009a). We ignored the results on *genbase* since LIBSVM predicted the empty labelset on all test examples. For the remaining missing datasets no results could be retrieved due to the higher memory requirements of the SVMs compared to the simple perceptrons. For *yeast* and *scene* we did not use the quadratic kernel simulation.

Table 8.5 shows the computational costs of QCMLPP2 with SVM as base classifier. We can observe an overall similar picture compared to the results of Table 8.3 on page 106, the pairwise approach clearly benefits from the QWeighted optimization.

However, while the reduction in number of required comparisons for the textual datasets is very similar, using LIBSVM seems to allow to further improve the ratio on the non-textual *scene*, *emotions* and *yeast*. The explanation can be seen in Table 8.6 on the facing page, which lists the prediction quality for BR and QCMLPP2: A very high precision is achieved by LIBSVM for these datasets due to predicting only a small number of labels. This cautious behavior of LIBSVM could already be observed for the *genbase* dataset. QCMLPP2 with perceptrons as base classifier for instance predicts 2.51 labels in average on the *emotions* test set, while with SVM as base classifier only 1.27 are predicted. This means for QCMLPP2 in average more than one additional QWeighted iteration for each example during classification, which is the reason for the further reduction of the computational costs.

Note that although the obtained reductions in number of base classifier evaluations is similar for both perceptrons and SVM, training the SVMs does usually require a higher amount of CPU-time. Except for *emotions*, for which the time is almost equal, and *yeast* and *scene*, which are not directly comparable due to the different feature representations used, the perceptrons are always faster, namely 2.3 times faster for *tmc2007* to even 29 times faster for *enron*.

Especially if we consider that the prediction quality of perceptrons and SVMs are very similar (at least for the text classification tasks), this constitutes an important point in defense of the perceptron algorithm. However, it is also interesting to observe that the distance between BR and QCMLPP is considerably reduced when using SVMs, which might be an indication for a higher robustness against weak base classifier for the pairwise approach.

8.5 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 chapter a time efficient algorithm based on the pairwise approach.

The proposed approach combines a technique that transforms a class ranking into a bipartite prediction by introducing an artificial thresholding class, called calibration (cf. Section 7.3.3 on page 93), with the QWeighted voting that stops the computation of the ranking when the bipartite separation is already determined. 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 $k + dk \log(k)$, in comparison to the evaluation of k in the case

Table 8.6: Multilabel performance of the different algorithms with SVM as base learner (in %, micro-averaged). For HammLoss low values are good, for the other three measures the higher the better. Bold values represent the best value for each dataset and measure combination. Note that the multilabel losses of QCMLPP are exactly equal to those of CMLPP.

		HammLoss		Precision		Recall		F1	
dataset	k	BR	CMLPP	BR	CMLPP	BR	CMLPP	BR	CMLPP
scene	6	12.57	**12.51**	**93.25**	93.04	32.16	**32.58**	47.77	**48.21**
emotions	6	27.56	**26.57**	**65.55**	64.98	34.34	**41.85**	45.07	**50.91**
yeast	14	22.51	22.51	**75.61**	75.60	37.81	**37.82**	50.41	**50.41**
tmc2007	22	6.99	**6.63**	66.16	**67.31**	62.33	**66.16**	64.19	**66.73**
medical	45	**1.09**	1.11	**83.12**	82.10	76.56	**76.79**	**79.70**	79.36
enron	53	5.70	**5.22**	55.87	**59.95**	48.64	**53.36**	52.00	**56.47**
r21578	120	0.56	**0.55**	71.23	**71.76**	**78.49**	78.34	74.68	**74.90**
bibtex	159	1.48	**1.39**	50.45	**54.65**	37.60	**39.32**	43.09	**45.73**

of the one-per-class approach and $O(k^2)$ 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.

The benefit in predictive quality of using CMLPP against using BR was shown by an extensive experimental evaluation on 14 datasets. Together with QWeighted CMLPP is able to achieve a good trade-off between predictive quality and speed in the multilabel setting. Additional experiments using state-of-the-art support vector machines as base learner instead of the perceptron algorithm initially used in MLPP confirmed that the binary relevance approach is outperformed by the pairwise approach. These experiments also show that the advantage of using the pairwise approach and QWeighted is independent of the base learner employed.

However, this novel algorithm still uses a quadratic number of base classifiers, i.e. the memory requirements grow quadratically to the number of classes. This problem will be addressed in the next chapter.

9 Combination of Multilabel Classification Decompositions

Contents

Recently, a special group of transformation methods that decompose the original multilabel classification problem into a series of simpler problems has been proposed (Tsoumakas et al., 2008; Fürnkranz et al., 2008). These decomposition-based methods focus on dealing with problems with a large number of labels. In this chapter, we compare and combine these two recently proposed decomposition-based methods. The HOMER approach (cf. Section 7.3.4 on page 94) decomposes the problem into a hierarchy of simpler problems, where each problem uses a reduced number of possible labels. The hierarchical structure of the labels is obtained by applying recursive clustering to the initial set of labels. The calibrated label ranking approach (cf. Section 7.3.3 on page 93) interprets a multilabel problem as a special case of a preference learning problem (Hüllermeier et al., 2008). The sets of relevant labels that are associated with the training examples are interpreted as a bipartite preference relation between relevant and irrelevant labels. Each possible pairwise preference is independently modeled with a binary classifier. The predictions of these classifiers are then combined into an overall ranking of all labels, and an artificial calibration label indicates the position where the ranking should be split into relevant and irrelevant classes.

As was already elaborated in the previous chapter, although we can in practice reduce the number of classifier evaluations of CLR in the prediction phase, the problem of having to *store* a quadratic number of classifiers still remains to be solved, despite some recent progress for particular families of base classifiers (Loza Mencía and Fürnkranz, 2008b). This quadratic memory complexity of CLR may still pose a bottleneck for large scale problems.

For this reason, we investigate the combination between HOMER and Calibrated Ranking, because the latter is expected to greatly benefit from the reduction in the

number of labels that is provided by the former. In fact, our experimental results indicate that HOMER not only significantly reduces the memory complexity, it is also able to improve the classification performance, training time, and classification time for the calibrated ranking approach as well as for the binary relevance approach.

In the following sections, we will mainly focus on an extensive empirical evaluation of this combination. The combination itself is straight-forward and will be briefly described in Section 9.1. The extensive experimental study is presented in Section 9.2 while our conclusions are included in the last section of this chapter.

9.1 Combination of HOMER and QCLR

As described in Chapter 8 on page 97, QCLR[1] is a recently proposed efficient approach for multilabel classification. This algorithm combines three components: a) the pairwise decomposition of multilabel problems (Loza Mencía and Fürnkranz, 2008c), b) the calibrated label ranking (Fürnkranz et al., 2008) for determining a bipartition (multilabel result) and c) an adaption of the QWeighted algorithm (Park and Fürnkranz, 2007) for efficient voting aggregation that is used for prediction. But the most important and single necessary property in this context is, that it is a learning algorithm for multilabel classification problems. So, it can be naturally integrated into HOMER as the base multilabel learner. Recall that HOMER (cf. Section 7.3.4 on page 94) is a meta-learner which employs internally an ensemble of (base) multilabel learners.

9.1.1 Memory-Complexity

Regarding our objective to reduce the memory complexity of CLR, it may be apparent that we can expect a significant reduction for the combination with HOMER. However, we provide a brief analysis here.

The used partitioning method is important for a further analysis and we will for now assume an *equal-size* partitioning, i.e. for a given partitioning number β, the set of labels are equally divided into the β partitions. This holds for two considered partitioning schemes in this work, which will be later further described and evaluated in Section 9.2.2.

The partitioning parameter restricts the number of (meta) labels for each of the resulting decomposed multilabel problems to at most β. So, the number of classifiers in each internal multilabel problem in HOMER+CLR is now independent of the number of classes k of the original multilabel problem, and may be picked such that $\beta \ll k$. The number of pairwise classifiers and classifiers involving the calibrated label for each inner node are thus (at most):

$$n_{\mathrm{C}} \leq \frac{\beta(\beta-1)}{2} + \beta$$

1 We will refer to the base-classifier independent decomposition-scheme of QCMLPP as QCLR (Quick calibrated label ranking).

For given k and β the resulting tree has at most a depth[2] of $\lceil \log_\beta k \rceil - 1$, excluding the leaf level[3]. Furthermore, each level l of the tree has at most β^l nodes. So, an upper bound for the number of resulting multilabel problems is:

$$n_{\text{ML}} = 1 + \beta + \beta^2 + \cdots + \beta^{\lceil \log_\beta k \rceil - 1} = \sum_{i=0}^{\lceil \log_\beta k \rceil - 1} \beta^i$$

and thus by geometric series:

$$n_{\text{ML}} = \frac{1 - \beta^y}{1 - \beta} \text{ with } y := \lceil \log_\beta k \rceil$$

Now, in total, an upper bound for the total number of classifiers generated by HOMER is:

$$\begin{aligned} n_{\text{ML}} \cdot n_{\text{C}} &= \frac{1 - \beta^y}{1 - \beta} \cdot \left(\frac{\beta(\beta - 1)}{2} + \beta \right) \\ &= \frac{\beta^y - 1}{\beta - 1} \cdot \left(\frac{\beta(\beta - 1)}{2} + \beta \right) \\ &= \frac{\beta(\beta^y - 1)}{2} + \frac{\beta(\beta^y - 1)}{\beta - 1} \\ &= \frac{\beta(\beta + 1)(\beta^y - 1)}{2(\beta - 1)} \end{aligned}$$

This number of classifiers is significantly smaller for increasing k and fixed β with $\beta \ll k$ compared to the usual complexity of $k(k+1)/2$ for CLR. For instance, for a dataset with $k = 150, 300, 900$ and fixed number of partitions $\beta = 10$ the corresponding number of classifiers for CLR are: 11325, 45150 and 405450 compared to 6105 for HOMER+CLR in all three cases (note that the above formula gives an upper bound, which is not necessarily tight).

Moreover, if we deviate from the worst-case scenario and take the liberty to relax the expected depth of the tree to $y := \log_\beta k$, the estimate regarding the number of classifiers turns to:

$$n_{\text{ML}} \cdot n_{\text{C}} = \frac{\beta(\beta + 1)(k - 1)}{2(\beta - 1)}$$

which roughly shows that for fixed β, the number of classifiers grows linearly in the number of classes k by a factor of β.

9.2 Experimental Evaluation

In this section, after the presentation of the experimental setup, we will discuss the effect of the several parameters of HOMER and then compare it in terms of training time, classification time and predictive performance against its base multilabel classifiers.

2 Here, the root level has a depth of 0.

3 In this analysis, we are only interested in nodes, which represent classifiers. Therefore, only non-leaf nodes are considered.

Table 9.1: Name, number of examples used for training and testing, number of features and labels, label cardinality and density, and number of distinct labelsets for each dataset used in the experiments

	examples						
name	train	test	features	labels	cardinality	density	distinct labelsets
hifind	16452	16519	98	632	37.304	0.059	32734
eccv2002	42379	4686	36	374	3.525	0.009	3175
jmlr2003	48859	16503	46	153	3.071	0.020	3115
mediamill	30993	12914	120	101	4.376	0.043	6555

9.2.1 Experimental Setup

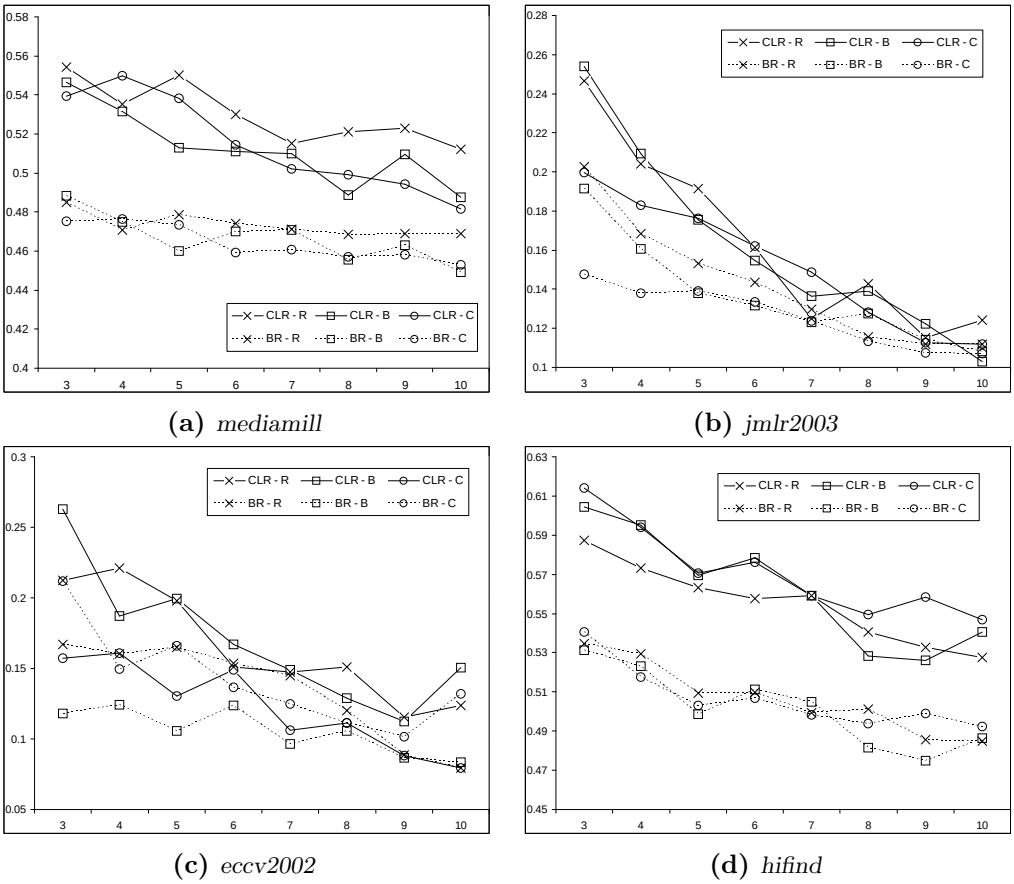
We conducted experiments on four large multilabel datasets with at least 100 labels and 10000 training examples. The first one, *hifind*, contains 32769 music titles annotated on average with 37 from 632 different labels (Pachet and Roy, 2009). The second dataset, *eccv2002* (Duygulu et al., 2002), is a popular benchmark for image classification and annotation methods. It is based on 5000 Corel images, 4500 of which are used for training and the rest 500 for testing. The third one, *jmlr2003*, is produced from the first (001) subset of the data accompanying (Barnard et al., 2003). It is based on 6932 images, 5188 of which are used to create the training set and the rest 1744 to create the test set. The last one is based on the *Mediamill* Challenge dataset (Snoek et al., 2006). It contains pre-computed low-level multimedia features from the 85 hours of international broadcast news video of the TRECVID 2005/2006 benchmark. Table 9.1 shows the number of examples used for training and testing, the number of features, the number of labels, the label cardinality and density, and the number of distinct labelsets for each dataset.

The experiments were conducted using the Mulan library of algorithms for multilabel learning (Tsoumakas and Vlahavas, 2007) and the decision tree learner J48 was used as base classifier.

Here, the effectiveness of all algorithms is evaluated with label-based micro-averaged recall, precision and F1 (cf. Section 7.2.1 on page 86). We also evaluate the efficiency of all algorithms based on their run time (for training and classification).

9.2.2 Results of HOMER with QCLR

This section presents and discusses the results of using HOMER together with QCLR as the multilabel algorithm for building models at each internal node of the hierarchy. We experimented with 8 different numbers of partitions (i.e., β ranges from 3 to 10) and 3 different methods for partitioning the set of labels at each internal node:



(a) *mediamill* **(b)** *jmlr2003*

(c) *eccv2002* **(d)** *hifind*

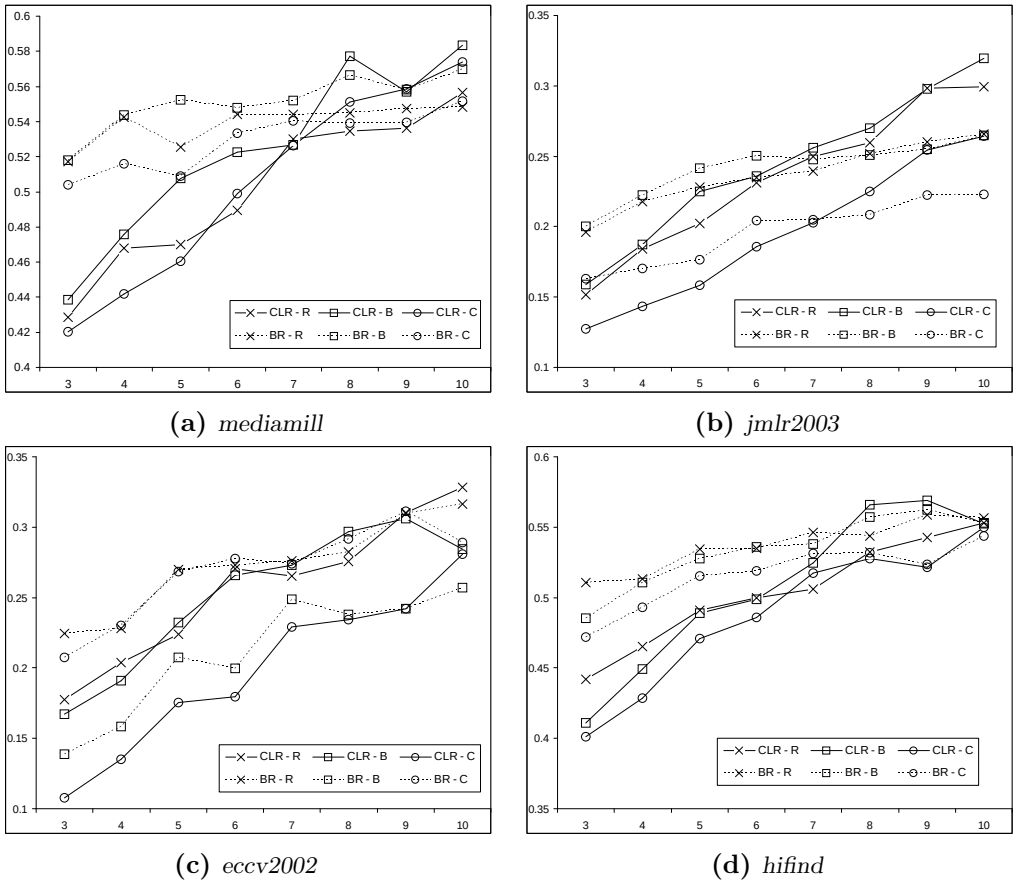
Figure 9.1: Micro recall over number of partitions for the six HOMER variants

- random and even distribution (R) of the labels to the children nodes,
- clustering (C) using the expectation minimization (EM) algorithm (as implemented in Weka (Hall et al., 2009)), and
- balanced clustering (B) using the algorithm introduced in (Tsoumakas et al., 2008), which is based on k-means clustering and imposes a constraint on the clusters such that their size is close to equal.

In addition to HOMER with QCLR[4] as multilabel classifier we ran the experiments using HOMER with BR[5] and also using the plain algorithms BR and QCLR without HOMER.

4 In the following graphs this combination is denoted as CLR-R, CLR-B, CLR-C respectively for all three different partitioning approaches.

5 In the following graphs this combination is denoted as BR-R, BR-B, BR-C respectively for all three different partitioning approaches.



(a) *mediamill* **(b)** *jmlr2003*

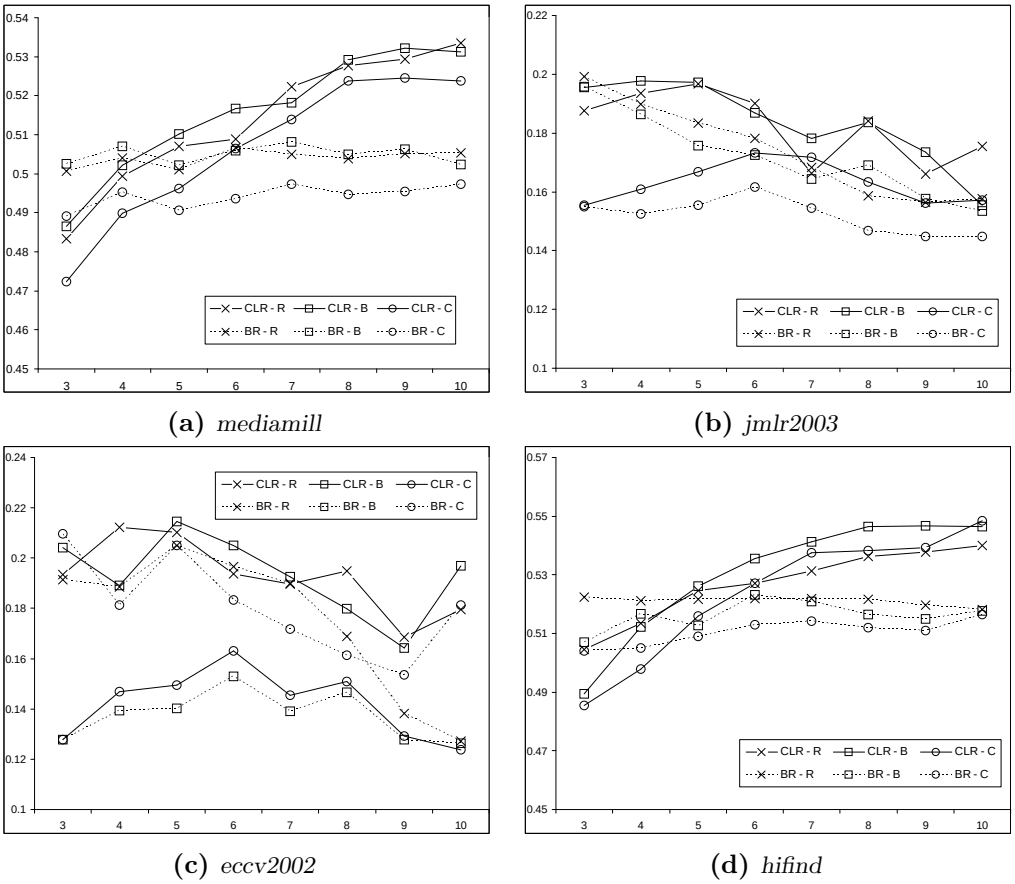
(c) *eccv2002* **(d)** *hifind*

Figure 9.2: Micro precision over number of partitions for the six HOMER variants

Prediction Quality

Figures 9.1 and 9.2 show the recall and precision results for the HOMER variants on all four datasets. We can see that recall decreases, while precision increases with the number of partitions, independently of the multilabel learner and partitioning method used. One potential reason for this behavior could be that smaller number of partitions lead to more general meta-labels that are more difficult to distinguish. Apparently this leads to a more relaxed prediction, so that at each inner node the multilabel classifier does predict more meta-labels and as a consequence more of the original labels, but with lower precision.

The recall of the CLR based HOMER variants seems to be larger than that of the BR based HOMER variants, irrespectively of the number of partitions. This is totally clear in *mediamill* and *hifind*, but less clear in *jmlr2003* and *eccv2002*, though it stills holds if we compare the two learners under the same partitioning method. As far as the partitioning method is concerned, there is no clear trend with respect to recall, while the plain clustering method seems to have the worst precision for both BR and CLR based HOMER.



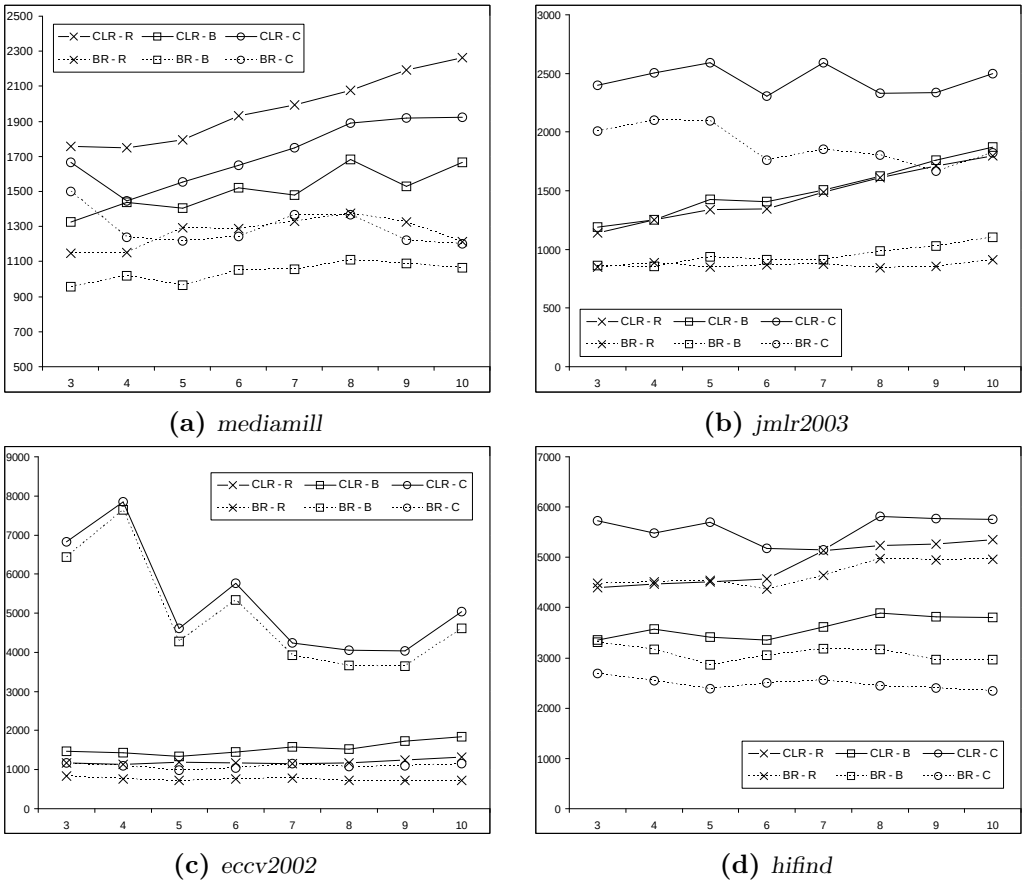
(a) *mediamill* **(b)** *jmlr2003*

(c) *eccv2002* **(d)** *hifind*

Figure 9.3: Micro F_1 over number of partitions for the six HOMER variants

The decrease in recall is stronger for CLR than for BR in the low density datasets *eccv2002* and *jmlr2003*. This means that in low density datasets, a small number of partitions favors the recall of HOMER with CLR. On the other hand the increase in precision is stronger for CLR than for BR in the high density datasets *mediamill* and *hifind*. This means that in high density datasets a large number of partitions favors the precision of HOMER. A potential reason is the fact that CLR underestimates the size of the predicted labelsets (Fürnkranz et al., 2008). It seems that this underestimation increases with the number of labels, as seen in the results of CLR that are discussed later on in this chapter.

Figure 9.3 shows the micro-averaged F_1 measure of HOMER for the datasets. As far as BR based HOMER is concerned no clear trend can be detected with respect to the number of partitions. With respect to the partitioning method, the plain clustering approach seems inferior to the rest, while no clear winner between balanced clustering and random partitioning can be announced. As far as CLR is concerned, as already outlined in the previous paragraphs, in low density datasets we notice a decrease of F_1 with respect to the number of partitions, while in high



(a) *mediamill* **(b)** *jmlr2003*

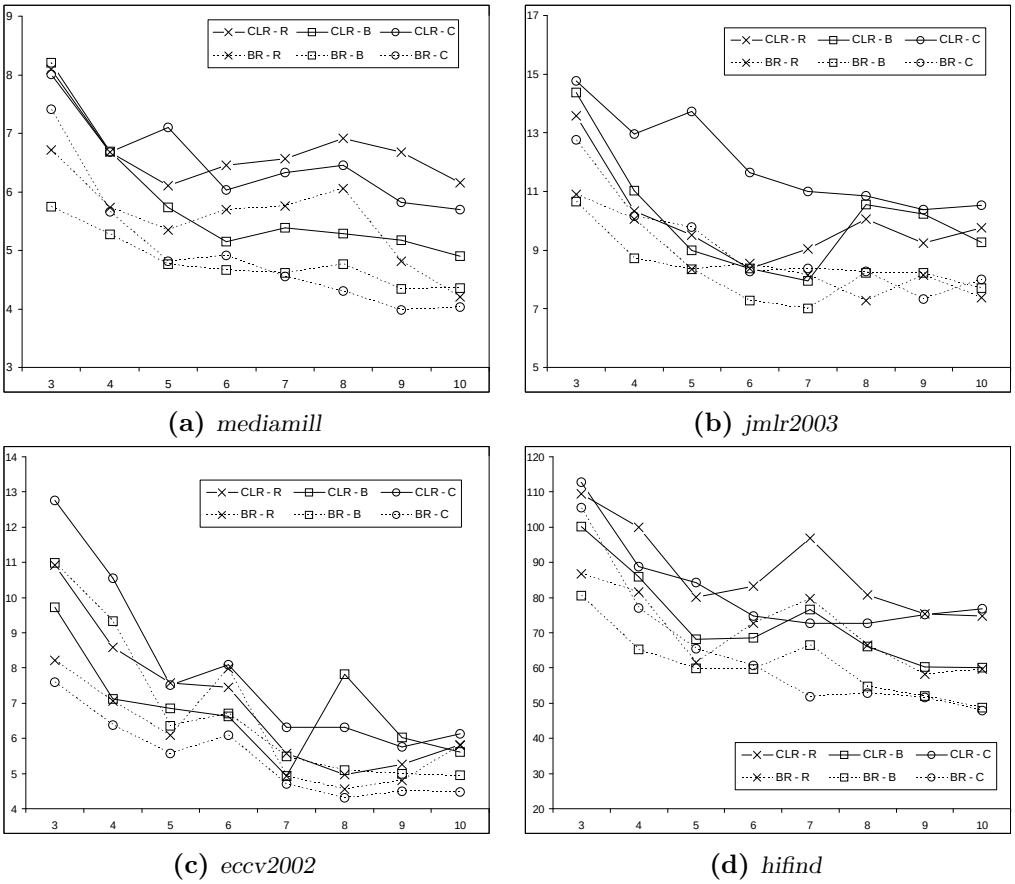
(c) *eccv2002* **(d)** *hifind*

Figure 9.4: Training time (in seconds) over number of partitions for the six HOMER variants

density datasets we notice an increase of F_1. We could therefore consider this as a guideline for selecting the number of partitions for HOMER with CLR based on the density of the dataset. Overall, the CLR based HOMER seems to be achieving better results for a larger percentage of different partition numbers, compared to the BR based HOMER. In terms of the partitioning method, the plain clustering approach seems inferior to the rest for both CLR and BR.

Training Time

Figure 9.4 shows the training time of the HOMER variants in seconds. We would expect that the training time of the random partitioning variant should be less than that of the balanced clustering variant, since they both deal with the same number of labels and create and train the same number of multilabel classifiers, but balanced clustering needs some additional time to distribute the labels according to similarity as well. However, this is clearly noticed only in *eccv2002*. In *jmlr2003* there is no clear winner for all numbers of partitions, while in *mediamill* and *hifind* we notice that



(a) *mediamill* **(b)** *jmlr2003*

(c) *eccv2002* **(d)** *hifind*

Figure 9.5: Testing time (in seconds) over number of partitions for the six HOMER variants

the balanced clustering approach requires less time, independently of the multilabel learning algorithm that is used (BR or CLR) and the number of partitions.

These results can be explained by the following observation. As clustering is based on the values of the labels, the children produced with balanced clustering, will contain labels that typically appear or do not appear together. This in turn means that more examples of the parent node will be filtered, leading to a reduced number of training examples. This was also observed in (Tsoumakas et al., 2008). Here, we notice that the gains in training time are higher for CLR compared to BR. This is an expected result based on the previous observation, because CLR trains its binary classifiers only on those examples where the values of the corresponding labels differ.

One issue that still needs to be explained is how this behavior is affected by the different datasets. In this direction, we notice that the gains in training time seem to be correlated with the density of the dataset. The reason, again based on the previous observation, is that the lower the number of label appearances with respect to the number of labels (density), the lower the gains that can be achieved by clustering co-occurring labels together.

Concerning the plain clustering partitioning method, we notice that it is clearly the worst one in terms of training apart from the *mediamill* dataset. The plain clustering method requires more time to perform the clustering as it is based on the expectation maximization algorithm. Dataset *mediamill* is also the smallest one, where it seems that the time required for clustering does not surpass the gains from the clustering process. This is why plain clustering appears to be better than random partitioning, especially for CLR. The loss in performance is more evident in *eccv2002* and *jmlr2003* due to the lower density.

Testing Time

Figure 9.5 on the previous page shows the testing times of the HOMER variants in seconds. Here the results are not as clear as in the case of the training time. Apart from the *jmlr2003* dataset, it seems that balanced clustering leads to less testing time compared to random partitioning irrespectively of the multilabel learning algorithm. Also plain clustering seems to be worse than the rest of the partitioning methods in *eccv2002* and *jmlr2003* for most of the partition numbers. Finally, we could comment that the classification time seems to decrease with respect to the number of partitions probably due the smaller height of the tree ($\log_\beta k$).

9.2.3 Comparison of HOMER against its base classifiers

For the direct comparison of HOMER against the flat approaches in Table 9.2 on the facing page and Table 9.3 on page 126 we chose the configuration with balanced clustering and 10 partitions. Note that no results could be retrieved for CLR on the *hifind* dataset due to the high memory requirements. To circumvent this problem for problems with a large number of classes was a main objective of combining HOMER with CLR as base classifier.

Prediction Quality

It is especially interesting to observe the opposite behavior in terms of recall and precision of the different approaches. CLR shows the best precision performance with a large margin over the other algorithms on all datasets. On the other hand, its recall values are particularly low. This confirms previous results that CLR does underestimate the size of the predicted labelsets (Fürnkranz et al., 2008). Our results indicate that this is particularly true for datasets with a high number of classes such as *eccv2002*, where CLR returns only 3.84 % of the correct labels, while 58.11 % of the returned labels are actually correct, compared to the 36.58 % by BR and around 28 % by both HOMERs. On the other hand, on the *mediamill* dataset, CLR's gain in precision seems to make up its low recall, thereby producing the highest average F1 value.

BR has a similar behavior of predicting relatively few labels with increasing number of labels. This is probably due to the greater imbalance of positive to negative examples for large problems, which leads to less frequent predictions of positive examples than the class distribution would suggest. HOMER shifts the trade-off between recall

Table 9.2: Performance measures on the different datasets. Results for Binary Relevance (BR), QWeighted calibrated label ranking (QCLR), HOMER with balanced clustering and 10 partitions and BR (H+BR), HOMER with balanced clustering and 10 partitions and QCLR (H+QCLR) are shown.

method	*mediamill*	*jmlr2003*	*eccv2002*	*hifind*
		`micro Precision`		
BR	58.00 %	32.27 %	36.58 %	59.43 %
QCLR	73.89 %	56.18 %	58.11 %	–
H+BR	56.98 %	26.48 %	28.91 %	55.31 %
H+QCLR	58.35 %	31.93 %	28.43 %	55.26 %
		`micro Recall`		
BR	44.79 %	9.85 %	7.42 %	45.73 %
QCLR	43.86 %	4.57 %	3.84 %	–
H+BR	44.91 %	10.81 %	13.21 %	48.64 %
H+QCLR	48.77 %	10.28 %	15.07 %	54.06 %
		`micro F1`		
BR	50.55 %	15.09 %	12.34 %	51.65 %
QCLR	55.04 %	8.45 %	7.21 %	–
H+BR	50.23 %	15.36 %	18.14 %	51.76 %
H+QCLR	53.13 %	15.55 %	19.70 %	54.65 %

and precision to a more balanced level, increasing recall but losing precision. The reason is probably the smaller problems in terms of number of classes that CLR has to solve in the HOMER setting. This was already shown in the correlative behavior between number of partitions and precision. The effect can also be seen when using BR as multilabel base classifier technique for HOMER, but it is less pronounced since the plain BR itself produces more balanced results.

Due to the great differences in recall and precision between the algorithms, we decided to omit the Hamming losses in Table 9.2 though this measure is usually used for evaluating multilabel algorithms, since Hamming loss generally favors algorithms with high precision and low recall,[6] which in this case means to favor CLR. The F1 measure, which returns the harmonic mean between recall and precision, allows a more commensurate analysis in this particular case since it penalizes greater differences to a higher degree. Except on the *mediamill* dataset, for which the approx. 100 classes do not show a great impact on CLR's recall, HOMER achieves the highest micro-averaged F1 value. In particular it outperforms BR on every dataset, which is especially interesting since HOMER is the direct competitor of BR in terms of computational costs. Similarly, HOMER+BR beats the plain BR except for *mediamill*, for which both algorithm are almost equal. HOMER+BR in general achieves less accurate predictions than using the pairwise approach as base classifier:

6 Returning zero labels to returning 50 of which 25 are correct would result in the same loss.

Table 9.3: Computational costs on the different datasets. Results for Binary Relevance (BR), QWeighted calibrated label ranking (QCLR), HOMER with balanced clustering and 10 partitions and BR (H+BR), HOMER with balanced clustering and 10 partitions and QCLR (H+QCLR) are shown.

method	*mediamill*	*jmlr2003*	*eccv2002*	*hifind*
		`Training Time (in seconds)`		
BR	2413.40	2801.17	2701.32	4179.66
QCLR	7423.19	6542.51	7460.14	–
H+BR	1065.21	1101.61	1144.47	2345.39
H+QCLR	1667.29	1871.00	1836.34	3801.53
		`Testing Time (in seconds)`		
BR	3.84	6.67	5.47	50.47
QCLR	103.59	119.28	154.65	–
H+BR	4.35	7.70	4.48	48.77
H+QCLR	4.90	9.26	5.62	60.02

in terms of F1 HOMER+BR is beaten on all datasets, in terms of recall and precision both algorithm are either almost equal (HOMER+BR slightly ahead) or HOMER+QCLR is clearly on top.

Computational Time

As shown in Table 9.3, HOMER is able to reduce the training time in comparison to plain BR approx. between 60 % and 44 % for using BR as base and between 33 % and 10 % for using QCLR. The first comparison is especially interesting since HOMER+BR has to train more base classifiers than BR: one classifier for each class at the leafs such as BR in addition to the classifiers in the inner nodes. However, this is done obviously with less training examples due to the filtering of examples at the inner nodes. Comparing the two HOMER variants, we can observe that the overhead of training the pairwise classifiers is always less than training the one-against-all classifiers. Note that QCLR has to train the same classifiers as BR for the comparison to the calibrating artificial class plus the pairwise classifiers between real classes. This may seem very surprising since training the pairwise classifiers requires $|P|/|D|$ times more training examples than training the BR classifiers[7], i.e. the amount is multiplied by the cardinality of the multilabel problem (cf. Fürnkranz et al., 2008). But when the base classifier has a super-linear complexity in terms of training examples, the reduced size of the binary subproblems by the pairwise approach may lead to a reduced complexity (Fürnkranz, 2002), which is the case for J48. In addition, another factor could be that by clustering the cardinality of the reduced multilabel problems

7 This estimation of training examples is too rough for the special case $|P|/|D| = 1$, which refers to a reduction to a multiclass problem. In this regard, the pairwise classifiers use in total fewer examples than the One-Against-All classifiers (Fürnkranz, 2002).

is often strongly reduced to the extreme case 1, where pairwise classifiers utilize in total fewer trainings examples than BR. However, we defer the investigation of this previously unseen observation for future work.

For testing, HOMER+BR is slightly slower than BR for the smaller *mediamill* and *jmlr2003*, but for the greater datasets *eccv2002* and *hifind* it requires less time. Although HOMER+BR has trained more base classifiers than the plain BR approach, it may invoke less base classifiers since great part of the classifier tree is pruned each time a meta-label is predicted as negative. This effect was already observed in previous work on a dataset with almost 1000 classes (Tsoumakas et al., 2008). HOMER+QCLR spends between 3 % and 40 % more time than BR, however, testing costs are so small for J48 compared to training time that this increase is almost not noticeable. Again, the overhead for evaluating the additional pairwise classifiers in HOMER+QCLR only require a small fraction of the time needed for the BR classifiers. Nevertheless it is not possible to simply compute the overhead as difference between the time for HOMER+BR and +QCLR since prediction accuracy, especially precision, also strongly influences the classification time. As expected, CLR requires the most computations for learning and predicting. However, the factor in training costs is proportional to the average labelset size per example, which makes the costs acceptable for most of the multilabel problems since the labelsets tend to be small. For prediction, the usage of QWeighted is able to considerably reduce the costs in comparison to the evaluation of all pairwise base classifier while maintaining the advantage of the pairwise approach in terms of predictive performance.

9.3 Conclusions

In this chapter, we performed an empirical study of the performance of HOMER. Compared to relevant previous work (Tsoumakas et al., 2008), the experimental part examines an additional multilabel learner for training each node of the hierarchy (QWeighted calibrated label ranking) on four large multilabel datasets with a variety of characteristics. Interestingly, the results showed that the instantiation of the multilabel learner of HOMER to QCLR can lead to better results compared to instantiating it to BR at a small expense in training and classification time. HOMER improves the training time of BR and this is even more important for QCLR. In terms of classification time HOMER substantially improves QCLR, while for BR the benefits appear for the two largest datasets in terms of number of labels. Except for the *mediamill* dataset (where the differences are rather small), HOMER managed to improve the performance of the base multilabel learner (both BR and QCLR).

HOMER also deals with the scalability problem of QCLR in terms of memory with respect to the number of labels, since it substantially reduces the amount of needed classifiers. In the same manner it provides a significant reduction in training and test time for the pairwise CLR methods. It is also shown that HOMER is able to equilibrate recall and precision, especially for QCLR which seems to underestimate the number of labels per instance for problems with a high number of labels.

10 Multilabel Classification with Label Constraints

Contents

A straight-forward approach for addressing multilabel classification is to model each class independently. In the binary relevance approach (cf. Section 7.3.1 on page 90), one binary classifier is trained for each possible label, in which all training examples for which the label is relevant are used as positives examples and all other examples as negative examples. However, in most real-world applications the predicted labels are not independent, so that the presence of one label may be indicative for other labels.

For this reason, several authors have extended the binary relevance approach to allow for incorporating dependencies between labels. For example, Crammer and Singer (2002a) have proposed a training scheme for a binary relevance classifier that does not optimize the 0/1-loss of each individual label, but instead optimizes a given ranking loss function over the entire one-against-all ensemble. Loza Mencía and Fürnkranz (2008c) have shown that this approach is outperformed by training a one-against-one ensemble, i.e., by having one classifier for each pair of labels.

In many applications, there are explicit constraints that must hold between the labels. For example, in the context of hierarchical classification, where the given set of labels has an inherent hierarchy structure[1], the relevance of one label in the hierarchy often also implies the relevance of all its ancestors. This situation can be modeled by a *subset constraint*, which specifies that whenever label λ_i is predicted as relevant, we must also predict λ_j. Similarly, one can imagine *exclusion constraints* specifying that

1 An example for a hierarchical classification problem is the considered task in (Lewis et al., 2004) (dataset RCV1), where news articles are classified into topics or labeled with relevant labels. Here, the existence of a hierarchy structure among labels is apparent and the used one can be viewed at `http://www.jmlr.org/papers/volume5/lewis04a/a02-orig-topics-hierarchy/rcv1.topics.hier.orig`

two labels λ_i and λ_j cannot be relevant at the same time. A typical example of this case would be if the set of labels contains labels of several orthogonal dimensions, each having a set of mutually exclusive labels, e.g., $L = \{\textit{male}, \textit{female}\} \cup \{\textit{child}, \textit{adult}\}$ and assuming, that the corresponding instances represent *single* persons.

In this chapter, we make two contributions: first, we will formally define the problem of multilabel learning with constraints and demonstrate the potential of this scenario on a simulated application with known constraints (Section 10.1). Second, we will evaluate an automated approach for discovering possible constraints on several well-known multilabel datasets (Section 10.2). Interestingly, we will see that in the automated approach, our results are mostly negative and cannot live up to the demonstrated potential on the artificial datasets.

10.1 Label Constraints

In this section, we describe the definition of constraints, and define straight-forward algorithms for correcting predictions that violate these constraints.

10.1.1 Definition of Label Constraints

In addition to the ordinary multilabel classification setting, we assume that we are given a set of *constraints* $Z = \{z_i \mid i = 1 \dots q\}$ on the labels L. Here, we consider two types of constraints: *subset* and *exclusion* constraints.

Subset constraints $\lambda_i \triangleright \lambda_j$ denote that if label λ_i is relevant for a given instance x than λ_j has to be also relevant. Formally,

$$\lambda_i \triangleright \lambda_j := \lambda_i \in P \rightarrow \lambda_j \in P \tag{10.1}$$

Exclusion constraints $\lambda_i \parallel \lambda_j$ denote that for all instances, labels λ_i and λ_j exclude each other, i.e., the two labels cannot be relevant or irrelevant at the same time. Formally,

$$\lambda_i \parallel \lambda_j := (\lambda_i \in P \leftrightarrow \lambda_j \in N) \wedge (\lambda_i \in N \leftrightarrow \lambda_j \in P) \tag{10.2}$$

We call subset or exclusion constraints *pairwise* if they have only one label on each side of their rule, and denote the space of all pairwise constraints for a given set of labels L as $\mathbb{Z}_2(L)$.

There are several other ways to define constraint types on labels for the multilabel setting. For example, one could also consider the following four types of constraints:

$$\begin{aligned}
\lambda_i \rhd \lambda_j &:= \lambda_i \in P \rightarrow \lambda_j \in P \\
\lambda_i \blacktriangleright \lambda_j &:= \lambda_i \in N \rightarrow \lambda_j \in N \\
\lambda_i \overline{\rhd} \lambda_j &:= \lambda_i \in P \rightarrow \lambda_j \in N \\
\lambda_i \overline{\blacktriangleright} \lambda_j &:= \lambda_i \in N \rightarrow \lambda_j \in P
\end{aligned}$$

Combined with logical connectors, these four basic constraints can represent a wide variety of constraints. For example, an exclusion constraint $\lambda_i \parallel \lambda_j$ may be viewed as a conjunction of the four constraints $(\lambda_i \overline{\rhd} \lambda_j) \wedge (\lambda_i \overline{\blacktriangleright} \lambda_j) \wedge (\lambda_j \overline{\rhd} \lambda_i) \wedge (\lambda_j \overline{\blacktriangleright} \lambda_i)$. But in this work we restrict ourselves to *subset* and *exclusion* constraints as a start, since they are intuitive and reasonably expressive.

Such constraints are quite similar to instance-level constraints that have been explored in semi-supervised or constraint-based clustering (Wagstaff and Cardie, 2000; Wagstaff et al., 2001; Bilenko et al., 2004), only that we define constraints between different labels (known groups of instances), whereas the constraints for semi-supervised clustering are defined between instances (e.g., this pair of instances must (not) belong to the same cluster).

In this chapter, we will often speak of preferences $pr_{i,j} \in [0, 1]$ instead of pairwise classifiers or pairwise predictions. The preference $pr_{i,j} = 1$ means that label λ_i is preferred over λ_j, also denoted as $\lambda_i \succ \lambda_j$, for a given test instance x (which will be often neglected). Conversely, the preference $pr_{i,j} = 0$ is interpreted as $\lambda_j \succ \lambda_i$.

10.1.2 Constraint-Based Correction of Predictions

Basically, label constraints can be integrated into the learning phase or testing phase of multilabel classification. Within the CLR framework (cf. Section 7.3.3 on page 93), pairwise subset constraints like $\lambda_i \rhd \lambda_j$ could be easily modeled in the learning phase, i.e. by substituting the pairwise preference $pr_{i,j}$ with a *constant* value of 1 or substituting its corresponding pairwise classifier $f_{i,j}$ with a constant function, which returns always 1. Therefore, this specific preference woulds always prefer λ_j and would not have to be learned anymore. However, this approach does not guarantee that the constraint is respected in the final prediction, because individual preferences may be over-ridden in the aggregation phase. Thus, we focus on integrating label constraints into the aggregation phase, where the predictions of the individual classifiers are combined.

Hence, we interpret the given constraints as immutable *hard* constraints, which must be respected by the final multilabel prediction. In addition, the predicted pairwise preferences are interpreted as *soft* constraints, which should be respected as well, but may be violated if necessary. These altering should be minimal for some distance measure. We consider two possible measures. First, the number of *preference swappings* that are needed to make the predicted preferences conform to the final prediction, and second, the number of *neighboring label swappings* in the predicted ranking. Our algorithm starts with an invalid predicted ranking and searches for a valid ranking which can be constructed by a minimal amount of preference or neighbor label swappings.

Minimizing Preference Swappings (PS)

The preference swappings measure is motivated by the assumption that an invalid ranking is caused by a few incorrectly predicted pairwise preferences. Errors among

Algorithm 9 PrefSwap

Require: Constraints Z, pairwise preferences PR, (invalid) ranking τ_0

$T_{\text{best}} \leftarrow \emptyset$
$T_{\text{all}} \leftarrow \{\tau_0\}$
$T_{\text{evaluated}} \leftarrow \{\tau_0\}$

repeat
  $T_{\text{new}} \leftarrow \emptyset$
  for each $\tau \in T_{\text{all}}$ **do** # expand new rankings
    for each $pr \in PR$ **do** # by iterating preferences
      $\tau_{\text{new}} \leftarrow \text{SWAPPREFERENCE}(\tau, pr)$
      if $\tau_{\text{new}} \notin T_{\text{evaluated}}$ **then** # enqueue only new rankings
        $T_{\text{new}} \leftarrow T_{\text{new}} \cup \{\tau_{\text{new}}\}$
  for each $\tau \in T_{\text{new}}$ **do** # check constraints
    if τ is valid **then**
      $T_{\text{best}} \leftarrow T_{\text{best}} \cup \{\tau\}$
  $T_{\text{evaluated}} \leftarrow T_{\text{evaluated}} \cup T_{\text{new}}$
  $T_{\text{all}} \leftarrow T_{\text{new}}$
until $T_{\text{best}} \neq \emptyset$ or $T_{\text{new}} = \emptyset$

return T_{best}

the pairwise base classifiers are assumed to be independent. Let $\mathbb{S}_k(Z)$ denote all permutations respectively rankings with $k = |L|$ which satisfy Z and let $a : \{0,1\}^n \to \mathbb{S}_k$ denote the aggregation function (here voting), which projects a set of preferences to a ranking. Then, for a given set of pairwise preferences PR, we are searching for a ranking $a(PR_1) \in \mathbb{S}_k(Z)$ of *corrected* preferences PR_1 which maximizes following measure:

$$d_{\text{PS}}(PR, PR_1) = |PR \cap PR_1|$$

Our implementation of finding a PS-minimal ranking is based on *breadth-first search*, and is presented as pseudocode in Algorithm 9. We start with an invalid predicted ranking τ_0. For every possible pairwise preference $pr \in \{pr_{i,j} \mid 1 \leq i < j \leq k\}$, the ranking τ_{new} is generated, which yields by swapping (negating) the preference, followed by voting-aggregation. Then, to avoid multiple checks of the same ranking, only new rankings are appended to T_{new}. After this expanding step, the candidate rankings $\tau \in T_{\text{new}}$ are checked if any satisfy the given constraints. If one ranking is determined as valid, the search process does not immediately stop but all remaining rankings in the set T_{new} will still be processed. We refer to this scheme in the further text as PS. Note that the elements in T_{new} represent rankings of the same level, the actual highest depth. So, all rankings $\tau \in T_{\text{new}}$, which satisfy the constraints,

are equal in terms of swapped preferences. If the PS-minimal ranking is not unique further selection criteria are evaluated, which are described later.

Minimizing Neighbor-Label Swappings (NLS)

Neighbor label swapping is motivated by the fact that swapping one preference yields at most to a swapping of two labels in the ranking, whose position difference is 1. We refer to these label pairs as neighboring or adjacent. However, many swappings of individual preferences will not yield a change in the predicted ranking. So as an approximation, one can use the needed swappings of neighbor or adjacent labels as a minimizing criteria.

In another view, minimizing NLS directly relates to one valid ranking with minimal RankLoss (cf. Section 7.2.2 on page 88) to the predicted ranking. Each NLS swaps an *adjacent* label pair, which increases the number of incorrectly ordered label pair by 1. The relative ordering of the remaining labels are not affected and label swappings resulting in a recurring (ranking) state will not be considered, as in Algorithm 9 on the preceding page.

If we denote $\tau_0 = (\lambda_1, \ldots, \lambda_k)$ as the predicted ranking, $S = (s_1, \ldots, s_z) \in \{1, \ldots, k-1\}^z$ as the set of non-empty finite sequences with terms in $\{1, \ldots, k-1\}$, $S_1 \in S$ as an arbitrary sequence of swap positions and $sw : \mathbb{S}_k \times S \to \mathbb{S}_k$ as a sequential swapping function

$$sw(\tau_0, S_1) := \begin{cases} (\lambda_1, \ldots, \lambda_{s_1-1}, \lambda_{s_1+1}, \lambda_{s_1}, \ldots, \lambda_k) & \text{if } |S_1| = 1 \\ sw(sw(\tau, s_1), (s_2, \ldots, s_{|S_1|})) & \text{otherwise} \end{cases}$$

then the sought ranking $\tau_1 \in \mathbb{S}_k(Z)$ should minimize following measure:

$$d_{\text{NLS}}(\tau_0, \tau_1) = \min\{z_1 \mid \exists S_1 \in S.|S_1| = z_1 \wedge \tau_1 = sw(\tau_0, S_1)\}$$

The algorithm to minimize Neighbor-Label Swappings (NLS) is a straight-forward adaption of PrefSwap, which iterates through all neighboring labels rather than through all pairwise preferences. Note that this scheme has a linear (in the number of labels) branching factor and PrefSwap a quadratic one.

Comparing and Tie-Breaking

Given $d_{\min} = \min_{\tau_i \in \mathbb{S}_k} d_{PS}(\tau_0, \tau_i)$ and several valid rankings τ_j with $d_{PS}(\tau_0, \tau_j) = d_{\min}$, we choose RankLoss as first criterion to further distinguish among them. This is consistent with the objective to minimize the changes of the initial invalid predicted ranking to satisfy some given constraints. NLS-minimized rankings omit this step, since they are by construction equal with respect to RankLoss.

There are cases in which this criterion is still equal for some rankings, see for example Figure 10.1 on the following page. Suppose the predicted ranking[2] is $\tau_0 = (AB|CD)$

2 We will use the notation $\tau_0 = (AB|CD)$, where the labels are ordered according to decreasing level of relevance (A is most relevant, D is least relevant), and the splitpoint between relevant and irrelevant labels is indicated with a "|".

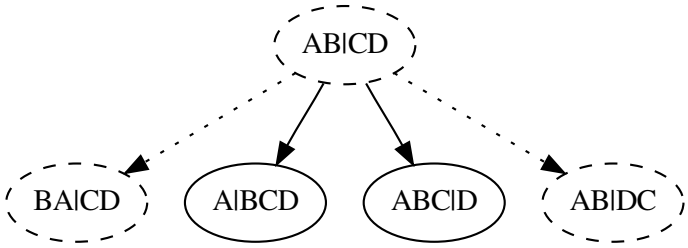


Figure 10.1: A simple example of constraint correction by neighbor label swapping. The predicted invalid ranking is $(AB|CD)$ and the constraint set consists of only one element: $B \rhd C$. NLS expands the initial ranking and returns two valid rankings. Dashed nodes represent invalid and solid nodes valid rankings.

and a domain expert has specified the constraint $z_1 = B \rhd C$. The ranking τ_0 does not satisfy c_1. It can be trivially repaired by swapping the position of the calibration label $|$ with one of its neighbors B or C, yielding the $\tau_1 = (A|BCD)$ and $\tau_2 = (ABC|D)$. Both are equal with respect to the NLS distance. Two other rankings, $(BA|CD)$ and $(AB|DC)$, can also be found at the same search depth, but these are invalid.

In order to decide for one of the two valid rankings, we first compute the RankLoss with respect to the originally predicted ranking τ_0. If this is also equal between the candidates (as in our example), we check whether the direct neighbors of the predicting split point violate the initial pairwise preferences. Let $s_p = \tau_i(\lambda_0)$ be the position of the splitpoint within a ranking τ_i, then we compute:

$$\left|P \cap \left\{\lambda_0 \succ \tau_i^{-1}(s_p - 1),\ \lambda_0 \prec \tau_i^{-1}(s_p + 1)\right\}\right|$$

So in other words, we count the number of wrongly ordered neighbor label pairs which are direct above or below the splitpoint. We select the one with the lowest number. Then if there are still ambiguous rankings, we select the one which minimizes the disordered number for all $k-1$ neighbor label pairs, not only the direct neighbors of the splitpoint. As a last separation step, a random selection is applied.

10.1.3 Experimental Evaluation

In the following, we show the results of the PS and NLS algorithms on artificial data. Note, that we reverted in the following evaluations AvgPrec, to bring this loss in line with the others so that an optimal value is 0.

Table 10.1: Experiments on synthetic data generated with constraints Z_1. The shown measures are macro-averaged and the lower the value the better the performance.

ERROR	RANKERR			RANKLOSS			1 − AVGPREC			# VIOL.
	VA	PS	NLS	VA	PS	NLS	VA	PS	NLS	
0.05	0.025	**0.024**	0.026	0.009	**0.008**	0.009	0.008	**0.007**	0.007	0.10
0.10	0.053	**0.052**	0.054	0.028	**0.028**	0.028	0.025	0.025	**0.024**	0.17
0.15	**0.085**	0.086	0.088	0.055	**0.053**	0.053	0.043	0.041	**0.041**	0.23
0.20	**0.125**	0.126	0.128	0.093	**0.091**	0.091	0.072	**0.070**	0.070	0.27
0.25	**0.168**	0.169	0.171	0.135	0.133	**0.133**	0.100	0.097	**0.096**	0.31
0.30	**0.227**	0.227	0.228	0.208	0.206	**0.206**	0.144	0.142	**0.140**	0.34

Table 10.2: Experiments on synthetic data generated with constraints Z_2. The shown measures are macro-averaged and the lower the value the better the performance.

ERROR	RANKERR			RANKLOSS			1 − AVGPREC			# VIOL.
	VA	PS	NLS	VA	PS	NLS	VA	PS	NLS	
0.05	0.023	**0.022**	0.022	0.008	0.007	**0.007**	0.008	0.007	**0.007**	0.06
0.10	0.052	0.050	**0.049**	0.028	0.027	**0.026**	0.028	0.027	**0.026**	0.13
0.15	0.085	0.082	**0.082**	0.061	0.057	**0.057**	0.055	0.053	**0.052**	0.19
0.20	0.123	0.119	**0.118**	0.101	0.097	**0.096**	0.083	0.081	**0.080**	0.24
0.25	0.169	0.163	**0.163**	0.156	0.149	**0.148**	0.118	0.114	**0.114**	0.29
0.30	0.223	0.217	**0.215**	0.219	0.213	**0.211**	0.159	0.157	**0.155**	0.34

Data Generation

Given a set of labels $L = \{\lambda_1 \dots \lambda_k\}$ and a set of pairwise label constraints $Z \subseteq \mathbb{Z}_2(L)$, n random permutations $\tau_1, \dots, \tau_n \in \mathbb{S}_k$ are generated, which satisfy Z. Each of the permutations τ_i is decomposed into the unique set $PR = \{pr_{i,j} \mid 1 \leq i < j \leq k\}$ of binary pairwise preferences. For example, if $\tau = (\lambda_1, \lambda_3, \lambda_2)$ then $PR = \{\lambda_1 \succ \lambda_2,\ \lambda_1 \succ \lambda_3,\ \lambda_3 \succ \lambda_2\}$ is the associated set of binary pairwise preferences. The classification error of the binary pairwise classifiers is modeled by swapping a ratio (ERROR = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3) of the pairwise preferences.

Evaluation

In a first experiment, we used the following two arbitrarily chosen constraint sets on 6 labels $L = \{\lambda_1, \dots, \lambda_6\}$ and generated $n = 5000$ training instances for each.

$$Z_1 = \{\lambda_1 \triangleright \lambda_2,\ \lambda_3 \triangleright \lambda_4,\ \lambda_4 \triangleright \lambda_5,\ \lambda_6 \triangleright \lambda_5\}$$
$$Z_2 = \{\lambda_1 \parallel \lambda_2,\ \lambda_2 \parallel \lambda_5,\ \lambda_3 \triangleright \lambda_6\}$$

Tables 10.1 and 10.2 show the results of the comparison between regular voting aggregation (VA), and the constraint-based corrections PS and NLS. For each loss function the values in the leftmost column are generated by voting-aggregation without

Table 10.3: Experiments on 100 random synthetic datasets. For each loss function the left values are generated by ordinary voting-aggregation. The right values show constraint-correction values based on neighbor-label swapping.

ERROR	RANKERR		RANKLOSS		MARGIN		1 − AVGPREC		# VIOL.
	VA	NLS	VA	NLS	VA	NLS	VA	NLS	
0.05	0.0245	**0.0231**	0.0100	**0.0081**	0.0120	**0.0098**	0.0074	**0.0065**	0.11
0.10	0.0525	**0.0494**	0.0311	**0.0260**	0.0372	**0.0311**	0.0223	**0.0196**	0.20
0.15	0.0857	**0.0811**	0.0652	**0.0566**	0.0774	**0.0670**	0.0449	**0.0402**	0.27
0.20	0.1266	**0.1206**	0.1124	**0.1009**	0.1320	**0.1181**	0.0750	**0.0686**	0.33
0.25	0.1733	**0.1662**	0.1709	**0.1577**	0.1979	**0.1816**	0.1098	**0.1024**	0.37
0.30	0.2276	**0.2203**	0.2434	**0.2294**	0.2763	**0.2586**	0.1519	**0.1439**	0.41

any constraint-based post-correction. The second and third column show constraint-correction values based on preference swapping and neighbor label swapping. The bold numbers describe the best values for a particular loss and ERROR combination. MARGIN error values are omitted for lack of space. Their relations among the different aggregations schemes are anyway mostly identical to the RANKLOSS values, more precisely, the aggregation scheme with the best MARGIN value for a particular ERROR is identical to the best one for RANKLOSS. For both set of constraints Z_1 and Z_2, PS or NLS tend to outperform VA, but the results are not entirely conclusive.

To obtain a more thorough evaluation, we used 100 datasets with random rankings for 6 labels, each with 1000 instances. The number of constraints was also randomly selected from 2 to 5. These constraints were first checked for consistency and finally evaluated for six ERROR values ($\epsilon = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3$). The average losses and ratios of violated instances are shown in Table 10.3.[3] In this evaluation, only NLS was used as correction scheme, since its evaluation takes significantly less time and its performance seem to be competitive to PS. The values clearly show the superior performance of NLS-minimizing constraint correction compared to simple voting-aggregation. For each ERROR - loss function combination NLS outperforms the baseline.

10.2 Discovering Label Constraints from Data

In many domains, sensible label constraints may be available from background knowledge about the target domain. However, even in domains in which such knowledge is not readily available, one may try to automatically discover the knowledge from data. In this section, we evaluate the use of *association rule learning* algorithms for this purpose.

3 Note that the experimental results in this chapter correct minor erroneous results previously published in (Park and Fürnkranz, 2008). A recently found software bug employed a false normalization for RANKLOSS. However, the original statements and conclusions were not affected by this bug. Please note also that in this thesis all predictive measures are normalized (cf. Section 7.2 on page 86) unlike in (Park and Fürnkranz, 2008).

10.2.1 Association Rules as Constraints

We define the problem of discovering label constraints in the data as an *association rule learning* problem. In a nutshell, in this field each instance (here called itemset or transaction) represents a subset of a fixed set I of items or objects and the main task is to find rules or patterns of the form:

> **if** item(s) $I_1 \subset I$ is present in an instance, **then** item(s) $I_2 \subset I$ is also present

with $I_1 \cap I_2 = \emptyset$ and $I_2 \neq \emptyset$. These rules, which make a statement about the relation of items, are typically rated by its *confidence* (how often the rule is true) and *support* (how often the precondition holds or, in other words, how general it is). The support of a rule $I_1 \rightarrow I_2$ is traditionally defined as the fraction of itemsets in the data, in which $I_1 \cup I_2$ occurs (Agrawal et al., 1996). But it may also refer to the fraction of itemsets in which only the precondition I_1 holds (Borgelt and Kruse, 2002). Here, we will use the latter definition. For further information on association rule learning and its related field *frequent itemset mining*, we refer to (Goethals, 2005).

The similarity to our task at hand is apparent and so, it is natural to use the machinery of this field to solve our problem. We construct one itemset for each training example x_i, which consists of the set of relevant labels P_i. We then use a association rule learner to discover rules of the form

$$\lambda_{i_1} \dots \lambda_{i_b} \rightarrow \lambda_j$$

with b labels in the antecedent and one label in the consequent. Negation can be handled by including negative labels of the form $-\lambda'$ with the semantic $\lambda' \in N$ into the itemsets. Thus, each example is associated with an itemset of length k, one item for each label denoted either as λ' or $-\lambda'$.

Typical association rule learning algorithms tend to generate redundant rules. These are justified in their original main application areas, e.g., market basket analysis, since their main goal is to find (all) interesting rules or relations between items rather than a compact set of rules. However, for our purpose, to use association rules as constraints, these redundant rules lead to unnecessary runtime growth. In this work we understand redundancy in the sense of inductive rule learning. We are thus interested in generating rules with minimal antecedent, as opposed to, e.g., closed itemset mining which considers rules with maximal antecedent (Goethals, 2005).

A rule $I_1 \rightarrow I_2$ consisting of body (antecedent) I_1 and head (consequent) I_2 is *redundant* with respect to rule $I_3 \rightarrow I_2$ if I_3 is a subset of I_1. If a rule is more *specific* than another, it is unnecessary to check, because the more general rule will be checked in any case. So in our evaluations we speed up the constraint correction process, by post-processing generated association rules with a *minimizing* step, which removes all rules except the most general ones. In the above example, if $I_3 \subseteq I_1$, then the rule $I_1 \rightarrow I_2$ will be removed.

Table 10.4: Experiments on real-world data: *yeast*. The right-most column shows the amount and the ratio of predicted rankings which the violated given constraint set.

Conf	Supp	RankLoss	Margin	1 − AvgPrec	# violated
VA		0.4614	0.3349	0.2426	
100	60	0.4614	0.3349	0.2426	28 (0.03)
	40	0.4612	0.3347	0.2425	102 (0.11)
	20	0.4620	0.3350	0.2430	303 (0.33)
95	60	0.4614	0.3349	0.2426	39 (0.04)
	40	0.4612	0.3345	0.2425	111 (0.12)
	20	0.4619	0.3345	0.2429	341 (0.37)
90	60	0.4614	0.3349	0.2426	40 (0.04)
	40	0.4612	0.3345	0.2426	174 (0.19)
	20	-	-	-	-

10.2.2 Experiments on Real-World Data

We compare simple voting aggregation and the constraint correction algorithm on two real-world multilabel datasets, namely *yeast* and *siam*.[4] The dataset *yeast* consists of 14 labels, 1500 training and 917 testing instances. It concerns the functional multilabel classification of yeast genes (cf. Section 8.4.1 on page 103). Dataset *siam* is a text-categorization problem, where multiple labels are associated to one document. It consists of 22 labels, 21519 training and 7077 testing data. We used the given training/test splits for evaluation.

The association rules were generated by the APriori algorithm (Agrawal et al., 1996) in its implementation by Borgelt (Borgelt, 2003). As a base learner, we used the support-vector machine implementation in LIBSVM (Chang and Lin, 2011) with a linear kernel in its default settings. The algorithms were compared according to the same metrics as above, except that we cannot give RankErr, since we did not have correct rankings of the datasets to compute this loss function.

Table 10.4 shows the result of the evaluation on the *yeast* dataset. The values in the first line represent performance values for aggregation of pairwise preferences by voting, which is used as our baseline. The next lines, beginning with various minimum confidence and support values, describe the result of NLS constraint correction with different sets of constraints, which are generated by association rule learning using stated parameters on the training data.

The last column of Table 10.4 describes the amount of violated instances, and therefore the number of instances to which constraint correction was applied. In all other cases, the predicted ranking was not changed. APriori with parameters Conf = 90 and Supp = 20 generated inconsistent rules, so no corresponding values are shown.

4 Available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html

Table 10.5: Experiments on real-world data: *siam*. The right-most column shows the amount and the ratio of predicted rankings which violated the given constraint set.

Conf	Supp	RankLoss	Margin	1 − AvgPrec	# violated
VA		0.0784	0.0759	0.1920	
100	60-20	0.0784	0.0759	0.1920	2 (0.00)
95	90-70	0.0790	0.0765	0.1967	1157 (0.16)
90	95	0.0789	0.0764	0.1958	768 (0.11)
	90	0.0791	0.0766	0.1977	1926 (0.27)
	85	0.0791	0.0766	0.1969	2205 (0.31)
	80-70	0.0793	0.0769	0.1985	2609 (0.37)

As one can see, constraint correction with association rules as constraints does not cause significant changes in the performance of multilabel classification. Even in cases where a considerable amount of instances had to be post-processed, for example Conf = 95 and Supp = 20, where 37 % of the predicted rankings violated some of the learned constraints, no real difference to the baseline can be observed. The results for *siam* (Table 10.5) even show a consistent deterioration in prediction performance, i.e., for all applications of constraint correction the evaluated losses are worse or equal than the baseline.

Some performance values for *siam* in Table 10.5 are identical for different support values with the same confidence, i.e. Conf = 100, Supp = 50 and Conf = 100, Supp = 30. This is caused by the fact, that identical association rules were generated for these parameters. More information regarding the used association rules as constraints can be seen in Tables 10.6 and 10.7, which show the number of generated constraints for the varying confidence and support values. In addition, the rightmost column shows the number of rules, which survived our crude redundancy filter, and were (as previously described) actually used in the constraint testing process.

10.3 Discussion

We introduced constraints into the multilabel classification setting, and studied two machine learning tasks in this context:

1. Integration of additional knowledge in form of label constraints into the multilabel classification setting

2. Automatically learning of label constraints

Regarding the first point, we experimented with two approaches which tackle the constraint integration problem by transforming it into a search problem - searching for a valid ranking with minimal distance from the ordinary predictions. The number of preference swappings (PS) and the number of neighbor-label swappings (NLS)

Table 10.6: Constraint Generation: *yeast*

Conf	Supp	# rules	# min
100	60	65	8
	40	735	11
	20	11321	46
95	60	245	21
	40	2067	33
	20	27042	99
90	60	305	31
	40	2398	44
	20	31708	127

Table 10.7: Constraint Generation: *siam*

Conf	Supp	# rules	# min
100	60	8	1
	50	2957	3
	40	35041	3
	30	168882	3
	20	466284	6
95	90	2296	143
	80	52204	191
	70	324442	198
90	95	109	70
	90	2416	182
	85	15905	239
	80	61652	273
	75	178920	281
	70	415861	288

seem to be intuitive and reasonable choices as distance functions within the CLR framework. Although empirical evaluations of PS and NLS on artificial datasets showed an improvement for multilabel classification, it failed for two commonly used real-world datasets, where we used automatically discovered constraints.

In our view, several points could be the reason for the negative results. At first, one could criticize that we had given the correct constraints for the artificial datasets, which was not the case for the real-world datasets. One is that the introduced setting with given true constraints may be too idealistic. Indeed, for our two evaluated real-world datasets, we have no evidence, even for rules with Conf = 100 that these rules hold for all instances from the true distribution, since the rules were generated on training data, which might differ from the true distribution. Small tests with association rules generated with parameters Conf = 100, Supp = 1 on training and testdata of *yeast* showed also no improvement.

Another point is that the artificial data was explicitly modeled by voting *de-aggregation*, i.e. given transitive (binary) pairwise preferences, the correct calibrated label-ranking is uniquely defined and vice-versa (if we exclude ties). But pairwise preferences in general do not have to be transitive.

Besides the failure on real-world data, we are aware that the shown algorithms are currently not applicable to practical problems. We perform an essentially exhaustive breadth-first search through all possible rankings, and also use a rather expensive pruning step for the association rule discovery. Without strong assumptions, i.e. that a valid ranking is relatively fast reachable by PS or NLS for an invalid ranking, the search process takes too long, since the number of possible candidates grows exponentially for each iteration of the search algorithm.

However, our main goal was to investigate whether this approach can, in principle, yield improved results. Despite the negative results with automatically discovered constraints, we nevertheless interpret our results as informative, and plan a deeper investigation of this learning scenario.

11 Summary

11.1 Summary

In this thesis, we presented several works focused on improving decomposition-based multiclass and multilabel classification, which are briefly summarized in the following.

Multiclass Classification

For multiclass classification, two methods for a faster *training* phase were shown. First, a general reduction of computational complexity was achieved by exploiting code redundancies in ECOC-based binary decomposition methods. The minimization of the corresponding learn redundancies was posed as a scheduling problem, which is related to the Steiner Tree Problem. An approximate solution for this problem was applied and yielded a positive result with respect to efficiency. The scheduling approach is applicable by incremental learners, but we showed also a promising adaptation based on adapted caching and weight reusing for the genuine batch learner SVM. Second, for the combination of ECOC and the base learner Naïve Bayes, we showed a tight alternative computation scheme, which significantly reduces the training effort from $O(n \cdot t \cdot g)$ to $O((n+t) \cdot g)$ using normal and discrete density estimation methods, where n, t and g are the number of classifiers, instances and features. For the case of kernel density estimators, the worst-case complexity is unchanged, but we advert in this case to the relationship of actual training complexity and the number of distinct feature values: The lower the number of feature values, the higher the possible reduction. Empirical support was given, and based on the majority of real-world datasets which, in our experience, typically exhibit such a low diversity, we expect a reduction even in this case.

Furthermore, a chapter was devoted on a more efficient *prediction* or *testing* phase for ECOC-based multiclass classification. The works on QWeighted from pairwise classification were generalized to arbitrary binary decompositions within the ECOC-framework resulting in the algorithm QuickECOC. The underlying basic idea, that it is not necessary to evaluate all classifiers to compute the classification prediction, holds also for the ECOC setting and was exploited similarly. The new degree of freedom regarding the selection of the next classifier was handled using a simple score based scheme, which prefers, roughly speaking, the classifier with the current top class against the highest number of other highly ranked classes. Using this heuristic, QuickECOC proceeds identically to QWeighted in the special case of pairwise codes. Extensive experimental evaluations showed the successful reduction of classifier evaluations on various decoding schemes and code types.

Multilabel Classification

QWeighted was also similarly adopted to the multilabel classification setting. Within the calibrated label ranking framework, the number of classifier evaluations were reduced by evaluating not more than a necessary number of classifiers. Here, after the initial evaluation of all incident classifiers of the artificial label, the algorithm QCLR proceeds to compute labels exceeding the relevance threshold (given by the voting mass of the artificial label). The reduction lies basically in *deliberately* neglecting further classifier evaluations of these labels for the purpose of estimating by which *amount* they exceed it. After all, this information is irrelevant for the multilabel prediction. Since the typical number of relevant labels in real-world multilabel problems is relatively small compared to the set of labels, this procedure can result in a significant reduction of the overall prediction complexity, as observed in our experimental evaluations.

Though we were able to reduce the computational complexity in the classification phase, the problem of maintaining a quadratic number of classifiers in number of the labels in *memory* remained. This poses a serious problem in context of large-scale multilabel problems with thousands of labels. For instance, the dataset *EUR-Lex* turned out to be an infeasible problem with our available computational resources. To tackle this problem, we experimented with a combination of HOMER and QCLR. HOMER transforms the original multilabel problem into a set of smaller multilabel problems in terms of number of labels and organizes them in a hierarchy. The decomposition to smaller problems significantly reduces the amount of classifiers and therefore the memory-complexity. But, this approach had also for the first-time an impact on the predictive performance among the approaches considered in this theses. Until then, all developed methods were able to improve the efficiency without affecting the predictive performance. However, it turned out that the combination lead also to a predictive improvement compared to QCLR. The experimental evaluations indicate that HOMER resolves the bias of QCLR towards precision such that a more balanced tradeoff between precision and recall is achieved.

Chapter 10 is different than the previous chapters. Here, we extended the multilabel classification setting with label constraints in hope to improve the predictive performance. We elaborated on label dependencies in common multilabel data, defined some constraint types and developed a prototype algorithm to incorporate them in the learning process. Within the decomposition-based calibrated label ranking approach for multilabel classification, we followed the assumption, that minimizing violations of such label constraints might improve the prediction quality. For this purpose, we considered the number of prediction or preference disagreements of base classifiers and the number of neighbored label swappings to fulfill some constraints as the minimizing objective. Furthermore, we tackled the learning of label constraints as an association rule learning problem and used this methods for the considered real-world datasets. Though first experiments on artificial data showed promising results, the evaluations on two real-world datasets were negative. We discussed about the possible reasons, which were mostly of methodical nature.

11.2 Outlook

The exploitation of code-redundancies by employing a (pseudo) minimal redundant training schedule considered only incremental learners. One could also consider to incorporate in addition decremental learners (cf., e.g., Cauwenberghs and Poggio, 2000). This would clearly make a more flexible training schedule possible. However, the complexity of the training graph would be further increased and the design of a practical algorithm for computing an adequate approximate Steiner Tree in this setting might be a challenging task.

For the methods based on QWeighted, there might be still some potential for improving the results with better heuristics for the selection of the next classifier. We have not yet thoroughly explored this parameter. For example, one could try to adapt ideas from *active learning* for this process. Nevertheless, however, we do not necessarily expect a high gain. Furthermore, we consider an in-depth analysis of existing fast decoding methods in coding theory, and the investigation of the transferability to the multiclass or multilabel classification setting as promising directions for future work.

A restriction of our QWeighted-based approaches is that they are only applicable to methods which combine predictions via voting or weighted voting and some compatible ones (cf. Section 6.1.5 on page 60). There are various other proposals for combining the class probability estimates of the base classifiers into an overall class probability distribution (in the case of pairwise classification, e.g., *pairwise coupling* (Hastie and Tibshirani, 1997; Wu et al., 2004)). Efficient methods for these alternative aggregation schemes poses another interesting topic for further research.

Furthermore, we are still anxious to continue our work on the learning of label constraints and the incorporation of such information into the learning process. The addressed topic remains interesting and it is for us not surprising that the related topic of label dependencies attracted many researchers (cf., e.g., Read et al., 2011; Zhang and Zhang, 2010; Dembczynski et al., 2010) in the meantime. We are still unsure about the failure of our prototype algorithms. It is unclear, if our approach for discovering label constraints by using association rule learning produced poor rules (e.g., we did not manually validate the constraints) or if the typical learning process entails such relations implicitly, so that we do not observe any differences regarding the predictive performance. It would be nice, if a further more thorough analysis could clarify these issues.

Besides these more or less concrete points for future work, an interesting general direction in this context are hashing or compression techniques. Recently, several works successfully utilized related concepts within machine learning in various ways. Besides the previously mentioned work of Hsu et al. (2009b) (cf. Section 6.4 on page 81) another interesting work is done by Lin et al. (2010, and related literature), which considers large-scale databases with a high number of features. The computational complexity of Nearest Neighbor on such datasets is significantly reduced by utilizing particular hash functions. The so-called *similarity-preserving* property of these functions make the otherwise contrary combination amenable.

Acknowledgments

This thesis was done at the Knowledge Engineering Group at the TU Darmstadt generously supported by the *German Science Foundation (DFG)*. In addition, without the computational resources which were provided from the *Frankfurt Center for Scientific Computing*, some of the computationally intensive experimental studies would not have been possible.

Foremost, I would like to thank my advisor Prof. Dr. Johannes Fürnkranz for his constant support and instruction over all these years. The numerous insightful discussions with him were very helpful and kept me going on. But above all, I am deeply thankful, that he has given me the opportunity to do a doctorate and, actually, that he has made me become aware, that this path might be a viable option for me.

I also would like to thank Prof. Dr. Eyke Hüllermeier for his many helpful suggestions during this time. His revealing comments often helped to greatly improve our work. In fact, a great deal of the main ideas behind the QWeighted algorithm is owed to him, which was in turn the basis for some follow-up publications.

I am thankful to Prof. Dr. Jan Peters, Prof. Dr. Stefan Katzenbeisser and Prof. Stefan Roth, Ph.D. for agreeing to be part of the board of examiners on relatively short notice.

Many thanks go to my collaborators Eneldo Loza Mencía, Dr. Grigorios Tsoumakas, Immanuel Schweizer, Dr. Ioannis Katakis, Kamill Panitzek, Lorenz Weizsäcker and Weiwei Cheng, with whom I had the pleasure to work with and to learn from. Besides, I have to thank Alexander Galitzki, Alexander Vitanyi, George-Petru Ciordas-Fanghäuser, Marian Wieczorek, Sandra Ebert and Tobias Plötz for their implementation support regarding the LPCforSOS framework and more.

I was very fortunate to work in a creative, supportive and (sometimes too) lively environment. My colleagues (besides the already mentioned ones) Dirk große Osterhues, Frederik Janssen, Gabriele Ploch, Dr. Heiko Paulheim, Jan-Nikolas Sulzmann and Kilian Kiekenap of the Knowledge Engineering Group provided a pleasant and relaxed atmosphere, for which I am grateful.

Finally, I would like to express my gratitude to my parents Chang-Hun and Sun-Lim Park and to my brother Dr. Sang-Min Park for always supporting me. I am particularly thankful to my brother for proof-reading early draft versions of this thesis.

Bibliography

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1996). Fast discovery of association rules. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. MIT Press.

Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141.

Angulo, C., Ruiz, F., González, L., and Ortega, J. A. (2006). Multi-classification by using tri-class svm. *Neural Processing Letters*, 23(1):89–101.

Asuncion, A. and Newman, D. (2010). UCI machine learning repository. Repository available at `http://archive.ics.uci.edu/ml`.

Banerjee, A. and Ghosh, J. (2006). Scalable clustering algorithms with balancing constraints. *Data Mining and Knowledge Discovery*, 13(3):365–395.

Barnard, K., Duygulu, P., de Freitas, N., Forsyth, D., Blei, D., and Jordan, M. I. (2003). Matching words and pictures. *Journal of Machine Learning Research*, 3:1107–1135.

Berger, A. (1999). Error-correcting output coding for text classification. In *Proceedings of IJCAI-99 Workshop on Machine Learning for Information Filtering (IJCAI99-MLIF, Stockholm, Sweden)*.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604. Software available at `http://sourceforge.net/projects/moa-datastream/`.

Bilenko, M., Basu, S., and Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In Brodley, C. E., editor, *Proceedings of the 21st International Conference on Machine Learning (ICML 2004, Banff, AB, Canada)*. ACM.

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

Blockeel, H. and Struyf, J. (2003). Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:621–650.

Borgelt, C. (2003). Efficient implementations of Apriori and Eclat. In *Proceedings of the 1st Workshop of Frequent Item Set Mining Implementations (FIMI-03, Melbourne, FL, USA)*.

Borgelt, C. and Kruse, R. (2002). Induction of association rules: Apriori implementation. In Härdle, W. and Rönz, B., editors, *Proceedings of the 15th International Conference on Computational Statistics (Compstat 2002, Berlin, Germany)*, pages 395–400. Physica-Verlag.

Bose, R. C. and Ray-Chaudhuri, D. K. (1960). On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79.

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

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Brenner, S. E., Koehl, P., and Levitt, M. (2000). The astral compendium for protein structure and sequence analysis. *Nucleic Acids Research*, 28(1):254–256.

Brinker, K., Fürnkranz, J., and Hüllermeier, E. (2006). A unified model for multilabel classification and ranking. In Brewka, G., Coradeshci, S., Perini, A., and Traverso, P., editors, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006, Riva del Garda, Italy)*, pages 489–493. IOS Press.

Cardoso, J. S. and da Costa, J. F. P. (2007). Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research*, 8:1393–1429.

Cauwenberghs, G. and Poggio, T. (2000). Incremental and decremental support vector machine learning. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Proceedings of the 14th Annual Conference on Neural Information Processing Systems (NIPS 2000, Denver, CO, USA)*, pages 409–415. MIT Press.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cohen, W. W. (1995). Fast effective rule induction. In Prieditis, A. and Russell, S. J., editors, *Proceedings of the 12th International Conference on Machine Learning (ICML 1995, Tahoe City, CA, USA)*, pages 115–123. Morgan Kaufmann.

Cohen, W. W. and Singer, Y. (1999). A simple, fast, and effictive rule learner. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999, Orlando, FL, USA)*, pages 335–342. MIT Press.

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

Crammer, K. and Singer, Y. (2002a). A new family of online algorithms for category ranking. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002, Tampere, Finland)*, pages 151–158. ACM.

Crammer, K. and Singer, Y. (2002b). On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233.

Crammer, K. and Singer, Y. (2003). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3(6):1025–1058.

Cutzu, F. (2003a). How to do multi-way classification with two-way classifiers. In Kaynak, O., Alpaydin, E., Oja, E., and Xu, L., editors, *Proceedings of the Joint International Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP 2003, Istanbul, Turkey)*, pages 375–384. Springer.

Cutzu, F. (2003b). Polychotomous classification with pairwise classifiers: A new voting principle. In Windeatt, T. and Roli, F., editors, *Proceedings of the 4th International Workshop on Multiple Classifier Systems (MCS 2003, Guilford, UK)*, pages 115–124. Springer.

Dembczynski, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2010). On label dependence in multi-label classification. In Zhang, M.-L., Tsoumakas, G., and Zhou, Z.-H., editors, *Proceedings of the 2nd International Workshop on Learning from Multi-Label Data (MLD'10, Haifa, Israel)*, pages 5–12.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge discovery and data mining (KDD 2000, Boston, MA, USA)*, pages 71–80. ACM.

Duygulu, P., Barnard, K., de Freitas, N., and Forsyth, D. (2002). Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Proceedings of the 7th European Conference on Computer Vision (ECCV 2002, Copenhagen, Denmark), Part IV*, pages 97–112. Springer.

Escalera, S., Pujol, O., and Radeva, P. (2006). Decoding of ternary error correcting output codes. In Trinidad, J. F. M., Carrasco-Ochoa, J. A., and Kittler, J., editors, *Proceedings of the 11th Iberoamerican Congress in Pattern Recognition (CIARP 2006, Cancun, Mexico)*, pages 753–763. Springer.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305.

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

Fürnkranz, J. (2002). Round robin classification. *Journal of Machine Learning Research*, 2:721–747.

Fürnkranz, J. (2003). Round robin ensembles. *Intelligent Data Analysis*, 7(5):385–403.

Fürnkranz, J., Hüllermeier, E., Mencía, E. L., and Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153.

Galar, M., Fernández, A., Tartas, E. B., Sola, H. B., and Herrera, F. (2011). An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776.

Gallager, R. G. (1968). *Information Theory and Reliable Communication*. Wiley.

Ghani, R. (2000). Using error-correcting codes for text classification. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000, Stanford, CA, USA)*, pages 303–310. Morgan Kaufmann.

Goethals, B. (2005). Frequent set mining. In Maimon, O. and Rokach, L., editors, *The Data Mining and Knowledge Discovery Handbook*, pages 377–397. Springer.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18.

Hand, D. J. and Till, R. J. (2001). A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

Hastie, T. and Tibshirani, R. (1997). Classification by pairwise coupling. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Proceedings of the 11th Annual Conference on Neural Information Processing Systems (NIPS 1997, Denver, CO, USA)*. MIT Press.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2009a). A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University. Available at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

Hsu, D., Kakade, S., Langford, J., and Zhang, T. (2009b). Multi-label prediction via compressed sensing. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS 2009, Vancouver, BC, Canada)*, pages 772–780. MIT Press.

Hüllermeier, E. and Fürnkranz, J. (2004). Comparison of ranking procedures in pairwise preference learning. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2004, Perugia, Italy)*.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17):1897–1916.

Hüllermeier, E. and Vanderlooy, S. (2010). Combining predictions in pairwise classification: An optimal adaptive voting strategy and its relation to weighted voting. *Pattern Recognition*, 43(1):128–142.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A. J., editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press.

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

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

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning (ICML 1997, Nashville, TN, USA)*, pages 170–178. Morgan Kaufmann.

Kong, E. B. and Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In Armand Prieditis, S. J. R., editor, *Proceedings of the 12th International Conference on Machine Learning (ICML 1995, Tahoe City, CA, USA)*, pages 313–321. Morgan Kaufmann.

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

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

Li, Y., Zaragoza, H., Herbrich, R., Shawe-Taylor, J., and Kandola, J. S. (2002). The Perceptron Algorithm with Uneven Margins. In Sammut, C. and Hoffmann, A. G., editors, *Proceedings of the 17th International Conference on Machine Learning (ICML 2002, Sydney, Australia)*, pages 379–386. Morgan Kaufmann.

Lin, R.-S., Ross, D. A., and Yagnik, J. (2010). Spec hashing: Similarity preserving algorithm for entropy-based coding. In *The 23rd IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA*, pages 848–854. IEEE Press.

Lorena, A. C., de Carvalho, A. C. P. L. F., and Gama, J. (2008). A review on the combination of binary classifiers in multiclass problems. *Artificial Intelligence Review*, 30(1-4):19–37.

Loza Mencía, E. (2006). *Paarweises Lernen von Multilabel-Klassifikationen mit dem Perzeptron-Algorithmus*. Master's thesis, Knowledge Engineering Group, TU Darmstadt. Diplom, in german.

Loza Mencía, E. and Fürnkranz, J. (2008a). Efficient multilabel classification algorithms for large-scale problems in the legal domain. In Montemagni, S., Tiscornia, D., Francesconi, E., and Peters, W., editors, *Proceedings of the Workshop on Semantic Processing of Legal Texts (LREC 2008, Marrakech, Morocco)*, pages 23–32.

Loza Mencía, E. and Fürnkranz, J. (2008b). Efficient pairwise multilabel classification for large-scale problems in the legal domain. In Daelemans, W., Goethals, B., and Morik, K., editors, *Proceedings of the 19th European Conference on Machine Learning and Principles and Practice of Knowledge Disocvery in Databases (ECML PKDD 2008, Antwerp, Belgium), Part II*, pages 50–65. Springer.

Loza Mencía, E. and Fürnkranz, J. (2008c). Pairwise learning of multilabel classifications with perceptrons. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2008, Hong Kong, China)*, pages 2900–2907. IEEE Press.

Loza Mencía, E., Park, S.-H., and Fürnkranz, J. (2010). Efficient voting prediction for pairwise multilabel classification. *Neurocomputing*, 73(7-9):1164–1176.

MacWilliams, F. J. and Sloane, N. J. A. (1983). *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library. North Holland.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133.

Melvin, I., Ie, E., Weston, J., Noble, W. S., and Leslie, C. (2007). Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, 8:1557–1581.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Moreira, M. and Mayoraz, E. (1998). Improved pairwise coupling classification with correcting classifiers. In Nedellec, C. and Rouveirol, C., editors, *Proceedings of the 10th European Conference on Machine Learning (ECML 1998, Chemnitz, Germany)*, pages 160–171. Springer.

Murzin, A. G., Brenner, S. E., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540.

Pachet, F. and Roy, P. (2009). Improving multilabel analysis of music titles: A large-scale validation of the correction approach. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(2):335–343.

Park, S.-H. (2006). *Effiziente Klassifikation und Ranking mit paarweisen Vergleichen*. Master's thesis, Knowledge Engineering Group, TU Darmstadt. Diplom, in german.

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

Park, S.-H. and Fürnkranz, J. (2008). Multi-label classification with label constraints. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML PKDD 2008 Workshop on Preference Learning (PL-08, Antwerp, Belgium)*, pages 157–171.

Park, S.-H. and Fürnkranz, J. (2011). A note on the efficient implementation of class-based decomposition schemes for naïve bayes. *Knowledge and Information Systems*. Submitted.

Park, S.-H. and Fürnkranz, J. (2012). Efficient prediction algorithms for binary decomposition techniques. *Data Mining and Knowledge Discovery*, 24(1):40–77.

Park, S.-H., Weizsäcker, L., and Fürnkranz, J. (2010). Exploiting code redundancies in ECOC. In Pfahringer, B., Holmes, G., and Hoffmann, A., editors, *Proceedings of the 13th International Conference on Discovery Science (DS 2010, Canberra, Australia)*, pages 266–280. Springer.

Pimenta, E., Gama, J., and Carvalho, A. (2007). Pursuing the best ECOC dimension for multiclass problems. In Wilson, D. and Sutcliffe, G., editors, *Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2007, Key West, FL, USA)*, pages 622–627. AAAI Press.

Pimenta, E., Gama, J., and de Leon Ferreira de Carvalho, A. C. P. (2008). The dimension of ECOCs for multiclass classification problems. *International Journal on Artificial Intelligence Tools*, 17(3):433–447.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (1999). Large margin DAGs for multiclass classification. In Solla, S. A., Leen, T. K., and Müller, K.-R., editors, *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NIPS 1999, Denver, CO, USA)*, pages 547–553. MIT Press.

Provost, F. J. and Domingos, P. (2003). Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215.

Pujol, O., Radeva, P., and Vitrià, J. (2006). Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6):1007–1012.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359.

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

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

Shalev-Shwartz, S. and Singer, Y. (2005). A new perspective on an old perceptron algorithm. In Auer, P. and Meir, R., editors, *Proceddings of the 18th Annual Conference on Learning Theory (COLT 2005, Bertinoro, Italy)*, pages 264–278. Springer.

Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C., and Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940.

Smola, A. J., Bartlett, P. L., Schölkopf, B., and Schuurmans, D., editors (2000). *Advances in Large Margin Classifiers*. MIT Press.

Snoek, C., Worring, M., van Gemert, J., Geusebroek, J.-M., and Smeulders, A. W. M. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia. In Nahrstedt, K., Turk, M., Rui, Y., Klas, W., and Mayer-Patel, K., editors, *Proceedings of the 14th ACM International Conference on Multimedia (ACM Multimedia 2006, Santa Barbara, CA, USA*, pages 421–430. ACM.

Sulzmann, J.-N., Fürnkranz, J., and Hüllermeier, E. (2007). On pairwise naive bayes classifiers. In Kok, J. N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., and Skowron, A., editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007, Warsaw, Poland)*, pages 371–381. Springer.

Tsampouka, P. and Shawe-Taylor, J. (2007). Approximate maximum margin algorithms with rules controlled by the number of mistakes. In Ghahramani, Z., editor, *Proceedings of the 24th International Conference on Machine Learning (ICML 2007, Corvalis, OR, USA)*, pages 903–910. ACM.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of the ECML PKDD 2008 Workshop on Mining Multidimensional Data (MMD-08, Antwerp, Belgium)*, pages 30–44.

Tsoumakas, G., Loza Mencía, E., Katakis, I., Park, S.-H., and Fürnkranz, J. (2009). On the combination of two decompositive multi-label classification methods. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML PKDD 2009 Workshop on Preference Learning (PL-09, Bled, Slovenia)*, pages 114–129.

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In Kok, J. N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., and Skowron, A., editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007, Warsaw, Poland)*, pages 406–417. Springer.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley.

Wagstaff, K. and Cardie, C. (2000). Clustering with instance-level constraints. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000, Stanford, CA, USA)*, pages 1103–1110. Morgan Kaufmann.

Wagstaff, K., Cardie, C., Rogers, S., and Schrödl, S. (2001). Constrained k-means clustering with background knowledge. In Brodley, C. E. and Danyluk, A. P., editors, *Proceedings of the 18th International Conference on Machine Learning (ICML 2001, Williamstown, MA, USA)*, pages 577–584. Morgan Kaufmann.

Windeatt, T. and Ghaderi, R. (2003). Coding and decoding strategies for multi-class learning problems. *Information Fusion*, 4(1):11–21.

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3 edition.

Wong, R. (1984). A dual ascent approach for steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287.

Wu, T.-F., Lin, C.-J., and Weng, R. C. (2004). Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning (ICML 1997, Nashville, TN, USA)*, pages 412–420. Morgan Kaufmann.

Zhang, M.-L. and Zhang, K. (2010). Multi-label learning by exploiting label dependency. In Rao, B., Krishnapuram, B., Tomkins, A., and Yang, Q., editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010, Washington, DC, USA)*, pages 999–1008. ACM.

Own Publications

Journal Publications

Park, S.-H. and Fürnkranz, J. (2012). Efficient prediction algorithms for binary decomposition techniques. *Data Mining and Knowledge Discovery*, 24(1):40–77.

Loza Mencía, E., Park, S.-H., and Fürnkranz, J. (2010). Efficient voting prediction for pairwise multilabel classification. *Neurocomputing*, 73(7-9):1164–1176.

Submitted Journal Publications

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. (2011). Towards preference-based reinforcement learning. *Machine Learning*. Submitted.

Park, S.-H. and Fürnkranz, J. (2011). A note on the efficient implementation of class-based decomposition schemes for naïve bayes. *Knowledge and Information Systems*. Submitted.

Conference Publications

Cheng, W., Fürnkranz, J., Hüllermeier, E., and Park, S.-H. (2011). Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In Gunopulos, D., Hofmann, T., Malerba, D., and Vazirgiannis, M., editors, *Proceedings of the 22nd European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2011, Athens, Greece), Part I*, pages 312–327. Springer.

Park, S.-H., Weizsäcker, L., and Fürnkranz, J. (2010). Exploiting code redundancies in ECOC. In Pfahringer, B., Holmes, G., and Hoffmann, A., editors, *Proceedings of the 13th International Conference on Discovery Science (DS 2010, Canberra, Australia)*, pages 266–280. Springer.

Loza Mencía, E., Park, S.-H., and Fürnkranz, J. (2009). Efficient voting prediction for pairwise multilabel classification. In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN 2009, Bruges, Belgium)*, pages 117–122. d-side publications.

Park, S.-H. and Fürnkranz, J. (2009). Efficient decoding of ternary error-correcting output codes for multiclass classification. In Buntine, W. L., Grobelnik, M., Mladenič, D., and Shawe-Taylor, J., editors, *Proceedings of the 20th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2009, Bled, Slovenia), Part II*, pages 189–204. Springer.

Schweizer, I., Panitzek, K., Park, S.-H., and Fürnkranz, J. (2009). An exploitative Monte-Carlo poker agent. In Mertsching, B., Hund, M., and Zaheer Aziz, M., editors, *Proceedings of the 32nd Annual German Conference on Artificial Intelligence (KI 2009, Paderborn, Germany)*, pages 65–72. Springer.

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

Workshop Publications

Loza Mencía, E., Park, S.-H., and Fürnkranz, J. (2009). Efficient voting prediction for pairwise multilabel classification. In Benz, D. and Janssen, F., editors, *Proceedings of the LWA 2009: Lernen - Wissen - Adaption, Workshop Knowledge Discovery, Data Mining and Machine Learning (KDML-09, Darmstadt, Germany)*, pages 72–75. Resubmission.

Schweizer, I., Panitzek, K., Park, S.-H., and Fürnkranz, J. (2009). An exploitative Monte-Carlo poker agent. In Benz, D. and Janssen, F., editors, *Proceedings of the LWA 2009: Lernen – Wissen – Adaption, Workshop Knowledge Discovery, Data Mining and Machine Learning (KDML-09, Darmstadt, Germany)*, pages 100–104. Resubmission.

Tsoumakas, G., Loza Mencía, E., Katakis, I., Park, S.-H., and Fürnkranz, J. (2009). On the combination of two decompositive multi-label classification methods. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML PKDD 2009 Workshop on Preference Learning (PL-09, Bled, Slovenia)*, pages 114–129.

Park, S.-H. and Fürnkranz, J. (2008). Multi-label classification with label constraints. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML PKDD 2008 Workshop on Preference Learning (PL-08, Antwerp, Belgium)*, pages 157–171.

Wissenschaftlicher Werdegang[1]

06/2000	Allgemeine Hochschulreife
10/2000 – 12/2006	Informatik-Studium an der Technischen Universität Darmstadt
01/2003 – 05/2005	Werkstudent am Fraunhofer-Institut für Sichere Informationstechnologie
12/2006	Diplom in Informatik
seit 05/2007	Doktorand und Wissenschaftlicher Mitarbeiter am Fachbereich Informatik, Technische Universität Darmstadt

Erklärung[2]

Hiermit erkläre ich, dass ich die vorliegende Arbeit, mit Ausnahme der ausdrücklich genannten Hilfsmittel, selbständig verfasst habe.

1 gemäß § 20 Abs. 3 der Promotionsordnung der TU Darmstadt
2 gemäß § 9 Abs. 1 der Promotionsordnung der TU Darmstadt

A Appendix

A.1 Exploiting ECOC Redundancies: Extended LibSVM Results

This section contains additional evaluation results for Chapter 4 using LibSVM as base learner. Table A.1 shows the predictive performance of ECOC classification using various code types and parameters. Furthermore, Table A.2, Table A.3 and Table A.4 show the training times of the different approaches exploiting code redundancies (cf. Section 4.2 on page 28) using cumulative exhaustive, exhaustive and random codes for cache sizes of 25, 50, 75 and 100 %.

Table A.1: Accuracy performance of ECOC with various code types

optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
		CUMULATIVE EXHAUSTIVE CODES			
		$l = 3$			
97.24 ± 0.37	91.63 ± 0.32	88.31 ± 0.88	85.16 ± 0.12	29.09 ± 4.89	45.35 ± 2.11
		$l = 4$			
97.14 ± 0.29	91.63 ± 0.32	88.14 ± 0.88	85.16 ± 0.12	27.37 ± 4.20	45.21 ± 2.11
		EXHAUSTIVE CODES			
		$l = 3$			
97.17 ± 0.40	91.63 ± 0.32	88.61 ± 1.14	85.16 ± 0.12	31.01 ± 2.73	45.28 ± 2.04
		$l = 4$			
97.03 ± 0.32	91.39 ± 0.30	88.48 ± 0.96	85.16 ± 0.12	27.17 ± 4.09	45.08 ± 2.10
		RANDOM CODES			
		$r_{zp} = 0.4$			
96.23 ± 0.60	91.39 ± 0.30	86.97 ± 1.18	85.16 ± 0.12	34.65 ± 4.45	48.99 ± 3.76
		$r_{zp} = 0.2$			
96.21 ± 0.74	91.08 ± 0.25	87.10 ± 1.18	85.16 ± 0.12	38.69 ± 3.95	49.52 ± 4.17

Table A.2: Training time in seconds of cumulative exhaustive codes with $l = 3$ and $l = 4$. For the first block (rows 1-8) the cache size is set to 25 % of the total size. The following blocks depict the values for a cache size of $50, 75$ and $100\,\%$.

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
			$l = 3$			
M1	92.28 ± 0.36	8.73 ± 0.19	6.56 ± 0.05	3.47 ± 0.07	5.80 ± 0.02	5.43 ± 0.03
M2	80.70 ± 0.37	8.32 ± 0.37	6.00 ± 0.03	4.30 ± 0.08	4.90 ± 0.02	5.62 ± 0.02
M3	76.93 ± 0.60	6.90 ± 0.18	6.94 ± 0.05	3.13 ± 0.16	6.28 ± 0.04	5.77 ± 0.03
M4	53.37 ± 0.40	2.93 ± 0.27	4.19 ± 0.05	1.70 ± 0.25	3.51 ± 0.01	2.98 ± 0.02
			$l = 4$			
M1	833.12 ± 14.98	24.66 ± 0.43	33.98 ± 0.21	18.61 ± 0.35	47.61 ± 0.08	40.42 ± 0.09
M2	666.02 ± 1.54	21.19 ± 0.80	28.69 ± 0.14	22.94 ± 0.52	36.72 ± 0.08	41.19 ± 0.11
M3	680.75 ± 8.23	18.30 ± 0.51	36.91 ± 0.39	15.08 ± 1.71	51.61 ± 0.15	41.79 ± 0.10
M4	410.44 ± 6.08	5.32 ± 0.53	17.18 ± 0.13	8.59 ± 1.27	25.26 ± 0.06	22.01 ± 0.10
			$l = 3$			
M1	91.95 ± 0.28	8.86 ± 0.34	6.42 ± 0.04	3.43 ± 0.06	5.66 ± 0.02	5.44 ± 0.02
M2	81.83 ± 0.21	8.16 ± 0.25	6.04 ± 0.04	4.26 ± 0.08	4.91 ± 0.03	5.48 ± 0.03
M3	77.07 ± 0.98	6.96 ± 0.18	6.94 ± 0.04	2.98 ± 0.13	6.30 ± 0.02	5.87 ± 0.03
M4	53.49 ± 0.35	2.97 ± 0.34	4.20 ± 0.03	1.10 ± 0.04	3.54 ± 0.03	1.85 ± 0.02
			$l = 4$			
M1	830.01 ± 9.43	24.18 ± 0.38	32.92 ± 0.34	18.44 ± 0.35	46.35 ± 0.07	40.21 ± 0.16
M2	670.60 ± 2.83	21.77 ± 0.87	28.84 ± 0.16	22.42 ± 0.59	37.00 ± 0.09	39.60 ± 0.07
M3	674.78 ± 10.01	18.24 ± 0.42	36.82 ± 0.43	13.82 ± 1.49	51.67 ± 0.23	42.01 ± 0.08
M4	409.74 ± 4.44	5.20 ± 0.34	17.17 ± 0.13	3.65 ± 0.30	25.70 ± 0.07	10.18 ± 0.03
			$l = 3$			
M1	91.70 ± 0.20	8.82 ± 0.31	6.41 ± 0.05	3.42 ± 0.05	5.51 ± 0.03	5.35 ± 0.02
M2	79.64 ± 0.27	8.23 ± 0.34	6.00 ± 0.04	4.22 ± 0.10	4.89 ± 0.03	5.45 ± 0.03
M3	76.47 ± 0.81	6.89 ± 0.13	6.92 ± 0.04	2.97 ± 0.13	6.28 ± 0.04	5.86 ± 0.01
M4	52.94 ± 0.55	2.80 ± 0.04	4.19 ± 0.03	1.09 ± 0.03	3.52 ± 0.02	1.83 ± 0.03
			$l = 4$			
M1	823.97 ± 6.22	24.46 ± 0.58	32.88 ± 0.29	18.36 ± 0.32	45.8 ± 0.09	39.74 ± 0.10
M2	653.97 ± 2.70	21.12 ± 0.91	28.57 ± 0.12	22.18 ± 0.58	36.61 ± 0.06	38.99 ± 0.10
M3	664.03 ± 7.39	17.92 ± 0.44	36.81 ± 0.38	13.78 ± 1.48	51.55 ± 0.15	42.13 ± 0.10
M4	403.34 ± 5.69	4.99 ± 0.14	17.09 ± 0.12	3.56 ± 0.27	25.52 ± 0.05	8.84 ± 0.07
			$l = 3$			
M1	94.26 ± 0.39	8.82 ± 0.12	6.39 ± 0.04	3.41 ± 0.05	5.59 ± 0.03	5.40 ± 0.02
M2	82.03 ± 0.28	8.11 ± 0.27	5.98 ± 0.03	4.22 ± 0.09	4.90 ± 0.02	5.51 ± 0.04
M3	76.93 ± 0.90	7.05 ± 0.11	6.93 ± 0.03	3.01 ± 0.12	6.33 ± 0.04	5.94 ± 0.02
M4	54.00 ± 0.23	2.82 ± 0.19	4.20 ± 0.02	1.11 ± 0.03	3.58 ± 0.02	1.88 ± 0.02
			$l = 4$			
M1	832.16 ± 7.05	23.93 ± 0.55	32.62 ± 0.19	18.28 ± 0.29	46.04 ± 0.11	39.90 ± 0.07
M2	674.43 ± 1.59	20.66 ± 0.22	28.47 ± 0.15	22.19 ± 0.57	36.74 ± 0.11	39.07 ± 0.10
M3	675.52 ± 5.52	17.52 ± 0.08	36.84 ± 0.32	13.89 ± 1.50	51.97 ± 0.14	42.50 ± 0.07
M4	406.10 ± 5.52	4.98 ± 0.26	16.99 ± 0.13	3.69 ± 0.28	25.97 ± 0.07	9.17 ± 0.03

Table A.3: Training time in seconds of exhaustive codes with $l = 3$ and $l = 4$. For the first block (rows 1-8) the cache size is set to 25 % of the total size. The following blocks depict the values for a cache size of $50, 75$ and $100\,\%$.

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
			$l = 3$			
M1	87.42 ± 0.35	7.63 ± 0.39	6.02 ± 0.03	3.17 ± 0.05	5.51 ± 0.03	5.11 ± 0.02
M2	75.28 ± 0.29	6.76 ± 0.12	5.48 ± 0.03	3.95 ± 0.07	4.58 ± 0.03	5.28 ± 0.01
M3	75.61 ± 1.04	7.09 ± 0.27	6.91 ± 0.04	3.13 ± 0.14	6.25 ± 0.03	5.83 ± 0.05
M4	53.13 ± 0.39	2.90 ± 0.21	4.13 ± 0.03	1.71 ± 0.25	3.48 ± 0.02	3.00 ± 0.02
			$l = 4$			
M1	735.76 ± 9.63	15.31 ± 0.49	27.13 ± 0.31	15.14 ± 0.28	41.78 ± 0.09	34.99 ± 0.08
M2	570.69 ± 1.93	12.72 ± 0.45	22.76 ± 0.13	18.72 ± 0.42	31.92 ± 0.06	35.73 ± 0.06
M3	646.6 ± 11.98	16.39 ± 0.44	34.24 ± 0.36	14.69 ± 1.59	49.75 ± 0.10	41.09 ± 0.10
M4	397.79 ± 5.07	4.76 ± 0.46	15.88 ± 0.09	8.45 ± 1.17	24.55 ± 0.10	21.71 ± 0.06
			$l = 3$			
M1	87.46 ± 0.19	7.64 ± 0.41	5.90 ± 0.04	3.13 ± 0.05	5.39 ± 0.02	5.13 ± 0.02
M2	76.09 ± 0.28	6.81 ± 0.38	5.45 ± 0.03	3.87 ± 0.08	4.58 ± 0.03	5.13 ± 0.02
M3	75.05 ± 0.42	7.00 ± 0.31	6.89 ± 0.03	2.98 ± 0.12	6.29 ± 0.03	5.95 ± 0.02
M4	53.04 ± 0.08	2.82 ± 0.24	4.14 ± 0.03	1.10 ± 0.04	3.55 ± 0.02	1.90 ± 0.03
			$l = 4$			
M1	732.33 ± 6.23	15.51 ± 0.35	26.38 ± 0.22	14.87 ± 0.27	40.87 ± 0.06	34.86 ± 0.06
M2	582.49 ± 1.65	12.48 ± 0.59	22.52 ± 0.15	17.97 ± 0.48	31.92 ± 0.04	33.82 ± 0.06
M3	644.21 ± 11.06	16.24 ± 0.33	33.94 ± 0.35	13.57 ± 1.31	49.56 ± 0.11	41.76 ± 0.08
M4	394.8 ± 4.05	4.73 ± 0.42	15.70 ± 0.09	3.55 ± 0.26	24.62 ± 0.07	9.95 ± 0.05
			$l = 3$			
M1	87.47 ± 0.10	7.48 ± 0.23	5.88 ± 0.03	3.16 ± 0.05	5.25 ± 0.04	5.09 ± 0.03
M2	76.06 ± 0.19	6.95 ± 0.21	5.46 ± 0.03	3.90 ± 0.09	4.61 ± 0.01	5.15 ± 0.03
M3	76.83 ± 0.78	7.09 ± 0.19	6.88 ± 0.03	2.98 ± 0.13	6.30 ± 0.02	5.97 ± 0.02
M4	52.98 ± 0.76	2.80 ± 0.19	4.12 ± 0.04	1.12 ± 0.02	3.55 ± 0.04	1.90 ± 0.02
			$l = 4$			
M1	732.28 ± 9.55	14.99 ± 0.47	26.52 ± 0.28	15.05 ± 0.26	40.72 ± 0.11	34.64 ± 0.09
M2	581.33 ± 1.28	12.26 ± 0.46	22.56 ± 0.12	18.19 ± 0.47	32.08 ± 0.06	33.71 ± 0.09
M3	653.98 ± 8.21	15.89 ± 0.41	34.06 ± 0.30	13.6 ± 1.30	49.96 ± 0.14	42.10 ± 0.11
M4	396.06 ± 3.55	4.55 ± 0.20	15.73 ± 0.11	3.60 ± 0.23	24.74 ± 0.13	8.93 ± 0.05
			$l = 3$			
M1	86.62 ± 0.28	7.56 ± 0.32	5.90 ± 0.03	3.15 ± 0.04	5.20 ± 0.01	5.05 ± 0.03
M2	74.97 ± 0.20	6.61 ± 0.06	5.47 ± 0.03	3.89 ± 0.09	4.59 ± 0.03	5.12 ± 0.02
M3	75.69 ± 0.66	7.12 ± 0.25	7.03 ± 0.05	3.05 ± 0.13	6.39 ± 0.04	6.15 ± 0.01
M4	53.81 ± 0.39	3.01 ± 0.10	4.28 ± 0.03	1.16 ± 0.04	3.65 ± 0.02	2.08 ± 0.02
			$l = 4$			
M1	727.85 ± 9.47	15.17 ± 0.33	26.94 ± 0.22	15.10 ± 0.27	40.3 ± 0.08	34.39 ± 0.06
M2	567.74 ± 1.38	12.56 ± 0.45	22.81 ± 0.15	18.14 ± 0.51	31.92 ± 0.08	33.51 ± 0.09
M3	653.31 ± 10.18	15.93 ± 0.22	34.78 ± 0.32	13.78 ± 1.33	50.52 ± 0.21	43.47 ± 0.08
M4	404.88 ± 5.08	5.31 ± 0.10	16.68 ± 0.08	3.79 ± 0.27	25.52 ± 0.09	10.63 ± 0.05

Table A.4: Training time in seconds of random codes with $r_{zp} = 0.4$ and $r_{zp} = 0.2$. For the first block (rows 1-8) the cache size is set to 25 % of the total size. The following blocks depict the values for a cache size of $50, 75$ and $100\,\%$.

	optdigits	*page-blocks*	*segment*	*solar-flare-c*	*vowel*	*yeast*
			$r_{zp} = 0.4$			
M1	1654.0 ± 22.6	25.7 ± 1.1	156.5 ± 1.7	34.7 ± 1.5	37.5 ± 0.6	46.9 ± 1.2
M2	1424.4 ± 32.8	24.3 ± 0.5	162.9 ± 0.8	46.1 ± 1.9	39.7 ± 0.7	52.1 ± 1.3
M3	1609.2 ± 44.3	22.6 ± 0.3	190.6 ± 3.8	39.9 ± 5.4	65.8 ± 2.3	79.1 ± 2.0
M4	1378.8 ± 34.4	5.7 ± 0.3	140.6 ± 3.0	25.9 ± 3.7	57.1 ± 2.5	64.5 ± 2.4
			$r_{zp} = 0.2$			
M1	2634.6 ± 59.5	10.2 ± 0.3	123.0 ± 0.9	48.2 ± 2.0	49.6 ± 0.4	67.2 ± 1.2
M2	2281.7 ± 29.6	8.6 ± 0.5	129.7 ± 1.4	63.2 ± 3.1	53.0 ± 0.4	74.1 ± 1.3
M3	3049.0 ± 48.3	12.7 ± 0.2	157.9 ± 1.4	57.6 ± 13.3	153.0 ± 2.0	157.5 ± 2.1
M4	2594.0 ± 64.8	3.6 ± 0.2	128.5 ± 2.4	39.1 ± 9.4	144.6 ± 1.7	144.0 ± 2.2
			$r_{zp} = 0.4$			
M1	1628.3 ± 25.2	26.2 ± 0.8	156.2 ± 1.1	34.6 ± 1.4	36.8 ± 0.7	46.3 ± 1.2
M2	1337.3 ± 25.7	24.0 ± 1.0	143.3 ± 1.0	45.7 ± 1.9	37.3 ± 0.7	51.8 ± 1.3
M3	1431.6 ± 22.2	22.2 ± 0.5	156.7 ± 1.6	36.2 ± 4.6	62.7 ± 2.1	80.5 ± 1.9
M4	1174.1 ± 21.0	5.8 ± 0.2	78.3 ± 1.0	8.1 ± 0.8	45.9 ± 2.2	50.8 ± 2.4
			$r_{zp} = 0.2$			
M1	2581.3 ± 46.5	10.3 ± 0.6	120.9 ± 1.3	47.9 ± 2.0	48.6 ± 0.4	66.3 ± 1.2
M2	1832.1 ± 19.5	8.7 ± 0.3	105.1 ± 0.7	62.0 ± 3.2	48.8 ± 0.4	72.6 ± 1.3
M3	2118.8 ± 54.5	12.6 ± 0.3	125.7 ± 1.0	51.7 ± 11.3	144.5 ± 1.9	158.4 ± 2.0
M4	1663.3 ± 43.8	3.8 ± 0.3	64.8 ± 1.3	10.3 ± 1.5	126.6 ± 1.8	124.3 ± 2.0
			$r_{zp} = 0.4$			
M1	1603.4 ± 22.2	25.8 ± 0.5	153.7 ± 1.5	34.3 ± 1.5	36.0 ± 0.6	45.6 ± 1.1
M2	1317.4 ± 16.3	23.0 ± 0.4	136.9 ± 1.0	45.1 ± 1.9	35.9 ± 0.6	51.2 ± 1.3
M3	1364.6 ± 53.6	22.4 ± 0.2	148.7 ± 1.3	35.8 ± 4.5	60.9 ± 2.0	82.6 ± 2.2
M4	1162.6 ± 27.6	5.5 ± 0.3	70.3 ± 0.4	7.9 ± 0.8	42.8 ± 2.0	27.4 ± 1.2
			$r_{zp} = 0.2$			
M1	2507.0 ± 33.8	10.3 ± 0.3	119.8 ± 1.2	47.6 ± 2.0	47.6 ± 0.4	65.3 ± 1.2
M2	1826.2 ± 21.3	8.5 ± 0.6	98.7 ± 0.6	61.3 ± 3.1	44.3 ± 0.4	70.6 ± 1.2
M3	2093.7 ± 38.6	12.4 ± 0.2	116.9 ± 0.8	51.0 ± 11.0	139.9 ± 1.8	163.7 ± 2.6
M4	1632.5 ± 40.2	3.9 ± 0.1	56.8 ± 0.3	10.0 ± 1.6	118.6 ± 1.6	87.7 ± 2.2
			$r_{zp} = 0.4$			
M1	1647.5 ± 26.7	26.6 ± 0.9	159.0 ± 0.9	35.7 ± 1.5	36.9 ± 0.7	47.7 ± 1.2
M2	1339.9 ± 22.6	24.5 ± 0.6	145.6 ± 1.4	46.5 ± 2.0	36.7 ± 0.6	52.6 ± 1.3
M3	1417.0 ± 45.0	22.7 ± 0.3	154.4 ± 1.7	36.4 ± 4.5	62.4 ± 2.0	85.5 ± 2.3
M4	1117.5 ± 27.6	5.5 ± 0.3	74.0 ± 0.8	8.0 ± 0.8	43.1 ± 1.9	21.3 ± 0.3
			$r_{zp} = 0.2$			
M1	2568.4 ± 45.0	10.9 ± 0.2	124.4 ± 0.5	49.6 ± 2.0	49.1 ± 0.4	68.7 ± 1.3
M2	1836.6 ± 20.2	8.8 ± 0.5	105.3 ± 0.9	63.2 ± 3.2	44.9 ± 0.3	71.5 ± 1.1
M3	2052.3 ± 110.0	13.0 ± 0.2	121.0 ± 0.7	51.9 ± 10.9	143.6 ± 2.6	168.9 ± 2.5
M4	1600.0 ± 56.3	3.5 ± 0.1	59.6 ± 0.5	10.2 ± 1.6	119.3 ± 1.6	65.1 ± 0.9