DECOMPOSITION SCHEMES FOR MULTICLASS CLASSIFICATION AND DOMAIN ADAPTATION

Fırat İsmailoğlu

Decomposition Schemes for Multiclass Classification and Domain Adaptation

DISSERTATION

to obtain the degree of Doctor at
Maastricht University,
on the authority of the Rector Magnificus,
Prof.dr. Rianne M. Letschert,
in accordance with the decision
of the Board of Deans,
to be defended in public
on Tuesday 26 September 2017 at 16:00 hours

by

Fırat İsmailoğlu

The research described in this thesis was financially supported by The Republic of Turkey Ministry of National Education.

ISBN 000-00-0000-000-0

Contents

1 Introduction

This thesis is in the field of machine learning. It presents research on decomposition schemes for the problem of multiclass classification. In this chapter the problem is briefly sketched and class decomposition schemes are informally introduced. Four research questions related to these class decomposition schemes are formulated and emphasized. They determine the research lines explored in the subsequent chapters.

1.1 Classification

The classification problem is a central problem in machine learning (Mitchell, 1997; Smirnov, 2001). It deals with objects which may (or may not) have some property in common, for instance a similar appearance, structure, or function (Smirnov, 2001). In this way they can be grouped together, to form one or several *classes*. The objects that belong to one and the same class are called *instances* of that class. Usually, the objects are represented by a number of features, which are somehow correlated with the classes of the objects and therefore provide information on the classes.

As an example let us consider a hypothetical domain of book sales, where the objects are the books from a certain publishing house. These books can be unified to form meaningful classes in different ways, for example by year of publication, or by author, or by book type (novel, essay, short story, biography). To make this example concrete, let us restrict attention to just the following two classes: books published in 1995 and books published in 2015 (i.e., after a significant change in the publishing industry happened, due to a strong increase in online and electronic publishing). In our terminology this means that we confine ourselves to instances of books published in 1995, which form the first class, and those published in 2015 which form the second class.

To handle this domain of book sales by a computer, the instances (the books), are first represented formally by a set of *features* that are considered relevant, and which make them into elements of a space called the *feature space*. In our example, the relevant features might be the number of words in the book X_1, and the book price X_2. The classes require a formal representation as well. Usually this is done by a (output) set of class labels. In this example of book sales, we can of course work with the two labels '1995' and '2015', but when only two classes are involved one often works with the labels'$-$' and '$+$' instead.

Once we have defined the instance space and the class-label set, we can complement any available book in the instance space with a label of its class. This makes up the *training data* we have at our disposal; we will use these data to explore the relationship between the features and the class-labels and to build classifiers. In Figure 1.1, our training data is displayed to consist of 22 instances: 8 instances with class label '$+$' and 14 instances with class label '$-$'. In the figure, the instances are displayed in a two dimensional instance space defined by the features X_1 and X_2.

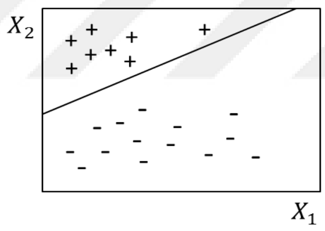


Figure 1.1: The instance space of the labeled training data, defined by the features X_1 (number of words) and X_2 (book price). The year of publication 1995 is represented by the label '$-$', the year of publication 2015 by the label '$+$'. The solid line shows a linear decision boundary for an optimal classifier.

Now, assume that we consider a book with a known number of words X_1 and a known book price X_2, and we wish to retrieve the unknown year of publication. This provides an example of the *classification problem* that in general can be stated as follows. Given a new query instance (in our case a book described by the features X_1 and X_2) and an unknown class-label (the year of publication) the problem is to estimate the true class-label (year of publication) of the instance, *given the training data*.

To solve such a classification problem, a *classifier* is derived from the training data using a *learning algorithm*. The learning algorithm searches a *hypothesis space* for the classifier that best fits the training data according to a certain *fitness criterion*. The hypothesis space is typically defined over the classifier parameters or the features of the instance space. The search strategy and the fitness criterion are specific for the algorithm. There are many choices possible; the latter explains the abundance of available learning algorithms in the literature.

Once the classifier is found, it is used for classification; i.e., to estimate the true class for any query instance. The usual assumptions are that: (1) the training instances and query instances are all generated independently from each other, and (2) the distribution that generates the data is fixed, i.e., it does not vary with the data. The first assumption allows one to ignore possible dependencies among the instances. This typically reduces the classifier complexity, as such dependencies need to be modeled by the classifier. The second assumption, on average, allows for more accurate classification: the training data and the query instances are generated by the same distribution; the classifier is based on the training data; and, thus, if an appropriate set of classifiers was searched, the classifier has a good chance of correctly classifying the query instance.

Getting back to our example for the domain of book sales, we may employ a basic classifier which belongs to a family of linear functions given by:

$$\theta_0 + \theta_1 X_1 + \theta_2 X_2,$$

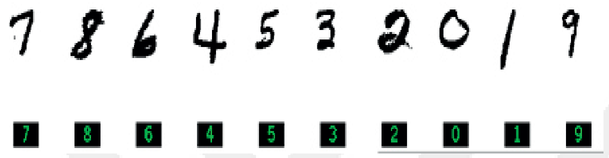
where θ_0, θ_1 and θ_2 are function parameters.

The learning algorithm then searches in the hypothesis space for the best possible parameter values θ_0, θ_1, and θ_2, taking into account the available training data and optimizing a certain fitness function. Once the optimal values are found, we receive the final classifier. Then, given a new query instance (a book) represented with feature values x_1 (number of words) and x_2 (book price), the classifier assigns the class label '$+$' (publication year 2015) if $\theta_1 x_1 + \theta_2 x_2 + \theta_0 > 0$. Otherwise, it assigns the class label '$-$' (publication year 1995). In this example, the classifier imposes a linear decision boundary, as also shown in Figure 1.1.

1.2 Binary Classification and Multiclass Classification

Depending on the number of classes involved, there are two main types of classification problems to distinguish. If there are *two classes*, the classification problem is a *binary classification problem.* If the number of classes is *larger than two*, it is a *multiclass classification problem.* An example of a binary classification problem was provided in the previous section and displayed in Figure 1.1. Other examples include: any (binary) medical test problem to diagnose the presence or the absence of a disease, any promotional mailing problem to decide whether to send an advertisement or not, any problem of quality-control with an accompanying binary test (pass/fail), and so on.

Examples of multiclass classification are at least as widespread, though, and conceptually and computationally more involved. Figure 1.2a presents ten instances of an optical character recognition problem: to assign one of the 10 classes (digits) to images of handwritten text characters. Figure 1.2b presents another multiclass classification problem: to assign a class from the set {paper, metal, plastic, glass} to waste images in order to automate the process of waste recycling. Other examples involve: breast cancer subtype classification (see Chapter 7), document categorization in text mining, speaker recognition in speech processing, and so on.



(a) Multiclass classification of handwritten digits.

(b) Multiclass classification of waste.

Figure 1.2: Multiclass classification examples.

The classifiers trained for binary classification problems are called *binary classifiers*. The classifiers trained for multiclass classification problems are called *multiclass classifiers*. Conceptually, binary classifiers are simpler than multiclass classifiers. In the worst case, a multiclass classifier for K classes may have to build $K(K-1)/2$ decision boundaries: one for each distinct pair of two classes. This explains why binary classifiers were extensively studied in the literature first. Among the most prominent binary classifiers that are worth mentioning, are linear discriminant analysis classifiers (Mitchell, 1978), the perceptron (Duda et al., 2000), logistic regression classifiers (Duda et al., 2000), version spaces (Mitchell, 1978), and support vector machines (Cortes and Vapnik, 1995).

Multiclass classifiers are usually more complex and have several different types. These types are usually divided into two groups: directly implementable multiclass classifiers and indirectly implementable multiclass classifiers (Dietterich and Bakiri, 1995). The classifiers in the first group are actually single classifiers that build all of the decision boundaries needed. Some of the classifiers in this group can be viewed as adaptations of binary classifiers such as support vector machines (Hsu and Chih-Jen, 2002) and Ripper decision rules (Cohen and Singer, 1996). Nevertheless, a vast majority of the classifiers in this group can handle multiclass classification problems without a need for an adaptation. Examples include: nearest neighbor classifiers (Duda et al., 2000), naive Bayes classifiers (Domingos and Pazzani, 1997), decision trees (Quinlan, 1986), etc.

The indirectly implementable multiclass classifiers are built using *class decomposition schemes* (Dietterich and Bakiri, 1995). Any class decomposition scheme considers a multiclass classification problem to consist of a set of binary classification problems. Then a binary classifier is built for each binary classification problem in that set (using a binary classification algorithm). The final multiclass classifier consists of several binary classifiers (in contrast to the directly implementable multiclass classifiers). When a query instance is provided, the class label for the instance is estimated according to some classification rule that combines the outputs of those binary classifiers.

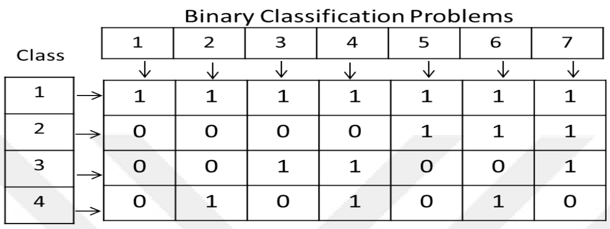
This thesis focuses on the indirectly implementable multiclass classifiers and class decomposition schemes. The next section provides a more detailed description of this topic.

1.3 Class Decomposition Schemes

As stated above, any class decomposition scheme considers a multiclass classification problem as a set of binary classification problems (Smirnov et al., 2009). These binary classification problems are represented by *binary class partitions*. If there are K classes, each such binary class partition assigns them to two disjoint families of classes: the classes in the first family (the first element of the binary partition) form a positive binary class and the classes in the second familiy (the second element of the binary partition) form a negative binary class. A class decomposition scheme is a collection of binary class partitions, such that any two classes (from the multiclass classification problem) can be uniquely determined from the scheme. I.e., there exists at least one partition (binary classification problem) for which the two classes belong to different binary classes of the partition.

Any class decomposition scheme can be represented by a *coding matrix* (Smirnov et al., 2009). The rows of the coding matrix correspond to the classes and the columns correspond to the binary class partitions. If a class is a member of a positive (negative) binary class of a binary class partition, the binary value of 1 (0) is assigned to the matrix element indicated by the class row and partition column. As an example, Figure 1.3 provides the coding matrix of an exhaustive class decomposition scheme for four-class problems (i.e., $K = 4$).

Each row in a coding matrix is considered as the *code word* of the corresponding class. Each column of a coding matrix is considered as the code word of the corresponding binary class partition. It will be clarified later that for several purposes it is important that the class code words and the partition code words have to be well separated.



Class	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	0	0	0	0	1	1	1
3	0	0	1	1	0	0	1
4	0	1	0	1	0	1	0

Binary Classification Problems

Figure 1.3: An exhaustive coding matrix for four-class problems.

To solve a multiclass classification problem using a class decomposition scheme we generally take three steps. First, we train binary classifiers, one for each binary class partition (binary classification problem). Second, we encode a query instance using these classifiers; i.e., we create an ‘instance code word’ consisting of bits (binary class labels) assigned by the binary classifiers for that instance. Third, we decode the class of the instance using the coding matrix of the class decomposition scheme used. We assign a class which code word matches best with the instance code word of the query instance.

We note that, in the third step, decoding the class of a query instance may correct a limited number of errors made by the binary classifiers. Therefore, even though some of the binary classifiers may misclassify a query instance, together they may still match the instance to its true class. This error-correcting capability increases the generalization performance of multiclass classifiers that employ a class decomposition scheme.

The classification process in step 3 of the class decomposition scheme approach (described above) builds on the assumption that the instance code words in step 2 are as true as possible (Escalera et al., 2008; Pujol et al., 2006). The correctness of instance code words depends on the accuracy of the binary classifiers trained for the binary classification problems defined by the coding matrix. Some of these binary classification problems can be relatively simple, but some may also be difficult: this depends on the application and the binary class partition. Difficult problems imply less accurate binary classifiers and thus less correct instance code words. The latter can result in instance misclassification. Thus, it is important to solve *the problem of difficult binary classification problems* in order to boost the generalization performance of multiclass classification.

There exist several approaches to this problem of difficult binary classification problems in decomposing multiclass classification problems (Hatemi, 2012; Escalera and Pujol, 2006; Marom et al., 2010; Escalera et al., 2008; Smith and Windeatt, 2010; Zor et al., 2010; Pujol et al., 2006). They are based on a search for coding matrices whose columns represent relatively simple classification problems. These approaches are divided in two groups: (1) those that adapt a given coding matrix to the data at hand (Hatemi, 2012; Escalera and Pujol, 2006; Marom et al., 2010; Escalera et al., 2008; Smith and Windeatt, 2010; Zor et al., 2010); and (2) those that learn the coding matrices from the data (Pujol et al., 2006). The reported experiments show that the approaches in both groups can be successful.

Another problem that is less obvious, is *the problem of error dependency of the binary classifiers* (Kuncheva and Whitaker, 2003a). When this problem becomes acute, the variance in the number of incorrect bits in instance code words increases. This means that error-correcting works less well and instance code words with errors are more prone to instance misclassification. We note that the problem of error dependency of the binary classifiers can occur even when the problem of difficult binary classification problems is not present.

The problem of error dependency of the binary classifiers was addressed in (Kuncheva and Whitaker, 2003a; Zhang et al., 2009; Liu et al., 2015; Furnkranz and Park, 2012). The approaches to avoid this problem can be divided into two groups. The first group includes approaches that reduce the error dependency by learning coding matrices from the data – as in the case of having difficult binary classification problems (Kuncheva and Whitaker, 2003a; Zhang et al., 2009). The second group includes approaches that reduce the error dependency when training the binary classifiers (Liu et al., 2015; Furnkranz and Park, 2012). The reported experiments show that the approaches in both groups are capable of boosting the generalization performance.

Studying the problem of difficult binary classification problems and the problem of error dependency of binary classifiers, led us to consider two other fields of machine learning as well: *reliable classification* and *domain adaptation*. In the next two sections, we briefly describe these two fields then relate them to class decomposition schemes.

1.4 Reliable Classification

Reliable classification essentially addresses classification problems where, in addition to estimating a class label for a query instance, one also aims to determine a confidence estimate for that label (Vovk et al., 2005). This is important if one needs to know to what extent one can rely on the predicted class labels, e.g., for decision making. Possible applications are usually risk-sensitive applications such as drug discovery, medical diagnosis, financial analysis (Papadopoulos et al., 2011), etc.

The framework for reliable classification that we study in this thesis in combination with class decomposition schemes is the *conformal framework* (Vovk et al., 2005). The conformal framework allows for computing the p-value for a class label assigned to a query instance as follows. First, it estimates the nonconformity ('unusualness') score for the query instance with that class label with respect to all the training instances. Then, it computes the p-value for this class-label as the proportion of the training instances of which the nonconformity scores are larger than or equal to that of the query instance. The computed p-values are then used for quantifying confidence of the classifier.

The validity of the conformal framework has been theoretically proven. It can be applied for any type of classifier (Vovk et al., 2005): nearest neighbors (Papadopoulos et al., 2011), SVM (Shi et al., 2013), neural networks (Papadopoulos, 2008), etc. In addition, the conformal framework can be used in the online mode and in the offline mode of training (Balasubramanian et al., 2014).

Despite the importance of the conformal framework for reliable (multiclass) classification, it was implemented for two types of class decomposition schemes only (Vovk et al., 2005; Shi et al., 2013). While being valid, these implementations suffer from being computationally inefficient. For this reason it is important to make class decomposition schemes computationally efficient for conformal classification, especially for practical applications.

1.5 Domain Adaptation

Domain adaptation, in the context of classification, addresses the problem of training classifiers when we are given a limited amount of labeled instances from a domain of interest (Wang and Mahadevan, 2011). If there exist similar domains that are abundant in labeled data, the key idea is to adapt these domains (i.e., their data), to derive better classifiers for the primary domain of interest. There exist several domain adaptation methods; some of them have been successfully employed in various applications, such as sentiment analysis (Blitzer et al. (2007)), computer vision ((Duan et al., 2010)), game playing ((Banerjee and Stone, 2007)), etc.

In the field of domain adaptation, the domain of interest is called the *target domain* and the provided auxiliary domains are called *source domains*. In the literature, research has mainly focused on *homogeneous domain adaptation*, where the target and source domains share the same instance space (Wu and Dietterich, 2004; Saenko et al., 2010; Hall, 2004). The observed differences between the domains arises due to the difference in the data distribution (Pan and Yang, 2010; Wang and Mahadevan, 2011), but the recorded features are the same. However, many real-world scenarios, like classifying images exploiting given tagged text, or classifying PET scans using MR images, require more advanced methods to adapt domains with totally different instance spaces. This gives rise to a relatively new research subfield, called *heterogeneous domain adaptation* which will also be of interest to us in this thesis.

The approaches to heterogeneous domain adaptation can be divided into two groups (Weiss et al., 2016). The first group consists of symmetric approaches (Wang and Mahadevan, 2011; Shi et al., 2010; Duan et al., 2012). These approaches transform the target and source domains into a common latent feature space. First, the target data and source data are projected into a common space and then the final classifiers are trained on the projected data. The second group consists of asymmetric approaches (Kulis et al., 2011;

Zhou et al., 2014). These approaches transform the source domain directly into the target domain. The classifiers are then trained on the target data and the projected source data jointly.

Despite the recent progress in heterogeneous domain adaptation, class decomposition schemes in heterogeneous domain adaptation have not yet been exploited, especially in the context of symmetric approaches. Hence several benefits that the decomposition schemes could bring when adapting the domains, such as error-correction, have not been utilized. This motivated us to use class decomposition schemes for heterogeneous domain adaptation.

1.6 Research Questions

In the previous three sections, we have discussed and argued that the further development of class decomposition schemes can follow two research lines:

(1) improving class decomposition schemes to boost the generalization performance of multiclass classifiers, and

(2) extending class decomposition schemes for new application fields such as reliable classification and heterogeneous domain adaptation.

The research questions of this thesis are formulated along these lines. The first two questions are related to the first research line and they read as follows:

(Q1) How can the problem of difficult binary classification problems be solved?

(Q2) How can the problem of error dependency of binary classifiers be solved?

As indicated previously, questions Q1 and Q2 are known in class decomposition research. The contribution of this thesis lies in improving existing approaches and studying new decomposition schemes that have a conceptually different basis.

The next two questions are related to the second research line and they read as follows:

(Q3) How to improve the computational performance of class decomposition schemes for reliable classification?

(Q4) How to apply class decomposition schemes for heterogeneous domain adaptation?

Questions Q3 and Q4 aim at bringing the class decomposition schemes beyond standard usage. This makes the thesis rather different compared to earlier work on class decomposition schemes.

1.7 Thesis Overview

The remainder of this thesis is divided into seven chapters briefly described below.

In Chapter 2 we provide a background on class decomposition schemes. We first formalize the classification problem and introduce class decomposition schemes. Then, we explain the class decomposition classification process and provide an overview of some standard class decomposition schemes.

In Chapters 3 and 4 we address the research questions related to the difficult binary classification problems and error dependency of binary classifiers. In Chapter 3 we propose *instance decomposition schemes* as an alternative to class decomposition schemes. We first define the new scheme and analyze its error-correction properties. Then, we provide experiments and conclusions. In Chapter 4, we approach to the problem of difficult binary classification problems from the perspective of weighted decoding where the binary classifiers are weighted, based on their generalization performances. In this respect we propose two novel weighted decoding algorithms: Fractional Programming Weighted decoding and Bipartite Graph Partitioning Weighted decoding. Having introduced these decoding algorithms, we then compare these algorithms with two-state-of the-art weighted decoding algorithms.

In Chapter 5 we address the research question related to improving the computational performance of class decomposition schemes for reliable classification. We propose two new implementations for conformal classification that are computationally efficient namely mean-based conformal class decomposition machines and Poisson conformal class decomposition machines. We first describe both machines, and, then, provide experiments and conclusions.

In Chapter 6 we address the research question of how to apply the class decomposition schemes for heterogeneous domain adaptation. We propose Bunching.HDA as a new symmetric heterogeneous-domain-adaptation algorithm based on the class decomposition scheme. We first describe the algorithm, and, then, provide experiments and conclusions.

In Chapter 7 we consider the problem of classifying breast cancer instances according to their immunohistochemistry subtypes. We propose to consider this problem in the context of the heterogeneous domain adaptation. We present the whole study of applying the Bunching.HDA algorithm to this problem: data derivation, data preprocessing, experiments, and conclusions.

In Chapter 8 we first summarize our answers the four research questions stated in this Chapter. Then, we complete the thesis by pointing out future research directions.

2
Background

This chapter provides background information on class decomposition schemes. Section 2.1 presents terminology used throughout this thesis and formalizes the classification problem. Section 2.2 introduces class decomposition schemes and coding matrices. The classification process based on class decomposition schemes is described in Section 2.3. Section 2.4 provides an overview of the class decomposition schemes employed in this thesis.

2.1 Classification Problem

Let X be an input space with r features $X(r)$ ($r \in \{1, \dots, R\}$) and Y be a finite output set of K class labels. We assume an unknown probability distribution p defined over $X \times Y$. Training data D is a multi-set of N labeled instances $(x_n, y_n) \in X \times Y$ generated independently from p (i.e., D is drawn iid from p). Given the training data D, the classification problem is to provide a good estimate $\hat{y} \in Y$ of the true class of a query instance $x_q \in X$ according to the probability distribution p.

To solve a classification problem we train a *classifier* on the data available using a learning algorithm. A classifier predicts a class for any query instance $x_q \in X$. We note that there exist two types of classifier: *discrete* and *scoring*. A discrete classifier is a function $f : X \rightarrow Y$. It outputs the estimated class label of the query instance. A scoring classifier is a function $f : X \rightarrow [0, +\infty)^K$. It assigns a positive score $s \in [0, \infty)$ for each label in the class label set for the query instance. The class label with the highest score is usually picked as the final class prediction.

Depending on the size K of the class label set Y we consider two types of the classification problems. If K is equal to 2, the classification problem is a *binary classification problem* (*BCP*). If K is greater than 2, the classification problem is a *multiclass classification problem* (*MCP*). The classifiers trained for binary classification problems are called *binary classifiers*. The classifiers trained for multiclass classification problems are called *multiclass classifiers*.

To make a compatible notation for binary discrete classifiers and binary scoring classifiers throughout this thesis we assume that any binary scoring classifier f outputs for any query instance x_q only one score $f(x_q)$ in the range of $[0, 1]$. This score is the normalized score for the positive class which implies that the normalized score for the negative class is $1 - f(x_q)$.

2.2 Class Decomposition Schemes and Coding Matrices

Consider a multiclass classification problem *MCP* defined over a class label set Y of size $K > 2$. To decompose *MCP* into B binary classification problems *BCP*$_b$ ($b \in \{1, \ldots, B\}$) we need to define these problems so that the binary class labels allow us to code any class $y \in Y$ (Smirnov et al., 2009). The class label set Y_b of any binary classification problem *BCP*$_b$ can be represented by a binary class partition P_b over the class label set Y.

Definition 1. (Binary Class Partition) *Given a class label set Y and an integer $b \in \{1, \ldots, B\}$, the set $P_b(Y)$ is a binary class partition of Y if and only if $P_b(Y)$ consists of two non-empty sets Y_b^- and Y_b^+ such that $Y_b^- \cap Y_b^+ = \emptyset$ and $Y_b^- \cup Y_b^+ = Y$.*

Since the sets Y_b^- and Y_b^+ are non-empty, they have their own meaning derived from the meaning of the classes they combine. Thus, without loss of generality we consider the set Y_b^- as a *negative super class* and the set Y_b^+ as a *positive super class* for the binary classification problem *BCP*$_b$.

Notation 1. (Super Class Labels) *The negative super class Y_b^- is denoted by the label 0. The positive super class Y_b^+ is denoted by the label 1. The class label set of a binary classification problem BCP$_b$ is denoted by Y_b.*

Using Definition 1 we introduce the notion of a *class decomposition scheme* (Smirnov et al., 2009).

Definition 2. (Class Decomposition Schemes) *Given a multiclass classification problem MCP and a positive integer $B > 0$, a* class decomposition scheme *of MCP is a set $SP(Y)$ of B binary class partitions $P_b(Y)$ with $b \in \{1, ..., B\}$ such that for any two classes $y_1, y_2 \in Y$ there exists a binary class partition $P_b(Y) \in SP(Y)$ which super classes Y_b^- and Y_b^+ separate y_1 and y_2; i.e. $\neg(y_1, y_2 \in Y_b^-) \wedge \neg(y_1, y_2 \in Y_b^+)$.*

According to Definition 2 for any class $y \in Y$ the set of super classes Y_b^- and Y_b^+ that include y over all $b \in \{1, ..., B\}$ is unique for that class. When this property holds, a set of binary class partitions is a class decomposition scheme.

Any class decomposition scheme $SP(Y)$ is typically represented by a *coding matrix* M (Dietterich and Bakiri, 1995). The matrix M is defined as a binary matrix of form $\{0, 1\}^{K \times B}$ whose entries $M(y, b)$ are defined according to the following rule:

$$M(y,b) = \begin{cases} 0, & \text{if class } y \in Y \text{ belongs to } Y_b^- \text{ of } P_b(Y); \\ 1, & \text{if class } y \in Y \text{ belongs to } Y_b^+ \text{ of } P_b(Y). \end{cases}$$

Any class label $y \in Y$ is represented by exactly one row $M(y, \cdot)$ of M. We call this row *code word for the class* y. Any binary class partition $P_b(Y)$ is represented by exactly one column $M(\cdot, b)$ of M. We call this column *code word for the binary class partition* $P_b(Y)$.

Class code words $M(y, \cdot)$ can be viewed as images of the classes $y \in Y$ in a *code space*. The code space plays an essential role in multi class classification using class decomposition schemes. It is formally defined below.

Definition 3. (Code Space) *Given a class decomposition scheme $SP(Y)$ with B partitions $P_b(Y)$, the code space $\mathcal{C}$ is equal to* $[0, 1]^B$.

Since any coding matrix M represents a class decomposition scheme, it has to satisfy four properties. Below we introduce these decomposition-induced properties and show how they can be logically derived using Definition 2.

1. *Row Uniqueness*. This property states that the rows $M(y, \cdot)$ of the matrix M have to be unique. Consider an arbitrary class $y_1 \in Y$. According to Definition 2 for any other class $y_2 \in Y$ there exists a binary class partition $P_b(Y) \in SP(Y)$ s.t. $\neg(y_1, y_2 \in Y_b^-) \wedge \neg(y_1, y_2 \in Y_b^+)$. This implies that for these two classes the associated rows $M(y_1, \cdot)$ and $M(y_2, \cdot)$ are different at least at position b. Thus, the row $M(y_1, \cdot)$ of class y_1 is unique, and, since y_1 is arbitrarily chosen, the property holds for any class $y \in Y$.

2. *Column Uniqueness*. This property states that the columns $M(\cdot, b)$ of the matrix M have to be unique. It follows directly from Definition 2. According to the definition the class decomposition scheme $SP(Y)$ is a set which implies that binary class partitions $P_b(Y) \in SP(Y)$ are different. Thus, the columns $M(\cdot, b)$ of M that correspond to different $P_b(Y)$ are different as well.

3. *Column Heterogeneity*. This property states that any column has to have at least one 1 and least one 0. It can be proven as follows. According to Definition 2 any column $M(\cdot, b)$ represents a class partition $P_b(Y)$. According to Definition 1 $P_b(Y)$ consists of non-empty sets Y_b^- and Y_b^+. Thus, $M(\cdot, b)$ has to have at least one 1 and least one 0.

4. *Column Nonequivalency*. This property states that any two columns of M cannot complement each other. It can be proven by contradiction. Assume that the columns indexed by $b_1, b_2 \in \{1, ..., B\}$ are complementary. This implies that $\forall y \in Y$, $M(y, b_1) + M(y, b_2) = 1$. Thus,

$$\begin{aligned} Y_{b_1}^{+} &= \{y \in Y \mid M(y, b_1) = 1\} \\ &= \{y \in Y \mid 1 - M(y, b_2) = 1\} \\ &= \{y \in Y \mid M(y, b_2) = 0\} \\ &= Y_{b_2}^{-}. \end{aligned}$$

Analogously, we can prove that $Y_{b_1}^{-}$ equals $Y_{b_2}^{+}$. Thus, $P_{b_1}(Y) = \{Y_{b_1}^{+}, Y_{b_1}^{-}\} = \{Y_{b_2}^{-}, Y_{b_2}^{+}\} = P_{b_2}(Y)$. The latter contradicts with Definition 2, since the class decomposition scheme $SP(Y)$ is set and it cannot contain two identical elements, i.e. binary class partitions.

In the rest of the thesis class decomposition schemes are presented mainly with coding matrices. So, we use the notions of class decomposition schemes and code matrices interchangeably.

2.3 Solving Multiclass Classification Problems using Class Decomposition Schemes

To solve a multiclass classification problem using a class decomposition scheme we follow three steps. First, we train a set of binary classifiers that correspond to that scheme. Then, we encode a query instance using those classifiers. Finally, we decode the class of the instance using the coding matrix of the class decomposition scheme used; i.e., we receive the final class estimate. This section describes formally all the three steps irrespective the types of the class decomposition schemes.

2.3.1 Training Binary Classifiers

To train binary classifiers we need to define B binary classification problems $\textit{BCP}_b$ according to the class decomposition scheme $SP(Y)$ provided. For any $b \in \{1, ..., B\}$ the binary classification problem $\textit{BCP}_b$ is defined in the input space X and the class label set Y_b. The binary training data set D_b is constructed from the training data set D using the coding matrix M of $SP(Y)$: any instance $(x, y) \in D$ is transformed to a new instance $(x, M(y, b)) \in D_b$. We note that D_b is drawn iid from $X \times Y_b$, since Y_b^{-} and $Y_b^{+} \in SP_b(Y)$ are super classes that combine the classes of Y and the data D is drawn iid from $X \times Y$.

Once the binary classification problems have been defined, we train binary classifiers f_b ($b \in \{1..B\}$) by means of a given classification algorithm . These classifiers form an ensemble classifier $f : X \to Y$ equal to $\{f_b\}_{b=1}^{B}$, given a classification rule for the coding matrix M (see class decoding function in Subsection 2.3.3).

2.3.2 Encoding

Once a query instance x_q is provided, it is encoded in the code space $\mathcal{C}$. For that purpose, the binary classifiers f_b output class estimations $f_b(x_q)$ and then the vector of these estimations forms an image of x_q in the code space $\mathcal{C}$. Formally, the encoding step is given with a function defined below.

Definition 4. (Encoding Function) *Encoding function f is a function from X to $\mathcal{C}$ that maps an instance $x \in X$ to:*

$$(f_1(x), ..., f_B(x)).$$

Given a query instance x_q, the code $f(x_q)$ is called the code word of that instance. We note that instance code words and class code words live in the same code space $\mathcal{C}$. Based on their match the final class estimation is determined (via decoding explained the next Subsection).

2.3.3 Decoding

Once the code word $f(x_q)$ of the query instance x_q is formed, it is decoded to the class $y \in Y$ which associated class code word $M(y, \cdot)$ is *nearest* to $f(x_q)$ according to some distance measure *dist* defined on $\mathcal{C}$. This class becomes the class estimation of the instance x_q. Formally, decoding is achieved using a function defined below.

Definition 5. (Decoding Function) *Decoding function g is a function from $\mathcal{C}$ to 2^Y that maps the code word $f(x)$ of an instance $x \in X$ to:*

$$\operatorname*{argmin}_{y \in Y} \mathit{dist}\left(f(x), M\left(y, \cdot\right)\right),$$

where dist is a distance measure on $\mathcal{C}$.

Given a query instance x_q, the decoding function g outputs a subset of classes $y \in Y$ whose code words are nearest to the instance code word $f(x_q)$. The final class estimate for x_q is chosen randomly from the classes in $g(f(x_q))$.

Below Figure 2.1 illustrates the coding and decoding phases guided by the coding matrix M.

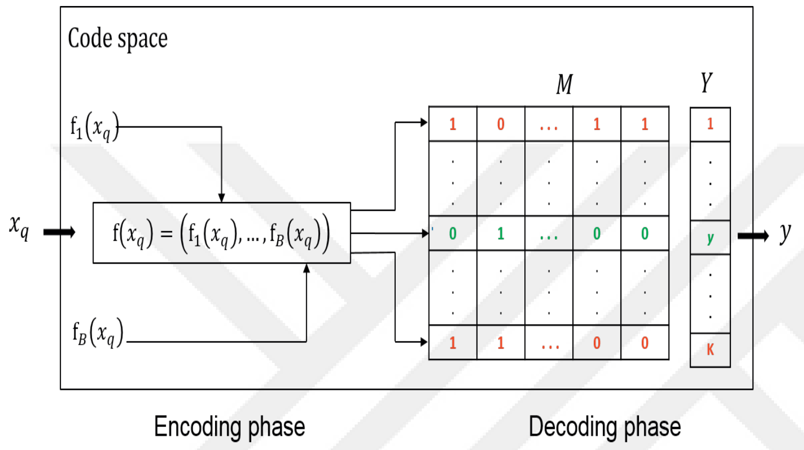


Figure 2.1: An illustration of the coding and decoding phases. An query instance x_q from the input space X is first encoded into the code word $f(x_q)$ equal to $(f_1(x_q), \ldots, f_B(x_q))$. Then, $f(x_q)$ is compared to each class code word of the coding matrix M. Finally, the class of x_q is decoded equal to the class with code word nearest to $f(x_q)$.

The classification process through decoding assumes that the instance code words are as correct as possible. The correctness of instance code words depends directly on the accuracy of the binary classifiers. The latter depends on the binary classification problems defined by the coding matrix M. Some of these problems can be relatively simple and some of these problems can be relatively difficult (see Section 3.1 for a detailed explanation). The difficult problems imply less precise binary classifiers and thus less correct instance code words. The latter can result in misclassification. Thus, it is important to handle the problem of difficult binary classification problems in order to boost the generalization performance of multi class classification.

Another problem to solve that is not so obvious is the problem of error dependency of the binary classifiers. When this problem becomes acute, the number of incorrect bit estimations in instance code words increases. This means that the instance code words become less correct which can result in misclassification. We note that the problem of error dependency of the binary classifiers can occur when the problem of difficult binary classification problems is not well present as soon as we have good binary classifiers that have a tendency to err simultaneously.

2.3.4 Separation Properties of Coding Matrices

Separation properties determine when a coding matrix results in an ensemble of binary classifiers that potentially has a good generalization performance. There exist two separation properties: the row-separation property and the column-separation property (Dietterich and Bakiri, 1995). Below we explain these properties in detail.

- *Row separation.* This property states that for any class $y_1 \in Y$ the code word $M(y_1, \cdot)$ in a coding matrix M has to be well-separated (distant) from the code word $M(y_2, \cdot)$ of any other class $y_2 \in Y \setminus \{y_1\}$. Since the code words $M(y_1, \cdot)$ and $M(y_2, \cdot)$ are the images of the classes y_1 and $y_2 \in Y$ in the code space $\mathcal{C}$, the row-separation property indicates how far the images of the classes in Y are in $\mathcal{C}$. Given a query instance x_q with true class y_q, a large row separation allows increasing the number of binary classifiers f_b with loss $|f_b(x_q) - M(y_q, b)|$ greater than 0.5 so that the class of the instance is correctly decoded. This means that the row-separation property counteracts the difficult binary classification problems and error dependency of the binary classifiers, and, thus, it plays an important role for the generalization performance of the ensembles based on class decomposition schemes.

  The row-separation capacity of a coding matrix is usually quantified using *the minimum Hamming distance R_m* between any pair of matrix's rows. We note that the Hamming distance is employed due to the binary nature of the coding matrix. When the binary classifiers are discrete, the minimum Hamming distance R_m of a coding matrix is used to compute the *correction number CN*. CN equals $\lfloor \frac{R_m - 1}{2} \rfloor$ and it represents the number of misclassifying binary classifiers which still results in correct classification. Since the row separation is related the problem of difficult binary classification problems and the problem of error dependency of the binary classifiers, the minimum Hamming distance R_m and the correction number CN are used to quantify the ability of the coding matrix (class decomposition scheme) to solve these problems.

- *Column separation.* This property states that for any $b_1 \in \{1, \ldots, B\}$ the partition code word $M(\cdot, b_1)$ in a coding matrix M has to be well-separated (distant) from the partition code word $M(\cdot, b_2)$ and its complement for any other $b_2 \in \{1, \ldots, B\} \setminus \{b_1\}$. In this way, binary classification problems *BCP*$_b$ become more different in terms of their super classes which in turn makes the binary classifiers f_b more diverse. The diversity of the binary classifiers f_b makes them more error-independent (Dietterich and Bakiri, 1995); i.e., it is less often that the classifiers commit the errors simultaneously. This can significantly decreases the distance to the true class during query-instance decoding which boosts the generalization performance of the ensembles based on class decomposition schemes.

  The column separation property of a coding matrix is usually quantified using the minimum Hamming distance C_m between any pair of its columns. The higher the distance C_m is, the more diversity and less error dependency we observe for the binary classifiers f_b. Thus, the minimum Hamming distance C_m can be used as a second measure to quantify the ability of the coding matrix to solve the problem of error dependency of the binary classifiers.

2.4 Standard Class Decomposition Schemes

There exist several standard class decomposition schemes. Of these, this section considers class decomposition schemes based on one-vs-all matrices (Rifkin and Klautau, 2004) and error-correcting output codes (Dietterich and Bakiri, 1995; Smirnov et al., 2009) which are used in the remainder of this thesis.

2.4.1 One-vs-All Class Decomposition Scheme

A one-vs-all class decomposition scheme is a scheme with a coding matrix M of type $\{0,1\}^{K\times K}$ such that for any class $y \in Y$ there exists a column with index $b \in \{1,...,K\}$ for which the condition $M(y,b)$ holds only for class y. Formally,

$$(\forall y \in Y)(\exists b \in \{1,...,K\})(M(y,b) = 1 \wedge (\forall y' \in Y \setminus \{y\})M(y',b) = 0).$$

Figure 2.2 provides an example of the one-vs-all coding matrix M for four classes. The matrix is a diagonal matrix. Hence, four binary classification problems are generated such that in each problem we discriminate the instances of one class against the instances of the other three classes. Following the example Figure 2.3 shows a decision boundary induced by a classifier trained for one of the four binary classification problems.

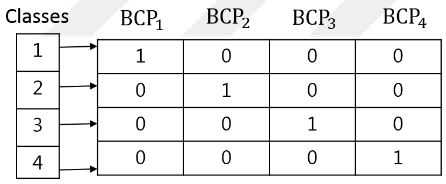
Decoding the class of a query instance is performed according to the decoding function provided in Definition 5. The distance function for one-vs-all is defined below.

Definition 6. (One-vs-all Distance Function) *One-vs-all distance function $dist_{OA}$ is a function from $\mathcal{C} \times \mathcal{C}$ to $[0,1]$ that maps the code word $f(x)$ for an instance $x \in X$ and the code word $M(y,\cdot)$ of a class $y \in Y$ to:*

$$1 - f(x) \cdot M(y,\cdot).$$

Given a query instance x_q, the value of $f(x_q) \cdot M(y,\cdot)$ is equal to the value of $f_b(x_q)$; i.e. the outcome of the binary classifier f_b for class $y \in Y$ with index $b \in \{1,...,K\}$ in the coding matrix M. It can be viewed as a similarity between the instance code word $f(x_q)$ and class code word $M(y,\cdot)$. This means that the value of $1 - f(x_q)$ is a distance between these words. Therefore, according to Definition 5 the classification (decoding) rule of one-vs-all class decomposition schemes is to assign that class $y \in Y$ to a query instance $x_q \in X$ that minimizes the distance $1 - f_b(x_q)$.

The one-vs-all coding matrices grow linearly with the number of classes. Since the number B of binary classifiers equals the number K of classes, they result in computationally efficient ensembles. However, the separation properties, the row-separation and column-separation properties, are poorly represented. The minimal row Hamming distance R_m equals 2 and, thus, the correction number CN equals 0. The minimal column Hamming distance C_m equals 2. Therefore, the one-vs-all class decomposition schemes are often outperformed by other types of class decomposition schemes. However,, it was shown in (Rifkin and Klautau, 2004)) that if the binary classifiers are scoring classifiers and they are properly tuned, the one-vs-all ensembles have a good generalization performance.



Classes	BCP_1	BCP_2	BCP_3	BCP_4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Figure 2.2: One-vs-all coding matrix for 4 classes.

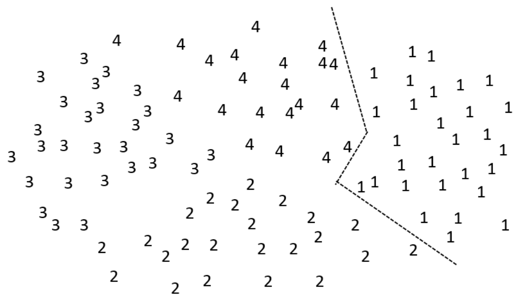


Figure 2.3: A decision boundary of a classifier trained for one of the binary classification problems defined by the four-class one-vs-all coding matrix.

2.4.2 Class Decomposition Schemes of Error-Correcting Output Codes

The class decomposition schemes of error-correcting output codes (ECOC) employ redundancy in the class code words to boost the generalization performance of the final ensembles (Dietterich and Bakiri, 1995). In the present subsection, first we consider three basic types of ECOC schemes and then provide the decoding procedures applied for discrete and scoring binary classifiers.

Basic Types of ECOC Schemes

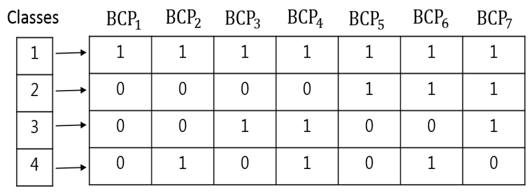
The ECOC class decomposition schemes differ in coding matrices they employ. We ahall consider two coding matrices in this type:exhaustive error-correcting output codes (eECOC) (Dietterich and Bakiri, 1995) and minimal error-correcting output codes (mECOC) (Smirnov et al., 2009). Then, we consider decomposition schemes of random error-correcting output codes (rECOC) as a intermediate type between these two schemes (Dietterich and Bakiri, 1995).

A. **Class Decomposition Schemes of Exhaustive Error-Correcting Output Codes**

A class decomposition scheme of exhaustive error-correcting output codes (eECOC) is a scheme with a coding matrix M of type $\{0,1\}^{K\times(2^{K-1}-1)}$ in which for any proper and nonempty subset $Y' \subset Y$ there exists a column with index $b \in \{1, ..., 2^{K-1}-1\}$ such that for any class $y' \in Y'$ and any other class $y'' \in Y \backslash Y'$ the condition $M(y', b)$ differs from the condition $M(y'', b)$. Formally,

$$(\forall Y' \in 2^{Y}\backslash\{\emptyset, Y\})(\exists b \in \{1, ..., B\})(\forall y \in Y')(\forall y'' \in Y\backslash Y')M(y', b) \neq M(y'', b).$$

Figure 2.4 provides an example of the eECOC coding matrix M for four classes. Seven binary classification problems are generated so that in each problem we discriminate a class subset Y' against the complement $Y \setminus Y'$. Following the example Figure 2.5 shows a decision boundary induced by a classifier trained for the fifth binary classification problem, i.e. *BCP*$_5$.



Classes	BCP_1	BCP_2	BCP_3	BCP_4	BCP_5	BCP_6	BCP_7
1	1	1	1	1	1	1	1
2	0	0	0	0	1	1	1
3	0	0	1	1	0	0	1
4	0	1	0	1	0	1	0

Figure 2.4: eECOC coding matrix for 4 classes.

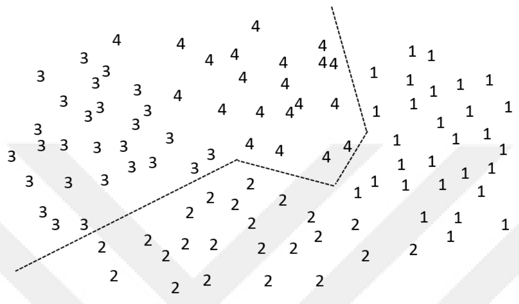


Figure 2.5: A decision boundary of a classifier trained for the fifth binary classification problem from the eECOC coding matrix from Figure 2.4.

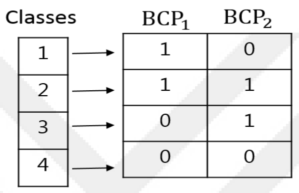
The number of columns in the eECOC coding matrices equals $2^{K-1} - 1$; i.e., it is exponential in the number of classes. This implies that the number of binary classification problems and number binary classifiers are exponential in K. As a result the ensembles based on eECOC coding matrices can be computationally inefficient. Because of that (Dietterich and Bakiri, 1995) advised to use the eECOC class decomposition schemes for no more than seven classes.

The separation properties of the eECOC coding matrices, the row-separation and column-separation properties, are very different compared to one-vs-all (Dietterich and Bakiri, 1995). The row minimum Hamming distance R_m is equal to 2^{K-2} and the correction number CN is equal to $\lfloor \frac{2^{K-2}-1}{2} \rfloor$. Thus, the class code words are quite distant in the code space $\mathcal{C}$ and the row-separation property is well represented. However, the column minimum Hamming distance C_m is equal to 1. Thus, the column-separation property is poorly represented which can make some of the binary classifiers more error dependent (Dietterich and Bakiri, 1995). Nevertheless, the ensembles based on eECOC coding matrices exhibit a good generalization performance, since the error dependence can be suppressed by the large amount of binary classifiers, especially for large K.

B. Class Decomposition Schemes of Minimal Error-Correcting Output Codes

A class decomposition scheme of minimal error-correcting output codes (mECOC) is a scheme with a coding matrix M of type $\{0, 1\}^{K \times \lceil \log_2(K) \rceil}$ (Smirnov et al., 2009). It is minimal in the sense that there is no matrix with smaller number of columns that represents a class decomposition scheme.

Figure 2.6 provides an example of the mECOC coding matrix M for four classes. Two binary classification problems are generated so that in each problem we discriminate a class subset Y' of size 2 against the complement $Y \setminus Y'$ of size 2 as well. Following the example, Figure 2.7 shows a decision boundary induced by a classifier trained for one of the two binary classification problems.



Classes		BCP_1	BCP_2
1	$\longrightarrow$	1	0
2	$\longrightarrow$	1	1
3	$\longrightarrow$	0	1
4	$\longrightarrow$	0	0

Figure 2.6: mECOC coding matrix for 4 classes.

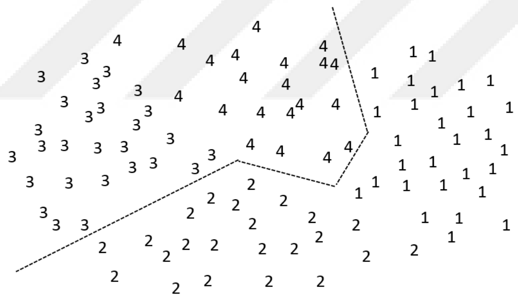


Figure 2.7: A decision boundary of a classifier trained for the second binary classification problem from the mECOC coding matrix from Figure 2.6.

The number of columns in the mECOC coding matrices equals $\lceil \log_2(K) \rceil$; i.e., it grows logarithmically with the number of classes. This implies that the number of binary classification problems and number binary classifiers are logarithmic in K. As a result, the ensembles based on mECOC coding matrices are the most computationally efficient and, thus, they can be used for very large number of classes.

The separation properties of the mECOC coding matrices, the row-separation and column-separation properties, are very poor. The row minimum Hamming distance R_m is equal to 1 and the correction number CN is equal to 0. This means that the class code words can be very close to one another in the code space $\mathcal{C}$ and, thus, the row-separation property is poorly represented. However, the column minimum Hamming distance C_m is equal to $\frac{K}{2}$. Thus, the row-separation property is well represented which helps the binary classifiers be more error independent. Nevertheless, it is often not enough for the ensembles based on mECOC coding matrices to exhibit the same generalization performance as the eECOC ensembles.

C. **Class Decomposition Schemes of Random Error-Correcting Output Codes**

A class decomposition scheme of random error-correcting output codes (rECOC) is a scheme with a matrix M of type $\{0,1\}^{K \times B}$ whose columns are randomly chosen from those of the corresponding eECOC matrix (Dietterich and Bakiri, 1995). The number B is in the range of $[\lceil \log_2(K) \rceil, 2^{K-1} - 1]$ and can be preset in advance. Since it is equal to the number of the binary classifiers, it is used to control the computational complexity of the rECOC ensembles.

The separation properties of the eECOC coding matrices, the row-separation and column-separation properties, depends on the randomness process: they can be well or poorly represented. In this context, we note that the row-separation property improves when the number B increases. However, the column-separation property does not necessarily improve when the number B decreases.

2.4.3 Decoding

Decoding the class of a query instance is performed according to the distance function in decoding function (Definition 5). The decoding function differs in the type of binary classifiers (i.e. discrete, scoring) employed. Below we elaborate on the decoding phase for discrete and scoring binary classifiers separately.

Hamming Decoding

When the binary classifiers are discrete, the obvious choice is to employ the Hamming distance over the code space $\mathcal{C}$. Below we define the Hamming distance function to find distances between instance and class code words.

Definition 7. (Hamming Distance Function) *Hamming distance function $dist_H$ is a function from $\mathcal{C} \times \mathcal{C}$ to $\{0, ..., B\}$ that maps the instance code word $f(x)$ for an instance $x \in X$ and the class code word $M(y, \cdot)$ for a class $y \in Y$ to:*

$$\sum_{b=1}^{B} [\![f_b\left(x\right) \neq M\left(y, b\right)]\!],$$

where $[\![\ ]\!]$ is an indicator function.

According to Definitions 5 and 7 the Hamming class decoding rule is as follows: a query instance $x_q \in X$ is assigned to class $y \in Y$ whose code word $M(y, \cdot)$ is nearest to the code word $f(x_q)$ in terms of the Hamming distance. The nice property of the class decoding rule lays in its error-correcting capabilities, meaning that even if the number of binary classifiers that misclassify the query instance x_q exceeds CN ($CN = \lfloor \frac{R_m - 1}{2} \rfloor$), the final class estimate for that instance can be still correct. This property is due to the fact that if we have CN mistakes in the code word $f(x_q)$, $f(x_q)$ will be still closer to the code word $M(y,\cdot)$ of the true class y. This is illustrated in Figure 2.8.

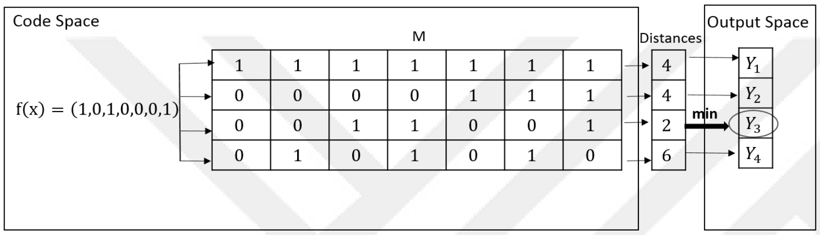


Figure 2.8: Error correction: the code word of a query instance with true class y_3 is nearest to its true class although the first and the third binary classifiers have misclassified that instance.

Loss-Based Decoding

When the binary classifiers are scoring, the Hamming distance is not applicable. In this case, loss-based distances over the code space $\mathcal{C}$ are employed (Escalera et al., 2008). Below we define the loss-based distance function to find distances between instance code words and class code words. i

Definition 8. (Loss-based Distance Function) *Loss-based distance function $dist_L$ is a function from $\mathcal{C} \times \mathcal{C}$ to $\{0, ..., B\}$ that maps the instance code word $f(x_q)$ for an instance $x \in X$ and the class code word $M(y,\cdot)$ for a class $y \in Y$ to:*

$$\sum_{b=1}^{B} L\left(f_b\left(x_q\right), M\left(y, b\right)\right)$$

where L is a loss function.

According to Definitions 5 and 8 the loss-based class decoding rule is as follows: a query instance $x_q \in X$ is assigned to class $y \in Y$ whose code word $M(y,\cdot)$ is nearest in terms of the loss-based distance to the code word $f(x_q)$. If the class y is indeed the true class of x_q, then the scores of all the binary classifiers f_b for x_q are being corrected. This explains why the loss-based class decoding rule often outperforms the Hamming class decoding rule.

There exist several options for the loss function (Flach, 2012). In this thesis we employ the square-loss function. It is defined as a function L from $[0,1] \times \{0,1\}$ to $\mathbb{R}^+$ that maps the code word $f(x)$ for an instance $x \in X$ and a code word $M(y,\cdot)$ for a class $y \in Y$ to $\sum_{b=1}^{B}(f_b(x) - M(y,b))^2$.

3 Instance-based Decompositions

This chapter is based on the following publication:
Ismailoglu, F., Smirnov, E., Nikolaev, N., and Peeters, R. (2015). Instance-Based Decompositions of Error Correcting Output Codes. In 12th Int.Workshop on Multiple Classifier Systems (MCS), LNCS, vol. 9132, pages 51–63. Springer International Publishing.

This chapter addresses the research questions related to the difficult binary classification problems and error dependency of binary classifiers. It proposes *instance* decomposition schemes as an alternative to class decomposition schemes. An instance decomposition scheme consists of several instance partitions. Any instance has a code word that indicates the instance set including that instance for each partition. Instance classification assumes two instance decomposition schemes: encoding and decoding. The encoding scheme is used to train binary classifiers and to encode any query instance. The decoding scheme is used to decode the instance class. This is achieved by finding the class of an instance with code word nearest to the code word of the query instance in the coding matrix of the decoding scheme.

Instance decomposition schemes are thoroughly analyzed in this Chapter. It is shown that they do have a mechanism to compensate the errors caused by the difficult binary classification problems and error dependency of binary classifiers. This is demonstrated in the experiments in which instance-based decomposition schemes outperform ECOC-based class decomposition schemes.

The chapter consists of 5 sections. Section 3.1 considers the problem of difficult binary classification problems and the problem of error dependency of binary classifiers together with related work. Section 3.2 introduces instance decomposition steps: definition, encoding and decoding steps, and initialization. Section 3.3 analyzes error correction property of instance decomposition schemes. The experiments are provided in Section 3.4. Section 3.5 concludes the chapter.

3.1 Problems and Related Work

In the previous two chapters we showed that when decomposing multiclass classification problems into a series of binary classification problems, one may encounter two serious problems: the problem of difficult binary classification problems and the problem of error dependency of the binary classifiers. In this section we consider these two problems and provide the related work.

The problem of difficult binary classification problems means that these problems are difficult for the binary classifiers employed. For example assume that the feature space of a given four class classification problem is similar to the one shown in Figure 3.1. Intuitively, the binary classification problem P1 that corresponds to super classes $\{+\}$ and $\{*, -, @\}$ is more difficult than the binary classification problem P2 that corresponds to super classes $\{*\}$ and $\{+, -, @\}$. This means that a binary classifier for problem P1 will err more often than a binary classifier for problem P2. Thus, more difficult classification problems imply more errors. When the total number of errors per a query instance exceeds the correction number *CN* (see the discussion on *CN* in Subsection 2.3.4), the instance may be misclassified. Thus, the difficult binary classification problems can reduce the error correcting capability of class decomposition schemes.

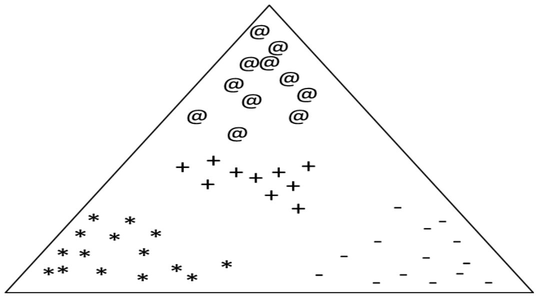


Figure 3.1: The feature space of a given four class problem.

There is no consensus how to measure the difficulty of (binary) classification problems. Ho and Basu (2000) considered several measures. Among them the most relevant for our research are measures for class separability: linear separability and mixture identifiability. A measure for linear separability indicates to what extent the instances of different classes are linearly separable. If the different class instances are found to be linearly separable, the classification problem can be solved by a linear classifier or a non-linear classifier, and, thus, it is considered as relatively simple. Applied to the example in Figure 3.1 a good linear-separability measure will indicate that the binary classification problem P1 with super classes $\{+\}$ and $\{*, -, @\}$ is less linearly separable than the binary classification problem P2 with super classes $\{*\}$ and $\{+, -, @\}$. Thus, the binary classification problem P2 can be considered as simpler than the binary classification problem P1.

A measure for mixture identifiability indicates whether the instances of different classes come from the same distribution. If for a classification problem the different class instances are found to come from different distributions, the problem is easier to solve, and, thus, it is considered as relatively simple. Applied to the example in Figure 3.1 a good measure for mixture identifiability will find that the instances of the super classes $\{+\}$ and $\{*, -, @\}$ are more probable to come from the same distribution than the instances of the super classes $\{*\}$ and $\{+, -, @\}$. Thus, the binary classification problem P2 with super classes $\{*\}$ and $\{+, -, @\}$ can be considered as simpler than the binary classification problem P1 with super classes $\{+\}$ and $\{*, -, @\}$. We note that there exist several measures for mixture identifiability. A recent example of such a measure is the p-value function proposed in (Zhou et al., 2017).

The problem of error dependency of binary classifiers means that some of the binary classifiers often err simultaneously. This increases the number of errors in instance code words which in turn can result in instance misclassifications (if the number of errors per query instance exceeds the correction number *CN*). We note that the problem of error dependency of binary classifiers can be quite persistent, since it can cause misclassifications even if the problem of difficult binary classification problems is not well present.

Approaches to avoid difficult binary classification problems are based on a search for coding matrices whose columns represent relatively simple classification problems. They are divided in two groups: approaches that adapt the coding matrices to the data and approaches that learn the coding matrices from the data.

The approaches in the first group assume a presence of initial problem-independent coding matrices (e.g. standard exhaustive ECOC matrices). The process of adaptation of these matrices can follow different scenarios. Below we describe the main approaches within this group.

The thinned-ECOC approach is a heuristical approach to coding matrix adaptation proposed in (Hatemi, 2012). It adapts the initial coding matrix by removing its columns that correspond to difficult binary classification problems, i.e. columns that induce binary classifiers with a low generalization performance. The process of adaptation employs the iterative thinning algorithm introduced in (Banfield et al., 2004). In each iteration the algorithm first computes the generalization performance of each binary classifier on a validation set. Then it compares the generalisation performances of the binary classifiers with the generalization performance of the ensemble (ecoc) classifier. It then removes the column of the coding matrix that corresponds to the binary classifier with the lowest generalization performance. The algorithm stops when a predefined number of columns has been removed and then it trains the ensemble based on the final reduced coding matrix. Although good experimental results were reported, the thinned-ECOC approach suffers from several problems. It is sensitive to the validation set and to the parameter that indicates the number of columns that can be removed. In addition, the thinned-ECOC approach is computationally inefficient, since the generalization performance of binary classifiers need to be computed for the validation set in each iteration.

(Escalera and Pujol, 2006)) and (Marom et al., 2010) proposed independently two similar heuristical approaches to coding matrix adaptation. These approaches perform in an opposite manner compared with the thinned-ECOC approach. Given an initial problem-independent coding matrix, they add news columns to the matrix if these columns represent relatively simple binary problems. The process of adaptation is iterative. At each iteration first a heuristical function determines a new column to be added. This is done by analysing a confusion matrix of the ensemble based on the current coding matrix on a separate validation set. Then, a column is added and the ensemble is retrained by employing the updated coding matrix. The process of adaptation stops when there is no further improve in the confusion matrix. The ensembles based on the final adapted coding matrices showed some improvement over the ensembles based on the initial coding matrices. However, we note that the process of adaptation is sensitive to the validation set (like the thinned-ECOC approach) and the heuristical functions for choosing columns. In addition, it is computationally expensive, since the confusion matrix of the ensemble is estimated on a validation set in each iteration.

Zor et al. (2010) proposed a heuristic approach to coding matrix adaptation that modifies the matrix columns. If the performance of a binary classifier for a class is worse than random, then the column bit that corresponds to that class is considered for flipping. The flipping process is realized iteratively starting from the class with the worst performance. At each iteration the influence of the flipped bit is estimated using the generalization performance on a validation set of the ensemble based on the current coding matrix. The bit is flipped if the generalization performance is improved. The process stops when all the problematic bits have been visited. The approach in (Zor et al., 2010) showed some improvement over ensembles based on the ECOC class decomposition schemes. However, it is sensitive to the validation set and is computationally inefficient (since for each bit change we need to test an ensemble on the validation set).

The second group of approaches to avoid difficult binary classification problems are those approaches that learn the coding matrices from the data. The main representative in this group is the Discriminant ECOC approach proposed in (Pujol et al., 2006). This heuristic approach first learns a column code binary tree from the data. The column code binary tree is a tree where each non-terminal node is a binary partition of a class set from the partition associated with the parent node. The partition from each non-terminal node is chosen to be the most discriminant. This is realized using a floating search based on the fast quadratic mutual information between instances and the binary labels induced by the partition. Once the column code binary tree has been learned, it is encoded in a matrix.

This matrix is the coding matrix used to train the final ensemble. The experiments showed that the Discriminant ECOC approach learns relatively short codes and can outperform one-against-all ensembles. The short codes are definitely due to the column code binary trees. However, it is questionable whether the tree-based codes are the most optimal for the data, since they are not redundant. In addition, learning trees is an unstable process: small mistakes especially close to the root can result in incorrect trees which in turn causes incorrect codes w.r.t. the data.

The problem of error dependency of the binary classifiers has been addressed by several authors. The approaches to avoid this problem can be divided into two groups. The first group includes approaches that reduce the error dependency by learning a coding matrix from the data. The second group includes approaches that reduce the error dependency when training the binary classifiers.

The oldest approach in the first group is the one proposed by Kuncheva and Whitaker (2003a). It is based on an assumption that to solve the problem of error dependency of the binary classifiers we need to diversify these classifiers. Kuncheva and Whitaker (2003a) introduced a heuristic measure for classifier diversity in coding matrices. To optimize this measure she proposed an evolutionary approach. Kuncheva showed experimentally that her approach can boost the diversity of binary classifiers and thus often the generalization performance of the final ensembles. However, the approach is computationally inefficient (like any other evolutionary approach). In addition, it has to be used with care, since diversity does not always imply better generalization performance of the ensembles (Kuncheva and Whitaker, 2003b).

Zhang et al. (2009) proposed one of the most advance approaches based on coding matrix learning. The approach first builds a class graph that reflects the similarities between the classes using support vector machines. Then, it computes the normalized Laplacian of this graph. The eigenvectors of the Laplacian are discretized and in this was they form the coding matrix. Since the eigenvectors are orthogonal, the columns of the coding matrix are highly uncorrelated. This implies that error dependency of the binary classifiers is minimized which boosts the generalization performance of the final ensemble. The latter was shown experimentally on several real-life datasets.

The second group of approaches reduces the error dependency when training the binary classifiers based on fixed coding matrices. Below we provide two main approaches within the group.

Liu et al. (2015) proposed an approach to jointly learning the binary classifiers. Under the assumption that the binary classifiers are linear, they introduced an objective function that takes into account the classifiers' data fit and dependency. The classifiers are trained simultaneously by minimizing that objective function. A regularization parameter is used to control the trade-off between the accuracy of the binary classifiers and their independence. Thus, in contrast with (Zhang et al., 2009) this approach does not boost the accuracy and independence of the binary classifiers simultaneously.

Furnkranz and Park (2012) proposed to use a multi-output classifier instead of the set of the binary classifiers used in the standard setting. They showed that if the multi-output classifier can handle the dependency between the outputs, the generalization performance of the final ensemble can improve. Therefore, this work allows transferring any progress in multi-output prediction to multiclass decomposition schemes.

If we summarize the related work, we can state that the current approaches to avoid difficult binary classification problems and error dependency of binary classifiers include a repertoire of techniques that employ:

- adapting/learning coding matrices (Hatemi, 2012; Escalera and Pujol, 2006; Marom et al., 2010; Zor et al., 2010; Kuncheva and Whitaker, 2003a; Zhang et al., 2009);
- modifying decoding rules (Escalera et al., 2008; Smith and Windeatt, 2010); or
- reducing dependency of the binary classifiers (Liu et al., 2015; Furnkranz and Park, 2012).

In this chapter we aim at extending the repertoire of these techniques. We propose instance decomposition schemes as an alternative to class decomposition schemes. We show that when instance decomposition schemes are properly initialized they can solve simultaneously the problem of difficult binary classification problems and the problem of error dependency of binary classifiers.

3.2 Instance-based Decomposition Schemes

This section introduces instance decomposition schemes for mapping any multiclass classification problem into a set of binary classification problems. Subsection 3.2.1 formalizes *instance decomposition schemes and coding matrices*. Subsection 3.2.2 provides a detailed explanation of *the encoding and decoding stages*. Subsection 3.2.3 explains how to initialize coding matrices of the instance decomposition schemes.

3.2.1 Instance Decomposition Schemes

Consider a multiclass classification problem *MCP* with a class set Y of size $K > 2$[1] . To decompose *MCP* into B binary classification problems BCP_b ($b \in \{1, \dots, B\}$) we introduce the notion of a *binary instance partition*.

[1] In this chapter instance decomposition schemes are considered for multiclass classification only. Extensions of instance decomposition schemes for binary classification problems will be addressed in future research.

Definition 9. (Binary Instance Partition) *Given data $D \subseteq X \times Y$ with N instances and an integer $b \in \{1, \dots, B\}$, the set $P_b(D)$ is said to be a binary instance partition of D iff $P_b(D)$ consists of two sets D_b^- and D_b^+ such that $D_b^- \cup D_b^+ = D$ and $D_b^- \cap D_b^+ = \emptyset$.*

Since the sets D_b^- and D_b^+ are non-empty, they have their own meaning derived from the meaning of the classes of the instances they combine. Thus, without loss of generality we consider the set D_b^- as a *negative set* and the set D_b^+ as a *positive set* for the binary classification problem *BCP*$_b$.

Definition 10. (Label Set) *The label set Y_b of a binary instance partition $P_b(D)$ is defined equal to $\{0, 1\}$, where 0 is the label of the instances in the negative set D_b^- of $P_b(D)$ and 1 is the label of the instances in the positive set D_b^+ of $P_b(D)$.*

Definition 9 allows us to introduce *instance decomposition schemes*. An instance decomposition scheme describes how to decompose a multiclass classification problem *MCP* into B binary classification problems *BCP*$_b$ ($b \in \{1, \dots, B\}$), as given in Definition 11.

Definition 11. (Instance Decomposition Scheme) *Given a multiclass classification problem MCP and a positive integer $B > 0$, an* instance decomposition scheme *of MCP is a set SP(D) of B different binary instance partitions $P_b(D)$ with $b \in \{1, \dots, B\}$.*

Any instance decomposition scheme can be represented by a coding matrix.

Definition 12. (Coding Matrix) *The coding matrix of an instance decomposition scheme SP(D) is a binary $N \times B$ matrix M iff for any $n \in \{1, \dots, N\}$ and $b \in \{1, \dots, B\}$:*

$$M(n, b) = \begin{cases} 0 \in Y_b & \textit{if } (x_n, y_n) \in D_b^-, \\ 1 \in Y_b & \textit{if } (x_n, y_n) \in D_b^+, \end{cases}$$

where $D_b^- \in P_b(D)$ and $D_b^+ \in P_b(D)$.

A row $M(n, \cdot)$ in the coding matrix M corresponds to a particular labeled instance $(x_n, y_n) \in D$ for $n \in \{1, \dots, N\}$ and it forms the *code word* of that instance, i.e. the instance code word (see Figure 3.2). We note that the meaning of any two bits $M(n, b_1)$ and $M(n, b_2)$ in an instance code word is different for $b_1 \neq b_2$, since they correspond to different binary instance partitions. Additionally, the instance code words are not restricted: the code words $M(n_1, \cdot)$ and $M(n_2, \cdot)$ of two different instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ can be either the same or different. To characterize the similarity/dissimilarity of the code words we introduce the notions of the row similarity set and the row distance set.

Definition 13. (Row sets) *Given a coding matrix M,*

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 3.2: Matrix M of an instance decomposition scheme $SP(D)$ for 4 classes. M is initialized according to the one-against-all class decomposition scheme. The first three rows correspond to the three instances of class y_1. The next two rows correspond to the two instances of class y_2, and so on.

(a) the row similarity set $RS_M(n_1, n_2)$ for any two instance indices $n_1, n_2 \in \{1, \ldots, N\}$ is the set of indices $b \in \{1, \ldots, B\}$ such that $M(n_1, b) = M(n_2, b)$, and

(b) the row dissimilarity set $RD_M(n_1, n_2)$ for any two instance indices $n_1, n_2 \in \{1, \ldots, N\}$ is the set of indices $b \in \{1, \ldots, B\}$ such that $M(n_1, b) \neq M(n_2, b)$.

The row similarity set RS_M for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ is the complement of the row distance set RD_M to the set $\{1, \ldots, B\}$. The size of the row distance set RD_M is the Hamming distance $dist_H$ between the codewords of these instances in the coding matrix M.

A column $M(\cdot, b)$ in the coding matrix M corresponds to a particular binary instance partition $P_b(D)$, for $b \in \{1, \ldots, B\}$, and it forms the *code word* of that partition, i.e. the partition code word (see Figure 3.2). By Definition 11 any two partition code words $M(\cdot, b_1)$ and $M(\cdot, b_2)$ are different if $b_1 \neq b_2$. To characterize the similarity/dissimilarity of the partition code words we introduce the notions of the column similarity set and the column dissimilarity set .

Definition 14. (Column sets) *Given a coding matrix M,*

(a) the column similarity set $CS_M(b_1, b_2)$ for any two partition indices $b_1, b_2 \in \{1, \ldots, B\}$ is the set of the indices $n \in \{1, \ldots, N\}$ such that $M(n, b_1) = M(n, b_2)$, and

(b) the column dissimilarity set $CD_M(b_1, b_2)$ for any two partition indices $b_1, b_2 \in \{1, \ldots, B\}$ is the set of the indices $n \in \{1, \ldots, N\}$ such that $M(n, b_1) \neq M(n, b_2)$.

The column similarity set $CS_M(b_1, b_2)$ for any two partition indices $b_1, b_2 \in \{1, \ldots, B\}$ is the complement of the column distance set $CD_M(b_1, b_2)$ to the set $\{1, \ldots, N\}$. The size of the column distance set $CD_M(b_1, b_2)$ is the Hamming distance $dist_H$ between the partition codewords with indices b_1 and b_2 in the coding matrix M.

3.2.2 Encoding and Decoding

To solve a multiclass classification problem *MCP* we need to employ two instance decomposition schemes: *encoding* instance decomposition scheme $SP^e(D)$ and *decoding* instance decomposition scheme $SP^d(D)$ (see Figure 3.3). Prior to the encoding stage $SP^e(D)$ is used: (1) to represent *MCP* via a set of binary classification problems BCP_b ($b \in \{1, \ldots, B\}$) and (2) to train binary classifiers for those problems. During the encoding stage a code word of a query instance is formed by the predictions provided by the binary classifiers. During the decoding stage the code word is used to decode the instance class. This is realized by finding the class of an instance with code word closest to the code word of the query instance in the coding matrix M^d of the decoding instance decomposition scheme $SP^d(D)$. Below we provide a detailed description of the stages considered.

$$M^e = \left(\begin{array}{ccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}\right) \qquad
M^d = \left(\begin{array}{ccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
1 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline
0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}\right)$$

Figure 3.3: **Left:** encoding matrix M^e of an encoding instance decomposition scheme $SP^e(D)$ for 4 classes. M^e is initialized according to the exhaustive ECOC class decomposition scheme (eECOC). The first two rows correspond to the two instances of class y_1. The next three rows correspond to the three instances of class y_2, and so on. **Right:** decoding matrix M^d of the decoding instance decomposition scheme $SP^d(D)$.

Prior to the encoding stage we first generate for each $b \in \{1, \ldots, B\}$ a binary classification problem BCP_b using the *coding* matrix M^e of the encoding instance decomposition scheme $SP^e(D)$. Any BCP_b is determined by code word $M^e(\cdot, b)$ of partition $P^e_b(D) \in SP^e(D)$. The data D_b of BCP_b is formed in $X \times Y_b$. More precisely, any instance $(x_n, y_n) \in D$ ($n \in \{1, \ldots, N\}$) is transformed to a new instance $(x_n, M^e(n, b)) \in D_b$. Once the binary classification problems BCP_b have been set, we train a binary classifier $f_b : X \to Y_b$ for each BCP_b. The binary classifiers f_b then form an ensemble classifier $f : X \to Y$ equal to $\{f_b\}_{b=1}^{B}$ with a classification rule defined below.

During the encoding stage, given a query instance $x_{N+1} \in X$ and an ensemble classifier f, we form the code word $f(x_{N+1})$ of x_{N+1} in the code space $\mathcal{C}$ from the predictions of the binary classifiers $f_b \in f$ (see Section 2.2 for the definition of $\mathcal{C}$). Then, during the decoding stage the code word $f(x_{N+1})$ is used to decode the instance class. For that purpose we employ the coding matrix M^d of the *decoding* instance decomposition scheme $SP^d(D)$. We apply the decoding function for instance decomposition schemes that maps $f(x_{N+1})$ into a set of instances (x_n, y_n) which code words $M^d(n, \cdot)$ in M^d are the closest.

Definition 15. (Decoding Function for Instance Decomposition Schemes) *Given a code word $f(x_{N+1})$ for an instance $x \in X$, the decoding function g_I is a function from $\mathcal{C}$ to $2^{X \times Y}$ defined equal to:*

$$\underset{(x_n, y_n) \in X \times Y}{\operatorname{argmin}} \; dist\,(f(x), M\,(n, \cdot))\,,$$

where dist is a distance measure on the code space $\mathcal{C}$.

The set $\hat{Y}$ of the final estimates of the true class for the query instance x_{N+1} is computed as a set of classes with majority of the instances in $g_I(f(x_{N+1}))$. More precisely,

$$\hat{Y} = \underset{y \in Y}{\operatorname{argmax}}(\#\{(x_n, y_n) \in g_I(f(x_{N+1}))|y_n = y\}).$$

Since the binary classifiers are discrete, the set $\hat{Y}$ can actually contain more than one element. Thus, the final class estimate $\hat{y}$ of instance x_{N+1} is chosen randomly from the classes in $\hat{Y}$.

We note that the classification rule we presented uses the decoding matrix M^d which can be different from encoding matrix M^e. To our knowledge, this is the first work that distinguishes the matrix used in the encoding step and the matrix used in the decoding step. In the next section we show how to initialize these matrices.

3.2.3 Initialization

The generalization performance of the ensemble f is sensitive to the initialization of instance decomposition schemes $SP^e(D)$ and $SP^d(D)$. Below we propose a simple procedure for initializing the coding matrices M^e and M^d of these schemes.

Definition 16. (Matrix Initialization Procedure)

(1) Given a coding matrix M of a class decomposition scheme, the coding matrix M^e of $SP^e(D)$ is initialized such that:

$$(\forall\, n \in \{1, \ldots, N\})M^e(n, \cdot) = M(n, \cdot).$$

(2) The decoding matrix M^d of $SP^d(D)$ is initialized such that:

$$(\forall n \in \{1, \ldots, N\}, b \in \{1, \ldots, B\})(M^d(n, b) = f_b(x_n)).$$

By Definition 16 the coding matrix M^e of the encoding instance decomposition scheme $SP^e(D)$ is initialized according to the coding matrix M of some *class* decomposition scheme (e.g., exhaustive/minimal/random ECOC etc. (Escalera et al., 2010)). This means that the instances of a class have the same code words and these words differ from the code words of the instances of the remaining classes.

By Definition 16 the coding matrix M^d of the decoding instance decomposition scheme $SP^d(D)$ is initialized using the predictions of the binary classifiers $f_b \in f$. Thus, *the key feature of the initialization procedure is that we learn* $SP^d(D)$. We note that this is done by first training binary classifiers $f_b \in f$ using the encoding instance decomposition scheme $SP^e(D)$ and then by applying these classifiers over the instances of D. Hence, the coding matrix M^d of $SP^d(D)$ consists of only those instance code words that are achievable through binary classifiers $f_b \in f$.

3.3 Error Correction Analysis

This section studies the error correction capability of the instance decomposition schemes initialized by the procedure from Definition 16. It first analyzes the separation properties of the coding matrices and then determines the types of errors that can be handled.

3.3.1 Separation Properties

The separation properties of the coding matrix M^e of an encoding instance decomposition scheme $SP^e(D)$ depends on the coding matrix M of the initialization class decomposition scheme (see Definition 16). The matrix M has to be chosen so that the matrix M^e is both, row-separated and column-separated, in terms of the Hamming distance (Dietterich and Bakiri, 1995). The row separation requirement comes from the fact that M^e is used to learn the decoding matrix M^d (through the binary classifiers). The column separation requirement comes from the fact that the binary classifiers should be as error independent as possible.

The minimum Hamming distance between any two rows of the encoding matrix M^e is equal to the size of the minimal row distance set $RD_{M^e}(n_1, n_2)$ over any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$. By Definition 16 it follows that this distance is equal to the minimal row Hamming distance of the coding matrix M of the class decomposition scheme used for initializing M^e.

The coding matrix M^d of the decoding instance decomposition scheme $SP^d(D)$ has to be well row-separated only. This is because M^d is used to decode binary predictions to form the final class estimate and is not used to train binary classifiers.

To characterize the minimal row Hamming distance in the matrix M^d of the *decoding* instance decomposition scheme $SP^d(D)$ we note that the binary classifiers may err differently for any two instances. Hence, we introduce the error set for an instance.

Definition 17. (Error set) *The error set $E_{M^e}(n)$ for an instance $(x_n, y_n) \in D$ is the set of indices $b \in \{1, \ldots, B\}$ of the binary classifiers $f_b \in f$, trained through the encoding matrix M^e, that err for that instance; i.e.*

$$E_{M^e}(n) = \{b \in \{1, \ldots, B\} \mid M^e(n, b) \neq M^d(n, b)\}.$$

Using error sets we can compute the row dissimilarity set for any two instances in the the decoding matrix M^d. The computation is realized according to Theorem 1 given below.

Theorem 1. *The row dissimilarity set $RD_{M^d}(n_1, n_2)$ for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ w.r.t. the decoding matrix M^d equals:*

$$\Big[RD_{M^e}(n_1, n_2) \setminus \big[[E_{M^e}(n_1) \setminus E_{M^e}(n_2)] \cup [E_{M^e}(n_2) \setminus E_{M^e}(n_1)]\big]\Big] \cup$$
$$\Big[RS_{M^e}(n_1, n_2) \cap \big[[E_{M^e}(n_1) \setminus E_{M^e}(n_2)] \cup [E_{M^e}(n_2) \setminus E_{M^e}(n_1)]\big]\Big].$$

Proof. By Definition 13, for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$:

$$RD_{M^d}(n_1, n_2) = \{b \in \{1, \ldots, B\} \mid M^d(n_1, b) \neq M^d(n_2, b)\}.$$

Since $RD_{M^e}(n_1, n_2) \cup RS_{M^e}(n_1, n_2) = \{1, \ldots, B\}$, we have:

$$\begin{aligned} RD_{M^d}(n_1, n_2) \quad = \quad & \{b \in RD_{M^e}(n_1, n_2) \mid M^d(n_1, b) \neq M^d(n_2, b)\} \cup \\ & \{b \in RS_{M^e}(n_1, n_2) \mid M^d(n_1, b) \neq M^d(n_2, b)\}. \qquad (3.1) \end{aligned}$$

The sets in the right had side of equation (3.1) are disjointed and, thus, we consider them separately. We start with the first set that we express using the error sets $E_{M^e}(n_1)$ and $E_{M^e}(n_2)$. This is done as follows:

$$\begin{aligned} & \{b \in RD_{M^e}(n_1, n_2) \mid M^d(n_1, b) \neq M^d(n_2, b)\} \\ = \quad & \{b \in RD_{M^e}(n_1, n_2) \mid (M^e(n_1, b) = M^d(n_1, b) \wedge M^e(n_2, b) = M^d(n_2, b)) \vee \\ & \qquad (M^e(n_1, b) \neq M^d(n_1, b) \wedge M^e(n_2, b) \neq M^d(n_2, b))\} \\ = \quad & \{b \in RD_{M^e}(n_1, n_2) \mid (M^e(n_1, b) = M^d(n_1, b) \vee M^e(n_2, b) \neq M^d(n_2, b)) \wedge \end{aligned}$$

$$
\begin{aligned}
& \qquad (M^e(n_1,b) \neq M^d(n_1,b) \vee M^e(n_2,b) = M^d(n_2,b))\} \\
= \;& \{b \in RD_{M^e}(n_1,n_2) \mid \neg((M^e(n_1,b) \neq M^d(n_1,b) \wedge M^e(n_2,b) = M^d(n_2,b)) \vee \\
& \qquad (M^e(n_1,b) = M^d(n_1,b) \wedge M^e(n_2,b) \neq M^d(n_2,b)))\} \\
= \;& RD_{M^e}(n_1,n_2) \setminus \big[[E_{M^e}(n_1) \setminus E_{M^e}(n_2)] \cup [E_{M^e}(n_2) \setminus E_{M^e}(n_1)]\big] \qquad (3.2)
\end{aligned}
$$

We continue the proof for the second set from the right had side of equation (3.1).

$$
\begin{aligned}
& \{b \in RS_{M^e}(n_1,n_2) \mid M^d(n_1,b) \neq M^d(n_2,b)\} \\
= \;& \{b \in RS_{M^e}(n_1,n_2) \mid M^e(n_1,b) \neq M^d(n_1,b) \oplus M^e(n_2,b) \neq M^d(n_2,b)\} \\
= \;& \{b \in RS_{M^e}(n_1,n_2) \mid (M^e(n_1,b) \neq M^d(n_1,b) \vee M^e(n_2,b) \neq M^d(n_2,b)) \wedge \\
& \qquad \neg(M^e(n_1,b) \neq M^d(n_1,b) \wedge M^e(n_2,b) \neq M^d(n_2,b))\} \\
= \;& \{b \in RS_{M^e}(n_1,n_2) \mid (M^e(n_1,b) \neq M^d(n_1,b) \vee M^e(n_2,b) \neq M^d(n_2,b)) \wedge \\
& \qquad (M^e(n_1,b) = M^d(n_1,b) \vee M^e(n_2,b) = M^d(n_2,b))\} \\
= \;& \{b \in RS_{M^e}(n_1,n_2) \mid (M^e(n_1,b) \neq M^d(n_1,b) \wedge M^e(n_2,b) = M^d(n_2,b)) \wedge \\
& \qquad (M^e(n_1,b) = M^d(n_1,b) \wedge M^e(n_2,b) \neq M^d(n_2,b))\} \\
= \;& RS_{M^e}(n_1,n_2) \cap \big[[E_{M^e}(n_1) \setminus E_{M^e}(n_2)] \cup [E_{M^e}(n_2) \setminus E_{M^e}(n_1)]\big] \qquad (3.3)
\end{aligned}
$$

By substituting the new definitions (3.2) and (3.3) in the right hand side of equation (3.1) the theorem is proven. □

The minimal row Hamming distance in the decoding matrix M^d is equal to the size of the minimal set $RD_{M^d}(n_1, n_2)$ over any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ given $y_{n_1} \neq y_{n_2}$. Due to non-uniform generalization performance of the binary classifiers there is no analytic way to express this distance in general. However, for particular row submatrices of the decoding matrix M^d the minimal row Hamming distance can be derived. Below we provide three possible cases of such matrices.

Case 1: *There exists a row submatrix M^d_r of the decoding matrix M^d such that for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ we have $E_{M^e}(n_1) = E_{M^e}(n_2)$.*

In this case by Theorem 1 for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ the row dissimilarity set $RD_{M^d}(n_1, n_2)$ in M^d_r equals the row dissimilarity set $RD_{M^e}(n_1, n_2)$ of the coding row submatrix M^e_r. The latter is a row submatrix of the encoding matrix M^e that corresponds to the same instances as M^d_r. Thus, the minimal row Hamming distance of the encoding row submatrix M^e_r equals that of the decoding row submatrix M^d_r. By Definition 16 the minimal row Hamming distances in M^e and the coding matrix M of the class decomposition scheme used for initializing M^e coincide. Thus, the minimal row Hamming distance in M_r^d equals the minimal row Hamming distance among codewords in M of classes present in M_r^d.

Example 1. *Take row 2 (class y_1) and row 4 (class y_2) in M^e and M^d in Figure 3.3. Then $E_{M^e}(2) = \{1, 2, 3\}$, and $E_{M^e}(4) = \{1, 2, 3\}$. Thus, $RD_{M^e}(2, 4) = RD_{M^d}(2, 4) = \{1, 2, 3, 4\}$.*

Case 2: *There exists a row submatrix M_r^d of the decoding matrix M^d such that for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ we have $E_{M^e}(n_1) \cap E_{M^e}(n_2) = \emptyset$ and $E_{M^e}(n_1) \cup E_{M^e}(n_2) = RS_{M^e}(n_1, n_2)$.*

In this case by Theorem 1 for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ the row dissimilarity set $RD_{M^d}(n_1, n_2)$ in M_r^d becomes maximum (i.e., it is equal to $\{1, \ldots, B\}$). Thus, the minimal row Hamming distance in the decoding row submatrix matrix M_r^d equals B.

Example 2. *Take row 6 (class y_3) and row 7 (class y_4) in M^e and M^d in Figure 3.3. Then $RS_{M^e}(6, 7) = \{1, 4, 5\}$, $E_{M^e}(6) = \{1\}$, and $E_{M^e}(7) = \{4, 5\}$. Thus, $RD_{M^d}(6, 7) = \{1, 2, 3, 4, 5, 6, 7\}$.*

Case 3: *There exists a row submatrix M_r^d of the decoding matrix M^d such that for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ we have $E_{M^e}(n_1) \cap E_{M^e}(n_2) = \emptyset$ and $E_{M^e}(n_1) \cup E_{M^e}(n_2) = RD_{M^e}(n_1, n_2)$.*

In this case by Theorem 1 for any two instances $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}) \in D$ the row dissimilarity set $RD_{M^d}(n_1, n_2)$ in M_r^d is minimal (i.e., it is equal to $\emptyset$). Thus, the minimal row Hamming distance in the decoding row submatrix M_r^d equals 0.

Example 3. *Take row 5 (class y_2) and row 8 (class y_4) in M^e and M^d in Figure 3.3. Then $RD_{M^e}(5, 8) = \{2, 4, 5, 7\}$, $E_{M^e}(5) = \{4\}$, and $E_{M^e}(8) = \{2, 5, 7\}$. Thus, $RD_{M^d}(5, 8) = \emptyset$.*

3.3.2 Error Handling

Using the results we received so far we discuss below when instance decomposition schemes can handle errors of the binary classifiers. We analyze two basic scenarios.

In the first scenario we consider a subset $S \subseteq D$ of instances (x_n, y_n) such that the error set E_{M^e} is the same for all the instances in S. According to Definition 17 the binary classifiers from the set $\{f_b | b \in E_{M^e}\}$ err simultaneously on the instances from S. This implies by Theorem 1 that the row distance sets RD_{M^e} and RD_{M^d} for any two instances from S stay equal (see Case 1). Thus, the minimal Hamming distance between the codewords in M^d of different-class instances in S equals the minimal Hamming distance between the codewords in M^e of these instances. This means that if the encoding matrix M^e has been initialized with an error-correcting property, this property is preserved between the codewords in M^d of any two instances from S that belong to different classes. Thus, we conclude that:

(1) the decoding matrix M^d of the instance decomposition scheme $SP^d(D)$ can handle the simultaneous errors of the binary classifiers.

In the second scenario we consider a subset $S_1 \subseteq D$ of instances (x_{n_1}, y_{n_1}) with common error set $E_{M_1^e}$ and a subset $S_2 \subseteq D$ of instances (x_{n_2}, y_{n_2}) with common error set $E_{M_2^e}$ such that $E_{M_1^e} \cap E_{M_2^e} = \emptyset$. According to Definition 17 the binary classifiers from the sets $\{f_b | b \in E_{M_1^e}\}$ and $\{f_b | b \in E_{M_2^e}\}$ err non-simultaneously on the instances from $S_1 \cup S_2$. By Theorem 1 for any two different-class instances $(x_{n_1}, y_{n_1}) \in S_1$ and $(x_{n_2}, y_{n_2}) \in S_2$:

(a) the error subsets $E_{M^e}(n_1) \cap RD_{M^e}(n_1, n_2)$ and $E_{M^e}(n_2) \cap RD_{M^e}(n_1, n_2)$ decreases the row dissimilarity set $RD_{M^d}(n_1, n_2)$;

(b) the error subsets $E_{M^e}(n_1) \cap RS_{M^e}(n_1, n_2)$ and $E_{M^e}(n_2) \cap RS_{M^e}(n_1, n_2)$ increases the row dissimilarity set $RD_{M^d}(n_1, n_2)$.

The decrease in the size of the row dissimilarity set $RD_{M^d}(n_1, n_2)$ caused by the error subsets $E_{M^e}(n_1) \cap RD_{M^e}(n_1, n_2)$ and $E_{M^e}(n_2) \cap RD_{M^e}(n_1, n_2)$ can be compensated by the increase in the size of the same set caused by the error subsets $E_{M^e}(n_1) \cap RS_{M^e}(n_1, n_2)$ and $E_{M^e}(n_2) \cap RS_{M^e}(n_1, n_2)$. This means that for any two different-class instances $(x_{n_1}, y_{n_1}) \in S_1$ and $(x_{n_2}, y_{n_2}) \in S_2$ the size of the row dissimilarity set $RD_{M^d}(n_1, n_2)$ can stay close to that of the size of the row dissimilarity set $RD_{M^e}(n_1, n_2)$. Thus, the minimal Hamming distance between the codewords in M^d of different-class instances in $S_1 \cup S_2$ can stay close to the minimal Hamming distance between the codewords in M^e of these instances. This means that if the encoding matrix M^e has been initialized with an error-correcting property, this property can be preserved between the codewords in M^d of the instances from $S_1 \cup S_2$. Thus, we can conclude that:

(2) the decoding matrix M^d of the instance decomposition scheme $SP^d(D)$ can handle the non-simultaneous errors of the binary classifiers.

The extent the non-simultaneous errors of the binary classifiers can be handled depends on the sizes of the row distance sets $RD_{M^d}(n_1, n_2)$ in the decoding matrix M^d. From points (a) and (b) (see above) it clear that these sizes depend on the sizes of the row distance sets RD_{M^e} and row similarity sets RS_{M^e} in the encoding matrix M^e. More precisely, the sizes of the row distance sets $RD_{M^d}(n_1, n_2)$ decrease with the sizes of the row distance sets RD_{M^e} and they increase with the size of the row similarity sets RS_{M^e}. For the worst case when the instance decomposition schemes are initialized using the exhaustive ECOC class decomposition scheme the sizes of the sets RD_{M^e} equal 2^{K-2} and the sizes of the sets RS_{M^e} equal $2^{K-2} - 1$. For any other case when the instance decomposition schemes are initialized using the non-exhaustive class decomposition schemes the sizes of the sets RD_{M^e} are always smaller than those of the sets RS_{M^e}. Thus, we may conclude that the non-simultaneous errors of the binary classifiers can be handled better when the instance decomposition schemes are initialized using the non-exhaustive class decomposition schemes.

We note that difficult classification problems can cause both simultaneous errors and non-simultaneous errors while error dependency of the binary classifiers causes essentially simultaneous errors. Thus, our main conclusion is that the instance decomposition schemes are capable of handling difficult classification problems and error dependency of the binary classifiers.

3.4 Experiments

This section studies ensembles based on the class decomposition schemes and ensembles based on the instance decomposition schemes experimentally. It first provides the experimental setup, and, then, the results and discussion.

3.4.1 Experimental Setup

The experiments had the following setup. The decomposition schemes were chosen depending on the number of classes (K) in the datasets considered. That is why, to reduce the computational cost we employed two types of the class decomposition schemes: the exhaustive ECOC class decomposition schemes (eECOC) for $K \leq 7$ and the random ECOC class decomposition schemes (rECOC) for $K > 7$. For the same reason, we employed two types of the instance decomposition schemes (IDS): the instance decomposition schemes initialized with eECOC for $K \leq 7$ and the instance decomposition schemes initialized with rECOC for $K > 7$. The binary classification algorithm was Logistic Regression (le Cessie and van Houwelingen, 1992). 13 multiclass datasets taken from the UCI repository (Bache and Lichman, 2013) were used in our experiments. They are summarized in Table 3.1. The evaluation method was 10-fold cross validation averaged over 10 runs. The classification accuracy of the classifiers was compared using the paired t-test (Nadeau and Bengio, 2001) at the 5% significance level.

Table 3.1: The UCI datasets used in the experiments.

Dataset	**# Instances**	**# Attributes**	**# Classes**
Balance	625	4	3
Iris	150	4	3
Thyroid	215	5	4
Car	1728	6	4
Vehicle	846	18	4
Dermatology	366	34	6
Glass	214	9	7
Segmentation	2310	19	7
Zoo	101	16	7
Ecoli	336	8	8
Mfeat-mor	2000	6	10
Optdigits	3823	64	10
Pendigits	7494	16	10

3.4.2 Experimental Results and Discussion

This subsection studies how the generalization performance of the ensembles based on ECOC and the ensembles based on IDS is influenced by the errors of the binary classifiers in those ensembles. The errors were systematically introduced via decreasing the complexity of the binary classifiers. Since the binary classifiers were logistic regressions, the complexity was decreased increasing the ridge parameter r.

Our first experiment was on the four-class car dataset from UCI (Bache and Lichman, 2013). We recorded the accuracy of the eECOC and IDS ensembles while decreasing complexity of the logistic regression binary classifiers. The complexity was controlled with the ridge parameter r from 1.0E-12 (high complexity) to 10000 (low complexity). To indicate the errors of all the binary classifiers, we employed the empirical probability of the joint error of the binary classifiers. Therefore, for each run of the ensembles (i.e. each value of r in the experiments) we recorded the empirical distribution of the joint-error probability of the logistic regression binary classifiers.

We note that the car dataset has 4 classes ($K = 4$). This implies that the eECOC ensemble can correct strictly at most one error of the binary classifiers ($2^{K-3} - 1 = 2^{4-3} - 1 = 1$). That is why, we plotted in Figure 3.4 the accuracy of eECOC and IDS ensembles against the empirical cumulative probability of $p(\#errors > 1)$. The figure shows that the accuracy of the IDS ensemble is bigger than that of the eECOC ensemble for $p(\#errors > 1)$ greater than 0.3. The IDS ensemble outperforms the eECOC ensemble for $p(\#errors > 1)$ equal to 0.16 when the accuracy difference is 10.17%. Then the accuracies of the ensembles gradually converge and for $p(\#errors > 1) = 0.3$ they are both equal to 70.02%. After this point the logistic regression binary

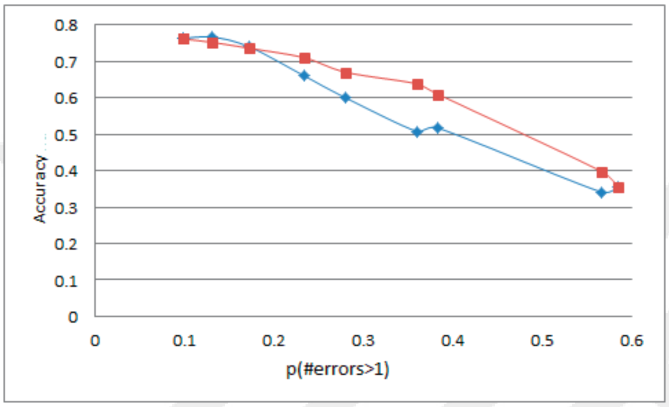


Figure 3.4: The accuracy vs the cumulative probability of $p(\#errors > 1)$ of the eECOC and IDS ensembles on the car data.

classifiers become majority-vote classifiers and the accuracies of the eECOC and IDS ensembles essentially are the same equal to 70.02%.

We observed similar behavior of the eECOC and IDS ensembles in the second set of experiments with 12 datasets from UCI (Bache and Lichman, 2013). The results are shown in Table 3.2. Table 3.2 shows the accuracy of the ensembles when the complexity of logistic regression binary classifiers decreases with the values of the ridge parameter r from 1 to 50. The IDS ensembles won in 64 out of 78 cases, 40 times significantly, and it lost in 8 out of 78 cases, 0 times significantly.

Table 3.2: The accuracy of the eECOC and IDS ensembles vs. complexity of logistic regression binary classifiers controlled by the ridge parameter r in $[1, 50]$. Bold numbers indicate statistically better results in group for a r-value.

	r=1		r=10		r=20		r=30		r=40		r=50	
Data	IDS	eECOC	IDS	eECOC	IDS	eECOC	IDS	eECOC	IDS	eECOC	IDS	eECOC
Balance	**91.67**	87.68	**91.67**	87.68	**91.67**	87.68	**91.67**	87.68	**91.67**	87.68	**91.67**	87.68
Iris	**95.60**	73.33	**90.40**	64.13	**90.00**	60.27	**88.93**	56.4	**86.13**	52.8	**82.8**	50.13
Thyroid	**95.5**	93.8	**94.24**	93.77	**94.17**	93.58	**94.08**	93.49	**94.02**	93.47	**94.00**	93.36
Vehicle	77.92	75.44	71.66	71.80	68.09	67.97	65.89	65.75	64.26	64.31	62.91	63.12
Glass	60.23	60.19	58.05	54.75	58.41	53.82	58.67	51.88	**60.69**	52.26	**60.74**	50.95
Dermatology	97.43	97.16	97.92	97.76	97.65	97.59	97.70	97.32	97.21	97.27	96.82	97.21
Segment	**90.25**	84.64	**87.61**	81.18	84.63	80.54	83.73	79.65	81.66	79.65	83.12	78.54
Ecoli	83.04	84.35	82.32	78.29	**79.23**	74.83	77.51	73.52	**77.86**	71.02	**78.64**	68.94
Zoo	90.29	90.69	89.89	88.11	89.89	87.71	**90.21**	83.18	**89.05**	81.78	**88.69**	79.44
Mfeat-mor	**66.32**	53.68	**65.92**	46.32	**64.08**	42.88	**63.60**	41.58	**64.64**	39.68	**67.68**	37.28
Pendigits	81.73	75.95	**83.03**	66.24	**81.41**	59.48	**79.56**	55.66	**79.08**	51.46	**77.98**	48.90
Optdigits	89.99	90.54	88.84	89.29	88.61	87.14	87.95	84.29	88.76	82.75	**88.42**	81.29

We conducted two-sided Wilcoxon's signed rank test (a.k.a Wilcoxons T test) for each ridge parameter value to test if the observed difference between the accuracy of IDS ensembles and that of eECOC ensembles is statistically significant when considering all

Table 3.3: The test statistics of Wilcoxon's signed rank test. Bold numbers indicate statistically significant results.

r	1	10	20	30	40	50
T_{Wilcox}	27	22	**0**	**0**	23	23

of the datasets Japkowicz and Shah (2014). Table 3.3 shows the test statistics we found. The critical value for two sided Wilcoxon's signed rank test at significance level 5% is 13, when the number of datasets is 12 (Japkowicz and Shah, 2014). The test statistic was below the critical value of 12, in fact 0, when the ridge parameter was set to 20 and 30. Therefore, for such large values of the ridge parameter, we rejected the null hypothesis that the IDS ensembles and the eECOC ensembles perform equally well. This means that the IDS ensembles and the eECOC ensembles are different, and the IDS ensembles are superior especially when complexity of the logistic regression binary classifiers decreases; i.e. when the number of binary errors increases.

3.5 Conclusion

In this chapter we proposed instance decomposition schemes (IDS) for solving multiclass classification problems. In contrast to class decomposition schemes, IDS consists of several instance partitions and thus any training instance is represented by a code word. Since the classes are given with instances, they are represented by the instance code words (i.e., one class is given with several code words). This allows for a richer class representation and more flexible decompositions.

The key feature of IDS is the usage of a separate encoding IDS and a separate decoding IDS. We proposed to initialize the encoding IDS according to some standard class decomposition scheme and to learn the decoding IDS from the data through the encoding IDS. In this case we showed that the errors caused by the difficulty of binary classification problems and error dependency of binary classifiers can be compensated. This is demonstrated in the experiments in which IDS outperformed ECOC-based class decomposition schemes.

The concept of IDS will be used in throughout this thesis. Chapter 6 will introduce an extension of IDS for domain adaptation (Pan and Yang, 2010) and Chapter 7 will provide a bioinformatics application.

4 Weighted Decoding via Fractional Programming and Bipartite Graph Partitioning

This chapter is based on the following publications:
Ismailoglu, F., Sprinkhuizen-Kuyper, I. G., Smirnov, E., Escalera, S. and Peeters, R. (2015). Fractional Programming Weighted Decoding for Error-Correcting Output Codes. In 12th Int.Workshop on Multiple Classifier Systems (MCS), LNCS, vol. 9132, pages 38-50. Springer International Publishing.

Ismailoglu, F. (2015). Weighted Decoding for Error-Correcting Output Codes via Bipartite Graph Partitioning. In Proc. of the 27th Benelux Conf. on Artificial Intelligence (BNAIC), Hasselt, Belgium.

This chapter addresses the problem of difficult binary classification problems from the perspective of weighting binary classifiers. We assume that binary classifiers that correspond to difficult binary classification problems are prone to err, while those that correspond to relatively easy binary classification problems tend to be more accurate. This suggests to weight binary classifiers according to their generalization performances. In doing so, one implicitly weights the binary classifiers based on how difficult the corresponding binary classifiers are. In class decomposition, such weighting is realized by means of weighted decoding (Escalera et al., 2008; Smith and Windeatt, 2010).

In this chapter we propose two novel weighted decoding algorithms: Fractional Programming Weighted decoding (in short FP.Weighted decoding) and Bipartite Graph Partitioning Weighted decoding (in short BGP.Weighted decoding). As opposed to previously proposed weighted decoding algorithms in the literature, the FP.Weighted decoding is a metric learning algorithm, since it learns a task-specific distance function employed in the decoding phase. Here the task is to learn a metric which produces a small value for the distance between an instance code word and its true class code word, whereas it produces larger values for the distances between the instance code word and the other class code words. Another distinguishing element of the FP.weighted decoding algorithm is that it estimates the weights of the binary classifiers by solving an optimization problem (a fractional programming problem) in contrast to the state-of-the-art algorithms proposed earlier

The second proposed weighted decoding algorithm, BGP.Weighted decoding, proceeds to estimate the weights from the binary classification problems themselves. It assumes that the smaller the distance between a positive super class and a negative super class is, the more difficult it is to handle the associated binary classification problem. To estimate the distance between a positive and a negative super class, the BGP.Weighted decoding algorithm utilizes the well-known inter-cluster distances from clustering, viewing the classes that form the super classes as clusters in the feature space. As a result, the computed weights are independent of the binary classifiers used, and therefore no recomputing is needed in case one switches to use a different binary classification algorithm. This is a novel property of the BGP.Weighted decoding algorithm.

The present chapter consists of five sections. In Section 4.1, we provide background information on weighted decoding and we briefly discuss the proposed decoding algorithms. Section 4.2 introduces the first proposed weighted decoding algorithm, FP.Weighted decoding. The optimization problem arising in the algorithm is of fractional programming type, and to explain it we provide a gentle introduction to fractional programming in Subsection 4.2.1. Section 4.3 introduces the second weighted decoding algorithm proposed in this chapter, BGP.Weighted decoding. In Section 4.4 we report on experimental results obtained for the proposed weighted decoding algorithms, for the unweighted decoding algorithm, and for two state-of-the-art weighted decoding algorithms. We evaluate these results and compare the performances in Subsection 4.4.2. Finally, in Section 4.5, we sum up the contributions and the limitations of the proposed weighted decoding algorithms.

4.1 Background and Related Work

As mentioned in the previous chapter, decomposing a multiclass classification problem into binary classification problems may leave one with difficult binary classification problems. In such a case, the generalization performances of the associated binary classifiers may be relatively poor. This in turn may result in instance misclassification. To tackle this problem, in addition to improving the encoding phase of class decomposition schemes as discussed in Chapter 3, one can also improve the decoding phase by weighting the outputs of binary classifiers. That is, one can perform weighted decoding.

In weighted decoding, a weight matrix W of size $K \times B$ is learned which has all its entries in the interval $[0, 1]$. The rows of W correspond to the K classes and its columns to the B binary classifiers. For every $y \in Y$ and $b \in \{1, \dots, B\}$, the entry $W(y, b)$ indicates the generalization performance of the binary classifier f_b over the class y. To estimate this generalization performance, different algorithms can be used (Escalera et al., 2008; Smith and Windeatt, 2010). Once W is estimated, it is used in conjunction with the distance function employed in the decoding phase to obtain weighted decoding (see Chapter 2 for the decoding phase of class decomposition schemes). From this point of view, it can be thought that weighted decoding takes the generalization performances of the binary classifiers into account during the decoding phase.

In weighted decoding, the Hamming distance function (7) and the loss-based distance function (8) are turned into a weighted Hamming distance function and a weighted loss-based distance function. The weighted Hamming distance is considered when the binary classifiers are of the discrete type. The weighted loss-based distance is considered when the binary classifiers are of the scoring type. Formal definitions of the weighted Hamming distance function and the weighted loss-based distance function are as follows.

Definition 18. (Weighted Hamming Distance Function) *Let M be a coding matrix, let $x \in X$ be an instance with a code word $f(x) \in \mathcal{C}$, and let $M(y, \cdot) \in \mathcal{C}$ be the code word for the class $y \in Y$ resulting from M. If $W \in [0, 1]^{K \times B}$ is a weight matrix, then the weighted Hamming distance between the code words $f(x)$ and $M(y, \cdot)$ is given by*

$$dist_{WH}(f(x), M(y, \cdot)) = \sum_{b=1}^{B} W(y, b) [\![f_b(x) \neq M(y, b)]\!],$$

where $[\![\;]\!]$ is the indicator function.

Definition 19. (Weighted Loss-based Distance Function) *Let M be a coding matrix, let $x \in X$ be an instance with a code word $f(x) \in \mathcal{C}$, and let $M(y, \cdot) \in \mathcal{C}$ be the code word for the class $y \in Y$ resulting from M. If $W \in [0, 1]^{K \times B}$ is a weight matrix and L is some loss function, then the weighted loss-based distance between the code words $f(x)$ and $M(y, \cdot)$ is given by*

$$dist_{WL}(f(x), M(y, \cdot)) = \sum_{b=1}^{B} W(y, b) L \left(f_b(x), M(y, b) \right).$$

In this chapter we are interested in weighting base classifiers of the scoring type. This leads us to focus on weighted loss-based decoding.

Weighted decoding algorithms differ in the way they determine the weight matrix W. Escalera et al. (2008) utilize the following observation to construct W. Binary classifiers may exhibit rather different generalization performances since each emerging binary problem has different characteristics. In addition to that, the accuracy of binary classifiers varies from class to class, because they are originally trained to separate positive and negative super classes, i.e., Y_b^+ and Y_b^- for $b \in \{1, \ldots, B\}$. As a result, Escalera et al. (2008) propose that $W(y, b)$ reflects the true positive rate of the classifier f_b on class y. To this end, the authors first define B binary classification problems using a coding matrix $M \in \{0, 1\}^{K \times B}$, and they then learn B binary classifiers f_b. The true positive rates of these binary classifiers over the classes in a validation set are then calculated as:

$$H(y, b) = \frac{1}{\mid D_y \mid} \sum_{x \in D_y} [\![f_b(x) = M(y, b)]\!], \tag{4.1}$$

where D_y denotes the subset of the validation set consisting of all the validation instances with class y. Having calculated $H(y, b)$ for all classes and for all binary problems, the weight matrix W is computed by normalizing H such that each row sums to 1. $W(y, b)$ therefore becomes:

$$W(y, b) = \frac{H(y, b)}{\sum_{b=1}^{B} H(y, b)}. \tag{4.2}$$

The rationale behind this normalization is to give an equal importance to each class, so as to avoid that the multiclass classifier becomes biased towards any class. Different frequencies for the classes in the data set may otherwise easily introduce such biases. In this chapter, we adopt the same approach. For brevity, we shall refer to this 'performance weighted' decoding algorithm as Perf.Weighted.

Differently, in Smith and Windeatt (2010) it is advocated that $W(y, b)$ should capture the information on how well a binary classifier f_b separates the members of class y from the instances of the other classes. To reflect this, they propose to choose $W(y, b)$ as follows. First an unnormalized weight is defined as:

$$H(y, b) = \max \left\{ 0, \left[\sum_{\substack{x \in D_y \\ x' \notin D_y}} C_b(x) C_b(x') - \sum_{\substack{x \in D_y \\ x' \notin D_y}} \overline{C_b}(x) \overline{C_b}(x') \right] \right\}, \tag{4.3}$$

where C_b is the correction function for the classifier f_b which takes the value of 1 if f_b correctly classifies the instance, and 0 otherwise. $\overline{C_b}$ is the complement of C_b which is given by $\overline{C_b}(x) = 1 - C_b(x)$. Next, the weight $W(y, b)$ is again obtained by normalizing each row sum to 1, as in (4.2). For brevity, we shall refer to this 'separability weighted' decoding algorithm as Sep.Weighted. We note that both algorithms to estimate a weight matrix W are essentially heuristic in nature; they both lack an underlying theoretical principle of optimization.

We also note that the weighted decoding algorithms explained above were initially proposed for the case where the binary classifiers are of discrete type. However, such algorithms can easily be extended for use with binary classifiers of scoring type in the following way: for all $b \in \{1, \ldots, B\}$ the scoring classifier is first turned into a discrete one, by choosing the class with the highest score as its binary class estimate (using some tie breaker in case of equal scores). With the resulting discrete classifier, the given algorithms become directly applicable.

Different from previous work in the literature, in this chapter we motivate the need for weighting binary classifiers from two novel perspectives. First we shall present weighting decoding as a *metric learning task* that is encountered in distance-based classification algorithms, such as nearest neighbor classification (Kulis, 2012). To this end, we learn a *task specific distance function*, where the task is to make the instance code word $f(x)$ be close to its true code word in the code space $\mathcal{C}$, while leaving it far away from the other code words in $\mathcal{C}$. Therefore we actually aim at increasing the margin of the multiclass classifier using a class decomposition scheme, which ultimately should lead to an increase in the classifier's accuracy.

Second, instead of weighting binary classifiers, we propose to weight the binary classification problems themselves by estimating how difficult it is to solve them, irrespective of the binary classification algorithm used. This gives rise to *classifier independent weighted decoding*, which is a flexible approach and can be employed with any classification algorithm. In doing so, we view the binary classification problems as bipartitions of a weighted graph where the nodes are the classes and the weighted edges indicate how easy it is to discriminate the classes that they connect. In the following sections, we first introduce the FP.Weighted decoding algorithm and then the BGP.Weighted decoding algorithm, which embody these two weighted decoding ideas.

4.2 Fractional Programming Weighted Decoding

Recall that in Chapter 3 we emphasized the resemblance between the underlying ideas in nearest neighbor (NN) classification and in the decoding phase of class decomposition. In the decoding phase the instance code word is assigned to the nearest class code word in the coding space $\mathcal{C}$. From this point of view, we argue that the idea of weighted decoding can be tied to that of weighted nearest neighbor (Paredes and Vidal, 2000; Marchiori, 2013). Following this, we further argue that the algorithms originally devised for weighted nearest neighbor can be adapted to weighted decoding.

In their weighted nearest neighbor work, Paredes and Vidal (2000) proposed a class-dependent weighted distance measure. The authors claim that the NN accuracy improves, if the distance measure employed by the NN classifier: (1) yields small values for the distances between instances coming from the same class, and (2) yields high values for the distances between instances from different classes. With this in mind, the objective function that they aim to minimize with respect to the weight matrix W, is chosen as:

$$J(W) = \frac{\sum_{(x,y)\in D} \text{dist}_{WE}(x, x^{=}_{nn})}{\sum_{(x,y)\in D} \text{dist}_{WE}(x, x^{\neq}_{nn})}, \tag{4.4}$$

where D is the data set, $x^{=}_{nn}$ is the nearest neighbor of x in its own class and $x^{\neq}_{nn}$ is again a nearest neighbor of x but in a different class. Here dist_{WE} stands for the weighted Euclidean distance. This distance dist_{WE} between any instance x' and an instance x that is known to belong to class $y \in Y$, is given as follows:

$$\text{dist}_{WE}(x', x) = \sqrt{\sum_{j=1}^{R} W(y,j)(x'(j) - x(j))^2} \tag{4.5}$$

where R denotes the dimension of x' and x (i.e., the number of features), and W is the weight matrix of size $K \times B$.

Following (Paredes and Vidal, 2000), we adapt the objective function in (4.4) to the weighted decoding problem. Our objective function then becomes:

$$J(W) = \frac{\sum_{(x_i,y_i)\in D} \text{dist}_{WL}(f(x_i), M(y_i, \cdot))}{\sum_{(x_i,y_i)\in D} \text{dist}_{WL}(f(x_i), M(y^{\neq}_i, \cdot))}, \tag{4.6}$$

with $W \in [0,1]^{K\times B}$ subject to the constraints

$$\sum_{b=1}^{B} W(y,b) = 1, \ \forall y \in Y.$$

Here $y_i \in Y$ is the class of the instance x_i and $y^{\neq}_i \in Y \setminus \{y_i\}$ is the *most confusing class* for x_i. By the term most confusing class, we mean that it is not the true class of x_i, but its associated code word is nearest to $f(x_i)$ in the code space $\mathcal{C}$ considering the other class code words in $\mathcal{C}$ (not using any weighting and employing a tie breaker rule if necessary). As a result, we see that the most confusing class $y^{\neq}$ for an instance x is determined by the binary classifiers.

We note that minimizing (4.6) can also loosely be interpreted as maximizing the margins of the binary classifiers for the instances using a class decomposition scheme. In fact, (Sun et al., 2005) defined the margin of a multiclass classifier for a labeled instance (x_i, y_i) using a class decomposition scheme as follows:

$$\rho(x_i) = \text{dist}_{WL}(f(x_i), M(y^{\neq}_i, \cdot)) - \text{dist}_{WL}(f(x_i), M(y_i, \cdot)). \tag{4.7}$$

A large value of $\rho(x_i)$ is obtained for a large first term and a small second term on the right hand side. The same properties will help to make $J(W)$ in (4.6) small. Thus we somehow also aim at maximizing our multiclass classifier margins by weighting the binary classifiers. This in turn is expected to result in higher classification accuracy (Sun et al., 2005).

The objective function defined in (4.6) is of *fractional programming* type (Sniedovich, 2011). To make the present chapter self-contained, we give a brief description of fractional programming in the following subsection.

4.2.1 Fractional Programming

The problems in fractional programming involve optimizing the ratio of two real-valued functions defined on a feasible set. More formally, such problems are given by (Sniedovich, 2011):

$$\min_{z\in Z}\{r(z)\}, \qquad \text{where } r(z) = \frac{s(z)}{t(z)}, \tag{4.8}$$

where s and t are real-valued functions on a feasible set Z (which the set of all points that satisfy the constraints). Here t is a positive function: $t(z) > 0, \forall z \in Z.$

If the functions s and t that form a fractional programming problem both are linear functions, then the problem is referred to as linear fractional programming. Otherwise, it is called nonlinear fractional programming (Sniedovich, 2011). Clearly, nonlinear fractional programming is a generalization of linear fractional programming and it is of interest to us throughout this chapter.

Among the algorithms aiming to tackle (4.8), the most common is the parametric algorithm which reduces (4.8) to the following parametric problem:

$$\min_{z\in Z}\{r_\lambda(z)\}, \text{where } r_\lambda(z) = s(z) - \lambda t(z), \lambda \in \mathbb{R}. \tag{4.9}$$

Assuming (4.9) has at least one optimal solution for each $\lambda \in \mathbb{R}$, it is known that there exists a value for λ such that every optimal solution to (4.9) is also an optimal solution for (4.8) and vice versa (Sniedovich, 2011, Appendix B). We now provide Dinkelbach's algorithm, which handles such a fractional programming problem by solving a sequence of parametric problems as in Eq. (4.9).

Algorithm 1 Dinkelbach's Algorithm

```
Input:   Real-valued functions: $s$ and $t$ defined on a set $Z$, where $t(z) > 0, \forall z \in Z$.
         A problem of fractional programming type: $\min_{z \in Z}\{\frac{s(z)}{t(z)}\}$.
         Convergence tolerance $\epsilon_{conv} > 0$.
Output:  Optimal $z \in Z$ which minimizes the given fractional programming problem.
1: Choose $z^{(1)} \in Z, l := 1, \lambda^{(1)} := s(z^{(1)})/t(z^{(1)})$,
   $\lambda^{(0)} := \lambda^{(1)} + 2\epsilon_{conv}$.
2: while $\lambda^{(l-1)} - \lambda^{(l)} > \epsilon_{conv}$ do
3:     Find an optimal solution $z^{(l+1)} \in Z$ of min$\{s(z) - \lambda^{(l)} t(z)\}$.
4:     Let $\lambda^{(l+1)} = s(z^{(l+1)})/t(z^{(l+1)})$.
5:     $l =: l + 1$.
6: end while
7: Output $z = z^{(l)}$.
```

While executing Dinkelbach's Algorithm, several auxiliary problems are generated and need to be solved, see line 3 of the algorithm. It is proved that, irrespective of the type of the functions s and t (linear or nonlinear), as the algorithm iterates, the sequence of solutions of the generated auxiliary problems monotonically converges to an optimal solution of the given fractional programming problem (Sniedovich, 2011). (Moreover, if Z consists of a finite number of points, then the algorithm is guaranteed to terminate and find an optimum for arbitrarily small $\epsilon_{conv} > 0$.)

4.2.2 The FP.Weighted Decoding Algorithm

We now detail our proposed weighted decoding algorithm, FP.Weighted decoding.

First note that the objective function in Eq. (4.6), which is to be minimized, gives rise to a fractional programming problem of the type (4.8). This problem can be written as:

$$\min_{W \in \mathcal{W}} \{r(W)\}, \text{where } r(W) = \frac{s(W)}{t(W)} = \frac{\sum_{x_i \in D} dist_{WL}(f(x_i), M(y_i, \cdot))}{\sum_{x_i \in D} dist_{WL}(f(x_i), M(y_i^{\neq}, \cdot))}. \quad (4.10)$$

Here $\mathcal{W}$ is the feasible set, i.e.: the set of weight matrices that satisfy the constraints $\sum_{b=1}^{B} W(y, b) = 1$, $\forall y \in Y$ and $W(y, b) \in [0, 1]$, $\forall y \in Y, \forall b \in B$.

When solving Eq. (4.10) with Dinkelbach's Algorithm, the auxiliary problems we encounter are of the following type:

$$\min_{W \in \mathcal{W}} \left\{ s(W) - \lambda^{(l)} t(W) \right\}, \quad (4.11)$$

where $\lambda^{(l)} \in \mathbb{R}$ is the (fixed) parameter at the iteration step l.

In this current form, the solution to (4.11), say W^*, can be quite sparse. In other words, only very few binary classifiers are often associated with non-zero weights per class. Therefore, outputs of the vast majority of the binary classifiers can usually be ignored, which in fact can lead to a shrinkage of the coding matrix, possibly causing the error-correcting property of ECOC to disappear. In order to avoid such extreme weights, we therefore employed negative entropy as a regularization term. Here, the use of entropy as a regularization term was motivated by the fact that entropy measures how uniform the weights are. In fact, a lower negative entropy tends to make the weights less extreme (Calafiore and Ghaoui, 2014). As a result the auxiliary problems then take the following form

$$\min_{W \in \mathcal{W}} \left\{ s(W) - \lambda^{(l)} t(W) + h \sum_{y \in Y} \sum_{b=1}^{B} W(y,b) \log W(y,b) \right\}. \tag{4.12}$$

The parameter $h \geq 0$ governs the relative importance of the regularization term compared with the original subproblem. The larger this coefficient is, the more equal the weights become, as the negative entropy function gets its minimum value for equal weights. We also emphasize that adding a regularization term to the auxiliary problems in Dinkelbach's Algorithm does not disturb the convergence of the solutions, provided that the added term is convex (Gugat, 1998). In our case, the added regularization term is convex. In fact, each variable $W(y,b)$ in this sum only appears in a single expression $W(y,b)\log W(y,b)$. As a consequence, to show that this regularization term is convex it is sufficient to check convexity of the function $p \log p$ for $p = W(y,b) \in [0,1]$. Clearly, its second derivative equals

$$\frac{d^2}{dp^2} p \log p = \frac{1}{p},$$

which is indeed nonnegative, proving convexity.

For a given value of $\lambda^{(l)}$, the constrained optimization problem (4.12) can be rewritten in the following form

$$\min_{W \in \mathcal{W}} \left\{ s(W) - \lambda^{(l)} t(W) + h \sum_{y \in Y} \sum_{b=1}^{B} W(y,b) \log W(y,b) \right\}. \tag{4.13}$$

The term $s(W)$ in Eq. (4.13) can be expressed as

$$s(W) = \sum_{(x_i,y_i) \in D} \mathrm{dist}_{WL}(f(x_i), M(y_i,\cdot)) = \sum_{y \in Y} \sum_{x_i \in D_y} \mathrm{dist}_{WL}(f(x_i), M(y_i,\cdot)).$$

The term $t(W)$ can likewise be expressed as

$$t(W) = \sum_{(x_i,y_i)\in D} \text{dist}_{WL}(f(x_i), M(y_i^{\neq}, \cdot)) = \sum_{y\in Y}\sum_{x_i\in D_y} \text{dist}_{WL}(f(x_i), M(y_i^{\neq}, \cdot)).$$

As a result we can write the expression (4.13) as:

$$\min_{W\in\mathcal{W}} \Bigg\{ \sum_{y\in Y} \Bigg(\sum_{x_i\in D_y} \big(\text{dist}_{WL}(f(x_i), M(y_i, \cdot)) - \lambda^{(l)}\text{dist}_{WL}(f(x_i), M(y_i^{\neq}, \cdot))\big) + \qquad (4.14)$$
$$+ h\sum_{b=1}^{B} W(y,b)\log(W(y,b))\Bigg)\Bigg\}.$$

Using the definition of a weighted loss-based function given at the beginning of this chapter, (4.14) then takes the form

$$\min_{W\in\mathcal{W}} \Bigg\{ \sum_{y\in Y} \Bigg(\sum_{x_i\in D_y} \big(\sum_{b=1}^{B} W(y,b)L(f_b(x_i), M(y_i,b)) - \lambda^{(l)}\sum_{b=1}^{B} W(y,b)L(f_b(x_i), M(y_i^{\neq},b))\big)$$
$$+ h\sum_{b=1}^{B} W(y,b)\log(W(y,b))\Bigg)\Bigg\}, \qquad (4.15)$$

where L is some loss function. Rearranging the summation for the second term in the summand, this optimization problem can be further expressed as

$$\min_{W\in\mathcal{W}} \Bigg\{ \sum_{y\in Y} \Bigg(\sum_{b=1}^{B} W(y,b)\big(\sum_{x_i\in D_y} L(f_b(x_i), M(y_i,b)) - \lambda^{(l)} \sum_{\{x_i\notin D_y \mid y_i^{\neq}=y\}} L(f_b(x_i), M(y_i,b))\big)$$
$$+ hW(y,b)\log(W(y,b))\Bigg)\Bigg\}. \qquad (4.16)$$

To highlight the structure of this optimization problem, it now is useful to define a function $F(y,b)$ as follows:

$$F(y,b) = \sum_{x_i\in D_y} L(f_b(x_i), M(y_i,b)) - \lambda^{(l)} \sum_{\{x_i\notin D_y \mid y_i^{\neq}=y\}} L(f_b(x_i), M(y_i,b)). \qquad (4.17)$$

The index of the second sum in (4.17) refers to the instances in the data set for which the most confusing class is y. Notice that $F(y,b)$ does not involve the weights in W. Regardless of the loss function chosen, $F(y,b)$ consists of two competing terms. The first term can be read as the total loss that the binary classifier f_b makes over the members of D_y, which reflects the heuristic minimization goal formulated in (Escalera et al., 2008). The second term quantifies how much the classifier f_b discriminates the members of D_y from the other classes, which reflects the heuristic maximization goal motivated in (Smith and Windeatt, 2010). As the first term is to be minimized and the second term to be maximized, they are combined into the single expression $F(y, b)$, to be minimized, by supplying the second term with a minus sign and the balancing parameter $\lambda^{(l)}$.

With this definition of $F(y, b)$, we arrive at the following form of the (weighted) auxiliary minimization problem:

$$\min_{W \in \mathcal{W}} \left\{ \sum_{y \in Y} \sum_{b=1}^{B} \Big(W(y, b)F(y, b) + hW(y, b)\log(W(y, b)) \Big) \right\}. \tag{4.18}$$

As a result, if $F(y, b)$ is large, which is to say that the output of the classifier f_b is closer to the b-th bit of the code word of the most confusing class other than that of the true class y, then minimization of the objective function promotes *punishing* the classifier f_b by reducing the corresponding entry $W(y, b)$. In case $F(y, b)$ is small, then minimization of the objective function promotes *rewarding* the classifier f_b by increasing $W(y, b)$. (Recall that the optimization space $\mathcal{W}$ is such that the weights for a fixed choice of y must sum to 1, inducing a relative trade off of the terms $F(y, b)$.) The regularization term with the parameter h acts to avoid that weights become zero too easily.

Interestingly, the resulting problem (4.18), is of the same form as the one encountered in (Domeniconi et al. (2007)), in which the authors are concerned with obtaining a weight vector per cluster so that the weighted distance between each instance and the center of its true cluster is minimal compared to those obtained from the remaining cluster centers. Naturally, the proposed solution to (4.18) can be obtained along similar lines as in (Domeniconi et al. (2007)). A detailed solution strategy for this problem (4.18) is as follows.

To solve (4.18), we first disregard the constraints $W(y, b) \in [0, 1],\ \forall y \in Y, \forall b \in B$; we will verify them later. For the moment, we only focus on the constraints $\sum_{b=1}^{B} W(y, b) = 1,\ \forall y \in Y$. We then set up the corresponding Lagrangian

$$\mathcal{L} = \sum_{y \in Y} \left\{ \sum_{b=1}^{B} \Big(W(y, b)F(y, b) + hW(y, b)\log(W(y, b)) \Big) + \lambda_y \big(1 - \sum_{b=1}^{B} W(y, b)\big) \right\}, \tag{4.19}$$

in which λ_y (for $y \in Y$) are the (unconstrained) Lagrange multipliers.

Setting the partial derivatives of $\mathcal{L}$ with respect to each of the weights $W(y, b)$ to zero, gives the following equations (for all $y \in Y$ and $b = 1, \ldots, B$):

$$F(y, b) + h \log W(y, b) + h - \lambda_y = 0, \tag{4.20}$$

together with the earlier constraints (for all $y \in Y$):

$$1 - \sum_{b=1}^{B} W(y, b) = 0. \tag{4.21}$$

Solving Eq. (4.20) with respect to $W(y, b)$ gives:

$$W(y, b) = \exp(-\frac{F(y, b)}{h} + \frac{\lambda_y}{h} - 1) = \exp(-\frac{F(y, b)}{h}) \exp(\frac{\lambda_y}{h} - 1)$$

$$= \frac{\exp(-F(y,b)/h)}{\exp(1-\lambda_y/h)}. \tag{4.22}$$

Plugging (4.22) into Eq. (4.21) yields:

$$1 - \sum_{b=1}^{B} \frac{\exp(-F(y,b)/h)}{\exp(1-\lambda_y/h)} = 0.$$

Solving this with respect to λ_y leads to:

$$\lambda_y = -h \log \sum_{b=1}^{B} \exp((-F(y,b)/h) - 1).$$

Finally, we substitute this into Eq. (4.22) to obtain the optimal weights $W^*(y,b)$, which are then found to be equal to:

$$\begin{aligned} W^*(y,b) &= \frac{\exp\left(-F(y,b)/h\right)}{\exp\left(1 + \log\left(\sum_{b=1}^{B} \exp\left(\left(-F(y,b)/h\right) - 1\right)\right)\right)} \\ &= \frac{\exp\left(-F(y,b)/h\right)}{\sum_{b=1}^{B} \exp\left(-F(y,b)/h\right)}. \end{aligned} \tag{4.23}$$

Note that the latter expression is easily verified to satisfy the constraints that the weights per class sum up to 1. What is also obvious from the exponential terms in the expression, is that these weights are all strictly positive. And as they all sum up to 1 per class, they individually are also strictly less than 1 (with $B > 1$). This verifies that the conditions we initially disregarded are automatically satisfied. It remains to verify that the stationary point of the Lagrangian we have computed, indeed provides a minimum for the optimization problem we set out to solve. For this, we note that the left-hand side expressions in (4.20) with $\lambda_y = 0$ give (unconstrained) partial derivatives of the criterion function to be minimized. If some $W(y,b)$ tends to zero then the corresponding partial derivative tends to $-\infty$, indicating maxima to occur at the boundary of the feasible set and hence a minimum to be contained in the interior of $\mathcal{W}$.

With the explicit solution (4.23) for the auxiliary minimization problem (4.12), Dinkelbach's Algorithm to optimize the objective function (4.6) can be made operational. (Recall that this optimization is approximate, as we have included a regularization term into the auxiliary problem (4.12).)

The resulting algorithm gives a weight estimation to be used in the FP.Weighted Decoding Algorithm, which is then readily set up. This weight estimation algorithm is explicitly given in pseudocode as Algorithm 2.

Note that $f(x)$ in line 5 of the "Weight estimation for the FP.Weighted Decoding Algorithm" is the *instance code word* of the instance $x \in D$. Indeed, after learning the binary classifiers in the first for-loop (line3), we let them run over the full training set. As a result, each training instance x is transformed into its instance code word $f(x) = (f_1(x), f_2(x), ..., f_B(x))$ in code space $\mathcal{C}$. In the rest of the algorithm only these instance code words are used, not the individual instances x. We thereby make use of the underlying idea of instance code words presented in the Chapter 3, while learning a weighted distance measure for the decoding phase.

Algorithm 2 Weight estimation for the FP.Weighted Decoding Algorithm

```
Input:   Training set $D$, coding matrix $M \in \{0,1\}^{K\times B}$,
         binary classification algorithm $\mathcal{A}$, regularization parameter $h$,
         loss function $L$, convergence tolerance $\epsilon_{conv}$.
Output:  Optimal weight matrix $W \in [0,1]^{K\times B}$ such that $\sum_{b=1}^{B} W(y,b) = 1,\ \forall y \in Y$.
1:  for $b := 1$ to $B$ do
2:      Relabel $D$ w.r.t. the super classes of the partition $M(\cdot, b)$.
3:      Learn a binary classifier $f_b$ using $\mathcal{A}$ on the relabeled $D$.
4:  end for
5:  For all instances $x$ in $D$, use the binary classifiers $f_b$ ($b = 1,\dots,B$) to form the
    instance code words: $f(x) = (f_1(x), ..., f_B(x))$.
6:  Define $s(W) = \sum_{x_i \in D} \mathrm{dist}_{WL}(f(x_i), M(y_i, \cdot))$,
    and $t(W) = \sum_{x_i \in D} \mathrm{dist}_{WL}(f(x_i), M(y_i^{\neq}, \cdot))$
    which make up a fractional programming problem $\min_{W \in \mathcal{W}} \left\{ \frac{s(W)}{t(W)} \right\}$.
7:  Choose $W^{(1)} \in \mathcal{W}$, $l := 1$, $\lambda^{(1)} := s(W^{(1)})/t(W^{(1)})$,
    $\lambda^{(0)} := \lambda^{(1)} + 2\epsilon_{conv}$.
8:  while $\lambda^{(l-1)} - \lambda^{(l)} > \epsilon_{conv}$ do
9:      for $y \in Y$ do
10:         for $b := 1$ to $B$ do
11:             Compute $F(y,b)$ as in Eq. (4.17).
12:             $W^{(l+1)}(y,b) := \frac{\exp(-F(y,b)/h)}{\sum_{b=1}^{B} \exp(-F(y,b)/h)}$.
13:         end for
14:     end for
15:     Let $\lambda^{(l+1)} = s(W^{(l+1)})/t(W^{(l+1)})$.
16:     $l := l + 1$
17: end while
18: Output $W = W^{(l)}$.
```

4.3 Bipartite Graph Partitioning Weighted Decoding

Weighted decoding algorithms in the literature, and also the FP.Weighted Decoding Algorithm just presented above, use weight matrices which depend on the results of the binary classifiers employed. As an alternative, we now present a *classifier independent* weighted decoding algorithm. To achieve this, we weight the binary classification problems themselves, by estimating how difficult it is to solve them. In particular, we explore the viewpoint that the closer the distance between a positive super class and a negative super class is, the more difficult it is to handle the corresponding binary classification problem. This then results in classifier independent weighting, in what is called the

BGP.Weighted Decoding Algorithm. The framework of bipartite graph partitioning offers a natural way to explain its key concepts and underlying ideas in more detail.

With a given multiclass classification problem, we can associate a weighted undirected complete graph $G = (Y, E, Q)$ for which the set of vertices Y corresponds to the classes in the data set. The set of edges E is defined to consist of all the possible pairs of vertices, and Q is a symmetric weight matrix of size $K \times K$ (where $K = |Y|$ denotes the number of classes). To qualify as a weight matrix, we require that the entries of Q on the main diagonal are all zero, while the off-diagonal entries of Q are all positive. We interpret each entry $Q(y, y')$ as the distance between the classes y and y' from Y. Therefore, the matrix Q is also called the *class distance matrix*. Each binary classification problem naturally corresponds to a binary partition of the class set Y and to a bipartition of the weighted graph G, and vice versa. This has also been also pointed out in Chapter 2, where it was explained that a class decomposition scheme $SP(Y)$ defines a set of binary class partitions of Y, i.e., $P_b(Y)$. As a result, each $P_b(Y)$ defines a cut: (Y_b^+, Y_b^-). Using the weighted graph G and the class distance matrix Q, the *cost* associated with a cut can then be defined as follows:

$$\text{cost}(Y_b^+, Y_b^-) = \sum_{y \in Y_b^+} \sum_{y' \in Y_b^-} Q(y, y'), \tag{4.24}$$

where Y_b^+ and Y_b^- are the negative and the positive super class of the binary class partition $P_b(Y)$, and $Q(y, y')$ is the distance between the classes y and y'. We argue that this cost can serve to quantify the difficulty of solving the b-th binary classification problem, as it expresses a degree of dissimilarity between the two subsets involved in the cut, i.e., in the partition of the weighted graph (Shi and Malik (2000)).

It is common to further normalize the cost of the cut, by dividing the unnormalized cost (4.24) by the number of edges crossing the cut, such as to make it invariant to the size of the super classes Y_b^+ and Y_b^-. Hence the normalized cost becomes

$$\text{Ncost}(Y_b^+, Y_b^-) = \frac{1}{\mid Y_b^+ \mid\mid Y_b^- \mid} \sum_{y \in Y_b^+} \sum_{y' \in Y_b^-} Q(y, y'), \tag{4.25}$$

where Ncost stands for the *normalized cost*. This gives rise to what is called Bipartite Graph Partitioning based weighting, in short BGP.Weighting, which can readily be employed for decoding.

For example, for a four-class classification problem with an exhaustive coding matrix M, the (undirected) weighted graph G of the classes and the corresponding value of Ncost for the first induced binary classification problem, is illustrated in Figure 4.1.

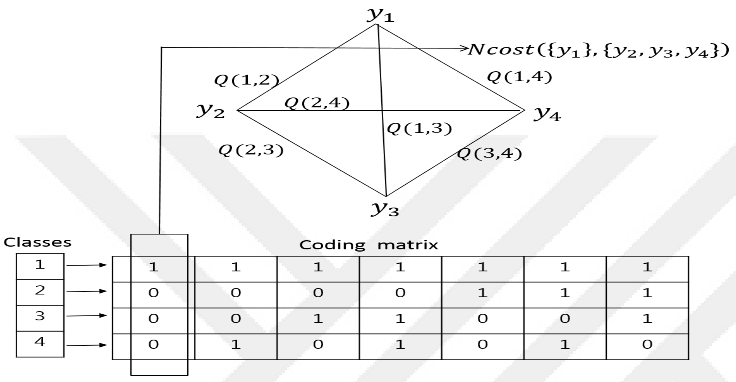


Figure 4.1: A weighted graph for a four-class classification problem, together with the normalized cost Ncost which corresponds to the first induced binary classification problem of the exhaustive coding matrix.

The normalized cost shown in Figure 4.1 is calculated as:

$$\text{Ncost}(\{y_1\},\{y_2,y_3,y_4\}) = \frac{1}{1\cdot 3}\sum_{i=2}^{4} Q(1,i).$$

The entries of the weight matrix W are assigned as:

$$W(y,b) = \text{Ncost}(Y_b^+, Y_b^-). \tag{4.26}$$

for all $y \in Y$ and $b = 1,\dots,B$ Algorithm 3. This weight estimation procedure for BGP.Weighted Decoding is summarized in 3. Once the weights have been computed, the instance classification is realized according to the decoding function of Definition 5 (see Chapter 2), using the weighted loss-based distance function from Definition 19.

Algorithm 3 Bipartite Graph Partitioning Weighted decoding algorithm

```
Input:  Training set $D$, class distance measure: $dist_{ic}$
Output: Weight matrix: $W \in [0,+\infty)^{K\times B}$
1: for $b \in \{1,...,B\}$ do
2:     $Ncost(Y_b^+,Y_b^-) := \frac{1}{|Y_b^+||Y_b^-|}\sum_{y\in Y_b^+}\sum_{y'\in Y_b^-} dist_{ic}(y,y')$;
3:     for $y \in Y$ do
4:         $W(y,b) := Ncost(Y_b^+,Y_b^-)$;
5:     end for
6: end for
7: Output $W$.
```

Here, the question that arises naturally is how to compute the distance between two classes: what are appropriate choices for an inter-class distance measure dist$_{ic}$ to build the class distance matrix Q through $Q(y, y') = \text{dist}_{ic}(y, y')$ for all $y, y' \in Y$. To answer this, one can employ distance measures from clustering, as explained in the following section.

4.3.1 Inter-Class Distance Measures

Inter-class distance measures can be readily borrowed from clustering, by simply viewing classes as clusters. We shall consider four particular inter-class distances: *single-link*, *complete-link*, *average-link*, and *centroid* distances; see (Tan et al., 2006). Pictorially, these four inter-class distances are shown in Figure 4.2. The single-link distance between two classes is the distance between the closest instances of the two classes. The complete-link distance between two classes is the distance between the instances that are farthest apart. The average-link distance between two classes is the average of all the distances between the instances of two classes, and finally the centroid distance between two classes is the distance between the two class centroids.

These four inter-class distances lead to four variants of the proposed BGP.Weighted Decoding algorithm. We refer to them as BGP.Sing, BGP.Comp, BGP.Aver and BGP.Cent, to reflect the single-link, the complete-link, the average-link and the centroid distances between classes, respectively.

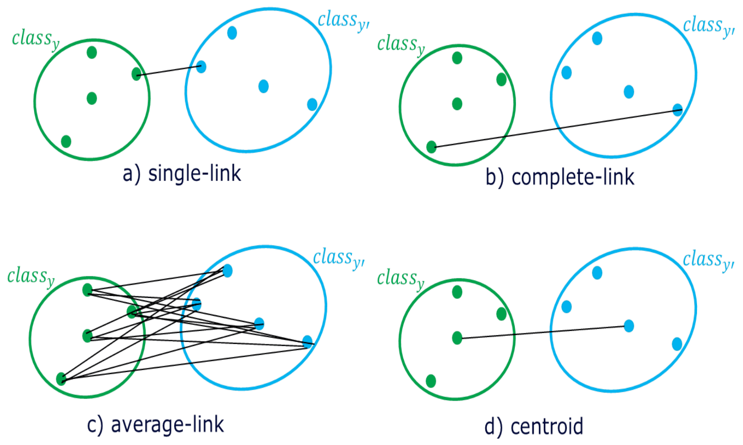


Figure 4.2: Inter-class distances.

4.4 Experiments

We tested the proposed weighted decoding algorithms, FP.Weighted Decoding and BGP.Weighted Decoding (with its four variants), on a number of UCI datasets which are summarized in Table 3.1 in Chapter 3. For these datasets, we compared these two novel algorithms against conventional (unweighted) decoding, and also against two state-of-the-art weighted decoding algorithms from the literature: Perf.Weighted (Escalera et al., 2008) and Sep.Weighted (Smith and Windeatt, 2010).

4.4.1 Settings

For each induced binary problem, we first learned a binary classifier f_b using the training set D. We then computed the weight matrix W for each of the weighted decoding algorithms using the training set D again. This setting enabled us to facilitate a fair comparison of the weighted decoding algorithms, as each considers the *same binary classifiers* and the *same training set* when computing a weight matrix for each dataset. Thus we promote that the differences in accuracy arise predominantly from using different weights in the decoding phase.

In our experiments, we opted for *exhaustive coding matrices* (eECOC) (Dietterich and Bakiri, 1995), for all the datasets with less than 10 classes, for all of the (weighted) decoding algorithms. For the datasets with $K = 10$ classes, we used a coding matrix M which consists of all balanced binary class partitions. That is: each column of M coincides with a binary class partition for which the sizes of the super classes are equal, i.e.: $|Y_b^+| = |Y_b^-|$, $\forall b \in \{1, \ldots, B\}$. In total, $B = \frac{K!}{2(\frac{K}{2}!)^2} = 126$ class partitions are obtained in this way, equaling the number of columns of the coding matrix M (Smirnov et al., 2009). The reason to choose such coding matrices is that they hold the property of having equidistant rows: the Hamming distance for any pair of rows of these coding matrices is the same. The FP.Weighted Decoding algorithm is thought to work best with a coding matrix with this property, because then the *most confusing class*, depends only on the accuracy of the binary classifiers. Otherwise, the (unbalanced) coding matrix being employed could introduce a bias when computing the most confusing classes. Finally, for the BGP.Weighted decoding, we employed the Euclidean distance on the feature space X when computing single-link, complete-link, average-link and centroid distances between classes.

As our binary classification algorithm in the experiments, we employed logistic regression. We used the square loss function in the weighted decoding (Flach, 2012). We note that using square loss is legitimate in the FP.Weighted Decoding as it always returns positive values. Therefore, the denominator in the objective function (4.6) cannot be negative, which is required by fractional programming. As for the inputs of FP.Weighted Decoding, we set the convergence tolerance ϵ_{conv} to 0.001 and tuned the regularization parameter h using internal cross validation in each experiment. For this purpose, we considered the set $\{0.01, 0.1, 1, 10, 100\}$ from which h takes its values. More precisely, for each value for h from the set, we performed 9-fold cross-validation (the tenth fold was set apart as the test set for the external cross validation), and we obtained the accuracy by averaging the accuracies over the (internal) folds. The regularization parameter h giving the highest averaged accuracy was then chosen to be used for the (external) cross-validation.

The method of evaluation is 10-fold cross validation. We note that for the purpose of maintaining a fair comparison among the weighting algorithms, in performing 10-fold cross validation we ensured that each weighting algorithm used the same folds for training and the same fold for testing in each run. For the Pendigits and the Optdigits datasets, however, where the training set and the test set are given separately, the method of evaluation was the hold-out method. Table 4.1 shows the results of the 10-fold cross validation experiments together with the hold-out experiments for the Pendigits and the Optdigits datasets. For the 10-fold cross validation experiments, the highlighted results indicate that the result is found statistically significantly better than that of the runner-up, according to the two-tailed paired t-test at significance level 0.05.

Table 4.1: Accuracies (in percentage) obtained using different weighted decoding algorithms when the base classifier is logistic regression. Statistically significant results are highlighted.

Dataset	Unweighted	FP.Weighted	Perf.Weighted	Sep.Weighted	BGP.Cent	BGP.Sing	BGP.Comp	BGP.Aver
Balance	86.66	87.83	91.32	91.44	87.16	86.95	86.88	86.88
Thyroid	95.8	95.8	95.8	95.8	95.8	95.8	95.8	95.8
Iris	96.66	98	97.33	97.33	97.33	97.33	96.66	97.33
Car	81.13	79.77	82.86	82.63	78.22	78.22	77.47	77.65
Vehicle	79.25	**81.02**	79.25	79.49	79.5	79.32	79.42	79.66
Glass	60.87	**63.24**	59.52	59.44	61.92	63.13	61.69	61.17
Dermatology	95.49	95.49	95.49	95.49	94.32	94.32	94.02	94.32
Segment	90.87	**93.05**	91.94	92.29	91.9	91.36	91.33	91.86
Ecoli	83.92	84.71	84.5	85.09	84.98	84.01	84.22	84.52
Zoo	91.28	**92.5**	92.03	92.13	91.13	91.43	91.39	91.32
Mfeat-mor	63.6	70.55	70.4	72.25	60.25	60.25	60.25	60.25
Pendigits	83.3	86.96	84.27	84.24	83.79	83.81	83.67	83.73
Optdigits	92.59	93.65	93.09	93.21	92.65	92.71	92.59	92.65

4.4.2 Evaluation of the Results

Table 4.1 reveals that the FP.Weighted decoding algorithm achieves the highest accuracy for seven UCI datasets (Iris, Vehicle, Glass, Pendigits, Segment, Zoo, and Optdigits) out of the 13 datasets considered. Of these, the accuracies obtained from Vehicle, Glass, Segment, and Zoo were found to be statistically significantly superior, based on the t-test at level 0.05. None of the four BGP.Weighted decoding variants, could achieve the highest accuracy for any of the datasets, though they outperformed the conventional unweighted decoding algorithm for nine datasets. However, the other weighted decoding algorithms, including FP.Weighted decoding, consistently outperformed the BGP.Weighted decoding variants.

Table 4.2: The average rank of the classifiers in Table 4.1.

Classifier	Unweighted	FP.Weighted	Perf.Weighted	Sep.Weighted	BGP.Cent	BGP.Sing	BGP.Comp	BGP.Aver
Av.rank	6.26	2	3.65	2.84	4.69	4.96	6.34	5.23

Taking the magnitude of the differences between the accuracies into account, we arrive at the following conclusions. Table 4.1 reveals that using FP.Weighted decoding in class decomposition, may improve the accuracy by up to 7%, if the binary classification algorithm is logistic regression and the square loss-based distance function is opted for in the decoding phase. On the other hand, when using BGP.Weighted decoding, the improvement in accuracy lies only in the range of 1% to 3%. When it comes to comparing FP.Weighted decoding and the two state-of-the-art decoding algorithms Perf.Weighted and Sep.Weighted, we found that FP.Weighted decoding increased the accuracy in the range of 1% to 3%.

In addition to comparing the proposed weighted decoding algorithms per dataset, we also compared them over all of the considered datasets. For this purpose we conducted the Friedman test and the Nemenyi test (Demsar, 2006). We first computed the average ranks of the weighting algorithms over the datasets. This is shown in Table 4.2. In order to show that the average ranks shown in

Table 4.2 are significantly different from the mean rank of 4.5, we conducted the Friedman test (Demsar, 2006). The Friedman test statistic F_F was found to be 7.93. This statistic is distributed according to the F-distribution with $(7, 84)$ degrees of freedom under the null hypothesis that the observed differences among the average ranks are not significant. The critical value of $F(7, 84)$ for $\alpha = 0.05$ is 2.12. Since the Friedman test statistic is larger than the critical value, we reject the null hypothesis. Therefore the measured differences between the average ranks of the weighted and unweighted decoding algorithms are found to be significant. This in turn implies that using different weighted algorithms in the decoding phase, leads to obtaining different accuracies (with FP.Weighted decoding showing the best performance).

Table 4.3 shows the absolute differences between the average ranks in Table 4.2. Based on the two-tailed Nemenyi test, the differences between the accuracies of two classifiers over multiple datasets is significantly different, if the difference between the corresponding average ranks exceeds the critical difference (Demsar, 2006). For 8 classifiers and 13 datasets, the critical difference is 2.91 for the significance level 0.05. In Table 4.3, the absolute differences that are larger than the critical difference of 2.91 are highlighted. Indeed, the difference between the average rank of FP.Weighted and that of unweighted decoding exceeds the critical difference, hence is highlighted. The same conclusion holds for Sep.Weighted decoding versus unweighted decoding. Also the performance of all of the variants of BGP.Weighted decoding but BGP.Cent is significantly worse than that of FP.Weighted decoding. On the other hand, the conducted Nemenyi test is not powerful enough to detect a significant difference between the other decoding algorithms.

Table 4.3: Average rank differences for Table 4.1 with statistically significant values highlighted.

Algorithm	Unweighted	FP.Weighted	Perf.Weighted	Sep.Weighted	BGP.Cent	BGP.Sing	BGP.Comp	BGP.Aver
Unweighted	0	**4.26**	2.61	**3.42**	1.57	1.3	0.8	1.03
FP.Weighted	**4.26**	0	1.65	0.84	2.69	**2.96**	**4.34**	**3.23**
Perf.Weighted	2.61	1.65	0	0.81	1.04	1.31	2.69	1.58
Sep.Weighted	**3.42**	0.84	0.81	0	1.85	2.12	**3.5**	2.39
BGP.Cent	1.57	2.69	1.04	1.85	0	0.27	1.65	0.54
BGP.Sing	1.3	**2.96**	1.31	2.12	0.27	0	1.38	0.27
BGP.Comp	0.08	**4.34**	2.69	**3.5**	1.65	1.38	0	1.11
BGP.Aver	1.03	**3.23**	1.58	2.39	0.54	0.27	1.11	0

4.5 Conclusion

In this chapter, we developed and presented two novel weighted decoding algorithms: the FP.Weighted Decoding Algorithm and the BGP.Weighted Decoding Algorithm. They aim to deal with the problem of difficult binary classification problems from the perspective of weighted decoding. Taking its root in weighted nearest neighbor algorithms, the FP.Weighted Decoding Algorithm employs a metric learning approach, where the goal is to learn a task-specific distance measure. Here, the task that FP.Weighted Decoding attempts to accomplish, is to reduce the distance between an instance code word and its true class code word, while at the same time increasing the distance between the instance code word and the other class code words. The optimization problem that arises in accomplishing this task, is of fractional programming type, and we handle it using regularization and Dinkelbach's Algorithm. In doing so, FP.Weighted Decoding increases the margin of the multiclass classifier, thus indeed leads to a significant increase in the accuracy obtained from the multiclass classifier. The performance of FP.Weighted maximizes when the used coding matrix has equidistant rows.

The BGP.Weighted Decoding Algorithm weights the binary classification problems associated with the class partitions induced by the coding matrix themselves, rather than weighting the binary classifiers f_b. This gives rise to a classifier independent weighting approach. For this propose, we cast the binary problems as bipartitions of the weighted graph of classes, where the weight on an edge indicates how difficult it is to discriminate the classes (vertices) that the edge connects. The cost of the bipartitions, which are computed by summing the weights of the crossed edges, becomes the weight of the corresponding binary problem.

Using 13 UCI datasets, we compared the FP.Weighted Decoding Algorithm and the BGP.Weighted Decoding Algorithm with the baseline (the conventional unweighted decoding) and with two state-of-the-art weighted decoding algorithms. The conducted tests demonstrated that FP.Weighted decoding achieved the best accuracy on 7 out of the 13 datasets. The BGP.Weighted decoding algorithm, on the other hand, was only found to be better than unweighted decoding for just two datasets. When it comes to compare FP.Weighted decoding and BGP.Weighted decoding with the two state-of-the-art weighted decoding algorithms, we found that the improvement in accuracy was relatively small and not statistically significant for the conducted experiments.

5 Conformal ECOC Machines

This chapter is based on the following publication:
Ismailoglu, F., Smirnov, E. and Peeters, R. (2015). Conformal ECOC Machines. In Proc of IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pages 361–368.

This chapter addresses the research question related to improving the computational performance of class decomposition schemes for reliable classification. The framework employed for reliable classification is the conformal framework (Vovk et al., 2005; Shafer and Vovk, 2008). It makes possible to compute confidence values for binary classifications to control the classification error. This is crucial for risk-sensitive applications such as drug discovery, medical diagnosis, or financial analysis (Papadopoulos, 2008).

Despite of the importance of the conformal framework, it was implemented for two class decomposition schemes only: the one-vs-all scheme and one-vs-one scheme (Vovk et al., 2005; Shi et al., 2013). While being valid, these implementations are computationally inefficient. Therefore, in this chapter we introduce two new implementations for conformal classification, namely mean-based conformal class decomposition machines (MCCD machines) and Poisson conformal class decomposition machines (PCCD machines). The MCCD machines and PCCD machines can employ any type of class decomposition schemes and are computationally efficient.

The chapter consists of 5 sections. The background information and related work are provided in Section 5.1. Section 5.2 introduces the conformal framework. The MCCD and PCCD machines are described in Section 5.3. Section 5.4 provides the experiments and Section 5.5 concludes the chapter.

5.1 Background

Conformal framework was proposed to provide confidence values to individual predictions in machine learning (Vovk et al., 2005; Shafer and Vovk, 2008). This is important when one needs to know to what extent s/he can rely on the classification estimated for any individual instance. Unlike discrete classifiers which only output most likely class $\hat{y}$ for a query instance x, the conformal framework allows for outputting a set Γ^ϵ of classes so that the true class y of x is a member of Γ^ϵ with a probability of at least $1 - \epsilon$ where ϵ is a significance level. The conformal framework can be applied for any type of classifiers (Vovk et al., 2005): nearest neighbors (Papadopoulos et al., 2011), SVM (Shi et al., 2013), neural networks (Papadopoulos, 2008), etc. In addition, the conformal framework can be used in the online mode and offline mode of training (Balasubramanian et al., 2014).

Due to the appealing properties of the conformal framework mentioned above, it has attracted an increasing attention and has been used extensively in various domains in combinations with various classification techniques. Ahlberg et al. (2015), for example, recently used conformal classifiers for the purpose of drug discovery where the risk analysis is of crucial importance. In medicine the conformal framework was employed to prove the validity of individual predictions of some medical conditions such as ovarian and breast cancers (Devetyarov et al., 2012) as well as acute abdominal pain (Papadopoulos et al., 2009). In (Balasubramanian et al., 2014) it was shown how to apply the conformal framework for time series analysis of network traffic flows. The reader is referred to (Balasubramanian et al., 2014) to acknowledge some other uses of the framework.

Despite the increasing interest in the conformal framework, its use within class decomposition schemes remains elusive. Vovk et al. (2005) allocated a brief section on combining conformal framework and class decomposition schemes. They proposed implementations for the one-vs-one scheme[1] and one-vs-all scheme only. While being valid, these implementations are computationally inefficient. Shi et al. (2013) improved the computational efficiency, however, with a price of less accurate conformal classification. The reader is referred to Section 5.3 where the work of (Vovk et al., 2005) and (Shi et al., 2013) is described in detail.

This chapter addresses the aforementioned problems of combining the conformal framework with class decomposition schemes. It introduces two implementations for conformal prediction based on class decomposition schemes that target the following objectives:

1. to generalize the implementation given by Vovk et al. (2005) for any class decomposition scheme, and
2. to reduce the computational complexity.

5.2 Conformal Framework

The conformal framework was proposed by Vovk et al. (2005) to provide confidence values to individual class predictions. The confidence values allow producing a class region (class set) $\Gamma^{\epsilon}(D, x_{N+1}) \subseteq Y$ for any query instance $x_{N+1} \in X$ and significance level $\epsilon \in [0, 1]$ such that Γ^{ϵ} contains the true class $y_{N+1} \in Y$ of x_{N+1} with a probability at least $1 - \epsilon$. In this way the classification error is bounded by ϵ in a long run; i.e. it can be controlled.

The conformal framework constructs the class region $\Gamma^{\epsilon}(D, x_{N+1}) \subseteq Y$ as follows. It iterates over the classes $y \in Y$ by considering each of them as an estimate of the true class of the query instance x_{N+1}. For any class y the framework first updates the data by adding the instance (x_{N+1}, y) to the data, i.e. $D' = D \cup \{(x_{N+1}, y)\}$. Then, it computes a nonconformity score α_n for any

[1]The one-vs-one scheme is an example of the ternary class decomposition schemes (Allwein et al., 2000). The ternary class decomposition schemes are not considered in this thesis and that is why in the rest of this chapter we do not treat the conformal implementation based the the one-vs-one scheme.

instance $(x_n, y_n) \in D'$ that indicates how unusual is the instance (x_n, y_n) from the instances in $D' \setminus \{(x_n, y_n)\}$. Finally, the framework computes the p-value for the class y equal to the proportion of the instances in the set D' which nonconformity scores α_n are greater than or equal to the nonconformity score of the instance (x_{N+1}, y). The class y is added to $\Gamma^{\epsilon}(D, x_{N+1}) \subseteq Y$ if its p-value is greater than the significance level ϵ.

To compute the nonconformity scores a *nonconformity function* has to be designed. The nonconformity function is of type $A : (X \times Y)^{(*)} \times (X \times Y) \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ [2]. Given data $D \in (X \times Y)^{(*)}$ and an instance $(x, y) \in (X \times Y)$, A yields a value $\alpha \in \mathbb{R}^+ \cup \{+\infty\}$ indicating how strange the instance (x, y) relative to those in D. The nonconformity function is usually related to the classifier employed (cf. Vovk et al. (2005)). The easiest choice in this case is the general nonconformity function given below.

Definition 20. (General Nonconformity Function) *General nonconformity function A is a function from $(X \times Y)^* \times (X \times Y)$ to $\mathbb{R}^+ \cup \{+\infty\}$ that maps a set $D \in (X \times Y)^*$ and an instance $(x, y) \in (X \times Y)$ to:*

$$\alpha = \sum_{y' \in Y \setminus \{y\}} s_{y'}, \tag{5.1}$$

where $s_{y'}$ is the score for class $y' \in Y$ produced by a scoring classifier f for x.

Armed with the definition of the general nonconformity function now we can describe the conformal prediction algorithm (CP). The input of the algorithm consists of a significance level $\epsilon \in [0, 1]$, a training data D, a nonconformity function A for a scoring classifier f, and a query instance $x_{N+1} \in X$. The output consists of a class region $\Gamma^{\epsilon}(D, x_{N+1}) \subseteq Y$ for x_{N+1}. To decide whether to include a class y in $\Gamma^{\epsilon}(D, x_{N+1})$, CP executes the following steps. First, it constructs training set D' by adding the labeled instance (x_{N+1}, y) to D. Then, in a leave-one-out manner it computes the nonconformity score α_n for each instance $(x_n, y_n) \in D'$. Finally, the algorithm sets the p-value p_y associated with class y for the instance x_{N+1}. The value p_y is computed as the proportion of the instances in the set D' with nonconformity scores α_n that are greater than or equal to the nonconformity score α_{N+1} of the instance (x_{N+1}, y). We note that this p-value is associated with the null hypothesis that the dataset D can be extended by the labeled instance (x_{N+1}, y) according to the unknown probability distribution p. Once the value p_y has been set, the conformal prediction algorithm includes the class y in $\Gamma^{\epsilon}(D, x_{N+1})$ if p_y is greater than the given significance level ϵ. The conformal prediction algorithm is given in Algorithm 4.

[2] $(X \times Y)^{(*)}$ denotes the set of all multi-sets defined over $X \times Y$.

Algorithm 4 Conformal Prediction Algorithm

```
Input:   Significance level $\epsilon$,
         training data $D$,
         query instance $x_{N+1}$,
         nonconformity function $A$ for scoring classifier $f$.
Output:  Class region $\Gamma^{\epsilon}(D, x_{N+1})$.

 1: Set class region $\Gamma^{\epsilon}(D, x_{N+1})$ equal to $\emptyset$.
 2: for each class $y \in Y$ do
 3:     Set $D'$ equal to $D \cup \{(x_{N+1}, y)\}$.
 4:     for $n := 1$ to $N+1$ do
 5:         Set nonconformity score $\alpha_n$ equal to $A(D' \backslash \{(x_n, y_n)\}, (x_n, y_n))$.
 6:     end for
 7:     Set $p$-value $p_y$ equal to $\frac{\#\{n=1,\ldots,N+1 | \alpha_i \geq \alpha_{N+1}\}}{N+1}$.
 8:     Include class $y$ in $\Gamma^{\epsilon}(D, x_{N+1})$ if $p_y > \epsilon$.
 9: end for
10: Output class region $\Gamma^{\epsilon}(D, x_{N+1})$.
```

The conformal prediction algorithm has to be valid and efficient. The algorithm is said to be valid if the class regions include the true class of the query instances with the probability of at least $1 - \epsilon$. The algorithm is said to be efficient if the class regions are nonempty and small.

5.3 Conformal Class Decomposition Machines

This section proposes two new implementations for conformal classification that can employ any type of class decomposition schemes, namely mean-based conformal class decomposition machines (MCCD machines) and Poisson conformal class decomposition machines (PCCD machines). For that purpose first a detailed description of relelated work is provided, the main problems are emphasized, and, then, the MCCD machines and PCCD machines are introduced.

5.3.1 Related Work

In the previous section we saw that the nonconformity function plays a central role in conformal classification. If we can find such a function for a class decomposition scheme, then in principle using the conformal prediction algorithm in Section 5.2 we can achieve conformal classification for this scheme. Below we briefly describe the related work in this context.

Vovk et al. (2005) proposed a nonconformity function for the one-against-all class decomposition scheme. This function aggregates the binary nonconformity scores; i.e. the scores outputted by the nonconformity functions related to the binary classifiers. We introduce the function as follows.

Assume that we have a coding matrix M of a class decomposition scheme $SP(Y)$ and a query instance instance x_{N+1}. Following the conformal prediction algorithm we hypothesize a class $y \in Y$ for x_{N+1} and form dataset D' equal to $D \cup \{(x_{N+1}, y)\}$. Since we use the coding matrix M, we form dataset D'_b for any $b \in \{1, \ldots, B\}$ equal to $\{(x_n, M(y_n, b))|(x_n, y_n) \in D'\}$. We compute the nonconformity score $\alpha_{n,b}^{M(y,b)}$ for any instance $(x_n, M(y_n, b)) \in D'_b$ [3]. If y is the class with index $b_1 \in \{1, \ldots, B\}$, then the aggregated nonconformity function outputs for instance (x_n, y_n) an nonconformity score α_n equal to:

$$\lambda \alpha_{n,b_1}^{M(y,b_1)} + (1-\lambda) \frac{\sum_{b \in \{1,\ldots,B\} \setminus \{b_1\}} \alpha_{n,b}^{M(y,b)}}{K-1} \tag{5.2}$$

where $\lambda \in [0, 1]$ is a user defined constant and K is the total number of classes.

It is stated in the book (Vovk et al. (2005)) that formula (5.2) is computed for all $y \in Y$. However, in doing so the nonconformity scores $\alpha_{n,b}^{M(y,b)}$ for different $y \in Y$ overlap (in fact they are mostly the same). Indeed, take two different classes y_1, y_2 from Y with indices $b_1, b_2 \in \{1, .., B\}$ respectively. For class y_1 the nonconformity scores needed to compute for Eq. (5.2) are $\alpha_{n,b_1}^{M(y,b_1)}$ and $\alpha_{n,b}^{M(y,b)} b \in \{1, ..., B\} \setminus \{b_1\}$. Similarly, for class y_2 the nonconformity scores needed to be computed are $\alpha_{n,b_2}^{M(y,b_2)}$ and $\alpha_{n,b}^{M(y,b)} b \in \{1, ..., B\} \setminus \{b_2\}$. Since $| \ \{\{1, ..., B\} \setminus \{b_1\}\} \cap \{\{1, ..., B\} \setminus \{b_2\}\} \ |$ is equal to $B-2$; $B-2$ nonconformity scores are the same.

The nonconformity score $\alpha_{n,b_1}^{M(y,b_1)}$ corresponds to the binary problem that discriminates the class y against all other classes; while each of the binary nonconformity scores $\alpha_{n,b_2}^{M(y,b_2)}$ corresponds to a binary problem that discriminates the class y together with $K-2$ other classes against the class with index b_2. The importance of these nonconformity scores is different for the labeled query instance (x_{N+1}, y). That is why, the aggregated nonconformity score α_n for this instance is λ-balanced.

If the aggregated nonconformity function for the one-against-all class decomposition scheme is employed in the conformal prediction algorithm from Algorithm 4, then we receive conformal classification for the one-against-all class decomposition scheme. The time complexity of the algorithm in this case is derived as follows. According to formula (5.2) to compute any aggregated nonconformity score α_n we need K number of calls of the binary nonconformity function A in line 5 of Algorithm 4. If this a general nonconformity function, then by Definition 20 to compute any α_n we need K of runs of the training algorithm of the binary classifiers f_b. The conformal prediction algorithm does this computation for all the instances in D' and all the classes $y \in Y$. Thus, the time complexity of the algorithm is $O(K^2NT)$, where T is the time complexity of the training algorithm of the binary classifiers f_b.

Shi et al. (2013) attempted to reduce the time complexity of the conformal prediction algorithm for the one-against-all class decomposition scheme. They proposed to employ a more computationally efficient but less accurate conformal prediction algorithm, namely the inductive conformal prediction algorithm (Papadopoulos, 2008). The inductive conformal prediction algorithm splits the data into training and calibration sets. A classifier is trained on the training set while the non-

[3]The superscript $M(y, b)$ in the nonconformity-score notation shows that class y is hypothesized for the query instance x_{N+1} given the coding matrix M.

conformity values are computed only for the instances from the calibration set using that classifier. Hence, the computational complexity is reduced with a price of an information loss when the p-values are being computed.

Analysing the approaches proposed in (Vovk et al., 2005) and (Shi et al., 2013), we observe that they are applicable for the one-against-all class decomposition scheme only due to the aggregated nonconformity function defined in (5.2). The approach in (Shi et al., 2013) is more computationally efficient than that from (Vovk et al., 2005), however, it is less accurate. In the next two subsections we show that the computational efficiency can be achieved without information loss. We introduce two solutions that overcome the aforementioned problems and comply with the main objectives stated in Section 5.1.

5.3.2 Mean-based Conformal Class Decomposition Machines

Consider the general case when we have a class decomposition scheme $SP(Y)$ given with a coding matrix M. We define a new aggregated nonconformity function that for any instance $(x_n, y_n) \in D'$ outputs:

$$\frac{1}{B}\sum_{b=1}^{B}\alpha_{n,b}^{M(y,b)}. \tag{5.3}$$

Equation Eq. (5.3) is similar to the equation Eq. (5.2) in terms of aggregating the nonconformity scores obtained from the binary classification problems. However, we propose to compute the nonconformity scores first for all classes in the class set. We then pool the resulting nonconformity scores. Having done that, when computing Eq. (5.3) for each class, we retrieve the related nonconformity scores from the pool. This trick enables us to avoid recomputing of the nonconformity scores unlike the case in Eq. (5.2).

The function outputs the average (mean) of the binary nonconformity scores $\alpha_{n,b}^{M(y,b)}$ related to the binary classifiers (analogously to the function defined in (5.2)). It can be weighted if any specific knowledge related to the class decomposition scheme used is available. The function is general: it can be applied for any coding matrix; i.e. any class decomposition scheme.

Given the new aggregated nonconformity function defined, we introduce the mean-based conformal class decomposition machines (MCCD machines). The MCCD machines are multiclass conformal classifiers that are able to employ any class decomposition schemes through the new nonconformity function. The key feature of the MCCD machines is that they compute all the binary nonconformity scores first and then proceed with the aggregated nonconformity scores following the class code words in the coding matrix M employed. In this way recomputing of the binary nonconformity scores is avoided which is the main reason for the complexity issues of the approach proposed by (Vovk et al., 2005) (described in the previous subsection).

The MCCD machines are given in Algorithm 5. The input of the MCCD machines consists of a significance level $\epsilon \in [0, 1]$, a training data D, a nonconformity function A for scoring binary classifiers f_b, a coding matrix $M \in \{0, 1\}^{K \times B}$, and a query instance $x_{N+1} \in X$. The output consists of a class region $\Gamma^\epsilon(D, x_{N+1}) \subseteq Y$ for x_{N+1}. From step 1 to step 9 the MCCD machines compute the binary nonconformity score $\alpha_{n,b}^{y_b}$ for any binary problem $b \in \{1, ..., B\}$, binary class

Algorithm 5 MCCD Machine

```
Input:  Significance level $\epsilon$,
        Training data $D$,
        Query instance $x_{N+1}$,
        Nonconformity function $A$ for scoring binary classifiers $f_b$,
        Coding matrix $M \in \{0,1\}^{K \times B}$.
Output: Class region $\Gamma^{\epsilon}(D, x_{N+1})$

1:  for each binary problem $b \in \{1, ..., B\}$ do
2:      Set $D_b$ equal to $\{(x_1, M(y_1, b)), ..., (x_n, M(y_n, b))\}$.
3:      for $y_b \in Y_b$ do
4:          Set $D'_b$ equal to $D_b \cup \{(x_{N+1}, y_b)\}$.
5:          for $n := 1$ to $N + 1$ do
6:              Set binary nonconformity score $\alpha^{y_b}_{n,b}$ equal to
                $A(D'_b \backslash \{(x_n, M(y_n, b))\}, (x_n, M(y_n, b)))$.
7:          end for
8:      end for
9:  end for
10: Set class region $\Gamma^{\epsilon}(D, x_{N+1})$ equal to $\emptyset$.
11: for each class $y \in Y$ do
12:     Set $D'$ equal to $D \cup \{(x_{N+1}, y)\}$.
13:     for $n := 1$ to $N + 1$ do
14:         Set aggregated nonconformity score $\alpha_n$ equal to $\frac{1}{B} \sum_{b=1}^{B} \alpha^{M(y,b)}_{n,b}$.
15:     end for
16:     Set $p$-value $p_y$ equal to $\frac{\#\{n=1,...,N+1 | \alpha_n \geq \alpha_{N+1}\}}{N+1}$.
17:     Include class $y$ in $\Gamma^{\epsilon}(D, x_{N+1})$ if $p_y > \epsilon$.
18: end for
19: Output class region $\Gamma^{\epsilon}(D, x_{N+1})$.
```

$y_b \in Y_b$, and instance $(x_n, y_n) \in D'$. From step 10 to step 19 the MCCD machines compute for any class $y \in Y$ the aggregated nonconformity scores α_n for all the instances $(x_n, y_n) \in D'$, and then calculate the p-value p_y of y when assigned to the query instance x_{N+1}. The class y is included in the class region $\Gamma^{\epsilon}(D, x_{N+1})$ of the query instance x_{N+1} if p_y is greater than the given significance level ϵ.

Assume that the binary nonconformity scores are computed using the general nonconformity function. Then, the time complexity of steps $1 \div 9$ equals $O(BNT)$ and the time complexity of steps $10 \div 19$ equals $O(BNK)$. Thus, the time complexity of the MCCD machines equals $O(BNT + BNK)$. The space complexity is $O(BN)$, since $B2N$ binary nonconformity scores have to stored.

If the MCCD machines are applied for the one-against-all class decomposition scheme, then the time complexity will be $O(KNT + K^2N)$. This means that the MCCD machines are more computationally efficient than the approach proposed by (Vovk et al., 2005) for that scheme. However, the MCCD machines require more space which is constant for the latter approach.

5.3.3 Poisson Conformal Class Decomposition Machines

As it is stated in Chapter 2 class decomposition schemes correct errors for discrete binary classifiers and for scoring binary classifiers differently. In this subsection we show how the error-correcting mechanism applicable for discrete binary classifiers can be adjusted for scoring binary classifiers. This is realized by a new aggregated nonconformity function based on the Poisson binomial distribution (Hong, 2013; Fernandez and Williams, 2010). The function is used in a new type of multi class conformal classifiers called Poisson conformal class decomposition machines (PCCD). Below we first introduce the new function and then the PCCD machines.

Assume that we have a class decomposition scheme $SP(Y)$ given with a coding matrix M and B probabilistic binary classifiers f_b. Given an instance $(x_n, y_n) \in D'$, each binary classifier f_b is first trained on $D'_b \setminus \{(x_n, M(y_n, b))\}$ and then outputs an estimated posterior probability $\hat{p}(1|x_n)$ that x_n belongs to the positive super class Y_b^+ [4]. If class $y \in Y$ is hypothesized for the query instance x_{N+1} and the general nonconformity function is employed (see Definition 20), then the binary nonconformity score $\alpha_{n,b}^{M(y,b)}$ for instance $(x_n, M(y_n, b))$ equals:

$$|M(y,b) - \hat{p}(1|x_n)|. \tag{5.4}$$

The binary nonconformity scores $\alpha_{n,b}^{M(y,b)}$ of an instance $(x_n, y_n) \in D'$ for all $b \in \{1, \ldots, B\}$ are estimates of the posterior probabilities of being opposite super class; i.e. they are estimations of the error probabilities. Since we aim at discreting error correcting, we are interested in probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ that number E of binary classifiers f_b that err for x_n is greater than the correction number CN of the class decomposition scheme $SP(Y)$, given error probability estimates $\alpha_{n,b}^{M(y,b)}$ for $b \in \{1, \ldots, B\}$. The probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ is equal to the following sum of probabilities:

$$\sum_{e=CN+1}^{B} p(E = e \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)}). \tag{5.5}$$

If we assume (naively) that the estimated error probabilities $\alpha_{n,b}^{y_b}$ are independent, it follows that the probability of $p(E = e \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ follows a *Poisson binomial distribution*[5]. This means that the probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ equals one minus the Poisson cumulative distribution function. It grows the more the instance (x_n, y_n) is nonconform with the instances in $D' \setminus \{(x_n, y_n)\}$ over the B binary classification problems defined by the class decomposition scheme employed. Thus, *we propose to use the probability* $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ *as an aggregated nonconformity score for any instance* $(x_n, y_n) \in D'$.

[4]We note that label "1" denotes the positive super class Y_b^+.

[5]The Poisson binomial distribution is the distribution of the sum of independent and non-identically distributed Bernoulli trials (Hong, 2013; Fernandez and Williams, 2010).

To compute the probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ we need to compute probabilities $p(E = e \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ (see equation (5.5)). The Poisson probability mass function $f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ equals:

$$\sum_{S \in \mathcal{F}_e} \prod_{b \in S} \alpha_{n,b}^{M(y,b)} \prod_{b \in S^c} (1 - \alpha_{n,b}^{M(y,b)}), \tag{5.6}$$

where $\mathcal{F}_e$ is the set of all subsets of e integers that can be selected from $\{1, \ldots, B\}$ and S^c is the complement of the set S to $\{1, \ldots, B\}$.

Computing the probability mass function $f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ is impractical, since the size of $\mathcal{F}_e$, i.e. the number of all possible subsets to be explicitly considered, is equal to $B!/(e! \times (B - e)!)$. Fortunately, there exists a recursive definition of the function given below:

$$\begin{aligned} f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)}) \quad = \quad & (1 - \alpha_{n,B}^{y_B}) f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B-1}^{M(y,B-1)}) + \\ & \alpha_{n,B}^{y_B} f(e-1; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B-1}^{M(y,B-1)}) \end{aligned} \tag{5.7}$$

under the assumption that $f(0; \emptyset)$ equals 1.

The left term and right term of equation (5.7) define recursive computations that intersect. That is why, we implemented equation (5.7) by storing all intermediate solutions (such as $f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B-1}^{M(y,B-1)})$ and $f(e-1; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B-1}^{M(y,B-1)})$). In this way we avoided re-computations which resulted in efficient implementation of the probability mass function $f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$.

To compute the probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ according to equation (5.5) all the solutions $f(e; \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ for e from $CN + 1$ to B have to be summed. Again, the recursive computations can intersect and that is why all intermediate solutions are stored to avoid re-computations. In this way computing the probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ has a time complexity of $O(B^2)$ at the worst case.

The final implementation of our Poisson aggregated nonconformity function follows the implementation described above. It outputs for any instance $(x_n, y_n) \in D'$ a nonconformity score equal to the probability $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$ and its time complexity is $O(B^2)$.

Given the Poisson aggregated nonconformity function defined, we introduce the Poisson conformal class decomposition machines (PCCD machines). The PCCD machines are multi class conformal classifiers that are able to employ any class decomposition schemes (through the new nonconformity function) and any type of probabilistic binary classifiers.

Algorithm 6 PCCD Machine

```
Input:   Significance level $\epsilon$,
         Training data $D$,
         Query instance $x_{N+1}$,
         Nonconformity function $A$ for probabilistic binary classifiers $f_b$,
         Coding matrix $M \in \{0,1\}^{K \times B}$.
Output:  Class region $\Gamma^{\epsilon}(D, x_{N+1})$

1:  for each binary problem $b \in \{1, \ldots, B\}$ do
2:      Set $D_b$ equal to $\{(x_1, M(y_1, b)), \ldots, (x_n, M(y_n, b))\}$.
3:      for $y_b \in Y_b$ do
4:          Set $D'_b$ equal to $D_b \cup \{(x_{N+1}, y_b)\}$.
5:          for $n := 1$ to $N + 1$ do
6:              Train a binary classifier $f_b$ on $D'_b \setminus \{(x_n, M(y_n, b))\}$.
7:              Set binary nonconformity score $\alpha_{n,b}^{y_b}$ equal to $| M(y_n, b) - Pr(1 \mid x_n) |$.
8:          end for
9:      end for
10: end for
11: Set class region $\Gamma^{\epsilon}(D, x_{N+1})$ equal to $\emptyset$.
12: for each class $y \in Y$ do
13:     Set $D'$ equal to $D \cup \{(x_{N+1}, y)\}$.
14:     for $n := 1$ to $N + 1$ do
15:         Set aggregated nonconformity score $\alpha_n$ equal to
              $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, \ldots, \alpha_{n,B}^{M(y,B)})$.
16:     end for
17:     Set $p$-value $p_y$ equal to $\frac{\#\{n=1,\ldots,N+1 | \alpha_n \geq \alpha_{N+1}\}}{N+1}$.
18:     Include class $y$ in $\Gamma^{\epsilon}(D, x_{N+1})$ if $p_y > \epsilon$.
19: end for
20: Output class region $\Gamma^{\epsilon}(D, x_{N+1})$.
```

The PCCD machines are given in Algorithm 6. The input of the PCCD machines consists of a significance level $\epsilon \in [0, 1]$, a training data D, a nonconformity function A for probabilistic binary classifiers f_b, a coding matrix $M \in \{0,1\}^{K \times B}$, and a query instance $x_{N+1} \in X$. The output consists of a class region $\Gamma^{\epsilon}(D, x_{N+1}) \subseteq Y$ for x_{N+1}. From step 1 to step 10 the PCCD machines compute the binary nonconformity score $\alpha_{n,b}^{y_b}$ (according to Eq. (5.4)) for any binary problem $b \in \{1, \ldots, B\}$, binary class $y_b \in Y_b$, and instance $(x_n, y_n) \in D'$. From step 11 to step 20 the PCCD machines compute for any class $y \in Y$ the Poisson aggregated nonconformity scores α_n for all the instances $(x_n, y_n) \in D'$, and then calculate the p-value p_y of y when assigned to the query instance x_{N+1}. The class y is included in the class region $\Gamma^{\epsilon}(D, x_{N+1})$ of the query instance

x_{N+1} if p_y is greater than the given significance level ϵ.

The complexity analysis of the PCCD machines is similar to that of the MCCD machines. The time complexity of steps $1 \div 10$ equals $O(BNT)$ and the time complexity of steps $11 \div 20$ equals $O(B^2NK)$. The first complexity coincides with that of the PCCD machines while the second one is different and requires an explanation. In steps $11 \div 20$ for all the K classes and all the $N + 1$ instances we compute the Poisson aggregated nonconformity value. This computation (step 15) has a complexity $O(B^2)$. This means the time complexity of steps $11 \div 20$ is indeed $O(B^2NK)$, and, thus, the time complexity of the MCCD machines is $O(BNT + B^2NK)$. The space complexity is $O(BN)$, since $B2N$ binary nonconformity scores have to stored.

If the PCCD machines are applied for the one-against-all class decomposition scheme, then the time complexity will be $O(KNT + K^3N)$. This means that the PCCD machines are more computationally efficient than the approach proposed by (Vovk et al., 2005) only if the time complexity T to train binary classifiers is higher that the time complexity to sum K^2 numbers. In addition, the PCCD machines require more space which is constant for the latter approach.

5.4 Experiments

This section provides experiments. It first introduces the evaluation metrics for validity and efficiency of conformal classifiers. Then, it provides experimental setup, and, finally, experimental results and discussion.

5.4.1 Evaluation Metrics

Given a significance level ϵ, a conformal classifier is said to be valid if the class regions include the true class of the query instances with the probability of at least $1 - \epsilon$ (Vovk et al., 2005). To establish validity we employ error rate e for class regions. It is defined as the proportion of the class regions that do not contain the correct classes. In this context, a conformal classifier is valid if for any $\epsilon \in [0, 1]$ we have that $e \leq \epsilon$. On error-significance graphs the validity can be established if the error curves are mostly under the diagonal line $y = x$.

Given a significance level ϵ, a conformal classifier is said to be efficient if the class regions it outputs are nonempty and as small as possible (Vovk et al., 2005). To establish efficiency we employ three metrics: the percentage p^e of empty-class regions S_e, the percentage p^s of single-class regions S_s, and the percentage p^m of multiclass regions S_m. All the regions have an associated error. The percentage p^e of empty-class regions is the error rate of S_e, since the true classes are not simply in empty-class regions. The error rate e^s on the set S_s is defined as the proportion of the invalid single-class regions in S_s. The error rate e^m on the set S_m is defined as the proportion of the invalid multiple-class regions in S_m. We note that these error components form the error e of a conformal algorithm; i.e. $e = p^e + p^s e^s + p^m e^m$.

Table 5.1: The UCI datasets used in the experiments.

Dataset	# Instance	# Attribute	# Class
Iris	150	4	3
Vehicle	846	18	4
Dermatology	366	34	6
Glass	214	9	7
Zoo	101	16	7
Mfeat	2000	6	10

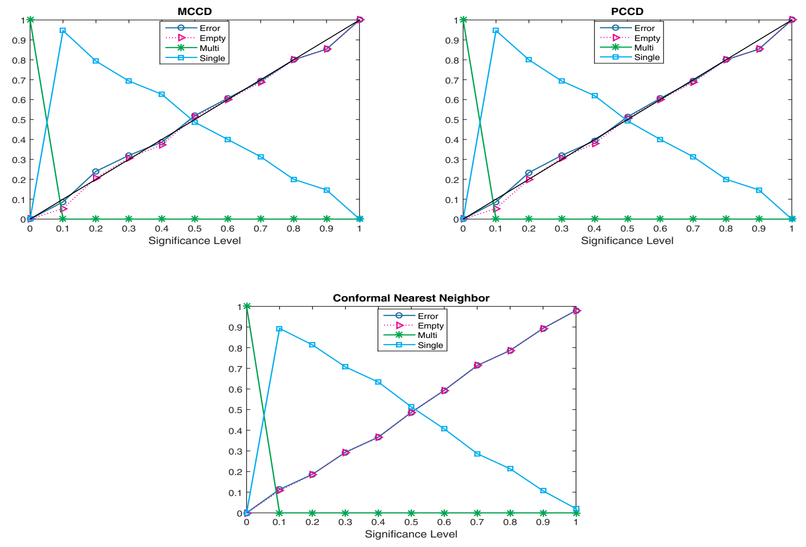


Figure 5.1: Iris

5.4.2 Experimental Setup

The experiments were performed with the MCCD machines, the PCCD machines, and the conformal nearest neighbor classifier (Proedrou et al., 2002). The latter is a standard conformal classifier and was used as a baseline classifier. The nonconformity scores for the three classifiers were computed using an internal 10-fold cross-validation process (instead of the original leave-one process to reduce computational complexity). The nonconformity function for the MCCD machines and PCCD machines was the general nonconformity function implemented over logistic regression classifiers with ridge parameter r set equal to 1.0. The nonconformity function for the conformal nearest neighbor classifier was the nearest neighbor function proposed in (Proedrou et al., 2002). The parameter k for nearest neighbors was set equal to 3.

The experiments with the three classifiers were performed on 6 datasets from UCI data repository (Bache and Lichman, 2013) summarized in Table 5.1. The method of evaluation was 10-fold cross validation.

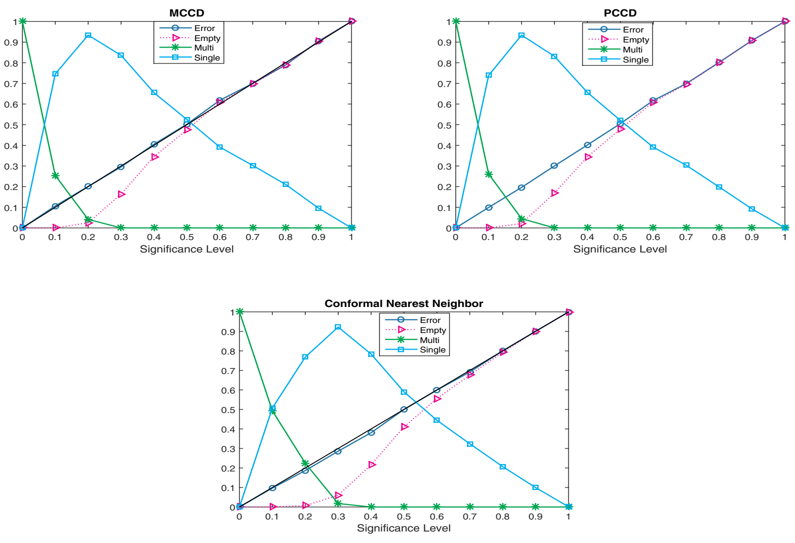


Figure 5.2: Vehicle

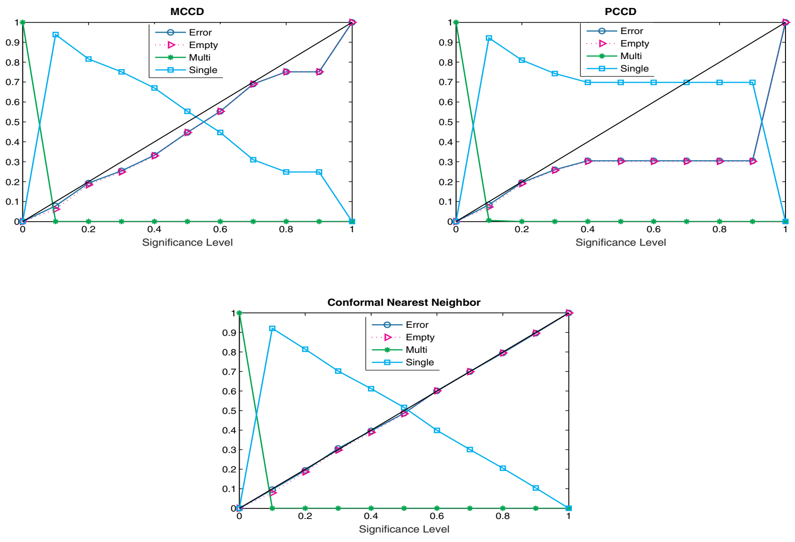


Figure 5.3: Dermatology

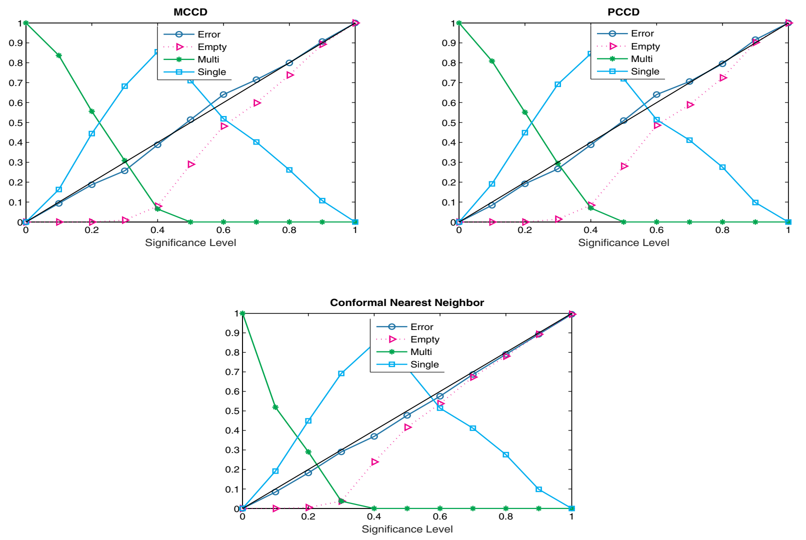


Figure 5.4: Glass

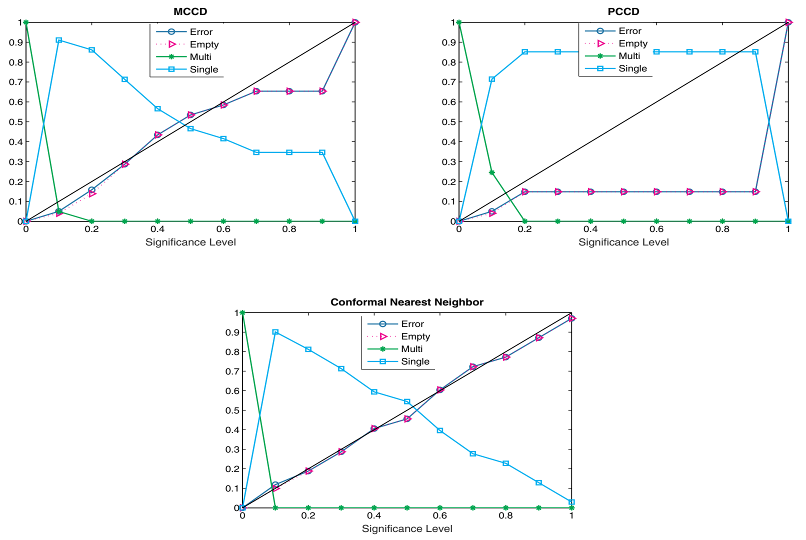


Figure 5.5: Zoo

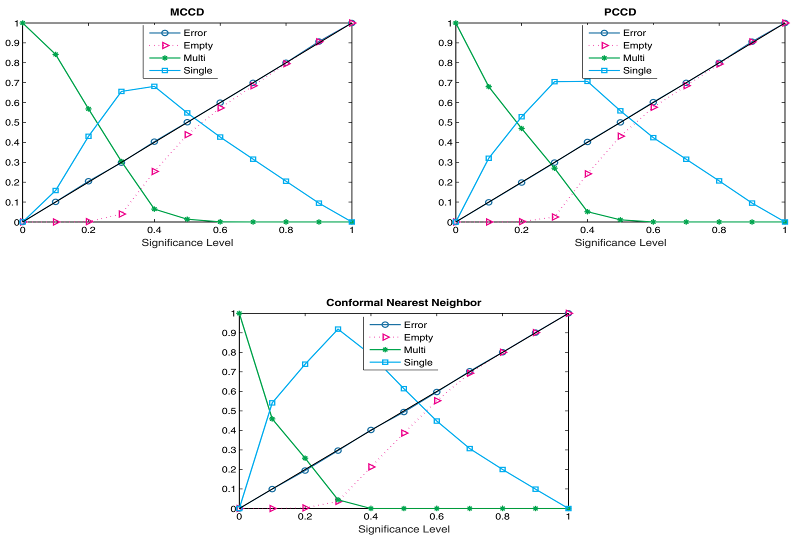


Figure 5.6: Mfeat

Table 5.2: The error rates of MCCD, PCCD and CNN

Data	MCCD PCCD CNN	$\epsilon = 0$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$	$\epsilon = 0.4$	$\epsilon = 0.5$	$\epsilon = 0.6$	$\epsilon = 0.7$	$\epsilon = 0.8$	$\epsilon = 0.9$	$\epsilon = 1$
Iris	MCCD	0	0.08	0.24	0.32	0.38	0.52	0.6	0.69	0.8	0.85	1
	PCCD	0	0.086	0.23	0.32	0.39	0.51	0.6	0.69	0.8	0.85	1
	CNN	0	0.116	0.18	0.29	0.36	0.48	0.59	0.71	0.78	0.89	0.98
Vehicle	MCCD	0	0.1	0.2	0.29	0.4	0.5	0.61	0.69	0.78	0.9	1
	PCCD	0	0.09	0.19	0.3	0.4	0.5	0.61	0.69	0.8	0.9	1
	CNN	0	0.09	0.18	0.28	0.38	0.5	0.59	0.69	0.8	0.9	0.99
Derm	MCCD	0	0.07	0.19	0.25	0.33	0.44	0.55	0.68	0.75	0.75	1
	PCCD	0	0.08	0.19	0.25	0.3	0.3	0.3	0.3	0.3	0.3	1
	CNN	0	0.09	0.19	0.3	0.39	0.48	0.6	0.69	0.79	0.89	1
Glass	MCCD	0	0.09	0.18	0.25	0.38	0.51	0.64	0.71	0.79	0.9	1
	PCCD	0	0.08	0.19	0.26	0.38	0.5	0.64	0.7	0.79	0.91	1
	CNN	0	0.08	0.18	0.28	0.36	0.47	0.57	0.68	0.78	0.89	0.99
Zoo	MCCD	0	0.04	0.15	0.28	0.43	0.53	0.58	0.65	0.65	0.65	1
	PCCD	0	0.04	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	1
	CNN	0	0.11	0.18	0.28	0.4	0.45	0.6	0.72	0.77	0.87	0.97
Mfeat	MCCD	0	0.1	0.2	0.29	0.4	0.5	0.59	0.69	0.79	0.9	1
	PCCD	0	0.09	0.19	0.29	0.4	0.5	0.6	0.69	0.79	0.9	1
	CNN	0	0.09	0.19	0.29	0.4	0.49	0.59	0.7	0.8	0.9	1

Table 5.3: The empty region rates of MCCD, PCCD and CNN

Data	MCCD PCCD CNN	$\epsilon = 0$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$	$\epsilon = 0.4$	$\epsilon = 0.5$	$\epsilon = 0.6$	$\epsilon = 0.7$	$\epsilon = 0.8$	$\epsilon = 0.9$	$\epsilon = 1$
Iris	MCCD	0	0.05	0.2	0.3	0.37	0.51	0.6	0.68	0.8	0.85	1
	PCCD	0	0.05	0.2	0.3	0.38	0.5	0.6	0.68	0.8	0.85	1
	CNN	0	0.1	0.18	0.29	0.36	0.48	0.59	0.71	0.78	0.89	0.98
Vehicle	MCCD	0	0	0.02	0.16	0.34	0.47	0.6	0.69	0.78	0.9	1
	PCCD	0	0	0.02	0.16	0.34	0.47	0.6	0.69	0.8	0.9	1
	CNN	0	0	0	0.06	0.21	0.41	0.55	0.67	0.79	0.9	0.99
Derm	MCCD	0	0.06	0.18	0.24	0.32	0.44	0.55	0.68	0.75	0.75	1
	PCCD	0	0.07	0.18	0.25	0.3	0.3	0.3	0.3	0.3	0.3	1
	CNN	0	0.07	0.18	0.29	0.38	0.48	0.6	0.69	0.79	0.89	1
Glass	MCCD	0	0	0	0	0.07	0.28	0.48	0.59	0.73	0.89	1
	PCCD	0	0	0	0.01	0.08	0.28	0.48	0.58	0.72	0.9	1
	CNN	0	0	0	0.03	0.23	0.41	0.53	0.67	0.78	0.89	0.99
Zoo	MCCD	0	0.04	0.13	0.28	0.43	0.53	0.58	0.65	0.65	0.65	1
	PCCD	0	0.04	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	1
	CNN	0	0.09	0.18	0.28	0.4	0.45	0.6	0.72	0.77	0.87	0.97
Mfeat	MCCD	0	0	0	0.03	0.25	0.43	0.57	0.68	0.79	0.9	1
	PCCD	0	0	0	0.02	0.24	0.43	0.57	0.68	0.79	0.9	1
	CNN	0	0	0	0.03	0.21	0.38	0.55	0.69	0.8	0.9	1

Table 5.4: The multi region rates of MCCD, PCCD and CNN

Data	MCCD PCCD CNN	$\epsilon = 0$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$	$\epsilon = 0.4$	$\epsilon = 0.5$	$\epsilon = 0.6$	$\epsilon = 0.7$	$\epsilon = 0.8$	$\epsilon = 0.9$	$\epsilon = 1$
Iris	MCCD	1	0	0	0	0	0	0	0	0	0	0
	PCCD	1	0	0	0	0	0	0	0	0	0	0
	CNN	1	0	0	0	0	0	0	0	0	0	0
Vehicle	MCCD	0	0	0.02	0.16	0.34	0.47	0.6	0.69	0.78	0.9	1
	PCCD	1	0.25	0.04	0	0	0	0	0	0	0	0
	CNN	1	0.49	0.22	0.01	0	0	0	0	0	0	0
Derm	MCCD	1	0	0	0	0	0	0	0	0	0	0
	PCCD	1	0	0	0	0	0	0	0	0	0	0
	CNN	1	0	0	0	0	0	0	0	0	0	0
Glass	MCCD	1	0.83	0.55	0.3	0.06	0	0	0	0	0	0
	PCCD	1	0.8	0.55	0.29	0.07	0	0	0	0	0	0
	CNN	1	0.51	0.28	0.03	0	0	0	0	0	0	0
Zoo	MCCD	1	0.04	0	0	0	0	0	0	0	0	0
	PCCD	1	0.24	0	0	0	0	0	0	0	0	0
	CNN	1	0	0	0	0	0	0	0	0	0	0
Mfeat	MCCD	1	0.84	0.56	0.3	0.06	0.01	0	0	0	0	0
	PCCD	1	0.67	0.46	0.27	0.05	0.01	0	0	0	0	0
	CNN	1	0.45	0.25	0.04	0	0	0	0	0	0	0

Table 5.5: The single region rates of MCCD, PCCD and CNN

Data	MCCD PCCD CNN	$\epsilon = 0$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$	$\epsilon = 0.4$	$\epsilon = 0.5$	$\epsilon = 0.6$	$\epsilon = 0.7$	$\epsilon = 0.8$	$\epsilon = 0.9$	$\epsilon = 1$
Iris	MCCD	0	0.94	0.79	0.69	0.62	0.48	0.4	0.31	0.2	0.14	0
	PCCD	0	0.94	0.8	0.69	0.62	0.49	0.4	0.31	0.2	0.14	0
	CNN	0	0.89	0.81	0.7	0.63	0.51	0.4	0.28	0.21	0.1	0,02
Vehicle	MCCD	0	0.74	0.93	0.83	0.65	0.52	0.39	0.3	0.21	0.09	0
	PCCD	0	0.73	0.93	0.83	0.65	0.52	0.39	0.3	0.19	0.09	0
	CNN	0	0.5	0.76	0.92	0.78	0.58	0.44	0.32	0.2	0.09	0
Derm	MCCD	0	0.93	0.81	0.75	0.67	0.55	0.44	0.31	0.24	0.24	0
	PCCD	0	0.92	0.81	0.74	0.69	0.69	0.69	0.69	0.69	0.69	0
	CNN	0	0.92	0.81	0.7	0.61	0.51	0.39	0.3	0.2	0.18	0
Glass	MCCD	0	0.16	0.44	0.68	0.85	0.71	0.51	0.4	0.26	0.1	0
	PCCD	0	0.19	0.44	0.69	0.84	0.71	0.51	0.41	0.27	0.09	0
	CNN	0	0.48	0.7	0.92	0.76	0.58	0.46	0.32	0.21	0.1	0
Zoo	MCCD	0	0.91	0.86	0.71	0.56	0.46	0.41	0.34	0.34	0.34	0
	PCCD	0	0.71	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0
	CNN	0	0.9	0.81	0.71	0.59	0.54	0.39	0.27	0.22	0.12	0.02
Mfeat	MCCD	0	0.15	0.43	0.65	0.68	0.54	0.42	0.31	0.2	0.09	0
	PCCD	0	0.32	0.52	0.7	0.7	0.55	0.42	0.31	0.2	0.09	0
	CNN	0	0.54	0.73	0.91	0.78	0.61	0.44	0.3	0.19	0.09	0

5.4.3 Results and Discussion

Figures 5.1 - 5.6 and Tables 5.2 - 5.4 show the experimental results. Analyzing the estimated error curves, we observe that they are mostly under the diagonal line; i.e. for any $\epsilon \in [0, 1]$ we have that $e \leq \epsilon$. This implies that the MCCD and PCCD machines are valid classifiers (on the experimental datasets together with the CNN classifier).

Analyzing the efficiency of the classifiers, we observed that:

(1) the empty-class region curves of the MCCD and PCCD machines are more shifted to the right compared with those of the CNN classifier; and

(2) the single-class region curves of the MCCD and PCCD classifiers are more shifted to the left compared with those of the CNN classifier.

Observations (1) and (2) imply that the MCCD and PCCD machines have smaller empty class regions and bigger single class regions compared with those of the CNN classifier for smaller significance levels. The only exception we found for the Mfeat data. Thus, we may conclude that the MCCD machines and PCCD machines are more efficient then the CNN classifier on the experimental datasets.

Comparing the MCCD machines and PCCD machines, we see that they have similar results. However, their performance on the Dermatology dataset and the Zoo dataset is rather different (see Figure 5.3 and Figure 5.5). The figures show that the PCCD machines (in contract with the MCCD machines) managed to convert a large percentage of empty class regions into single class regions that contain correct classes. This resulted in a significant decrease of the error e and a better efficiency. To explain this result we note that the average error correlation of the binary classifiers (within the PCCD machines) for the Dermatology dataset and the Zoo dataset is negative (see Figure 5.7). The negative average error correlation increases the average difference between any two binary nonconformity scores $\alpha_{n,b_1}^{M(y,b_1)}$ and $\alpha_{n,b_2}^{M(y,b_2)}$ for $b_1, b_2 \in \{1, ..., B\}$. This shifts the Poison distribution determined by the binary nonconformity scores $\alpha_{n,1}^{M(y,1)}, ..., \alpha_{n,B}^{M(y,B)}$ to the left; i.e. the distribution becomes more positively skewed. The latter decreases the probabilities $p(E > CN \mid \alpha_{n,1}^{M(y,1)}, ..., \alpha_{n,B}^{M(y,B)})$; i.e. the aggregated nonconformity scores for the true classes. This implies that the true classes receive higher p-values compared with those of other classes which results in a significant decrease of the error e and a significant increase of the percentage p^s of single-class regions.

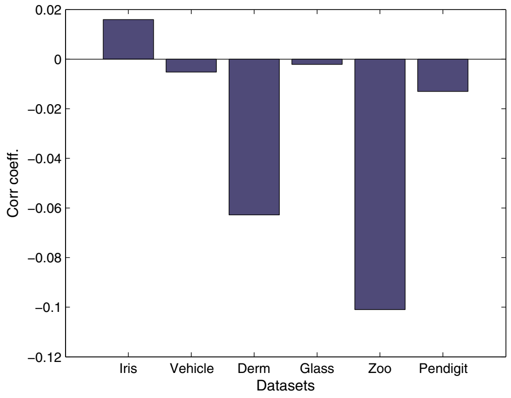


Figure 5.7: Averaged error correlation of the binary classifiers.

5.5 Conclusion

This chapter introduced the MCCD machines and PCCD machines as new class decomposition implementations of the conformal framework. The MCCD machines and PCCD machines employ aggregating nonconformity functions that do not depend on the type of the class decomposition scheme employed. Thus, they can be applied for any class decomposition schemes and can be viewed as generalizations of the conformal implementations for the One-Against-All scheme proposed in (Vovk et al., 2005).

We showed that the MCCD machines and PCCD machines employ a very different approach when computing the aggregated nonconformity scores. They first compute all possible binary nonconformity scores and then for any new instance and any class they compute the aggregated nonconformity score using pre-computed binary nonconformity scores. This contrasts with the conformal implementations for the One-Against-All scheme that always re-compute all the binary nonconformity scores whenever a new aggregated nonconformity score is required. Thus, the MCCD machines and PCCD machines are more computationally efficient than the conformal implementations for the One-Against-All scheme.

The MCCD machines and PCCD machines differ in the way that they correct errors. Both machines employ loss-based error-correcting mechanisms through their aggregating nonconformity functions. However, the mechanism of the PCCD machines is related to the error-correcting mechanism applicable for discrete binary classifiers. Its nonconformity function outputs an estimate of the probability that more than CN binary classifiers will err. It was demonstrated that, when the average error correlation between the binary classifiers is negative, this function allows for generating class regions with low error and high efficiency.

The experiments showed that the MCCD machines and PCCD machines are valid and efficient classifiers in the context of reliable classification. For most of the datasets, their efficiency is superior to that of the standard conformal classifier (CNN).

6 The Bunching.HDA Algorithm for Heterogeneous Domain Adaptation

This chapter addresses the research question of how to apply class decomposition schemes for heterogeneous domain adaptation. Assume that we have data from a domain of interest that we call *target domain* and data from an auxiliary domain that we call *source domain* under condition that the feature spaces of both domains are different. The classification problem, we have, is to provide a good estimate of the true class of a query instance from the target domain using a classification model that employs the source data in addition to the target data.

This chapter proposes the Bunching.HDA algorithm as a class decomposition solution for the considered classification problem when the target and source domains share the same output set of class labels. In this case the code space associated with any class decomposition scheme is the same for both domains and thus it can be considered as a common latent feature space where the target and source data can be projected. The algorithm is based on this finding and operates as follows. First, it builds the encoding functions (mappings) for the target data, source data, and class labels. Then it employs these functions to project all the data and class labels in the common code space. Once all the data and class labels are presented in the code space, two Bunching.HDA classification rules can be applied: the decoding rule of the class decomposition schemes and the decoding rule of the instance decomposition schemes.

The Bunching.HDA algorithm is an alternating minimization algorithm. Its main functions are theoretically derived and justified. The algorithm is experimentally analyzed: a convergence analysis is provided together with experiments on three domain adaptation problems. The latter shows that the Bunching.HDA algorithm is capable of outperforming some state-of-the-art heterogeneous domain-adaptation methods.

The chapter consists of 6 sections. Section 6.1 introduces a background information on domain adaptation. Related work is given in Section 6.2. Section 6.3 formalizes the classification problem for domain adaptation. The Bunching.HDA algorithm is introduced in Section 6.4. Section 6.5 describes the experiments. The chapter is concluded in Section 6.6.

6.1 Background

To obtain an accurate classification model in a particular domain of interest, one needs to feed the classification algorithm with a sufficient amount of labeled data (Wang and Mahadevan, 2011). Obtaining such amount of data is often a time consuming and an expensive process (e.g. human annotators are needed for labeling data) (Duan et al., 2012; Wu and Dietterich, 2004). However, in many situations there exist similar domains that are abundant in labeled data. Then, a natural question arises whether we can exploit the labeled data from these domains to derive better classification models compared with the models based on the initially given data. Attempts that seek for an answer to this question resulted in a field of *domain adaptation* (DA) usually considered as a branch of transfer learning (Pan and Yang, 2010).

There exist several DA methods, some of them have been successfully employed in various application fields. Examples include applications in sentiment analysis (Blitzer et al., 2007), computer vision (Duan et al., 2010), game playing (Banerjee and Stone, 2007), etc. Recently, DA methods have been used to improve the classification accuracy of a newly developed image recognition system by levering millions of annotated Instagram and Twitter photos (Gong et al., 2012).

In the DA field, the domain of interest is called *target domain* and the provided auxiliary domain is called *source domain*. So far, the DA research has mainly focused on the *homogeneous domain adaptation* (hDA). In hDA the target and source domains share the same feature space (Wu and Dietterich, 2004; Saenko et al., 2010; Hall, 2004). The observed difference between the domains arises due to the difference between their data distributions (Pan and Yang, 2010; Wang and Mahadevan, 2011). However, many real-world scenarios, like classifying images exploiting given tagged text, or classifying PET scans using MR images, require more advanced approaches to adapt domains with totally different feature spaces. This gives rise to a relatively new DA subfield called *heterogeneous domain adaptation* (HDA) which is the main topic of this chapter.

6.2 Related Work

The approaches to HDA are divided into two groups (Weiss et al., 2016). The first group consists of symmetric approaches. These approaches transform the target and source domains into a common latent feature space. The target data and source data are first projected into the common space and then the final prediction models are trained over the projected data. The second group consists of asymmetric approaches. These approaches transform the source domain directly into the target domain. That is, the source data is first projected into the target domain and then the final prediction models are trained on the target data and the projected source data. Below we briefly describe the main approaches from the two groups and specify the requirements for the HDA approach proposed in this chapter.

We consider three approaches to symmetric heterogeneous domain adaptation. The first approach is Domain Adaptation Manifold Alignment (DAMA) (Wang and Mahadevan, 2011). DAMA is applicable for problems when there are one target domain and $P-1$ source domains that have common in the same class-label set. Under the assumption that any domain can be viewed as a manifold, it creates P transformation functions, one for each domain. This is realized by a manifold alignment algorithm. The algorithm maintains a combinatorial graph Laplacian matrix to reflect inter-domain class similarity, a combinatorial graph Laplacian matrix to reflect inter-domain class dissimilarity, and a diagonal matrix of graph Laplacian matrices each reflecting instance similarities within a domain. The algorithm minimizes an objective function defined over the matrices so that the resulting P transformation functions match same class instances and that separate different class instances when projected into the common latent feature space. Consequently, the local topology is preserved as much as possible. DAMA has its own prediction algorithm. It first trains regression models for the source data projected, and then adapts these models to the target domain using manifold regularization based on the projected target data.

Heterogeneous Spectral Mapping (HeMAP) is a symmetric approach for problems with a target domain and a source domain that do not share common class-label set (Shi et al., 2010). It employs a spectral mapping algorithm to learn transformation functions. The defined objective function has two components: the first one measures whether the topology of the target data and source data is preserved while the second one measures the difference between the target data and source data. By minimizing the objective function the algorithm learns target and source transformation functions to preserve the topology of the target data and source data as well as to minimize their differences in the common latent feature space. HeMAP has its own classification algorithm. The algorithm employs the total-probability law to combine information from the source and target domains.

Heterogeneous Feature Augmentation (HFA) is a symmetric approach for problems with a target domain and a source domain that share common class-label set (Duan et al., 2012). As its name suggests the common latent feature space is augmented by the target-domain features and source-domain features. The transformation functions are jointly represented by a matrix. To find the joint transformation matrix an optimization problem to minimize the structural risk functional of SVMs is defined. The problem is tackled by an alternating optimization algorithm that simultaneously solves the dual problem of SVMs and finds the corresponding optimal joint transformation matrix. The final classifier for HFA is the SVM classifier derived together with the transformation matrix (i.e. the transformation functions represented by the matrix are related to SVMs).

We consider two approaches to asymmetric heterogeneous domain adaptation. The first approach is Asymmetric Regularized Cross-Domain Transformation (ARC-t) (Kulis et al., 2011). ARC-t is applicable for problems containing a target domain and a source domain that have in common the same class-label set. The transformation function is represented by a matrix W that maps an instance from the source domain to an instance in the target domain. The matrix W is learned in a non-linear Gaussian RBF kernel space. This is done by minimizing a matrix regularizer and a set of constraints imposed on any pair of target instance and projected source instance (for example using class information).

Sparse Heterogeneous Feature Representation (SFHR) is a asymmetric approach for problems with a target domain and a source domain that share common class-label set (Duan et al., 2012). The key idea is similar to that of the ARC-t approach. The difference is that the target data and source data is first represented in a code space based on some class decomposition scheme and then the transformation matrix W is being learned. This is done with nonnegative LASSO optimization. The experiments showed that SFHR is sensitive to the type of the class decomposition scheme employed. The best results are reported for the exhaustive ECOC scheme.

Table 6.1: HDA Approaches

HDA Method	# Source Domains	Domain Assumption	Class Correspondence	Topology Preservation	Classifier Independence	Specific Classifiers
DAMA	≥ 1	yes	yes	yes	yes	yes
HeMAP	1	no	no	yes	no	yes
HFA	1	no	yes	no	yes	yes
ARC-t	1	no	yes	yes	yes	no
SFHR	1	no	yes	yes	yes	no

Below in Table 6.1 we summarize the HDA approaches considered so far. The approaches are described in terms of:

- number of source domains that can be adapted;
- domain assumption;
- class correspondence used for learning transformation functions;
- data topology preservation after domain adaptation;
- classifier independence (in the sense that any classifier can be employed after domain adaptation);
- specific classifiers that the approaches provide.

From Table 6.1 we may conclude that we need a HDA algorithm that can adapt several source domains without imposing any assumptions. The algorithm has to employ class correspondence to match better (projected) same-class instances and it has to preserve the data topology (at least locally). The adapted datasets have to applicable for any classification model. Still, it is desirable that the algorithm can provide a specific classifier tailored to the transformations employed.

In the rest of this chapter we introduce a HDA algorithm that accommodate the requirements we described above. The algorithm follows the symmetric approach to heterogeneous domain adaptation. It is based on class decomposition schemes which makes it unique in this field.

6.3 Problem Formulation

We consider the problem of domain adaptation in the context of the classification problem (see Chapter 2). We define a classification domain as a triple that consists of an input space X with d features $X(j)$ ($j \in \{1, \ldots, d\}$), a finite output set Y of K class labels, and an unknown probability distribution p over $X \times Y$. We assume the presence of a target domain and at least one source domain. The target domain is the domain of interest. It is given by a target input space X^T with d_T features, an output set Y^T of class labels, and a target probability distribution p^T over $X^T \times Y^T$. The target training data D_T is a multi-set of N_T labeled instances $(x_i^T, y_i^T) \in X^T \times Y^T$ generated independently from p^T. The source domain is an auxiliary domain. It is given analogously; i.e. by a source input space X^S with d_S features, an output set Y^S of class labels, and a source probability distribution p^S over $X^S \times Y^S$. The source training data D_S (as D_T) is a multi-set of N_S labeled instances $(x_i^S, y_i^S) \in X^S \times Y^S$ generated independently from p^S.

Given the target training data D_T and the source training data D_S (typically $N_T << N_S$), the classification problem in domain adaptation is to provide a good estimate $\hat{y} \in Y$ of the true class of a target query instance $x_q^T \in X^T$ according to the probability distribution p^T.

The difficulty of the classification problem in domain adaptation depends mainly on the relationship between the target and source input spaces X^T and X^S. If X^T and X^S are the same, the domain adaptation is called homogeneous and is relatively simple, since the target data and source data lie in the same feature space. If X^T and X^S are different, the domain adaptation is called heterogeneous and is relatively difficult, since we need to find common feature space. This space can be target input space (assymettric transformation), or a common latent space (symmetric transformation).

In this chapter we consider a classification problem in the heterogeneous domain adaptation. The problem is characterized as follows:

- the target input space X^T is given by d_T continuous features $X^T(j)$ for $j \in \{1, \ldots, d_T\}$, i.e. $X^T \subseteq \mathbb{R}^{d_T}$;
- the source input space X^S is given by d_S continuous features $X^S(j)$ for $j \in \{1, \ldots, d_S\}$, i.e. $X^S \subseteq \mathbb{R}^{d_S}$;
- the target and source input spaces X^T and X^S are different, i.e. $X^T \neq X^S$;
- the target and source domains share the same output set Y of class labels, i.e. $Y^T = Y^S = Y$;
- the number K of class labels in Y is greater than 2, i.e. $K > 2$.

6.4 Bunching Algorithm for Heterogeneous Domain Adaptation

In this section we introduce Bunching.HDA as an algorithm for the classification problem in heterogeneous domain adaptation specified in the previous section. The algorithm follows the symmetric approach to the heterogeneous domain adaptation. It projects the target data and source data into a common latent feature space using a class decomposition scheme. Given that the target and source domains share the same output set Y of class labels, the common latent feature space is the code space associated with the class decomposition scheme used. Therefore, the algorithm first builds the encoding functions (mappings) for the target data, source data, and class labels. Then it employs these functions to project all the data and class labels in the code space, i.e. the common latent feature space.

The encoding functions are built using an optimization technique similar to that used in the Bunching algorithm proposed in (Dekel and Singer, 2002) [1]. An important feature of this technique is that the optimization process employs the class labels projected in the code space (i.e. class code words) in addition to the projected target and source instances. This allows two classification rules to be implemented on the level of the code space: the decoding (classification) rule of the class

[1]This explains the name of the proposed algorithm.

decomposition schemes (see Chapter 2, Subsection 2.3.3) and the decoding (classification) rule of the instance decomposition schemes (see Chapter 3, Subsection 3.2.2).

The Bunching.HDA algorithm is described in detail in subsection 6.4.1. The classification rules associated with the algorithm are provided in subsection 6.4.2.

6.4.1 Description

Given a class decomposition scheme with B class partitions, the Bunching.HDA algorithm adapts the target and source domains by projecting the target instances from $X^T \subseteq \mathbb{R}^{d_T}$, the source instances from $X^S \subseteq \mathbb{R}^{d_S}$, and the class labels from Y into a code space $\mathcal{C} \subseteq [0,1]^B$ so that the projected instances are grouped around the code words of their classes. The algorithm operates under the assumption that the output set Y consists of K standard unit vectors in $\mathbb{R}^K$ (i.e. $Y \subseteq \mathbb{R}^K$) so that the label of the k-th class ($k \in \{1, \dots, K\}$) is given by a standard unit vector whose k-th bit equals 1. In this case the projection process is realized using three composite mappings:

(1) $\sigma \circ T^T : \mathbb{R}^{d_T} \to [0,1]^B$ from the target feature space X^T to the code space $\mathcal{C}$, and

(2) $\sigma \circ T^S : \mathbb{R}^{d_S} \to [0,1]^B$ from the source feature space X^S to the code space $\mathcal{C}$, and

(3) $\sigma \circ C : \mathbb{R}^K \to [0,1]^B$ from the output set Y of K class labels to the code space $\mathcal{C}$,

where

- $\sigma : \mathbb{R}^B \to [0,1]^B$ is a multivariate logistic function defined as $\sigma_b(w) = (1+e^{-w(b)})^{-1}, b \in \{1, ..., B\}$, and
- $T^T : \mathbb{R}^{d_T} \to \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times d_T}$, and
- $T^T : \mathbb{R}^{d_S} \to \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times d_S}$, and
- $C : \mathbb{R}^K \to \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times K}$.

The main task of the Bunching.HDA algorithm is to learn the mappings (matrices) T^T, T^S, and C using the target and source data. We note that the composite mappings $\sigma \circ T^T$ and $\sigma \circ T^S$ are essentially encoding functions of the class decomposition scheme used for the target domain and the source domain, respectively. More precisely, the rows of the target matrix T^T represent logistic regression models of the binary classifiers trained on the target data while the rows of the source matrix T^S represent logistic regression models of the binary classifiers trained on the source data. The composite mapping $\sigma \circ C$ is a class encoding function: it determines the code word for each class label and it is common for the target domain and the source domain. Due to the unit-vector domain of the output set Y of class labels, the matrix C is the transposed coding matrix used; i.e. any class is represented by a column in C. We note that in standard decomposition schemes (such as One-vs-all, eECOC, mECOC) the matrix C stays fixed. Bunching.HDA, however, adjusts C to better fit the target and source domains.

Once the mappings T^T, T^S, and C have been learned, any target instance $x^T \in X^T$ is projected into $\sigma(T^T x^T) \in \mathcal{C}$, any source instance $x^S \in X^S$ is projected into $\sigma(T^S x^S) \in \mathcal{C}$, and any class label $y \in Y$ is projected into $\sigma(Cy) \in \mathcal{C}$. The projection scheme is provided in Figure 6.1.

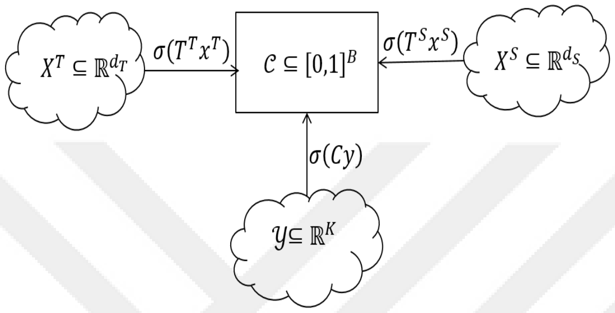

$X^T \subseteq \mathbb{R}^{d_T}$
$\sigma(T^T x^T)$
$\mathcal{C} \subseteq [0,1]^B$
$\sigma(T^S x^S)$
$X^S \subseteq \mathbb{R}^{d_S}$
$\sigma(Cy)$
$\mathcal{Y} \subseteq \mathbb{R}^K$

Figure 6.1: Projection Scheme

We note that any element p of the code space $\mathcal{C}$ can be viewed as a parameter vector for a multivariate Bernoulli variable[2]. If p is a true parameter vector in $\mathcal{C}$ and q is an approximate parameter vector in $\mathcal{C}$, then to quantify the information gain from q to p we can use the *KL* divergence defined as follows:

$$KL\,[p\|q] = \sum_{b=1}^{B} \left[p(b) \log \left(\frac{p(b)}{q(b)} \right) + (1 - p(b)) \log \left(\frac{1 - p(b)}{1 - q(b)} \right) \right]. \tag{6.1}$$

The Bunching.HDA algorithm aims at mapping instances and their corresponding labels to similar locations in the code space $\mathcal{C}$. This leads us to define a loss l for each instance label pair $(x, y) \in D_T \cup D_S$. As our measure in $\mathcal{C}$ is the *KL* divergence, a loss l of the instance x for class label y w.r.t. the mappings T and C can be readily defined as:

$$l(x, y \mid C, T) = KL\,[\sigma\,(Cy)\,\|\sigma\,(Tx)]\,,$$

where

$$T = \begin{cases} T^T & \text{if } (x, y) \in D_T, \\ T^S & \text{if } (x, y) \in D_S. \end{cases}$$

The loss $\mathcal{L}$ for all N_T target instances $(x_i^T, y_i^T) \in D_T$ is defined as:

$$\mathcal{L}(D_T \mid C, T^T) = \sum_{i=1}^{N_T} l(x_i^T, y_i^T \mid C, T^T),$$

and the loss $\mathcal{L}$ for all N_S source instances $(x_i^S, y_i^S) \in D_S$ is defined as:

$$\mathcal{L}(D_S \mid C, T^S) = \sum_{i=1}^{N_S} l(x_i^S, y_i^S \mid C, T^S).$$

Minimizing $\mathcal{L}(D_T \mid C, T^T)$ with respect to T^T and C forces the projected target instances $\sigma(T^T x_i^T)$ to be close to the code words $\sigma(C y_i^T)$ of their class labels y_i^T in the code space $\mathcal{C}$. Like-

[2]If X is a multivariate Bernoulli variable, then the probability mass function is given by $f(X = (X(1), ..., X(B))', p) = \prod_{b=1}^{B} p(b)^{X(b)} (1 - p(b))^{1 - X(b)}$ for $(X(1), ..., X(B))' \in \{0, 1\}^B$.

wise, minimizing $\mathcal{L}(D_S \mid C, T^S)$ with respect to T^S and C forces the projected source instances $\sigma(T^S x_i^S)$ to be close to the code words $\sigma(Cy_i^S)$ of their class labels y_i^S in the code space $\mathcal{C}$. Taken together, we conclude that minimizing:

$$\mathcal{L}(D_T \mid C, T^T) + \mathcal{L}(D_S \mid C, T^S) \tag{6.2}$$

causes the target and source instances from the same class to be projected to nearby points in the code space $\mathcal{C}$ no matter which domain they belong to. For this reason we pay a particular attention to the mapping (matrix) C.

The matrix C' has to have large separation properties[3]. The large row separation property means that for any class label $y_1 \in Y$ the code word Cy_1 has to be well-separated (distant) from the code Cy_2 of any other class label $y_2 \in Y$ (see Section 2.3.4 in Chapter 2). If this property holds, then the code words $\sigma(Cy_1)$ and $\sigma(Cy_2)$ are well-separated in the code space $\mathcal{C}$ in terms of the KL-divergence; i.e. $KL\left[\sigma\left(Cy_1\right) \| \sigma\left(Cy_2\right)\right]$ and $KL\left[\sigma\left(Cy_2\right) \| \sigma\left(Cy_1\right)\right]$ are both large. Since the projected instances arrive close to the code words of the their class labels, this causes that the projected instances of different class labels to be well separated (distant) in the code space $\mathcal{C}$.

The large row separation property means that for any $b_1 \in \{1, \dots, B\}$ the partition code word $C'(\cdot, b_1)$ has to be well-separated (distant) from the partition code word $C'(\cdot, b_2)$ and its complement for any other $b_2 \in \{1, \dots, B\} \setminus \{b_1\}$ (see Section 2.3.4 in Chapter 2). If this property holds, then the logistic regression models represented by columns $C'(\cdot, b_1)$ and $C'(\cdot, b_2)$ commit less errors simultaneously. This causes the loss $l(x, y \mid C, T)$ (defined through Equation (6.1)) for any labeled instance (x, y) to decrease which in turn decreases the total loss (6.2) for all the projected target and source instances. The latter means that the projected instances are grouped more around the code words of their classes in the code space $\mathcal{C}$.

From the above we may conclude that it is desirable that (the transpose of) the matrix C has large separation properties. To guarantee this we propose to initialize the matrix C using a coding matrix C_{ref} with known large separation properties (e.g. the eECOC coding matrix). Since the matrix C is being learned, we have to make sure that it is close to the matrix C_{ref}. For this purpose, we add the following *loss* term to the objective function (6.2) to penalize the KL divergence between the projections of the classes obtained by C and C_{ref} for the target and the source instances:

$$\mathcal{L}(D_T \mid C_{ref}, C) + \mathcal{L}(D_S \mid C_{ref}, C), \tag{6.3}$$

where

$$\mathcal{L}(D_T \mid C_{ref}, C) \quad = \sum_{(x_i^T, y_i^T) \in D_T} KL\left[\sigma\left(Cy_i^T\right) \| \sigma\left(C_{ref}\, y_i^T\right)\right], \text{ and}$$

[3]We note that the matrix C is the transposed coding matrix.

$$\mathcal{L}(D_S \mid C_{ref}, C) \quad = \quad \sum_{(x_i^S, y_i^S) \in D_S} KL\left[\sigma\left(C y_i^S\right) \| \sigma\left(C_{ref}\, y_i^S\right)\right].$$

The final objective function that we are to minimize with respect to T^T, T^S and C is defined as follows:

$$\begin{aligned} \mathcal{O}(D_T, D_S \mid T^T, T^S, C) =& \mathcal{L}(D_T \mid C, T^T) + \mathcal{L}(D_S \mid C, T^S) \\ &+ \alpha \left(\mathcal{L}(D_T \mid C_{ref}, C) + \mathcal{L}(D_S \mid C_{ref}, C)\right), \end{aligned} \tag{6.4}$$

where $\alpha > 0$ is a regularization parameter that balances the mismatch between C and C_{ref}.

Once the objective function (6.4) has been defined, we introduce the Bunching.HDA algorithm. The algorithm learns the mappings (matrices) T^T, T^S, and C from the target and source data by minimizing this function. It is given in Figure 7. The Bunching.HDA algorithm is of *Alternating Minimization* (AM) type (Csiszar and Tusnady, 1984). It first improves the matrix T^T w.r.t. the matrix C, then improves the matrix T^S w.r.t. the matrix C, and, finally, improves the matrix C w.r.t. the matrices T^T and T^S. The process is repeated $maxIterOut$ times to minimize the objective function (6.4).

The IMPROVE$-T$ function improves the mapping T^T (resp. T^S) w.r.t. C. This is done by minimizing the loss term $\mathcal{L}(D_T|C, T^T)$ (resp. $\mathcal{L}(D_S|C, T^S)$) of the objective function Eq. (6.4). The IMPROVE$-T$ function is a gradient descent algorithm with the backtracking line-search method (Gradient derivation is given in Preposition 1, Appendix A). In each iteration it first computes the gradient matrix G and then executes the backtracking method using a parameter β to determine a step size η that sufficiently reduces the objective function Eq. (6.4) (see lines 7-9 in IMPROVE-T). The function stops after the $maxIterIn$ iteration and outputs the improved matrix T^T (resp. T^S).

The IMPROVE$-C$ function improves the mapping C w.r.t. T^T and T^S. This is done by minimizing the loss term (6.3) in the objective function Eq. (6.4). Minimizing is analytical, since the objective function Eq. (6.4) has a unique stationary point for a fixed choice of T^T and T^S (see Preposition 2, Appendix A).

We note that the objective function (6.4) can be extended to a classification problem for which:

- we have a target domain and no source domain, i.e. the classification problem is standard. The Bunching.HDA algorithm in this case does not process source data and is modified as follows: lines 4 in the Bunching.HDA algorithm and the function IMPROVE$-C$ are skipped and line 5 in the function IMPROVE$-C$ is modified to reflect target data only.In this way Bunching.HDA algorithm is similar to the Bunching algorithm proposed in (Dekel and Singer, 2002). The difference primarily lies in the optimization technique that improves the mapping T^T.
- we have a target domain and several source domains, i.e. the classification problem is a multi-domain adaptation problem. The Bunching.HDA algorithm in this case is modified as follows: the IMPROVE$-T$ function in the loop of the Bunching.HDA algorithm is called for each source domain p to improve the mapping T^{S_p} and line 5 in the function IMPROVE$-C$ is modified to reflect the data from all the source domains.

Algorithm 7 The Bunching.HDA Algorithm

Bunching.HDA

Input: target data D_T, source data D_S, initial projection matrices T_0^T and T_0^S, reference projection matrix C_{ref}, step parameter $\beta \in (0,1)$, regularization parameter $\alpha > 0$, and iteration numbers $maxIterOut > 1$ and $maxIterIn > 1$.

Output: projection matrices T^T, T^S and C.

```
1: $C_0 := C_{ref}$;
2: for $t := 1$ to $maxIterOut$ do
3:     $T_t^T :=$ IMPROVE-T($D_T, C_{t-1}, T_{t-1}^T, maxIterIn$);
4:     $T_t^S :=$ IMPROVE-T($D_S, C_{t-1}, T_{t-1}^S, maxIterIn$);
5:     $C_t :=$ IMPROVE-C($D_T, D_S, \alpha, C_{ref}, T_{t-1}^T, T_{t-1}^S$);
6: end for
7: return $T^T$, $T^S$, and $C$
```

IMPROVE-T

Input: data D, projection matrices C and T, and iteration number $maxIterIn > 1$.

Output: projection matrix T.

```
1: let $B$ and $d$ be the sizes of $T$;
2: for $t := 1$ to $maxIterIn$ do
3:     Set $\eta$ equal to 1;
4:     for $i := 1$ to $B$ and $j := 1$ to $d$ do
5:         $G(i,j) = \sum_{(x,y)\in D} x(j) \Big(\sigma\big(-C(i,\cdot)y\big)\sigma\big(T(i,\cdot)x\big) - \sigma\big(C(i,\cdot)y\big)\sigma\big(-T(i,\cdot)x\big)\Big)$;
6:     end for
7:     while $\mathcal{L}(D|C, T-\eta G) > \mathcal{L}(D|C,T) - \frac{\eta}{4}\|G\|^2$ do
           $\eta := \beta\eta$;
8:     end while
9:     $T := T - \eta \times G$;
10: end for
11: return $T$.
```

IMPROVE-C

Input: target data D_T, source data D_S, regularization parameter $\alpha > 0$, reference projection matrix C_{ref}, and projection matrices T^T and T^S.

Output: projection matrix C.

```
1: let $B$ and $K$ be the sizes of $C_{ref}$;
2: for $i := 1$ to $B$ and $j := 1$ to $K$
3:     $D_{Tj} := \Big\{\big(x^T, y\big) \in D_T \mid y = y_j \wedge y_j \in Y\Big\}$;
4:     $D_{Sj} := \Big\{\big(x^S, y\big) \in D_S \mid y = y_j \wedge y_j \in Y\Big\}$;
5:     $C(i,j) := \frac{1}{1+\alpha}\left(\alpha C_{ref} + \frac{\sum_{(x^T,y)\in D_{Tj}} T^T x^T + \sum_{(x^S,y)\in D_{Sj}} T^S x^S}{|D_{Tj}|+|D_{Sj}|}\right)$;
6: end for
7: return $C$.
```

6.4.2 Classification of Target Instances with Bunching.HDA

Once the Bunching.HDA algorithm has learned the mappings T^T, T^S, and C, any target instance $x^T \in X^T$ is projected into $\sigma(T^T x^T) \in \mathcal{C}$, any source instance $x^S \in X^S$ is projected into $\sigma(T^S x^S) \in \mathcal{C}$, and any class label $y \in Y$ is projected into $\sigma(Cy) \in \mathcal{C}$. We can use these three types of projections to define the Bunching.HDA classification rules for target query instances. The rules are described below in detail.

The first classification rule associated with the Bunching.HDA algorithm is the classification rule of the class decomposition schemes. Given the code words $\sigma(Cy)$ for class labels $y \in Y$ and a target query instance $x_q^T \in X^T$, the instance is first projected into $\sigma(T^T x_q^T)$ in the code space $\mathcal{C}$ (i.e. the instance is encoded). Then we compute set $\hat{Y}$ of class labels $y \in Y$ whose code words $\sigma(Cy)$ in $\mathcal{C}$ are closest to the projection $\sigma(T^T x_q^T)$ in terms of the KL divergence:

$$\hat{Y} = \operatorname*{argmin}_{y \in Y} KL[\sigma(Cy) \parallel \sigma(T^T x_q^T)]. \tag{6.5}$$

If $|\hat{Y}| > 1$, the final class label estimate $\hat{y}$ for the target query instance x_q^T is chosen randomly from the classes in $\hat{Y}$ (i.e. the instance is decoded). We note the presented classification rule is a prototype-based classification rule (Mitchell, 1997). That is why, it is denoted by Bunching.HDA.Pr.

The second classification rule associated with the Bunching.HDA algorithm is the classification rule of the instance decomposition schemes. Given the set $D_{\mathcal{C}}$ of the projected target and source instances[4] and a target query instance $x_q^T \in X^T$, the instance is first projected into $\sigma(T^T x_q^T)$ in the code space $\mathcal{C}$ (i.e. the instance is encoded). Then the set NN of the nearest target and source neighbors of $\sigma(T^T x_q^T)$ in $D_{\mathcal{C}}$ is computed equal to:

$$\operatorname*{argmin}_{x \in D_{\mathcal{C}}} KL[x \parallel \sigma(T^T x_q^T)].$$

The set $\hat{Y}$ of possible estimates of the true class label for x_q^T is computed as a set of class labels with majority of the instances in NN. More precisely,

$$\hat{Y} = \operatorname*{argmax}_{y \in Y} \#\{x_i \in NN | y_i = y\}.$$

If $|\hat{Y}| > 1$, the final class label estimate $\hat{y}$ for the target query instance x_q^T is chosen randomly from the classes in $\hat{Y}$ (i.e. the instance is decoded). We note the presented classification rule is a nearest-neighbor classification rule (Mitchell, 1997). That is why, it is denoted by Bunching.HDA.NN.

In addition to the Bunching.HDA.Pr classification rule and the Bunching.HDA.NN classification rule, other classification rules are also possible. Once the target data and source data are projected in the code space $\mathcal{C}$ we can train any classification model on the union of these data. This suggests that Bunching.HDA is a classifier independent approach. We demonstrate this property in the next Chapter using Random Forest classifiers and Naive Bayes classifiers trained in the code space $\mathcal{C}$.

[4] $D_{\mathcal{C}} = \{\sigma(T^T x_i^T) | (x_i^T, y_i^T) \in D_T\} \cup \{\sigma(T^S x_i^S) | (x_i^S, y_i^S) \in D_S\}$.

6.5 Experiments

This section presents our experiments, results, and initial conclusions. The experiments are divided into two groups. The first group of experiments, given in Subsection 6.5.1, were done to provide a convergence analysis of the Bunching.HDA algorithm. The second group of experiments, given in Subsection 6.5.1, were done to estimate the generalization performance of the algorithm on three domain adaptation datasets. The estimated performance is compared with that of baseline classifiers trained on the target data only and two domain adaptation approaches.

6.5.1 Convergence Analysis Experiments

For the convergence analysis we run two series of experiments. First, we investigated the convergence properties of the function IMPROVE-T in Bunching.HDA. Second, we investigated the convergence properties of the Bunching.HDA algorithm. Both series were run for values of the regularization parameter α from $\{0.04, 0.02, 1, 5, 25\}$ and two coding matrices: the mECOC matrix and the eECOC matrix. The matrices were chosen to represent two extreme cases in terms of the column number (see Chapter 2, Section 2.4.2) and thus allowing convergence analysis for minimal and maximal number of features for the code space $\mathcal{C}$ (i.e., the common space).

The experiments were performed on a domain adaptation problem that we created based on the Segment dataset (Bache and Lichman, 2013). The Segment dataset has 19 features and 2310 instances grouped into 7 distinct classes. To create the target data and the source data, we first randomly partitioned the set of features into two disjointed subsets with 10 and 9 features, respectively. Then, we randomly selected 500 instances for the target data and the remaining 1810 instances for the source data so that the seven Segment classes are present in each dataset.

To investigate the convergence properties of the function IMPROVE-T over the target data we sequentially executed the function IMPROVE-T and function IMPROVE-C 10 times (lines 3 and 5 of the Bunching.HDA algorithm). The $maxIterIn$ parameter in IMPROVE-T was set 10; i.e. the gradient descent was iterated 10 times in each run of IMPROVE-T. We plotted the loss $\mathcal{L}(D_T \mid C, T^T)$ associated with IMPROVE-T on the target data against the total number of the iterations of IMPROVE-T. The loss was normalized by the column size of the coding matrices used to make the results comparable for the mECOC and eECOC matrices.

To investigate the convergence properties of the function IMPROVE-T over the source data we run an analogous experiment. The experimental settings were the same. The results are shown in Figure 6.3.

Analyzing the results presented above we may reach the following conclusions for the functions IMPROVE-T and function IMPROVE-C:

- both functions reduce the losses $\mathcal{L}(D_T \mid C, T^T)$ and $\mathcal{L}(D_S \mid C, T^S)$. The loss reduction is larger for earlier iterations.
- the function IMPROVE-T converges rather quickly given a fixed matrix C. Hence, the parameter $maxIterIn$ can be set small.
- the function IMPROVE-T converges irrespective of the regularization parameter α.

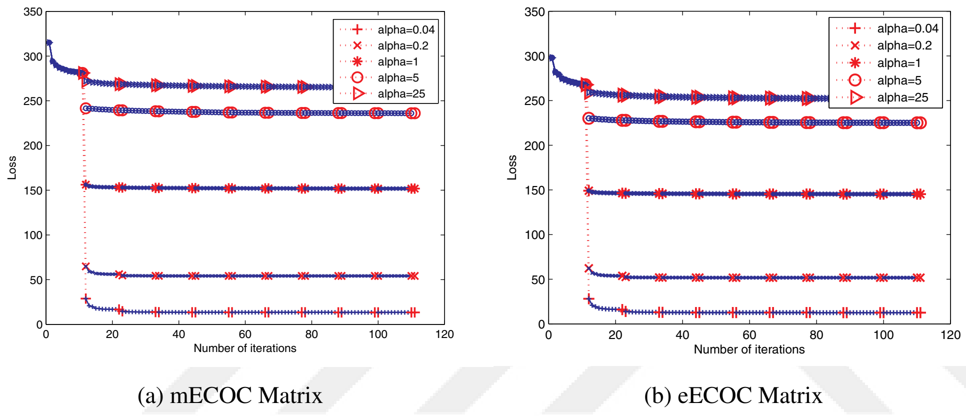


Figure 6.2: Loss associated with the functions IMPROVE-T and IMPROVE-C on the target data. The blue segments of the lines indicate the reduction in the loss caused by IMPROVE-T and the red segments indicate the reduction caused by IMPROVE-C.

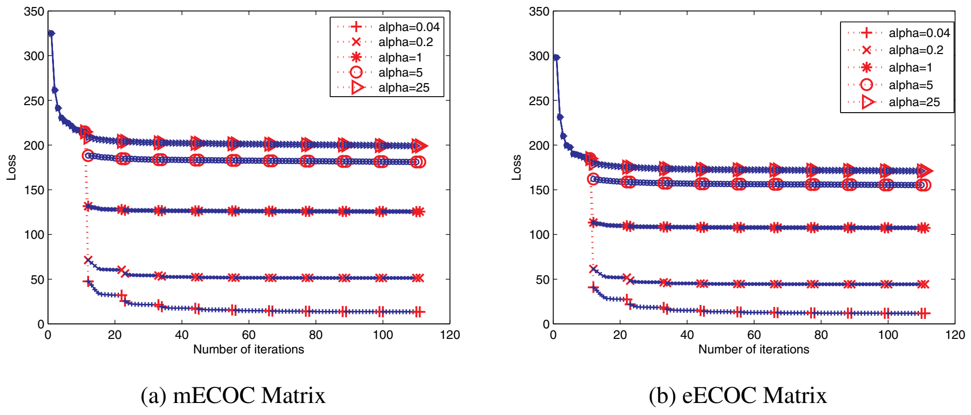
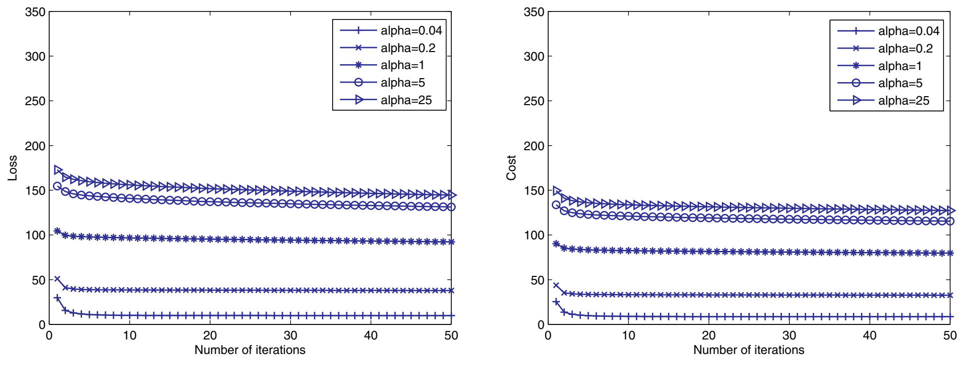


Figure 6.3: Loss associated with the functions IMPROVE-T and IMPROVE-C on the source data. The blue segments of the lines indicate the reduction in the loss caused by IMPROVE-T and the red segments indicate the reduction caused by IMPROVE-C.

- the losses $\mathcal{L}(D_T \mid C, T^T)$ and $\mathcal{L}(D_S \mid C, T^S)$ associated with the function IMPROVE-T are reduced depending on the coding matrix used. The normalized loss of the mECOC matrix is larger than that of the eECOC matrix. This can be explained by the column size of the eECOC matrix which results in more classifiers that commit less errors simultaneously.

To investigate the convergence properties of the Bunching.HDA algorithm we run the algorithm for $maxIterIn$ equal to 10 and $maxIterOut$ equal to 50. We plotted the total loss (6.4) associated with Bunching.HDA against the number of the iterations of Bunching.HDA in Figure 6.4. Again, the loss was normalized by the column size of the coding matrices.



(a) mECOC Matrix (b) eECOC Matrix

Figure 6.4: Loss associated with the Bunching.HDA algorithm.

Analyzing the results presented above we may reach the following conclusions for the Bunching.HDA algorithm:

- it reduces the total loss (6.4). The loss reduction is larger for earlier iterations.
- it converges rather quickly. Hence, the parameter $maxIterOut$ can be set small.
- it converges irrespectively of the regularization parameter α.
- the total loss (6.4) is reduced depending on the coding matrix used. The normalized loss of the mECOC matrix is larger than that of the eECOC matrix. This can be explained by the fact (established above) that the losses $\mathcal{L}(D_T \mid C, T^T)$ and $\mathcal{L}(D_S \mid C, T^S)$ in the total loss (6.4) are reduced proportionally to the column size of the coding matrices.

6.5.2 Experiments on Three Domain Adaptation Problems

This subsection provides the experiments of the Bunching.HDA algorithm applied on three domain adaptation datasets: the Office dataset (Saenko et al., 2010), the Wikipedia dataset (Rasiwasia et al., 2010) and the Multiple Feature (Mfeat) dataset (Bache and Lichman, 2013). The generalization performance of the classification rules associated with the algorithm is estimated and compared with that of baseline classifiers trained on the target data only and two domain adaptation approaches.

Settings of the Bunching.HDA Algorithm

The Bunching.HDA algorithm was set up as follows. The reference coding matrix C_{ref} was set equal to the coding matrix of the One-vs-All class decomposition scheme. The regularization parameter α took values from the set $\{0.04, 0.02, 1, 5, 25\}$ and the results are reported for the α value that maximizes the accuracy. The iteration parameters $maxIterIn$ and $maxIterOut$ were set to 10 and 50, respectively (following the recommendation from the previous section). The parameter of the backtracking method β of the function IMPROVE-T was set to 0.5. The two classification rules associated with the Bunching.HDA algorithm were applied: Bunching.HDA.Pr and Bunching.HDA.NN.

We note that the reference coding matrix C_{ref} was set equal to the One-vs-All coding matrix for two reasons: (1) to reduce the computational complexity (C_{ref} is a $K \times K$ matrix) and (2) to make the setup of the Bunching.HDA algorithm comparable with that of the baseline classifiers (given in the next subsection). In addition, the C_{ref} setup ensures a kind of worst-case generalization performance of the Bunching.HDA.Pr and Bunching.HDA.NN classification rules due to small row and column separation of the One-vs-All matrices. This means that the comparison with the baseline classifiers is rather fair: the setup of Bunching.HDA is unfavorable for the algorithm.

Baselines

We compare the Bunching.HDA algorithm against two groups of baseline classifiers. The first group is the group of classifiers trained on the target data only[5]. It consists of:

- kNN.T, a first nearest neighbor classifier (Mitchell, 1997).
- SVM.T, a multiclass SVM classifier with a linear kernel and default setting provided in LIBSVM (Chang and Lin, 2011). It employs the One-vs-All class decomposition scheme for multi class classification problems.
- Bunching.T, the Bunching algorithm (Dekel and Singer, 2002) with One-vs-All reference coding matrix C_{ref}, regularization parameter α equal to 1.0, and the number of iteration equal to 100.

The second group is the group of domain adaptation approaches trained on the target data and source data. It consists of:

- HFA, the Heterogeneous Feature Augmentation approach with default setting (Duan et al., 2012): the regularization parameter was set to 1 and the parameter that controls the complexities of the learned transformation matrices was set to 100. HFA employed the One-vs-All ensemble of SVM classifiers with linear kernel in the common space.
- SHFR, the Sparse Heterogeneous Feature Representation approach with default setting (the One-vs-All class decomposition scheme was used for asymmetric transformation). SFHR employed the One-vs-All ensemble of SVM classifiers with linear kernel in the common space.

[5]This is indicated with postfix T.

Experiments on the Office Dataset

The Office dataset was introduced in (Saenko et al., 2010) and was used in (Duan et al., 2012; Kulis et al., 2011; Saenko et al., 2010). It contains 4652 images from three domains: Amazon, dSLR, and webcam, that share the same 31 object categories (classes). The images in each domain were captured with different photo machines under different lighting conditions. Figure 6.5 shows images of laptops that form one of 31 categories of interest in the Amazon, dSLR and webcam domains.

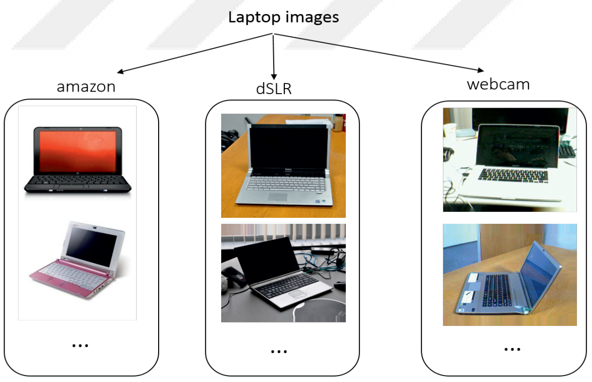


Figure 6.5: Laptop images taken from different domains.

To represent the images (Saenko et al., 2010; Duan et al., 2012; Kulis et al., 2011) extracted SURF features (Bay et al., 2006). The images in the amazon and webcam domains are represented by 800 histogram features whereas the images in the dSLR domain are represented by 600 histogram features. We defined two classification problems. The first problem (resp. second problem) considers the dSLR domain as target domain and the amazon domain (resp. the webcam domain) as source domain. To estimate the accuracy of the classifiers we followed the evaluation (hold-out) protocol given in (Duan et al., 2012; Kulis et al., 2011). 20 (resp. 8) training images were randomly selected for the source domain amazon (resp. webcam) and 3 training images were randomly selected from the target domain dSLR from each category. The remaining target dSLR images were served as target test instances. The evaluation process was repeated 10 times. The results are provided in Figure 6.2.

Table 6.2: The classification accuracies (in percent) for the Office Data. Bold numbers indicate accuracies that are statistically greater than others based on the two-tailed t-test on significance level of 0.05.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR
dSLR	amazon	46.12	53.13	49.17	54.79	52.66
	webcam				53.77	55.15

(a) Baseline results

Target Domain	Source Domain	Bunching.HDA.NN	Bunching.HDA.Pr
dSLR	amazon	54.66	**57.7**
	webcam	54.13	**58.65**

(b) Bunching.HDA results

Experiments on the Wikipedia Dataset

The Wikipedia dataset was introduced in (Rasiwasia et al., 2010). It contains 5732 instances from two domains: text domain and image domain. Both domains have 2866 instances and share the same 10 object categories (classes): art, biology, geography, history, literature, media, music, royalty, sport, and welfare. The instances in the text domain were preprocessed in the following way. The features were derived by a Latent Dirichlet Allocation (LDA) model to identify 10 continuous features so that each feature corresponds to one of the ten categories and shows the probability of being that category. The instances in the image domain were preprocessed as well. First SIFT keypoints were extracted from the image instances (Lowe, 2004) and then the k-means clustering was applied to convert the keypoints to a bag of visual words. Since k was set to 128, the image instances were represented by 128 features.

For the Wikipedia dataset we defined one classification problem. This problem considers the text domain as target domain and the image domain as source domain. To estimate the accuracy of the classifiers we followed the evaluation protocol similar to that of the office dataset. For each category 10 training text instances were randomly selected as target instances and the remaining 2856 text instances were used as target test instances. All the image instances served as source instances. The evaluation process was repeated 10 times. The results are provided in Figure 6.2.

Table 6.3: The classification accuracies (in percent) for the Wikipedia Data.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR
Text	Image	59.95	60.02	61.66	62.35	63.59

(a) Baseline results

Target Domain	Source Domain	Bunching.HDA.NN	Bunching.HDA.Pr
Text	Image	56.88	64.53

(b) Bunching.HDA results

Experiments on the Multiple Feature Dataset

The Multiple Feature (Mfeat) dataset is provided by (Bache and Lichman, 2013). It contains 2000 hand-written images of ten numbers from 0 to 9 (classes). The images were preprocessed using different techniques which resulted in six different domains: mfeat-fou (given with 76 Fourier coefficients), mfeat-fac (given with 216 profile correlation features), mfeat-kar (given with 64 Karhunen-Love coefficients), mfeat-pix (given with 240 pixel average features), mfeat-zer (given with 47 Zernike moment features), and mfeat-mor (given with 6 morphological features). The mfeat-mor domain is considered as target domain. In this context, we defined five classification problems so that each remaining domain is considered as source domain once. To estimate the accuracy of the classifiers we followed the evaluation protocol similar to that given in the previous two subsections. For each category (number) 10 training mfeat-mor instances were randomly selected as target instances and the remaining 1990 mfeat-mor instances were used as target test instances. All the instances of the corresponding source domain served as source instances. The evaluation process was repeated 10 times. The results are provided in Figure 6.4.

Table 6.4: The classification accuracies (in percent) for the Multiple Feature Data. Bold numbers indicate accuracies that are statistically better than others based on the two-tailed t-test on significance level of 0.05.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR
mfeat-mor	mfeat-fou	44.71	43.7	50.14	63.03	64.22
	mfeat-fac				64.4	65.28
	mfeat-pix				63.07	64.45
	mfeat-kar				63.5	66.16
	mfeat-zer				63.51	65.12

(a) Baseline results

Target Domain	Source Domain	Bunching.HDA.NN	Bunching.HDA.Pr
mfeat-mor	mfeat-fou	**70.33**	61.55
	mfeat-fac	**70.91**	62.47
	mfeat-pix	**69.41**	58.12
	mfeat-kar	**71.1**	60.53
	mfeat-zer	**70.01**	65.01

(b) Bunching.HDA results

6.5.3 Results and Discussions

Tables 6.2, 6.3 and 6.4 show the accuracies of the Bunching.HDA classification rules and the baseline methods obtained on the Office dataset, Wikipedia dataset, and Mfeat dataset respectively. When comparing these accuracies we observe that the HDA methods outperform the classifiers trained on the target data in general. For example, the Bunching.HDA.NN rule improves the accuracy with 25% on the Mfeat dataset compared with the kNN.T classifier (a classifier with a similar classification rule) and the Bunching.HDA.Pr rule improves the accuracy in the range of $[2.87\%, 14.87\%]$ compared with the Bunching.T classifier (a target-domain version of Bunching.HDA). Thus, we conclude that using source data from different feature spaces can improve the accuracy of the classification on unseen target instances.

When comparing the Bunching.HDA classification rules with the baseline HDA methods, HFA and SHFR, we observe that they have a similar accuracy on the Wikipedia dataset while on the Office dataset and the Mfeat dataset at least one of the Bunching.HDA classification rules is a clear winner (the improvement in accuracy is in the range of $1\% - 4\%$ and it is statistically significant). Thus, we may conclude that the Bunching.HDA classification rules are capable of outperforming the HFA and SHFR methods (especially, taking into account the unfavorable experimental setting of the Bunching.HDA algorithm)

When comparing the Bunching.HDA classification rules between each other, we observe that the rules exhibit rather different performances. The Bunching.HDA.Pr rule outperforms the Bunching.HDA.NN rule on the Office dataset and Wikipedia dataset while the Bunching.HDA.NN rule outperforms the Bunching.HDA.Pr rule on the Mfeat dataset. The Bunching.HDA.Pr rule is preferable when the projected instances are more grouped around the code words of their classes in the code space $\mathcal{C}$. When this condition is violated, the Bunching.HDA.Pr tends to deteriorate. This happens for the case of the Mfeat dataset. The Bunching.HDA.Pr rule achieves the highest accuracy when the regularization parameter α is set too high ($\alpha = 50$). However, the accuracy is still lower than the accuracy obtained by the Bunching.HDA.NN rule (the average accuracy improvement is around 10%). From this we may conclude that the Bunching.HDA classification rules correspond to quite different states of HDA. Thus, to decide which rule to apply we need to characterize the HDA state produced by the Bunching.HDA algorithm. For example, relatively low values of the sub-loss (6.2) minimized by the Bunching.HDA algorithm imply HDA states that fit better the Bunching.HDA.Pr rule. Relatively high values of the sub-loss (6.2) imply HDA states that fit better the Bunching.HDA.NN rule.

6.6 Conclusion

This chapter proposed the Bunching.HDA algorithm for the classification problem in the presence of a target domain and several source domains. The algorithm first builds the encoding functions (mappings) for the target data, source data, and class labels. Then it employs these functions to project all the data and class labels in the common code space $\mathcal{C}$. In this context we note that the Bunching.HDA algorithm does not make any assumption on the underlying structure of the target and source domains. It only necessities common class label information for a domain correspondence.

To build the encoding functions the Bunching.HDA algorithm minimizes the total loss function (6.4). This groups the projected target and source instances around the code words of their classes in the code space $\mathcal{C}$. To make different class instances more separable in $\mathcal{C}$ the class encoding function is initialized using a coding matrix with well-presented separation properties.

The Bunching.HDA algorithm is an alternating-minimization algorithm. Its functions that improve target and source encoding functions converge quickly in each iteration. This causes the whole algorithm to converge quickly as well.

The availability of projected target and source instances together with the code words of class labels in the code space $\mathcal{C}$ allows two built-in classification rules: the decoding rule of the class decomposition schemes (Bunching.HDA.Pr) and the decoding rule of the instance decomposition schemes (Bunching.HDA.NN). Bunching.HDA.Pr (resp. Bunching.HDA.NN) is preferable when the sub-loss (6.2) is minimized for relatively low (high) values; i.e. when the spread of the same class instances is relatively low (high) around the code words of their classes. In addition, it was shown that any other classification rule is applicable: once the target data and source data are projected in the code space $\mathcal{C}$ we can train any classification model on the union of these data.

The Bunching.HDA algorithm was experimentally tested on three domain adaptation problems. The experiments showed that the Bunching.HDA algorithm is capable of outperforming standard domain adaptation methods.

Appendix A: Derivation of the Bunching.HDA Algorithm

Lemma 1. (Properties of the logistic function)
For the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$, it holds that:

(a) $\sigma(x)$ *is monotonically increasing with* $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

(b) $\sigma(-x) = 1 - \sigma(x)$.

(c) $\log\left(\frac{\sigma(x)}{1-\sigma(x)}\right) = x$ *and* $\sigma'(x) = \log\frac{x}{1-x}$.

Proof: By direct computation. □

Lemma 2. (Partial derivatives of the KL divergence)
The partial derivatives of the KL divergence of the multivariate Bernoulli distributions with parameter vectors $p, q \in (0,1)^B$ *are given by:*

(a) $\frac{\partial}{\partial p(b)} KL\left[p||q\right] = \log\left(\frac{p(b)}{1-p(b)}\right) - \log\left(\frac{q(b)}{1-q(b)}\right)$,

(b) $\frac{\partial}{\partial q(b)} KL\left[p||q\right] = \frac{q(b)-p(b)}{q(b)(1-q(b))}$.

Proof: By direct computation. □

Lemma 3. (Differentiation of the loss function)

(a) For any $i \in \{1,\dots,B\}$, *and* $j \in \{1,\dots,d\}$, *the differentiations of the loss function is given by:*

$$\frac{\partial KL\left[\sigma\left(Cy\right)||\sigma\left(Tx\right)\right]}{\partial T(i,j)} = \left(\sigma\left(-C\left(i,\cdot\right)y\right)\sigma\left(T\left(i,\cdot\right)y\right) - \sigma\left(C\left(i,\cdot\right)y\right)\sigma\left(-T\left(i,\cdot\right)y\right)\right)x(j).$$

(b) For any $i \in \{1,\dots,B\}$ *and* $j \in \{1,\dots,K\}$, *the differentiations of the loss function is given by:*

$$\frac{\partial KL\left[\sigma\left(Cy\right)||\sigma\left(Tx\right)\right]}{\partial C(i,j)} = \left(C\left(i,\cdot\right)y - T\left(i,\cdot\right)x\right)\sigma\left(C\left(i,\cdot\right)y\right)\sigma\left(-C\left(i,\cdot\right)y\right)y(j),$$

where

- $d = d_T$, $(x,y) \in X^T \times Y$ *and* $T = T^T$ *if the domain is target;*
- $d = d_S$, $(x,y) \in X^S \times Y$ *and* $T = T^S$ *if the domain is source.*

Proof: The differentiations can be obtained by using Lemma 1 and Lemma 2 and applying the chain rule for differentiation. □

Preposition 1. (Gradients of the loss function $\mathcal{O}$)

(a) For any $i \in \{1, \ldots, B\}$, *and* $j \in \{1, \ldots, d_T\}$*:*

$$\frac{\partial \mathcal{O}}{\partial T^T(i,j)} = \sum_{(x^T, y^T) \in D_T} x^T(j) \left(\sigma\left(-C(i,\cdot)\, y^T\right) \sigma\left(T^T(i,\cdot)\, x^T\right) - \sigma\left(C(i,\cdot)\, y^T\right) \sigma\left(-T^T(i,\cdot)\, x^T\right) \right),$$

(b) For any $i \in \{1, \ldots, B\}$, *and* $j \in \{1, \ldots, d_S\}$*:*

$$\frac{\partial \mathcal{O}}{\partial T^S(i,j)} = \sum_{(x^S, y^S) \in D_S} x^S(j) \left(\sigma\left(-C(i,\cdot)\, y^S\right) \sigma\left(T^S(i,\cdot)\, x^S\right) - \sigma\left(C(i,\cdot)\, y^S\right) \sigma\left(-T^S(i,\cdot)\, x^S\right) \right),$$

(c) For any $i \in \{1, \ldots, B\}$ *and* $j \in \{1, \ldots, K\}$*:*

$$\begin{aligned}\frac{\partial \mathcal{O}}{\partial C(i,j)} =& \Big[(1+\alpha) + (|D_{Tj}| + |D_{Sj}|)C(i,j) - \alpha(|D_{Tj}| + |D_{Sj}|)C_{\mathit{ref}}(i,j) \\ & -\big(\sum_{(x^T, y^T) \in D_{Tj}} (T^T(i,\cdot)x^T + \sum_{(x^S, y^S) \in D_{Sj}} T^S(i,\cdot)x^S)\big)\Big] \sigma(C(i,j))\sigma(-C_{\mathit{ref}}(i,j)).\end{aligned}$$

where

- D_{Tj} *is the subset of* D_T *of instances* (x^T, y^T) *with* $y^T = y_j \in Y$*;*
- D_{Sj} *is the subset of* D_S *of instances* (x^S, y^S) *with* $y^S = y_j \in Y$*.*

Proof:
(a) The first-order partial derivative of

$$\mathcal{O} = \mathcal{L}(D_T \mid C, T^T) + \mathcal{L}(D_S \mid C, T^S) + \alpha\left(\mathcal{L}(D_T \mid C_{\mathit{ref}}, C) + \mathcal{L}(D_S \mid C_{\mathit{ref}}, C)\right)$$

is equal to that of $\mathcal{L}(D_T \mid T^T, C)$. Thus it is equal to

$$\frac{\partial \mathcal{O}}{\partial T^T(i,j)} = \frac{\partial \mathcal{L}(D_T \mid C, T^T)}{\partial T^T(i,j)} = \frac{\partial}{\partial T^T(i,j)} \sum_{(x^T, y^T) \in D_T} KL\left[\sigma\left(Cy^T\right) \| \sigma\left(T^T x^T\right)\right].$$

Using Lemma 3a, this gives part **(a)** of the preposition. □

(b) Similarly, the partial derivative of $\mathcal{O}$ w.r.t. $T^S(i,j)$ is equal to that of $\mathcal{L}(D_S \mid T^S, C)$. Thus this part of the preposition can be proven as part **(a)**. □

(c) For $\frac{\partial \mathcal{O}}{\partial C(i,j)}$, we use Lemma 3b, which gives the expression:

$$\begin{aligned}&\sum_{(x^T, y^T) \in D_T} \left(C(i,\cdot)\, y^T - T^T(i,\cdot)\, x^T\right) \sigma\left(C(i,\cdot)\, y^T\right) \sigma\left(-C(i,\cdot)\, y^T\right) y^T(j) + \\ &+ \sum_{(x^S, y^S) \in D_S} \left(C(i,\cdot)\, y^S - T^S(i,\cdot)\, x^S\right) \sigma\left(C(i,\cdot)\, y^S\right) \sigma\left(-C(i,\cdot)\, y^S\right) y^S(j) +\end{aligned}$$

$$+\alpha \sum_{(x,y)\in D_T+D_S} \left(C\left(i,\cdot\right)y - C_{ref}\left(i,\cdot\right)y\right)\sigma\left(C\left(i,\cdot\right)y\right)\sigma\left(-C\left(i,\cdot\right)y\right)y(j),$$

where $D_T + D_S$ denotes the concatenation of the datasets D_T and D_S. Also, here it is noted that the factor $y^T(j), y^S(j)$ or $y(j)$ is either 1 (if $y^T(j), y^S(j), y(j) = e_j \in Y$ (i.e. class j is involved)) or zero otherwise. Then also $C(i,\cdot)y^T, C(i,\cdot)y^S$ or $C(i,\cdot)y$ equal $C(i,j)$ and likewise $C_{ref}(i,\cdot)y = C_{ref}(i,j)$. Introducing the sets D_{Tj} and D_{Sj}, as indicated, we obtain that:

$$\frac{\partial \mathcal{O}}{\partial C(i,j)} = \Bigg(\sum_{(x^T,y^T)\in D_{Tj}} \left(C\left(i,j\right) - T^T\left(i,\cdot\right)x^T\right) + \sum_{(x^S,y^S)\in D_{Sj}} \left(C\left(i,j\right) - T^S\left(i,\cdot\right)x^S\right)$$
$$+\alpha \sum_{(x,y)\in D_{Tj}+D_{Sj}} \left(C\left(i,j\right) - C_{ref}\left(i,j\right)\right)\Bigg)\sigma\left(C\left(i,j\right)\right)\sigma\left(-C\left(i,j\right)\right),$$

from which part **(c)** of the proposition follows.□

Preposition 2. (Optimal choice of the mapping C)
For a fixed choice of T^T and T^S, the criterion $\mathcal{O}$ has a unique stationary point, which is the minimum, achieved at:

$$C(i,j) = \frac{\alpha}{1+\alpha}C_{ref}\left(i,j\right) + \frac{1}{1+\alpha}\left(\frac{1}{|D_{Tj}|+|D_{Sj}|}\Big(\sum_{(x^T,y^T)\in D_{Tj}} T^T\left(i,\cdot\right)x^T + \sum_{(x^S,y^S)\in D_{Sj}} T^S\left(i,\cdot\right)x^S\Big)\right)$$

with $i \in \{1,...,B\}$ and $j \in \{1,...,K\}$.

Proof: The fact that $\mathcal{O}$ has a unique stationary point for $\mathcal{C}$ when T^T and T^S are kept fixed, follows from Preposition 1c, which makes that $\frac{\partial \mathcal{O}}{C(i,j)} = 0$, is equivalent to a linear equation for $C(i,j)$. It is a minimum, because $\mathcal{O}$ clearly has a minimum (with a diagonal Hessian w.r.t. C). □

Note that Proposition 2 makes clear what the effect of the regularization parameter α is: it achieves a linear interpolation with weight $\frac{\alpha}{1+\alpha}$ and $\frac{1}{1+\alpha}$ between the *proposed prototype* $C_{ref}\left(i,j\right)$ and the *data-average for class* j:

$$\frac{1}{|D_{Tj}|+|D_{Sj}|}\Big(\sum_{(x^T,y^T)\in D_{Tj}} T^T\left(i,\cdot\right)x^T + \sum_{(x^S,y^S)\in D_{Sj}} T^S\left(i,\cdot\right)x^S\Big),$$

considering target and source data.

7

IHC Classification of Breast Cancer Subtypes Using Bunching.HDA

This chapter is based on the following submission:
Ismailoglu, F., Cavill R., Smirnov, E., Zhou, S., Collins, P. and Peeters, R. (2017). Heterogeneous Domain Adaptation for IHC Classification of Breast Cancer Subtypes.
To appear in IEEE/ACM Transactions on Computational Biology and Bioinformatics.

This chapter considers the problem of classifying breast cancer instances according to their immunohistochemistry (IHC) subtypes (Onitilo et al., 2009). This problem is important, since breast cancer is the most commonly diagnosed cancer type and its subtypes vary greatly in terms of clinical outcomes and survival times (Siegel et al., 2016; Dent et al., 2007). The data we used in this chapter come from two breast-cancer domains that have different feature spaces. The first domain represents instances with protein measurements while the second domain represents instances with gene measurements. Both domains share the same set of class labels; i.e. IHC subtypes.

Solving the IHC classification problem is an open question in bioinformatics, since the protein data and gene data need to be integrated. In this context we note that the main characteristics of the problem such as data heterogeneity and common class labels match well with the scope of the classification problems for the Bunching.HDA algorithm (introduced in the previous chapter). Thus we consider the IHC classification of breast cancer as a heterogeneous domain adaptation problem and employ the Bunching.HDA algorithm. We take the current chapter as an application of the Bunching.HDA algorithm. The experiments we conducted in this chapter show that the Bunching.HDA algorithm solves successfully the IHC classification problem with a substantial accuracy gain compared with baseline classifiers trained on a single domain.

This chapter consists of 7 sections. It begins with the background section 7.1 that emphasizes the importance of breast-cancer classification and motivates the use of the Bunching.HDA algorithm for the IHC classification problem. In Section 7.2, we reveal the potential benefits of domain adaptation for bioinformatics reviewing some prominent applications. In Section 7.3 we provide an insight into the subtypes of breast cancer based on the immunochemistry. In Section 7.4, we explain the process of deriving the protein data and gene data. In section 7.5, we report the experimental results for the Bunching.HDA algorithm and the baseline classifiers. Finally, in Section 7.6, we conclude the chapter.

7.1 Background

Breast cancer is by far the most common cancer type and the leading cause of cancer death in women worldwide with over 1.6 million new cases diagnosed each year (Siegel et al., 2016). Depending on the active mutations in the tumor, breast cancers differ considerably in molecular alterations, cellular composition, and clinical outcomes. This diversity is reflected by its immunohistochemistry (IHC) subtypes. Disclosing these subtypes is very important for diagnosing and treating breast cancer.

In bioinformatics, a good deal of study on breast cancer have been conducted using breast-cancer datasets collected using various measurement technologies (Li et al., 2002; Campiglio et al., 2008). The data collected in one set is usually not sufficient to build an accurate classifier that predicts the breast cancer subtypes. This raises the need for integrating datasets and makes the data integration an important research topic in bioinformatics (Palsson and Zengler, 2010). However, several alternatives thats have been recently proposed in machine learning were not fully exploited in bioinformatics. Some of them are methods from the *domain adaptation* (DA) field that allow using multiple data types in the learning phase (Mei et al., 2011).

In this chapter, we explore the potential of domain adaptation for the problem of classifying breast cancer instances according to their immunohistochemistry (IHC) subtypes (Dent et al., 2007). The data we use are based on two breast-cancer domains that have different feature spaces. The first domain (that we call protein domain) represents instances with protein measurements while the second domain (that we call gene domain) represents instances with gene measurements. The data heterogeneity implies that the IHC classification problem is essentially a heterogeneous domain adaptation (HDA) problem. Since the protein and gene domains share the same set of class labels (i.e. IHC subtypes), the Bunching.HDA algorithm is applicable in this case. We follow two scenarios when applying the algorithm. In the first scenario, we consider the protein domain as target domain and the gene domain as source domain. That is, the protein data represents the distribution from which future instances will come and the gene data is just an auxiliary domain that is used only to improve classification models. In the second scenario, we shift the roles of the protein data and the gene data, meaning that the gene data become target data and the protein data become source data.

Applying the Bunching.HDA algorithm for the IHC classification problem has several benefits. First, the algorithm makes no assumptions regarding the structure of the datasets that it tries to adapt (in contrast with the post popular DAMA approach (Wang and Mahadevan, 2011)). Second, it converges fast. Third, the algorithm allows for a variety of classification rules as it classifier independent. Bunching.HDA.Pr, Bunching.HDA.NN, and any classifier can be trained on the projected data.

These benefits come with a drawback shared with the other domain adaptation methods that are symmetric. Concretely, the features of the common space created by the Bunching.HDA are difficult to interpret and thus it is hard for experts to employ the classification models trained on the projected data for hypothesis generation. Nevertheless, the application of the Bunching.HDA algorithm for the IHC classification problem is one of the first HDA works in the context of bioinformatics.

7.2 Data Integration and Domain Adaptation in Bioinformatics

The data integration and domain adaptation methods have an enormous potential in Bioinformatics. There are many current bioinformatics problems where the same elements have been measured by different technologies, for instance, gene expression measured by microarrays and RNAseq. Mapping these different datasets to a common space, accounting for the different background distributions of measurements and noise with the different technologies might, for instance, allow the reuse of archived microarray data with the more recently acquired RNAseq. The data-integration and domain-adaptation methods may also be applicable to the widespread problem of batch effects in omics data (Leek et al., 2010), where data acquired in one setting is not immediately relatable to data acquired from a similar population in another setting (for instance data processed in different laboratories, at different times or samples collected in different hospitals). Additionally, the data-integration and domain-adaptation methods may prove useful when the same experiments are performed on different species, to relate the measurements between the two studies or in toxicogenomics between omics measurements taken after dosing with similar compounds.

7.2.1 Data Integration Applications

Current data integration methods in bioinformatics tend to come from one of four classes of methods (Cavill et al., 2015): concatenation methods, correlation methods, pathway methods, and multivariate model methods. The concatenation methods make a combined feature space by concatenating the datasets, possibly with block weighting to prevent the over dominance of the larger dataset and then apply standard methods on the concatenated data. An example of this would be the integration of transcriptomic and metabolomic data by Shen et al. (2012) for the classification of glioblastomas. The correlation methods look for correlative relationships between items in different datasets. However, as these methods tend to be descriptive of the datasets, they are rarely used when classification is the aim. The pathway methods map the data onto biological pathways or networks and then analyse these mappings (Cavill et al., 2011). The multivariate model methods generally utilise the PLS family of methods, for instance O2PLS (Trygg and Wold, 2003), to relate the datasets to each other in a linear model. O2PLS is conceptually the most similar to domain adaptation. It is a symmetric model, which represents two datasets in terms of five matrices, the matrix modelling the joint variation, two matrices modelling the unique variation in each dataset, and two residual matrices containing the unmodelled variation for each dataset.

7.2.2 Domain Adaptation Applications

The DA has a few applications in bioinformatics, as it is a relatively new field. Most of the applications are homogeneous domain adaptation (hDA) applications when the domains to be adapted share the same feature space. We are aware of just one heterogeneous domain adaptation (HDA) application when the domains have different feature spaces. Below we first review the hDA applications and then the HDA one.

Ganchev et al. (2011) proposed one of the first applications in the hDA that is based on classification rule adaptation. It first derives classification rules on the data from a medical clinic, which represents a source domain, to predict leukemia or lung cancer among the patients. Then, the rules are transferred to the target domain, the clinic of interest with very few patient's cases. The rules are adapted to reflect the target data using some incremental rule-learning techniques. The application is pretty straightforward, however, it has to be used with a care. Continuous features need to have the same range over the domains so that the discretization process (used for rule formation) results in the same discrete features.

Dahlmeier and Ng (2010) proposed to perform semantic role labeling (SRL) on biomedical articles by adapting an SRL system from the newswire domain. A hDA process was clearly needed because of the lack of a large annotated biomedical corpus that is available in the bioinformatics domain. Three hDA methods were applied: instance weighting (Jiang and Zha, 2007), instance augmenting (Hall, 2004), and instance pruning (Jiang and Zha, 2007). The experiments showed that using hDA it is possible successfully train a SRL system for biomedical articles with less annotation efforts (i.e. costs).

Schweikert et al. (2009) investigated the problem of how to recognize acceptor splice sites on a gene-sequence level in the less studied model organisms such as C.remanei, P.pacificus and D.melanogaster. Due to the small amount of the available data, they considered the problem as a hDA problem. As a source domain they employed the domain of the organism C.elegans as it is a well-studied organism. Six SVM-based hDA methods were applied. The experiments showed the hDA significantly improves models for the domains of C.remanei, P.pacificus, and D.melanogaster compared with models trained on a single data.

To our knowledge, the only work in HDA related to bioinformatics has been recently presented by Breckels et al. (2016). The task was to predict subcellular localization of protein sequences. In addition to the GO terms, a wide variety of auxiliary sources such as Human Protein Atlas and immunocytochemistry data were exploited. For this purpose, the authors adapted to HDA two methods, k-NN TL and SVM TL, originally proposed in (Wu and Dietterich, 2004) for hDA. Although, the experimental results look promising, applying the methods can be difficult, since the query (test) instances require a view (features values) from the source domains in addition to the target ones. This differs significantly from the Bunching.HDA algorithm which assumes a view from one domain only.

7.3 IHC Subtypes of Breast Cancer

There are four common subtypes of breast cancer, these correlate with groups defined by three proteins ER, PR, and HER2 (Onitilo et al., 2009): HER2 overexpressing (HER2 positive), triple negative (basal-like), luminal A, and luminal B. HER2 overexpressing breast cancer is characterized by the presence of HER2. In triple negative breast cancer, however, all three proteins are absent. The shortest survival times are observed in patients who have HER2 positive or triple negative subtypes. In fact it was reported that triple negative cancers are more aggressive and more resistant to treatment (Dent et al., 2007). In contrast to the other two subtypes, luminal A and luminal B were first identified using whole genome gene expression data from microarrays (Perou et al., 2000) and do not directly relate to these three proteins. However, luminal A subtype tends to have either a high expression of oestrogen receptor (ER) or progesterone receptor (PR) and (HER2) is negative. Luminal B is similar to luminal A in terms of the existence of ER and PR markers, yet it differs with respect to HER2 which tends to be positive in this case. When it comes to the clinical outcomes luminal A is more favourable than luminal B, but patients with luminal A or B tumors respond similarly to endocrine therapies (Prat et al., 2012).

The three proteins ER, PR, and HER2 which characterize these subtypes can be measured through immunohistochemistry and thus these classes are assigned in this way in a clinical setting. However, we know that the protein and gene measurements taken from these tissues and measured through high-throughput technologies should allow the prediction of these immunohistochemical markers, and therefore also the subtype classes identified above.

The classification of breast cancer samples, particularly from gene expression data, is a well studied problem. There have been many proposed gene signatures (Prat et al., 2011; Sotiriou and Pusztai, 2009; Weigelt et al., 2010). However the overlap between proposed gene signatures, particularly those predicting prognosis, was often very small (Weigelt et al., 2011). However, many of these gene signatures focussed on prognosis rather than classification of subtypes as we do. But, as prognosis strongly relates to these subtypes, the genes/proteins distinguishing these subtypes feature will heavily in these signatures of prognosis.

Table 7.1: The subtypes of breast cancer based on IHC biomarkers.

ER	PR	HER2	IHC Subclass
1	1	0	Luminal-A
0	1	0	
1	0	0	
1	1	1	Luminal-B
0	1	1	
1	0	1	
0	0	0	Triple Negative
0	0	1	Her2 Over.

7.4 Data Derivation and Pre-processing

This section explains the heterogeneous data for the IHC classification problem. First, it reveals how the data was derived and then how it was preprocessed.

7.4.1 Derivation

The data used in this study were taken from The Cancer Genome Atlas (TCGA) (Weinstein et al., 2013). This is a large repository of datasets from many types of tumors. In total there are over 11,000 samples in the database, and breast cancer is the most represented cancer type with over 1000 samples stored. Having removed protein and gene data from samples which were missing data on the IHC biomarker(s), we had 578 samples with protein data available and 419 with gene expression data measured through microarrays.

7.4.2 Pre-processing

Originally, there were 284 protein features in the protein data. However some proteins in the data were isoforms and in this dataset we see only a single non-zero entry per patient amongst the isoforms of a single protein. This led us to merge the isoform and to construct a single column for a set of isoforms. As a result we ended up with 211 distinct protein-related features.

The gene dataset contained 17815 features per sample. Although the employed domain adaptation algorithm Bunching.HDA can handle such big data in theory, from the practical point of view, execution times restrict using this amount of data. Therefore, we reduced the number of genes using a subset selection method Support Vector Machine–Recursive Feature Elimination (SVM-RFE) for multiclass problems (Zhou and Tuck, 2007).

MSVM-RFE is an iterative backward-elimination method. In each iteration it first builds K SVM classifiers using the One-vs-All class decomposition scheme (the number K of classes is equal to 4 in our case). This results in a matrix W with entries $w(i, j)$ each corresponding the coefficient of the i-th SVM classifier for the j-th feature ($i \in \{1, ..., K\}$ and $j \in \{1, ..., d\}$ where d is the number of features in the current iteration). The features j are then ranked with respect to the ranking criterion $\sum_{r=1}^{K} w_{rj}^2$. Finally, the feature, i.e. the gene in our case, with the smallest ranking criterion is eliminated. We applied the MSVM-RFE method on the gene dataset. We stopped the process of backward-elimination when the coefficients became relatively large. As a result we obtained a subset of 1069 genes (features) that we used in our experiments to represent the gene data.

In the protein and the gene data, the patients are labeled according to their immunohistochemically measured ER, PR, and HER2 status resulting in 8 potential groups. We used the correlation between these markers and the four breast cancer subtypes: luminal A, luminal B, HER2 overexpressing, and triple negative as described in the introduction, to aggregate these groups into clinically meaningful classes. This resulted in the class distributions for the protein and the gene data shown in Table 7.2.

Table 7.2: Class Frequencies in the Protein and the Gene Datasets.

Dataset	luminal A	luminal B	Triple Neg.	HER2 Over.
Protein	349	98	95	36
Gene	257	78	60	24

7.5 Experiments

This section presents our experiments, results, and initial conclusions for the IHC classification problem. The data we employed for this problem is the heterogeneous data which derivation is provided in the previous section. The experiments involve the Bunching.HDA algorithm together with several associated classification rules. The generalization performance of these rules is compared with that of baseline classifiers trained on one-domain data.

7.5.1 Settings of the Bunching.HDA Algorithm and Classification Rules

The Bunching.HDA algorithm was set up as follows. The reference coding matrix C_{ref} was set to the coding matrix of the eECOC class decomposition scheme. The regularization parameter α took values from the set $\{0.04, 0.02, 1, 5, 25\}$. The iteration parameters $maxIterIn$ and $maxIterOut$ were set to 10 and 50, respectively. The parameter of the backtracking method β of the function IMPROVE-T was set to 0.5.

We note that the reference coding matrix C_{ref} was set to the eECOC coding matrix for three reasons:

(1) the number of classes (K) is 4; i.e. is small, which results a 4×7 C_{ref} matrix[1]. These matrix sizes makes the Bunching.HDA computations feasible.

(2) the number of latent features in the common code space $\mathcal{C}$ created by the Bunching.HDA algorithm is maximized and it is equal to 7. The number may not be big, but is the largest possible compared with the cases when we use any other class decomposition scheme.

(3) the distance between the code words in C_{ref} of different class labels is maximized. In fact, each code word is 4 bits away from the other code words in our case. This helps instances of different breast cancer subtypes to be projected to different locations in the common space $\mathcal{C}$.

Four possible classification rules associated with the Bunching.HDA algorithm were applied: the Bunching.HDA.Pr rule, the Bunching.HDA.NN rule, the Bunching.HDA.RF rule, and the Bunching.HDA.NB rule. The first two rules were defined in Section 6.4.2. The Bunching.HDA.RF rule is a Random Forest classifier (with 30 random decision trees) (Breiman, 2001) trained on the projected data in the common code space. The Bunching.HDA.NB rule is a Naive Bayes classifier (Domingos and Pazzani, 1997) trained on the projected data in the common code space.

7.5.2 Settings of the Baseline Classifiers

We compared the accuracy of the four Bunching.HDA classification rules against four baseline classifiers. The baseline classifiers were trained on one domain data only[2] and are specified as follows:

- kNN.T is a first nearest neighbor classifier (Mitchell, 1997).

[1]In general, the eECOC coding matrix is a $K \times 2^{K-1} - 1$ matrix.
[2]This is indicated with postfix T.

- Bunching.T is a Bunching algorithm (Dekel and Singer, 2002) with an eECOC reference coding matrix C_{ref}, regularization parameter α equal to 1, and the number of iteration equal to 100.

- RF.T is a random forest classifier with 30 random decision trees. For the protein (gene) data each tree is based on 15 (32) randomly chosen features. Here 15 (32) is approximately the square root of 211 (1069), the number of protein-related (gene-related) features.

- NB.T is a Naive Bayes classifier where the features are assumed to follow normal distribution for each class.

The baseline classifiers were used to show whether the heterogeneous domain adaptation provided by the Bunching.HDA algorithm improves the accuracy when considering the results yielded by the classifiers trained on a single data. To make the comparisons fair the comparison was made between the Bunching rules and similar baseline classifiers. More precisely, the Bunching.HDA.Pr rule was compared with Bunching.T, the Bunching.HDA.NN rule was compared with kNN.T, the Bunching.HDA.RF rule was compared with RF.T, and the Bunching.HDA.NB rule was compared with NB.T.

7.5.3 Evaluation Settings

We used the gene and the protein datasets in both target and source domain roles. That is, each was used as the target and the source data in turn. The accuracy of the baseline classifiers was estimated using the conventional 5-fold cross validation method applied on one domain. The accuracy of the Bunching.HDA classification rules was estimated using a different 5-fold cross validation method. This 5-fold cross validation was organized as follows: for each $k \in \{1, \dots, 5\}$ the training data consisted of four randomly chosen folds of the target data and the same source data both projected in the common code space $\mathcal{C}$. The remaining fifth fold of the target data also projected in $\mathcal{C}$ was used for testing. In this way the estimated accuracy indicated the influence of the heterogeneous domain adaptation provided by the Bunching.HDA algorithm.

7.5.4 Experimental Results

Table 7.3 and Table 7.4 show accuracy of the Bunching.HDA classification rules for six values of the regularization parameter α from the set $\{0.04, 0.02, 1, 5, 25\}$. The highest accuracy for each rule is achieved for different value of α compared with other rules. This can be explained by the different nature (inductive bias) of the rules (Mitchell, 1997).

Table 7.3: Accuracy (in percent) of the Bunching.HDA classification rules for different values of the regularization parameter α. The target data is the protein data and the source data is the gene data. The best results are given in bold.

Method	0.01	0.1	1	10	50	100
Bunching.HDA.NN	66.96	68.14	69.03	**69.26**	69.19	68.89
Bunching.HDA.Pr	73.20	**74.01**	**74.01**	73.53	73.65	73.58
Bunching.HDA.RF	**74.38**	73.01	70.58	73.52	73.52	73.86
Bunching.HDA.NB	74.73	74.57	74.74	**75.43**	74.56	74.90

Table 7.4: Accuracy (in percent) of the Bunching.HDA classification rules for different values of the regularization parameter α. The target data is the gene data and the source data is the protein data. The best results are given in bold.

Method	0.01	0.1	1	10	50	100
Bunching.HDA.NN	67.51	70.40	74.70	75.41	73.51	**75.65**
Bunching.HDA.Pr	72.55	74.69	**75.89**	74.28	74.38	74.27
Bunching.HDA.RF	71.60	78.04	74.46	76.37	76.37	**78.99**
Bunching.HDA.NB	73.50	72.78	77.33	77.33	77.79	**78.51**

In addition to the conventional accuracy shown in Tables 7.3 and 7.4, we also evaluated the generalization performance of all the classifiers in terms of *balanced accuracy*, as both protein and gene data suffer from imbalanced classes (See Table 7.2). Concretely, balanced accuracy is the arithmetic average of the class specific accuracies thus is invariant to class distributions and is often preferable to the conventional accuracy in the presence of imbalanced data (Brodersen et al., 2009). Formally, given a confusion matrix M the balanced accuracy λ in percent is given by:

$$\lambda = \frac{1}{K}\sum_{j=1}^{K}\frac{M(j,j)}{\sum_{i=1}^{K} M(i,j)} \times 100, \tag{7.1}$$

where K is the number of classes and the rows (resp. columns) of M correspond to the actual (resp. predicted) classes.

Table 7.5 and Table 7.6 show balanced accuracy of the Bunching.HDA classification rules for the same six values of the regularization parameter α. Again, the highest balanced accuracy for each rule is achieved for different value of α compared with other rules. Comparing the results in Tables 7.3 and 7.4 with those in Tables 7.5 and 7.6, we note that the best performance values of α are different for the conventional accuracy and balance accuracy. Thus the choice of α also depends on the evaluation method.

Table 7.5: Balanced accuracy (in percent) of the Bunching.HDA classification rules for different values of the regularization parameter α. The target data is the protein data and the source data is the gene data. The best results are given in bold.

Method	0.01	0.1	1	10	50	100
Bunching.HDA.NN	58.29	58.46	**61.99**	61.30	60.61	61.06
Bunching.HDA.Pr	56.5	**64.31**	57.63	55.94	57.41	55.27
Bunching.HDA.RF	**64.58**	64.23	60.43	63.64	63.32	62.96
Bunching.HDA.NB	63.73	67.70	**68.20**	68.14	67.57	67.62

Table 7.6: Balanced accuracy (in percent) of the Bunching.HDA classification rules for different values of the regularization parameter α. The target data is the gene data and the source data is the protein data. The best results are given in bold.

Method	0.01	0.1	1	10	50	100
Bunching.HDA.NN	56.96	59.80	61.51	**67.66**	64.29	63.60
Bunching.HDA.Pr	61.42	62.41	**63.12**	62.01	61.49	62.70
Bunching.HDA.RF	55.95	64.85	62.17	62.38	62.91	**66.14**
Bunching.HDA.NB	60.36	54.39	58.2	57.63	57.81	**63.61**

Table 7.7 shows the (conventional) accuracies of the Bunching.HDA.Pr rule, Bunching.HDA.NN rule, Bunching.HDA.RF rule, and Bunching.HDA.NB rule together with those of the baseline classifiers. Each Bunching.HDA rule was compared with a corresponded baseline classifier. The improvement in the breast cancer classification ranges from 2% to 12% and these improvements were found to be statistically significant based on two-tailored t-test at 0.05. Thus, our main conclusion is that adapting the protein data and the gene data together results in a more accurate classification of the IHC breast cancer subtypes compared with that of the baseline classifiers trained on one domain.

Table 7.7: Accuracies (in percent) for the protein data and gene data. Bold numbers indicate statistically better results obtained by a paired t-test at 0.05 significance level within groups given with double bars.

Target Domain	Source Domain	kNN.T	Bunching.HDA.NN	bunching.T	Bunching.HDA.Pr
Protein	Gene	62.39	**69.26**	62.98	**74.01**
Gene	Protein	72.16	**75.65**	73.87	**75.89**

Target Domain	Source Domain	RF.T	Bunching.HDA.RF	NB.T	Bunching.HDA.NB
Protein	Gene	71.42	**74.38**	63.25	**75.43**
Gene	Protein	76.42	**78.99**	77.01	**78.51**

The balanced accuracy results are parallel to the conventional accuracy results, as shown in Table 7.8. Indeed, it is fair to claim that adapting the protein data and gene data together results in class balanced classifiers that are more accurate for each breast cancer subtype, especially when the protein data is the target domain. In this case, the balanced accuracy is improved by up to 8% over the respective baseline classifier. In addition, we note that the Bunching.HDA classification rules do not impose extra problems when handling class-unbalanced data compared to the baseline methods. This can be seen by the fact that the reduction in the balance accuracy performance is more or less the same for the Bunchihg.HDA rules and the baseline classifiers.

Table 7.8: Balanced accuracies (in percent) for the protein data and gene data. Bold numbers indicate statistically better results obtained by a paired t-test at 0.05 significance level given with double bars.

Target Domain	Source Domain	kNN.T	Bunching.HDA.NN	bunching.T	Bunching.HDA.Pr
Protein	Gene	55.26	**61.99**	61.87	**64.31**
Gene	Protein	65.14	**67.66**	64.49	63.12

Target Domain	Source Domain	RF.T	Bunching.HDA.RF	NB.T	Bunching.HDA.NB
Protein	Gene	56.23	64.58	61.49	68.20
Gene	Protein	60.63	66.14	69.85	63.61

Figure 7.1 shows four projections in the common code space $\mathcal{C}$ where in each case instances of one breast cancer subtype are shown using the dimensions provided by the Principle Component Analysis (PCA). Clearly, for all breast cancer subtypes, the instances of the protein data and those of the gene data are mapped nearby in the common space, as desired. Additionally, this may suggest that the Bunching.HDA algorithm can be used to find out similar gene expression patterns for a given protein expression pattern and vice versa. Considering the centers of the subtypes, Figure 7.1 reveals that for both the protein and the gene data breast cancers of type luminal A and luminal B are near to each other which is to be expected given the ambiguity between the original definitions of these subtypes based on the three immunohistochemical markers. Also, we observe that the triple negative instances are far apart from the luminal A and luminal B instances, as expected.

7.6 Conclusion

This chapter showed that the problem of classification of IHC subtypes of breast cancer can be viewed as a HDA problem. There are at least two different feature space domains, the protein domain and gene domain, and they share the same class labels. The latter feature makes possible the use of the Bunching.HDA algorithm among other HDA methods. The algorithm was chosen because: it does not make assumptions on the data structure, it can handle class imbalanced problems, and it allows several classification rules. The algorithm results in an accuracy gain of the Bunching.HDA rules when applied for the IHC classification. It was shown experimentally that this gain is statistically significant and ranged from 2% to 12% compared with the baseline classifiers. Thus, our main conclusion is that adapting the protein data and the gene data together results in a more accurate classification of the IHC breast cancer subtypes compared with those trained on one domain..

The Bunching.HDA benefits come with a drawback shared with other symmetric HDA methods: the Bunching.HDA features in the common code space are difficult to interpret and thus it is hard for experts to employ the classification models trained on the projected data for hypothesis

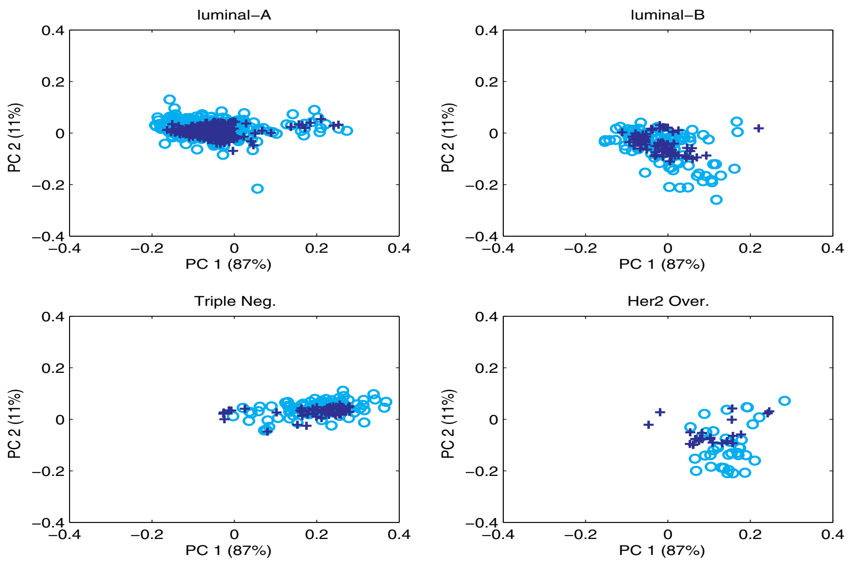


Figure 7.1: The protein data and gene data in the common code space given with two PCA dimension PC1 and PC2 preserving 98% of the data variance. In each subplot the protein samples are marked with circles and the gene samples are marked with pluses.

generation. Nevertheless, the application of the Bunching.HDA algorithm for the IHC classification problem is one of the first HDA works in bioinformatics. Therefore, this chapter shows the promise of using HDA methods in bioinformatics in improving generalization performance when multiple omics datasets are available.

8 Conclusions

This is the final chapter that concludes this thesis. We first answer the research questions formulated in Chapter 1, in view of our findings in Chapters 3-7. Then, we sketch future work by providing possible research directions.

8.1 Answers to the Research Questions

In Chapter 1, we argued that further development of class decomposition schemes can proceed along two research lines. Below we first provide answers to the two research questions which fall within the first research line, and then we discuss our answers to the other two research questions, which fall within the second research line.

The first research line was related to improving class decomposition schemes for boosting the generalization performance. The research questions formulated in this context are as follows:

(Q1) How can the problem of difficult binary classification problems be solved?

(Q2) How can the problem of error dependency of binary classifiers be solved?

These research questions were both addressed in Chapters 3 and 4. We first provide the answers from Chapter 3 and then from Chapter 4.

In Chapter 3 we proposed instance decomposition schemes (IDS) as an alternative to class decomposition schemes. An instance decomposition scheme consists of several instance partitions. Every instance has a code word of which the bits indicate the set of the instance in each partition. Instance classification assumes two instance decomposition schemes: encoding and decoding. The encoding IDS is used to train binary classifiers and to encode any query instance. The decoding IDS is used to decode the class of an instance.

We proposed to initialize the encoding IDS according to some standard class decomposition scheme, and to learn the decoding IDS from the data through the encoding IDS. In this case we showed that the simultaneous and non-simultaneous errors of the binary classifiers can be compensated for, during the classification process. We note that difficult classification problems can cause both simultaneous errors and non-simultaneous errors, while error dependency of the binary classifiers causes essentially simultaneous errors. Thus, we conclude that the instance decomposition schemes are capable of handling difficult classification problems and error dependency of the binary classifiers.

In Chapter 4 we proposed two weighted decoding algorithms to handle difficult binary classification problems from the perspective of weighting the binary classifiers. First we proposed the Fractional Programming Weighted decoding (FP.Weighted decoding) algorithm. The FP.Weighted decoding algorithm guarantees that the query instance will be far from the code word of its most confused class, while it will be close to the code word of its true class in the code space. Here the most confused class for an instance implies the class which is not the true class for the instance but its code word is nearest to the instance in the code space. In doing so, the FP.Weighted decoding algorithm credits larger weights to the binary classifiers that are more accurate; while it credits relatively smaller weights to the binary classifiers that are less accurate. Thus the binary classifiers that correspond to difficult binary classification problems receive small weights. Second, we proposed the Bipartite Graph Partitioning Weighted decoding (BGP.Weighted decoding) algorithm. The weight matrix produced by the BGP.Weighted decoding algorithm is learned by estimating how difficult the binary classification problems are. By contrast, the existing weighted decoding algorithms produce the weight matrix based on the generalization performances of the built binary classifiers , which makes the resulting weight matrix dependent on the binary classifiers employed. For the BGP.Weighted decoding, the weight matrix is invariant to the employed binary classifiers. Therefore, recomputing of the weight matrix is unnecessary in case a different binary classification algorithm is used.

The second research line of this thesis was related to extending class decomposition schemes for new application fields. The first research question formulated in this context is as follows:

(Q3) How to improve the computational performance of class decomposition schemes for reliable classification?

This research question was addressed in Chapter 5. There we proposed the mean-based conformal class decomposition machines (MCCD machines) and the Poisson conformal class decomposition machines (PCCD machines). We showed that the MCCD machines and PCCD machines employ a different approach when computing the aggregated nonconformity scores. They first compute all possible binary nonconformity scores and then, for any new query instance and any class, they compute the aggregated nonconformity score using the pre-computed binary nonconformity scores. This contrasts with the existing conformal implementations based on class decomposition schemes: they recompute all the binary nonconformity scores for any new aggregated nonconformity score – which makes them slow. Thus, we conclude that the MCCD machines and PCCD machines are computationally more efficient than the existing conformal implementations based on class decomposition schemes.

In addition, we showed that:

- The MCCD machines and PCCD machines employ aggregating nonconformity functions that do not depend on the type of the class decomposition scheme employed. Thus, they can be applied for any class decomposition scheme and can be viewed as generalizations of the conformal implementations based on class decomposition schemes that were proposed earlier.

- The MCCD machines and PCCD machines differ in the way that they correct errors of the binary classifiers. Both machines employ loss-based error-correcting mechanisms through their aggregating non-conformity functions. However, the mechanism of the PCCD machines is related to the error-correcting mechanism applicable for discrete binary classifiers. Its nonconformity function produces an estimate of the probability that the errors of the binary classifiers cannot be corrected. It was demonstrated that, when the average error correlation between the binary classifiers is negative, this function allows generating class regions (class sets) with low error and high efficiency.

In conclusion, we state that the MCCD machines and PCCD machines are valid and efficient classifiers for reliable classification. For most of the experiments their efficiency is superior to that of the standard conformal classifiers, which can be explained by the error correction mechanism.

The second research question within the second research line is the last question formulated in the context of this thesis. It reads as follows:

(Q4) How to apply class decomposition schemes for heterogeneous domain adaptation?

This research question was addressed in Chapter 6. In this Chapter we proposed the Bunching.HDA algorithm as a class decomposition algorithm for classification problems in the context of heterogeneous domain adaptation. For these problems the target and source domains have different feature spaces but share the same class labels. In this case, the code space associated with any class decomposition scheme used is the same for the domains and thus it can be interpreted as a common latent feature space where the target and source data can be projected. The Bunching.HDA algorithm is based on this viewpoint and operates as follows. First, it builds the encoding functions (mappings) for the target data, source data, and class labels. Then it employs these functions to project all the data and class labels in the common code space. In doing so, the Bunching.HDA algorithm does not make any assumption on the underlying structure of the target and source domains. It operates by using only common class label information as a domain correspondence.

The Bunching.HDA algorithm allows two embedded classification rules: the decoding rule of the class decomposition schemes and the decoding rule of the instance decomposition schemes. In addition, one can build any type of classifier in the common space created by the Bunching.HDA algorithm. Thus, Bunching.HDA can be viewed as a classifier independent algorithm.

The experiments showed that the the two classification rules built using the Bunching.HDA are accurate on three real-world domain adaptation problems and they are capable of outperforming standard domain adaptation methods. Thus, since the Bunching.HDA algorithm is a class-decomposition algorithm, we conclude that class decomposition schemes can be applied for heterogeneous domain adaptation with a significant accuracy gain.

Chapter 7 is a follow-up of Chapter 6. In Chapter 7 we studied the problem of classifying breast cancer instances according to their immunohistochemistry (IHC) subtypes. The data for this problem come from two breast-cancer domains that have different feature spaces. The first domain represents instances with protein measurements while the second domain represents instances with gene measurements. Both domains share the same set of class labels, i.e., IHC subtypes. We showed that the IHC classification problem can be viewed as a problem of heterogeneous domain adaptation and one can achieve more accurate classification using Bunching.HDA. It was shown experimentally that the accuracy gain is statistically significant and ranged from 2% to 12% compared with the baseline classifiers that were trained using either the gene data or the protein data only. Thus, our main conclusion is that adapting the protein data and the gene data together results in a more accurate classification of the breast cancer subtypes.

8.2 Future Research

We forsee several directions for future research. We first provide those that are specific to the decomposition schemes and algorithms proposed in this thesis. Then, we provide more general research directions related to the whole field of decomposition schemes.

The instance-based decomposition schemes introduced in Chapter 3 employ a classification (class decoding) rule which is essentially a first nearest-neighbor classifier (Mitchell, 1997). Therefore, two research directions are possible: (1) to reduce the computational complexity and (2) to improve the generalization performance. To reduce computational complexity we need to reduce the storage; i.e., the number of rows and the number of columns in the decoding matrices. To reduce the number of rows we propose to employ several techniques from edited nearest neighbor rules (Marchiori, 2008). Multi-variate feature selection (Guyon and Elisseeff, 2003) can be used to reduce the number of columns. To improve the generalization performance of the first nearest-neighbor classifier, distances in the code space can be adjusted for the data. This can be achieved by employing techniques from metric learning (Kulis, 2012).

The FP.Weighted decoding algorithm and the BGP.Weighted decoding algorithm introduced in Chapter 4, used standard coding matrices, i.e., coding matrices of exhaustive type (eECOC) and random coding matrices (rECOC). It is worth trying to use these algorithms with some more advanced coding matrices that are learned from the data at hand, such as DECOC (Pujol et al., 2006), or Spectral ECOC (Zhang et al., 2009). As a result one expects to better solve multiclass classification problems. Also, we only used off-the-shelf UCI datasets during the experiments in Chapter 4. However the use of FP.Weighted decoding and that of BGP.Weighted decoding for real-world applications are also an open area of research.

The conformal class decomposition machines proposed in Chapter 5 can be optimized further. In the PCCD machines we employed the nonconformity function that estimates the probability estimation that the errors of the binary classifiers cannot be corrected. The function is based on the Poisson binomial distribution. However, this distribution is based on the assumption that the binary experiments (in our case the binary classifiers) are independent. This assumption is strong and in practice can be violated. For this reason, it is important to develop new nonconformity functions that can approximate the Poisson binomial distribution for the case of dependent binary experiments. This is expected to result in conformal class decomposition machines that will be valid (by construction) and much more efficient; i.e., they will produce smaller non-empty class regions.

The Bunching.HDA algorithm proposed in Chapter 6 is a domain adaptation algorithm that projects target data and source data to a common latent space. Recall that in Chapter 7 we built naive Bayes and random forest classifiers in the common space created by the Bunching.HDA algorithm. These classifiers treat the source instances and the target instances in the common space equally. Alternatively, one can consider building a classifier such as TrAdaBoost (Dai et al., 2007) that is biased towards the target instances. This may provide a better classification.

In Chapter 7 we showed that adapting the protein and gene domains allows for a more accurate classification for the problem of the IHC breast cancer subtypes. This medical problem is just one example of classification problems based on data coming from domains with different feature spaces. Other examples can be found, e.g., in education (predicting student drop-out by adapting student performance data and demographic data), marketing (predicting customer purchase power by adapting customer market-basket data and geolocation data), sociology (predicting polls by adapting social-economic data and education data of the voters), etc. These examples show that we are just at the beginning of the new era of adapting domains.

If we take a more general look on further developing class decomposition schemes, we may view this research from the perspective of big data (Mayer-Schnberger and Cukier, 2014). One of the main streams of research in this field is related to data velocity. In classification, data velocity implies two problems for decomposition schemes:

(1) the instances can arrive and depart over time, and

(2) the classes can arrive and depart over time.

The decomposition schemes can handle the first problem by employing online learning algorithms (Hoi et al., 2014) which track and adapt to the data distributions. The second problem, however, is still an open problem. Solving this problem assumes developing new instance and class decomposition schemes that can adapt the coding matrices to the number of classes. The adaptation process requires smart modifications, since increasing (decreasing) the number of classes increases (decreases) both the computational complexity and the generalization performance, and, thus, a trade-off needs to be sought.

References

Ahlberg, E., Spjuth, O., Hasselgren, C., and Carlsson, L. (2015). Interpretation of conformal prediction classification models. In *3rd International Symposium on SLDS*, pages 323–334.

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.

Bache, K. and Lichman, M. (2013). Uci machine learning repository.

Balasubramanian, V., Ho, S.-S., and Vovk, V. (2014). *Conformal Prediction for Reliable Machine Learning*. Morgan Kaufmann, 1 edition.

Banerjee, B. and Stone, P. (2007). General game learning using knowledge transfer. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 672–677.

Banfield, R. E., Hall, L. O., and Bowyer, K. W. (2004). *A Comparison of Ensemble Creation Techniques*, pages 223–232. Springer International Publishing.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *Proceedings of ECCV*, LNCS, pages 404 – 417. Springer.

Blitzer, J., Dredze, M., and Pereira, F. (2007). Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of ACL*, pages 187–205.

Breckels, L. M., Holden, S. B., Wojnar, D., Mulvey, C. M., Christoforou, A., and Groen, A. (2016). Learning from heterogeneous data sources: An application in spatial proteomics. *PLoS Computational Biology*, 12(5).

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5 – 32.

Brodersen, K. H., Ong, C. S., Stephan, K. E., and Buhmann, J. M. (2009). The balanced accuracy and its posterior distribution. In *In Proceedings of ICPR*, pages 3121–3124.

Calafiore, G. and Ghaoui, L. E. (2014). *Optimization Models*. Cambridge Press, 1 edition.

Campiglio, M., Mnard, S., Palazzo, J. P., and Rosenberg, A. (2008). Microrna gene expression deregulation in human breast cancer. *Cancer Research*, 65(16):7065–7070.

Cavill, R., Jennen, D., and Briede, J. J. (2015). Transcriptomic and metabolomic data integration. *Briefings in Bioinformatics*, pages 1–11.

Cavill, R., Kamburov, A., Ellis, J. K., Athersuch, T. J., and Blagrove, M. S. C. (2011). Consensus-phenotype integration of transcriptomic and metabolomic data implies a role for metabolism in the chemosensitivity of tumour cells. *PLoS Computational Biology*, pages 7–12.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.

Cohen, W. W. and Singer, Y. (1996). Context-sensitive learning methods for text categorization. In *Proc.of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 307–315.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machiene Learning Journal*, 20:273–297.

Csiszar, I. and Tusnady, G. (1984). Information geometry and alternating minimization procedures. *Statistics and Decisions*, 1:205–237.

Dahlmeier, D. and Ng, H. T. (2010). Domain adaptation for semantic role labeling in the biomedical domain. *Journal Bioinformatics*, 26:1098–1104.

Dai, W., Yang, Q., and Xue, G.-R. (2007). Boosting for transfer learning. In *In Proc. of the 24th International Conference on Machine Learning*, pages 193–438.

Dekel, O. and Singer, Y. (2002). Multiclass learning by probabilistic embeddings. In *Advances in Neural Information Processing Systems 15*, pages 945–952.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Dent, R., Trudeau, M., Pritchard, K. I., Hanna, W. M., Kahn, H. K., Sawka, C. A., and Lickley, L. A. (2007). Triple-negative breast cancer: Clinical features and patterns of recurrence. *Clinical Cancer Research*, 13:4429–4434.

Devetyarov, D., Nouretdinov, I., Burford, B., Camuzeaux, S., and Gentry-Maharaj, A. (2012). Conformal predictors in early diagnostics of ovarian and breast cancers. *Progress in Artificial Intelligence*, 1:245–257.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Domeniconi, C., Gunopulos, D., Ma, S., Yan, B., Al-Razgan, M., and Papadopoulos, D. (2007). Locally adaptive metrics for clustering high dimensional data. *Journal of Data Mining and Knowledge Discovery*, 14:63–67.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2):103 – 130.

Duan, L., Xu, D., and Tsang, I. W. (2012). Learning with augmented features for heterogeneous domain adaptation. In *Proceedings of Internatiobal Conference on Machine Learning*.

Duan, L., Xu, D., Tsang, I. W., and Luo, J. (2010). Visual event recognition in videos by learning from web data. In *Proceedings of Computer Vision and Pattern Recognition*, pages 1959 – 1966.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley Press, 2 edition.

Escalera, S. and Pujol, O. (2006). Ecoc-one: A novel coding and decoding strategy. In *18th International Conference on Pattern Recognition (ICPR'06), Hong Kong*, pages 578–581.

Escalera, S., Pujol, O., and Radeva, P. (2008). Loss-weighted decoding for error-correcting output codes. In *International Conference on Computer Vision Theory and Applications*, pages 145–155.

Escalera, S., Pujol, O., and Radeva, P. (2010). Error-correcting ouput codes library. *Journal of Machine Learning Research*, 11:661–664.

Fernandez, M. and Williams, S. (2010). Closed-form expression for the poisson-binomial probability density function. *IEEE Transactions on Aerospace and Electronic Systems*, 46:803–817.

Flach, P. (2012). *Machine Learning :The Art and Science of Algorithms that Make Sense of Data*. Cambridge Press, 1 edition.

Furnkranz, J. and Park, S.-H. (2012). Error-correcting output codes as a transformation from multi-class to multi-label prediction. In *Discovery Science*, pages 254–267.

Ganchev, P., Malehorn, D., Bigbee, W. L., and Gopalakrishnana, V. (2011). Transfer learning of classification rules for biomarker discovery and from molecular profiling studies. *Journal of Biomedical Informatics*, 44:17–23.

Gong, B., Shi, Y., Sha, F., and Grauman, K. (2012). Geodesic flow kernel for unsupervised domain adaptationl. In *Procedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gugat, M. (1998). Prox-regularization methods for generalized fractional programming. *Journal of Optimization Theory and Applications*, 99:691–722.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Hall, D. I. (2004). Improving svm accuracy by training on auxiliary data sources. In *Proceedings of ACL*, pages 256–263.

Hatemi, N. (2012). Thinned-ecoc ensemble based on sequential code shrinking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(1):936–947.

Ho, T. K. and Basu, M. (2000). Measuring the complexity of classification problems. In *International Conference on Paatern Recognition*, pages 2043–2047.

Hoi, S. C., Wang, J., and Zhao, P. (2014). Libol: A library for online learning algorithms. *The Journal of Machine Learning Research*, 15(3):495–499.

Hong, Y. (2013). On computing the distribution function for the poisson binomial distribution. *Computational Statistics and Data Analysis*, 59:41–51.

Hsu, C.-W. and Chih-Jen, L. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

Japkowicz, N. and Shah, M. (2014). *Evaluating Learning Algorithms. A Classification Perspective*. Cambridge Press, 1 edition.

Jiang, J. and Zha, i. C. (2007). Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007)*, pages 264–271.

Kulis, B. (2012). Distance metric learning for large margin nearest neighbor classification. *Foundations and Trends in Machine Learning*, 5(4):287–364.

Kulis, B., Saenko, K., and Trevor, D. (2011). What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Procedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1785–1792.

Kuncheva, L. I. and Whitaker, C. J. (2003a). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–2007.

Kuncheva, L. I. and Whitaker, C. J. (2003b). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207.

le Cessie, S. and van Houwelingen, J. (1992). Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201.

Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., and Langmead, B. (2010). Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11:733–739.

Li, J., Zhang, Z., Rosenzweig, J., Wang, Y. Y., and Chan, D. W. (2002). Proteomics and bioinformatics approaches for identification of serum biomarkers to detect breast cancer. *Clinical Chemistry*, 48(8):1296–1304.

Liu, M., Zhang, D., Chen, S., and Xue, H. (2015). Joint binary classifier learning for ecoc-based multiclass classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99:1399–1404.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. Journal of Comp. Vision*, pages 91–110.

Marchiori, E. (2008). Hit miss networks with applications to instance selection. *Journal of Machine Learning Research*, 9:997–1017.

Marchiori, E. (2013). Class dependent feature weighting and k-nearest neighbor classification. In *8th IAPR international conference on Pattern Recognition in Bioinformatics*, pages 69–78.

Marom, N. D., Rokach, L., and Shmilovici, A. (2010). Using the confusion matrix for improving ensemble classifiers. In *IEEE 26th Convention of Electrical and Electronics Engineers in Israel*, pages 000555 – 000559.

Mayer-Schnberger, V. and Cukier, K. (2014). *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. John Murray.

Mei, S., Fei, W., and Zhou, S. (2011). Gene ontology based transfer learning for protein subcellular localization. *BMC Bioinformatics*, 12:44–56.

Mitchell, T. (1978). *Version Spaces: An Approach to Concept Learning, PhD Thesis*. PhD thesis, Stanford Univerity.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill, 1 edition.

Nadeau, C. and Bengio, Y. (2001). Inference for the generalization error. In *Advances in Neural Information Processing Systems 12*, pages 307–313.

Onitilo, A. A., Engel, J. M., Greenlee, R. T., PhD, and Mukesh, B. N. (2009). Breast cancer subtypes based on er/pr and her2 expression: Comparison of clinicopathologic features and survival. *Clinical Medicine and Research*, 7:4–13.

Palsson, B. O. and Zengler, K. (2010). The challenges of integrating multi-omics data set. *Nature Chemical Biology*, 6:787–798.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345 – 1359.

Papadopoulos, H. (2008). Inductive conformal prediction: Theory and application to neural networks. *Tools in artificial intelligence*, 18:315–330.

Papadopoulos, H., Gammerman, A., and Vovk, V. (2009). Reliable diagnosis of acute abdominal pain with conformal prediction. *Engineering Intelligent Systems*, 2:127–137.

Papadopoulos, H., Shafer, G., Vovk, V., and Gammerman, A. (2011). Regression conformal prediction with nearest neighbours. *Journal of Artificial Intelligence Research*, 40:815–840.

Paredes, R. and Vidal, E. (2000). A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recognition Letters*, 21:1027–1036.

Perou, C. M., Srlie, T., Eisen, M. B., Rijn, M., Jeffrey, S. S., and Rees, C. A. (2000). Molecular portraits of human breast tumours. *Nature, International Weekly of Science*, 406:747–752.

Prat, A., Chon, M. U., Cheang, M. M., Parker, J. S., Carrasco, E., and Caballero, R. (2012). Prognostic significance of progesterone receptorpositive tumor cells within immunohistochemically defined luminal a breast cancer. *Journal of Clinical Oncology*, 31:203–209.

Prat, A., Chon, M. U., and Perou, C. M. (2011). Practical implications of gene-expression-based assays for breast oncologistsr. *Journal of Clinical Oncology*, 9:48–57.

Proedrou, K., Nouretdinov, I., Vovk, and Gammerman, A. (2002). Transductive confidence machines for pattern recognition. In *Proceedings of ECML*, LNCS, pages 381–390. Springer.

Pujol, O., Radeva, P., and Vitrià, J. (2006). Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(6):1007–1012.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–86.

Rasiwasia, N., Pereira, J. C., Coviello, E., Doyle, G., Lanckriet, G., Levy, R., and Vasconcelos, N. (2010). A new approach to cross-modal multimedia retrieval. In *Proceedings of ACM Conference on Multimedia*.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141.

Saenko, K., Kulis, B., Fritz, M., and Darrell, T. (2010). Adapting visual category models to new domains. In *Proceedings of ECCV*, LNCS, pages 213 – 226. Springer.

Schweikert, G., Widmer, C., Rtsch, G., and Schlkopf, B. (2009). An empirical analysis of domain adaptation algorithms for genomic sequence analysis. In *Advances in Neural Information Processing Systems*.

Shafer, G. and Vovk, V. (2008). A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421.

Shen, R., Mo, Q., Schultz, N., Seshan, V. E., Olshen, A. B., Huse, J., Ladanyi, M., and Sander, C. (2012). Integrative subtype discovery in glioblastoma using icluster. *PLos ONE*, 7(4).

Shi, F., Ong, S., and Leckie, C. (2013). Applications of class-conditional conformal predictor in multi class classification. In *12th International Conference on Machine Learning and Applications*, pages 235–239.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905.

Shi, X., Liu, Q., Fan, W., and Yu, P. S. (2010). Transfer learning on heterogeneous feature spaces via spectral transformation. In *Proceedings of International Conference on Data Mining*, pages 1049 – 1054.

Shuang Zhou, S., Schoenmakers, G., Smirnov, E., Peeters, R., and Driessens, K. (2015). Largest source subset selection for instance transfer. In *In Proceedings of ACML*, pages 423–438.

Siegel, R. L., Miller, K. D., and Jemal, A. (2016). Cancer statistics, 2016. *A Cancer Journal for Clinicians*, 66:7–30.

Smirnov, E. (2001). *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets, PhD Thesis*. PhD thesis, Universiteit Maastricht.

Smirnov, E., Moed, M., Nalbantov, G., and Sprinkhuizen-Kuyper, I. (2009). Minimally-sized balanced decomposition schemes for multiclass classification. In Okun, O., Valentini, G., and Matteo, R., editors, *Ensembles in machine learning applications*, pages 39–58. Springer.

Smith, R. and Windeatt, T. (2010). Class-separability weighting and bootstrapping in error correcting output code ensembles. In *9th International Workshop on Multiple Classifier Systems*, pages 185–194.

Sniedovich, M. (2011). *Dynamic Programming Foundations and Principles*. CRC Press, 2 edition.

Sotiriou, C. and Pusztai, L. (2009). Gene-expression signatures in breast cancer. *New England Journal of Medicine*, 360:790–800.

Sun, Y., Todorovic, S., Li, J., and Wu, D. (2005). Unifying the error-correcting output code adaboost within the margin framework. In *International Conference on Machine Learning*, pages 872–879.

Tan, P.-N., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Addison-Wesley, 1 edition.

Torrey, L. and Shavlik, L. (2009). *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI, 1 edition.

Trygg, J. and Wold, S. (2003). O2-pls, a two-block (x-y) latent variable regression (lvr) method with an integral osc filter. *Journal of Chemometrics*, 17:53–64.

Vovk, V., Gammerman, A., and Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer, 1 edition.

Wang, C. and Mahadevan, S. (2011). Heterogeneous domain adaptation using manifold alignment. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1541–1546.

Weigelt, B., Baehner, F. L., and Reis-Filho, J. S. (2010). The contribution of gene expression profiling to breast cancer classification, prognostication and prediction: a retrospective of the last decade. *The Journal of Pathology*, 220:263–280.

Weigelt, B., Pusztai, L., Ashworth, A., and Reis-Filho, J. S. (2011). Challenges translating breast cancer gene signatures into the clinic. *Nat Rev Clin Oncol.*, 9:58–64.

Weinstein, J. N., Collisson, E. A., Mills, G. B., and Mills, K. R. (2013). The cancer genome atlas pan-cancer analysis project. *Nature Genetics*, 45:1113–1120.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey on transfer learning. *Journal of Big Data*, 3(9):12 – 44.

Wu, P. and Dietterich, T. G. (2004). Improving svm accuracy by training on auxiliary data sources. In *Proceedings of International Conference on Machine Learning*.

Zhang, X., Liang, L., and Shum, H.-Y. (2009). Spectral error correcting output codes for efficient multiclass recognition. In *IEEE 12th International Conference on Computer Vision, Kyoto*, pages 1111–1117.

Zhou, J. T., Tsang, I. W., Pan, S. J., and Tan, M. (2014). Heterogeneous domain adaptation for multiple classes. In *Proceedings of 17th Conference on AISTATS*. Journal of Machine Learning Research.

Zhou, S., Smirnov, E., Shoenmakers, G., Driessens, K., and Peeters, R. (2017). Testing exchangeability for transfer decision. *Pattern Recognition Letters*, 88:64–71.

Zhou, X. and Tuck, D. P. (2007). Gene expression msvm-rfe: extensions of svm-rfe for multiclass gene selection on dna microarray data. *Bioinformatics*, 9:1106–1114.

Zor, C., Yanikoglu, B. A., Windeatt, T., and Alpaydin, E. (2010). FLIP-ECOC: A greedy optimization of the ECOC matrix. In *Proceedings of the 25th International Symposium on Computer and Information Sciences, London, UK, September 22-24, 2010*, pages 149–154.

List of Figures

List of Tables

Publications

Ismailoglu, F., Smirnov, E., Nikolaev, N., and Peeters, R. (2015). Instance-Based Decompositions of Error Correcting Output Codes. In 12th Int.Workshop on Multiple Classifier Systems (MCS), LNCS, vol. 9132, pages 51-63. Springer International Publishing.

Ismailoglu, F., Sprinkhuizen-Kuyper, I. G., Smirnov, E., Escalera, S. and Peeters, R. (2015). Fractional Programming Weighted Decoding for Error-Correcting Output Codes. In 12th Int.Workshop on Multiple Classifier Systems (MCS), LNCS, vol. 9132, pages 38-50. Springer International Publishing.

Ismailoglu, F. Weighted Decoding for Error-Correcting Output Codes via Bipartite Graph Partitioning. (2015). In Proc. of the 27th Benelux Conf. on Artificial Intelligence (BNAIC).

Ismailoglu, F., Smirnov, E. and Peeters, R. (2015). Conformal ECOC Machines. In Proc of IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pages 361–368.

Ismailoglu, F., Cavill R., Smirnov, E., Zhou, S., Collins, P. and Peeters, R. (2017). Heterogeneous Domain Adaptation for IHC Classification of Breast Cancer Subtypes.
To appear in IEEE/ACM Transactions on Computational Biology and Bioinformatics.

About the Author

Firat Ismailoglu was born in Mersin, Turkey, on March 16, 1984. He graduated from Selcuk University, Turkey with a B.Sc in Mathematics in 2007. Thereafter, he did a master in Topology at Mersin University, Turkey for one year. He then shifted his academic interest more towards computer science. In this sense, he obtained a master in Knowledge Discovery and Data Mining from University of East Anglia, UK in 2010. His master thesis entitled *A Medical Data Mining Application Covering Patients Over 90 Years Old by Using the CART Algorithm* was published at the journal Archives of Gerontology and Geriatrics. As of September 2011, he started working his PhD research at the Department of Data Science and Knowledge Engineering, Maastricht University, the Netherlands. During his PhD research he has co-supervised a master thesis and published several international conference and journal publications of high quality. These publications are collected in this thesis. Since January 2017, he has been working as a postdoc at the Department of Mathematics and Computer Science, Eindhoven University of Technology.