Max-Margin Stacking with Group Sparse Regularization for Classifier Combination

by

Mehmet Umut SEN

Submitted to
the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

SABANCI UNIVERSITY

September 2011

MAX-MARGIN STACKING WITH GROUP SPARSE REGULARIZATION FOR CLASSIFIER COMBINATION

APPROVED BY

Assist. Prof. Dr. Hakan ERDOĞAN (Thesis Supervisor) .....

Assoc. Prof. Dr. Berrin YANIKOĞLU .....

Assoc. Prof. Dr. Zehra ÇATALTEPE .....

Assist. Prof. Dr. Müjdat ÇETİN .....

Assoc. Prof. Dr. Özgür ERÇETIN .....

DATE OF APPROVAL: 08.08.2011

To my family...

Acknowledgements

I would like to express my deep and sincere gratitude to my thesis supervisor Hakan Erdoğan for his invaluable guidance, tolerance, positiveness, support and encouragement throughout my thesis.

I am grateful to my committee members Berrin Yanıkoğlu, Zehra Çataltepe, Müjdat Çetin and Özgür Erçetin for taking the time to read and comment on my thesis.

I would like to thank TÜBİTAK for providing the necessary financial support for my masters education.

My deepest gratitude goes to my parents for their unflagging love and support throughout my life. This dissertation would not have been possible without them.

MAX-MARGIN STACKING WITH GROUP SPARSE REGULARIZATION FOR CLASSIFIER COMBINATION

MEHMET UMUT SEN

EE, M.Sc. Thesis, 2011

Thesis Supervisor: Hakan Erdoğan

Keywords: stacked generalization, classifier combination, hinge loss, group sparsity, kernel trick

Abstract

Multiple classifier systems are shown to be effective in terms of accuracy for multiclass classification problems with the expense of increased complexity. Classifier combination studies deal with the methods of combining the outputs of base classifiers of an ensemble. Stacked generalization, or stacking, is shown to be a strong combination scheme among combination algorithms; and in this thesis, we improve stacking's performance further in terms of both accuracy and complexity. We investigate four main issues for this purpose. First, we show that margin maximizing combiners outperform the conventional least-squares estimation of the weights. Second we incorporate the idea of group sparsity into regularization to facilitate classifier selection. Third, we develop non-linear versions of class-conscious linear combination types by transforming datasets into binary classification datasets; then applying the kernel trick. And finally, we derive a new optimization algorithm based on the majorization-minimization framework for a particular linear combination type, which we show is the most preferable one.

SINIFLANDIRICI BİRLEŞTİRME İÇİN GRUP SEYREKLİĞİ İLE BERABER SINIR ENBÜYÜKLEYEN YIĞITLAMA

MEHMET UMUT ŞEN

EE, Yüksek Lisans Tezi, 2011

Tez Danışmanı: Hakan Erdoğan

Anahtar Kelimeler: yığıtlı genelleme, sınıflandırıcı birleştirme, menteşe kaybı, grup seyrekliği, kernel hilesi

Özet

Çoklu sınıflandırıcı sistemlerinin, çok sınıflı sınıflandırma problemlerinde karmaşık fakat doğruluk oranı yüksek bir sınıflandırma yöntemi olduğu, örüntü tanıma literatüründe sıkça işlenmiştir. Sınıflandırıcı birleştirme, verilen bir sınıflandırıcı kümesini nasıl birleştirilmesi gerektiği problemini çözmeye çalışır ve yığıtlı genelleme, başka bir deyişle yığıtlama, çok güçlü sınıflandırıcı birleştiricilerden biridir. Bu tezde yığıtlamanın performansını hem doğruluk oranı açısından, hem de karmaşıklık açısından artırıyoruz. Katkılarımız dört ana başlıkta toplanabilir. Öncelikle, birleştiriciyi öğrenirken sınırı en-büyükleyen menteşe kayıp fonksiyonu kullanmanın, literatürde daha önce kullanılan en küçük kareler kayıp kestiriminden daha iyi sonuçlar verdiğini gösterdik. İkinci olarak, düzenlileştirme için grup seyrekliği kullanarak otomatik sınıflandırıcı seçmeyi kolaylaştırıyoruz. Üçüncü olarak, sınıf-bilinçli doğrusal birleştiricilerin doğrusal olmayan sürümlerini elde etmek için, veritabanını dönüştüren bir yöntem geliştiriyoruz. Son olarak, doğrusal bir sınıflandırıcı birleştirme yontemi için MM algoritmalarını kullanarak bir çözüm buluyoruz.

Contents

List of Figures

List of Tables

Abbreviations

CS	**C**rammer-**S**inger
CV	**C**ross **V**alidation
CWS	**C**lass-Dependent **W**eighted **S**um
DP	**D**ecision **P**rofile
DT	**D**ecision **T**emplate
LSG	**L**inear **S**tacked **G**eneralization
LS-SVM	**L**east **S**quares **SVM**
MLR	**M**ulti-Response **L**inear **R**egression
MM	**M**ajorize **M**inimize
RBF	**R**adial **B**asis **F**unction
RERM	**R**egularized **E**mpirical **R**isk **M**inimization
SVM	**S**upport **V**ector **M**achines
WS	**W**eighted **S**um

Chapter 1

Introduction

1.1 Motivation

This thesis is concerned with multiclass classification problems, which constitute a wide part of the vast pattern recognition literature and have a broad range of applications such as protein structure classification, heartbeat arrhythmia identification, hand-written character recognition, sketch recognition, object recognition in computer vision and many others. Multiclass classification deals with assigning one of several class labels to an input object. This problem is sometimes misguidedly called "Multi-label classification", which actually deals with problems in which examples are associated with a set of labels, instead of with only one label. For instance, a movie may belong to categories *comedy* and *drama* at the same time. In this thesis, we work on single-label, multiclass classification problems. It is needless to say that our developed methods are also applicable to binary classification problems, since binary classification is a special case of multiclass classification.

Early work in machine learning focused on using single classifier systems, but recently there is an enormous amount of work on multiple classifier systems in which multiple classifiers are trained and combined. A multiple classifier system mainly consists of two subsequent problems: 1. How to construct the base classifiers of the ensemble? 2. How to combine the outputs of the base classifiers? There are a large number of works that try to solve these problems separately, moreover there are some methodologies that try to solve these two problems simultaneously [3, 4]. In this thesis, we focus on the latter

problem, i.e., we try to increase the performance of the combiner for any given base classifier set and dataset. Among different combination methods in the literature, we work on *stacked generalization*, also known as *stacking*, and improve its performance further. The main performance criterion of a combiner is the generalization accuracy, i.e., accuracy on test data. In addition, complexity of the combiner is also a crucial issue, because some applications may necessitate small training time and/or testing time. Our work contains methods each of which either increases the accuracy or decreases the complexity of a combiner.

In Chapter 2, we give a literature review for multiple classifier systems and give brief descriptions of well known combination methodologies. In Chapter 3, we introduce stacking, internal cross validation, different linear combination types, which we call *weighted sum* (WS), *class-dependent weighted sum* (CWS), *linear stacked generalization* (LSG) and explain some stacking algorithms that are present in the literature. After this literature review, we present our contributions in Chapters 4,5 and 6. In Chapter 4, we propose to use the hinge loss function for learning the combiner and show that it outperforms the conventional least-squares estimation that is used in the literature for stacked generalization. We also propose to use group sparsity in the regularization function of the learner, rather than using the l_1 norm regularization which is used in previous works, to facilitate classifier selection. We also describe a unifying framework for different linear combination types, which is helpful for deriving optimization algorithms for class-conscious linear combination types. With this framework, after obtaining a solution to the most general linear combination type, i.e. linear stacked generalization(LSG), we are able to obtain the solutions of other linear combination types by adjusting and incorporating tying matrices into the solutions. In Chapter 5, we work on nonlinear combination with stacking. We derive methods to obtain the nonlinear versions of WS and CWS combination with kernel trick for least-squares estimation. In Chapter 6, we give an optimization algorithm for CWS combination using majorize-minimize (MM) algorithms. We find a solution for only CWS because experiments given in Chapter 4 suggest that CWS seems to be the best type of linear combination considering accuracy and training time together. In Chapter 7, we summarize our work, conclude the thesis and present possible future directions.

1.2 Contributions of the thesis

- We propose using the hinge loss function for learning the weights and obtain statistically significantly better results compared to the previous methods which use the least-squares loss function.
- We consider all three different linear combination types and compare them, where previous works usually only attack one of them.
- We obtain a unifying framework for different linear combination types, which is helpful for obtaining solutions to simpler linear combination types.
- We propose to use group sparsity in regularization to facilitate classifier selection and obtain better results compared to the conventional l_1 norm regularization.
- We give methods to obtain non-linear versions of WS and CWS combination types for the least-squares loss function.
- We obtain a solution for CWS combination using majorize-minimize (MM) algorithms.

Chapter 2

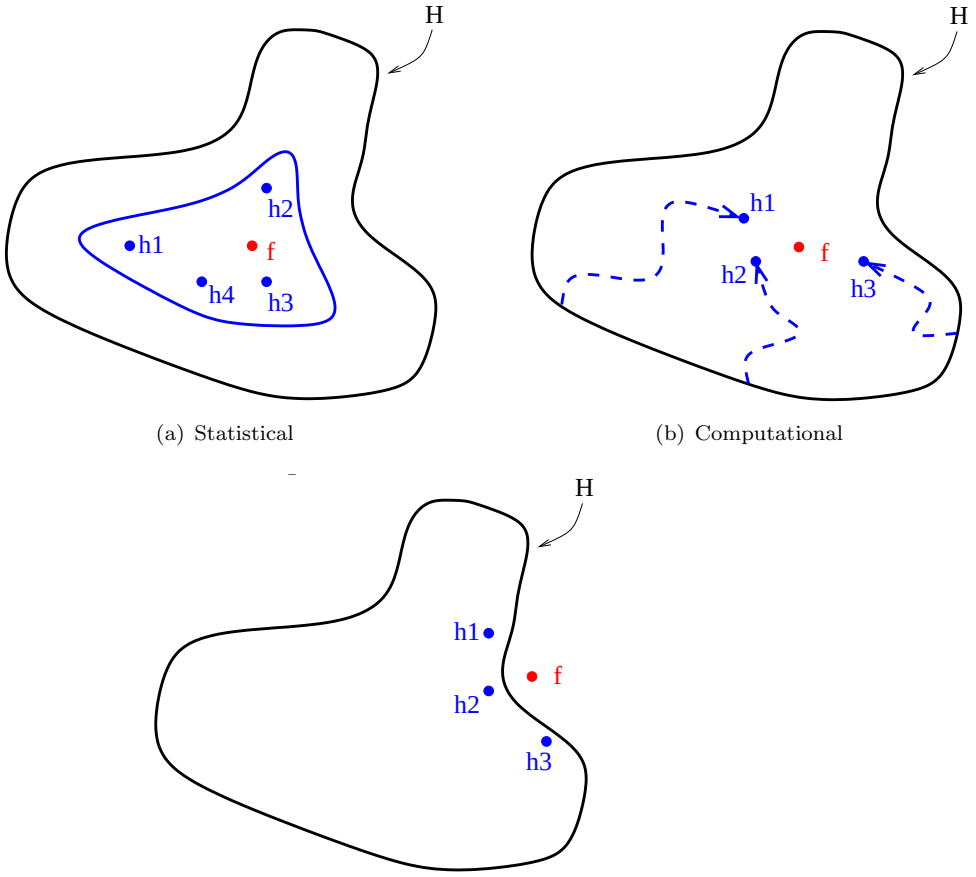
Multiple Classifier Systems

2.1 Introduction

When individuals are about to make a decision; they often seek others' opinions, process all the information obtained from these opinions and reach a final decision. This decision may effect their life significantly as in a financial, medical way; or may not. Even when they are about to buy an electronic device, they search forums on the Internet and try to come up with a decision that is optimal for their benefits. Sometimes they reach the perfect solution, sometimes they do not. However, consulting several resources, regardless of the level of the expertise of these resources, prevents individuals for making terrible choices at the end. This phenomenon has been deeply investigated and applied to several areas in pattern recognition. Classifier ensembles; also known under various other names, such as multiple classifier systems, committee of classifiers, hybrid methods, cooperative agents, opinion pool, or ensemble based systems; have shown to outperform single-expert systems for a broad range of applications in a variety of scenarios. In this chapter, we give a summary of the vast literature on multiple classifier systems.

2.2 Why classifier combination?

Why might an ensemble work better than a single classifier? Dietterich [1] suggested three types of reasons for that question: *statistical*, *computational*, and *representational* reasons. We add another category, namely *natural* reasons and explain all of them.



(a) Statistical

(b) Computational

(c) Representational

FIGURE 2.1: Three fundamental reasons of why an ensemble may work better than a single classifier suggested by Dietterich. Figure taken from [1].

2.2.1 Statistical Reasons

In classification problems, good performance on training data does not necessarily lead to good generalization performance, defined as the performance of the classifier on data not seen during training, i.e., test data. For example, two classifiers that give the same accuracy on training data may have different test accuracies. This problem arises from insufficient number of training examples compared to the complexity of the problem, which is often the case in pattern recognition problems. In such cases, training and combining more than one classifier reduces the risk of an unfortunate selection of a poorly performing classifier. Combined selection may not beat a single classifier on a particular data-point, but it will surely beat most of the classifiers; and given the differing performance of base classifiers on different datasets and even on different subsets of datasets, an ensemble leads to good generalization performance in general. An illustration is given in Figure 2.1(a), in which, f is the optimal classifier, H is the classifier

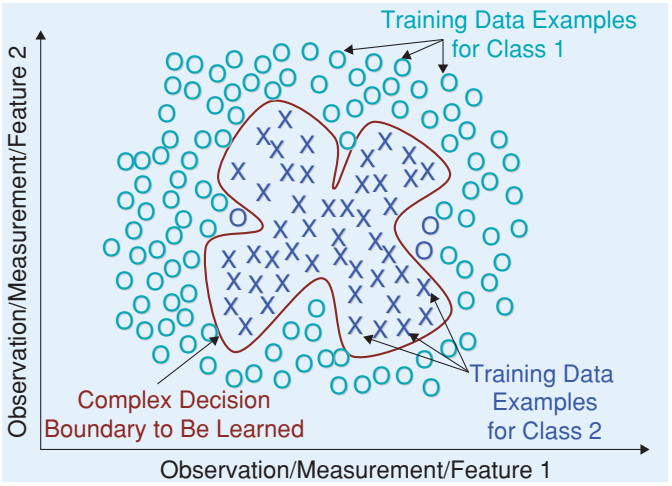
space and $h1, h2, h3, h4$ are the individual classifiers in the ensemble. We try to find a classification hypothesis which is as close to f as possible by combining individual classifiers. Another statistical reason that motivates ensembles systems, which is addressed in [2], is named *too little data*. In the absence of adequate training data, resampling techniques can be used for drawing overlapping random subsets of the available data, and from each subset a base classifier can be trained. Ensembles constructed this way are proven to be effective.

2.2.2 Computational Reasons

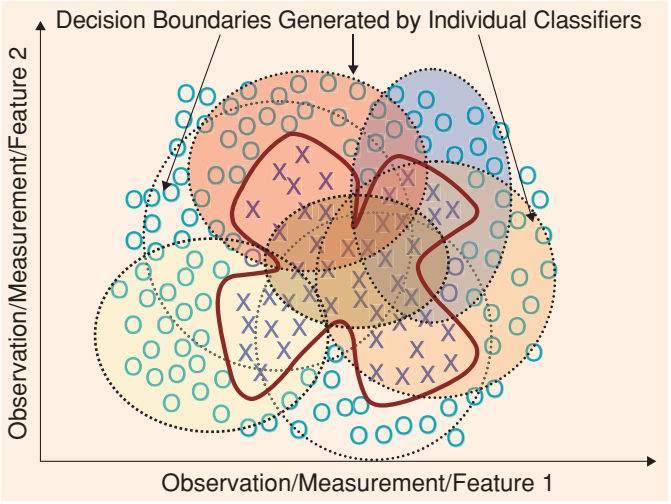
Classifiers in the ensemble are learned by some algorithms and there are some computational problems concerning these learning algorithms. Training of a classifier starts from a point in the classifier space and we want it to end near f as illustrated in Figure 2.1(b). Some training algorithms perform local search that may get stuck in local optima. We can deal with this problem by running the local search from many different starting points and obtain a better approximation to the true unknown function than any of the individual classifiers [1]. Another issue regarding training time for large volumes of data is addressed in [2]. In the case of a large amount of data to be analyzed, a single classifier may not be able to effectively handle it. In this case, dividing the data into overlapping or non-overlapping subsets, training a single classifier from each subset and combining them may result in faster training time overall and in better accuracy.

2.2.3 Representational Reasons

In most of the cases, the classifier space H does not contain the true classification hypothesis, as illustrated in Figure 2.1(b). In such cases, combining several classifiers in the classifier space H may lead to a classifier that is out of H and closer to the optimal classification f. An illustration for such a case is given in Figures 2.2(a) and 2.2(b). The optimal classification boundary is not reachable with linear or quadratic classifiers, since it is much more complex. In that case, combining several quadratic classifiers are helpful to get close to the more complex optimal classifier as illustrated in Figure 2.2(b).



(a) Optimal boundary



(b) Base classifiers

FIGURE 2.2: A binary problem that cannot be learned by quadratic classifiers and but can be learned by an ensemble of quadratic classifiers. Figure taken from [2].

2.2.4 Natural Reasons

In some cases, we may have a classification problem in which data are obtained from various sources leading to sets of features that are heterogeneous. These feature sets may have dissimilar characteristics, such as different distributions, or in the worst case some features can be categorical and some of them can be numerical. In such cases, a single classifier may not be used to learn the information contained in all of the data and this situation induces a natural decomposition of the problem: training base classifiers from each set of features and combining them, that is more natural and leads to better performance than using a single classifier. Ho says that "In these cases, the input can be considered as vectors projected to the subspaces of the fill feature space, which have to

be matched by different procedures" [5]. An example to such a case is the audio-visual speech recognition problem [6, 7], in which there are features from both audio and video of the same class. Another example is authentication by several biometrics, such as fingerprint, voice, face, etc.

2.3 Types of Classifier Outputs

Combiner of an ensemble receives the outputs of base classifiers and makes a final decision after a combination procedure. Base classifiers can produce different kinds of outputs and some combination methods are not applicable to certain kinds of outputs types. Types of classifier outputs are as follows:

- **Class Labels:** Given a data-point, a base classifier outputs the estimated label of the data-point. This type of output is called "the abstract level" output in [8]. In this case, combiners like majority voting or weighted majority voting can be used.
- **Class Ranks:** In this type, each base classifier produce a sequence of labels decreasing in probability of being the correct class for a given data-point. This type of output is called "the rank level" output in [8]. Class ranks gives more information than the abstract level outputs.
- **Confidence Scores:** In this type, each base classifier produces a continuous-valued score for each class that represents the degree of support for a given data-point. They can be interpreted as confidences in the suggested labels or estimates of the posterior probabilities for the classes [8]. Former thinking is more reasonable since for most of the classifier types, support values may not be very close to the actual posterior probabilities even if the data is dense, because classifiers generally do not try to estimate the posterior probabilities, but try to classify the data instances correctly; so they usually only try to force the true class' score to be the maximum. Confidence scores have the most information among output types and most ensemble and combiner methods work with this type of output. In this thesis, we deal with the combination of continuous valued outputs.

2.4 Ensemble Construction Methods

How to obtain different base classifiers for a given problem is a crucial issue in multiple classifier systems. There are four elements that can differ between two base classifiers:

- Classifier type
- Classifier meta-parameters
- Training data-points
- Features

Latter two elements result in more diverse ensembles. There are different measures of diversity of an ensemble, but diversity simply means that base classifiers make errors on different examples. Diverse ensembles result in higher performance increase with a reasonable combiner. There is a trade-off between the accuracies of the base classifiers and diversity. In fact, as the base classifiers get more accurate, their correlations increase, i.e., diversity decreases. Therefore, it is usually beneficial to include weak base classifiers in the ensemble rather than constructing the ensemble with only strong base classifiers. Because, weak base classifiers contains complementary information, which is helpful for increasing the performance of a multiple classifier system.

Below, we explain two well-known ensemble construction methods, namely *bagging* and *boosting*.

2.4.1 Bagging

Bagging is a term introduced by Breiman as an acronym for Bootstrap AGGregatING [9]. The basic idea of bagging is constructing base classifiers from randomly selected subsets of data-instances. A uniform distribution is used for selection and data-instances are selected with replacement, i.e., a data-instance can be included more than once. After constructing the base classifiers, majority voting is applied for combination; but the prominent idea of bagging is how the base classifiers are constructed, rather than how they are combined.

2.4.2 Boosting

Origins of boosting algorithms rely on the answer by Schapire [10] to the question posed by Kearns [11]: can a set of weak learners create a single strong learner? The basic idea of boosting is to build the base classifiers iteratively such that each base classifier tries to compensate the weaknesses of previous base classifiers. To achieve this goal, each data-instance is given a probability to be included in the training data and this probability distribution is initialized to be uniform. At each iteration, a base classifier is trained with data-instances that are randomly selected with this probability distribution. After the training process, distribution is updated according to some criteria based on the classification results of the training data with the recently trained classifier and previous classifiers. These criteria basically increase the probability of a data-instance to be selected if that data-instance is believed to be a "difficult" point to be correctly classified. Base classifiers obtained this way reflect some sort of local information and they are weak learners. But with a good combination scheme, their ensemble constitutes a strong classification algorithm. A disadvantage of boosting is shown to be that it may be sensitive to the outliers in the dataset, but good boosting algorithms are able to eliminate this problem successfully. There are a wide range of algorithms that rely on the boosting method such as the Adaboost algorithm [4], arc-x4 algorithm [3], etc.

In this thesis, we are not interested in the methods of obtaining an ensemble, but we investigate various linear and nonlinear combination types and learners for a given set of base classifiers.

2.5 Combiners

Combination methods can be grouped as trainable and non-trainable combiners. We first define the combination problem, than describe some well known trainable and non-trainable combiners.

2.5.1 Problem Formulation

In the classifier combination problem with confidence score outputs, inputs to the combiner are the posterior scores belonging to different classes obtained from the base

classifiers. Let p_m^n be the posterior score of class n obtained from classifier m for any data instance. Let $\boldsymbol{p}_m = [p_m^1, p_m^2, \ldots, p_m^N]^T$, then the input to the combiner is $\boldsymbol{f} = [\boldsymbol{p}_1^T, \boldsymbol{p}_2^T, \ldots, \boldsymbol{p}_M^T]^T$, where N is the number of classes and M is the number of classifiers. Outputs of the combiner are N different scores representing the degree of support for each class. Let r^n be the combined score of class n and let $\mathbf{r} = [r^1, \ldots, r^N]^T$; then in general the combiner is defined as a function $g : \mathbb{R}^{MN} \rightarrow \mathbb{R}^N$ such that $\mathbf{r} = g(\boldsymbol{f})$. On the test phase, label of a data instance is assigned as follows:

$$\hat{y} = \underset{n \in [N]}{\arg\max}\, r^n, \tag{2.1}$$

where $[N] = \{1, \ldots, N\}$. Block diagram of the classifier combination problem with confidence score outputs is given in Figure 2.3.

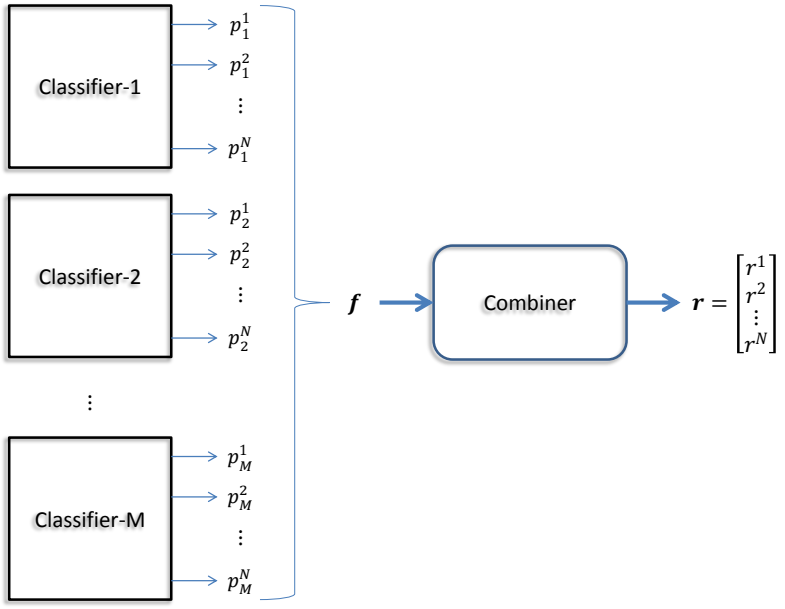


FIGURE 2.3: Outputs of the base classifiers and the combiner.

2.5.2 Non-trainable Combiners

This category contains simple rules that constitutes a large volume of the multiple classifier system literature. These simple rules may be preferable to trainable combiners in the case of inadequate training data.

2.5.2.1 Mean Rule

This type of combination is also called the *sum rule* or the *average rule*. The basic idea is simply summing, or averaging, the confidence scores of a particular class through base classifiers to obtain the final score of that class,:

$$r^n = \frac{1}{M} \sum_{m=1}^{M} p_m^n \tag{2.2}$$

Mean rule is one of the strongest non-trainable combination types.

2.5.2.2 Trimmed Mean

This combination rule tries to eliminate harms of the outliers in the ensemble for the mean rule. Some classifiers may give unusually high or low scores to a particular class and these scores are not included in the averaging. For a U% trimmed mean, we remove U% of the scores from each end, and take average; so that we are able to avoid extreme support values.

2.5.2.3 Product Rule

In the product rule, instead of summing the scores as in the mean rule, we multiply the scores. This rule is very sensitive to the outliers in the ensemble, because if a score is very low or very high, it has a huge effect on the resulting score. But if the posterior scores are estimated correctly, then the product rule provides a good estimate. The rule is as follows:

$$r^n = \prod_{m=1}^{M} p_m^n \tag{2.3}$$

2.5.2.4 Minimum Rule

We just simply take the minimum among the classifiers' output scores:

$$r^n = \min_m p_m^n \tag{2.4}$$

The motivation behind this rule is to assign the test instance to the class for which there is no base classifier that disagrees on the decision. Performance of the minimum rule tends to increase as the base classifiers are all strong.

2.5.2.5 Maximum Rule

In this type of combination, we just take the Maximum among the classifiers' output scores:

$$r^n = \max_m p_m^n \tag{2.5}$$

With this combination, if one base classifier insists on a particular class for a given test instance, final decision assigns the test instance to that class; even if all other base classifiers disagree.

2.5.2.6 Median Rule

For median rule, we simply take the Median among the classifiers' output scores:

$$r^n = \operatorname*{median}_m p_m^n \tag{2.6}$$

This decision rule also provide robustness to outliers, like the trimmed mean rule.

2.5.2.7 Majority Voting

This is another one of the strongest non-trainable combiners besides the sum rule. To find the final score of a class, we simply count the number of classifiers that chooses that particular class, i.e., classifiers that assign the highest score to the class. This rule is also suitable for ensembles that outputs class labels or class ranks, in fact, it does not use the scores, it just uses the class labels. If only class labels are obtained from base classifiers, majority voting is the optimal rule under minor assumptions: (1) The number of classifiers are odd and the problem is a binary classification problem, (2) The probability of each classifier for choosing any class is equal for any instance, (3) Base classifiers are independent [2].

2.5.3 Trainable Combiners

With trainable combiners, we learn some parameters, usually viewed as and called *weights*, from a set of training data. Let I be the number of training data instances of the combiner, $\boldsymbol{f}_i$ contain the scores for training data point i obtained from base classifiers and y_i be the corresponding class label; then our aim is to learn the g function using the data $\{(\boldsymbol{f}_i, y_i)\}_{i=1}^{I}$. Given a dataset to train the whole ensemble, including base classifiers and the combiner, handling of the dataset is a crucial issue for trainable combiners. Duin [12] suggests that, if we use the same data-instances to train both base classifiers and the combiner, we should not overtrain the base classifiers because the combiner will be biased in this case. But if we train the base classifiers and the combiner from two disjoint subsets of the dataset, than base classifiers can be overtrained, since the bias will be corrected by training the combiner on the separate training set. But when we use separate training sets for the base classifiers and the combiner, training dataset eventually is not efficiently used. Wolpert [13] deals with this problem using internal cross-validation (CV). We explain internal CV in section 3.3.

We consider weighted mean rule and weighted product rule under trainable combiners, even though they are much simple compared to other trainable combiners such as decision templates, Dempster-Shafer based combination and stacked generalization that we describe below. The reason is, we may still use the training data for finding the weights, even if the learning methods are simple such as determining the weights according to cross-validation accuracies of the base classifiers. In fact, we include the weighted mean rule in our experiments under the framework of stacked generalization. We define the most well known trainable combiners in the subsequent sections.

2.5.3.1 Weighted Mean Rule

The basic idea of weighted mean rule, which is also called weighted average rule, is reflecting confidences of the individual classifiers to the mean rule. We assign each classifier a weight and take the weighted average:

$$r^n = \frac{1}{M} \sum_{m=1}^{M} u_m p_m^n, \tag{2.7}$$

where u_m is the weight of classifier m. Learning the weights is a crucial issue, especially if the diversity of the ensemble is large. We include this type of combination in our experiments under the framework of stacked generalization.

2.5.3.2 Weighted Product Rule

With weighted product rule, the final score of class n is estimated as follows:

$$r^n = \prod_{m=1}^{M} (p_m^n)^{u_m}, \tag{2.8}$$

where u_m is the weight of classifier m. This rule is equivalent to taking logarithms of posterior scores and then applying weighted mean rule, which follows from the following fact:

$$\arg\max_n \sum_{m=1}^{M} u_m \ln p_m^n = \arg\max_n \prod_{m=1}^{M} (p_m^n)^{u_m} \tag{2.9}$$

2.5.3.3 Decision Templates

Kuncheva described decision templates in 2001 [14]. On the training phase of decision templates, we find a decision template for each class. On the test phase, we find the decision profile of a given data-instance and find the distance of this decision profile to decision templates of each class with a particular distance metric. We assign the test instance to the class that has the minimum distance. Given a data-instance $\boldsymbol{x}$, a decision profile $\mathrm{DP} \in \mathbb{R}^{M \times N}$ is a matrix, containing the scores obtained from base classifiers: $[\mathrm{DP}(\boldsymbol{x})]_{m,n} = p_m^n$. An illustration for a decision profile matrix is given in Figure 2.4.

Decision template of a class is found by averaging the decision profiles of training data-instances that belong to that particular class:

$$\mathrm{DT}_n = \frac{1}{|A_n|} \sum_{i \in A_n} \mathrm{DP}(\boldsymbol{x}_i), \tag{2.10}$$

where A_n is the set of data-points that has the label n. After learning the decision templates from training data, we find the score of a data-instance by calculating the

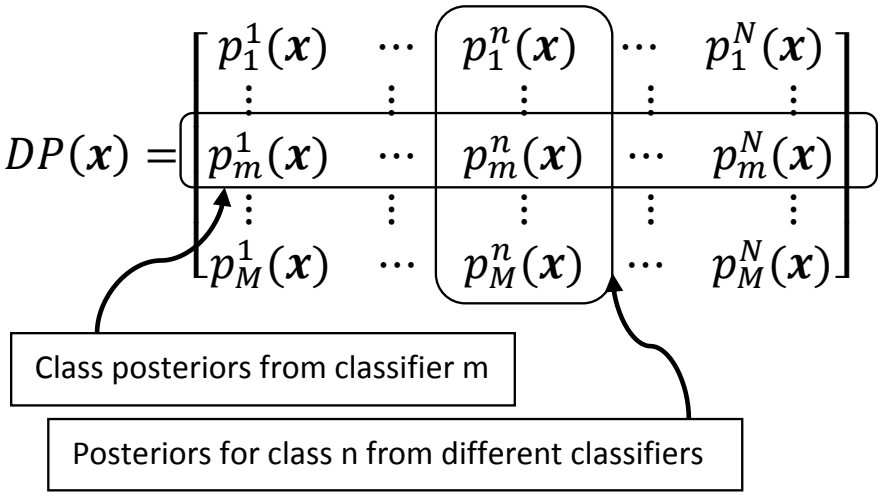


FIGURE 2.4: Decision profile matrix.

similarity S between $\mathrm{DP}(\boldsymbol{x})$ and DT_n for each class n:

$$r^n(\boldsymbol{x}) = S(\mathrm{DP}(\boldsymbol{x}), \mathrm{DT}_n), \quad n = 1, \ldots, N \tag{2.11}$$

The similarity measure is usually found by the square of the Euclidean distance:

$$r^n = 1 - \frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} (\mathrm{DT}_n(m,n) - p_m^n)^2, \tag{2.12}$$

where $\mathrm{DT}_n(m,n)$ is the element of DT_n at row m and column n. Decision templates with squared Euclidean distance can be formulated in the framework of stacking. In particular, it corresponds to the nearest mean classifier, which is a linear classifier, on posterior scores of the base classifiers.

2.5.3.4 Dempster-Shafer Based Combination

The Dempster-Shafer (DS) Theory [15], which is the basis of many data fusion techniques, is also applied to many decision making problems, including classifier combination [16, 17]. Instead of combining data from different sources as in data fusion problems, DS theory is used to combine the evidence provided by ensemble classifiers trained on data coming from the same source. Let DT_n^m denote the m^{th} row of the decision template DT_n. Then we calculate a *proximity* value, $\Phi_{n,m}(\boldsymbol{x})$, for classifier m, class n and for data-instance $\boldsymbol{x}$ as follows:

$$\Phi_{n,m}(\boldsymbol{x}) = \frac{(1 + \|\mathrm{DT}_n^m - \boldsymbol{p}_m\|^2)^{-1}}{\sum_{n'=1}^{N}(1 + \|\mathrm{DT}_{n'}^m - \boldsymbol{p}_m\|^2)^{-1}}, \tag{2.13}$$

where, $\boldsymbol{p}_m$ is defined in Section 2.5.1. After calculating these proximities for each class and classifier, we compute the *belief*, or evidence, that classifier m correctly identifies instance $\boldsymbol{x}$ as class n:

$$b_n(\boldsymbol{p}_m) = \frac{\Phi_{n,m}(\boldsymbol{x}) \prod_{n' \neq n}(1 - \Phi_{n',m}(\boldsymbol{x}))}{1 - \Phi_{n,m}(\boldsymbol{x})(1 - \prod_{n' \neq n}(1 - \Phi_{n',m}(\boldsymbol{x})))} \tag{2.14}$$

After we obtain the belief values for each classifier, we combine them using Dempster's rule of combination:

$$r^n = K \prod_{m=1}^{M} b_n(\boldsymbol{p}_m), \tag{2.15}$$

where K is a normalization constant.

2.5.3.5 Stacked Generalization

Stacked generalization, also known as *stacking*, is first introduced by Wolpert in 1992 [13]. It works with the assumption that there are still some patterns at the posterior score level after classification with the base classifiers. Hence, another generalizer/classifier is applied to posterior scores to catch these patterns. All the work in this thesis follows the idea of stacked generalization and we improve stacking's performance further, in terms of both accuracy and train/test time. We give a comprehensive introduction to stacking in Chapter 3.

Chapter 3

Stacked Generalization

3.1 Introduction

A novel approach has been introduced in 1992 known as stacked generalization or stacking [13]. The basic idea of stacking is applying a meta-level (or level-1) generalizer to the outputs of base classifiers (or level-0 classifiers). This method makes the assumption that there are still some patterns after the classification by the base classifier and the combiner tries to catch these patterns. Wolpert focused on the regression problem and he combined the predictions of individual classifiers with this framework as if they are features. He also points out that the meta-feature space can be augmented with the original inputs or with other relevant measures. He used internal cross-validation to use the training data more efficiently for learning the combiner. Internal cross-validation is explained in Section 3.3. Ting & Witten applied stacking to classification problems by combining continuous valued probabilistic predictions of base classifiers [18].

Seewald in [19] showed that stacking is universal in the sense that most ensemble learning schemes can be mapped onto stacking via specialized meta classifiers. He presented operational definitions of these meta classifiers for voting, selection by cross-validation, grading, and bagging. In addition, decision templates with squared Euclidean distance, which is a more sophisticated method compared to the schemes given above, can also be formulated in the framework of stacking. In particular, it corresponds to a naive learning of the weights of Linear Stacked Generalization (LSG) combination, which is described in Section 3.4.3.

3.2 Problem Formulation

Among combination types, linear ones are shown to be powerful for the classifier combination problem. For linear combiners, the g function introduced in Section 2.5.1 has the following form:

$$g(\mathbf{f}) = \boldsymbol{W}\mathbf{f} + \mathbf{b}. \tag{3.1}$$

In this case, we aim to learn the elements of $\boldsymbol{W} \in \mathbb{R}^{N \times MN}$ and $\mathbf{b} \in \mathbb{R}^{N}$ using the database $\{(\boldsymbol{f}_i, y_i)\}_{i=1}^{I}$. So, the number of parameters to be learned is $MN^2 + N$. This type of combination is the most general form of linear combiners and called type-3 combination in [20]. In the framework of stacking, we call it linear stacked generalization (LSG) combination. One disadvantage of this type of combination is that, since the number of parameters is high, learning the combiner takes a lot of time and may require a large amount of training data. To overcome this disadvantage, simpler but still strong combiner types are introduced with the help of the knowledge that p_m^n is the posterior score of class n. We call these methods weighted sum (WS) rule and class-dependent weighted sum (CWS) rule. These types are categorized as class-conscious combinations in [8].

3.3 Internal Cross Validation

For training the level-1 classifier, we need the confidence scores (Level-1 Data) of the training data, but training the combiner with the same data instances which are used for training the base classifiers will lead to overfitting the database and eventually result in poor generalization performance. So we should split the dataset into two disjoint subsets for training the base classifiers and the combiner. But this partitioning leads to inefficient usage of the dataset. Wolpert deals with this problem by a sophisticated cross-validation method (internal CV), in which training data of the combiner is obtained by cross validation. In k-fold cross-validation, training data is divided into k parts and each part of the data is tested with the base classifiers that are trained with the other $k-1$ parts of data. So at the end, each training instance's score is obtained from the base classifiers whose training data does not contain that particular instance. This procedure is repeated for each base classifier in the ensemble. An illustration of 4-fold internal CV

for just one base classifier is given in Figure 3.1. We apply this procedure for the three different linear combination types.

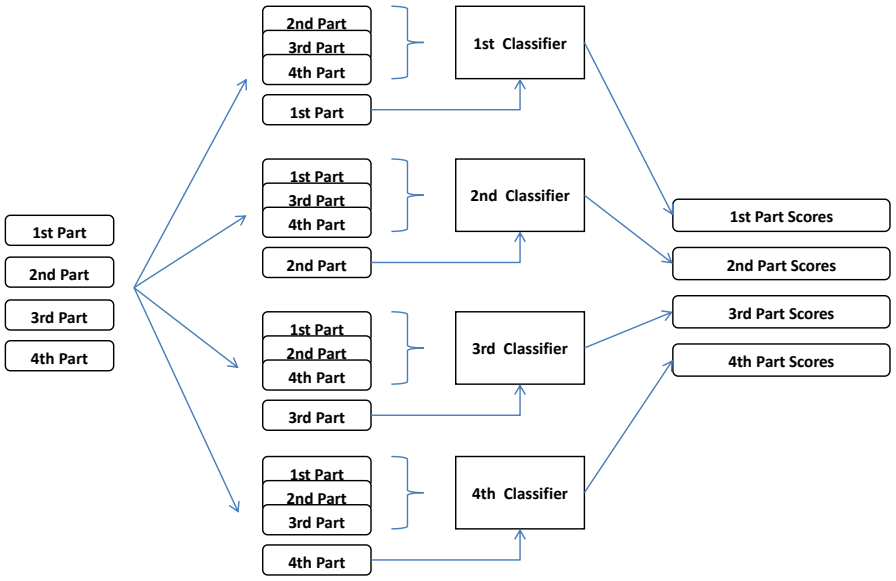


FIGURE 3.1: An illustration of 4-fold internal CV.

Let Π_m, Q_m, and F_m be the meta-parameters, subsets of training data-point indices and subsets of feature indices that are given as inputs to the classifier C_m, respectively. Q_m may contain repeated indices, as in the case of *bagging*. Let $\boldsymbol{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{I}$ be a training dataset and let O_m be the resulting model: $O_m = C_m(\boldsymbol{D}; Q_m, F_m, \Pi_m)$. Let T map a set of test instances to posterior scores using a given model: $\boldsymbol{P}_m^R = T(\{\boldsymbol{x}_i\}_{i \in R}, O_m)$ where $[\boldsymbol{P}_m]_{i,n}$ is the confidence score of class n for data point i and R contains the test data points. Then we give the overall stacking procedure, including the test phase, with L-fold internal cross validation in Algorithm 1.

3.4 Linear Combination Types

In this section, we describe and analyze three combination types, namely *weighted sum* rule (WS), *class-dependent weighted sum* rule (CWS) and *linear stacked generalization* (LSG) where LSG is already defined in (3.1).

Algorithm 1 Training and test procedure of stacked generalization with internal CV

Receive training data: $\boldsymbol{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{I}$, Indices: $Q = \{1, \ldots, I\}$.
Receive base classifier types and parameters: C_m, Q_m, F_m, Π_m for $m = 1, \ldots, M$
Set parameter L ▷ Number of stacks for internal CV
Split the set Q into non-overlapping almost equal sized subsets $\{Q^1, \ldots, Q^L\}$
for $m = 1, \ldots, M$ **do**
    for $l = 1, \ldots, L$ **do**
        $O_m^l = C_m(\boldsymbol{D}; Q_m^{-l}, F_m, \Pi_m)$ where $Q_m^{-l} = \{Q \backslash Q^l\} \cap Q_m$ ▷ Train base classifier
        $\boldsymbol{P}_m^l = T(\{\boldsymbol{x}_i\}_{i \in Q_m^l}, O_m^l)$, where $Q_m^l = Q_m \cap Q^l$ ▷ Obtain the posterior scores of stack l
    end for
    $\boldsymbol{P}_m = [\boldsymbol{P}_m^{1\,T}, \ldots, \boldsymbol{P}_m^{L\,T}]^T$ ▷ Concatenate posterior scores of classifier m
end for
$\boldsymbol{F} = [\boldsymbol{P}_1, \ldots, \boldsymbol{P}_M]$ ▷ Concatenate posterior scores ($\boldsymbol{F} = [\boldsymbol{f}_1, \ldots, \boldsymbol{f}_I]^T$)
Learn the combiner g using $\{(\boldsymbol{f}_i, y_i)\}_{i=1}^{I}$
for $m = 1, \ldots, M$ **do** ▷ Train base classifiers for test
    $O_m = C_m(\boldsymbol{D}; Q_m, F_m, \Pi_m)$ ▷ Train base classifier
end for
Receive test data: $\boldsymbol{D}' = \{\boldsymbol{x}_i'\}_{i=1}^{I'}$ ▷ Started test phase
for $m = 1, \ldots, M$ **do**
    $\boldsymbol{P}_m' = T(\boldsymbol{D}', O_m)$ ▷ Obtain posterior scores with base classifiers
end for
$\boldsymbol{F}' = [\boldsymbol{P}_1', \ldots, \boldsymbol{P}_M']$
$\boldsymbol{r}_i = g(\boldsymbol{f}_i')$ for $i = 1, \ldots, I'$ ▷ Combine the posterior scores using combiner g

3.4.1 Weighted Sum Rule

In this type of combination, each classifier is given a weight, so there are totally M different weights. Let u_m be the weight of classifier m, then the final score of class n is estimated as follows:

$$r^n = \sum_{m=1}^{M} u_m p_m^n = \boldsymbol{u}^T \boldsymbol{f}^n \quad , \quad n = 1, \ldots, N, \tag{3.2}$$

where $\boldsymbol{f}^n$ contains the scores of class n: $\boldsymbol{f}^n = [p_1^n, \ldots, p_M^n]^T$ and $\boldsymbol{u} = [u_1, \ldots, u_M]^T$. An illustration of WS combination for $M = 2$ and $N = 3$ is given in Figure 3.2. For the framework given in (3.1), WS combination can be obtained by letting $\mathbf{b} = 0$ and $\boldsymbol{W}$ be the concatenation of constant diagonal matrices:

$$\boldsymbol{W} = [u_1 \boldsymbol{I}_N | \ldots | u_M \boldsymbol{I}_N], \tag{3.3}$$

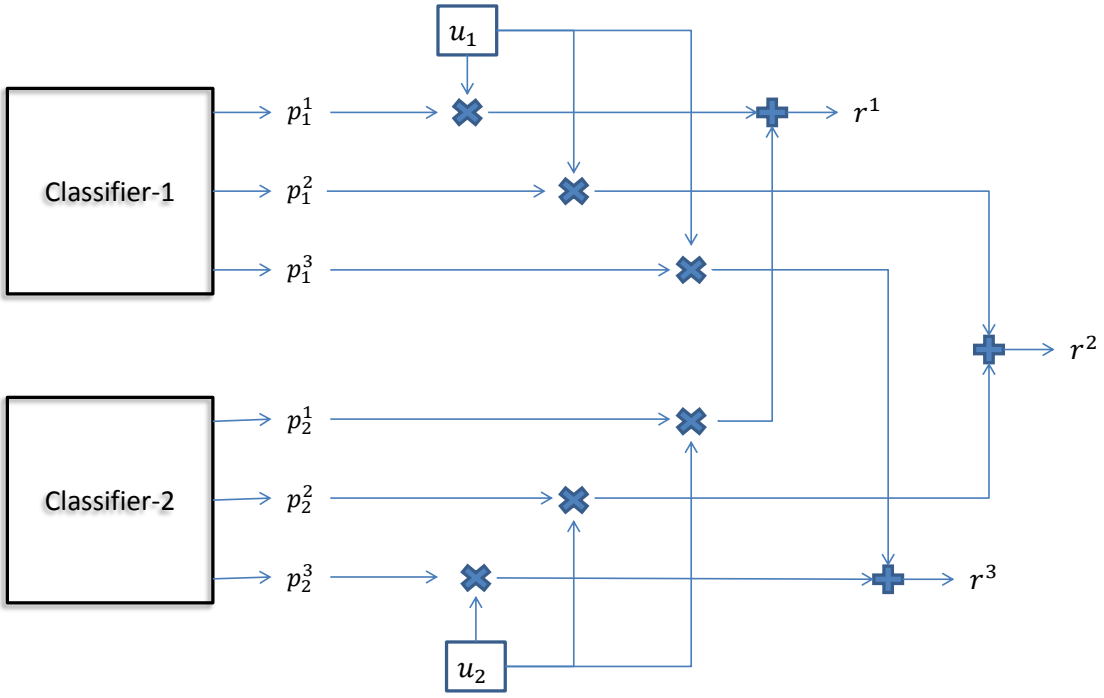


FIGURE 3.2: Illustration of WS combination for $M = 2$ and $N = 3$.

where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. We expect to obtain higher weights for stronger base classifiers after learning the weights from the database.

3.4.2 Class-Dependent Weighted Sum Rule

The performances of base classifiers may differ for different classes and it may be better to use a different weight distribution for each class. We call this type of combination CWS rule. Let v_m^n be the weight of classifier m for class n, then the final score of class n is estimated as follows:

$$r^n = \sum_{m=1}^{M} v_m^n p_m^n = \boldsymbol{v}_n^T \boldsymbol{f}^n \quad , \quad n = 1, \ldots, N, \tag{3.4}$$

where $\boldsymbol{v}_n = [v_1^n, \ldots, v_M^n]^T$. There are MN parameters in a CWS combiner. An illustration of CWS combination for $M = 2$ and $N = 3$ is given in Figure 3.3. For the framework given in (3.1), CWS combination can be obtained by letting $\mathbf{b} = 0$ and $\boldsymbol{W}$ to be the concatenation of diagonal matrices; but unlike in WS, diagonals are not constant:

$$\boldsymbol{W} = [\boldsymbol{W}_1 | \boldsymbol{W}_2 | \ldots | \boldsymbol{W}_M], \tag{3.5}$$

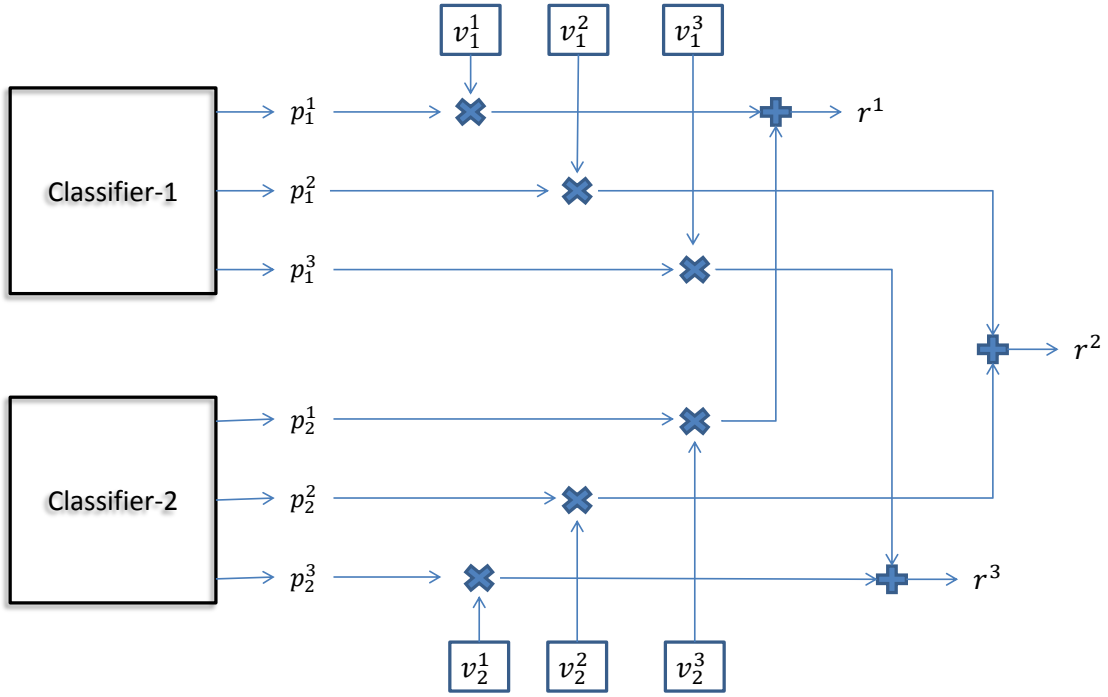


FIGURE 3.3: Illustration of CWS combination for $M = 2$ and $N = 3$.

where $\boldsymbol{W}_m \in \mathbb{R}^{N \times N}$ are diagonal for $m = 1, \ldots, M$.

3.4.3 Linear Stacked Generalization

This type of combination is the most general form of supervised linear combinations and is already defined in (3.1). With LSG, the score of class n is estimated as follows:

$$r^n = \boldsymbol{w}_n^T \boldsymbol{f} + b_n \quad , \quad n = 1, \ldots, N, \tag{3.6}$$

where $\boldsymbol{w}_n \in \mathbb{R}^{MN}$ is the n^{th} row of $\boldsymbol{W}$ and b_n is the n^{th} element of $\mathbf{b}$. LSG can be interpreted as feeding the base classifiers' outputs to a linear multiclass classifier as a new set of features. This type of combination may result in overfitting the database and may yield lower accuracy than WS and CWS combinations when there is not enough training data. From this point of view, WS and CWS combination can be treated as regularized versions of LSG. A crucial disadvantage of LSG is that the number of parameters to be learned is $MN^2 + N$ which will result in a long training period.

There is not a single superior one among these three combination types since results are shown to be data dependent [21]. A convenient way of choosing the combination type is selecting the one that gives the best performance in cross-validation.

3.5 Previous Stacking Algorithms

After obtaining level-1 data, there are two main problems remaining for a linear combination: (1.) Which type of combination method should be used? (2.) Given a combination type, how should we learn the parameters of the combiner? For the former problem, Ueda [20] defined three linear combination types namely type-1, type-2 and type-3; for which, we use the descriptive names weighted sum (WS), class-dependent weighted sum (CWS) and linear stacked generalization (LSG), respectively and investigate all of them. LSG is used in [22, 23], and CWS combination is proposed in [18]. For the second main problem described above, Ting & Witten proposed a multi-response linear regression algorithm for learning the weights [18]. Ueda in [20] proposed using minimum classification error (MCE) criterion for estimating optimal weights, which increased the accuracies. MCE criterion is an approximation to the zero-one loss function which is not convex, so finding a global optimizer is not always possible. Ueda derived algorithms for different types of combinations with MCE loss using stochastic gradient methods. Both of these studies ignored "regularization" which has a huge effect on the performance, especially if the number of base classifiers is large. Reid & Grudic in [24] regularized the standard linear least squares estimation of the weights with CWS and improved the performance of stacking. They applied l_2 norm penalty, l_1 norm penalty and combination of the two (elastic net regression).

Another issue, recently addressed in [25], is combination with a sparse weight vector so that we do not use all of the ensemble. Since we do not have to use classifiers which have zero weight on the test phase, overall test time will be much less. Zhang formulated this problem as a linear programming problem for only the WS combination type [25]. Reid used l_1 norm regularization for CWS combination [24].

Chapter 4

Max-Margin Stacking & Sparse Regularization

4.1 Introduction

The main principle of stacked generalization is using a second-level generalizer to combine the outputs of base classifiers in an ensemble. In this chapter, we investigate and compare different combination types under the stacking framework; namely weighted sum (WS), class-dependent weighted sum (CWS) and linear stacked generalization (LSG). For learning the weights, we propose using regularized empirical risk minimization with the hinge loss. In addition, we propose using group sparsity for regularization to facilitate classifier selection. We performed experiments using two different ensemble setups with differing diversities on 8 real-world datasets. Results show the power of regularized learning with the hinge loss function. Using sparse regularization, we are able to reduce the number of selected classifiers of the diverse ensemble without sacrificing accuracy. With the non-diverse ensembles, we even gain accuracy on average by using group sparse regularization. [1]

[1] Preliminary works of this chapter are published at International Conference on Pattern Recognition, 2010 [21] and 18^{th} IEEE conference on Signal Processing and Communication Applications [26].

4.2 Learning the Combiner

We use the regularized empirical risk minimization (RERM) framework [27] for learning the weights. In this framework, learning is formulated as an unconstrained minimization problem and the objective function consists of a summation of empirical risk function over data instances and a regularization function. Empirical risk is obtained as a sum of "loss" values obtained from each sample. In general, we want to minimize the following objective function:

$$\phi(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{I} \sum_{i=1}^{I} \sum_{n=1}^{N} L(\boldsymbol{f}_i, y_i, n, \boldsymbol{w}_n) + \lambda R(\boldsymbol{W}). \tag{4.1}$$

where, L is the loss function. Different choices of loss functions and regularization functions correspond to different classifiers. Using the hinge loss function with l_2 norm regularization is equivalent to support vector machines (SVM). It has been shown in studies that the hinge loss function yields much better classification performance as compared to the least-squares (LS) loss function in general. Earlier classifier combination literature uses LS loss function [18, 23, 24], which is less favorable as compared to the hinge loss that we promote and use in this thesis. Least-squares loss function is as follows:

$$L(\boldsymbol{f}_i, y_i, n, \boldsymbol{w}) = (s(y_i, n) - \boldsymbol{f}_i^T \boldsymbol{w}_n - b_n)^2, \tag{4.2}$$

where $s(y_i, n) = 1$ if $y_i = n$, -1 otherwise and b_n is the n^{th} element of $\boldsymbol{b}$. Instead of the s function, we can use the $\delta(y_i, n)$ which is zero if $y_i \neq n$ instead of -1. LS loss function forces the true class' scores to be one and wrong classes' scores to be zero or -1. This problem can be seen as a regression problem. Using least-squares with l_2 regularization is equivalent to applying least-squares support vector machine (LS-SVM) [28] to the level-1 data.

As mentioned above, we promote to use the hinge loss function for the combiner. Using the hinge loss function with the l_2 norm regularization is equivalent to using Support Vector Machine classifier. SVMs were originally designed for binary classification and there are a lot of ongoing research on how to effectively extend it for multiclass classification. Current methods can be grouped into direct and indirect multiclass SVMS. Indirect methods construct several binary SVMs and combine them, whereas direct methods consider all classes at once. Two well-known indirect multiclass methods are

one-versus-one [29] and one-versus-all [30] methods. In one-versus-one method, we train a binary SVM using only datapoints of two particular classes for each class pair. Test phase is done by a max-wins voting strategy, in which each binary classifier assigns the instance to one of the two classes, then finally the class with most votes determines the instance classification. In one-versus-all method, a binary SVM is trained by treating one class as the positive class, and the rest of the classes as negative class. In test phase, an instance is tested with each binary SVM and it is assigned to the class which has the maximum score. Another method of indirect multiclass classification is an application of error correcting output codes (ECOC) to the multiclass classification problem [31].

Direct multiclass SVM methods try to solve one problem only [32–35]. We use the method defined by Crammer and Singer [33]. With this method, we find the linear separating hyper-plane that maximizes the margin between true class and the most offending wrong class. When we apply this idea to our problem, we obtain the following unconstrained minimization problem for LSG:

$$\phi_{LSG}(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{I} \sum_{i=1}^{I} (1 - r_i^{y_i} + \max_{n \neq y_i} r_i^n)_+ + \lambda R_{LSG}(\boldsymbol{W}), \tag{4.3}$$

where $R_{LSG}(\boldsymbol{W})$ is the regularization function, $(x)_+ = \max(0, x)$ and r_i^n is the posterior score of data instance i for class n:

$$r_i^n = \boldsymbol{w}_n^T \boldsymbol{f}_i + b_n. \tag{4.4}$$

$\lambda \in \mathbb{R}$ in (4.3) is the regularization parameter which is usually learned by cross validation. The objective function given in (4.3) encourages the distance between the true class' score and the most offending wrong class' score to be larger than one. A conventional regularization function is the Frobenius norm of $\boldsymbol{W}$:

$$R_{LSG}(\boldsymbol{W}) = ||\boldsymbol{W}||_F^2 = \sum_{n=1}^{N} ||\boldsymbol{w}_n||_2^2, \tag{4.5}$$

Equation (4.3) is given for LSG but it can be modified for other types of combinations using the unifying framework described in [21]. But we also give objective functions for

WS and CWS explicitly. The objective function for WS is as follows:

$$\phi_{WS}(\boldsymbol{u}) = \frac{1}{I}\sum_{i=1}^{I}(1 - \boldsymbol{u}^T \boldsymbol{f}_i^{y_i} + \max_{n \neq y_i}(\boldsymbol{u}^T \boldsymbol{f}_i^n))_+ + \lambda R_{WS}(\boldsymbol{u}). \tag{4.6}$$

For regularization, we use the l_2 norm of $\boldsymbol{u}$: $R_{WS} = ||\boldsymbol{u}||_2^2$. For CWS, we have the following objective function:

$$\phi_{CWS}(\boldsymbol{V}) = \frac{1}{I}\sum_{i=1}^{I}(1 - \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i} + \max_{n \neq y_i}(\boldsymbol{v}_n^T \boldsymbol{f}_i^n))_+ + \lambda R_{CWS}(\boldsymbol{V}), \tag{4.7}$$

where $\boldsymbol{V} \in \mathbb{R}^{M \times N}$ contains the weights for different classes: $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]$. As for LSG, conventional regularization function for CWS is the Frobenious norm of $\boldsymbol{V}$: $R_{CWS}(\boldsymbol{V}) = ||\boldsymbol{V}||_F^2$.

4.3 Unifying Framework

Another issue we address in this thesis is the solution algorithms for the objective functions defined in the previous chapter. For the LSG combination defined in (4.3), state-of-the-art SVM solutions can be used [36]. However, for the WS and CWS combinations, there is no solution available. One possible solution to this problem might be modifying these algorithms. Another possible choice is using a tying matrix in the objective function of LSG (4.3) to obtain WS and CWS combinations as described in [21]. Letting $\boldsymbol{w}_n = \boldsymbol{A}_n \boldsymbol{u}$ and $b_n = 0$ in the objective function of LSG leads to WS combination, where $\boldsymbol{A}_n \in \mathbb{R}^{MN \times M}$ is the fixed tying matrix associated with class n. Letting $\boldsymbol{w}_n = \boldsymbol{A}_n \tilde{\boldsymbol{v}}$ and again $b_n = 0$ leads to CWS combination where $\boldsymbol{A}_n \in \mathbb{R}^{MN \times MN}$ and $\tilde{\boldsymbol{v}}$ is the concatenation of the weights of CWS combination: $\tilde{\boldsymbol{v}} = [\boldsymbol{v}_1^T, \ldots, \boldsymbol{v}_N^T]^T$. The tying matrices of WS and CWS combinations for a problem with $N = 3$ and $M = 2$ are given in Figures 4.1(a) and 4.1(b) respectively. Once we obtain an optimization algorithm for LSG combination, by incorporating these tying matrices into the algorithms, we can obtain solutions of WS and CWS combinations.

We give an optimization algorithm for the CWS combination in Chapter 6, but the unifying framework described here is applicable to different loss functions, and once an algorithm for the LSG combination type is found, the unifying framework can be used to adapt the algorithm to obtain solutions for the WS and CWS combinations.

$$\boldsymbol{A}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \boldsymbol{A}_3 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

(a) WS combination

$$\boldsymbol{A}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \widetilde{\boldsymbol{v}} = \begin{bmatrix} v_1^1 \\ v_2^1 \\ v_1^2 \\ v_2^2 \\ v_1^3 \\ v_2^3 \end{bmatrix}$$

(b) CWS combination

FIGURE 4.1: Tying matrices A_n and unique weights of WS and CWS combination for $N = 3$ and $M = 2$.

4.4 Sparse Regularization

In this section, we define a set of regularization functions for enforcing sparsity on the weights so that the resulting combiner will not use all the base classifiers leading to a shorter test time. This method can be seen as a classifier selection algorithm, but here classifiers are selected automatically and we cannot determine the number of selected classifiers beforehand. But we can lower this number by increasing the weight of the regularization function (λ), and vice versa. With sparse regularization, λ has two main effects on the resulting combiner. First, it will determine how much the combiner should fit the data. Decreasing λ results in more fitting the training data and decreasing it too much results in overfitting, on the other hand, increasing it too much prevents the combiner to learn from the data and the accuracy drops dramatically. Secondly, as mentioned before, it will determine the number of selected classifiers. As λ increases, the number of selected classifiers decreases.

4.4.1 Regularization with the l_1 Norm

The most successful approach for inducing sparsity is using the l_1 norm of the weight vector for WS [25]. For CWS and LSG, in which the combiner consists of matrices, we can concatenate the weights in a vector and take the l_1 norm or equivalently we can sum the l_1 norms of the rows (or columns) of the weight matrices. We have the following

sparse regularization functions for WS, CWS and LSG respectively:

$$R_{WS}(\boldsymbol{u}) = ||\boldsymbol{u}||_1, \tag{4.8}$$

$$R_{CWS}(\boldsymbol{V}) = ||\boldsymbol{V}||_{1,1} = \sum_{n=1}^{N} ||\boldsymbol{v}_n||_1, \tag{4.9}$$

$$R_{LSG}(\boldsymbol{W}) = ||\boldsymbol{W}||_{1,1} = \sum_{n=1}^{N} ||\boldsymbol{w}_n||_1. \tag{4.10}$$

If all weights of a classifier are zero, that classifier will be eliminated and we do not have to use that base classifier for a test instance, so that testing will be faster. But the problem with l_1-norm regularizations for CWS and LSG is that we are not able to use all the information from a selected base classifier, because a classifier may receive both zero and non-zero weights. To overcome this problem, we propose to use group sparsity, as explained in the next section.

4.4.2 Regularization with Group Sparsity

We define another set of regularization functions which are embedded by group sparsity [37] for LSG and CWS to enforce classifier selection. The main principle of group sparsity is enforcing all elements that belong to a group to be zero altogether. Grouping of the elements are done before learning. In classifier combination, posterior scores obtained from each base classifier form a group. The following regularization function yields group sparsity for LSG:

$$R_{LSG}(\boldsymbol{W}) = \sum_{m=1}^{M} ||\boldsymbol{W}_m||_F. \tag{4.11}$$

For CWS, we use the following regularization:

$$R_{CWS}(\boldsymbol{V}) = ||\boldsymbol{V}||_{1,2} = \sum_{m=1}^{M} ||\boldsymbol{v}^m||_2, \tag{4.12}$$

where $\boldsymbol{v}^m$ is the m^{th} row of $\boldsymbol{V}$, so it contains the weights of the classifier m. After the learning process, the elements of $\boldsymbol{v}^m$ for any m are either all zero or all non-zero. This leads to better performance than l_1 regularization for automatic classifier selection, as we show in Section 4.6. In the next section, we describe the setup of the experiments.

4.5 Experimental Setups

We have performed extensive experiments in eight real-world datasets from the UCI repository [38]. For a summary of the characteristics of the datasets, see Table 4.1. In order to obtain statistically significant results, we applied 5x2 cross-validation [39] which is based on 5 iterations of 2-fold cross-validation (CV). In this method, for each CV, data are randomly split into two stacks as training and testing resulting in overall 10 stacks for each database.

We constructed two ensembles which differ in their diversity. In the first ensemble, we construct 10 different subsets randomly which contain 80% of the original data. Then, 13 different classifiers are trained with each subset resulting in a total of 130 base classifiers. We used PR-Tools [40] and Libsvm toolbox [41] for obtaining the base classifiers. These 13 different classifiers are: normal densities based linear classifier, normal densities based quadratic classifier, nearest mean classifier, k-nearest neighbor classifier, polynomial classifier, general kernel/dissimilarity based classification, normal densities based classifier with independent features, Parzen classifier, binary decision tree classifier, linear perceptron, SVM with linear kernel, polynomial kernel, and radial basis function (RBF) kernel. We used default parameters of the toolboxes. In the second ensemble setup, we trained a total of 154 SVM's with different kernel functions and parameters. Latter method produces less diverse base classifiers with respect to the former one. Training data of the combiner is obtained by 4-fold internal CV. For each stack in 5×2 CV, 2-fold CV is used to obtain the optimal λ in the regularization function, i.e., λ which gives the best average accuracy in CV [2]. For the minimization of the objective functions, we used the CVX-toolbox [42]. We use the Wilcoxon signed-rank test for identifying the statistical significance of the results with one-tailed significant level $\alpha = 0.05$ [43].

4.6 Results

First, we investigate the performance of regularized learning of the weights with the hinge loss compared to the conventional least squares loss [24] and the multi-response linear regression (MLR) method which does not contain regularization [18] with the

[2] We searched for λ in $\{10^{-11}, 10^{-9}, 10^{-7}, 10^{-5}, 10^{-3}, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 10\}$

TABLE 4.1: Properties of the data sets used in the experiments

DB	# of Instances	# of classes	# of features
Segment [3]	2310	7	19
Waveform [4]	5000	3	21
Robot [5]	5456	4	24
Statlog [6]	846	4	18
Vowel [7]	990	11	10
Wine	178	3	13
Yeast	1484	9	8
Steel [8]	1941	7	27

diverse ensemble setup described in section 4.5. It should be noted that results shown here and in [18, 24] are not directly comparable since construction of the ensembles is different. Error percentages of our method (Hinge Loss with l_2 regularization), least squares method, and MLR method for WS, CWS and LSG are given in Table 4.2. Results for the simple sum rule, which is equivalent to using equal weights in the WS, are also given in the column titled *EW*. The first entries in the boxes are the means of error percentages over 5×2 CV stacks and the second entries are the standard deviations. Star symbols (*) under the hinge loss column indicate that results of the hinge loss function are significantly different from the results of the least squares loss function with the corresponding combination type, i.e., WS, CWS, or LSG.

TABLE 4.2: Error percentages in the diverse ensemble setup (*mean* $\pm$ *standard deviation*).

DB	Hinge Loss with l_2 regularization			Least Squares Loss with l_2 regularization			MLR			EW
	WS	CWS	LSG	WS	CWS	LSG	WS	CWS	LSG	
Segment	5.02 ± 0.88 *	**3.53 ± 0.99**	3.60 ± 1.05	6.34 ± 0.78	3.54 ± 0.82	3.57 ± 0.96	7.20 ± 1.02	6.66 ± 6.64	61.28 ± 9.35	7.37 ± 1.03
Waveform	13.20 ± 0.69	13.08 ± 0.76	**13.05 ± 0.65** *	13.19 ± 0.73	13.17 ± 0.72	13.18 ± 0.69	13.33 ± 0.68	14.10 ± 0.56	18.40 ± 7.06	14.17 ± 0.60
Robot	3.95 ± 0.42 *	**2.53 ± 0.28**	2.61 ± 0.28 *	5.29 ± 0.61	2.55 ± 0.30	2.53 ± 0.31	5.05 ± 0.62	2.58 ± 0.30	3.19 ± 0.49	18.58 ± 0.61
Statlog	16.34 ± 1.15	**16.12 ± 1.94** *	16.36 ± 1.67	16.78 ± 1.62	16.74 ± 1.91	16.88 ± 1.71	17.73 ± 2.11	58.01 ± 15.38	75.72 ± 6.18	23.03 ± 2.33
Vowel	13.84 ± 2.73 *	6.97 ± 1.73	**6.32 ± 1.99**	13.90 ± 2.63	6.42 ± 2.06	6.46 ± 2.22	17.15 ± 2.31	10.08 ± 1.75	9.76 ± 1.14	14.53 ± 3.30
Wine	1.57 ± 1.09	**1.01 ± 1.45** *	1.69 ± 1.32	2.36 ± 1.54	2.13 ± 2.21	2.13 ± 1.79	3.71 ± 2.31	8.20 ± 16.19	2.47 ± 1.66	2.81 ± 1.52
Yeast	40.36 ± 1.21	40.63 ± 1.21	40.32 ± 1.19	**40.26 ± 1.06**	40.62 ± 1.44	40.94 ± 1.70	41.05 ± 1.04	53.11 ± 6.88	74.45 ± 6.42	40.26 ± 1.10
Steel	29.85 ± 1.86 *	**27.37 ± 1.18**	27.41 ± 1.22	30.73 ± 2.02	27.52 ± 1.17	27.64 ± 1.47	30.35 ± 1.34	51.40 ± 14.66	77.12 ± 7.82	31.57 ± 2.07

For seven datasets, the lowest error means are obtained with the hinge loss function and for one dataset lowest error mean is obtained with the least-squares loss function. On all datasets, MLR method results in higher error percentages compared to other methods, and this shows the power of regularized learning, especially if the number of

[3] The full name of *Segment* dataset is *"Image Segmentation"*
[4] The full name of *Waveform* dataset is *"Waveform Database Generator (Version 1)"*
[5] The full name of *Robot* dataset is *"Wall-Following Robot Navigation Data"*
[6] The full name of *Statlog* dataset is *"Statlog (Vehicle Silhouettes)"*
[7] The full name of *Vowel* dataset is *"Connectionist Bench (Vowel Recognition - Deterding Data)"*
[8] The full name of *Steel* dataset is *"Steel Plates Faults"*

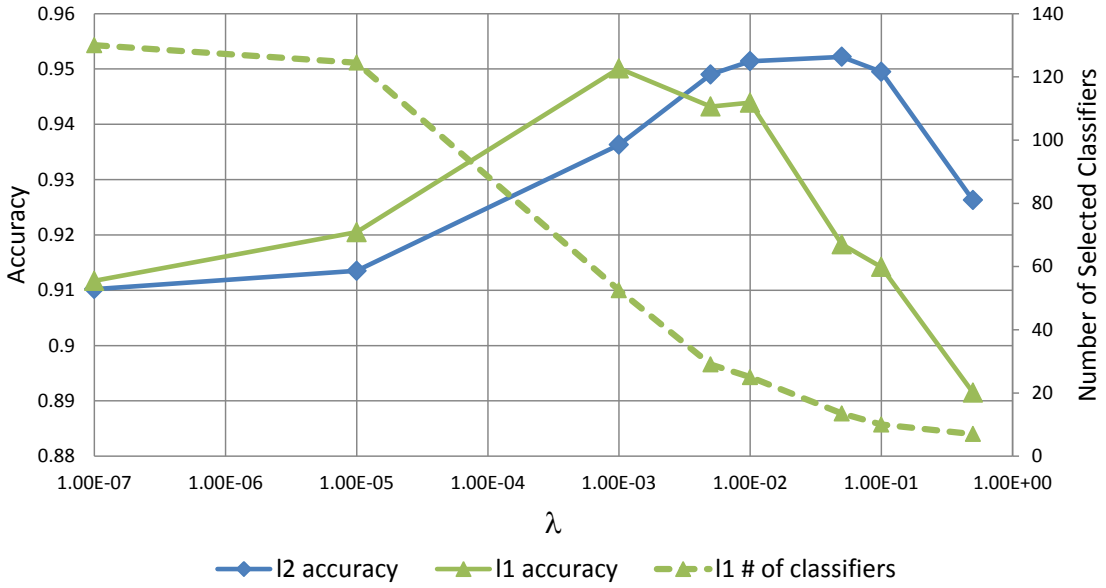


FIGURE 4.2: Accuracy and number of selected classifiers vs. λ for WS combination of Robot data in the diverse ensemble setup.

base classifiers is high. It should be noted that in [18], 3 base classifiers are used and here we use 130 base classifiers.

We also investigated the performance of sparse regularization with the hinge loss function. We used two different ensemble setups described in the beginning of this section. Regularization parameter λ given in the objective functions (4.3,4.6,4.7) is an important parameter and if we minimize the objective functions also over λ, the combiner will overfit the training data, which will result in poor generalization performance. Therefore, we used 2-fold cross-validation to learn the optimal parameter. We plot the relation of λ with accuracies and the number of selected classifiers for different regularizations with WS, CWS and LSG for the *Robot* dataset in Figures 4.2, 4.4 and 4.6 respectively.
In these figures, dashed lines correspond to the number of selected classifiers and solid lines correspond to the accuracies. The $l_1 - l_2$ label represents group sparsity. In all sparse regularizations, the best accuracies are obtained when most of the base classifiers are eliminated. For all regularizations, accuracies make a peak at λ values between 0.001 and 0.1. For l_1 norm regularization, accuracies drop dramatically with a small increase in λ. However, with group sparsity regularization, accuracies remain high in a larger range for λ than that with the l_1 norm regularization. Thus the performance of l_1 regularization is more sensitive to the selection of λ. So we can say that the $l_1 - l_2$ norm regularization is more robust than the l_1 norm regularization. As the number of

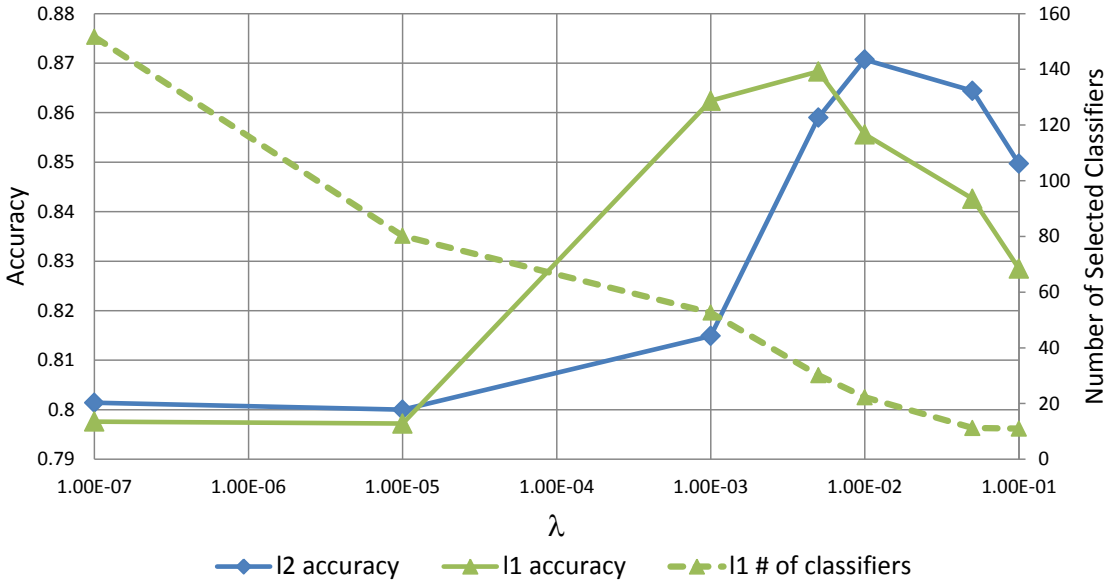


FIGURE 4.3: Accuracy and number of selected classifiers vs. λ for WS combination of Robot data in the non-diverse ensemble setup.

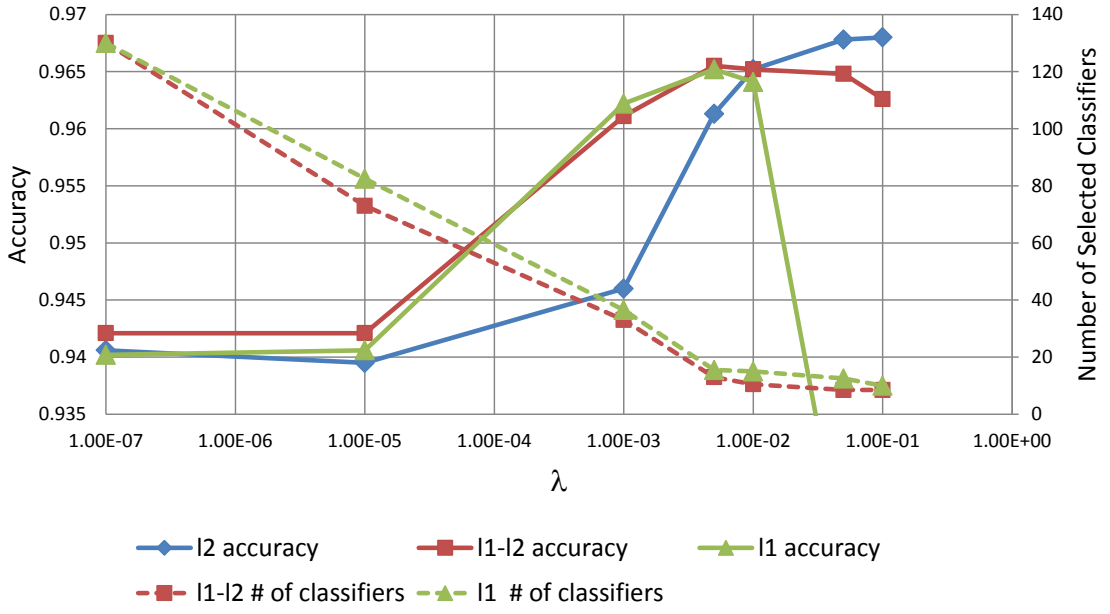


FIGURE 4.4: Accuracy and number of selected classifiers vs. λ for CWS combination of Robot data in the diverse ensemble setup.

selected classifiers decreases, accuracies increase for a large range of λ in general, but this increase in the accuracy cannot be attributed only to the classifier selection, because λ also determines how much the combiner should fit the data.

Next, we show the test results for all combination types with various regularization functions. Error percentages (mean $\pm$ standard deviation) are shown in Table 4.3 for

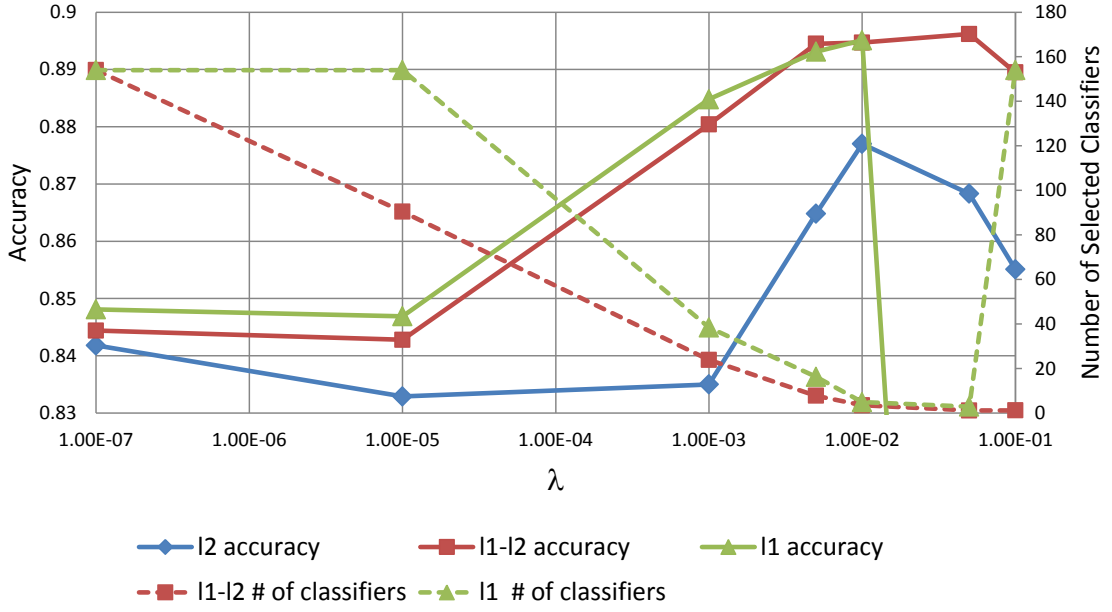


FIGURE 4.5: Accuracy and number of selected classifiers vs. λ for CWS combination of Robot data in the non-diverse ensemble setup.

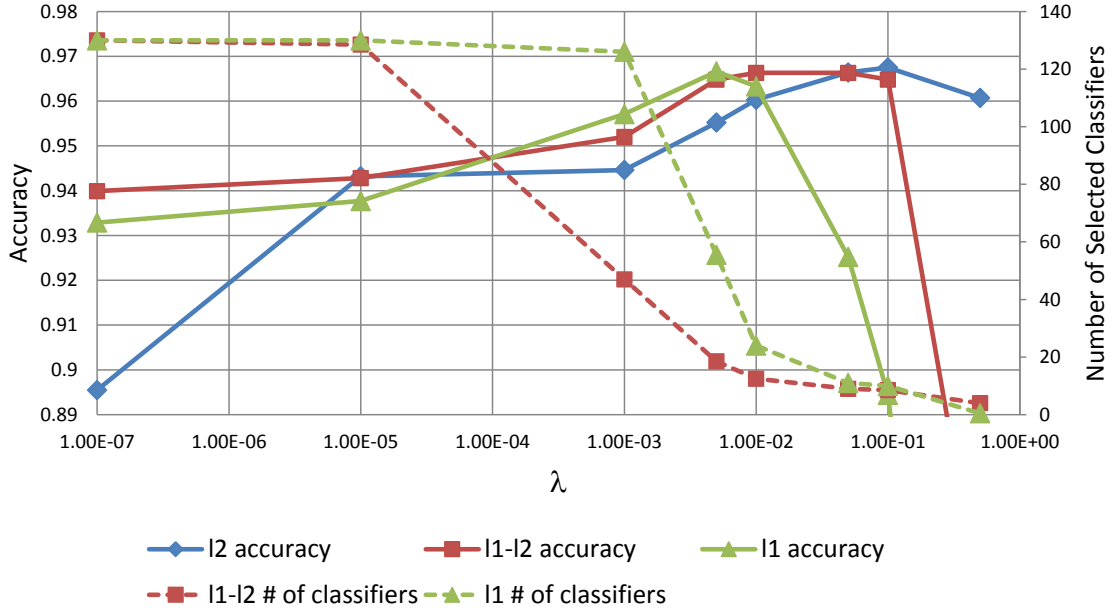


FIGURE 4.6: Accuracy and number of selected classifiers vs. λ for LSG combination of Robot data in the diverse ensemble setup.

the diverse ensemble setup and corresponding number of selected classifiers are shown in Table 4.4. In the significance column, denoted by SIG, the letters "a,b,c,d,e," denote that the performances between l_2 - l_1 for WS, l_2-$l_1 - l_2$, l_1-$l_1 - l_2$ for CWS and l_1-$l_1 - l_2$ for LSG are statistically significant respectively.

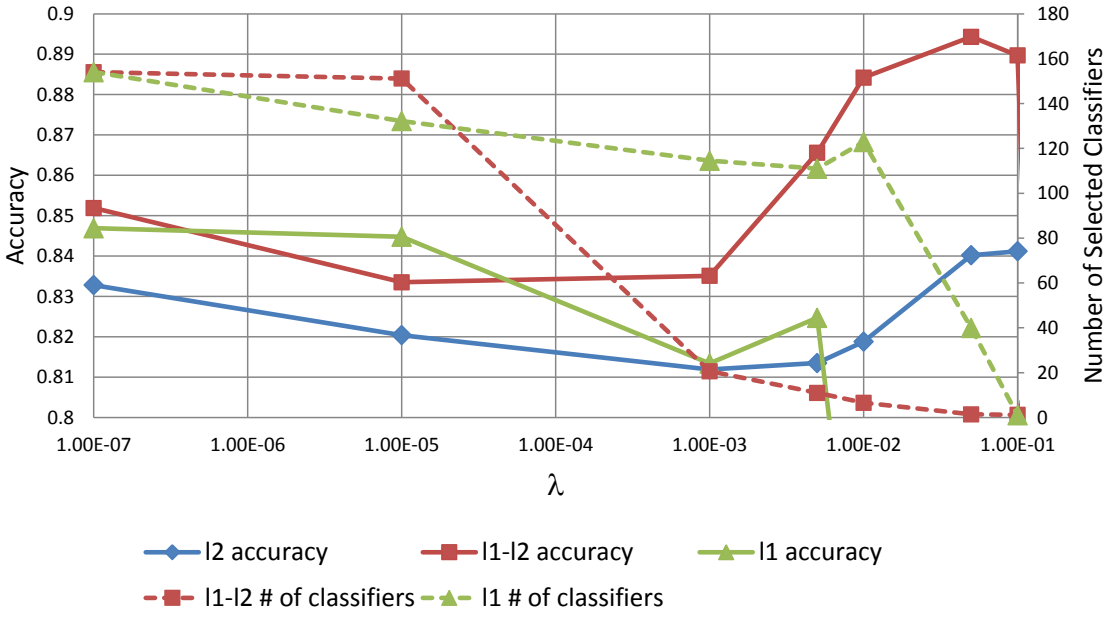


FIGURE 4.7: Accuracy and number of selected classifiers vs. λ for LSG combination of Robot data in the non-diverse ensemble setup.

In general, we are able to use much less base classifiers with sparse regularizations with the cost of a small decrease in the accuracies. For LSG, average error percentage of group sparsity is a little less than that of the l_1 norm regularization. But the number of selected base classifiers is much less. So if classifier selection is desired, we suggest to use either CWS or LSG combination with $l_1 - l_2$ regularization. If training time is also crucial, CWS with $l_1 - l_2$ regularization seems to be the best option.

Error percentages and number of selected classifiers for the non-diverse ensembles are given in Tables 4.5 and 4.6 respectively. We should note that the results in these tables are from an earlier experiment and they are obtained using the l_2 norm of the weight vector for regularization rather than the square of the l_2 norm; however, we expect the results to be similar. With the non-diverse ensembles we are even able to increase the accuracy with much less number of base classifiers with sparse regularization in CWS and LSG. On average, $l_1 - l_2$ regularization results in lower error percentages for both CWS and LSG, but the results are not statistically significant. But, the number of selected classifiers is much less with $l_1 - l_2$ regularization than that of l_1 regularization. Except the *statlog* dataset, the lowest error percentages are obtained with the sparse combinations with much less base classifiers than that of l_2 regularization which uses 154 base classifiers. If we compare different combination types with the l_2 norm, on average

we see that, unlike in the diverse ensemble setup, WS and CWS outperforms LSG in four databases. We can conclude that if the posterior scores obtained from base classifiers are correlated, non-complex combiners are more powerful since complex combiners may result in overfitting.

4.7 Conclusion

In this chapter, we suggested using the hinge loss function with regularization to learn the parameters (or weights) of linear combiners in stacked generalization. We are able to obtain better accuracies with the hinge loss function than conventional least-squares estimation of the weights. Results also indicate the importance of regularized learning of the weights. We also proposed $l_1 - l_2$ norm regularization (or group sparsity) to obtain a reduced number of base classifiers so that the test time is shortened. Results indicate that we can use smaller number of base classifiers with a small sacrifice in the accuracy with the diverse ensemble. We show that $l_1 - l_2$ regularization outperforms l_1 regularization in terms of both accuracy and the number of selected classifiers. With the non-diverse ensemble setup, we even obtain better accuracies using sparse regularizations. If training time is crucial, we suggest using the CWS type combination. And if test time is important, we suggest using group sparsity regularization.

TABLE 4.3: Error percentages with the diverse ensemble setup (*mean* ± *standard deviation*). Bold values are the lowest error percentages of sparse regularizations (l_1 or $l_1 - l_2$ regularizations)

DB	WS		CWS			LSG			EW	SIG
	l_2	l_1	l_2	l_1	$l_1 - l_2$	l_2	l_1	$l_1 - l_2$		
Segment	5.02 ± 0.88	4.90 ± 0.99	3.53 ± 0.99	3.62 ± 0.62	3.74 ± 0.40	3.60 ± 1.05	3.79 ± 1.05	**3.29 ± 0.55**	7.37 ± 1.03	
Waveform	13.20 ± 0.69	13.38 ± 0.70	13.08 ± 0.76	13.46 ± 0.74	13.42 ± 0.76	13.05 ± 0.65	13.33 ± 0.71	**13.24 ± 0.64**	14.17 ± 0.60	bd
Robot	3.95 ± 0.42	4.00 ± 0.38	2.53 ± 0.28	2.57 ± 0.35	**2.49 ± 0.33**	2.61 ± 0.28	2.54 ± 0.35	2.52 ± 0.32	18.58 ± 0.61	c
Statlog	16.34 ± 1.15	**17.19 ± 1.63**	16.12 ± 1.94	17.45 ± 1.74	17.33 ± 1.42	16.36 ± 1.67	17.40 ± 1.34	17.45 ± 1.51	23.03 ± 2.33	abd
Vowel	13.84 ± 2.73	14.40 ± 2.27	6.97 ± 1.73	7.62 ± 2.02	7.17 ± 1.50	6.32 ± 1.99	**6.18 ± 1.19**	6.79 ± 1.17	14.53 ± 3.30	a
Wine	1.57 ± 1.09	2.13 ± 1.63	1.01 ± 1.45	2.25 ± 1.18	**1.91 ± 1.30**	1.69 ± 1.32	2.25 ± 1.59	2.36 ± 1.54	2.81 ± 1.52	d
Yeast	40.36 ± 1.21	**40.38 ± 1.06**	40.63 ± 1.21	42.53 ± 4.42	41.19 ± 1.57	40.32 ± 1.19	48.09 ± 18.30	41.67 ± 1.31	40.26 ± 1.10	d
Steel	29.85 ± 1.86	30.00 ± 2.61	27.37 ± 1.18	28.31 ± 1.39	**27.41 ± 1.21**	27.41 ± 1.22	28.09 ± 1.03	27.50 ± 1.24	31.57 ± 2.07	c

TABLE 4.4: Number of selected classifiers with the diverse ensemble setup out of 130 (*mean* ± *standard deviation*).

DB	WS	CWS		LSG	
	l_1	l_1	$l_1 - l_2$	l_1	$l_1 - l_2$
Segment	**21.50 ± 4.62**	63.50 ± 25.72	30.80 ± 34.92	97.40 ± 24.40	80.40 ± 14.93
Waveform	36.60 ± 49.44	23.30 ± 37.59	47.00 ± 57.31	**11.20 ± 2.30**	12.10 ± 5.38
Robot	41.80 ± 9.02	18.60 ± 5.97	14.00 ± 4.55	18.50 ± 4.53	**13.30 ± 2.63**
Statlog	36.10 ± 34.75	14.30 ± 10.85	49.20 ± 56.13	30.60 ± 36.31	**11.20 ± 12.42**
Vowel	108.90 ± 44.48	37.80 ± 32.62	57.30 ± 62.64	128.00 ± 6.32	**13.80 ± 3.99**
Wine	130.00 ± 0.00	121.30 ± 18.60	117.10 ± 40.44	93.50 ± 58.86	**91.60 ± 61.83**
Yeast	119.10 ± 34.47	121.00 ± 28.46	40.40 ± 47.33	130.00 ± 0.00	**9.80 ± 3.46**
Steel	41.90 ± 32.05	42.10 ± 6.85	35.30 ± 8.10	51.00 ± 16.62	**35.20 ± 11.93**

TABLE 4.5: Error percentages with the non-diverse ensemble setup(*mean* ± *standard deviation*). Bold values are the lowest error percentages of sparse regularizations (l_1 or $l_1 - l_2$ regularizations).

DB	WS		CWS			LSG			EW	SIG
	l_2	l_1	l_2	l_1	$l_1 - l_2$	l_2	l_1	$l_1 - l_2$		
Segment	4.35 ± 0.66	4.49 ± 0.71	4.30 ± 0.71	4.21 ± 0.80	4.33 ± 0.74	5.11 ± 0.90	9.78 ± 17.11	4.35 ± 0.75	11.37 ± 0.74	ad
Waveform	13.23 ± 0.73	13.12 ± 0.81	13.30 ± 0.80	13.33 ± 0.75	13.24 ± 0.78	13.22 ± 0.76	13.20 ± 0.70	13.25 ± 0.68	13.34 ± 0.80	a
Robot	7.99 ± 0.66	7.98 ± 0.70	7.94 ± 0.63	8.13 ± 0.40	7.94 ± 0.49	8.01 ± 0.70	8.13 ± 0.55	7.99 ± 0.56	10.70 ± 0.47	ce
Statlog	18.65 ± 2.18	18.87 ± 2.05	18.49 ± 1.76	18.77 ± 1.74	19.24 ± 1.81	19.62 ± 1.15	19.17 ± 2.17	19.10 ± 1.56	28.32 ± 1.82	bc
Vowel	6.91 ± 2.34	9.88 ± 3.46	8.04 ± 2.12	6.34 ± 2.29	6.08 ± 2.37	8.57 ± 2.10	7.72 ± 2.24	6.10 ± 2.30	20.16 ± 2.74	abde
Wine	9.21 ± 1.82	8.88 ± 2.45	8.76 ± 1.97	8.65 ± 3.05	8.20 ± 3.63	14.94 ± 9.22	8.65 ± 2.31	8.99 ± 2.95	27.75 ± 7.57	d

TABLE 4.6: Number of selected classifiers out of 154 with the non-diverse ensemble setup.

DB	WS	CWS		LSG	
	l_1	l_1	$l_1 - l_2$	l_1	$l_1 - l_2$
Segment	29.40 ± 8.13	13.60 ± 6.93	8.40 ± 6.93	51.80 ± 70.80	**2.60 ± 2.67**
Waveform	32.40 ± 64.10	51.60 ± 70.98	95.60 ± 75.54	**5.10 ± 3.18**	36.70 ± 62.37
Robot	43.80 ± 16.19	30.60 ± 10.20	28.30 ± 16.57	20.50 ± 15.71	**13.70 ± 3.40**
Statlog	18.90 ± 9.46	14.50 ± 11.98	8.90 ± 9.64	25.30 ± 46.07	**7.80 ± 5.03**
Vowel	69.20 ± 59.27	8.70 ± 10.12	3.00 ± 2.83	125.70 ± 45.82	**1.40 ± 0.70**
Wine	65.00 ± 73.24	39.30 ± 60.96	**21.90 ± 46.09**	34.80 ± 62.84	78.90 ± 79.19

Chapter 5

Kernel Based Nonlinear Stacking

5.1 Introduction

Supervised linear combiners are shown to be strong combiners; but nonlinear combination methods are not well investigated in the literature. In the framework of stacking, posterior scores that are obtained from base classifiers are treated as a new set of features and the combiner tries to catch some patterns that are present in posterior scores. Linear combiners work with the assumption that classes are linearly separable in this new feature space. In this chapter, we question this assumption and do experiments with nonlinear combiners. We generalize the WS, CWS, and LSG combinations to be nonlinear, even though the descriptive names imply linear combinations. For WS combination, the combiner, $g : \mathbb{R}^M \rightarrow \mathbb{R}$, takes M different input arguments for each class and outputs the final score:

$$r^n = g(p_1^n, \ldots, p_M^n). \tag{5.1}$$

For the CWS combination, each class has its own combiner, $g_n : \mathbb{R}^M \rightarrow \mathbb{R}$:

$$r^n = g_n(p_1^n, \ldots, p_M^n). \tag{5.2}$$

For the LSG combination, each class has its own combiner, $g_n : \mathbb{R}^{MN} \rightarrow \mathbb{R}$, and each combiner takes MN different input arguments and outputs the final score:

$$r^n = g_n(p_1^1, \ldots, p_1^N, p_2^1, \ldots, p_2^N, \ldots\ldots, p_M^1, \ldots, p_M^N). \tag{5.3}$$

For CWS and LSG combination, the combiner functions g_n are estimated jointly for each n with the direct multiclass hinge loss described in (4.3) and (4.7). But with the least-squares loss function, we show that estimation of each g_n function can be done independently from each other.

Nonlinear version of the LSG combination can be obtained by directly applying the kernel trick to the posterior scores. The WS and CWS combinations are actually linear combinations of the posterior scores, but they can be also seen as linear boundaries that pass through the origin in the posterior score space. So, our aim is to find nonlinear boundaries that again pass through the origin in the same spaces. We obtain the nonlinear versions of WS and CWS combination types for least squares loss function by first transforming the dataset into binary datasets, then applying the kernel trick. This data transformation, however, is not valid for the direct multiclass hinge loss function given in (4.6) and (4.7). For regularization function we only use the l_2 norm, since the kernel trick is not applicable to the l_1 norm or group sparse regularization to our knowledge. [1]

5.2 WS combination using binary classifiers

For LSG combination, we use the following objective function of regularized empirical risk minimization framework:

$$\phi(\boldsymbol{w}) = \frac{1}{I}\sum_{i=1}^{I}\sum_{n=1}^{N} L(\boldsymbol{f}_i, y_i, n, \boldsymbol{w}) + \lambda R(\boldsymbol{w}). \tag{5.4}$$

where L is the loss function. Previous objective function given in (4.3) was given for the hinge loss with a modification. With this modification, instead of summing loss functions also over classes, we only considered the most offending wrong class. This modification prevents the data transformation method described below to be valid; but it is valid for the least squares loss function. Here, we derive the combination method for the least-squares loss function, which is as follows:

$$L(\boldsymbol{f}_i, y_i, n, \boldsymbol{w}) = (s(y_i, n) - \boldsymbol{f}_i^T \boldsymbol{w}_n)^2, \tag{5.5}$$

[1] A preliminary work of this chapter is published at 19^{th} IEEE conference on Signal Processing and Communication Applications [44].

where, $s(y_i, n) = 1$ if $y_i = n$, -1 otherwise. Using the unifying framework defined in Section 4.3, the LS loss function for WS combination becomes:

$$L(\boldsymbol{f}_i, y_i, n, \boldsymbol{w}) = (s(y_i, n) - \boldsymbol{f}_i^T \mathbf{A}_n \boldsymbol{u})^2 \tag{5.6}$$

$$= (\tilde{y}_i^n - \boldsymbol{u}^T \boldsymbol{f}_i^n)^2 \tag{5.7}$$

$$= \tilde{L}(\boldsymbol{f}_i^n, \tilde{y}_i^n, \boldsymbol{u}) \tag{5.8}$$

where $\mathbf{A}_n$ is the tying matrix for class n and $\tilde{y}_i^n \in \{-1, +1\}$ is equal to $s(y_i, n)$. With this loss function, the objective function becomes as follows:

$$\phi_{WS}(\boldsymbol{u}) = \frac{1}{I} \sum_{i=1}^{I} \sum_{n=1}^{N} \tilde{L}(\boldsymbol{f}_i^n, \tilde{y}_i^n, \boldsymbol{u}) + \lambda R(\boldsymbol{u}). \tag{5.9}$$

Minimizing this objective function is equivalent to learning a least-squares support vector machine (LS-SVM) binary classifier using the binary dataset $\{\{(\boldsymbol{f}_i^n, \tilde{y}_i^n\}_{n=1}^N\}_{i=1}^I$. This transformation for a dataset with 3 classes and 2 base classifiers is illustrated in Figure 5.1. We can obtain the nonlinear version of WS combination by applying the kernel trick [45] to the LS-SVM classifier. Kernel trick is explained in Section 5.4.

It should be noted that this transformation, and also the transformation for CWS combination which is explained in the following section, is valid for the least-squares loss function and it leads to an approximation for the hinge loss function. But we also applied this transformation to the hinge loss function in the experiments.

5.3 CWS combination using binary classifiers

For the least-squares loss function, we can split the objective function of CWS for different classes:

$$\phi_{CWS}(\boldsymbol{V}) = \sum_{n=1}^{N} \phi_n(\boldsymbol{v}_n), \tag{5.10}$$

where,

$$\phi_n(\boldsymbol{v}_n) = \frac{1}{I} \sum_{i=1}^{I} (\tilde{y}_i^n - \boldsymbol{v}_n^T \boldsymbol{f}_i^n)^2 + \lambda \tilde{R}(\boldsymbol{v}_n). \tag{5.11}$$

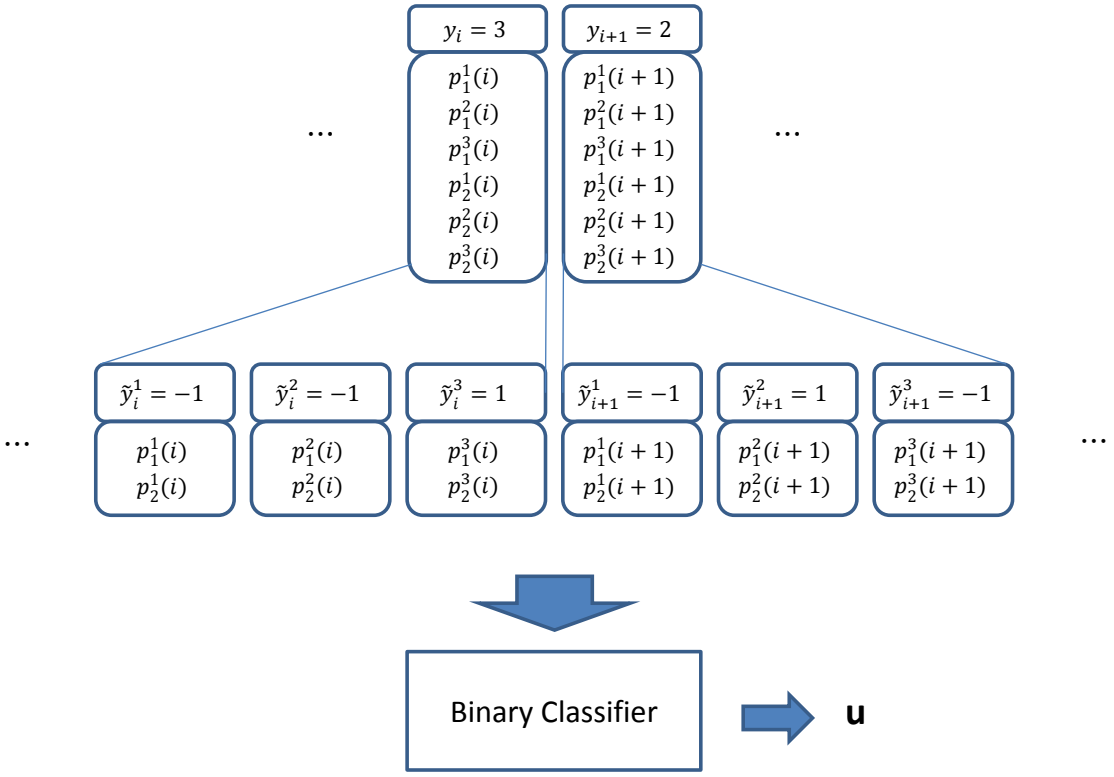


FIGURE 5.1: Transformation of a dataset with $N = 3$ and $M = 2$ for WS combination.

For this separation, regularization function should satisfy the following condition:

$$R(\boldsymbol{V}) = \sum_{n=1}^{N} \tilde{R}(\boldsymbol{v}_n) \tag{5.12}$$

l_2 norm and l_1 norm regularizations satisfy this condition. Since the parameters of each ϕ_n is independent from each other, instead of minimizing ϕ_{CWS} (5.10), we minimize ϕ_n (5.11) for each n separately. Minimizing ϕ_n is equivalent to learning a binary LS-SVM classifier from the dataset $\{\boldsymbol{f}_i^n, \tilde{y}_i^n\}_{i=1}^I$. This transformation for a dataset with 3 classes and 2 base classifiers is illustrated in Figure 5.2. This data transformation looks like one-vs-all multiclass classification method, but it is not the same since input data are different for each binary problem. We can obtain the nonlinear version of CWS combination by applying the kernel trick to the LS-SVM classifier. The kernel trick is explained in the following section.

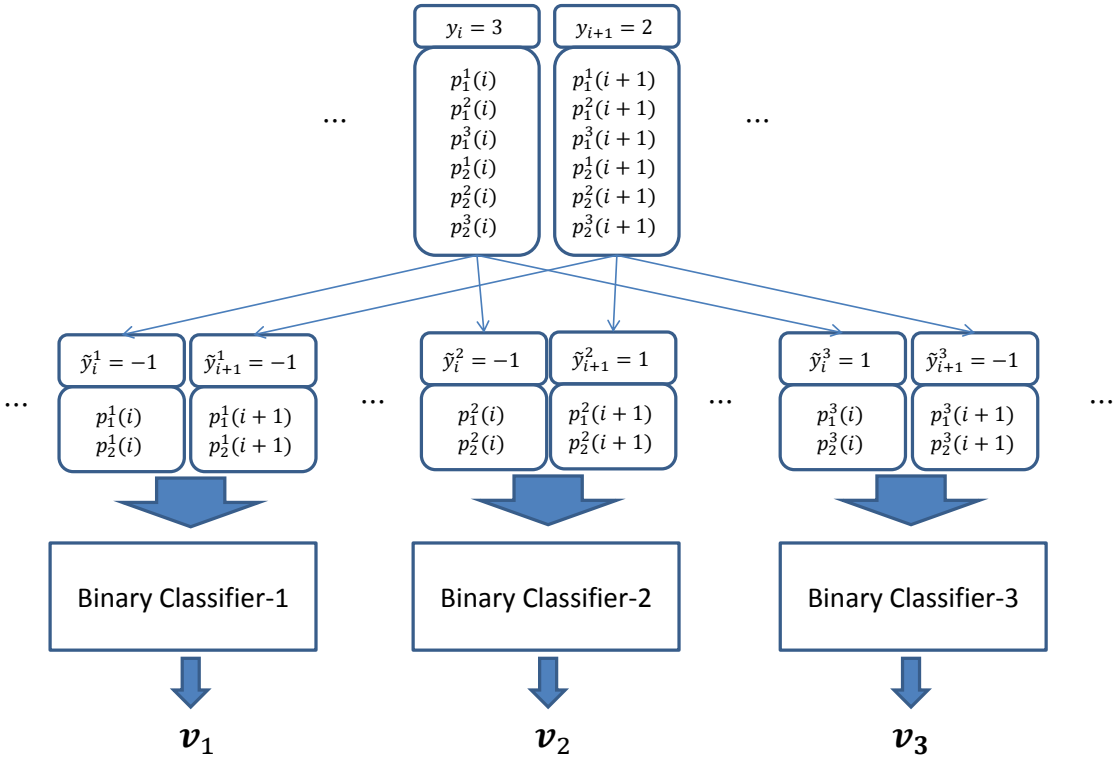


FIGURE 5.2: Transformation of a dataset with $N = 3$ and $M = 2$ for CWS combination.

5.4 The Kernel Trick

The kernel trick is a way of obtaining a non-linear classification method. The basic idea is mapping the observations into a much higher dimensional space and applying a linear classification in this space; instead of obtaining a nonlinear classifier directly. Since linear classification only needs the inner product pairs of data instances, instead of mapping the data instances and taking inner products, a kernel function is produced; so that nonlinear classification is faster.

First, we apply the kernel trick to the CWS combination. We have the following objective function to minimize for each class with l_2 regularization:

$$\min_{\boldsymbol{v}_n}\{\phi_n(\boldsymbol{v}_n)\} = \min_{\boldsymbol{v}_n}\{\frac{1}{I}\sum_{i=1}^{I}(\tilde{y}_i^n - \boldsymbol{v}_n^T\boldsymbol{f}_i^n)^2 + \lambda\boldsymbol{v}_n^T\boldsymbol{v}_n\}. \tag{5.13}$$

According to the Representer Theorem [46] the minimizer of (5.13) lies in the data-space. So we can write the $\boldsymbol{v}_n$ as a linear combination of data instances:

$$\boldsymbol{v}_n = \sum_{i=1}^{N} \alpha_i^n \tilde{y}_i^n \boldsymbol{f}_i^n, \tag{5.14}$$

for some $\alpha_i^n > 0$ for each i. If we replace above equation into (5.13), we obtain the following objective function:

$$\min_{\boldsymbol{\alpha}_n}\{\phi_n(\boldsymbol{\alpha}_n)\} = \min_{\boldsymbol{\alpha}_n}\{\frac{1}{I}\sum_{i=1}^{I}(\tilde{y}_i^n - \sum_{j=1}^{I}\alpha_j^n\tilde{y}_j^n < \boldsymbol{f}_j^n, \boldsymbol{f}_i^n >)^2 + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i^n\alpha_j^n\tilde{y}_i^n\tilde{y}_j^n < \boldsymbol{f}_j^n, \boldsymbol{f}_i^n >\} \tag{5.15}$$

$$\text{subject to}$$
$$\boldsymbol{\alpha} \geq 0$$

where, $\boldsymbol{\alpha}_n = [\alpha_1^n, \ldots, \alpha_I^n]^T$. We changed the optimization variable from $\boldsymbol{v}_n$ to $\boldsymbol{\alpha}_n$. Notice that posterior scores in above equation are present only in pairwise dot products: $< \boldsymbol{f}_i^n, \boldsymbol{f}_j^n >$. Now we map the posterior scores into a higher dimensional space with $\varphi : F_n \rightarrow U_n$, where F_n is the observation space for class n, i.e. $F_n = \text{span}(\boldsymbol{f}_1^n, \ldots, \boldsymbol{f}_I^n)$, U_n is an inner product space such that $\dim(F_n) < \dim(U_n)$. Instead of mapping the data and taking inner product, we construct a kernel function $k : F_n \times F_n \rightarrow \mathbb{R}$ as follows:

$$k(\boldsymbol{f}_i^n, \boldsymbol{f}_j^n) =< \varphi(\boldsymbol{f}_i^n), \varphi(\boldsymbol{f}_j^n) > \tag{5.16}$$

Given a mapping function φ, we can find the corresponding kernel function easily, but given a kernel function it may not be easy to find the corresponding mapping function. However, we do not require an explicit representation for φ: it is sufficient to know that there is a corresponding φ, i.e. it is sufficient for U_n to be an inner product space. For U_n to be an inner product space, kernel function should satisfy the Mercer's condition: There exists a mapping φ such that equation (5.16) holds, if and only if, for any g function such that $\int g(\boldsymbol{x})^2 d\boldsymbol{x}$ is finite, the following holds:

$$\int\int k(\boldsymbol{x}, \boldsymbol{y}) g(\boldsymbol{x}) g(\boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \geq 0 \tag{5.17}$$

We used two well known kernel functions that satisfy Mercer's Condition. First one is the polynomial kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + 1)^d, \tag{5.18}$$

where d is a kernel parameter called "degree". Latter kernel function is the radial basis function (RBF):

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma||\boldsymbol{x}-\boldsymbol{y}||^2} \tag{5.19}$$

Both the polynomial kernel and the RBF kernel corresponds to a mapping to an infinite dimensional space. When we replace the inner products with kernel functions in (5.15), we obtain the following objective function for the CWS combination:

$$\phi_n(\boldsymbol{\alpha}_n) = \frac{1}{I}\sum_{i=1}^{I}(\tilde{y}_i^n - \sum_{j=1}^{I}\alpha_j^n\tilde{y}_j^n k(\boldsymbol{f}_j^n, \boldsymbol{f}_i^n))^2 + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i^n\alpha_j^n\tilde{y}_i^n\tilde{y}_j^n k(\boldsymbol{f}_j^n, \boldsymbol{f}_i^n) \tag{5.20}$$

We minimize above function with respect to $\boldsymbol{\alpha}_n$ with the constraint that $\boldsymbol{\alpha}_n \geq 0$. Formulation given in (5.20) is in primal. For test phase, we keep the $\boldsymbol{\alpha}_n$ which minimize the objective function:

$$\hat{\boldsymbol{\alpha}}_n = \underset{\boldsymbol{\alpha}_n}{\operatorname{argmin}}\, \phi_n(\boldsymbol{\alpha}_n). \tag{5.21}$$

In the test phase, given a test instance $\boldsymbol{f}$, we find the score of class n as follows:

$$r^n = \sum_{i=1}^{I}\hat{\alpha}_i^n\tilde{y}_i^n k(\boldsymbol{f}_i^n, \boldsymbol{f}^n), \tag{5.22}$$

where $\boldsymbol{f}^n$ contains the elements in $\boldsymbol{f}$ associated with class n.

When we apply the kernel trick into (5.9) for the WS combination, we obtain the following objective function:

$$\phi_{WS}(\boldsymbol{\alpha}) = \frac{1}{I}\sum_{i=1}^{I}\sum_{n=1}^{N}(\tilde{y}_i^n - \sum_{j=1}^{I}\sum_{n'=1}^{N}\alpha_j^{n'}\tilde{y}_j^{n'}k(\boldsymbol{f}_i^n, \boldsymbol{f}_j^{n'}))^2 + \lambda\sum_{i=1}^{I}\sum_{n=1}^{N}\sum_{j=1}^{I}\sum_{n'=1}^{N}\alpha_i^n\alpha_j^{n'}\tilde{y}_i^n\tilde{y}_j^{n'}k(\boldsymbol{f}_j^{n'}, \boldsymbol{f}_i^n), \tag{5.23}$$

where $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1^T, \ldots, \boldsymbol{\alpha}_N^T]^T$ and $\boldsymbol{\alpha} \geq 0$. In the test phase, for a given test instance $\boldsymbol{f}$, we find the score of class n as follows:

$$r^n = \sum_{i=1}^{I}\sum_{n'=1}^{N}\hat{\alpha}_i^{n'}\tilde{y}_i^{n'}k(\boldsymbol{f}_i^{n'}, \boldsymbol{f}^n) \tag{5.24}$$

The kernel method, that we apply to the WS and CWS combinations, is equivalent to mapping the data nonlinearly into a higher dimensional space and applying linear classification in there. The generalization performance of the combiner may be sensitive to the selection of the parameters d in polynomial kernel and γ in the RBF kernel. They

TABLE 5.1: Properties of the data sets used in the experiments

DB	Train Instances	Test Instances	# of classes	# of features
optdigit	3823	1797	10	64
satellite	4435	2000	6	36
segment	1000	1310	7	19
waveform	2000	3000	3	21

are usually chosen according to cross-validation performances, as in the case of selection of the regularization parameter λ. In the next section, we explain the experiment setups, present results, and discuss them.

5.5 Experiments

We conduct experiments with 2 different ensembles. First, we constructed ensembles using 4 datasets and obtained results for both the LS loss and the hinge loss function. Then, we conducted experiments with the diverse ensembles obtained in Chapter 4 for 8 datasets using the hinge loss function.

5.5.1 Experimental setup - 1

For the former ensembles, we used 4 datasets from the UCI Machine Learning Repository [38]. Some properties of these datasets are summarized in Table 5.1.

We constructed an ensemble with 13 different base classifiers for each dataset. These 13 different classifiers are: normal densities based linear classifier, normal densities based quadratic classifier, nearest mean classifier, k-nearest neighbor classifier, polynomial classifier, general kernel/dissimilarity based classification, normal densities based classifier with independent features, parzen classifier, binary decision tree classifier, linear perceptron, SVM with linear kernel, polynomial kernel, and radial basis function (RBF) kernel.

2-fold internal CV is applied for obtaining the training data of the combiner. Even though data transformation methods explained in Sections 5.2 and 5.3 are not exact for the hinge loss function, we also perform experiments with the hinge loss function. We used the polynomial kernel and the RBF kernel functions. We searched for the

regularization parameter (λ) and kernel parameters (d or γ) with 2-fold cross-validation. We choose the parameter pairs that gives the best average combination accuracy, rather than the ones that give best average binary classification accuracy. Combiners are trained using the LS-SVM Toolbox in Matlab [28] for least-squares loss function and the Libsvm Toolbox [41] for the hinge loss function.

TABLE 5.2: Error percentages for the WS combination

DB	Hinge Loss			LS Loss		
	Lin	Poly	RBF	Lin	Poly	RBF
optdigit	**1.67**	1.95	2.28	1.78	1.78	3.36
satellite	10.05	10.55	10.60	**9.95**	**9.95**	12.25
segment	4.2	5.73	**3.66**	4.35	4.35	4.73
waveform	13.10	13.23	**12.87**	13.37	13.37	13.20

TABLE 5.3: Error percentages for the CWS combination

DB	Hinge Loss			LS Loss		
	Lin	Poly	RBF	Lin	Poly	RBF
optdigit	2.95	2.95	4.06	**1.89**	**1.89**	1.95
satellite	12.85	10.70	**10.40**	12.20	12.20	12.60
segment	3.59	3.59	3.66	3.59	3.59	**3.51**
waveform	13.10	**12.83**	13.10	13.37	13.37	13.07

TABLE 5.4: Error percentages for the LSG combination with the hinge loss

DB	EW	Lin	Poly	RBF
optdigit	3.78	**2.29**	2.90	**2.29**
satellite	12.40	13.15	13.55	**13.05**
segment	7.33	3.90	5.65	**3.21**
waveform	13.37	**13.00**	13.14	**13.00**

For the WS and CWS combinations, we obtained results for both LS and the hinge loss function. In Table 5.2, we give the results for the WS combination. Linear combiner error percentages are given at the column titled *Lin* and nonlinear combination results with polynomial and RBF kernel functions are given in the columns titled *Poly* and *RBF* respectively. We give the error percentages for the CWS combination in Table 5.2. We also give the error percentages for the LSG combination with the hinge loss function and the mean rule (in the column titled *EW*) in Table 5.4. For all combination types, i.e. WS, CWS and LSG, we are able to obtain the lowest errors using the nonlinear combiners. In general, RBF kernel function gives better accuracies than the polynomial kernel. For LSG combination, RBF kernel outperforms both the linear combiner and the polynomial kernel.

In general we see that, nonlinear combination works slightly better than the linear combination, but there is not much difference. This could result from the non-complexity of the classifier combination problem, but it is a fact that there is not much study of the nonlinear classifier combination problem. Additional improvements in this problem may result in increased accuracies. One possible improvement might be finding or deriving special kernel functions that are suitable for the classifier combination problem.

Developed data transformation methods are used for obtaining nonlinear versions of WS and CWS combinations. However, even for linear combinations, the data transformation may have a computational advantage. Since we transform a multiclass dataset into binary datasets, training time will be much less; because binary classifiers work faster than multiclass classifiers in general. As the number of data instances increase, time advantage of our method would increase, especially if the optimization algorithm of the combiner is in the primal domain.

5.5.2 Experimental setup - 2

For the second experimental setup, we used the ensembles obtained in Chapter 4. Since results indicate that we can use the data transformation for the hinge loss even though it is not valid for the hinge loss, we obtained results for the hinge loss function only for the second experimental setup. We used the Libsvm Toolbox [41] for the combiner. For the LSG combination, we used one-versus-one multiclass combination. For statistical significance, we used Wilcoxon signed rank test with one tailed significance level of $\alpha = 0.05$ [43]. Results are shown in Table 5.5. The columns *LIN* indicate the linear kernel and *SIG* indicate the significance. The letter "a,b,c" under the column *SIG* indicate that performance difference between linear and RBF kernel are statistically significant for WS,CWS and LSG combinations respectively.

TABLE 5.5: Error percentages with the diverse ensemble setup (*mean* ± *standard deviation*).

DB	WS		CWS		LSG		EW	SIG
	LIN	RBF	LIN	RBF	LIN	RBF		
Segment	3.44 ± 0.67	**3.08 ± 0.72**	3.18 ± 0.79	**3.01 ± 0.76**	3.21 ± 0.53	**2.99 ± 0.43**	7.37 ± 1.03	ab
Waveform	13.56 ± 0.66	**13.10 ± 0.65**	**13.08 ± 0.70**	13.11 ± 0.65	13.20 ± 0.67	**13.18 ± 0.64**	14.17 ± 0.60	a
Robot	2.58 ± 0.38	**2.48 ± 0.38**	2.43 ± 0.31	**2.41 ± 0.28**	**2.57 ± 0.37**	2.63 ± 0.41	18.58 ± 0.61	
Statlog	18.18 ± 1.69	**16.45 ± 1.50**	16.43 ± 1.27	16.43 ± 1.55	**16.64 ± 1.47**	17.40 ± 1.98	23.03 ± 2.33	a
Vowel	6.02 ± 1.97	**5.88 ± 2.09**	**6.06 ± 1.97**	6.22 ± 1.88	5.60 ± 1.57	**5.43 ± 1.47**	14.53 ± 3.30	
Wine	**1.57 ± 1.42**	1.69 ± 1.52	**1.57 ± 1.52**	2.02 ± 1.57	**1.35 ± 1.48**	2.13 ± 1.45	2.81 ± 1.52	bc
Yeast	41.28 ± 0.78	**39.99 ± 0.91**	**40.58 ± 1.48**	40.75 ± 0.95	**40.58 ± 0.88**	40.70 ± 0.91	40.26 ± 1.10	a
Steel	27.46 ± 1.36	**27.07 ± 1.19**	26.76 ± 0.73	**26.69 ± 0.65**	**26.30 ± 0.91**	26.38 ± 0.98	31.57 ± 2.07	

In general, we see that, nonlinear combination with RBF kernel works much better than the linear combination for the WS combination type. But there is not much difference for CWS and LSG combinations.

Next, we compare the data transformation method for the hinge loss with linear kernel in Table 5.6 with Crammer-Singer (CS) type WS and CWS combination types as found in Chapter 4. The letter "a,b" under the significance column indicate that the differences in the errors between data transformation method (DT) and the direct combination using the Crammer-Singer method (CS) are statistically significant for WS and CWS combinations respectively. On at least half of the datasets, using the data transformation and then applying the hinge loss function works better than using the objective function of Crammer-Singer with both WS and CWS combinations.

TABLE 5.6: Error percentages of Crammer-Singer method and Data transformation with the diverse ensemble setup for WS and CWS. (*mean* ± *standard deviation*).

DB	WS		CWS		SIG
	DT	CS	DT	CS	
Segment	**3.44 ± 0.67**	5.02 ± 0.88	**3.18 ± 0.79**	3.53 ± 0.99	ab
Waveform	13.56 ± 0.66	**13.20 ± 0.69**	13.08 ± 0.70	13.08 ± 0.76	a
Robot	**2.58 ± 0.38**	3.95 ± 0.42	**2.43 ± 0.31**	2.53 ± 0.28	ab
Statlog	18.18 ± 1.69	**16.34 ± 1.15**	16.43 ± 1.27	**16.12 ± 1.94**	a
Vowel	**6.02 ± 1.97**	13.84 ± 2.73	**6.06 ± 1.97**	6.97 ± 1.73	ab
Wine	1.57 ± 1.42	1.57 ± 1.09	1.57 ± 1.52	**1.01 ± 1.45**	
Yeast	41.28 ± 0.78	**40.36 ± 1.21**	**40.58 ± 1.48**	40.63 ± 1.21	a
Steel	**27.46 ± 1.36**	29.85 ± 1.86	**26.76 ± 0.73**	27.37 ± 1.18	a

Next, we compare the Crammer-Singer multiclass SVM with the one-versus-one SVM for the LSG combination in Table 5.7. The results of datasets *Waveform* and *Steel* are statistically significant. In general we see that, neither of them are superior over the other one. So we conjecture that, lower error percentages of the data transformation as compared to the Crammer-Singer method for WS and CWS combination types result from the data-transformation rather than the bad performance of the Crammer-Singer method.

TABLE 5.7: Error percentages of one-versus-one (OVO) versus Crammer-Singer (CS) methods for LSG. (*mean* ± *standard deviation*).

DB	OVO	CS
Segment	**3.21 ± 0.53**	3.60 ± 1.05
Waveform	13.20 ± 0.67	**13.05 ± 0.65**
Robot	**2.57 ± 0.37**	2.61 ± 0.28
Statlog	16.64 ± 1.47	**16.36 ± 1.67**
Vowel	**5.60 ± 1.57**	6.32 ± 1.99
Wine	**1.35 ± 1.48**	1.69 ± 1.32
Yeast	40.58 ± 0.88	**40.32 ± 1.19**
Steel	**26.30 ± 0.91**	27.41 ± 1.22

Chapter 6

An MM Algorithm for CWS Combination

6.1 Introduction

The results of the experiments in Chapter 4 suggest that, the CWS combination is the most preferable method among linear combination types. Previous studies that perform stacked generalization use the least-squares loss function, but we promote the hinge loss function in Chapter 4. Optimization algorithms for least-squares estimation have been thoroughly investigated in the literature and there are numerous algorithms/solutions. But there is not much study for the multiclass SVM. Moreover, there is no specific solution or software for the CWS combination, but only generic optimization toolkits. These generic optimization toolkits, usually, solve the dual problem, which leads to slower training as the number of data-instances increase, compared to the primal solution. In this chapter, we derive an optimization algorithm for the CWS combination in the primal domain. We apply majorize-minimize (MM) algorithms with coordinate descent method to our problem including l_2, l_1 and group sparse regularizations.

6.2 MM Algorithms

MM algorithms are first proposed by Ortega and Rheinboldt in 1969 [47]. MM stands for "Majorize-minimize" for a minimization problem and "minorize-maximize" for a

maximization problem. MM algorithms are iterative methods and they are similar to the expectation maximization (EM) algorithms. In fact, every EM algorithm is a special case of the more general class of MM optimization algorithms [48]. Given an objective function to minimize, the basic idea of MM is, at each iteration, finding a majorizing function, called surrogate function, at the current point and minimizing this surrogate function. We first discuss MM algorithms in a one-dimensional problem and move onto higher dimensional cases. Let $f(\theta)$ be a real valued function of the parameter θ and we want to minimize this function iteratively. We find a majorizing function of f at the current point $\theta^{(t)}$ and minimize this function rather than the original objective function. t represents the iteration, so at each iteration, we find a majorizing function at the point $\theta^{(t)}$. The real valued function $g(\theta \mid \theta^{(t)})$ is said to majorize $f(\theta)$ at the point $\theta^{(t)}$ if the following conditions hold:

$$g(\theta \mid \theta^{(t)}) \geq f(\theta) \quad \text{for all} \quad \theta \tag{6.1}$$

$$g(\theta^{(t)} \mid \theta^{(t)}) = f(\theta^{(t)}). \tag{6.2}$$

An illustration is given in Figure 6.1. Let $\theta^{(t+1)}$ be the minimizer of the surrogate

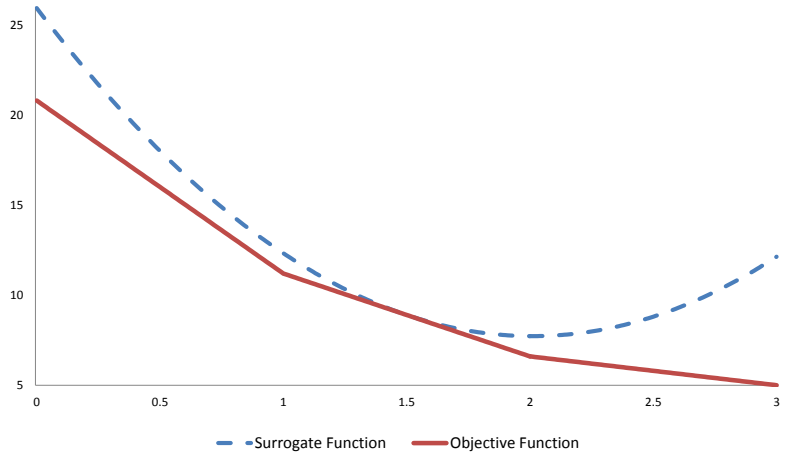


FIGURE 6.1: A quadratic majorizing function of an objective function at $\theta^{(t)} = 1.5$.

function $g(\theta \mid \theta^{(t)})$. Using the conditions in (6.1) and (6.2), we can see that MM algorithms ensure monotonic decrease of the objective function:

$$f(\theta^{(t+1)}) \leq g(\theta^{(t+1)} \mid \theta^{(t)}) \leq g(\theta^{(t)} \mid \theta^{(t)}) = f(\theta^{(t)}). \tag{6.3}$$

This descent property lends an MM algorithm remarkable numerical stability. If the majorizing function is quadratic, it can be minimized in one step. So we construct quadratic majorizing functions. A quadratic majorizing function of $f(\theta)$ at point $\theta^{(t)}$ can be formulated as follows:

$$g(\theta \mid \theta^{(t)}) = f(\theta^{(t)}) + \dot{f}(\theta^{(t)})(\theta - \theta^{(t)}) + \frac{1}{2}c(\theta^{(t)})(\theta - \theta^{(t)})^2, \tag{6.4}$$

where $\dot{f}(\theta^{(t)}) = \frac{\partial f(\theta)}{\partial \theta}\big|_{\theta=\theta^{(t)}}$. The formulation above follows from the facts that $g(\theta^{(t)} \mid \theta^{(t)}) = f(\theta^{(t)})$ and first derivatives of f and g are equal at $\theta^{(t)}$ since the surrogate function is tangent to the objective function at that point. $c(\theta^{(t)})$ in (6.4) is the curvature value and it can be chosen from the range $[c_{opt}, \infty)$ where c_{opt} is the minimum curvature value such that condition (6.1) holds. In general, we want to choose the curvature value as low as possible since lower curvature values lead to faster convergence. So we call c_{opt} the optimum curvature value. For convex objective functions, this curvature value results in a surrogate function such that it is tangent to the objective function at another point besides $\theta^{(t)}$. For special objective functions, there are simple ways to find the optimal curvature. We use the following theorem which is proved in [49] to find the optimal curvature for our problem:

Theorem 1. Let $f(\theta)$ be a one dimensional function such that its first derivative is continuous. If $\dot{f}(\theta)$ has the property of odd symmetry with respect to a point a, in other words if there is a b value such that

$$\dot{f}(a + \theta) - b = b - \dot{f}(a - \theta) \tag{6.5}$$

and if $\dot{f}(\theta)$ is convex for $\theta < a$, then the minimum curvature such that condition (6.1) holds can be found as follows:

$$c(\theta^{(t)}) = \left| \frac{\dot{f}(\theta^{(t)}) - \dot{f}(2a - \theta^{(t)})}{2(\theta^{(t)} - a)} \right| \tag{6.6}$$

Theorem 1 is used to find the minimum curvature possible such that the surrogate function majorizes the objective function at $\theta^{(t)}$. We can also find the minimum curvature for all θ values such that condition (6.1) holds. In [50], it is shown that minimum curvature such that (6.1) holds for all θ is the maximum second derivative of the objective function: $c_{max} = \max_\theta \ddot{f}(\theta)$. We call this curvature *maximum curvature*. It will lead

to slower convergence on each iteration, but since the curvature value is fixed, saved time by not calculating the curvature on each iteration may lead to faster convergence overall.

Another property of MM algorithms that we use in our problem is that majorization relation between functions is closed under the formation of sums. Let $g_i(\theta \mid \theta^{(t)})$ be the majorizer of the function $f_i(\theta)$ at $\theta^{(t)}$ for $i = 1, \ldots, B$ and let $\tilde{f}(\theta) = \sum_{i=1}^{B} f_i(\theta)$. Then $\tilde{g}(\theta \mid \theta^{(t)}) = \sum_{i=1}^{B} g_i(\theta \mid \theta^{(t)})$ majorizes $\tilde{f}(\theta)$ at the point $\theta^{(t)}$. This rule permit us to work piecemeal in simplifying complicated objective functions. In the following sections, we derive an MM algorithm for the CWS combination.

6.3 Problem Formulation

Recall that we have the following objective function for CWS combination with the hinge loss function as described in (4.7):

$$\phi(\boldsymbol{V}) = \frac{1}{I} \sum_{i=1}^{I} (1 - \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i} + \max_{n \neq y_i} (\boldsymbol{v}_n^T \boldsymbol{f}_i^n))_+ + \lambda R(\boldsymbol{V}), \tag{6.7}$$

where $\boldsymbol{v}_n$ is the n^{th} column of $\boldsymbol{V} \in \mathbb{R}^{M \times N}$ and it contains the weights for class n. The m^{th} row of $\boldsymbol{V}$ contain the weights of base classifier m for different classes. We write this objective function as follows:

$$\phi(\boldsymbol{V}) = L(\boldsymbol{V}) + \lambda R(\boldsymbol{V}), \tag{6.8}$$

Here $L(\boldsymbol{V})$ gives the summation of loss values over data-instances:

$$L(\boldsymbol{V}) = \frac{1}{I} \sum_{i=1}^{I} h(z_i(\boldsymbol{V})), \tag{6.9}$$

where $h(z) = \max(0, 1 - z)$ and

$$z_i(\boldsymbol{V}) = \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i} - \max_{n \neq y_i} (\boldsymbol{v}_n^T \boldsymbol{f}_i^n). \tag{6.10}$$

Since maximum among wrong classes leads to a non-differentiable function, we approximate the maximum operator with logarithm of sum of exponentials, which is also called

softmax:

$$\max_{n \neq y_i} (\boldsymbol{v}_n^T \boldsymbol{f}_i^n) \approx \log \sum_{n \neq y_i} e^{\boldsymbol{v}_n^T \boldsymbol{f}_i^n} \tag{6.11}$$

When we apply this approximation, we obtain the following z_i function:

$$z_i(\boldsymbol{V}) = \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i} - \log \sum_{n \neq y_i} e^{\boldsymbol{v}_n^T \boldsymbol{f}_i^n}. \tag{6.12}$$

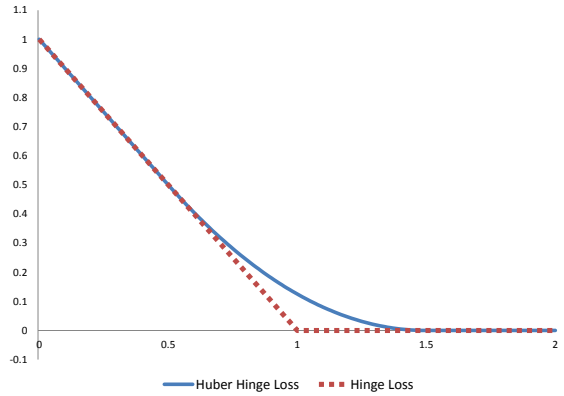
$z_i(\boldsymbol{V})$ is an approximation of the difference between the score of the correct class and the score of the most offending wrong class for data point i with the current combiner $\boldsymbol{V}$. From now, we sometimes drop the subscript i and the argument $\boldsymbol{V}$ of $z_i(\boldsymbol{V})$ for simplicity. $h(z)$ is the hinge loss function and it is not differentiable at $z = 1$. For this reason, as z gets close to one, c_{opt} goes to infinity. When c_{opt} is not bounded, there is no guarantee of convergence [51]. To overcome this problem, we use the Huber-hinge loss function defined as follows:

$$h(z, \tau) = \begin{cases} 1 - z & \text{if } z \leq 1 - \tau \\ \frac{1}{4\tau}(z - (1 + \tau))^2 & \text{if } 1 - \tau < z \leq 1 + \tau \\ 0 & \text{if } z > 1 + \tau \end{cases} \tag{6.13}$$

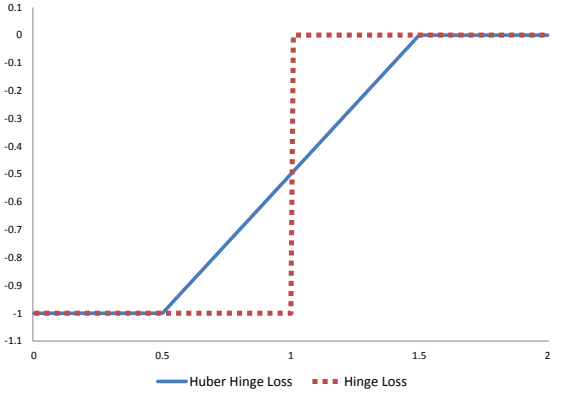
Huber hinge loss function smoothes the hinge loss function near $z = 1$. It should be noted that the derivatives of the hinge loss function and Huber-hinge loss function at the points $z = 1 - \tau$ and $z = 1 + \tau$ are equal. Plot of the hinge loss function and the Huber-hinge loss function and their derivatives with respect to z are given in Figures 6.2(a) and 6.2(b) for $\tau = 0.5$. The derivative of the Huber-hinge loss function is as follows:

$$\dot{h}(z, \tau) = \begin{cases} -1 & \text{if } z \leq 1 - \tau \\ \frac{1}{2\tau}(z - (1 + \tau)) & \text{if } 1 - \tau < z \leq 1 + \tau \\ 0 & \text{if } z > 1 + \tau \end{cases} \tag{6.14}$$

In the next section, we find quadratic surrogate functions for the Huber-hinge loss function with respect to z.



(a) Loss functions



(b) Derivatives

FIGURE 6.2: Hinge Loss and Huber Hinge Loss and their derivatives for $\tau = 0.5$.

6.4 Quadratic Majorizing Functions

We want to minimize the objective function defined in (6.8). We solve this problem iteratively. For each iteration, we construct a majorizing function at the current combiner and minimize it using coordinate descent. In this section, we find majorizing functions of the objective function, which consists of a loss value and regularization value. We first derive a majorizer for the loss value in the next subsection, then handle the regularization part in Section 6.4.2.

6.4.1 Majorizer of the loss function

We want to find a quadratic majorizer of the loss value given in (6.9). Then we minimize this majorizer using a coordinate descent algorithm, which optimizes the majorizer with respect to $v_{n,m}$ for each n and m consecutively, where $v_{n,m}$ is the weight of classifier m for class n. We find the majorizer, $g_i(\boldsymbol{V}; \boldsymbol{V}^{(t)})$, of the Huber-hinge loss function for the

data-instance i by applying the one dimensional quadratic expansion defined in (6.4) to our multi-dimensional problem as follows:

$$g_i(\boldsymbol{V};\boldsymbol{V}^{(t)}) = h(z_i(\boldsymbol{V}^{(t)}))+\dot{h}(z_i(\boldsymbol{V}^{(t)}))\big(\frac{\partial z(\boldsymbol{V})}{\partial \tilde{\boldsymbol{v}}}\Big|_{\tilde{\boldsymbol{v}}=\tilde{\boldsymbol{v}}^{(t)}}\big)^T(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)})+\frac{1}{2}(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)})^T\boldsymbol{C}_i^{(t)}(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)}), \tag{6.15}$$

where $\tilde{\boldsymbol{v}} \in \mathbb{R}^{MN}$ contains the weights, $\boldsymbol{C}_i^{(t)} \in \mathbb{R}^{MN\times MN}$ is the curvature matrix at $\boldsymbol{V}^{(t)}$. For this multi-dimensional case, there is no method for finding the optimal curvature values to our knowledge. To simplify this problem, instead of finding a majorizing function with respect to $\boldsymbol{V}$, we find the majorizer with respect to the one-dimensional variable z_i for each i:

$$\tilde{g}_i(z_i; z_i^{(t)}) = h(z_i^{(t)}) + \dot{h}(z_i^{(t)})(z_i - z_i^{(t)}) + \frac{1}{2}c(z_i^{(t)})(z_i - z_i^{(t)})^2, \tag{6.16}$$

Above function also majorizes $h(z)$ at $\boldsymbol{V}^{(t)}$ as $g_i(\boldsymbol{V};\boldsymbol{V}^{(t)})$ in (6.15). But unlike g_i, $\tilde{g}_i$ function is not quadratic with respect to $\boldsymbol{V}$, but it is quadratic with respect to z only. We obtain the majorizer of $L(\boldsymbol{V})$ at $\boldsymbol{V}^{(t)}$ as follows:

$$\tilde{G}_L(\boldsymbol{V},\boldsymbol{V}^{(t)}) = \frac{1}{I}\sum_{i=1}^{I}\tilde{g}_i(z_i, z_i^t) \tag{6.17}$$

$$= L(\boldsymbol{V}^t) + \frac{1}{I}\sum_{i=1}^{I}\dot{h}_i^{(t)}(z_i(\boldsymbol{V}) - z_i^t) + \frac{1}{2I}\sum_{i=1}^{I}c_i^t(z_i(\boldsymbol{V}) - z_i^t)^2 \tag{6.18}$$

where $\dot{h}_i^{(t)} = \dot{h}(z_i(\boldsymbol{V}^{(t)}))$, $c_i^t = c(z_i(\boldsymbol{V}^{(t)}))$ and $z_i^{(t)} = z_i(\boldsymbol{V}^{(t)})$. $\tilde{G}_L$ majorizes the loss function $L(\boldsymbol{V})$ at $\boldsymbol{V}^{(t)}$, but it is not quadratic. So we find the second order Taylor approximation of $\tilde{G}_L$ at $\boldsymbol{V}^{(t)}$:

$$\begin{aligned} Q_L(\boldsymbol{V},\boldsymbol{V}^{(t)}) = \tilde{G}_L(\boldsymbol{V}^{(t)},\boldsymbol{V}^{(t)}) + (\frac{\partial\tilde{G}_L(\boldsymbol{V},\boldsymbol{V}^{(t)})}{\partial\tilde{\boldsymbol{v}}}\Big|_{\tilde{\boldsymbol{v}}=\tilde{\boldsymbol{v}}^{(t)}})^T(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)}) \\ + \frac{1}{2}(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)})^T(\frac{\partial^2\tilde{G}_L(\boldsymbol{V},\boldsymbol{V}^{(t)})}{\partial\tilde{\boldsymbol{v}}^2}\Big|_{\tilde{\boldsymbol{v}}=\tilde{\boldsymbol{v}}^{(t)}})(\tilde{\boldsymbol{v}}-\tilde{\boldsymbol{v}}^{(t)}). \end{aligned} \tag{6.19}$$

The function above is now quadratic, but there is no guarantee that it will majorize the loss function. But in the experiments, we observe the monotonic decrease of the loss function, which shows that above approximation does not perturb the convergence of the solution. Here, we make the assumption that $z(\boldsymbol{V})$ is linear in $v_{n,m}$ for any n, m and we find the majorizer of $L(\boldsymbol{V})$ with respect to z values. z_i is linear in $v_{n,m}$ if $y_i = n$,

approximately linear if n is the most offending wrong class; but it is not linear if $v_{n,m}$ neither belongs to true class nor to the most offending wrong class. Linearity assumption simplifies the problem a lot, especially the calculation of the optimal curvature values. By Theorem 1 the optimal curvature for the Huber-hinge loss is found as follows:

$$c_{opt}(z) = \begin{cases} \frac{1}{2|1-z|} & \text{if } |z-1| > \tau \\ \frac{1}{2\tau} & \text{if } |z-1| \leq \tau \end{cases} \tag{6.20}$$

Maximum curvature for the Huber-hinge loss, which is equal to the maximum second derivative of the objective function, is found as follows:

$$c_{max}(z) = \frac{1}{2\tau} \tag{6.21}$$

We plot the quadratic majorizing function with optimal curvature and maximum curvature of the Huber-hinge loss function with $\tau = 0.5$ at $z = 0$ in Figure 6.3. Next, we

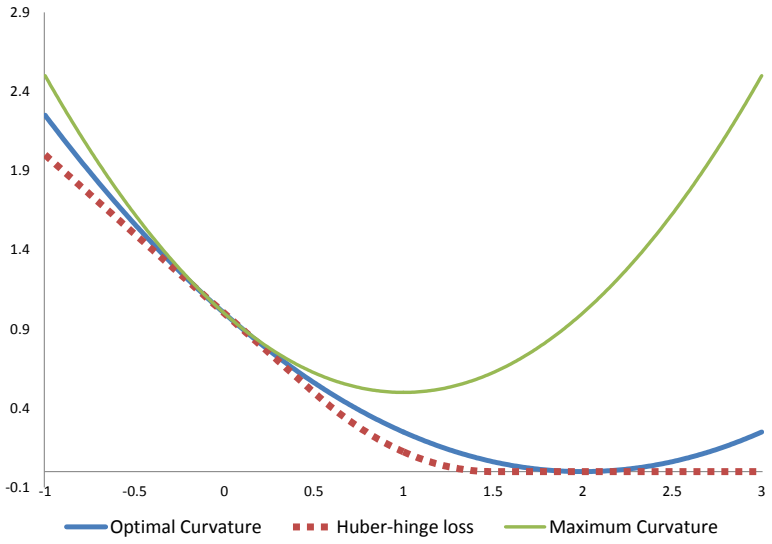


FIGURE 6.3: Quadratic majorizing function of the Huber-hinge loss with optimal curvature and maximum curvature at $z = 0$.

show methods for incorporating regularization parts into the algorithms.

6.4.2 Handling regularizations

We derive methods for l_2, l_1 and group sparse regularizations in the subsequent subsections.

6.4.2.1 l_2 regularization

If the regularization function is the usual l_2 norm squared, the surrogate function of it can be the regularization function itself since the l_2 norm square is a quadratic function. So we add the regularization function to the surrogate function of the loss part to find the overall surrogate function that majorizes $\phi(\boldsymbol{V})$ at $\boldsymbol{V}^{(t)}$ and minimize it.

$$Q(\boldsymbol{V}, \boldsymbol{V}^{(t)}) = Q_L(\boldsymbol{V}, \boldsymbol{V}^{(t)}) + \lambda ||\boldsymbol{V}||^2_{Fro}, \tag{6.22}$$

where $||\ .\ ||_{Fro}$ is the Frobenious norm.

6.4.2.2 l_1 regularization

For l_1 norm regularization, instead of finding a majorizing function, we use the shrinkage thresholding operator. This method is applicable to our problem since the majorizer of the loss part is approximately quadratic. Let $\hat{v}_{n,m}$ be the minimizer of the surrogate function given in (6.19) in one dimension:

$$\hat{v}_{n,m} = \arg\min_{v_{n,m}} Q_L(\boldsymbol{V}, \boldsymbol{V}^{(t)}) \tag{6.23}$$

Now it can be shown that the one-dimensional objective function of $v_{n,m}$ can be approximated as follows:

$$Q(\boldsymbol{V}, \boldsymbol{V}^{(t)}) \approx C + \frac{1}{2} D''_{n,m}(v_{n,m} - \hat{v}_{n,m})^2 + \lambda |v_{n,m}|, \tag{6.24}$$

where C is a constant independent of $v_{n,m}$ and $D''_{n,m} = \frac{\partial^2 Q(\boldsymbol{V}, \boldsymbol{V}^{(t)})}{\partial v^2_{n,m}}$. We can then find the minimizer of the overall one-dimensional function which is a sum of an approximate parabola and an absolute value function using the shrinkage thresholding operator:

$$v^*_{n,m} = S\left(\hat{v}_{n,m}, \frac{\lambda}{D''_{n,m}}\right) \tag{6.25}$$

$$S(v, \gamma) = \text{sign}(v)(|v| - \gamma)_+, \tag{6.26}$$

where $(x)_+ = \max(0, x)$. $S(v, \gamma)$ is the shrinkage thresholding operator which shrinks a parameter v towards zero. This result follows from the simple analysis of a minimum

value of a single dimensional function that is a sum of a parabola and an absolute value function.

6.4.2.3 $l_1 - l_2$ regularization

Recall that for group sparse regularization, we have the following regularization function:

$$R(\boldsymbol{V}) = ||\boldsymbol{V}||_{1,2} = \sum_{m=1}^{M} ||\boldsymbol{v}^m||_2 = \sum_{m=1}^{M} \sqrt{\sum_{n=1}^{N} v_{n,m}^2}, \tag{6.27}$$

where $\boldsymbol{v}^m$ is the m^{th} row of $\boldsymbol{V}$, so it contains the weights of classifier m. After learning, any row of $\boldsymbol{V}$ is either all zero or all non-zero. Since this regularization is not quadratic, we should find a quadratic majorizer of it. Since we are going to apply coordinate descent algorithm to the surrogate function, we write the regularization function in one dimension as follows:

$$R_{n,m}(v_{n,m}) = C_1 + \sqrt{C_2 + v_{n,m}^2} \tag{6.28}$$

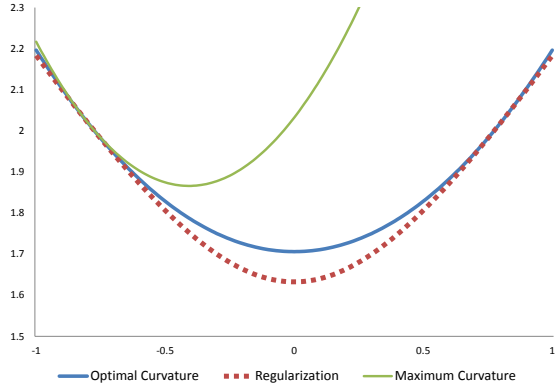
where C_1 and C_2 are independent of $v_{n,m}$:

$$C_1 = \sum_{m' \neq m} \|\boldsymbol{v}^{m'}\|_2, \tag{6.29}$$

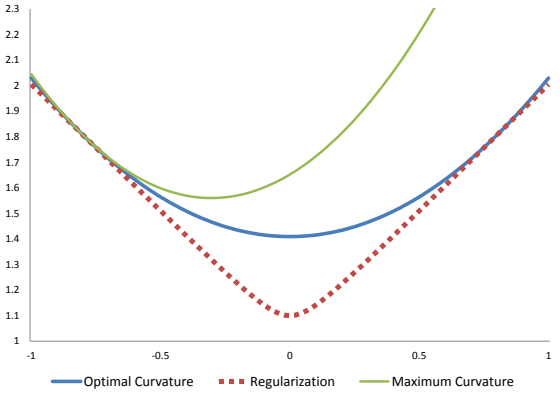
$$C_2 = \sum_{n' \neq n} v_{n',m}^2. \tag{6.30}$$

As C_2 gets smaller, the regularization gets close to the l_1 normalization and if $C_2 = 0$, it is exactly the l_1 norm regularization and in this case, we can use the shrinkage thresholding operator to find the minimizer of the objective function. But if a classifier have nonzero weights at some iteration, these weights cannot be exactly zero but will get close to zero. And as C_2 converges to zero, the optimal curvature value will increase, resulting in a slower convergence. So, to speed up the convergence, we threshold the C_2 value for all n, m and if it is smaller than some value, ϵ, we apply the shrinkage thresholding operator. If C_2 is not close to zero, we find the quadratic majorizer of the regularization function and minimize it. We can write this quadratic majorizer as follows:

$$Q_{n,m}(v_{n,m}, v_{n,m}^{(t)}) = R_{n,m}(v_{n,m}^{(t)}) + \dot{R}_{n,m}(v_{n,m}^{(t)})(v_{n,m} - v_{n,m}^{(t)}) + \frac{1}{2} c_{n,m}(v_{n,m}^{(t)})(v_{n,m} - v_{n,m}^{(t)})^2 \tag{6.31}$$



(a) High C_2



(b) Low C_2

FIGURE 6.4: Quadratic majorizing functions of group sparse regularizations for low and high C_2 values at $v_{n,m} = -0.8$.

The optimal curvature is found by Theorem 1 as follows:

$$c_{n,m}^{opt}(v_{n,m}) = \frac{1}{\sqrt{C_2 + v_{n,m}^2}} \tag{6.32}$$

The maximum curvature is found as follows:

$$c_{n,m}^{max}(v_{n,m}) = \frac{1}{\sqrt{C_2}} \tag{6.33}$$

The majorizer of group sparse regularizations with optimal curvature and maximal curvature for low and high C_2 are given in Figures 6.4(a) and 6.4(b) respectively. Notice that, for low C_2, regularization is close to the l_1 norm regularization. We add the surrogate function of the regularization function to the surrogate function of the loss function to find the majorizer of the overall objective function.

6.5 Coordinate Descent Algorithm

Coordinate descent algorithms update a single parameter while keeping other parameters constant. We minimize the majorizing function using coordinate descent algorithms, i.e., we minimize with respect to $v_{n,m}$ one by one for each n and m. Since the majorizing function is quadratic, we minimize it in one step using Newton's method for a given n and m:

$$\hat{v}_{n,m} = v_{n,m}^{(t)} - \frac{E'_{n,m}}{E''_{n,m}}, \tag{6.34}$$

where $E'_{n,m}$ and $E''_{n,m}$ are the first and second derivatives of the quadratic majorizing function of the objective function respectively. Recall that for l_2 regularization, we add the regularization itself to the majorizing function and for $l_1 - l_2$ regularization we add the majorizing function of the regularization function. For l_2 regularization, we find these derivatives as follows:

$$E'_{n,m} = D'_{n,m} + 2\lambda v_{n,m} \tag{6.35}$$

$$E''_{n,m} = D''_{n,m} + 2\lambda, \tag{6.36}$$

where $D'_{n,m}$ and $D''_{n,m}$ are the first and second derivatives of the surrogate function of the loss part. $D'_{n,m}$ is found to be as follows:

$$D'_{n,m} = \frac{\partial Q_L(\boldsymbol{V}, \boldsymbol{V}^t)}{\partial v_{n,m}} = \frac{1}{I}\sum_{i \in A_n} (\dot{h}_i^{(t)} + c_i^t(z_i - z_i^t)) f_i^{n,m} - \frac{1}{I}\sum_{i \in A'_n} (\dot{h}_i^{(t)} + c_i^t(z_i - z_i^t)) \frac{e^{\boldsymbol{v}_n^T \boldsymbol{f}_i^n} f_i^{n,m}}{\sum_{n' \neq y_i} e^{\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}}} \tag{6.37}$$

where $A_n = \{i : y_i = n\}$ and $A'_n = \{i : y_i \neq n\}$. $D''_{n,m}$ is found to be as follows:

$$D''_{n,m} = \frac{\partial^2 Q_L(\boldsymbol{V}, \boldsymbol{V}^t)}{\partial v_{n,m}^2} = \frac{1}{I}\sum_{i \in A_n} c_i^t (f_i^{n,m})^2 + \frac{1}{I}\sum_{i \in A'_n} \frac{(f_i^{n,m})^2 \exp(\boldsymbol{v}_n^T \boldsymbol{f}_i^n)}{(\sum_{n' \neq y_i} \exp(\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}))^2}$$
$$\{c_i^t \exp(\boldsymbol{v}_n^T \boldsymbol{f}_i^n) - (\dot{h}_i^{(t)} + c_i^t(z_i - z_i^t))(\textstyle\sum_{n' \neq y_i} \exp(\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}) - \exp(\boldsymbol{v}_n^T \boldsymbol{f}_i^n))\} \tag{6.38}$$

For group sparse regularization, i.e., $l_1 - l_2$ regularization, first and second derivatives of the overall objective functions are as follows:

$$E'_{n,m} = D'_{n,m} + \lambda \frac{v_{n,m}}{\sqrt{C_2 + v_{n,m}^2}} \tag{6.39}$$

$$E''_{n,m} = D''_{n,m} + \lambda c(v_{n,m}), \tag{6.40}$$

where $c(v_{n,m})$ is found by (6.32) for optimal curvature and maximum curvature is given in (6.33).

For l_1 norm regularization, we first minimize over the loss part and use the shrinkage operator to incorporate the regularization part. For group sparse regularization, if C_2 value is smaller than ϵ, we reduce the regularization part into l_1 norm regularization. In order to lower the computation time of the first and second derivatives of the surrogate function of loss part, we define three accumalator arrays, namely $\boldsymbol{T} \in \mathbb{R}^{I \times N}$, $\boldsymbol{q} \in \mathbb{R}^I$ and $\boldsymbol{r} \in \mathbb{R}^I$, whose elements are defined as follows:

$$T_{i,n} = \exp\left(\boldsymbol{v}_n^T \boldsymbol{f}_i^n\right); \tag{6.41}$$

$$q_i = \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i}; \tag{6.42}$$

$$r_i = \textstyle\sum_{n' \neq y_i} \exp\left(\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}\right); \tag{6.43}$$

We give the overall solution in Algorithm 2.

Algorithm 2 MM Coordinate descent algorithm

Receive posterior Scores $\{f_i^{n,m} \in \mathbb{R} : i = 1, \ldots, I; n = 1, \ldots, N\ m = 1, \ldots, M\}$

Receive labels $\{y_i \in \{1, 2, \ldots, N\} : i = 1, \ldots, I\}$

Choose the curvature type $\in \{MC, OC\}$

Choose the regularization $\in \{l_1, l_2, l_1 - l_2\}$

Initialize $\boldsymbol{v}_n = \boldsymbol{v}_n^0$ and calculate $z_i = \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i} - \log \sum_{n' \neq y_i} \exp\left(\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}\right)$ for all i

Initialize accumulators $T_{i,n} = \exp\left(\boldsymbol{v}_n^T \boldsymbol{f}_i^n\right)$, $q_i = \boldsymbol{v}_{y_i}^T \boldsymbol{f}_i^{y_i}$ and $r_i = \sum_{n' \neq y_i} \exp\left(\boldsymbol{v}_{n'}^T \boldsymbol{f}_i^{n'}\right)$ for all i, n

Set parameters τ, λ, ϵ

if MC **then**

  $c_i^{(t)} = \frac{1}{2\tau}$ for $i = 1, \ldots, I$

end if

for $t \leftarrow 0, NITER - 1$ **do**

  $\dot{h}_i^{(t)} = -[[z_i < 1 - \tau]] + 1/2\tau^{-1}(z_i - (1 + \tau))[[|z_i - 1| < \tau]]$ for $i = 1, \ldots, I$

  if OC **then**

    $c_i^{(t)} = 1/2|1 - z_i|^{-1}[[|z_i - 1| > \tau]] + 1/2\tau^{-1}[[|z_i - 1| < \tau]]$ for $i = 1, \ldots, I$

  end if

  Decide on coordinate update schedule $\Sigma = [(\sigma_1(l), \sigma_2(l)) : l = 1, \ldots, S]$.

  for $l = 1$ to S **do**

    Set $n = \sigma_1(l)$ (class) and $m = \sigma_2(l)$ (classifier)

    $D'_{n,m} \leftarrow 0, D''_{n,m} \leftarrow 0$

    for i s.t. $y_i = n$ **do**

      $D'_{n,m} := D'_{n,m} + f_i^{n,m}\{\dot{h}_i^{(t)} + c_i^{(t)}(q_i - \log(r_i) - z_i^{(t)})\}$

      $D''_{n,m} := D''_{n,m} + c_i^{(t)}(f_i^{n,m})^2$

    end for

    for i s.t. $y_i \neq n$ **do**

      $D'_{n,m} := D'_{n,m} - T_{i,n}(\dot{h}_i^{(t)} + c_i^{(t)}(q_i - \log(r_i) - z_i^{(t)}))/r_i$

      $D''_{n,m} := D''_{n,m} + (f_i^{n,m})^2 T_{i,n}(c_i^{(t)} T_{i,n} - (\dot{h}_i^{(t)} + c_i^{(t)}(q_i - \log(r_i) - z_i^{(t)}))(r_i - T_{i,n}))/{r_i}^2$

    end for

    $D'_{n,m} := D'_{n,m}/I, \quad D''_{n,m} := D''_{n,m}/I$

if $l_1 - l_2$ **then**
  $C_2 = \sum_{n' \neq n} v_{n',m}^2$
  if $C_2 > \epsilon$ **then**
    $D'_{n,m} := D'_{n,m} + \lambda \frac{v_{n,m}}{\sqrt{C_2 + v_{n,m}^2}}$
    if OC **then**
      $D''_{n,m} := D''_{n,m} + \frac{\lambda}{\sqrt{C_2 + v_{n,m}^2}}$
    else
      if MC **then**
        $D''_{n,m} := D''_{n,m} + \frac{\lambda}{\sqrt{C_2}}$
      end if
    end if
  end if
else
  if l_2 **then**
    $D'_{n,m} := D'_{n,m} + 2\lambda v_{n,m}$
    $D''_{n,m} := D''_{n,m} + 2\lambda$
  end if
end if
$v_{n,m}^{\text{old}} = v_{n,m}, \quad v_{n,m} := v_{n,m} - D'_{n,m}/D''_{n,m}$
if $l_1 \quad or \quad (l_1 - l_2 \quad and \quad C_2 \leq \epsilon)$ **then**
  $v_{n,m} := S(v_{n,m}, \lambda / D''_{n,m})$
end if
$\Delta v_{n,m} = v_{n,m} - v_{n,m}^{\text{old}}$
for i s.t. $y_i = n$ **do**
  $q_i := q_i + \Delta v_{n,m} f_i^{n,m}$
  $T_{i,n} = \exp(q_i)$
end for
for i s.t. $y_i \neq n$ **do**
  $r_i := r_i - T_{i,n}$
  $T_{i,n} := T_{i,n} \exp(\Delta v_{n,m} f_i^{n,m})$
  $r_i := r_i + T_{i,n}$
end for
end for
Compute $z_i^{(t+1)} = q_i - \log(r_i)$ for all i
end for

6.6 Experiments

First, we obtained the results for the ensembles that are used in Chapter 4 and Chapter 5 for the 8 datasets. We used the optimal curvature for the surrogate function. We initialized the combiner with zeros. For stopping criterion, we checked whether δ_{max} is smaller than $\epsilon_2 w_{max}$; where δ_{max} is the maximum change of the weights and w_{max} is the maximum weight. We set the parameters: $\epsilon_2 = 0.05$, $\epsilon = 0.00001$, $\tau = 0.5$, $NITER = 15$. We used the same regularization parameters that are obtained and used in Chapter 4. We compared our algorithm with the solution of the cvx-toolbox, SeDuMi [52]. The results are shown in Table 6.1. We applied Wilcoxon signed ranks test for statistical significance with one tailed significance level of $\alpha = 0.05$. The letters "a,b,c" in the significance column titled *SIG* indicate that the accuracies between our algorithm and SeDuMi for l_2, l_1 and group sparse regularization respectively are statistically significant.

TABLE 6.1: Error percentages with the MM algorithm and the SeDuMi for l_2, l_1, and $l_1 - l_2$ norm regularization. (*mean* ± *standard deviation*).

DB	MM			SeDuMi			SIG
	l_2	l_1	$l_1 - l_2$	l_2	l_1	$l_1 - l_2$	
Segment	4.94 ± 0.95	3.98 ± 1.16	4.58 ± 1.00	3.90 ± 1.00	3.62 ± 0.62	3.74 ± 0.40	ac
Waveform	13.08 ± 0.83	19.14 ± 17.07	29.42 ± 25.63	13.05 ± 0.72	13.46 ± 0.74	13.42 ± 0.76	bc
Robot	3.50 ± 0.62	3.02 ± 0.56	2.93 ± 0.48	2.59 ± 0.33	2.57 ± 0.35	2.49 ± 0.33	abc
Statlog	17.23 ± 2.00	16.86 ± 1.28	34.40 ± 28.16	16.12 ± 1.53	17.45 ± 1.74	17.33 ± 1.42	a
Vowel	10.08 ± 3.07	7.03 ± 2.12	48.77 ± 44.27	7.66 ± 2.29	7.62 ± 2.02	7.17 ± 1.50	a
Wine	1.46 ± 1.41	14.61 ± 27.26	21.12 ± 31.19	1.12 ± 1.40	2.25 ± 1.18	1.91 ± 1.30	
Yeast	41.86 ± 1.54	70.58 ± 17.73	70.90 ± 20.49	40.23 ± 1.29	42.40 ± 4.10	41.19 ± 1.57	abc
Steel	29.29 ± 2.25	28.16 ± 1.26	28.42 ± 1.25	28.27 ± 1.38	28.31 ± 1.39	27.41 ± 1.21	ac

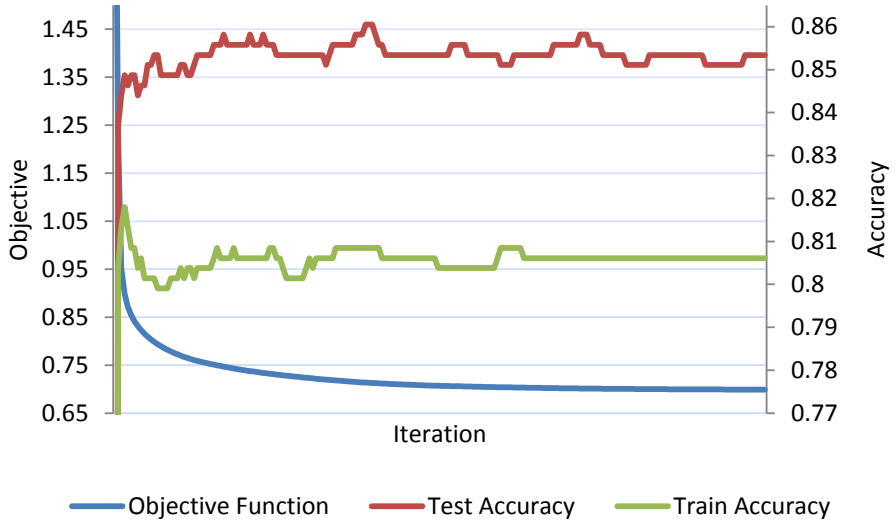
TABLE 6.2: Elapsed CPU times with the MM algorithm and the SeDuMi for l_2, l_1, and $l_1 - l_2$ norm regularization. (*mean* ± *standard deviation*). Bold values are the lowest CPU times among the algorithms for that regularization

DB	MM			SeDuMi		
	l_2	l_1	$l_1 - l_2$	l_2	l_1	$l_1 - l_2$
Segment	**119.24 ± 2.77**	**118.35 ± 4.29**	**45.27 ± 19.95**	309.41 ± 65.13	215.44 ± 79.48	115.74 ± 22.97
Waveform	109.10 ± 3.91	**78.97 ± 26.69**	**21.70 ± 16.48**	**70.71 ± 17.99**	116.46 ± 35.99	60.25 ± 13.00
Robot	154.44 ± 20.10	**119.75 ± 25.45**	**69.88 ± 8.57**	**143.32 ± 47.94**	157.31 ± 40.25	80.40 ± 23.82
Statlog	24.77 ± 1.18	**24.09 ± 3.14**	**7.14 ± 5.31**	**19.24 ± 2.87**	32.30 ± 10.78	15.02 ± 9.06
Vowel	59.51 ± 27.33	77.62 ± 10.53	**22.07 ± 20.12**	**33.30 ± 5.56**	**54.62 ± 6.90**	53.42 ± 24.02
Wine	2.75 ± 1.10	**0.88 ± 0.50**	**0.47 ± 0.28**	**1.47 ± 0.90**	1.87 ± 0.95	1.42 ± 0.77
Yeast	**93.39 ± 8.62**	**40.12 ± 33.38**	**11.92 ± 6.43**	206.98 ± 72.53	75.72 ± 17.97	65.83 ± 23.22
Steel	**100.57 ± 3.76**	**86.14 ± 21.20**	**38.59 ± 10.30**	271.06 ± 57.60	166.04 ± 58.71	105.54 ± 25.23

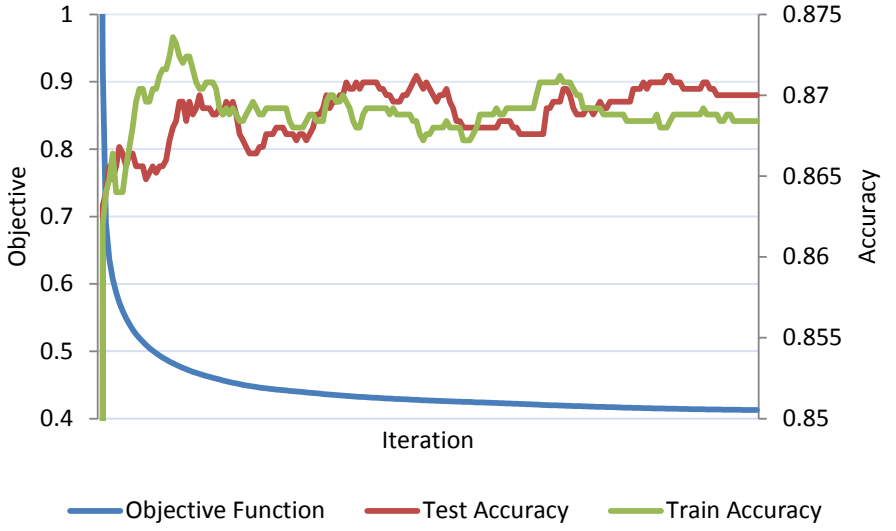
In general, the softmax and Huber loss approximations drop the accuracy. For some databases and for some combination types these accuracy drops are significant. On some stacks of 5×2 cross validation, l_1 norm and group sparse regularization resulted in

high error percentages with the MM algorithm. But the running time of our algorithm is much more smaller compared to the SeDuMi algorithm.

We conducted additional experiments on datasets *Statlog* and *Waveform*. We randomly split the datasets into two as train and test parts and trained base classifiers with the framework described in Chapter 4 as the *diverse ensembles*. We set the parameters of the algorithm as follows: $\lambda = 0.1$, $\tau = 0.5$. We initialize the weights with zeros. It should be noted that results shown here are not directly comparable to the results in Chapter 4 since we do not perform 5x2 CV here. First we show the change in objective values, train and test accuracies for the two datasets in Figures 6.5(a) and 6.5(b). We

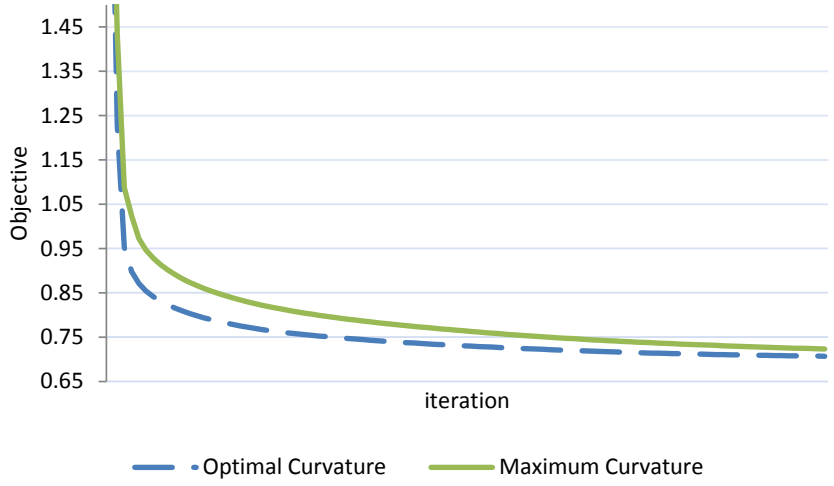


(a) *Statlog* Dataset

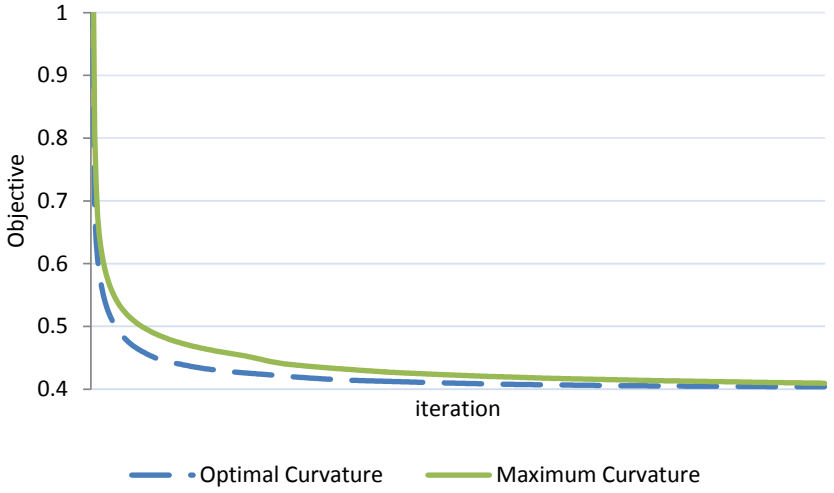


(b) *Waveform* Dataset

FIGURE 6.5: Change in train, test accuracies and objective function for *Statlog* and *Waveform* datasets.



(a) *Statlog*

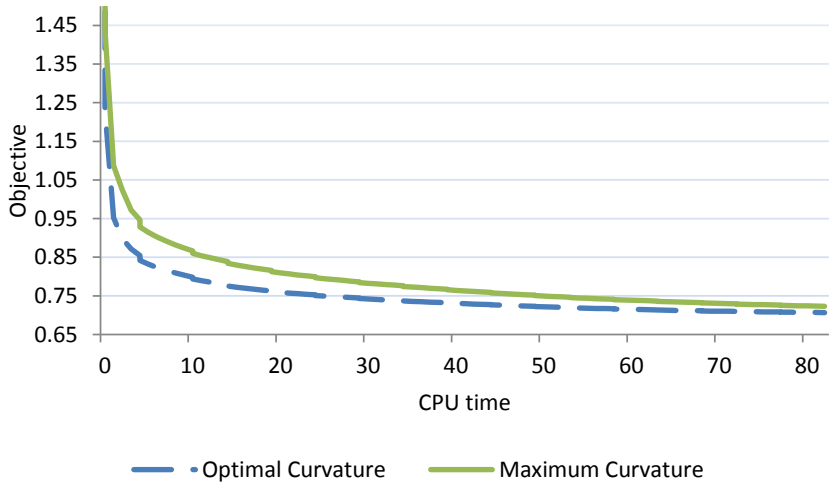


(b) *Waveform*

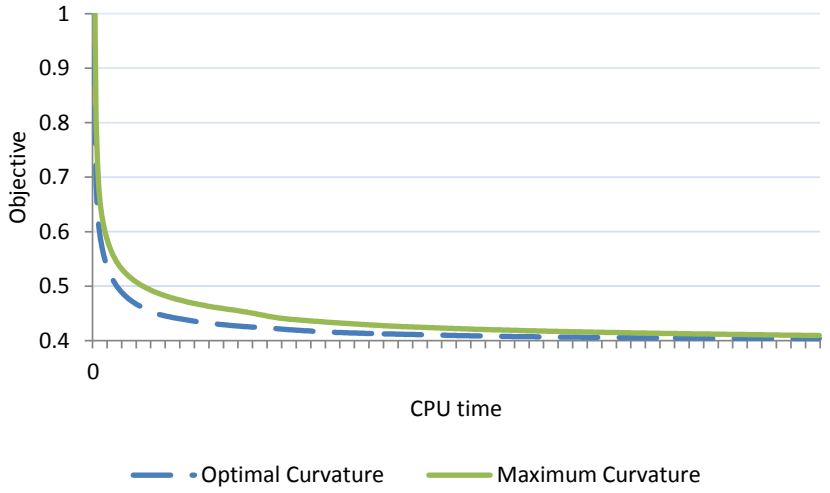
FIGURE 6.6: Comparison of optimal curvature and maximum curvature with respect to iteration *Statlog* and *Waveform* datasets.

see the monotonic decrease of the objective values, but the accuracies contain changes even when the objective value gets close to the convergence point. Even though these changes are not significant, it shows us that the stopping criteria cannot be based on the change in the objective value, but it should depend on the change in the weights.

Next we show the comparisons of optimal curvature and maximum curvature with respect to iteration number in Figures 6.6(a) and 6.6(b) and with respect to time in Figures 6.7(a) and 6.7(b). We see that optimal curvature outperforms the maximum curvature. Maximum curvature will result in much higher convergence time, especially if the stopping criteria is tight. Another issue regarding group sparsity is the parameter ϵ that thresholds C_2 values. When we set it to zero, i.e., not thresholding, the weights will



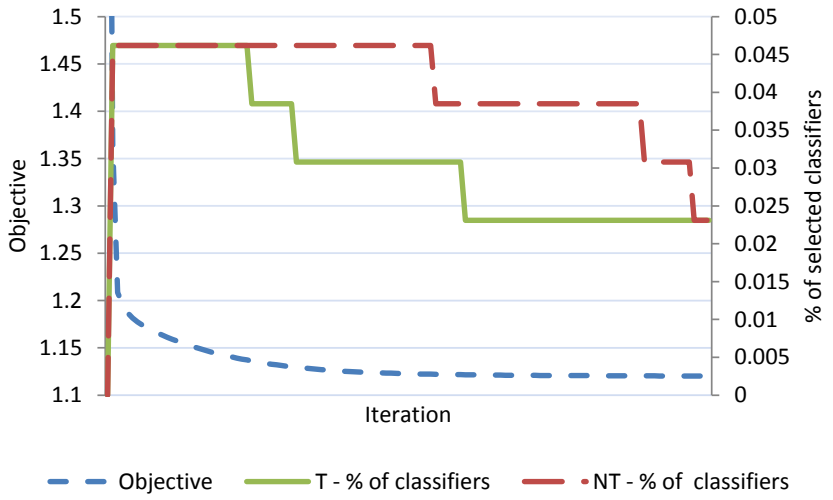
(a) *Statlog*



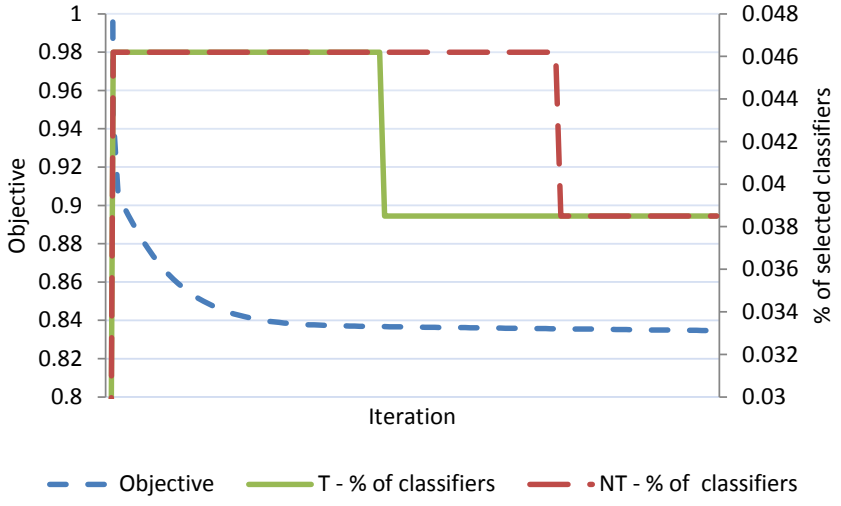
(b) *Waveform*

FIGURE 6.7: Comparison of optimal curvature and maximum curvature with respect to CPU time for *Statlog* and *Waveform* datasets.

never be zero, but get close to zero. This will lead to slower convergence and this effect is shown in Figures 6.8(a) and 6.8(b). We see that, with $\epsilon = 0.00001$, sparse solution is obtained much faster and the objective value still monotonically decreases, causing no convergence trouble.



(a) *Statlog* dataset



(b) *Waveform* dataset

FIGURE 6.8: Change in the percentage of selected classifiers for no-thresholding (NT) and thresholding (T) with $\epsilon = 0.00001$ for group sparse regularization for *Statlog* and *Waveform* datasets.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we worked on improving a novel approach for classifier combination: stacked generalization. Our contributions improved the performance of stacking in terms of both accuracy and speed.

For accuracy improvement, we proposed using the hinge loss function for learning the weights. This loss function results in a combiner such that the margin is maximized and our extensive experiments show that there is a statistically significant increase in the accuracy for all combination types. Our experiments involve three different combination types; namely weighted sum (WS), class-dependent weighted sum (CWS) and linear stacked generalization (LSG); that are used in the literature and provides a good comparison between these combination types. Experiments suggest that CWS outperforms the WS combination significantly. However, higher accuracies of the LSG combination with respect to the CWS combination are not statistically significant. Considering the much higher number of parameters of the LSG combination with respect to CWS, we suggest to use the CWS combination.

In order to speed up the test process, we implemented sparse regularizations in our experiments. Besides the conventional l_1 regularization that has been used in the literature for stacking, we proposed to use group sparse regularization, in which posterior scores of a particular classifier forms a group. Experiments show the superiority of group sparse

regularization as compared to the l_1 norm regularization, in terms of accuracy, number of selected classifiers, and robustness.

In Chapter 5, we worked on non-linear combinations under the stacking framework. Since the LSG combination is equivalent to applying a linear classifier to the posterior scores of base classifiers as if they comprise a new set of features, we can obtain a non-linear version of it by using the kernel trick. However, non-linear versions of WS and CWS combinations are not straightforward. For WS combination, we developed a method, in which level-1 data are transformed, together with the class labels, into a binary classification dataset. We implement the combiner by applying a binary classifier on this transformed dataset. For the least-squares loss function, we apply LS-SVM to the binary dataset. This method is not necessarily exactly equivalent to the original problem for a given loss function such as the hinge loss, but it is equivalent for the LS loss function. For the CWS combination, we apply the same procedure for each class, obtain N different binary datasets and train a binary classifier from each of them to obtain the scores of different classes. Experiments show that we can obtain the best accuracies by using the non-linear combiners. But the results are not statistically significant.

Since there is not a specific optimization algorithm for CWS combination in the literature; we derive a primal optimization algorithm for the CWS combination using MM algorithms in Chapter 6. Since MM algorithms ensure monotonic decrease of the objective function and our objective function is convex, our proposed algorithm converges to the optimal solution. We find the optimal curvatures and maximum curvatures for our problem, and observe that optimal curvature outperforms the maximum curvature in terms of speed of convergence. We also see that, by assuming the weights of a classifier to be exactly zero when they are close to zero, we speed up the the algorithm for group sparse regularization.

7.2 Future Work

In this section, we present possible future directions. A major contribution of the thesis is using group sparse regularization of the weights for learning the combiner, and in this case, the regularization parameter (λ) has two main effects on the resulting combiner as mentioned. First, it determines how much the combiner should fit the data, as in

the case of the l_2 norm regularization. Second, it effects the number of selected base classifiers. These two effects play a major role in the performance of the combiner, so it could be better to adjust these effects separately. One possible way to do this is first doing cross validation with $l_1 - l_2$ norm regularization, selecting the appropriate regularization parameter and keeping track of the base classifiers that are selected, then doing another cross validation with the l_2 norm regularization but with only the base classifiers that are selected in the previous cross-validation, choosing the λ that results in the best accuracy, and finally training the combiner with the final selected λ and the base classifiers with the l_2 norm regularization.

Another possible extension of the thesis could be about the nonlinear combination. We used the RBF kernel, which is favorable for classification problem in general; but other kernels could result in better accuracy for classifier combination problem. One possible direction could be deriving a kernel function specifically for classifier combination problem.

In Chapter 6, we derived an optimization algorithm for the CWS combination with l_2, l_1 and $l_1 - l_2$ norm regularizations. Unfortunately, the algorithm results in very low accuracies for some stacks of some datasets for l_1 and group sparse regularizations, which use the shrinkage thresholding operator. One possible future work is investigating the reason behind this and finding a solution to the problem.

Bibliography

[1] Thomas G. Dietterich. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*, pages 1–15. Springer-Verlag, 2000.

[2] R Polikar. Ensemble based systems in decision making. *Ieee Circuits And Systems Magazine*, 6(3):21–45, 2006.

[3] Leo Breiman. Arcing Classifiers. *The Annals of Statistics*, 26(3):801–824, 1998. ISSN 00905364.

[4] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2.

[5] Tin Kam Ho. *Multiple Classifier Combination: Lessons and Next Steps*, volume 47 of *Series in Machine Perception and Artificial Intelligence*, chapter 7, pages 171–198. World Scientific, 2001.

[6] I.S. Topkaya, M.U. Sen, M.B. Yilmaz, and H. Erdogan. Improving speech recognition with audio-visual tandem classifiers and their fusions. In *Signal Processing and Communications Applications (SIU), 2011 IEEE 19th Conference on*, pages 407 –410, april 2011. doi: 10.1109/SIU.2011.5929673.

[7] I.S. Topkaya, M.B. Yilmaz, M.U. Sen, Tarasov A., and H. Erdogan. An audio-visual speech recognition system with live inputs. In *Proceddings eNTERFACE'10, Summer Workshop on Multimodal Interfaces*, pages 48 –57, 2010.

[8] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004. ISBN 0471210781.

[9] Leo Breiman and Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[10] Robert E. Schapire. The strength of weak learnability, 1990.

[11] Michael Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 1988.

[12] *The combining classifier: to train or not to train?*, volume 2, 2002. doi: 10.1109/ICPR.2002.1048415.

[13] David H. Wolpert. Stacked generalization. *Neural Netw.*, 5(2):241–259, 1992. ISSN 0893-6080.

[14] Ludmila I. Kuncheva, James C. Bezdek, and Robert P. W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34: 299–314, 2001.

[15] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[16] Yi Lu. Knowledge integration in a multiple classifier system. *Applied Intelligence*, 6:75–86, 1996. ISSN 0924-669X. URL `http://dx.doi.org/10.1007/BF00117809`. 10.1007/BF00117809.

[17] Galina Rogova. Combining the results of several neural network classifiers. *Neural Netw.*, 7:777–781, May 1994. ISSN 0893-6080. doi: 10.1016/0893-6080(94)90099-X.

[18] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *J. Artif. Int. Res.*, 10:271–289, May 1999. ISSN 1076-9757.

[19] Seewald AK. *Towards understanding stacking - studies of a general ensemble learning scheme*. PhD thesis, TU Wien, 2003.

[20] Naonori Ueda. Optimal linear combination of neural networks for improving classification performance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(2):207–215, 2000. ISSN 0162-8828.

[21] H. Erdogan and M.U. Sen. A unifying framework for learning the linear combiners for classifier ensembles. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2985 –2988, aug. 2010.

[22] Michael LeBlanc and Robert Tibshirani. Combining estimates in regression and classification. Technical report, Journal of the American Statistical Association, 1993.

[23] Kai Ming Ting and Ian H. Witten. Stacking bagged and dagged models. In *In Proc. 14th International Conference on Machine Learning*, pages 367–375. Morgan Kaufmann, 1997.

[24] Sam Reid and Greg Grudic. Regularized linear models in stacked generalization. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, MCS '09, pages 112–121, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02325-5.

[25] Li Zhang and Wei-Da Zhou. Sparse ensembles using weighted combination methods based on linear programming. *Pattern Recognition*, 44(1):97 – 106, 2011. ISSN 0031-3203.

[26] H. Erdogan and M.U. Sen. A combined approach to regularized linear combiner learning. In *Signal Processing and Communications Applications Conference (SIU), 2010 IEEE 18th*, pages 483 –486, april 2010.

[27] Yann Lecun, Sumit Chopra, Raia Hadsell, Fu Jie Huang, G. Bakir, T. Hofman, B. Scholkopf, A. Smola, and B. Taskar (eds. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

[28] J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9:293–300, 1999. ISSN 1370-4621.

[29] Ulrich H.-G. Kressel. *Pairwise classification and support vector machines*, pages 255–268. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19416-3.

[30] *Comparison of classifier methods: a case study in handwritten digit recognition*, volume 2, October 1994.

[31] Jrg Kindermann, J Org Kindermann, Edda Leopold, and Gerhard Paass. Multi-class classification with error correcting codes. Technical report, In, 2000.

[32] J. Weston and C. Watkins. Multi-class support vector machines, 1998. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9594`.

[33] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *In Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 35–46, 2000.

[34] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99:67–81, 2004.

[35] Emmanuel Monfrini and Yann Guermeur. A quadratic loss multi-class svm. *CoRR*, abs/0804.4898, 2008.

[36] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:2003, 2001.

[37] A. Majumdar and R.K. Ward. Classification via group sparsity promoting regularization. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 861 –864, april 2009. doi: 10.1109/ICASSP.2009.4959720.

[38] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html`.

[39] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998.

[40] Duin R.P.W., Juszczak P., Paclik P., Pekalska E., de Ridder D., Tax D.M.J., and Verzakov S. *PRTools4.1, A Matlab Toolbox for Pattern Recognition*, 2007. Delft University of Technology.

[41] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[42] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21, April 2011. Available at http://cvxr.com/cvx.

[43] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006. ISSN 1532-4435.

[44] M.U. Sen and H. Erdogan. Nonlinear classifier combination for simple combination types. In *Signal Processing and Communications Applications (SIU), 2011 IEEE 19th Conference on*, pages 1032 –1035, april 2011.

[45] A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[46] G. Wahba. *Spline models for observational data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1990.

[47] James M. Ortega and Werner C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. ISBN 0-89871-461-3.

[48] David R. Hunter and Kenneth Lange. A tutorial on mm algorithms. *American Statistician*, 58:30–37, 2004.

[49] H. Erdogan. Tubitak research project 1. progress report, project no: 110e041. Technical report, Sabanci University, 2011.

[50] Jan de Leeuw and Kenneth Lange. Sharp quadratic majorization in one dimension. *Comput. Stat. Data Anal.*, 53:2471–2484, May 2009. ISSN 0167-9473.

[51] M.W. Jacobson and J.A. Fessler. An expanded theoretical treatment of iteration-dependent majorize-minimize algorithms. *Image Processing, IEEE Transactions on*, 16(10):2411 –2422, oct. 2007. ISSN 1057-7149.

[52] Jos F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones, 1998.