INAUGURAL — DISSERTATION

zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich-Mathematischen Gesamtfakultät
der
Ruprecht – Karls – Universität
Heidelberg

vorgelegt von
Dipl.-Ing. Thomas Hörnlein
aus Hildburghausen
Tag der mündlichen Prüfung: 21.07.2010

Thema

Boosted Feature Generation for Classification Problems Involving High Numbers of Inputs and Classes

Gutachter: **Prof. Dr. Bernd Jähne**
Prof. Dr. Dr. h. c. Hans Georg Bock

Abstract

Classification problems involving high numbers of inputs and classes play an important role in the field of machine learning. Image classification, in particular, is a very active field of research with numerous applications. In addition to their high number, inputs of image classification problems often show significant correlation. Also, in proportion to the number of inputs, the number of available training samples is usually low. Therefore techniques combining low susceptibility to overfitting with good classification performance have to be found. Since for many tasks data has to be processed in real time, computational efficiency is crucial as well.

Boosting is a machine learning technique, which is used successfully in a number of application areas, in particular in the field of machine vision. Due to it's modular design and flexibility, Boosting can be adapted to new problems easily. In addition, techniques for optimizing classifiers produced by Boosting with respect to computational efficiency exist. Boosting builds linear ensembles of base classifiers in a stage-wise fashion. Sample-weights reflect whether training samples are hard-to-classify or not. Therefore Boosting is able to adapt to the given classification problem over the course of training.

The present work deals with the design of techniques for adapting Boosting to problems involving high numbers of inputs and classes. In the first part, application of Boosting to multi-class problems is analyzed. After giving an overview of existing approaches, a new formulation for base-classifiers solving multi-class problems by splitting them into pair-wise binary subproblems is presented. Experimental evaluation shows the good performance and computational efficiency of the proposed technique compared to state-of-the-art techniques.

In the second part of the work, techniques that use Boosting for feature generation are presented. These techniques use the distribution of sample weights, produced by Boosting, to learn features that are adapted to the problems solved in each Boosting stage. By using smoothing-spline base classifiers, gradient descent schemes can be incorporated to find features that minimize the cost function of the current base classifier. Experimental evaluation shows, that Boosting with linear projective features significantly outperforms state-of-the-art approaches like e.g. SVM and Random Forests.

In order to be applicable to image classification problems, the presented feature generation scheme is extended to produce shift-invariant features. The utilized features are inspired by the features used in Convolutional Neural Networks and perform a combination of convolution and subsampling. Experimental evaluation for classification of handwritten digits and car side-views shows that the proposed system is competitive to the best published results. The presented scheme has the advantages of being very simple and involving a low number of design parameters only.

Zusammenfassung

Klassifikationsprobleme, welche hohe Anzahlen von Eingangsmerkmalen und Klassen aufweisen spielen eine wichtige Rolle auf dem Gebiet des Maschinenlernens. Besonders Bildverarbeitung stellt ein sehr aktives Forschungsfeld mit unzähligen Anwendungen dar. Häufig sind im Verhältnis zur Anzahl der Eingangsmerkmale nur wenig Trainingsbeispiele verfügbar. Deswegen müssen Techniken gefunden werden, die sich nicht zu stark an die Trainingsdaten anpassen. Aufgrund von Echtzeit-Anforderungen vieler Anwendungen, ist effiziente Implementierbarkeit ebenso von großer Bedeutung.

Boosting ist ein Lernverfahren, das insbesondere im Gebiet der Bildverarbeitung erfolgreich eingesetzt wird. Boosting konstruiert lineare Ensembles von Basis-Klassifikatoren in einer rundenbasierten Vorgehensweise. Modulares Design und hohe Flexibilität ermöglichen einfache Anpassung an neue Problemstellungen. Durch entsprechende Techniken kann der Rechenaufwand von Boosting-Klassifikatoren optimiert werden. Mit den Trainingsbeispielen assoziierte Gewichte weisen auf schwer zu klassifizierende Beispiele hin, was ein Anpassung an ein gegebenes Problem während des Trainings ermöglicht.

Die vorliegende Arbeit befasst sich mit dem Entwurf von Techniken welche Boosting besser an Klassifikationsprobleme mit vielen Eingangsmerkmalen und Klassen anpassen. Im ersten Teil wird die Anwendung von Boosting für Multiklassenprobleme analysiert. Nach einem Überblick über existierende Verfahren wird eine neue Formulierung für Basis-Klassifikatoren vorgestellt, welche auf einer Zerlegung des Multiklassenproblems in binäre Teilprobleme basieren. Experimente zeigen die gute Klassifikationsleistung und Recheneffizienz im Vergleich zum Stand der Technik.

Im zweiten Teil der Arbeit werden Techniken, welche Boosting zum Lernen von Merkmalskombinationen verwenden, vorgestellt. Dabei werden die Gewichte der Trainingsbeispiele herangezogen um an das aktuelle Trainingsproblem angepasste Merkmale zu lernen. Durch die Verwendung geglätteter Splines als Basisklassifikatoren, können Gradienten-Abstiegsverfahren verwendet werden um gute Merkmale zu finden. Experimente zeigen, dass Boosting mit linearen projektiven Merkmalen signifikant bessere Klassifikationsleistung als andere populäre Verfahren wie beispielsweise *SVM* und *Random Forests* erreichen.

Das vorgestellte Verfahren wird für die Anwendung auf Bilddaten erweitert, indem verschiebungsinvariante Merkmale trainiert werden. Diese sind inspiriert von Merkmalen welche in *Convolutional Neural Networks* eingesetzt werden und eine Kombination aus Faltung und Unterabtastung durchführen. Experimente an den Beispielen Klassifikation von Ziffern und von Seitenansichten von PKWs zeigen, dass das vorgestellte System Klassifikationsleistung vergleichbar zu den besten veröffentlichen Ergebnissen erreicht. Vorteile liegen in der einfachen Struktur und geringen Anzahl einstellbarer Trainingsparameter.

Danksagung

Mein Dank gilt allen, die mich bei der Erstellung der vorliegenden Arbeit unterstützt haben. Als Erstes möchte ich Prof. Dr. Bernd Jähne vom Interdisziplinären Zentrum für Wissenschaftliches Rechnen der Universität Heidelberg für die Betreuung meiner Dissertation danken. Über die gesamte Zeit hat er mit Diskussionen und Vorschlägen geholfen die Arbeit voran zu treiben.

Für die materielle Unterstützung danke ich der Robert Bosch GmbH. Weiterhin möchte ich der Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences für die Unterstützung von Konferenzbesuchen und die Möglichkeit des Besuchs weiterbildender Seminare danken.

Den Hildesheimer Doktoranden, sowie den Kollegen bei Bosch in Hildesheim möchte ich für das entspannte und kreative Umfeld sowie die vielen lehrreichen Diskussionen und Gespräche danken. Insbesondere gilt mein Dank Jochen Wingbermühle, meinem fachlichen Betreuer bei Bosch. Den Kollegen am HCI in Heidelberg danke ich für die Möglichkeit meine Arbeit in Vorträgen vorzustellen und die damit verbundenen hilfreichen Diskussionen.

Nicht zuletzt möchte ich meiner zukünftigen Frau Katharina danken, die mir die nötige emotionale Unterstützung gegeben hat, um die vorliegende Arbeit fertig zu stellen. Ein besonderes Dankeschön gilt auch meiner Tochter Paula, die mir durch ihre Pünktlichkeit die nötige Zeit gegeben hat die Arbeit zu beenden.

Contents

1. Introduction

1.1. Motivation and Background

Classification problems occur in a wide range of applications. Due to the ever growing amount of recorded data and the resulting need for automation, the demand for systems performing automatic data interpretation is high. An important field of automated data interpretation is machine vision. Machine vision helps to solve a range of problem which would otherwise have to be solved using human interaction. Some examples of application problems, that can be solved using image processing are the following:

- Recognition of handwritten postal codes on envelopes. The digits are recorded using digital cameras. The resulting intensity values are quantized into gray scales. The output is then processed to classify the depicted digits.
- Detection of humans in surveillance scenes. The recorded image streams are analyzed in order to find patterns of appearance and motion indicating presence of persons in the observed scene.
- Driver assistance systems: data from different sensors (radar, video) are used to assess the situation a vehicle is in, recognize traffic signs or lane markings, etc.
- Identification of skin cancer. Images of moles are analyzed for specifics of malignant melanoma. The system helps to achieve earlier diagnosis and therefore improves the probability of successful treatment.

A wide range of sensors for collecting visual data is available. Over recent years, the density of cameras grew dramatically. Reasons for that are, for example, the high number of surveillance cameras or the integration of cameras into mobile devices. For digital signal processing, the image data captured by the cameras is transformed into a matrix of numbers (image pixels), using for example CMOS technology. Data processing can be performed on standard digital computer hardware, for example multi purpose processors, GPUs (graphics processing units), FPGAs (field programmable gate arrays) or DSPs (digital signal processors). The hardware used for a particular application heavily

depends on the requirements and constraints of that application. Parallel to the improvement of performances of both cameras and processing units, their prices reduced due to their introduction into the mass market. The presence of cheap and powerful hardware for image processing tasks is another key factor for it's increasing commercial success. However, though the increase of processing power of the utilized devices is dramatic, processing time is still often a limiting factor for practical applicability of machine vision techniques.

Vision is the dominant sense for humans and the majority of higher animals. Visual information are interpreted to build a model of the scene, in which an organism resides. The visual system solves very complex tasks, including localization, estimation of distances and distance ratios, as well as identification and categorization of objects. For the human observer it is very simple to solve image classification tasks. In particular, humans show astonishing generalization abilities and are able to learn new categories of objects using only a low number of examples. In addition the visual system is able to perform it's tasks under difficult conditions, including poor image quality and various environment conditions. These skills can, at least partially, be explained by the large amount of prior knowledge, that humans collect during the course of their lives. The visual cortex of humans and other organisms use massive parallel processing to achieve the stated capabilities. How the image processing in the visual cortex works is not fully understood and subject to ongoing research.

Compared to the capabilities of the human visual cortex, machine vision still has a long way to go. The problems that can be solved satisfactory using machine vision often have strong limitations, for example putting constraints on illumination, object appearance and number of object categories. On the other hand, large improvements have been achieved in recent years leading to algorithms that solve complex problems under "real world" conditions. In some aspects machine vision already has advantages over biological systems. For example, humans grow tired when performing monotonous activities over a long period of time. Machine vision algorithms can help to relieve the human operator and thereby reduce the risk of mistakes. Also, machine vision systems are very good in performing exact measurements.

Different philosophies for solving image classification problems exist. The most successful approaches are based on learning the solution for the given problem from examples-in contrast to handcrafted expert systems. One group of approaches attempts to model the functionality of the human visual cortex. Though the potential of this approach is indisputably, in many cases it fails produce results feasible for practical applications. This is mainly due to the unavoidable simplifications in the modeling process and the lack of understanding of the processes in the visual cortex.

Another group of approaches interprets the image processing task from a more mathematical point of view. The training samples are interpreted as a number of points in a high dimensional feature-space, spanned by the input pixels. The task to be solved is to approximate the distribution of the object categories in the feature-space. These approaches have the advantage that the underlying mechanisms are - in general - well understood. In addition, the resulting models tend to be less complex and therefore are more feasible for implementation in practical systems. One example of techniques that approximate class distributions in feature space is Boosting. It was successfully applied in a range of image processing tasks over the last few years. One reason of it's success is the availability of techniques for building very efficient classifiers, namely the use of classifier cascades. This work deals with extensions for Boosting, which help to improve it's performance for image classification problems and extend it's range of possible applications.

1.2. Objectives

The main objective of this work is to study the use of Boosting for solving complex image classification problems. In particular, problems involving high numbers of inputs or classes are of interest. In order to improve the Performance of Boosting for these scenarios, techniques for solving multi-class problems and feature-generation techniques are analyzed.

Image classification problems have a special character, when compared to other classification problems. In general, the number of input features, corresponding to the input pixels of the image, are very high. In relation to the high number of inputs, the number of samples is usually rather small. In addition, many applications require the input data to be processed in real time. To achieve good performance under these conditions, algorithms have to be found, that take into account the characteristics of image classification problems and, at the same time, can be implemented flexibly and make efficient use of computational resources. According to the high number of object categories found in real world scenarios and the particular application, approaches are needed, that are able to discriminate a high number of classes efficiently. This is in particular important, since human visual object categorization heavily depends on using context information. Utilizing context information to solve image classification tasks has been shown to improve performance [Lampert and Blaschko, 2008]. Generating context information, however, requires the ability to solve multi-class problems.

To solve these problems, the proposed techniques should be designed with a focus on achieving good performance and at the same time having low compu-

tational requirements. In order to achieve good performance, the information of the inputs features needs to be utilized efficiently. In many cases using the input features directly doesn't lead to the aspired results. In the present work, feature generation techniques are presented, that help to use the information more efficiently. In addition, techniques for using features more efficiently in multi-class context are discussed.

Another important aspect of the proposed classification techniques are techniques for regularization of the trained classifiers. Regularization is necessary to prevent the classifier from fitting to the training data too closely, which is known to increase error rates on unseen data. Therefore flexibility of the trained models needs to be adjustable in order to prevent overfitting to the training data. Regularization is even more important, when the number of training samples is small compared to the number of input features.

Also, the proposed system should be able to adapt to new problems flexibly. The system should allow additional constraints and information about the problem at hand to be incorporated easily. The resulting techniques should not be restricted to be used for image classification problems.

Finally, the proposed techniques should be compared to other state-of-the-art approaches in order to judge the performances. Empirical evaluation of the techniques is especially important, since theoretical results for performances of classifiers can't be used to assess performances quantitatively. Statistical tests will be used to assess significance of results where possible.

1.3. Contribution

The present work deals with design and evaluation of techniques for extending Boosting [Freund and Schapire, 1995]. The goal is to improve the performance and applicability of Boosting for image classification problems. Boosting is a technique that combines a number of simple rules (weak- or base classifiers) to form a "strong" rule, which solves the given classification problem. In order to compose the decision of the ensemble, the outputs of the simple rules are combined by performing a majority vote. The generation of the simple classification rules is controlled by assigning weights to each training sample. Sample weights are initialized with a uniform distribution over the training samples. After each stage of training, weights of samples - that were assigned wrong outputs by the new simple rule - are increased. Generation of simple rules puts more emphasis on samples with high weights. Therefore, boosted classifiers adapt to the training problem in each round. Note that boosting is a meta-technique that can be used to combine arbitrary base classifiers.

Numerous flavors of Boosting can be found in the literature. While there are

good motivations for use of each of the flavors, the observed performance differences between them are often marginal. Accordingly, this work doesn't propose new Boosting flavors. Instead a standard Boosting approach (GentleBoost, see Friedman *et al.* [1998]) is used for all experiments. GentleBoost has been shown to achieve competitive performance to other Boosting techniques. In addition, it is very simple to implement and numerically stable. Instead of proposing new Boosting flavors, this work focuses on the design of new techniques for training base classifiers, that help to improve the performance of the boosted ensemble. Optimizing the performance of boosted ensembles by improving base classifiers has not yet achieved much attention in the literature and therefore hold potential for further performance improvements.

Boosting is well suited for solving image classification problems with real time requirements. One advantage of Boosting over other techniques is that special techniques (e.g. cascades) can be used to dramatically reduce computational load, while preserving classification performance. Another advantage is that Boosting can be used for feature selection. That is, given a large set of candidate features, Boosting can select a subset of features that solves a classification problem. Due to it's stage-wise function, Boosting scales well with the complexity of the problem at hand. Only few base classifiers are needed to solve simple classification problems, while the number increases as the complexity of the problem grows. Boosting has a low number of adjustable parameters, so effort for parameter selection is manageable.

On the other hand, Boosting has a number of disadvantages too. For example, it is almost always used with decision tree base classifiers. Decision trees, however, are known to be susceptible to overfitting. When used for feature selection, the initial feature set has to contain good features that are feasible to solve the given classification problem. In order to provide such a set of features, one has to have profound knowledge about the classification problem at hand. That degree of knowledge is not always available.

There is only a limited amount of comparative studies on Boosting for multi-class problems. The existing studies focus on classification performance and don't examine optimization with respect to computational efficiency. One contribution of this work is an experimental comparison of Boosting for multi-class problems. For this, standard wrapper approaches, as well as techniques specific to Boosting are analyzed. The utilized cost functions of multi-class Boosting approaches are compared. The results are used to design a new base-classifier scheme, based on pairwise class comparisons, which achieves good classification performance and is computational efficient.

In order to improve the generalization performance of Boosting, the use of base classifiers returning smooth output functions in feature-space are analyzed.

Smoothing splines, as an example of these class of base classifiers, are discussed. Classes probabilities are assumed to show smooth distribution in feature space for real world problems. Therefore, by using smooth base classifiers, the susceptibility of Boosting to overfitting should be reduced. Smooth functions are better suited to approximate the class distributions usually need less degrees of freedom to achieve the same quality of approximation. Another advantage is the differentiability of smooth base classifiers. Based on this, a scheme for training linear features is designed. The trained features optimize the cost functions of the base classifiers trained in each stage of Boosting and therefore improve convergence. By using the ability of Boosting to adjust to the problem at hand, features can be calculated during the course of training. Training of features "on the fly" alleviates the need for providing a suitable feature set.

The proposed scheme for generating linear features for general classification problems is extended for application to image classification problems. Linear features don't achieve performance competitive to other specialized techniques, adapted to image classification tasks. Therefore a scheme for training shift-invariant features, based on boosting smooth functions, is proposed. Backpropagation training is used to train the shift-invariant features. Extensions to other feature types is straightforward. Since the sample weights returned by boosting are used, the shift-invariant features too adapt to the given problem and remove the need for providing an initial feature-set.

1.4. Outline of the Thesis.

Since the techniques presented in this work are based on Boosting, an introduction to Boosting is presented in Chap. 2. The base classifiers used most frequently with Boosting are discussed. In particular, base-classifiers returning smooth output functions are discussed (Sect. 2.4.3). In order to achieve good generalization performance, the capacity of a classifier needs to be adjusted to the given classification problem. Section 2.5.4 discusses, how the outputs of the base-classifiers and thereby the output of the boosted ensemble can be regularized. Penalization of roughness of the output-function in feature space delivers an appropriate measure, which can be used with arbitrary base-classifiers. In addition to the discussions on Boosting, a range of state-of-the-art classification approaches are introduced, e.g. Support Vector Machines and Random Forests. These are used for reference in the following chapters. Also the understanding of other classification approaches compared to Boosting helps to show the particular properties of Boosting that make it well suited for the problems discussed in this work.

For many image processing problems the number of classes involved is

higher than two. Efficient algorithms to solve multi-class problems using Boosting are discussed in Chap. 3. After describing the problem formulation, standard approaches for solving multi-class problems are discussed. These approaches construct multi-class classifiers by combining a number of binary classifiers (multi-class wrappers, Sect. 3.4). Multi-class wrappers can be used with arbitrary classifiers. On the other hand, they don't incorporate prior knowledge about the particulars of the used classifiers. Section 3.5 introduces specialized techniques that extend Boosting to multi-class problems. They are based on the multi-class wrappers discussed previously. However, they are tailored to take into account the specifics of Boosting, for example the stage-wise construction of the final classifier. Out of the numerous approaches proposed in the literature, the fundamental approaches are presented. The cost functions of the approaches are compared in order to better understand their function. Approaches using base classifiers building pairwise classifiers provide the best qualitative approximation of the empirical error function. Their drawback is that the number of pairwise classifiers grows quadratically with the number of classes involved in the classification problem. In Sect. 3.5.5 a new formulation for training of base-classifiers with pairwise class comparisons is proposed. The new formulation is computational efficient and scales linearly with number of classes. The discussed approaches for multi-class Boosting are compared empirically in Sect. 3.6.

For many classification problems, the provided features are not optimal for class separation. In addition, whether features are suitable to solve a given problem or not strongly depends on the incorporated classifiers. In Chap. 4 a technique for training augmented features in Boosting context is presented. The new features are linear combinations of the input features. Using these features, problems - where the classes are non separable using the input features - can be solved. By using differentiable base classifiers, namely smoothing splines, gradient descent schemes can be used to learn features optimizing base classifier costs in each stage of Boosting. Since each base classifiers works on a one-dimensional projection of the data, the added features do not lead to overfitting of the classifier. In experimental section (Sect. 4.4), the performance of the proposed classification scheme is compared to state-of-the-art classifiers. Statistical evaluation on a set of 30 publicly available data sets is conducted to show the significance of the results.

Using the scheme for training linear features, presented in Chap. 4, will not yield satisfactory performance for image classification problems. The reason is that the input features are treated as independent in feature construction. This assumption is, however, not true for image classification problems, where neighboring pixels usually show strong correlation. Chapter 5 discusses, how

prior knowledge about the nature of image processing problems can be incorporated in order to improve the classification performance. In a first step, local features are constructed. For local features the support on the input image is limited to a small neighborhood of pixels. This already significantly improves the resulting classification performance. In cases, where overfitting is critical, a roughness penalty can be applied to the trained features. For many image classification problems, small local shifts of the features on the input patch should not affect the output of the classifier, that is: features should be shift-invariant. Consequently, many techniques for image processing utilize shift-invariance to optimize classification performance. One possibility for incorporating shift-invariance into the proposed feature generation scheme is using features performing convolution followed by subsampling. In order to be able to learn these features, the feature generation scheme from Chap. 4 is extended to be able to train non-linear features. To achieve this, backpropagation training is used. With that shift-invariant features for image classification in Boosting context is straightforward. The performance of the proposed approach is evaluated on two image processing problems: USPS handwritten zip digits and UIUC car side views database. The results are compared to published performances of state-of-the-art classifiers.

2. Classification

This chapter introduces classification techniques used through the following chapters. *Boosting* classifiers introduced by Friedman *et al.* [1998] are introduced in more depth than the others, since they represent the basis for the techniques introduced in this work. Boosting is a technique, used to combine weak classifiers to form strong ensemble-classifiers. Examples of this "weak" classifiers - e.g. neural networks, decision trees and smoothing splines - are introduced. In addition, state-of-the art classification techniques that are used as references in the following chapters are introduced. Compared to these other classifiers, the specifics of Boosting are pointed out.

2.1. Basic Concepts

A classification problem is given as a collection of N samples $(\mathbf{x}_i, y_i),\ i = 1, \ldots, N$. Each sample is described by a feature vector $\mathbf{x}_i \in \mathbb{R}^F$ and a class-label $y_i \in \mathcal{Y}$, where F is the number of input dimensions and $\mathcal{Y}$ is the set of possible labels. The task of a learning algorithm is to induce a rule $H(\mathbf{x})$ for predicting the label of unseen instances, such that the probability of predicting the wrong label is minimized:

$$\hat{y} \leftarrow H(\mathbf{x}) \text{ so that } \epsilon = E\left[\mathrm{I}_{(y \neq \hat{y})}\right] \text{ is minimized } . \tag{2.1}$$

If not stated otherwise, formulations for binary classification are presented. Classification problems involving more than two classes are discussed in Chap. 3.

Two main principles for building classifiers may be followed. Generative models estimate class probabilities in whole feature space. Discriminative models focus on approximation of decision surfaces. Most of this work will focus on discriminative models only, due to their stronger performance for classification tasks.

Generative Models. Generative Models perform regression in order to estimate the distribution $p(y = k, \mathbf{x})$ for all classes $k = 1, \ldots, C$. Estimation can be performed, for example, by fitting a Gaussian mixture model to the data using the expectation maximization algorithm. The posterior probabilities can be

induced using Bayesian inference:

$$p(y = k|\mathbf{x}) = \frac{p(y = k)p(y = k, \mathbf{x})}{p(\mathbf{x})} \quad . \tag{2.2}$$

The decision of the classifier is then

$$\hat{y} \leftarrow \arg\max_k \left(p(y = k|\mathbf{x})\right) \quad . \tag{2.3}$$

The problem of using generative models for classification, is that class probabilities are estimated for the whole feature space. However, in order to build a good classifier, accurate decision boundaries are essential. Generative approaches don't especially account for decision surfaces. Therefore good global approximation might be achieved at the cost of increasing error rate. Consequently, benchmark experiments show that generative models are regularly outperformed by discriminative models, in the field of classification. On the other hand generative models are very flexible. For example, class-dependent misclassification costs can easily be incorporated without having to retrain the model - as it would be necessary for discriminative models.

Discriminative Models. Discriminative models don't attempt to model class probabilities globally. Instead they focus on modelling decision surfaces. Since there are less free parameters needed to fit only decision surfaces, the resulting models are simpler and can be fitted more easily. This leads, in general, to improved results for classification tasks. On the other hand, the trained models are more restricted than models produced by generative approaches. For example, if the relative misclassification costs of classes change, a new model has to be trained. Also, models are hard to interpret.

2.1.1. Assessing Performance

Generalization error of a classifier $H(\mathbf{x})$ is defined as:

$$\epsilon = E[\mathrm{I}_{(H(\mathbf{x}_j) \neq y_j)}] \quad , \tag{2.4}$$

where the samples $\mathbf{x}_j, \mathbf{y}_j$ are drawn from the generating distribution independently of the training samples. The generalization error ϵ can't be optimized directly. Therefore the performance of the classification rule has to be assessed using the available data, namely the training samples. The concepts needed for this are discussed briefly in this section.

Cost Functions

Numeruous cost functions for assessing results of classification and regression exist. The target of model fitting, for both settings, is to minimize the error on unseen data.

In classification settings, the objective is to minimize the generalization error. Since the generalization error is not accessible, the empirical error is often used as an approximation[1] of the generalization error that can be calculated on the training samples:

$$\hat{\epsilon} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{I}_{(H(\mathbf{x}_i) \neq y_i)} \quad . \tag{2.5}$$

In cases, where the set of training samples is small or the classifier is very flexible, the empiric error may underestimate the generalization error dramatically. Also, optimizing the empirical error directly is hard, since it is not differentiable. For these reasons, other - more practical - cost functions are incorporated to solve classification problems. These cost functions give an approximation of the empirical error, but are differentiable. Also some cost functions allow for a more realistic assessment of generalization than empirical error. The specific choice of cost function depends on the classification scheme used.

In regression settings, an unknown function $f(\mathbf{x})$ needs to be estimated using a model $h(.)$. For most problems, measurements y is assumed to be corrupted by gaussian noise:

$$y = f(\mathbf{x}) + \varepsilon \quad . \tag{2.6}$$

The task is to estimate $f(.)$ as tightly as possible, without adapting to noise. In case of gaussian noise, the most-likelihood estimation can be found by minimizing the squared error of the fit:

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} (h(\mathbf{x}) - y)^2 \quad . \tag{2.7}$$

In practice, using squared error is the standard choice even if the actual distribution of errors is not known. There are, however, cases where squared error is not the most appropriate choice, e.g. when errors are known to have another distribution than gaussian. Another problem of the squared error is it's high sensibility to outliers. So, if outliers may be an issue, other cost functions may be more appropriate.

[1] More general, class-dependent misclassification costs may be used. Using misclassification costs enables prioritization of the classes. If costs are equal for all samples, this is equivalent to minimizing the empirical error.

Table 2.1 shows a number of cost functions used for regression and classification problems. The classification costs are tailored to binary problems with $y \in \{-1, 1\}$. Figure 2.1 depicts some of the cost functions. While cost functions used with regression problems are symmetrical, cost functions for classification are not. This reflects that for classification the qualitative result is important. Therefore, the output can't be *too* high, as long as the sign is correct. Note, that all cost functions introduced in this section assess performance on training data only.

name	equation	optimal output
quadratic error	$\epsilon = \sum_i c_i \left(\hat{y} - y_i\right)^2$	$E_c[y\|\mathbf{x}]$
linear error	$\epsilon = \sum_i c_i \lvert\hat{y} - y_i\rvert$	solution of constrained linear optimization problem
linear classification costs	$\epsilon = \sum_i c_i \max((1 - \hat{y}y_i), 0)$	$a \operatorname{sgn}\left(E_c[y_i\|\mathbf{x}]\right)$
exponential costs	$\epsilon = \sum_i e^{-\hat{y}y_i}$	$\frac{1}{2} \log \frac{E_c[\mathrm{I}_{(y_i=1)}]}{E_c[\mathrm{I}_{(y_i=-1)}]}$
empirical error	$\epsilon = \sum_i c_i (\mathrm{I}_{(y_i \neq \hat{y}_i)})$	$\operatorname{sgn}\left(E_c[y_i\|\mathbf{x}]\right)$

Table 2.1.: Cost functions for regression (upper) and binary classification ($y \in \pm 1$). The constant a for linear classification error depends on regularization and is equal to ∞ without regularization.

Adjusting Generalization

Finding a good trade-off between quality of model fit on training data, assessed by the cost functions introduced above, and complexity of the model is essential for finding models with good generalization capabilities. A good example, showing that performance on training data is not sufficient for assessing performance of a classifier is the 1-Nearest-Neighbor classifier. It will always represent training data perfectly, resulting in perfect training error. However, Voronoi regions are generated for each training sample assigning the respective class in it's neighborhood, even if the posterior probability of that class is very low at that specific location in feature space. Therefore no good generalization can be expected for arbitrary problems[2].

In order to asses the generalization capabilities of a model, the performance on the training data, as well as the flexibility of the model needs to be taken into account. Examples of general measures for model flexibility are VC-dimension

[2] An example of a problem, that leads to poor performance for 1-Nearest-Neighbor classifiers is a superposition of two classes with uniform distribution over the same region of feature-space, where one class has higher a-priori probability. A Bayesian classifier should always vote for the class with higher probability. A One-Nearest-Neighbor classifier will generate voronoi regions for each sample of the "weaker" class - thereby deteriorating performance.

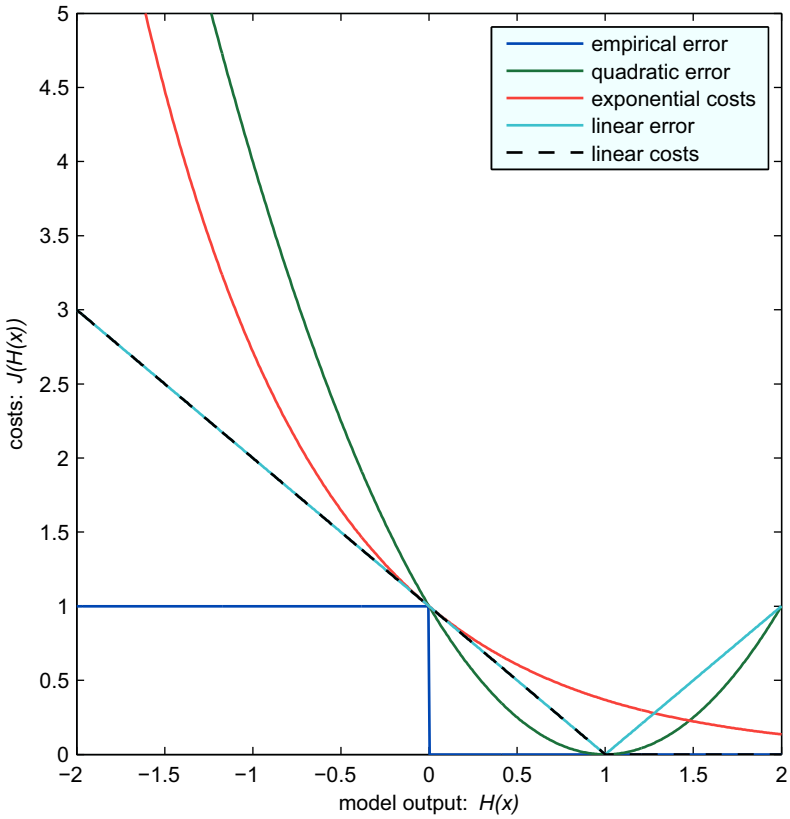

Figure 2.1.: Cost functions for classification and regression settings. Target output is $y_i = 1$.

Name	Definition
Akaike Information Criterion (AIC)	$AIC = 2d + N[\ln(RSS/N)]$
Bayessian Information Criterion (BIC)	$BIC = d\ln(N) + N[\ln(RSS/N)]$
Generalized Cross Validation (GCV)	$GCV = \frac{RSS}{N-d}$

Table 2.2.: Information criteria for assessing model performance. $RSS = \sum \epsilon_i^2$ is the residual sum of squares, N is the number of samples and d are the effective degrees of freedom, approximated in an appropriate way. For all given formulas, unknown and equally distributed errors are assumed.

[Vapnik and Chervonenkis, 1971] for classifiers and effective number of degrees of freedom for linear models in regression settings. The principle of finding a trade off between quality of model fit and model complexity is coined Occam's Razor. It states, that when choosing from models with equal training error, the simplest model should be selected. The practical application, however, is more complicated. For example when comparing two models with approximately equal training errors and complexities, it is not straightforward how to decide which model to prefer. For regression problems using linear models, the information criteria shown in Tab. 2.2 can be used to find a decision. Unfortunately, no equivalent criteria exist for the cost functions typically used with classification. For many classification approaches theoretical bounds for generalization errors exist. However, most of these are so pessimistic, that they largely overestimate generalization error and are only of limited practical use.

An approach which leads to very good practical results is cross-validation (see Appendix B). This approach repeatedly excludes samples from the training set and uses them to estimate generalization error of the trained classifiers. While this approach is very general and quite accurate, it leads to a significant increase in computational load, since classifiers need to be trained multiple times (typically 5 to 10). Nevertheless, cross validation is a very powerful tool and will be used in the experiments presented in this work.

Bias-vs-Variance

The terms bias and variance are related to the flexibility and robustness of a classifier. They can be used to asses the complexity of models built by a given classifier.

A model's bias measures, how well it can adapt to a given problem. Higher bias corresponds to less flexibility. For examples models based on linear decision thresholds in feature space: $\hat{y} = \mathbf{w}^\mathsf{T}\mathbf{x} + b$ ($\mathbf{w}$ is the normal of the decision

surface and b an offset) are biased for models that are not linearly separable. However, the optimal model can have a certain bias, preventing it from overfitting the data.

The variance of a model measures how it reacts to small changes of the training data. Since these small changes should not alter the overall statistical distribution of the data, the model should remain stable. If the model changes significantly, it can't be trusted.

A good model should exhibit small variance and small bias. These goals are conflicting, since reducing bias increases variance and vice versa. Quality of a model will depend on how good the tradeoff is.

2.2. Support Vector Machines

Support Vector Machines (SVM, Boser *et al.* [1992]) are used to construct binary classifiers. In derivation of SVM's the samples are viewed as points in an F-dimensional feature space. The points shall then be separated using a hyperplane which leads to the following model:

$$H(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\mathsf{T}\mathbf{x} + b\right) \tag{2.8}$$

If the points are linearly separable, in general a set of feasible separating hyperplanes exist. The selection problem is solved by choosing the hyperplane giving the largest margin. Vapnik [1982] shows that this, under very mild assumptions on the underlying data distribution, minimizes the expected generalization error. This means that the hyperplane is optimized to maximize the minimal distance of points from the hyperplane. For the problem to be feasible, perfect separation of the samples using a hyperplane has to be possible. That means a hyperplane exists such that all sample of the first class lie on one side of the hyperplane and all samples of the second class on the other. The optimization can then be expressed as a quadratic program:

$$\begin{aligned} &\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \\ &\quad \text{s.t. } \mathbf{w}^\mathsf{T}\mathbf{x}_i \geq 1 \;\forall i \;\;, \end{aligned} \tag{2.9}$$

which can be solved efficiently. The solution of the quadratic program is guaranteed to be globally optimal.

For general classification problems the classes will not be linearly separable. Also, using Eq. (2.9), the classifier might concentrate too much on outliers, leading to overfitting. This problems led to the design of SVM's with soft margins [Cortes and Vapnik, 1995]. With this extension the samples are no longer required to be linearly separable. Instead costs are assigned to samples which are

classified wrong. The costs depend on the distance of the samples from the decision boundary. These weighted costs are then added to the SVM optimization problem. The weight of the misclassification costs with respect to the margin width is adjusted using the slack penalty C. Therefore the classifier implements a trade-off between achieved margin and costs produced by samples which are on the wrong side of the decision boundary. This leads to the primal objective of soft margin SVMs:

$$\begin{aligned} \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t. } \mathbf{w}^\mathsf{T}\mathbf{x}_i \geq 1-\xi_i \text{ and } \xi_i \geq 0 \; \forall i; \end{aligned} \tag{2.10}$$

Using the Lagrangian polynomial, this can be transfered into the dual objective of soft margin Support Vector Machines:

$$\begin{aligned} \max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j \\ \text{s.t. } \sum_i \alpha_i y_i = 0 \quad \text{and } 0 \leq \alpha_i \leq C \; \forall i \; . \end{aligned} \tag{2.11}$$

If the Lagrangian multiplier α_i of an sample is zero, it has no influence on classification at all. This illustrates, how SVMs find the minimal set of samples defining the hyperplane that minimizes the given cost function. All other samples could be removed from the training set without changing the result. Often the samples irrelevant for construction of the decision surface will make up a significant part of training samples.

For many classification tasks, separation in input space is not possible (e.g. XOR problem). By using the so called kernel trick, it is possible to build nonlinear classifier using the SVM framework. Equation (2.11) shows that classification results depend on the inner product of samples only. Using this finding, samples can be transformed into a virtual feature space $\Phi(\mathbf{x})$[3]. As long as it is possible to calculate the inner product of two samples in the virtual space efficiently, the dimensionality of the space is irrelevant. This has the advantage, that the feature space doesn't need to be defined explicitly. The SVM optimization will produce a linear hyperplane in the virtual feature space. Transforming back into the input space leads to nonlinear decision surfaces. Thus, by using kernels SVMs can be applied to a much wider range of problems.

While theoretical bounds for generalization errors of SVMs exist, these are in general too loose to be used in for parameter selection. Therefore, in practice,

[3]By Mercer's theorem, $\Phi(\mathbf{x})$ needs to be a symmetric, continuous, positive semi-definite kernel function (Mercer Kernel).

the optimal parameters (slack penalty C and kernel parameters) are adjusted using cross validation. Advantages of SVMs include its sound theoretical foundation, as well as the use of quadratic programming to solve the optimization problem. SVMs define state-of-the art performance for a wide range of classification problems. Therefore they are an important benchmark for newly proposed classifiers. Disadvantages include expensive training and evaluation for training problems involving many training samples.

2.3. Artificial Neural Networks

Artificial Neural Networks (ANN, e.g. Rosenblatt [1958]) are motivated by two very different approaches. The first views ANNs as simple models of the function of biological neural networks - this more classical view was the original motivation for designing ANNs. The second line [e.g. Hornik *et al.*, 1989] views ANNs as general function approximator and focusses on practical applicability instead of modelling the complex operation of biological neural networks. This work will not deal with biological interpretations of ANNs.

Neural networks are constructed from simple processing units (neurons) and the connections between these units. The processing units calculate the sum of their weighted inputs, which are either input features or outputs of preceding processing units, and apply a transfer function to the result. Despite the simple nature of the processing units, complex global behavior of the network is possible, depending on the connections of the network. Figure 2.2 shows a feed-forward neural network with one hidden layer. This work will only discuss simple neural networks with feed-forward architecture (no loops, delays or memory). Neural networks are trained by repeatedly presenting the training patterns and adjusting the connection weights in order to produce the desired output values.

Popularity of neural networks received a dent upon the introduction of Support Vector Machines. The reason for that is the firmer theoretical foundation of SVMs along with their better usability. While high numbers of design parameters need to be set for neural networks, good results are achieved by SVMs adjusting only two parameters (kernel properties and slack penalty). On the other hand ANNs still are benchmark for some machine learning problems and are more flexibly applicable than SVMs.

In order to model non-linear functions in feature space, neural networks with hidden layers are needed. The flexibility of a neural network depends on the number of neurons used and the number of hidden layers. For many problems, one hidden layer will be sufficient. For some cases using more than one hidden layer is advantageous. Neural Networks with more than two layers are not

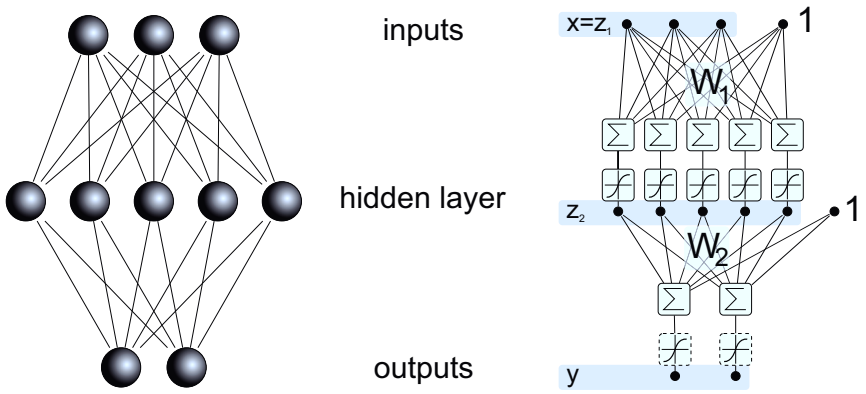

Figure 2.2.: Feed forward neural network with one hidden layer. Left: architecture, right: mathematical structure. Biases are realized using additional inputs units with constant input. In regression settings the output transfer functions will be removed.

used frequently for general problems. However - they are useful together with special network architectures tailored to solve certain classification tasks (e.g. the networks used for recognition of handwritten digits used in LeCun *et al.* [1990]).

The prediction of a feed forward network with $T-1$ hidden layers can be calculated in iterative manner as follows:

$$z_t = \sigma_t \left(\mathbf{W}_t^T \begin{bmatrix} \mathbf{z}_{t-1} \\ 1 \end{bmatrix} \right) \tag{2.12}$$

With the layer numbers $t = 1, \ldots, T$, inputs $\mathbf{z}_0 = \mathbf{x}$, outputs $y = \mathbf{z}_T$ and transfer function of the t-th layer σ_t. The Matrices $\mathbf{W}_t$ represent the connection weights which are tuned during training.

To train neural networks efficiently, backpropagation training is used [Rumelhart *et al.*, 1986]. This technique allows to calculate desired output values of hidden units[4]. With error backpropagation training of multi-layer networks using gradient descent algorithms becomes feasible. Backpropagation training works in two steps. In the first step (forward propagation) the network is presented with the inputs and the respective outputs of all layers are calculated. In the second step (backward propagation) the outputs of the network are compared to the desired outputs to calculate the errors and gradients of the output layer. By using the chain rule, the errors and gradients of hidden layers can be derived

[4]Only the desired outputs of the output units are known directly.

from the results of the output layer. Details are shown in Alg. 1. In contrast to SVMs, in general only a local minimum of the network cost function is found.

A number of parameters need to be tuned to adjust the flexibility of ANNs to the problem at hand: number of hidden neurons, number of hidden layers, step width and associated parameters for gradient descent algorithm, number of training epochs, et cetera. This high number of tunable parameters is a disadvantage of neural networks, since it is hard to find an optimal setting. Other techniques as Support Vector Machines and Boosting typically have a lower number of tunable parameters and their performance is less sensitive to the parameter settings.

Algorithm 1: Training of a neural network classifier using error backpropagation

Input: Training samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
Network architecture (number of layers T, number of hidden units in each layer)

Output: Neural network with trained weights

Initialize the weight matrices $\mathbf{W}_t,\ t = 1, \ldots, T$ with small random weights;
Set $\mathbf{O}_0 = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$;
repeat
  for $t = 1, \ldots, T$ **do**
    Calculate outputs of current layer $\mathbf{O}_t = \mathbf{O}_{t-1}\mathbf{W}_t$;
  end
  Calculate errors $\boldsymbol{\delta}_T = \mathbf{y} - \mathbf{o}_T$ of output neurons;
  for $t \leftarrow T - 1$ *downto* 1 **do**
    Error backprop: $\boldsymbol{\delta}_t = \boldsymbol{\Sigma_t}'\mathbf{W}_t\boldsymbol{\delta}_{t+1}$;
    Update weights: $\Delta\mathbf{W}_t = \mathbf{W}_t - \gamma\boldsymbol{\delta_t}\mathbf{o_{t-1}}$;
  end
until *until stopping criterion is satisfied*;

2.4. Partition Based Classifiers

Partition Based Classifiers divide the feature space into a number of regions. Examples are decision trees, histograms and smoothing splines. Each region in feature space is assigned a constant output value (trees, histograms) or a simple local function (splines). Advantages of partition-based classifiers include simple implementation and cheap evaluation. The outputs are tuned to optimize a

given cost criterion. In the case of constant outputs, the output of each region can be optimized independently from other regions[5].

This work only deals with classifiers using explicit partitioning. Classifiers using implicit partitioning, e.g. by using neighborhood relationships as used by k-Nearest-Neighbors are not considered. Also, only classifier dividing the feature space using hyperplanes are discussed. The direction of the hyperplanes are described by linear projections of the inputs: $h(\mathbf{x}) = f(\mathbf{w}^\mathsf{T}\mathbf{x})$. Linear features are the most common choice used with partition based classifiers. However, other choices are possible as well.

2.4.1. Decision Stumps

Decision stumps split the feature space into two half-spaces. Each half-space is then assigned an output value. The model of a decision stump is:

$$h(\mathbf{x}) = a + b\,\mathrm{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x} + s) \quad , \tag{2.13}$$

where a and b are constants, that are optimize in order to minimize the given cost function. The possible outputs of the model are therefore $a - b$, $a + b$ and a. The projection $\mathbf{w}$ and offset s describe the decision surface. An important special case are projections vectors $\mathbf{w}$, where all but one element are zero - therefore one input component is selected (component-wise projection). For a given decision threshold the values of a and b can be calculated directly. Evaluation of decision stumps can be performed in constant time $O(1)$.

Often the projection $\mathbf{w}$ is given and only the threshold s needs to be determined in order to fit the stump to the data. This can be done very efficiently by first sorting the samples along the projection direction. Then the best threshold can be found in linear time. So the overall costs have complexity $O(N \log(N))$.

The advantages of decision stumps are their cheap training (for given $\mathbf{w}$) and evaluation. Also, decision stumps can be used to build decision trees, extending their applicability. Regularization of decision stumps can be done by defining a minimal number of samples allowed on either side of the decision surface. A more powerful approach is regulating the output values defined by a and b.

2.4.2. Histograms

Histograms divide the feature space along a linear projection $\mathbf{w}$ by placing a number of thresholds. Each region, limited by two thresholds, is assigned a

[5]This is not true, if regularization is used to smooth the outputs (see discussion in Sect. 2.4.3).

constant output value. This leads to the following model of a histogram with T regions

$$h(\mathbf{x}) = \sum_{t=1}^{T} a_t \mathrm{I}_{\left(s_{t-1}<\mathbf{w}^\mathsf{T}\mathbf{x}<s_t\right)} \qquad \text{with } s_0 = -\infty, s_T = \infty \ . \tag{2.14}$$

Similar to stumps, histograms often work on one input feature only. Output values a_t can be calculated directly as soon as the samples are assigned to the output regions.

The thresholds $s_1, \ldots, s_{K-1}$ could be placed in order to minimize costs. However, finding optimal thresholds is impractical since the number of possible partitions grows exponential with the number of regions. So often the thresholds are placed equidistant in feature space. This also has the advantage that evaluation can be performed using table lookups. Also, training using equidistant thresholds is cheap. The effort is dominated by sorting the samples leading to training complexity $O(N \log(N))$.

If the problem can't be modelled along one dimension efficiently, intersections of one-dimensional histograms can be used. However, this approach is limited to low numbers of dimensions. For higher numbers of dimensions the average number of samples in each partition approaches zeros fast.

Histograms can be trained and evaluated efficiently. Often, more than one decision stump is needed to achieve the same performance as one histogram. Therefore histograms may be the better classifier choice in cases where feature calculation dominates the classifier costs.

2.4.3. Univariate Smoothing Splines

Instead of using constant outputs in each region - like histograms, smoothing splines use a simple function to approximate the output function. This has the advantage, that often less regions are needed to reach the same approximation error. In addition, continuous outputs are expected to be a more realistic model of real world class distributions.

Use of smoothing spline base classifiers for training of projective features is discussed in Sect. 4. While smoothing splines can also be used to fit distributions in low dimensional feature spaces, this work deals with univariate smoothing splines only.

Smoothing splines express functions as a weighted sum of spline base functions:

$$h(z) = \mathbf{a}^\mathsf{T}\mathbf{b}(z) \qquad \text{with } z = \mathbf{w}^\mathsf{T}\mathbf{x} \ , \tag{2.15}$$

where z is a scalar input value and $\mathbf{b}(z)$ return the vector containing the values of the base functions evaluated at z. To construct a fit from scalar inputs z_i to

outputs y_i, the weights $\mathbf{a}$ need to be calculated by solving a linear system of equations. In order to prevent overfitting, a tradeoff between approximation error and complexity has to be found. P-Splines from Eilers and Marx [1996] are used for fitting penalized splines: a fixed high number of equidistant support points is used and a parameter λ is tuned to adjust the amount of smoothing. P-Splines use finite differences of the weights $\mathbf{a}$ of the spline functions to approximate roughness. The weights $\mathbf{a}$ can then be calculated using

$$\mathbf{a} = \left(\mathbf{B}\Delta_\mathbf{c}\mathbf{B}^\mathsf{T} + \lambda\mathbf{D}\mathbf{D}^\mathsf{T}\right)^{-1}\mathbf{B}\Delta_c\mathbf{y} \ , \tag{2.16}$$

where $\mathbf{B} = [\mathbf{b}(z_1)\dots\mathbf{b}(z_N)]^\mathsf{T}$ denotes the matrix of values of the spline basisfunctions evaluated at $z_1,\dots,z_N$, $\mathbf{y} = [y_1\dots y_N]^\mathsf{T}$ contains the sample class and $\Delta_c \in \mathbb{R}^{N\times N}$ is a diagonal matrix containing the sample weights $c_1,\dots,c_N$. The expression $\mathbf{a}^\mathsf{T}\mathbf{D}$ calculates finite differences of a given degree on $\mathbf{a}$. The roughness penalty can be chosen using cross validation. Note that the size of the linear system of equations in (2.16) depends on the number of spline base functions only.

The importance of regularization is visualized by Fig. 2.3. Without regularization the spline approximation overshoots the true function and shows ringing effects, which is clearly undesirable in a classification setting. Penalization of first and second order finite differences is shown. First order penalties lead to a more conservative approximation of the true function, tending towards a constant output for high roughness penalties: $h(\mathbf{x}) = \mathbf{c}^\mathsf{T}\mathbf{y} = const$ for $\lambda \to \infty$. Second order penalties don't prevent overshooting. Therefore first order finite differences are the more appropriate choice for classification settings.

2.4.4. Decision Trees

For classification problems, where the separation of the classes along the feature directions is poor, the partition-based classifiers discussed above will not yield satisfactory results. Since they work in a component-wise fashion, they return very weak hypotheses. In these cases, decision trees provide a simple scheme to build hierarchical classifiers. They work by combining classifiers into a tree structure. In most cases decision stumps are used to build the nodes of the tree. Since attempting to build optimal trees is not practical, greedy approaches are used, growing the tree starting with the root and sequentially adding the nodes giving best cost improvement. Well-known approaches for constructing trees are ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1996]. In the experiments of this work a very simple tree growing algorithm[6] is used, as shown in Alg. 2.

[6] In particular no pruning is used.

Algorithm 2: Decision tree growing algorithm. The node learning algorithm will return a decision rule minimizing the given cost function. Note, that this decision tree algorithm is very simple, but sufficient to achieve good performance in cooperation with boosting schemes.

Input: samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
Node learner
Cost function
Stopping criterion
Output: decision tree

```
Create node with C outputs, where C is the number of possible outcomes;
Construct decision rule h(x^(f)) by calling node learner on all input features;
Select decision rule h(x) = h(x^(f*)) of the feature f* giving minimal costs
  and assign it to current node;
Split samples into sets depending on the outcome of h(x);
for all possible outcomes do
    Evaluate cost function on associated set;
    if stopping condition is not satisfied then
        Call tree growing algorithm with samples in set;
        Replace output by returned subtree;
    end
end
```

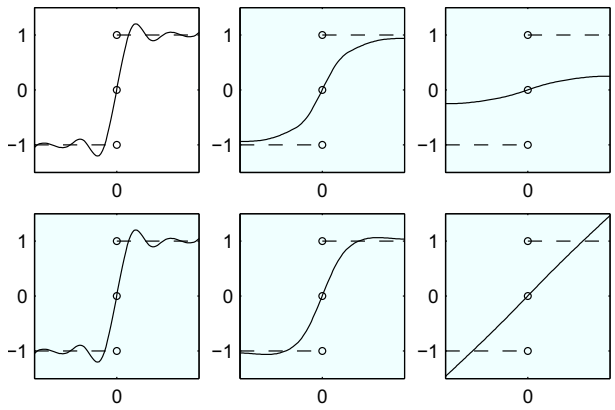

Figure 2.3.: spline approximations of the sign function. First and second derivative penalty in the respective rows. From left to right: no, medium and high roughness penalty. The spline fits in each column have the same squared fitting error

A tree defines a hierarchical partitioning of feature space. Even with very simple node-function (e.g. decision stumps), arbitrary decision boundaries can be approximated. However, using too weak node functions[7] will lead to overly complex decision rules and hurt generalization.

There are different approaches for adjusting model complexity of decision trees. One possibility is limiting the number of nodes used to build the tree. This limits the number of output regions and therefore increases the average number of samples in each region. However, limiting the number of nodes will not prevent the tree from generating regions with very few samples. Another important factor for the generalization performance of trees is its depth. Higher depths lead to poorer Generalization [Reyzin and Schapire, 2006]. This also means that for a given number of nodes, more balanced trees will likely be less susceptible to overfitting.

An important technique for building trees with better performance is pruning. First a tree is built that perfectly separates all training samples – allowing overfitting to the training data. In the second step, starting from the leaves, nodes are removed from the tree. A node will only be left in the tree if it gives a minimum improvement of a specific quality criterion. This corresponds to Occam's Razor – only nodes, that achieve a certain performance improvement with respect to the increase in complexity they cause, are left in the tree.

[7] If the separation of classes is poor along the available projections $z = \mathbf{w}^\mathsf{T}\mathbf{x}$, it may still be possible to separate the classes perfectly. But it is likely that node decisions represent random perturbations of training data rather than the true class distribution. Therefore the generalization performance is expected to be poor for these settings.

2.5. Boosting Algorithms

Boosting is a technique to construct linear ensembles of weak "base classifiers". Base classifiers use simple models and give relatively poor individual performance. By combining a number of base classifiers, ensembles with arbitrary low training error can be built. The first practical boosting algorithm was introduced in Freund and Schapire [1995]. Since then, numerous new Boosting flavors have been proposed.

Boosting is the central classifier model, dealt with in this work. It is an example of a family of classifier models that gained prominence in recent years[8]. All of these algorithms have in common that they build a strong classifier by combining a number of weak basic classifiers into a linear ensemble. The outputs of the classifiers are combined in a voting scheme:

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \quad . \tag{2.17}$$

In the case of boosting, the training problem is relaxed so that the performance requirements for each base classifier is low. The relaxed problems can then be solved using very simple base classifier models. This simplicity in turn leads to a small number of adjustable parameters and therefore good applicability.

Boosting achieves good results in many empirical studies. In comparison to SVM classifiers, the theoretical foundation of Boosting's good performance is not so well understood. Consequentially, the theoretical bounds for generalization performance of boosting classifiers are very loose. The lack of theoretical foundations is also responsible for the high number of available boosting flavors. Since the performance differences of the individual boosting flavors is, in general, negligible - only the most popular boosting flavors are discussed in this work.

Technically, Boosting algorithms work by holding a distribution of sample weights $c_i,\ i = 1, \ldots, N$. Before start of training all samples have equal weights. In each boosting stage a base classifier is fitted, taking the weights c_i into account. Through the course of training, the weights are adjusted in order to put more weight on samples that are classified wrong by the ensemble of already trained base classifiers. After each round of boosting the weights of training samples are increased or decreased, depending on whether they were correctly classified by the new base classifier. In consequence, new base classifiers will spend more effort on getting samples right that were misclassified by preceding stages.

[8] Another example: Bagging is discussed in Sect. 2.6.

2.5.1. PAC Learning Theory

PAC stands for Probably Approximately Correct and provides a framework for analysis of machine learning problems and was proposed in Valiant [1984]. The PAC theory defines conditions under which a classification problem is considered to be learnable. Learnability of a problem is defined as the existence of an algorithm which is able to find a solution with certain properties in polynomial time.

A strong learner has to be able to build a model that exceeds an error rate ϵ with probability of less than δ:

$$P\left[P\left[H(\mathbf{x}_i) \neq y_i\right] > \epsilon\right] < \delta \quad . \tag{2.18}$$

The training complexity in memory space and time shall not be more expensive than polynomial in $1/\epsilon$, $1/\delta$ and N. PAC theory also defines the notion of a weak learner[9]. A weak learner only needs to achieve error rates ϵ slightly better than random guessing.

Boosting is a concrete rule for building a strong model satisfying Eq. 2.18 in compliance with the requirements of PAC learning theory. In Schapire [1990] it is shown that it is possible to construct arbitrary strong hypothesis from weak hypothesis, thus laying the foundation for boosting algorithms.

2.5.2. Gradient Boosting

The first practically applicable boosting algorithm AdaBoost was proposed in Freund and Schapire [1995]. AdaBoost, as well as a number of derived boosting flavors are based on using a gradient descent on a given cost function [Friedman *et al.*, 1998]. A gradient step on a misclassification-cost function is performed in each round of boosting. For this, in each boosting stage, the cost function is evaluated for the results of the preceding classifiers. From the results the gradient descent direction of the next boosting round is derived. The next base classifier is then constructed to perform the gradient step. Alg. 3 shows the full course of boosting training in a general formulation.

Though a large number of boosting flavors exists, here only the most frequently used are discussed. All algorithms work as shown in Alg. 3. The specifics of the different flavors are shown in Tab. 2.3. In the experiments presented in this work only GentleBoost is used. GentleBoost is very simple and at the same time shows good resistance to overfitting [Friedman *et al.*, 1998][10].

[9] The notions weak learner and base classifier are used interchangeable in this work.

[10] Note that AdaBoost and RealBoost are corrective Boosting Algorithms, this means that the costs of the preceding base classifier with the updated weights are the same as for random

Algorithm 3: Boosting - general boosting algorithm performing a gradient descent on exponential cost function (binary problems)

Input: Training samples $\mathbf{x}_i, y_i,\ i = 1, \ldots, N$
Number of boosting rounds T
Base learner
Sample reweighting function $f()$
Output: Classifier: $H(\mathbf{x}) = \sum_{t=0}^{T} \alpha_t h_t(\mathbf{x})$

Set $h_0(\mathbf{x}) = \overline{y}$;
Calculate α_0;
for $t = 1, \ldots, T$ **do**
    Calculate current hypotheses: $\hat{y}_i = \sum_{s=0}^{t-1} \alpha_s h_s(\mathbf{x}_i)$;
    Calculate sample weights using current hypotheses: $c_i = \exp(-y_i \hat{y}_i)$;
    Normalize sample weights so that $\sum_i c_i = 1$;
    Run base learner on weighted problem $\{\mathbf{x}_i, y_i, c_i\}$ to generate new base classifier $h_t(\mathbf{x})$;
    Calculate base classifier weight α_t ;
end

Name	α_t	optimal output h^*
AdaBoost	$\alpha_t = \log\left((1 - e_t)/e_t\right)$ with $e_t = E_c[\mathrm{I}_{(y \neq h_t(\mathbf{x}))}]$	$\mathrm{sgn}(E_c[y]\|\mathbf{x})$
Real AdaBoost	1	$\frac{1}{2}\log(\frac{E_c[y=1\|\mathbf{x}]}{E_c[y=-1\|\mathbf{x}]})$
GentleBoost	1	$E_c[y\|\mathbf{x}]$

Table 2.3.: Comparison of gradient-based boosting algorithms.

The base classifiers are controlled using a weight distribution c_i on the training samples. For GentleBoost the weights are calculated as:

$$c_{i,t} \leftarrow exp\left(-y_i \sum_{s=1}^{t-1} \alpha_s h_s(\mathbf{x}_i)\right) \qquad \text{with } \alpha_s = 1 \ . \tag{2.19}$$

The weights make sure new base classifiers focus on samples, classified incorrect by the ensemble of preceding classifiers. Samples that are classified correctly by the preceding classifiers will receive lower weights and therefore less attention of the base classifiers. For GentleBoost the base classifiers minimize the weighted squared error of the hypothesis:

$$h_{t+1} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{N} c_i \left(h(\mathbf{x}) - y_i\right)^2 \ . \tag{2.20}$$

This is equivalent to performing Gauss-Newton steps in function space on the cost function J [Friedman *et al.*, 1998], which is defined by

$$J_t = \sum_{i=1}^{N} c_{i,t} \ . \tag{2.21}$$

In order to minimize the classifiers costs, the weights of the samples have to be minimized. It can be shown, that the costs J on the training samples asymptotically approach zero, as long as all base classifiers do better than random guessing. Achieving performance slightly better than random guessing is simple. However, the slight advantage over random guessing might be modelling random perturbations of training data and not represent the true distribution of the classification problem. Therefore some mechanism to control significance of the hypothesis should be used - for example by applying a threshold on the base classifier costs.

2.5.3. Linear Programming Boosting

Another line of boosting schemes doesn't use gradient descent, but instead solves a linear program to construct an ensemble. Examples are LPBoost [Demiriz *et al.*, 2002] and SoftBoost [Warmuth *et al.*, 2008]. The routines for calculating the sample weights can be derived from the linear program defining the boosting scheme. The weights are calculated, such that every new base classifier is

guessing. GentleBoost doesn't have this property—this, however, doesn't result in poorer performance in empirical studies.

independent from all preceding base classifiers (total corrective property). Linear programming boosting scheme's have a close connection to SVM [Warmuth *et al.*, 2008].

One advantage of linear programming boosting is faster convergence compared to gradient based approaches. This can result in a reduced number of base classifiers needed to solve a problem and therefore cheaper evaluation of the boosted classifier. In addition the schemes have measures to regularize the sample weights, which is expected to improve generalization. A disadvantage is the increased number of adjustable parameters of the boosting scheme itself[11]. Another disadvantage is the costly calculation of sample weights, that can easily be more expensive than base classifier training itself.

This work doesn't examine linear programming boosting schemes in more detail, due to the limitations mentioned above. Nevertheless, it is emphasized that utilization of linear programming boosting together with the feature generation schemes presented in Sect. 4 seems to be a promising path. It allows to reach equivalent result with less base classifiers, which is interesting especially in fields where evaluation costs of base classifiers are critical.

2.5.4. Regularization

In contrast to former belief Friedman *et al.* [1998], overfitting may be a problem for Boosting algorithms (see discussion in Buehlmann and Hothorn [2008]). Therefore measures should be taken to prevent overfitting. One approach is to restrict the amount of weight put on samples, preventing Boosting from focusing too much on outliers. This is implemented for example in the linear programming boosting algorithms, discussed in the preceding section. Another way to prevent overfitting is limiting the number of boosting rounds.

Yet another technique to prevent overfitting is to adapt the flexibility of the base classifiers to the problem at hand. If base classifiers are too weak, convergence may be slow. On the other hand, if base classifiers are too flexible, they will overfit instantly. Since Boosting is much more a bias-reducing, than a variance-reducing technique, it will have difficulties to cope with overly flexible base classifiers[12].

For base classifiers working by partitioning feature space (Sect. 2.4), flexibility can be adjusted by setting the number of regions. A disadvantage is that

[11] For Boosting schemes based on gradient descent, in general the only parameter to be optimized is the number of boosting rounds. For many classification problems, Boosting will not overfit and therefore performance will saturate after a minimum number of rounds has been reached. In contrast, additional parameters need to be set for LPBoost and SoftBoost .

[12] Other ensemble construction techniques, like e.g. Bagging (see below) are more appropriate for reducing variance.

the regularization has to be set in integer steps. Also, limiting the number of regions may not be sufficient for regularization, since overfitting may also occur if regions contain only few training samples. A more general applicable regularization strategy for arbitrary base classifiers works by penalizing roughness of the fitted functions. For base classifiers minimizing weighted quadratic fitting error this leads to

$$h \leftarrow \min_{h \in \mathcal{H}} \left(\sum_{i}^{N} c_i (y_i - h(\mathbf{x}_i))^2 + \lambda P(h(\mathbf{x})) \right) \quad , \tag{2.22}$$

were $P(.)$ measures the roughness, for example

$$P = \int_{z_{min}}^{z_{max}} \left(\frac{\partial h(z)}{\partial z} \right)^2 dz \tag{2.23}$$

with $h(.)$ being a univariate function of $z = f(\mathbf{x}) \in \mathbb{R}$. The amount of smoothing can be adjusted continuously by tuning the roughness penalty λ. Selection of λ can be done using cross-validation.

2.5.5. Boosting for Feature Selection

An important application of boosting schemes is feature selection. For feature selection, base classifiers working on one input feature only (component-wise base classifiers) are used: $h(\mathbf{x}) = h(z)$ with $z = \mathbf{x}^{(j)}$[13]. Typical examples of classifiers working in a component-wise fashion are decision stumps, histograms and smoothing splines.

Given a weighted training problem $\{\mathbf{x}_i, y_i, c_i\},\ i = 1, \ldots, N$ the base learner will fit F univariate base classifiers, one for each input. The base classifier with minimal costs is added to the Boosting ensemble. Since the sample weights ensure that base classifiers are independent and helpful for the classification problem, the selected features will exhibit the same properties. For problems with very high numbers of inputs features, performance often saturates long before all features were selected. Therefore - often - boosted classifiers work on a subset of the input features only. So, a part of the evaluation costs, associated with feature calculation, can be saved during evaluation of the classifier.

Input feature sets used for feature selection with Boosting have to contain a subset of features that models the classification problem well enough to be solved by the given base classifiers. Often, component-wise base classifiers like

[13] It is assumed, that a fixed set of input features is available. However, these may include features derived from the original inputs.

decision stumps are used. Component-wise base classifiers may fail for comparatively simple problems. For example, the well known XOR problem can't be solved using component-wise models. Note that decision tree base classifiers will also have difficulties with problems where class separation along the given features is poor, since they too add features one-at-a-time. On the other hand, boosting deals with redundant features very well. Therefore using extended feature sets might be beneficial. Techniques for learning expressive features are discussed in Chapt. 4.

A successful example of the use of boosting for feature selection is the use of Haar-Features for classification of face and non-face patterns in Viola and Jones [2001]. A relatively small number of features is drawn from a very large set of haar-like features ($F > 10^5$). This leads to very cost-efficient and at the same time accurate classifiers. In summary, boosting is one of the best and most simple schemes for feature selection. It deals well with large feature sets, containing many correlated features. Problems only occur, if features are not strong enough to build a component-based model.

2.6. Bagging

This section briefly introduces Bagging (Bootstrap Aggregating). Bagging - like Boosting - is also a technique for building linear ensembles of base classifiers. With Bagging base classifiers with high variance can be combined to build a classifier with low variance. Since the base classifiers are allowed to have high variance, no regularization of the base classifiers is necessary.

In contrast to Boosting, Bagging doesn't work by assigning sample weights in each round of training. Instead for each base classifier a selection of samples is drawn randomly from all training samples. Typically N samples are drawn with replacement. The probability to be drawn is equal for all samples. Each base classifier is then fitted to one random training set. For evaluation, all base classifiers vote equally. The detailed algorithm is shown in Alg. 4. By using random selections of samples for each base classifier, a certain degree of independence of the base classifiers is achieved. If the assumption of independence holds, using more base classifiers will result in a reduction of variance of the ensemble.

Since each base classifier uses only a subset of the training samples, the remaining samples can be used to assess their performance. This is referred to as out-of-bag error estimate. The resulting approximation of generalization is somehow pessimistic, but still works much better than theoretic error bounds. Qualitative comparison of out-of-bag error estimates for different parameter sets are very useful for parameter selection.

Advantages of bagging include the possibility to parallelize the training. Also, since no sample no sample weights are used, bagging can be used with virtually any out of the box classifier. Out-of-bag error estimates are helpful for tuning parameters of the base classifier, without having to use costly cross-validation. In addition bagging is not prune to overfitting. The main disadvantage of bagging is that its performance is often poorer than the performance of boosting classifiers or SVMs.

Algorithm 4: Bagging (binary problems)

Input: Training samples $\mathbf{x}_i, y_i,\ i = 1, \ldots, N$
Number of base classifiers to use T
Base learner

Output: Classifier: $H(\mathbf{x}) = \sum_{t=0}^{T} h_t(\mathbf{x})$

1 **for** $t = 1, \ldots, T$ **do**
2 Build random training set by drawing N samples from $1, \ldots, N$ with replacement;
3 Run base learner on random training set to generate new base classifier $h_t(\mathbf{x})$;
4 **end**

2.6.1. Random Forests

Random Forests [Breiman, 2001] are an extension of the bagging scheme. A "forest" of decision trees is constructed. Each tree is grown until it classifies all of his training samples correctly. In addition to randomly selecting the samples for each base classifier, the features used in each node of the decision tree are randomly selected. So in each node not all features are evaluated, but only a random subset. Typically $\lfloor \sqrt{F} \rfloor$ features are used, where F is the number of available features. Algorithm 5 shows the training of a decision tree as part of a random forest. Using decision trees with randomized features together with Bagging results in Random Forests.

By randomizing the use of features, the independence of the base classifiers increases. Randomized trees are an extreme case of a base classifier with no bias but very high variance. Each individual tree will exhibit poor generalization. However, combining a sufficiently high number of randomized trees using bagging, results in competitive performance on a wide range of machine learning problems.

The most important advantage of Random Forests is the absence of tunable parameters. In addition training of random forests is extremely efficient and can be parallelized easily. In Sect. 4.4 random forests are used as a benchmark algorithm.

Algorithm 5: Randomized tree growing algorithm used for Random Forests

Input: Samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
Node learner
Output: decision tree

```
Assign all samples to tree root;
while Any training sample is classified incorrectly do
    for Each node with misclassified training samples do
        Select random feature set F by drawing ⌊√F⌋ features randomly
        without replacement;
        Construct decision rule h(x^(f)) by calling node learner on the
        samples of the current node with all features in F;
        Select decision rule h(x) = h(x^(f*)) of the feature f* giving minimal
        costs and assign it to current node;
        Assign samples to the respective outputs of the current node,
        according to the output of h(x);
    end
end
```

3. Efficient Multi-Class Boosting

3.1. Motivation

Numerous applications in machine learning and image classification involve more than two output classes. A typical example from image processing is cathegorization of objects or scenes depicted on photos. Another typical example is classification of syllables for speech recognition problems. Multi-class problems become more and more important as new applications of machine learning emerge. Therefore approaches implementing efficient techniques for solving multi-class problems are an important field of ongoing research. In particular techniques scaling gracefully as number of samples and number of classes grow are sought.

Some of the current benchmark classification techniques (e.g. Boosting and SVMs) are not instantly applicable to multi-class problems. By design they are tailored to binary problems. This limitation may be seen as a drawback. On the other hand, powerful techniques for building multi-class classifiers by combining binary classifiers exist. Section 3.4 introduces these techniques briefly. Their application to multi-class Boosting is discussed in Sect. 3.5. A new approach for training base classifiers for multi-class boosting is introduced in Sect. 3.5.5. An experimental section comparing the performance of the discussed techniques concludes the chapter. The newly proposed scheme achieves good performance compared to state-of-the-art techniques.

3.2. Problem formulation

A multi-class training problem is given as a collection of N training samples. Each sample consists of a feature vector $\mathbf{x}_i \in \mathbb{R}^F$ and a Label $y_i \in \mathcal{Y}$. To improve readability, without loss of generality, it is assumed that for a training problem involving $|\mathcal{Y}| = C$ classes, the labels are $\mathcal{Y} = \{1, \ldots, C\}$.

The machine learning systems attempts to find a function $H(\mathbf{x})$ inducing the label y_i for a given feature vector $\mathbf{x}_i$. For binary problems, the usual approach is to train $H(\mathbf{x})$ so that sign $(H(\mathbf{x}))$ indicates the class of the hypothesis $\hat{y}_i$. For problems involving more than two classes, often a more complex design of the classifier is needed. Different routes may be taken to formulate a the problem

in way compatible to the utilized training scheme. Some examples of these approaches are introduced in the following sections. In general, a function $K(.)$ is used to map the output of the classifier onto the labels: $\hat{y} \leftarrow K(H(\mathbf{x}))$.

Solving multi-class problems is - in general - harder than solving binary classification problems. For example the classification error rate achieved by trivial classifiers[1] rises as entropy of the class labels increases. Constant classifiers $K(H(\mathbf{x})) = const = \max_k p(k)$ for $k \in \mathcal{Y}$ attain empirical error rates of

$$\epsilon_{\min} = 1 - \max_{k \in \{1,\ldots,C\}} \left(\frac{1}{N} \sum_{i=1}^{N} \mathrm{I}_{(y_i=k)} \right) \quad . \tag{3.1}$$

Random guessing without prior knowledge leads to empirical error rates of $1/C$. When using optimal Bayes classifiers, the absolute number of errors increases continuously as new classes are added to the problem. Furthermore, complexity of decision surfaces grows as number of classes increases. The number of hyperplanes needed to separate $C > 2$ classes grows as C increases.

3.3. Multi-class Capable Classifiers

Many classification techniques can deal with multi-class problems directly. A short introduction of frequently used examples is given here for a better understanding of the techniques introduced later in this chapter. In particular the techniques used to solve multi-class problems utilized with neural networks are closely related to the approaches of Sect. 3.5, where multi-class base classifiers for Boosting are discussed.

Decision Trees The feature-space is divided into a number of regions. Each region is assigned the label of the class most probable to be observed in this region. In order to prevent masking effects, the number of regions has to be chosen sufficiently high. This, on the other hand, increases the risk of overfitting. For each region only the most probable class needs to be stored. This gives a very compact representation. On the other hand, a lot of information is lost by discarding the information regarding frequency of all-but-one class.

Artificial Neural Networks One output neuron is used to indicate the activation for each class label. If the network is presented a sample of a certain class, the corresponding output neuron should take on a high value, while all remaining output neurons take low values. Depending on the specific

[1] Classifiers not using the feature vector $\mathbf{x}$ to compute their output (e.g. a-priori classifiers).

transfer and error functions, this approach is very similar to the 1-vs-all base classifiers discussed in Sect. 3.5.2.

Generative Models In generative models, multi-class decisions arise very naturally. The class with the highest probability $p(y = k|\mathbf{x})$ for the observed feature vector $\mathbf{x}$ is chosen.

3.4. Multi-class Wrappers

Boosting and SVMs, amongst other classification approaches, don't allow to solve multi-class problems directly [Hsu and Lin, 2002]. Their original derivation was restricted to binary problems only. This section introduces multi-class wrappers. They are used to construct multi-class classifiers using binary classifiers. Therefore they can be used to adapt arbitrary classification approaches to the multi-class case.

A multi-class problem is split into a number of binary problems:

$$y \in \mathcal{Y} \mapsto \mathbf{y} \in \mathbf{Y} \quad . \tag{3.2}$$

The Matrix $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_C\} \in \{-1, 1, d\}^{M,C}$ defines the binary problems to be solved. Samples marked with d will be ignored for that binary problem. The M Rows correspond to the derived binary problems, columns correspond to input class labels. Each row defines a binary classification problem. The original multi-class labels of the samples are replaced with the binary labels of the respective columns. A column of $\mathbf{Y}$ reflects the assignment of one class over all binary problems and will be denoted as class prototype $\mathbf{y}_k$. A class prototype may also be viewed as a coordinate in a M-dimensional label space. The class prototype $\mathbf{y}_{y_i}$corresponding to the class of the i-th sample with class label y_i will be abbreviated $\mathbf{y}_i$.

Each of the binary problems defined by $\mathbf{Y}$ is solved by a binary classifier:

$$\hat{\mathbf{y}}_i^{(m)} = H^{(m)}(\mathbf{x}_i) \quad \text{with } m = 1, \ldots, M \quad , \tag{3.3}$$

where M denotes the number of binary problems defined by $\mathbf{Y}$. Again, to shorten notation, vector notion will be used: $\hat{\mathbf{y}} = H(\mathbf{x})$. The dimension of $h(.)$ and $H(.)$ respectively can be deduced from context.

Assignment to an input class label is achieved using the mapping function $K(.)$:

$$\hat{y} = K(H^{(1)}, \ldots, H^{(M)}) \quad . \tag{3.4}$$

The advantage of multi-class wrapper approaches is the use of "off-the-shelf" standard binary classifiers for each $H^{(m)}$. In addition, classifiers and multi-class

extension are separated, allowing for a modular combination of techniques. A possible drawback of multi-class wrappers is the potentially poorer efficiency of the resulting classifiers, due to missing information exchange between binary classifiers.

Multi-class wrapper techniques can be divided into three general approaches: one-vs-all (1-vs-all), on-vs-on (1-vs-1) and Error Correcting Codes (ECC)[2]. These approaches are introduced briefly in the following.

3.4.1. One vs all (1-vs-all)

One vs all proposed in Bottou *et al.* [1994] was the first approach for combining binary classifiers to build multi-class classifiers. The multi-class problem is split into $M = C$ binary problems. Each binary problem separates one class from the remaining classes:

$$\mathbf{y}_k^{(m)} = 2 \cdot \mathrm{I}_{(m=k)} - 1 \quad \text{with } m \in 1, \ldots, M \ . \tag{3.5}$$

Since all input samples are used for each binary problem, the training as well as evaluation effort grows linearly with the number of classes C[3].

All classifiers are evaluated in order to induce a class hypothesis for a given feature vector. A sample is assigned the label of the respective classifier giving the highest output:

$$\hat{y} = K(H^{(1)}(\mathbf{x}), \ldots, H^{(C)}(\mathbf{x})) = \arg\max_k \left(H^{(k)}(\mathbf{x})\right) \ . \tag{3.6}$$

An extension of the 1-vs-all approach to improve its performance is proposed in Garcia-Pedrajas and Ortiz-Boyer [2006]. By combining 1-vs-all and 1-vs-1 (see next section) approaches, the performance can be improved slightly.

3.4.2. One vs one (1-vs-1)

The motivation behind the one vs one approach is that it is expected that pairwise sub-problems can be solved using less complex classifiers (see figure 3.1 for a visualization). Thus the overall classifier bias - compared to 1-vs-all classifiers - is reduced using this technique. Because of that, less powerful classifiers can be used to solve the binary problems generated by the 1-vs-1 technique.

[2] Also denoted as Error Correcting Output Codes (ECOC).

[3] This statement is valid for most, but not all types of classifiers. For example the evaluation effort of SVM's will, in most cases, scale more favorable than linear, since support vectors may be shared between binary problems and thus expansive kernel-evaluations are reduced.

Algorithm 6: Multi-class training using 1-vs-all wrapper.

Input: samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
binary classifier
number of columns of code matrix
Output: multi-class classifier $H(\mathbf{x}) \leftarrow \arg\max_{k \in \mathcal{Y}} \left(H^{(k)}(\mathbf{x}) \right)$

1 **for** *for k in $\mathcal{Y}$* **do**
2 Assign binary labels: $y'_i = 2 \cdot \mathrm{I}_{(y_i = k)} - 1$;
3 Train binary classifier $H^{(k)}(\mathbf{x})$ on binary problem;
4 **end**

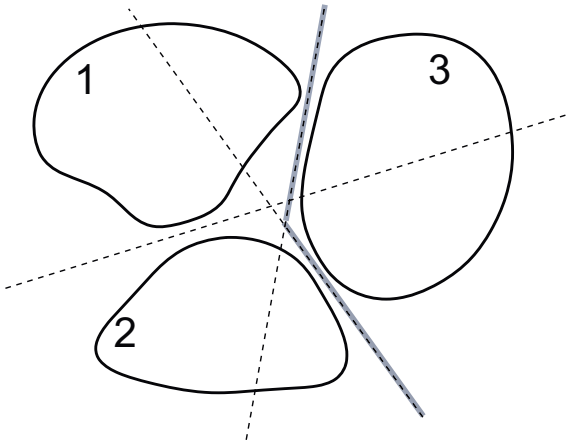

Figure 3.1.: Motivation of 1-vs-1 approach in comparison to 1-vs-all for a 3-class problem. The image shows possible data separations for the 1-vs-1 subproblems (broken lines) and one example of a 1-vs-all subproblem (gray line). The binary sub-problems may be separated using linear classifiers. This is not possible for the 1 vs all problems

The 1-vs-1 technique was first proposed in Friedman [1996]. Each binary problem deals with two input classes. All possible combinations of two classes are considered. Therefore a total of $C(C-1)/2$ classifiers are built. The binary problems can be written as:

$$\mathbf{y}^{(k_1 k_2)} = \begin{cases} +1 & \text{for} y = k_1 \\ -1 & \text{for} y = k_2 \\ d \text{ (don't care)} & \text{otherwise} \end{cases} \qquad \text{for } k_1 \neq k_2 \ , \tag{3.7}$$

In the binary problems the samples labeled with "don't care" are ignored. From an evaluation of a binary classifier $H^{(k_1 k_2)}$ it can be concluded which of the two classes k_1 and k_2 is more probable. Through evaluation of one particular binary classifier $H^{(k_1 k_2)}(\mathbf{x}_i)$, however, no final conclusion on the label y_i can be drawn. To be able to determine the class label, all binary classifiers have to be taken into consideration.

Since the overall number of necessary binary classifiers depends quadratically on the number of classes C, training seems to be prohibitively expensive. On the other hand, each binary classifier needs to consider samples from two classes only, involving on average $2/C$ samples for each binary problem. In practice, training costs depend strongly on how the particular classifier scales depending on the number of samples. For example, 1-vs-1 is especially well suited to be used together with support machine classifiers. When training SVM's, the training effort depends quadratically on the number of samples involved. In consequence, the higher number of classifiers and the lower effort to train the separate classifiers cancel out one another. Thus, the training effort for SVM's with a fixed number of samples is approximately constant, independent of the number of classes involved. In general, classifiers with poor scaling with regard to the number of training samples, benefit from the splitting into smaller subproblems as performed by the 1-vs-1 approach.

Determination of final class labels is not as straightforward as for the 1-vs-all approach. A number of approaches, each with specific advantages have been proposed. The two most common approaches are discussed briefly here. A deeper discussion can be found for example in Phetkaew *et al.* [2003].

Voting scheme. In order to induce a decision, all binary classifiers are evaluated. For each class, it's number of wins for classifiers trained using this class is accumulated. A sample is assigned the label of the class receiving the most votes.

$$\hat{y} = \arg\max_k \left(\sum_{k' \neq k} \mathrm{I}_{\left(H^{(kk')}(\mathbf{x})>0\right)} \right) \ . \tag{3.8}$$

Algorithm 7: Multi-class training using 1-vs-1 wrapper.

Input: Samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
Binary classifier
Number of columns of code matrix
Output: Multi-class classifier
Voting scheme: $H(\mathbf{x}) \leftarrow \arg\max_{k \in \mathcal{Y}} \sum_{k' \neq k} H^{(k,k')}(\mathbf{x})$
Graph based: $H(\mathbf{x}) \leftarrow K\left(H^{(1,2)}(\mathbf{x}), \ldots, H^{(C-1,C)}(\mathbf{x})\right)$,
where $K(.)$ represents a traversion of a directed graph

```
for i = 1, ..., C − 1 do
    for j = i + 1, ..., C do
        Train binary classifier H^(i,j)(x) using samples from classes i and j;
    end
end
```

For confidence-rated classifiers, the respective confidences of the relevant classifiers are accumulated and sample is assigned the label of the class with highest accumulated confidences. Therefore, in contrast to the graph-based approaches discussed below, the voting scheme is able to deal with confidence rated classifiers. When using the voting scheme, all binary classifiers have to be evaluated in order to generate a hypothesis on the label of a sample $\mathbf{x}$. Therefore evaluation effort grows quadratically with the number of classes involved. For classifiers, where the evaluation costs are not dominated by the actual classifier evaluation, better scaling is achieved. For example the evaluation costs of SVMs is dominated by the kernel calculations. If the same support vectors are used in multiple binary problems, the kernel has to be evaluated only once. In consequence, costs of SVM evaluation does not depend on the number of classes C, as long as the complexity of the classification problems is comparable.

Graph-Based Schemes For this approach, the binary classifiers are arranged in directed graphs. One example of the technique is DDAG (Directed Decision Acyclic Graphs), proposed by Platt *et al.* [2000]. The $C(C-1)/2$ classifiers are arranged as depicted in Fig. 3.2. The number of levels of the graph depends linearly on the number of classes C. Thus evaluation of the graph is more efficient than using a voting scheme. For evaluation of a sample, the tree is traversed from root to terminal leave. In each level of the decision graph, one class is removed from the set of possible output labels. Finally, samples are assigned the label of the only remaining class. If one classifier in the path commits an incorrect decision, the sample can't be assigned the true label. Depending on the ar-

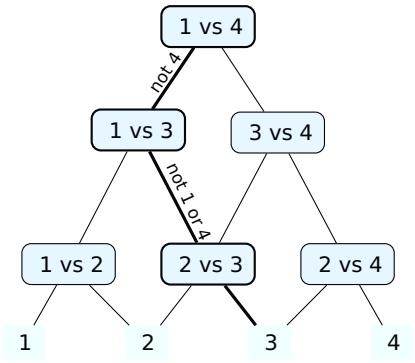

Figure 3.2.: Directed decision acyclic graph for multi-class decisions. At the root node all outcomes are possible. In each layer one class is removed from the set of possible outcomes until only the final output class remains. The evaluations for one sample with final decision for class 3 is highlighted. Along the used connections the excluded classes are shown. Note that samples of Class 2 need to win just one comparison in order to be chosen, while samples of class 1 or 4 need to win 3 comparisons.

rangement of the classes in the graph, the number of wins for a class to become the output class differs. Class-labels with a lower number of comparisons have an advantage over class-labels requiring a higher number of wins. Therefore the scheme prefers some classes over others, depending on the arrangement of classes in the graph. Derived techniques based on DDAG's overcome this drawback at the cost of increased complexity (e.g. Phetkaew *et al.* [2003]). In contrast to the voting scheme, the use of confidence-rated classifiers is not possible with graph-based schemes.

3.4.3. Error correcting output codes (ECOC)

Error Correcting Output Codes represent a different approach for building multi-class wrappers. The technique is motivated by channel coding used for transmission of digital signals. For channel coding redundant binary codes are assigned to each signal. The codes used differ in more than one digit. Therefore separated mutations of the transmitted code can be detected and potentially corrected. By increasing the redundancy of the codes, the signal can be made insensitive to multiple mutations of the transmitted code. The technique assumes that channel errors occur randomly and independent.

A similar approach can be taken to solve multi-class classification problems. The technique was proposed in Dietterich and Bakiri [1995]. Each class $k \in \mathcal{Y}$ is assigned a redundant binary class-prototype $\mathbf{y}_k$. The vectors together build a coding matrix $\mathbf{Y} = \{-1, 1\}^{C \times M}$. The coding matrix is chosen such that the minimal distance between two rows of the coding matrix (representing class-prototypes) is maximized. Each column of the coding matrix defines a binary classification problem, similar to the coding matrix used in the 1-vs-all scheme. One binary classifier is trained for each column. The assignment of the classifier outputs to a class hypothesis is done as follows

$$\hat{\mathbf{y}} = \arg\min_k \left(\|H(\mathbf{x}) - \mathbf{y}_k\|_1 \right) \quad , \tag{3.9}$$

where $H(\mathbf{x}) = [H^{(1)}(\mathbf{x}), \ldots, H^{(M)}(\mathbf{x})]^\mathsf{T}$ combines the output of all binary classifiers. Utilization of other distance functions is possible.

A coding matrix with M columns is - with no redundancy - sufficient to separate $2^M - 1$ classes. In order to achieve a fair level of redundancy and thus error tolerance, typically M is chosen considerably higher than the minimal necessary number of columns.

For suitable choice of coding matrix $\mathbf{Y}$ and cost function in Eq. (3.9), ECOC is equivalent to the 1-vs-all approach. Extensions to formulate 1-vs-1 problems through coding matrices are straightforward. Thus ECOC can be seen as a basic principle for building multi-class classifier wrappers, where 1-vs-1 and 1-vs-all

are special cases. For the sake of clarity, the approaches are treated independently in this work.

A potential problem of using redundant codes for building multi-class wrappers is that errors - in contrast to channel coding applications - can't be assumed to be independent for classification. Since some pairs of classes share more similarity than others, binary classifiers will more likely produce errors in these cases. When a pair of classes is very similar, the binary classifiers will tend to assign the same binary labels to them. Adding more codes will not help to solve this problem. Or put in another way: the separation of classes can't be improved arbitrarily by adding columns to the coding matrix. Thus, classification performance is expected to saturate after a certain minimum number of columns in the coding matrix is reached.

Algorithm 8: Multi-class training using error correcting codes.

Input: Samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$
Binary classifier
Number of columns of code matrix

Output: Multi-class classifier $H(\mathbf{x}) \leftarrow \arg\min_{k \in \mathcal{Y}} \sum_{m=1}^{M} |\mathbf{Y}^{(k,m)} - h_m(\mathbf{x})|$

1 Generate code matrix $\mathbf{Y} \in \{-1, 1\}^{C \times M}$;
2 **for** *for each column* m *of* $\mathbf{Y}$ **do**
3 Assign binary labels to the input classes, depending on the m-th column of $\mathbf{Y}$;
4 Call binary classifier to construct classification rule $h_m(\mathbf{x})$;
5 **end**

3.5. Multi-Class Boosting Techniques

Standard boosting techniques for binary problems require the base classifiers to achieve empirical error rates of less than 50%. For binary problems, this means that the classifier needs to be only slightly better than random guessing. For multi-class problems, random guessing will result in error rates of approximately $1 - 1/C$. Therefore base classifiers have to perform substantially better than random guessing to achieve errors below 50%. Thus, applying standard Boosting formulations on multi-class problems puts higher demands on the base classifiers. Using more powerful classifiers to achieve the required error rates is not desirable, since it will likely lead to overfitting problems. These problems can be avoided by utilizing boosting techniques designed specifically

for multi-class problems, which remove the hard requirement on the empirical error rates achieved by the base classifiers. The techniques work by adapting boosting cost functions and sample reweighting to the multi-class case and will be discussed in the following.

Some multi-class boosting techniques only apply small adaptions to the original boosting approaches. They merely adapt the cost functions in a way to reduce demands on performance of the base classifiers and thus prevent early stopping of the algorithms (e.g. Zhu *et al.* [2009]). For multi-class problems the acceptable error rate of the base classifiers is raised to $(C-1)/C$. If the weighted training error of each new base classifier is lower, the combined training error of all base classifiers converges to zero.

The most direct approach to build multi-class boosting classifiers is to use categorical base-classifiers that directly return multi-class labels. By design, it is not possible to utilize confidences when using this technique. Therefore only a part of the available information is used. Since only the label of the strongest class is returned, no assumptions can be made about the relative frequency of classes. The relative frequency of the strongest class may range between $1/C$ and 1. Therefore the amount of information lost, by using categorical outputs, increases as the number of classes C grows. On the other hand, by only storing the strongest label, categorical base classifiers are very memory efficient.

Other approaches for solving multi-class problems using boosting techniques are evaluated in more detail in the following sections. These approaches are closely related to the multi-class wrapper approaches discussed in Sect. 3.4. The difference is, that the multi-class wrappers are directly incorporated in the multi-class boosting techniques. Binary base classifiers are sufficient, since the boosting-algorithm itself deals with the multi-class specifics. Therefore out-of-the-box classifiers can be used without adaption. In particular the use of confidence-rated base classifiers is possible, which will - in general - lead to improved efficiency of the boosted classifiers.

By merging multi-class wrappers with boosting algorithms, some shortcomings of using boosting algorithms inside multi-class wrappers are avoided. For example when using boosting for features selection in combination with multi-class wrappers, features are selected independently for each binary classification problem. Thus it is not possible to share features among different binary classification problems, which could lead to performance improvement. Torralba *et al.* [2004] show that sharing features will not only improve computational efficiency, but also improve generalization performance by using inter-class concepts[4]. Furthermore multi-class wrappers assume each binary problem to have equal complexity, e.g. same number of boosting rounds and same base

[4]For example the concept of wheels is shared between various vehicles, e.g. cars, bicycles.

classifiers. Therefore every binary classifier will use approximately the same model complexity. In general this assumption will not hold. Each individual binary problem solved in order to build a multi-class classifier will pose different requirements on the classifiers. Therefore the optimal parameter settings will also differ. Tuning the parameters for each subproblem leads to increased complexity and computational load. By directly incorporating multi-class extensions into boosting algorithms, this problem can be solved elegantly. The algorithms will automatically focus on more complicated subproblems and thus spent effort as needed. This, again, holds potential for building more efficient classifiers.

The different wrapper approaches discussed in Sect. 3.4 lead to specific multi-class boosting flavors. The following sections introduce these and also discuss how base-classifiers need to be designed. A new multi-class boosting flavor based on the 1-vs-1 approach is proposed in Sect. 3.5.5 and formulations for optimized base-classifiers are given. Section 3.6 shows a quantitative comparison of the presented techniques.

3.5.1. Error Correcting Code Base Classifiers

Similar to the wrapper approach discussed in Sect. 3.4.3, a code matrix $\mathbf{Y}$ is generated to define binary problems. But instead of building a full classifier for each binary problem, one binary base classifier is generated for each column of the code matrix. Since the columns of the code matrix can be generated during training, it is possible to adapt the partitioning of the classes to the training problem at hand. Examples of boosting flavors using error correcting codes are AdaBoost.M2, AdaBoost.OC [Schapire, 1997] and AdaBoost.ECC [Guruswami and Sahai, 1999] and derived techniques [Sun *et al.*, 2007] - also gives a comparison of the existing approaches). Algorithm 9 shows Boosting with error correcting codes. Individual boosting flavors using error correcting codes differ mainly in how the weights are calculated and new columns of the code matrix are generated. For simplicity, only GentleBoost.ECC[5] is discussed in the following and used for performance comparison in Sect. 3.6

Since in each round only one binary problem is solved, arbitrary base classifiers can be used without modifications. The weights of the samples for the binary problems solved in each boosting round, depends on the partitioning of the classes in the current boosting round and the outputs of the preceding base classifiers. Lower weights are assigned to samples that are well separated from the samples with opposite label. Samples with poor separation from the classes

[5]GentleBoost.ECC algorithm is similar to AdaBoost.ECC, but uses the same Gauss-Newton updates as GentleBoost.

Algorithm 9: GentleBoost.ECC

Input: Training samples $\{\mathbf{x}_i, y_i\},\ i = 1, \ldots, N$
Number of boosting rounds T
base learner

Output: multi-class classifier $H(\mathbf{x}) \leftarrow \arg\min_{k \in \mathcal{Y}} \sum_{t=1}^{T} |\mathbf{Y}^{(k,m)} - h_m(\mathbf{x})|$

Initialize sample weights: $c_i^{(k)} = \begin{cases} \frac{1}{N(C-1)} & \text{if } k \neq y_i \\ 0 & \text{else} \end{cases}$;

for $t = 1, \ldots, T$ **do**
  Generate new column of coding matrix: $\mathbf{Y}^{(:,t)} \in \{+1, -1\}^{C \times 1}$;
  assign binary labels to the input classes, depending on $\mathbf{Y}^{(:,t)}$;
  calculate sample weights for binary problem:
  $c_i' = \sum_{k \in \dagger} c_i^{(k)} \mathrm{I}_{\left(\mathbf{Y}^{(y_i,t)} \neq \mathbf{Y}^{(k,t)}\right)}$;
  Construct decision rule $h_t(\mathbf{x})$ by calling base learner for the current binary problem;
  Update sample weights: $c_i^k = c_i^k \exp\left(\frac{1}{2} h_t(\mathbf{x}_i)(\mathbf{Y}^{(k,t)} - \mathbf{Y}^{(y_i,t)})\right)$;
end

with other label receive higher weights respectively. Therefore the algorithm concentrates on discrimination from the critical classes for each sample. The weight of a sample always depends on the combination of classes with opposite label. Therefore the algorithm can't enforce separation of one sample from a particular class[6].

An interesting issue for the training of boosting with error correcting codes (discussed for example in Schapire [1997]), is the partitioning of the classes preceding each boosting round. The cost reduction achievable with the next base classifier strongly depends on this partitioning. By advantageous partitioning the task of the base-classifier can be simplified and thus the convergence of the algorithm can be improved. The number of possible partitionings grows exponentially, as the number of classes increases. Since the cost reduction depends not only on the partitioning, but also actual base classifier, it is not suitable to search for the optimal partitioning for problems involving more than 7 classes.

Techniques exist for finding good partitionings by solving optimization problems - but their application is complicated and performance of the base classifiers is not taken into account. In Schapire [1997] these techniques are discussed.

[6]Partitioning so that one sample can be discriminated from a particular class is possible, but since the same partitioning is used for all samples, it will not be optimal for all samples.

It is however concluded, that their application will not lead to a substantial performance improvement. Instead, it is proposed to use random code matrices[7]. Since the approach is much simpler and effects on performance are reported to be negligible, random code matrices are used in this work as well.

3.5.2. 1-vs-all Base Classifiers

In this setting boosting algorithms are combined with the 1-vs-all approach presented in Sect. 3.4.1. In contrast to boosting of error correcting codes, more than one binary problem is solved in each round of boosting. For the final classifier to predict the true output class, the following has to hold:

$$h^{(y_i)}(\mathbf{x}_i) > h^{(k)}(\mathbf{x}_i) \quad \forall k \neq y_i \ . \tag{3.10}$$

A solution can be found by maximizing $\mathbf{y}_i^\mathsf{T} h(\mathbf{x}_i)$ over all i. Boosting of 1-vs-all base classifiers was proposed under the name of AdaBoost.MH in Schapire and Singer [1999]. An adaption, using the same cost function as GentleBoost can be found in Alg. 10.

Algorithm 10: GentleBoost.MH algorithm

Input: Training samples $\mathbf{x}_i, y_i, \ i = 1, \ldots, N$
Number of boosting rounds T
Base learner

Output: Multi-class classifier $H(\mathbf{x}) = \arg\max_{k \in \mathcal{Y}} \sum_{t=1}^{T} h_t^{(k)}(\mathbf{x})$

Initialize sample weights: $c_i^{(k)} = \frac{1}{N*C}$;
for $t = 1, \ldots, T$ **do**
  for $k = 1, \ldots, C$ **do**
    Assign binary labels to the input classes: $y_i' = \begin{cases} 1 & \text{if } y_i = k \\ -1 & \text{else} \end{cases}$;
    Train binary classifier $h_t^{(k)}(\mathbf{x})$ using labels y_i' and weights $c_i^{(k)}$;
    Tpdate weights: $c_i^{(k)} = c_i^{(k)} \exp\left(-y_i' h_t^{(k)}(\mathbf{x}_i)\right)$;
  end
end

[7] Note that the random partitionings do not depend on the training performance of the problem and may be fixed before start of the actual training.

The binary classification problems defined by the 1-vs-all scheme are solved in each round of boosting.

$$y = k \in \mathcal{Y} \rightarrow \mathbf{y} \in \{-1, 1\}^{C \times 1} \text{ with } \mathbf{y}^{(j)} = \begin{cases} +1 & \text{for } j = k \\ -1 & \text{otherwise} \end{cases} . \tag{3.11}$$

When using confidence-rated base classifiers, each classifier may be viewed as constructing a regression function to the weighted classification problem in the label space spanned by the class prototypes $\mathbf{y}_k$.

The base classifiers $h(\mathbf{x}) \mapsto \mathbb{R}^{C \times 1}$ minimize the respective cost function of the used boosting scheme. For GentleBoost the costs are defined as

$$J = \sum_{i=1}^{N} \sum_{k=1}^{C} c_i^{(k)} \ , \tag{3.12}$$

with the sample weights

$$\mathbf{c}_i^{(k)} = \exp\left(-\mathbf{y}_i^{(k)} H^{(k)}(\mathbf{x})\right) \tag{3.13}$$

and $H(\mathbf{x}) = \sum_{s=1}^{t} h_t(\mathbf{x})$ in the t-th round of boosting. From the calculation of the sample weights, it becomes clear, that no interaction between the base classifiers takes place. Therefore optimization of the base classifiers $h_t(\mathbf{x})$ can be performed for each component separately. The approach is therefore equivalent to using GentleBoost in a 1-vs-all wrapper.

Boosting of 1-vs-all base classifiers can, however, be used to select features that are shared across multiple binary subproblems. The optimization rule becomes

$$\min_{f()} J\left(h(f(\mathbf{x}))\right) \ . \tag{3.14}$$

The optimization problems of the base classifiers is now connected through the feature selection process. Sharing of features leads to classifiers achieving equivalent performance with a reduced number of features.

3.5.3. 1-vs-1 Base Classifiers

The base classifiers solve a number of 1-vs-1 binary classification problems. For the base classifier to return the correct decision, the following must hold:

$$h^{(y_i)}(\mathbf{x}_i) > \max_{k \neq y_i} h^{(k)}(\mathbf{x}_i) \ . \tag{3.15}$$

A solution can be found by maximizing $h^{(y_i)}(\mathbf{x}_i) - \max_{k \neq y_i} h^{(k)}(\mathbf{x}_i)$. Note the slight differences between the formulations for 1-vs-all and 1-vs-1 base classifiers. The

1-vs-1 approach focusses on discrimination against the most critical class and ignores all remaining classes. In Freund and Schapire [1995] a multi-class boosting scheme based on the 1-vs-1 approach, named AdaBoost.M2, is proposed.

The cost function to be minimized for boosting of 1-vs-1 base classifiers in boosting schemes using exponential costs is

$$\begin{aligned} J &= \sum_{i=1}^{N} \exp\left(-h(y=y_i|\mathbf{x}) + h(y=k_i|\mathbf{x})\right) \\ &\text{with} \quad k_i = \arg\max_{k \neq y_i} h(y=k|\mathbf{x}) \quad . \end{aligned} \tag{3.16}$$

This leads to the following weights in boosting round t:

$$c_i^{(k)} = \begin{cases} \exp\left(\sum_{s=1}^{t}(-h_s(y=y_i|\mathbf{x}) + h_s(y=k|\mathbf{x}))\right) \\ \qquad \text{for } k = \arg\max_{k \neq y_i} \sum_{s=1}^{t} h_s(y=k|\mathbf{x}) \\ 0 \qquad \text{else} \end{cases} \tag{3.17}$$

In order to minimize the cost function, base classifiers (for GentleBoost) are generated following:

$$\begin{aligned} h(\mathbf{x}) &= \arg\min_{h \in \mathcal{H}} \left(\sum_{i=1}^{N} c_i^{(k_i)} (h(y=y_i|\mathbf{x}) - h(y=k_i|\mathbf{x}))^2 \right) \\ &\text{with} \quad k_i = \arg\max_{k \neq y_i} h^k(\mathbf{x}) \quad . \end{aligned} \tag{3.18}$$

The algorithm focuses on discrimination from the most critical false class. This approach may be numerically unstable. For a more stable solution, equation (3.16) can be transformed to factor in all binary comparisons:

$$J = \sum_{i=1}^{N} \sum_{k \neq y_i} \exp\left(-h(y=y_i|\mathbf{x}) + h(y=k|\mathbf{x})\right) \quad . \tag{3.19}$$

This leads to the sample weights:

$$c_i^{(k)} = \exp \sum_{s=1}^{t} (-h_s(y=y_i|\mathbf{x}) + h_s(y=k|\mathbf{x})) \tag{3.20}$$

and the following optimization rule for the base classifiers:

$$h(\mathbf{x}) = \arg\min_{h \in \mathcal{H}} \left(\sum_{i=1}^{N} \sum_{k \neq y_i} c_i^{(k)} (h(y=y_i|\mathbf{x}) - h(y=k|\mathbf{x}))^2 \right) \tag{3.21}$$

The cost function (Eq. (3.19)), as well as the optimization rule (Eq. (3.21)) are approximations of the cost function (3.16) and optimization rule (3.18) respectively. Instead of using the $\max$-function a quadratic approximation is used. This improves numerical stability of the optimization. As the difference in confidence between the strongest false class and the remaining false classes grows, the approximations approach the exact formulations. Both formulations aim at maximizing the difference between the outputs for the true class y_i and the strongest remaining class.

In the literature [Freund and Schapire, 1995], the same approach for training base classifiers as with 1-vs-all approach is used. This is, however, not optimal in 1-vs-1 settings. One approach to directly optimize Eq. (3.21) is to build one base classifier for each binary classification problem. In order to do so, the class prototypes are defined to represent all binary 1-vs-1 subproblems:

$$\mathbf{y}_i^{(y_i,k)} = 2\mathrm{I}_{(y_i<k)} - 1, \qquad \forall k \neq y_i \tag{3.22}$$

$$\mathbf{y}_i^{(k_1,k_2)} = \text{don't care} \qquad \forall k_1 \neq y_i \wedge k_2 \neq k_1 \ . \tag{3.23}$$

With that, 3.18 can be rewritten as:

$$\begin{aligned} h(\mathbf{x}) = \arg\min_{h\in\mathcal{H}} \left(\sum_{i=1}^{N} \sum_{k_1=1}^{C-1} \sum_{k_2=k_1+1}^{C} c_i^{(k_1k_2)} \left(y^{(k_1k_2)} - h^{(k_1k_2)}(\mathbf{x}) \right)^2 \right) \\ \text{with } c_i^{(k_1k_2)} = 0 \text{ if } y_i \neq k_1 \wedge y_i \neq k_2 \ . \end{aligned} \tag{3.24}$$

Since the classification problems are split up into independent components, the base classifiers may work on separated components. Evaluation can be performed similar as in 1-vs-1 wrappers, for example by using the voting scheme. A drawback of this direct approach is the quadratic dependency from the number of classes C of the number of components involved, leading to a high number of base classifiers to be evaluated. Also, when the technique is to be used for feature selection, treating the components independently will prevent the algorithm from sharing features across binary problems. In Sect. 3.5.5 a more practical formulation for building base classifiers in 1-vs-1 boosting settings is introduced. This formulation scales linearly with the number of classes C and can be used for feature selection advantageously.

3.5.4. Cost Functions of Multi-Class Boosting Techniques

This section analyzes the differences between the multi-class boosting schemes by analyzing the cost-functions for a three class problem. It is assumed, that

returned confidences for each of the possible output labels add up to one:

$$\sum_{k=1}^{C} H^{(k)}(\mathbf{x}) = 1 \ . \tag{3.25}$$

All multi-class boosting classifiers discussed above can be adapted to fulfill this condition. This scaling is loosely related to the condition $\sum_{k=1}^{C} p(y = k|\mathbf{x}) = 1$ for probabilities. However, not the same constraints apply for the outputs of the classifiers. The class prototypes span an equilateral simplex in label space. All outputs of classifiers, normalized[8] to fulfill Eq. (3.25) lie in a $C - 1$ dimensional hyperplane, spanned by the class prototypes. Figure 3.3 shows the resulting plane for the three class problem. Note, that the class prototypes can be constructed in an $C - 1$-dimensional subspace such that Eq. 3.25 holds automatically. The measure to be minimized in classification problems is the number of erroneous classification on unseen data (generalization error). Since calculation of generalization error on training samples is not possible, the empirical error is used to approximate the generalization error. For large numbers of training samples:$N \to \infty$, the empirical error converges to the value of the generalization error. The costs of a misclassification doesn't depend on whether it occurred in proximity of the decision boundaries or not. Due to the discrete nature of the empirical error, it is hard to optimize.

The practical multi-class boosting algorithms discussed above use continuous approximations of the empirical error - that can be optimized easily. Also, optimizing continuous cost functions leads to maximization of classifier margin [Schapire *et al.*, 1998] - which is known to help improving generalization performance. In order to approximate the decision surface tightly, the cost functions should closely resemble the qualitative characteristics of the empirical error.

The cost function for the different multi-class boosting schemes with exponential cost function are shown in Fig. 3.4[9]. The level-lines in the plots show, that the 1-vs-1 approach with $\max$-function gives the best qualitative approximation of the empirical error. The quadratic approximation of the 1-vs-1 costs also gives a good approximation for high output confidences of $H^{(k)}(\mathbf{x})$. Therefore for higher numbers of boosting rounds, the quadratic approximation will converge towards the exact 1-vs-1 costs. The 1-vs-all costs are a poor approximation of the empirical error. In particular, the gradient of the 1-vs-all cost function doesn't point towards the next relevant decision surface (as it is the

[8] For base classifiers returning a weighted mean of the class prototypes of the individual samples (e.g. decision stumps), the condition is always fulfilled.

[9] The cost function associated with error-correcting-codes is not depicted here, since it is identical to the cost function of the 1-vs-all approach.

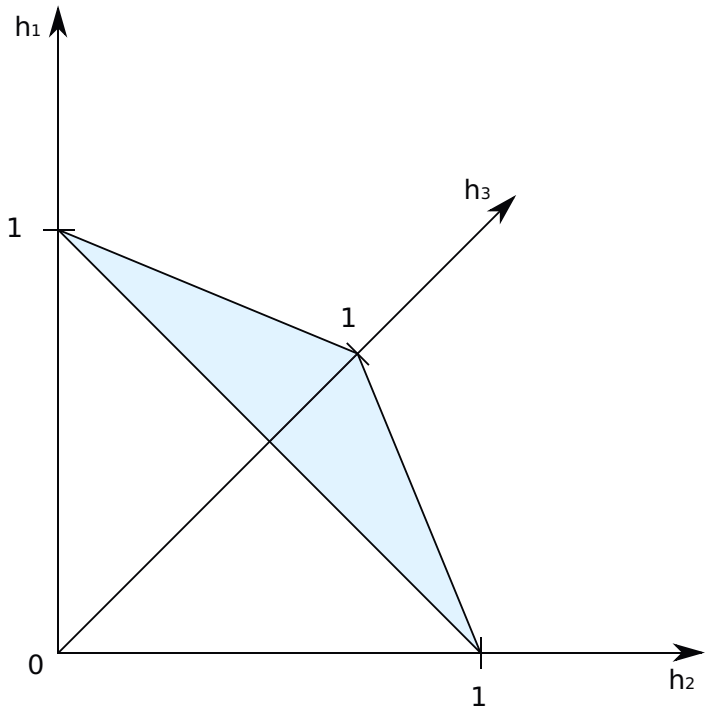

Figure 3.3.: Normalized outputs of base classifiers for a three class problem. The sum of the outputs is scaled such that the sum of the outputs over all classes is one. Note that under the condition $0 \leq h_j \leq 1, \forall j$, the outputs of the base classifiers can be interpreted as probabilities.

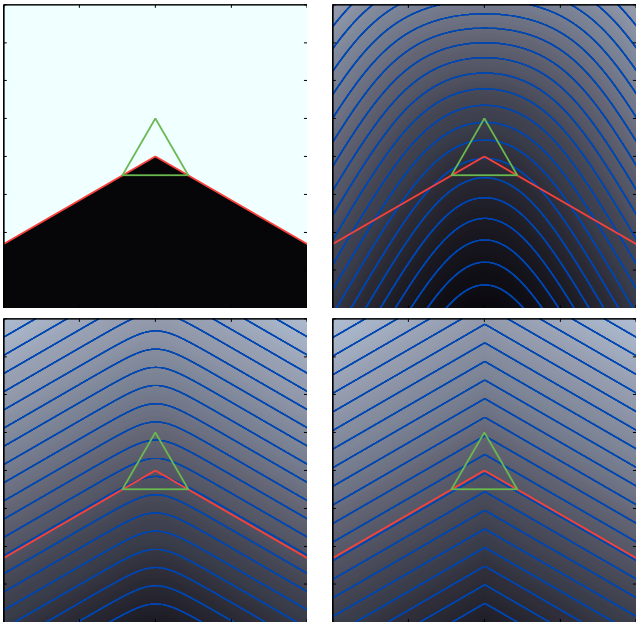

Figure 3.4.: Multi class costs functions. For visuallization, the 2-dimensional plane spanned by the class prototypes is shown. It is assumed that all classifier outputs lie within this plan. Class boundaries are depicted by red lines. The region of the true class is pointing downwards (red line). Upper left: empirical error, upper right: 1-vs-all error (Eq. (3.12)), lower left: 1-vs-1 approximation (Eq. (3.19)) , lower right 1-vs-1 max (Eq. (3.16)). The green triangles depict the area for which the outputs of the classifier are smaller than one.

case for the 1-vs-1 approach). Due to this, more effort than necessary is used to move samples to the right side of the decision surface.

The analysis of the cost functions of the presented multi-class boosting schemes shows, that the cost functions used with the 1-vs-1 approaches give a closer approximation of the empirical error. Through this, the classification problem can be solved more efficiently. On the other hand, training boosting schemes using 1-vs-1 costs is more expensive. In which way the different cost functions affect the classification performance quantitatively will be examined in the experiments of Sect. 3.6.

3.5.5. Efficient 1-vs-1 Base Classifiers

Optimizing the 1-vs-1 base classifiers following the scheme presented in Sect. 3.5.3 (Eq. (3.18)) is not practically suitable due to the high number of base classifiers to train and the bad scaling behavior. In addition, evaluation at runtime is expensive. In AdaBoost.M2, proposed in Freund and Schapire [1995], the class prototypes are build in the same manner as for 1-vs-all problems (Sect. 3.5.2) and classifiers are optimized on a component-wise basis. This approach, however, is not optimal for building 1-vs-1 base classifiers, since the cost function of the base classifiers doesn't correspond to the cost function of the boosting scheme.

In this section, another approach to constructing 1-vs-1 base classifiers is proposed. Instead of training independent pairwise base classifiers, a system of connected base classifiers is used. These can be used to reconstruct all pairwise decisions. The number of components used in this system depends linearly on the number of classes C.

Optimization takes part in a $C-1$-dimensional label space. The class prototypes are defined by the following properties:

- The class prototypes are defined in an minimal spanning label space: $\mathbf{y} \in \mathbb{R}^{C-1}$.
- All class prototypes have the same distance from the point of origin: $\|\mathbf{y}_k\|_2 = 1 \quad \forall k \in \mathcal{Y}$.
- The pairwise distance between the class prototypes are constant for all class prototypes: $\|\mathbf{y}_m - \mathbf{y}_n\|_2 = b$ für $m \neq n$.

This construction defines an equilateral simplex with corners representing the class prototypes. Therefore the technique will be refered to as Boosting.simplex in the following. The class prototypes for the three class case are shown in Fig. 3.3. For problems involving only two classes the class prototypes reduce to the

standard formulation for binary problems with $y_1 = 1$ and $y_2 = -1$. Hence - in contrast to the 1-vs-all formulation - binary problems are consistently within this formulation.

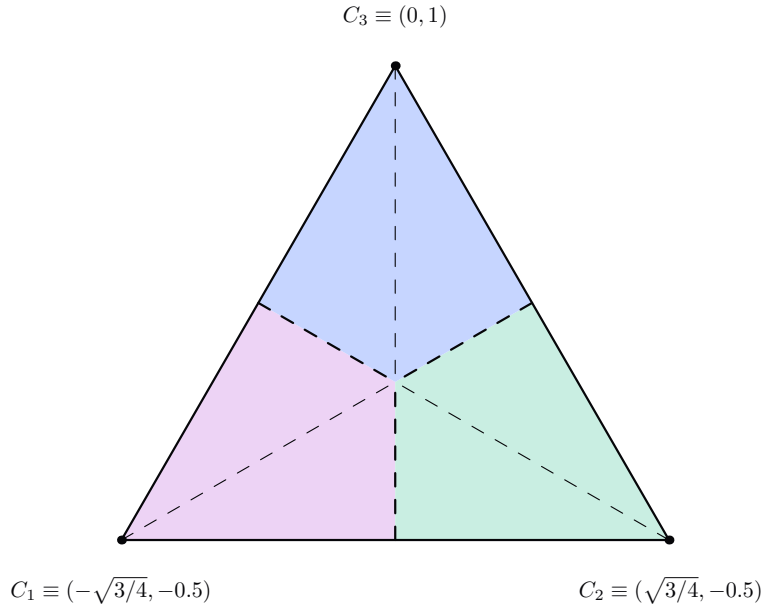

Figure 3.5.: Class prototypes for three class problem. The bold dashed lines mark the relevant class thresholds. The light dashed lines mark binary thresholds that are irrelevant since they are masked by another class. Only class surfaces marked by bold dashed lines are relevant for classificationDerivation of class prototypes from class probabilities.

The confidences of the classes can be calculated as:

$$h(y = k|\mathbf{x}) = \mathbf{y}_k^\mathsf{T} h(\mathbf{x}) \quad . \tag{3.26}$$

By construction of the class prototypes, the following holds for the sum of confidences:

$$\sum_{k=1}^{C} h(y = k|\mathbf{x}) = 0 \quad \forall \mathbf{x} \quad . \tag{3.27}$$

With this, Eq. (3.19) can be reformulated as:

$$J = \sum_{i=1}^{N} \sum_{k \neq y_i} \exp\left((-\mathbf{y}_i^\mathsf{T} + \mathbf{y}_k^\mathsf{T}) h(\mathbf{x})\right) \quad . \tag{3.28}$$

Analogous to the derivation of the optimization function for base classifiers used with GentleBoost [Friedman *et al.*, 1998] the following optimization rule can be formed:

$$h(\mathbf{x}) = \arg\min_{h \in \mathcal{H}} \left(\sum_{i=1}^{N} \sum_{k \neq y_i} c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} (h(\mathbf{x}) - \mathbf{y}_i) \right)^2 \right) \quad . \tag{3.29}$$

This rule can be used to derive optimal solutions for various base classifiers, as shown below for partition based classifiers. The derivations for neural networks and smoothing splines can be found in the appendix (Appendix D).

Algorithm 11: GentleBoost.simplex algorithm.

Input: Training samples $\mathbf{x}_i, y_i,\ i = 1, \ldots, N$
Number of boosting rounds T
Base learner

Output: Multi-class classifier $H(\mathbf{x}) = \arg\max_{k \in \mathcal{Y}} \mathbf{y}_k^\mathsf{T} \left(\sum_{t=1}^{T} \mathbf{h}_t(\mathbf{x}) \right)$

1 Transform labels into $C-1$-dimensional label space: $y_i \mapsto \mathbf{y}_i \in \mathbb{R}^{(C-1)\times 1}$;

2 Set initial hypothesis to $\mathbf{h}_0(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i$;

3 Initialize sample weights: $\mathbf{c}_i^{(k)} = 1$;

4 **for** $t = 1, \ldots, T$ **do**

5 Calculate sample weights:
$\mathbf{c}_i^{(k)} = \mathbf{c}_i^{(k)} \exp\left(\mathbf{h}_t^\mathsf{T}(\mathbf{x}_i)(\mathbf{y}_i - \mathbf{y}_k) \right)$ for $y_i \neq y_k$
$\mathbf{c}_i^{(k)} = 0$ else;

6 Train multi-class base classifier:
$\mathbf{h}_t(\mathbf{x}) = \frac{E_\mathbf{c}\left[\sum_k \mathbf{c}^{(k)} (\mathbf{y}_k - \mathbf{y}_i)^\mathsf{T} (\mathbf{y}_k - \mathbf{y}_i) \mathbf{y}_i \right]}{E_\mathbf{c}\left[\sum_k \mathbf{c}^{(k)} (\mathbf{y}_k - \mathbf{y}_i)^\mathsf{T} (\mathbf{y}_k - \mathbf{y}_i) \right]}$;

7 **end**

Optimal Output-Values for Partition based Base Classifiers

Partition-based base classifiers divide the feature space into regions. A set of training samples P_m is located in each region. Each region is assigned a constant output value h_m, that minimizes equation (3.19), for the samples in $\mathcal{P}_m$. Typical examples for partition-based base classifiers are decision stumps, decision trees and histograms.

The optimization routine for 1-vs-1 base classifiers is

$$\hat{h}_m = \arg\min_{h \in \mathcal{H}} \left(\sum_{i \in \mathcal{P}_m} \sum_{k \neq y_i} c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} (h - \mathbf{y}_i) \right)^2 \right) \quad . \tag{3.30}$$

To find the minimum, the derivative with respect to h is calculated and component-wise set to zero:

$$\sum_{i \in \mathcal{P}_k} \sum_{k \neq y_i} c_i^{(k)} (\mathbf{y}_i - \mathbf{y}_k)(\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} (h - \mathbf{y}_i) = \mathbf{0} \quad . \tag{3.31}$$

Solving for h gives the result:

$$\begin{aligned} h^\mathsf{T} = & \left(\sum_{i \in P_k} \sum_{k \neq y_i} c_i^{(k)} \mathbf{y}_i^\mathsf{T} (\mathbf{y}_i - \mathbf{y}_k)(\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \right) \\ & \times \left(\sum_{i \in P_k} \sum_{k \neq y_i} c_i^{(k)} (\mathbf{y}_i - \mathbf{y}_k)(\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \right)^{-1} \end{aligned} \tag{3.32}$$

Derivation of optimal output values for Artificial Neural Networks, smoothing-splines and SVMs can be found in Appendix D.

Comparison to other Multi-Class Boosting Schemes

By construction of the label space, the sum of the confidences of a sample over all classes is constant. The exponential cost of a sample is as follows:

$$e^{-\hat{\mathbf{y}}_i^\mathsf{T} \mathbf{y}_i} = \begin{cases} e^{-1} & \text{if} \quad \hat{\mathbf{y}}_i = \mathbf{y}_i \\ e^{\frac{1}{C-1}} & \text{else} \end{cases} \quad . \tag{3.33}$$

Before the training $\hat{\mathbf{y}}_i^\mathsf{T} \mathbf{y}_i = 0$ holds for all samples and therefore exponential costs of 1 are produced by each sample. For binary problems, the classifier will focus on not to have any false classification outputs with high confidences, since the overall costs are dominated by false classifications[10]. For problems with higher number of classes the overall costs will be more and more dominated by the correct classifications[11]. This reflects the fact, that it is more difficult to find the true class for high number of classes involved.

[10] Binary: A correct classification with confidence 1 leads to cost improvement by $1 - 1/e \approx 0.63$, while a false classification with confidence 1 increases the costs by $e - 1 \approx 1.72$.

[11] Three classes: A correct classification with confidence 1 leads to cost improvement by $1 - 1/e \approx 0.63$, while a false classification with confidence 1 increases the costs by $e^{0.5} - 1 \approx 0.64$.

Also, when measuring the performance of base classifiers using the linear edge $\hat{\mathbf{y}}_i^{\mathsf{T}}\mathbf{y}_i$, an edge of 0 is achieved with random guessing. This is consistent with the behavior in the binary case. In other work [e.g. Zhu *et al.*, 2009] normalization factors are used to adjust the edge of the base classifiers to prevent the early stopping problems encountered with multi-class problems. Defining class prototypes in an equilateral simplex is an elegant and consistent approach to achieve the same results without altering the boosting algorithm itself.

3.6. Experimental Section

In order to analyze the performance of the multi-class classification approaches discussed in this chapter, a number of experiments on the UCI machine learning data base[12] was conducted. The 24 datasets utilized in the experiments were chosen to represent a fair variety of multi-class classification problems. A descritption of the datasets can be found in appendix C. All data bases provide an appropriate set of input features, so that no feature selection or feature generation has to be performed. Due to the public availability of the datasets, a good comparability of the results is given.

This chapter focussed on the construction of boosted classifiers for solving multi-class problems. The optimization target was to find classifiers which solve the given problems very efficiently and - at the same time - achieve competitive performance. Therefore classifiers of similar computational complexity are compared in the experiments. The computational complexity is measured by the number of feature evaluations used by a given classification rule. The motivation for this metric is that for image processing tasks, the computational performance of the classifiers is often dominated by the costs associated with feature evaluation. When Boosting is used as a feature selection scheme [e.g. with Haar-like features in Viola and Jones, 2002b], efficient utilization of features is crucial since often computational resources are limited.

3.6.1. Results

For all experiments the GentleBoost scheme was used as the foundation for the derived multi-class classification scheme[13]. Thus, the base classifiers were trained to minimize the weighted squared error, according to the optimization rules introduced above. Also, for all experiments decision tree base classifiers are used. The trees are constructed following Alg. 2. Three sets of experiments

[12] `http://archive.ics.uci.edu/ml/`

[13] The only exception is the AdaBoost scheme used with boosting of error correcting codes (eccada).

Abbreviation	Algorithm
1-vs-1-voting	Wrapper based on one-vs-one scheme (Sect. 3.4.2). The voting scheme is used to calculate the output label of the classifier.
1-vs-1-ddag	Wrapper based on one-vs-one scheme. The Directed Decision Acyclic Graph scheme is used for evaluation.
1-vs-all	Wrapper based on one-vs-all scheme (Sect. 3.4.1).
ecoc	Wrapper using error correcting output codes (Sect. 3.4.3).
ecc-ada	Boosting error correcting codes with AdaBoost as base algorithm.
ecc-gentle	Boosting error correcting codes with GentleBoost as base algorithm (Sect. 3.5.1).
mh	Boosting 1-vs-all base classifiers (Sect. 3.5.2).
simplex	Boosting of 1-vs-1 base classifiers in the simplex (Sect. 3.5.5). Quadratic approximation of the max-function is used.

Table 3.1.: Classifiers used for evaluation of multi-class boosting schemes.

are conducted. The complexity of the used decision trees differs between the experiments. In the first experiment (Tab. 3.2) decision stumps are used. In the second experiment, decision trees with a fixed capacity of five internal nodes are used (Tab. 3.3). In the third experiment, the number of internal nodes of the decision trees is tuned for optimal performance (Tab. 3.4). In all experiments 200 rounds of boosting are trained. The performance of the classifiers is evaluated using tenfold cross validation (see Sect. B).

The multi-class boosting schemes used in the experiments are described in Tab. 3.1. For all classifiers, but AdaBoost.ECC (ecc-ada), the GentleBoost scheme is used. AdaBoost.ECC was included because it was the actual scheme proposed in Guruswami and Sahai [1999] and also to compare the performance of AdaBoost and GentleBoost.

Dataset	ecc-gentle	ecc-ada	1vs1-full	1vs1-ddag	1vsAll	ecoc	mh	simplex
anneal	0.0111	0.0200	0.0111	0.0067	0.0100	0.0167	0.0033	**0.0022**
audiology	0.2263	0.2834	0.2656	0.3455	0.2304	0.3678	0.1907	**0.1810**
autos	0.1998	0.2188	0.1802	0.1605	0.2093	0.3360	0.1855	**0.1407**
balance-scale	0.0768	0.0832	0.0368	0.0320	0.0752	0.0752	0.0751	**0.0159**
dermatology	0.0216	0.0270	0.0271	0.0378	0.0189	0.0299	0.0217	**0.0162**
ecoli	0.1731	0.1463	0.1463	0.2948	**0.1344**	0.1556	0.1347	**0.1344**
glass	0.2576	0.2673	0.2723	0.2911	0.2671	0.3550	0.2528	**0.2294**
hypothyroid	0.0042	0.0127	0.0037	0.0034	0.0037	0.0040	0.0032	**0.0026**
iris	0.0533	0.0533	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**
led24	0.2660	0.2660	0.2640	**0.2550**	0.2580	0.3090	0.2690	**0.2550**
letters	0.3756	0.4220	0.3808	0.2358	0.3731	0.5060	0.1622	**0.1556**
lymph	0.1681	0.1481	0.1486	0.1281	0.1548	0.1614	**0.1214**	0.1424
optdigits_train	0.0685	0.0719	0.0970	0.0343	0.0654	0.1253	0.0293	**0.0262**
page_blocks	0.0316	0.0338	0.0336	0.0303	0.0303	0.0402	0.0280	**0.0276**
pendigits_train	0.0699	0.0917	0.1120	0.0262	0.0628	0.1548	0.0209	**0.0145**
primary-tumor	**0.5073**	0.5250	0.5546	0.5573	0.5340	0.5369	0.5276	0.5277
satimage	0.1260	0.1466	0.1302	0.1097	0.1284	0.1539	0.1085	**0.1082**
segmentation_train	0.0857	0.0905	0.1000	0.0952	0.1143	0.1190	**0.0762**	0.0810
soybean	0.0673	0.1187	0.1464	0.1877	0.0879	0.1479	**0.0556**	0.0571
vehicle	0.2564	0.2622	0.2292	0.2268	0.2468	0.2610	**0.2186**	0.2244
vote	**0.0344**	**0.0344**	**0.0344**	**0.0344**	**0.0344**	**0.0344**	**0.0344**	**0.0344**
vowel	0.3030	0.3909	0.3737	0.1687	0.2838	0.4566	0.1273	**0.0657**
waveform-5000	0.1536	0.1544	0.1638	0.1610	**0.1522**	**0.1522**	0.1534	0.1602
yeast	0.4030	0.4158	0.4057	0.4642	**0.3882**	0.4549	0.4037	0.3969
number of wins	2	1	2	3	5	3	6	17
avg. ranks	4.5833	5.8750	5.2083	4.4167	4.0000	6.9167	2.8333	2.1667

Table 3.2.: Comparison of multi-class boosting techniques with decision-stump base classifiers.

Dataset	ecc-gentle	ecc-ada	1-vs-1-voting	1-vs-1-ddag	1-vs-all	ecoc	mh	simplex
anneal	0.0033	0.0033	**0.0011**	**0.0011**	0.0022	0.0033	**0.0011**	**0.0011**
audiology	**0.1597**	0.1603	0.2306	0.3632	0.1862	0.1990	0.1899	0.1899
autos	0.1414	**0.1412**	0.1707	0.1748	0.1755	0.2000	0.1855	0.1662
balance-scale	0.1119	0.1071	0.0960	0.0992	0.0895	0.0895	0.0833	**0.0640**
dermatology	0.0271	0.0324	0.0271	0.0406	0.0434	**0.0243**	0.0297	0.0325
ecoli	0.1463	0.1371	0.1580	0.3068	0.1521	0.1463	**0.1226**	0.1462
glass	0.2251	0.2158	0.2251	0.2344	0.2431	0.2342	0.2208	**0.1738**
hypothyroid	0.0026	0.0034	0.0034	0.0034	0.0034	0.0032	0.0027	**0.0021**
iris	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**
led24	0.2650	**0.2630**	0.2920	0.2950	0.2700	**0.2630**	0.2720	0.2700
letters	0.2262	0.2754	0.2566	0.1196	0.2384	0.3543	0.0564	**0.0529**
lymph	**0.1071**	0.1214	0.1414	0.1210	0.1548	0.1419	0.1286	0.1214
optdigits_train	0.0264	0.0351	0.0502	0.0262	0.0371	0.0633	0.0173	**0.0162**
page_blocks	0.0278	0.0272	0.0270	0.0274	0.0261	**0.0247**	0.0276	0.0259
pendigits_train	0.0136	0.0200	0.0412	0.0109	0.0180	0.0650	**0.0045**	**0.0045**
primary-tumor	0.5250	**0.5102**	0.5428	0.5959	0.5692	0.5310	0.5337	0.5424
satimage	0.1035	0.1075	0.0999	0.0920	0.0993	0.1260	**0.0844**	0.0887
segmentation_train	0.0667	0.0762	0.0952	0.1048	0.1048	0.0714	0.0667	**0.0619**
soybean	**0.0469**	0.0557	0.0762	0.1701	0.0585	0.0542	0.0528	0.0585
vehicle	0.2257	0.2150	0.2079	0.2067	0.2363	0.2090	0.2103	**0.2032**
vote	**0.0275**	0.0319	0.0366	0.0366	0.0366	0.0366	0.0345	0.0366
vowel	0.0667	0.0909	0.1677	0.1273	0.1010	0.1677	0.0303	**0.0202**
waveform-5000	0.1534	0.1530	0.1614	0.1566	**0.1528**	**0.1528**	0.1572	0.1582
yeast	0.4050	**0.3983**	0.4030	0.4683	0.4023	0.4016	0.4084	0.4003
number of wins	3	4	1	3	3	3	5	11
avg. ranks	3.5833	3.8750	5.3750	5.4583	5.4583	5.1667	3.7500	3.3333

Table 3.3.: Comparison of multi-class boosting techniques with decision-tree base classifiers with 5 internal nodes.

Dataset	Ecc-gentle	Ecc-ada	1-vs-1-full	1-vs-1-ddag	1-vs-all	ecoc	mh	simplex
anneal	0.0022	0.0022	**0.0011**	**0.0011**	0.0022	0.0033	**0.0011**	**0.0011**
audiology	0.1595	**0.1593**	0.2306	0.2306	0.1862	0.1909	0.1899	0.1773
autos	0.1414	0.1412	0.1605	0.1707	0.1755	0.1902	0.1705	**0.1407**
balance-scale	0.0768	0.0832	0.0320	0.0368	0.0752	0.0752	0.0703	**0.0159**
dermatology	0.0216	0.0244	0.0271	0.0271	0.0189	0.0216	0.0217	**0.0162**
ecoli	0.1374	**0.1195**	0.1580	0.1463	0.1344	0.1405	0.1226	0.1317
glass	0.1965	0.2069	0.2251	0.2251	0.2431	0.2342	0.2063	**0.1738**
hypothyroid	0.0026	0.0034	0.0034	0.0034	0.0034	0.0032	0.0026	**0.0021**
iris	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**
led24	0.2650	0.2630	**0.2550**	0.2640	0.2580	0.2610	0.2660	**0.2550**
letters	0.1687	0.2117	0.0882	0.1196	0.1863	0.2880	0.0412	**0.0398**
lymph	**0.1071**	0.1148	0.1081	0.1210	0.1281	0.1214	0.1214	0.1148
optdigits_train	0.0199	0.0230	0.0262	0.0262	0.0324	0.0413	**0.0128**	0.0144
page_blocks	0.0265	0.0272	0.0261	0.0270	0.0259	**0.0247**	0.0269	0.0259
pendigits_train	0.0081	0.0125	0.0109	0.0109	0.0125	0.0354	**0.0033**	0.0039
primary-tumor	**0.5073**	0.5102	0.5428	0.5428	0.5338	0.5222	0.5129	0.5130
satimage	0.0923	0.0990	**0.0828**	0.0920	0.0943	0.1148	0.0844	0.0833
segmentation_train	0.0667	0.0667	0.0952	0.0905	0.1048	0.0714	0.0667	**0.0524**
soybean	**0.0396**	0.0454	0.0762	0.0762	0.0585	0.0542	0.0498	0.0557
vehicle	0.2067	0.2150	0.2055	0.2032	0.2327	0.2079	0.2090	**0.1972**
vote	**0.0275**	0.0319	0.0344	0.0344	0.0344	0.0344	0.0344	0.0344
vowel	0.0657	0.0859	0.0909	0.1273	0.1000	0.1677	0.0303	**0.0202**
waveform-5000	0.1534	**0.1500**	0.1566	0.1566	0.1522	0.1522	0.1534	0.1582
yeast	0.3969	0.3956	0.4030	0.3989	**0.3882**	0.3909	0.4037	0.3969
number of wins	5	4	4	2	2	2	4	12
avg. ranks	3.2917	4.1250	4.5000	5.6667	5.4583	5.6667	4.2917	3.000

Table 3.4.: Comparison of multi-class boosting techniques with decision-tree base classifiers. Number of internal nodes was tuned between one and five nodes by cross-validation.

	Ecc-ada	1-vs-1-full	1-vs-1-ddag	1-vs-all	ecoc	mh	simplex
ecc-gentle	6-15-3	7-16-1	6-17-1	7-15-2	5-17-2	8-12-4	16-6-2
ecc-ada		9-13-2	7-15-2	7-13-4	7-16-1	12-10-2	15-7-2
1-vs-1-voting			4-8-12	8-13-3	10-12-2	13-8-3	17-3-4
1-vs-1-ddag				10-11-3	11-11-2	16-5-3	20-1-3
1-vs-all					8-12-4	16-6-2	19-2-3
ecoc						15-6-3	18-4-2
mh							15-6-3

Table 3.5.: win-loss records for multi-class boosting algorithms with decision trees (optimal number of internal nodes)

3.6.2. Discussion

Decision Stump Base Classifiers

The GentleBoost.simplex scheme proposed in Sect. 3.5.5 achieves the best performance on 17 out of 22 datasets. GentleBoost.MH and the 1-vs-all wrapper approach, winning 6 and 5 datasets respectively are the best remaining schemes. Boosting schemes with error correcting codes show relatively poor performance.

In order to judge, whether the observed difference in the performances of the different multi-class schemes are significant, a Friedman Test is conducted (Demšar [2006] and Sect. B). The null-hypothesis states that all algorithms are equivalent. The Friedman Statistic for the given average ranks of the classifiers is $\chi_F^2 = 66.88$. The null-hypothesis can be accepted at a confidence level of $\alpha = 0.05$ for $\chi_F^2 < 14.07$. Therefore the null-hypothesis is rejected and significant differences of the classifiers are evident.

A two-tailed Nemeny post-hoc test is used to find pairwise significant performance differences. The null-hypothesis states that two classifiers have equivalent performance. For the null-hypothesis to be accepted,

$$|R_m - R_n| < q_\alpha \sqrt{\frac{n_{algs}(n_{algs}+1)}{6n_{sets}}} \tag{3.34}$$

must hold, where R_m is the average rank of the m-th classifier, n_{algs} is the number of algorithms examined and n_{sets} denotes the number on datasets used. The values of q_α are distributed according to the studentized range statistic. For a confidence level of $\alpha = 0.05$, a critical value of $q_\alpha = 2.14$ results. Therefore GentleBoost.simplex performs significantly better than all other algorithms, besides from GentleBoost.MH and the 1-vs-all wrapper approach. Other significant differences can be found in Tab. 3.2.

Decision stump base classifiers have high bias. Also in this experiment the lowest number of feature evaluations was performed. The results show the

Boosting.simplex is especially helpful for problems where base classifiers are weak and the number of feature evaluations has to be minimized.

Decision Tree Base Classifiers with 5 Internal Nodes

GentleBoost.simplex again achieves the best performance of all algorithms. Multi-class boosting schemes show better performance than wrapper approaches. The performance differences between the multi-class boosting schemes with and without error correcting codes is relatively small. GentleBoost.MH and GentleBoost.simplex show better performance of datasets with high numbers of features/samples, whereas GentleBoost.ECC and AdaBoost.ECC seem to have advantages for datasets where overfitting is an issue.

Again, a Friedman Test is performed to asses the performance differences between the algorithms. The Friedman statistic takes a value of $\chi_F = 24.80$. Therefore the null-hypothesis is rejected. The Nemenyi post-hoc test indicates, that for example GentleBoost.simplex performs significantly better than all multi-class wrapper approaches. However, in total the differences of the performances of the algorithms is smaller than in the preceding experiment.

For most of the utilized datasets, the number of boosting rounds used together with the base classifiers used, are sufficient for the algorithms to converge. Therefore efficient utilization of base classifiers is not as important as for decision stump base classifiers. Consequently the performance advantage of the schemes using vector-valued base classifiers is low.

Decision Tree Base Classifier with optimal Number of Internal Nodes

When the number of internal nodes is optimized using cross-validation, GentleBoost.simplex achieves the best performance. Again, the specialized multi-class Boosting approaches outperform the wrapper schemes. GentleBoost.simplex and GentleBoost.ECC achieve approximately the same average ranks, while the direct comparison is more clearly in favor of GentleBoost.simplex (see. Tab. 3.5).

The value of the Friedman statistic is $\chi_F = 30.14$, leading to a rejection of the null-hypothesis. The post-hoc Nemeny test indicates significant performance differences between GentleBoost.simplex / GentleBoost.ECC and some wrapper approaches.

3.7. Conclusions

In this chapter multi-class boosting algorithms where examined. Numerous approaches can be found in the literature. However, exhaustive comparisons between different approaches are not presented frequently. In this chapter not

only a new multi-class boosting algorithm was presented, but also a experimental performance evaluation of state-of-the art multi-class boosting schemes was presented.

The first important result is that using multi-class wrappers together with boosting will not give optimal performance. One factor for this is that features are only used for binary subproblems. Also, schemes incorporating multi-class boosting directly are more flexible in adapting the problem at hand: which binary problems are critical, which classes share features and so on. In order to achieve better results with wrapper approaches, more knowledge of the classifier used needs to be incorporated. This, however, leads directly to multi-class boosting schemes.

Out of the direct multi-class boosting schemes, GentleBoost.simplex gives the best performance. GentleBoost.ECC and GentleBoost.MH follow closely. The differences between the two types of multi-class boosting approaches is only significant in scenarios where the overall bias of the boosted ensemble is relatively high, as in the experiments with decision stump base classifiers. In these cases GentleBoost.simplex and GentleBoost.MH do a better job in using features and base classifiers more efficiently. This property may help for feature selection problems, where the number of features necessary to reach a certain level of performance should be as low as possible[14].

Since only the outputs of binary classifiers need to be stored for multi-class boosting schemes based on error correcting codes, the memory requirements for one base classifier are lower than for the multi-class boosting schemes based on 1-vs-1 or 1-vs-all. It is not clear, whether this will lead to reduced memory requirements of the trained ensemble, since boosting ensembles using error correcting codes used a higher number of base classifiers in the experiments to reach the same error rates.

Putting all together, the decision which multi-class boosting scheme to use depends on a number of factors. The nature of the given classification problem (numbers of features, samples, classes) strongly affects the reachable performance. Also the choice of base classifier influences performance. In order to achieve good performance, the performances with GentleBoost.ECC and GentleBoost.simplex[15] should be evaluated in cross validation and the algorithm performing better for the given data should be chosen.

[14] Note, that the datasets used for performance evaluation did not include feature-selection problems. It is - however - expected, that the number of base classifier evaluations for the given scenarios correlates with the number of selected features in feature selection scenarios.

[15] GentleBoost.simplex and GentleBoost.MH can be used exchangeable, since their performances are close. In practice, GentleBoost.MH can be implemented more easily and training is cheaper.

A promissing direction for future work is incorporation of hierarchical schemes to improve scaling of the presented approaches with respect to the number of classes. Though GentleBoost.simplex scales advantageous compared to other multi-class boosting techniques, linear dependency on the number of classes is likely to be prohibitive for problems involving very high number of classes. Hierarchical schemes could be used to alleviate classification in these cases. In order to reduce the number of classes that needs to be handled in parallel, the classes should be clustered into concepts first, such that classes that are very similar will belong to the same concepts. The classification problem can then be solved in a multistage fashion. Clusters of classes and the respective sub-clusters can be arranged in a tree structure. With each stage, number of possible output classes is reduced, since all candidates belong to the succeeding cluster of the preceeding stage. Since each multiclass classifier will have to deal with a limited number of concepts only, the number of classes to handle in parallel is effectively reduced. In addition, the resulting classifiers can be expected to achieve better generalization than classifiers dealing with the full multi-class problem in parallel, since clusstering of the classes will increase the number of samples available to the classifiers of the first stages. The biggest challenge is to place classes into concept clusters. However, the weights returned by multi-class boosting-schemes can be incorporated to find the clustering, since similar classes will be linked by high weights.

4. Boosted Projections

4.1. Motivation

Decision trees 2.4.4 are the most popular base classifiers used with Boosting. They work by combining axis-parallel decisions to build a partitioning of feature space. The same principle is used for example in histogram and smoothing-spline base classifiers. Section 2.5.5 deals with Boosting of base classifiers working directly on input features (component-wise base classifiers), that can be used for feature selection, and discusses their properties. Given "rich" feature sets, possibly including many features that share information, these base classifiers are able to select a subset of features, sufficient to solve the given classification task.

Component-wise base classifiers achieve good performance for many classification problems. However, performance deteriorates if approximations of decision boundaries cannot be constructed efficiently using the given features. Also, if input features are correlated or each input gives only little discriminative information, approximation of decision boundaries is either very coarse or description becomes very complex. A good example of classification problems in which component-wise base classifiers give poor performance is classification of image patches based on pixel values.

To achieve good performance using component-wise base classifiers in cases where input features are weak, augmented features sets are commonly used. By combining input features, new features with better class separation are built. Providing an appropriate set of features is a challenging task, for which prior knowledge of the nature of the classification problem is necessary.

A new approach for solving the problem of generating feature sets is introduced in Sect. 4.2. By combining feature generation with Boosting, Features are trained as necessary and only limited prior knowledge is needed. This removes the need for designing feature sets before training starts. The features used in the proposed system are linear projections. The utilized base classifiers are smoothing splines. This simple scheme is able to model almost arbitrary decision surfaces[1]. At the same time it can be regularized conveniently. The number of design parameters of the system is low, making the adaption to new

[1]The actual surfaces that can be modeled are limited by the allowed complexity of the smoothing-splines.

problems easy. In Sect. 4.3 related work is presented and similarities and differences to our approach are discussed.

The performance of the presented approach is evaluated in Sect. 4.4. A set of experiments on 30 datasets from UCI machine learning repository shows that results of the new system are significantly better than the results of state-of-the-art classifiers.

4.2. Training Boosted Projections

A disadvantage of component-wise base classifiers (Sect. 2.5.5) is that they cannot model arbitrary decision surfaces. Also, component-wise base classifier will learn too complex models when lacking good input features. To be able to approximate arbitrary decision surfaces, one could use multivariate base classifiers. However, fitting these is difficult due to the "curse of dimensionality". In addition evaluation of multivariate base classifiers is often relatively expensive.

Another common approach is to use component-wise base classifiers on augmented feature sets. By adding features the approximation of more complex decision surfaces is possible. At the same time, favorable properties of component-wise base classifiers are preserved. Figure 4.1 compares the output of component-wise base classifiers on original inputs and an augmented feature set for a two-dimensional problem. The decision surface cannot be modeled using component-wise functions. As expected, Boosting of component-wise base classifiers gives poor performance. An augmented feature set is built using linear combinations of inputs, giving 8 feature directions evenly distributed in the 2D plane. Boosting component-wise base classifiers on this augmented feature set significantly improves performance, since by using the new features, class separation using component-wise base classifiers becomes possible. The output of a classifier built by boosting decision trees is shown for comparison. Though decision trees are able to separate the classes using input features only, the resulting decision surface is very ragged.

As seen above, use of augmented feature sets can significantly improve performance. The performance will, however, strongly depend on the given feature set. Boosting component-wise base classifiers is very useful in situations where separation using the given features is possible but redundant features need to be sorted out. On the other hand, modeling of feature interactions is not possible. So, if discriminative information cannot be extracted from the given features directly, there is no other way for the training system to do so. Providing a set of features that will work well in combination with all necessary information is hard, since one cannot know beforehand which information will become important during the course of Boosting training. Usually the strategy

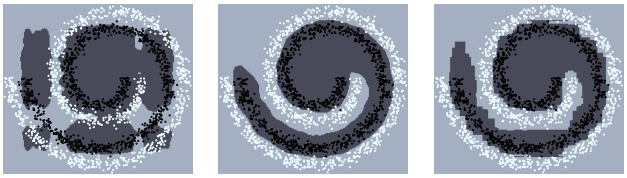

Figure 4.1.: Classification of 2D swirl distribution. Left: Boosted p-Splines working directly on the 2 input dimensions (error rate 23.6%). Center: Boosted p-Splines operating on 8 augmented dimensions (error rate 0.7%). Right: Boosted Decision Trees working on input dimensions (error rate 1.3%). The number of Boosting rounds was fixed to 50 for all classifiers, remaining parameters were tuned to give optimal generalization.

is to provide a large set of features and assume that it will contain the features necessary to give good performance. However, as the number of input dimensions increases, the number of possible features grows exponentially. The size of the augmented feature set is limited by computational resources. Thus, in order to use a finite number of features, applicable feature sets become more and more sparse with respect to all possible features. This, however, increases the risk of missing important information. Prior information can be used to restrict the set of possible features. However, prior information is not always available and its incorporation is non-trivial.

The new approach presented in this work, coined *Boosted Projections* avoids the problems associated with providing features by generating features in each Boosting stage. The features are trained to minimize base classifier costs. This approach is similar to feature selection using Boosting, but instead of providing a set of features to choose from, all features of a given model are considered. The number of possible features is - in principle - infinite. An important assumption, necessary for the approach to work, is that the cost function exhibits a certain degree of smoothness with respect to the features. If this assumption holds, given a starting point, a search for good features in its vicinity can be performed. By using a feature generation scheme, no features - that might become useful later - are ruled out in advance.

4.2.1. General Approach

Boosted Projections build an ensemble of ridge functions:

$$H(\mathbf{x}) = \sum_{t}^{T} h_t(\mathbf{w}_t^\mathsf{T}\mathbf{x}) \quad , \tag{4.1}$$

where $h(.)$ are arbitrary univariate functions. Ridge functions are constant on planes perpendicular to a given projection direction in feature space. Since calculation of projective features $z = \mathbf{w}^\mathsf{T}\mathbf{x}$ involves only one scalar product, feature evaluation is cheap[2].

An ensemble of ridge functions can be seen as a special neural network with one hidden layer, as depicted in Fig. 4.2. In contrast to other neural networks, the transfer function of the hidden layer is adaptive. The connection weights of the output layer depend on the boosting flavor, for GentleBoost they are fixed. Hidden units are added in a stage-wise manner. Therefore only a small number of parameters is trained in parallel. Due to the adaptive transfer functions, the

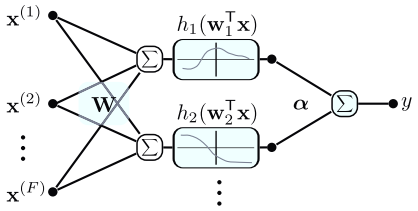

Figure 4.2.: Boosted Projections can be seen as neural networks with one hidden layer. The transfer functions of the hidden units are adaptive. One hidden unit is added in each Boosting stage.

system is very flexible - despite the simple architecture[3]. Building an ensemble of nonlinear functions on linear mappings is a very powerful concept, see discussion in Diaconis and Shahshahani [1984] for approximation properties in regression settings. In principle, depending on the base functions used, arbitrary output functions can be modeled. Note that requirements in classification setting are lower, since only the qualitative output of $H(\mathbf{x})$ needs to be correct.

The base classifiers $h(.)$ may be arbitrary univariate functions. While other univariate smoothers might be used as well (e.g. Parzen windows), p-Splines are very well suited and thus used in this work. Their main advantages are simplicity and cheap evaluation. The use of component-wise smoothing splines in Boosting schemes received a lot of attention recently [e.g. Buhlmann and Yu, 2003; Schmid and Hothorn, 2008; Buehlmann and Hothorn, 2008][4]. Most work

[2]Note that linear scaling of inputs will not influence training. Therefore regularization of data is not as critical as for example with systems based on radial basis functions.

[3]Neural networks with fixed $tansig$-like transfer functions need at least two hidden layers to reach the same level of flexibility.

[4]An implementation for boosting smoothing spline base classifiers is available for the R-

deals with regression settings. Fitting of smoothing splines is more expensive than fitting of stumps or histograms, since a linear system of equation needs to be solved. However, evaluation of smoothing splines is almost as fast as evaluating other base classifiers, like decision stumps or histograms. In addition, the flexibility of smoothing splines can be adapted continuously (see Sect. 2.4.3).

For GentleBoost, base classifier costs are minimized by a weighted least squares fit to the data. For a given univariate base classifier model the costs depend on the feature only. So in order to minimize the cost the best performing feature is sought:

$$\mathbf{w}_t \leftarrow \min_{\mathbf{w}} \left(\epsilon(\mathbf{w})\right) \stackrel{GB}{=} \min_{\mathbf{w}} \left(\sum_{i=1}^{N} c_i (h(\mathbf{w}^\mathsf{T}\mathbf{x}_i) - y_i)^2 \right) \quad . \tag{4.2}$$

For feature selection it is possible to visit all features in order to find the one giving best performance. The same is not possible for Boosted Projections since the number of features is infinite. Therefore, in order to find good features, a gradient descent scheme is used. For any given feature $\mathbf{w}_0$ the gradient of costs with respect to the feature can be calculated as

$$\left.\frac{\partial \epsilon}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_0} = 2 \sum_{i=1}^{N} c_i (h(\mathbf{w}_0^\mathsf{T}\mathbf{x}_i) - y_i) \frac{\partial h(\mathbf{w}_0^\mathsf{T}\mathbf{x}_i)}{\partial \mathbf{w}_0^\mathsf{T}\mathbf{x}_i} \mathbf{x}_i \quad . \tag{4.3}$$

Using this result, an iterative procedure can be set up. The starting point for feature search is initialized randomly. By using alternating steps of fitting and weight update a minimum of the cost function is sought. The procedure is shown in Alg. 12. Any gradient descent scheme may be used. For the experiments presented in Sect. 4.4 Levenberg-Marquardt optimization was used to train feature weights.

In general, the base classifier costs $\epsilon(\mathbf{w})$ cannot be assumed to be a convex function of $\mathbf{w}$. Therefore convergence to a global minimum of equation (4.2) cannot be ensured. On the other hand, Boosting does not depend on finding global minima - as long a the base classifiers do better than random guessing. A possible approach for preventing too weak classifiers from being added to the ensemble is training a number of candidate base classifiers. For each candidate a feature is trained starting from a random initialization. The candidate giving lowest costs is then added to the ensemble. This mainly helps to reduce the number of base classifiers necessary to reach a certain performance level, while the final performance is not affected much. No other measures for enforcing global convergence were used. The richer set of possible features in Boosted

language (*mboost* - model base boosting)

Algorithm 12: Boosted Projections (with GentleBoost)

Input: Training samples $\{\mathbf{x}, y\}_i,\ i = 1, \ldots, N$
Number of boosting rounds T
Smoothing parameter λ
Number of Feature-Initializations: I
Output: Classifier: $H(\mathbf{x}) = \sum_{t=0}^{T} h_t(\mathbf{w}_t^\mathsf{T}\mathbf{x})$

$h_0(\mathbf{x}) = \overline{y}$
for $t = 1, \ldots, T$ **do**
  $c_i \leftarrow e^{-y_i H(\mathbf{x}_i)}$, $c_i \leftarrow c_i / (\sum_{i=1}^{N} c_i)$
  $\epsilon_{min} \leftarrow \infty$
  for $i = 1$ *to* I **do**
    Randomly initialize $\mathbf{w}$
    repeat
      Fit base-classifier $h(\mathbf{wx})$ to $\{\mathbf{w}^\mathsf{T}\mathbf{x}_i, y_i, c_i\}$
      Calculate gradient $\frac{\partial \epsilon}{\partial \mathbf{w}}$ using Eq. (4.3)
      Update weights $\mathbf{w}$ (e.g. using Levenberg Marquardt)
    until *convergence or maximum number of rounds reached*
    $\epsilon \leftarrow \sum_{i=1}^{N} c_i \left(y_i - h(\mathbf{w}^\mathsf{T}\mathbf{x})\right)^2$
    if $\epsilon < \epsilon_{min}$ **then**
      $\epsilon_{min} \leftarrow \epsilon,\ \mathbf{w}_t \leftarrow \mathbf{w}$
    end
  end
  Fit base-classifier $h_t(z)$ to $\{z_i = \mathbf{w}_t^\mathsf{T}\mathbf{x}_i, y_i, c_i\}$
  Add h_t to ensemble
end

Projections is expected to outweigh possible problems associated with it's local convergence. The empirical results of Sect. 4.4 suggest that this assumption is valid. It was not evaluated, whether performance would improve if more sophisticated optimization schemes were used.

The parameters of Boosted Projections, namely the number of boosting rounds T and the smoothness penalty λ need to be set to optimize performance. In regression settings, criteria using training error and model complexity to predict generalization performance exist (e.g. Akaike Information Criterion). To the best knowledge of the author, no equivalent criteria exist for classification settings. Unfortunately, applying regression systems in classification settings leads to poor performance or - at least - slow convergence. Therefore cross validation was used to tune the parameters in the experiments reported in Sect. 4.4.

4.2.2. Extensions

Random subspaces

In Ho [1998] it is proposed to use random subsets of inputs for training base classifiers in an ensemble. Each base classifier is trained in a subspace spanned by a random selection of input features. By using this technique, the base classifiers of an ensemble exhibit less interdependence. Reducing interdependence of base classifiers has been shown to be critical for building strong ensembles[Murua, 2002]. Breiman [Breiman, 2001] combines the random subspace method used on decision trees with bagging to form the popular random forest classifier (see Sect. 2.6.1).

Using random subspaces with Boosted Projections has a number of motivations. The training and evaluation effort is reduced, since feature calculations are performed on less dimensions. Also, by not using all input dimensions, information from features which are otherwise masked by stronger features can be revealed.

Adapting Alg. 12 to use random subspaces can be achieved by selecting a random subspace in line 6 and constraining projection weights of unselected inputs to be zero. The precise number of inputs used to build the subspace is not critical. For experiments with data sets from UCI repository, (Sect. 4.4) the dimensionality of the random subspaces was set to $\lfloor\sqrt{F}\rfloor$, where F is the number of input features.

Extension to Multi-Class Problems

Up till now only binary classification has been discussed. Many real-world problems involve more than two classes. Extensions of Boosting for multi-class problems are discussed in Sect. 3. In the following the GentleBoost.MH scheme (Alg. 10) is used to solve multi-class problems. This scheme uses a 1-vs-all approach to build base classifiers. All binary base classifiers produced in one round of boosting share the same feature. GentleBoost.MH combines simple and cheap training with competitive performance. This helps to reduce the number of features needed to achieve a certain performance.

4.2.3. Examples

Operation of Boosted Projections is visualized using two simple two-dimensional classification problems[5]. Both problems cannot be solved using

[5]Note that feature generation is not crucial for final convergence in two-dimensional problems. Random features would eventually converge to the same results. However, the examples show that important directions are picked first in Boosted Projections and thus training con-

component-wise base classifiers. The first problem is the well known XOR distribution (Fig. 4.3). The samples (200 per class) are drawn from a uniform random distribution in $[0, 1]^2$. Samples with $(\mathbf{x}^{(1)} - 0.5)(\mathbf{x}^{(2)} - 0.5) > 0$ form class 1 all other samples class -1. Boosted Projections find a satisfactory solution using only two base classifiers. The result after ten rounds of boosting shows good generalization, in particular when taking into account the low number of samples. The second problem (Fig. 4.4) is formed by two interleaving spirals. The features added in the first two rounds are good choices for the problem. So the scheme finds a coarse approximation of the class distribution quickly and refines it as the training progresses. The result after 50 rounds visualizes the margin maximization property of Boosting. Boosted Projections reached the optimal error rate of 0.6% after 26 rounds. For comparison: a optimally tuned SVM using a gaussian kernel achieved the same final error rate using 378 support vectors.

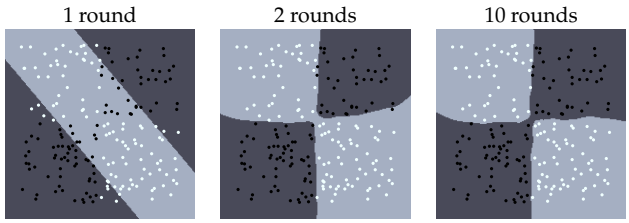

Figure 4.3.: Illustration of Boosted Projection classification - XOR problem

4.2.4. Discussion

Boosted Projections combine a number of advantageous properties. The approach can model almost arbitrary decision surfaces. However, if distribution of classes is simple, Boosted Projections will adapt and build simple models as well. This makes the scheme applicable to a wide range of classification problems. Capacity of the final classifier can be tuned conveniently by adjusting the roughness penalty. This makes the Boosted Projections easy to set up. Table 4.1 shows a summary of properties of the classifiers examined in the experiments of Sect. 4.4. Though the featured properties are just some factors influencing classification performance, they indicate some areas where Boosted Projections have advantages over other classifiers.

The last row of table 4.1 shows a qualitative comparison of the classifiers with respect to training time (including tuning of parameters as necessary).

verges quickly.

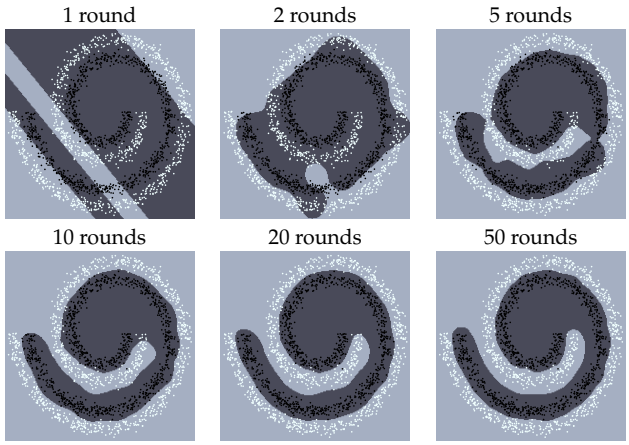

Figure 4.4.: Illustration of Boosted Projection classification - Swirl problem

Property	BP	SVM	BS	BDT	RF
Invulnerability to linear scaling of inputs	+	-	+	+	+
Smooth decision surfaces	+	+	+	-	-
Smooth regularization	+	+	+	-	o
Representation of arbitrary decision surfaces	+	+	-	+	+
Training time (ranks)	5	4	3	2	1

Table 4.1.: Advantages/Disadvantages of classifiers

Random Forest can be trained most efficient, due to its simple tree base classifier and their absence of tunable parameters. Boosted Decision Trees also use tree base classifiers and can thus be trained very fast. Boosted Projections and component-wise smoothing splines both use straightforward MATLAB implementations[6]. For both approaches training time is dominated by spline fitting. The number of iterations for training features in Boosted Projections is usually higher than the number of Inputs for which component-wise splines need to be fitted. Therefore component-wise smoothing splines are the faster of the two. The average training times for Boosted Projections and kernel-SVM are in the same order of magnitude. While the SVM is very fast for small data sets,

[6]Training time should be reduced significantly for an optimized implementation in a compiled language.

Boosted Projections have an advantage on larger data sets[7].

Due to the random feature initialization in Boosted Projections, multiple training runs with same training data and parameters will not produce the same classifiers. For higher numbers of boosting rounds (e.g. $T > 100$) the effect will be negligible. However for ensembles of few base classifiers, random initializations will lead to increased variance of the boosted ensemble and may lead to a deterioration of performance. In these cases, one has to take measures to reduce the effect. One approach is to increase the number of random initializations in Alg. 12. This can prevent overly weak classifiers (e.g. due to poor starting points from random initialization) from being added to the ensemble. Another approach is to use a boosting scheme that removes weak base classifiers automatically (e.g. FloatBoost in Li *et al.* [2002]).

Boosted Projections, as presented in this work, don't regularize projection weights $\mathbf{w}$. It will be an interesting direction for future research to see how the roughness penalty used in Boosted Projections interacts with ridge or lasso weight regularization.

4.3. Related Work

Models using nonlinear functions on linear projections, similar to (4.1), are used in various application areas [see Pinkus, 1993, for a discussion]. In physics they occur in solutions of special types of differential equations and are referred to as planar waves. In computed tomography the base functions $h(\mathbf{w}^\mathsf{T}\mathbf{x})$ are called ridge functions, they emerge naturally as description of the observed measurements. In statistics the same models are used in regression settings under the name of Projection Pursuit Regression. Projection Pursuit Regression was proposed in Friedman and Stuetzle [1981] and was refined and extended in numerous publications [e.g. Roosen and Hastie, 1994; Lingjærde and Liestøl, 1999]. The models $h_t(\mathbf{w}_t^\mathsf{T}\mathbf{x})$ are fitted in a stage-wise forward manner, adjusting every next model to the residuals of preceding models. Various extensions have been proposed mainly focusing on selection of the number of models T and on automatically tuning the amount of smoothing. The main difference of Projection Pursuit Regression and Boosted Projections, is the cost function linking the stages of model fitting. While Projection Pursuit Regression uses residual errors directly, Boosted Projections use a Boosting reweighting scheme. In fact, Boosted Projections scheme using the L_2Boost cost function [Buhlmann and Yu,

[7] Training time of Boosted Projections is dominated by evaluation of spline bases, which scales with $O(N \log(N))$. Optimization to reach $O(N)$ is possible by using quantized input values together with a lookup table.

2003] is very similar to projection pursuit regression[8]. By using Boosting cost functions, Boosted Projections are better suited for classification settings.

The use of a boosting-like feature generation scheme is discussed in García-Pedrajas *et al.* [2007]. Their method, coined *Nonlinear Boosting Projections* (NLBP), provides augmented feature sets $\mathbf{z}_t = \sigma(\mathbf{W}_t^\intercal \mathbf{x})$ (Where $\sigma(.)$ is a nonlinear $tansig$-like transfer function) in each stage. The feature sets are constructed using samples misclassified by preceding stages in order to alleviate classification in the current stage. The resulting nonlinear projection features are then fed into off-the-shelf classifiers to build an ensemble. So, in contrast to Boosting, instead of adjusting weights to grow ensembles, each base-classifier works on a different set of features. The performance of the approach is tested on a wide range of data sets from UCI machine learning repository and is shown to outperform classification ensembles build with other techniques. In contrast to NLBP, the approach proposed in the present work puts emphasis on building compact representations of decision surfaces. While NLBP uses rather complex base classifiers (e.g. ANN or kernel SVM), the presented system goes in the opposite direction. Also NLBP in most cases doesn't tune design parameters of base classifiers, while this is necessary to achieve good performance with Boosted Projections.

Another scheme for feature generation in boosting context named Kullback-Leibler-Boosting is proposed in Liu and Shum [2003]. They use histogram base-classifiers evaluated on linear projective features $z_t = \mathbf{w}_t^\intercal \mathbf{x}$. The features are trained iteratively to maximize the Kullback-Leibler divergence of the base-classifiers. Since their base-classifier is not differentiable, component-wise line search is used to generate features. In high-dimensional feature spaces, component-wise optimization of input features is not feasible. This problem is solved by providing a set of intermediate features $\mathbf{x}^* = f(\mathbf{x})$. These features are selected in order to cover "interesting" directions in feature space. Component-wise optimization is used to combine the intermediate features into the final features. For image classification problems standard feature sets (Haar-, Gabor-wavelets) are used as intermediate features. The approach is shown to achieve good performance for face-detection. However, the performance depends on the intermediate feature set, so prior knowledge of the problem domain is necessary.

In Schwenk and Bengio [2000], neural network base classifiers are used in a boosting procedure. The hidden layers of the neural networks act as feature generators, where each hidden unit builds a derived feature as a nonlinear function of a linear combination of its inputs. The results show that boosting is successful

[8]Though in Projection Pursuit Regression the basis functions are trained to optimize a fitness criterion, which is in general not equivalent to the squared error used in L_2Boost.

not only with decision tree base classifiers, but with "stronger" base-classifiers as well. The performance of boosted neural networks is equivalent and in some cases improves upon performance achieved with boosted decision-trees. The numbers of hidden layers and hidden units need to be adjusted to the training problem.

4.4. Experiments

In order to compare the performance of Boosted Projections with state-of-the-art classifiers, classification performance on 30 datasets from UCI machine learning repository [Asuncion and Newman, 2007][9] was evaluated. The 30 datasets were chosen to represent of wide range of problems with numbers of features ranging between 4 and 69 and numbers of samples between 148 and 20000. The characteristics of the data sets are summarized in Tab. C.1.

To be able to use all classifiers on the same inputs, data sets were preprocessed as follows: (1) categorical inputs with k levels were split into k binary variables (2) Missing values in categorical data are treated as a separate level (3) Missing values in continuous data are replaced by the mean value of the respective attribute (4) Inputs are scaled, so that maximal and minimal values are 0 and 1 respectively. The performance figures were calculated using ten-fold cross validation (10$\times$cv, see Sect. B). The same splits into training and test sets were used for all algorithms. For classification problems with dedicated training- and test-set, only the training set was used in cross validation.

Design parameters of algorithms are tuned to achieve optimal performance. The maximal number of base classifiers was fixed to 1000 for all algorithms building ensembles. However, performance was evaluated for number of base classifiers in range $1, \ldots, 1000$ to find optimal ensemble size. For all boosted classifiers, GentleBoost scheme [Friedman *et al.*, 1998] was used. For multi-class problems the GentleBoost.MH scheme (Alg. 10) was used.

Boosted projections vs. random projections.

To see how performance of Boosted Projections depends on training of projection weights, it is compared against *Random Projections*. Instead of training projection weights, random weights are used with Random Projections. For construction of the classifiers, Alg. 12 - with the maximal number of iterations for feature training set to zero - is used.

Random Projections are used as a bottom-line comparison here, to see whether Boosted Projections benefit from training of projection weights, it

[9] `http://archive.ics.uci.edu/ml/`

should be noted that random projections have some favorable properties. Random projections can be used for subspace induction and theoretic limits for mapping error exist Hegde *et al.* [2008]. In Fradkin and Madigan [2003] random projections are used for subspace induction in machine learning. The results show that random projections perform comparable to principal component analysis for subspace induction.

Boosted Projections with and without training of projection weights were trained to build an ensemble of 1000 base classifiers. The detailed results can be found in Tab. 4.4. Table 4.3 gives a quick summary of qualitative results. Boosted Projections with trained projective features wins 21 - 6. This result clearly shows that performance is improved by training projection weights. On the other hand, performance of Random Projections is still quite competitive[10]. The datasets on which Random Projections did better have a low number of dimensions (≤ 16 for binary and ≤ 10 for continuous data). It seems that using random projections for low dimensional problems can help to reduce the tendency of overfitting, while feature-space is sampled densely enough to allow for good class separation.

Effects of using random subspaces.
In order to evaluate influences on classification, performance results of Boosting projections with and without utilization of random subspaces are compared. The results are shown in Tab. 4.2. In ten, out of thirteen, experiments utilization of random subspaces leads to the best results. Only on two datasets, the use of random subspaces deteriorates classification performance. Both datasets have a relatively low number of input features.

The results indicate, that utilization of random subspaces improves performance of Boosted Projections. Most likely the reason for this behavior is improved resistance to overfitting caused by using random projections. In addition, classifiers trained using random projections use less multiplications, since each feature processes only a subset of the input features.

Boosted Projections vs. state-of-the-art Classifiers
This section compares Boosted Projections to a selection of state-of-the-art classifiers. Classifiers have been chosen to be a good representation for off-the-shelf classification algorithms.

Boosted Projections The scheme is used directly as in Alg. 12. The number of random initializations is fixed to 1. Random subspaces with $\left\lfloor \sqrt{F} \right\rfloor$ inputs are used. Roughness penalty λ is tuned in 11 logarithmic steps.

[10] They outperform all state-of-the-art classifiers in direct comparison.

Dataset	With RS	Without RS
anneal	0.0011	0.0044
autos	0.1355	0.1798
balance-scale	0.0432	0.0112
dermatology	0.0135	0.0243
ecoli	0.1165	0.1225
glass	0.1829	0.2301
iris	0.0267	0.0267
led24	0.2510	0.2560
lymph	0.1010	0.1152
page-blocks	0.0259	0.0254
segmentation-train	0.0524	0.1095
vehicle	0.1724	0.1866
vowel	0.0071	0.0061

Table 4.2.: Performance of Boosted Projections with and without utilization of random subspaces.

Gauss-kernel SVM The *libSvm* Chang and Lin [2001] support vector machine library was used. The C-SVM version with linear slack penalties was used. Kernel width γ and slack penalty C were tuned on a 19×19 grid.

Boosted Component-wise P-Splines (BS) In each stage, P-Splines are fitted to each input to minimize weighted least squares error. The base classifier with lowest costs is added to the ensemble in each round of boosting. The spline roughness penalty λ is adjusted is adjusted in 11 logarithmic steps for regularization.

Boosted Decision Trees (BDT) Decision trees are the most commonly used base classifiers for Boosting. Our implementation uses a greedy tree growing scheme. In each step the node giving best cost-improvement is added to the tree. Leaves have continuous outputs. Nodes are added in a greedy fashion until the specified maximal number is reached. No pruning is performed. Number of nodes is tuned to regularize trees.

Random Forest (RF) The *R-Language* random forest library[11] is used. The number of features visited for training a node is set to be $\left\lfloor \sqrt{F} \right\rfloor$.

Table 4.4 shows the detailed results for all classifiers. Figure 4.5 shows error-error plots of Boosted Projections vs the remaining classifiers to visualize perfor-

[11] `http://cran.r-project.org/web/packages/randomForest/index.html`

	SVM	BS	BDT	RDF	BPR
BP	23-6-1	27-3-0	23-4-3	28-2-0	21-6-3
SVM		16-14-0	15-13-2	18-12-0	11-17-2
BS			10-17-3	17-12-1	9-21-0
BDT				18-9-3	7-21-2
RDF					6-24-0

Table 4.3.: Qualitative comparison of classifier performance (row wins - column wins - draw)

mance. Finally Tab. 4.3 gives a qualitative summary of results: for all combinations of two classifiers, the numbers of wins, losses and draws of the respective classifiers is presented. Note that the differences of performances on one data set are not tested for significant - so even small differences will be counted as wins/losses.

Boosted Projections show good performance in comparison to the state-of-the-art classifiers, winning at least for 23 of 30 data sets. In order to judge overall performance, average ranks of classifiers are calculated. For each data set the classifiers are ranked according to their average error in $10 \times cv$. If classifiers are tied, each of them is assigned the average rank of all classifiers with same performance. The following table shows the average ranks $\overline{r}$ over all data sets for each classifier.

	BP	SVM	BS	BDT	RF
$\overline{r}$	1.56	3.05	3.47	3.12	3.80
$\overline{r} - \overline{r}_{BP}$		1.49	1.91	1.56	2.24

The second line shows the difference of the classifiers average rank and Boosted Projections average rank. The average rank of Boosted Projections is much lower than average ranks of the compared classifiers. The differences of pairs of the remaining classifiers are rather small.

To see whether the observed results are due to chance or if they represent significant differences of performances, a significance test for comparing multiple classifiers over multiple data sets [Demšar, 2006] is performed. Following the discussion of Demšar no tests for significant performance differences on single data sets are performed. The requirements for this type of tests, especially independence of runs, are violated for the utilized 10×cv setup. Therefore results of significance tests on single data sets are to be handled with care, if not meaningless. Furthermore, result of comparisons on one dataset can't be used to predict

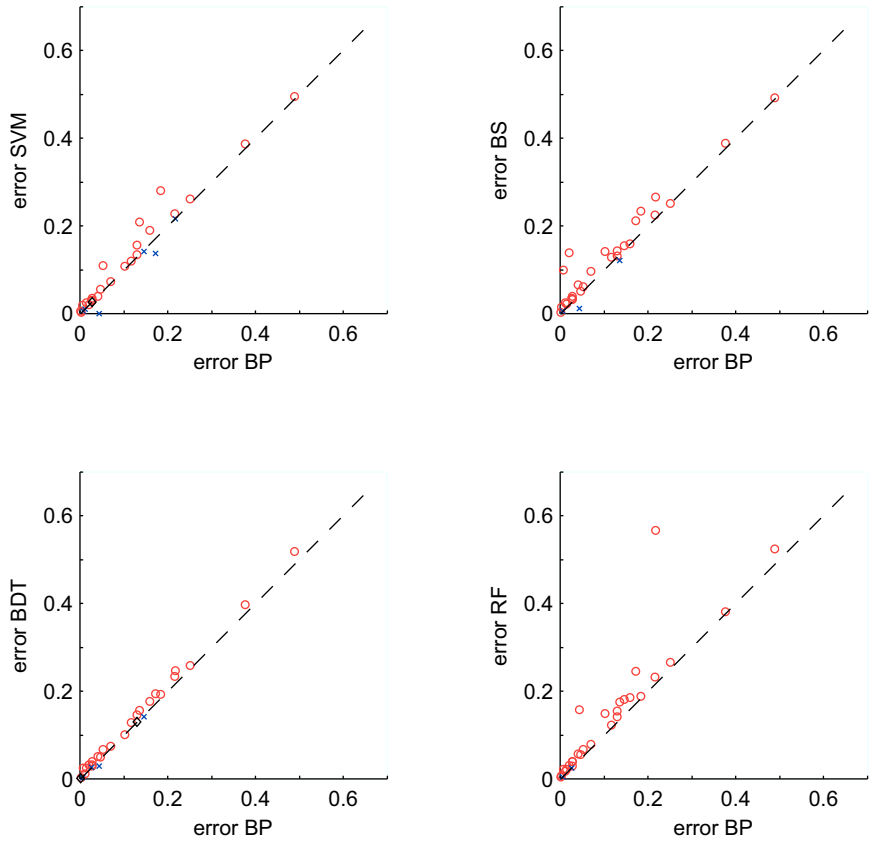

Figure 4.5.: Comparison of classification error rates of Boosted Projections with other state-of-the-art classifiers (o: Boosted Projections win, x: Boosted Projections loose, ♢: draw).

performance on other datasets, unless the classification problems are very similar. For general problems, the best guess for the classifier to use is the classifier giving best performance over a wide range of classification problems.

The null-hypothesis for the test is that all classifiers will perform equally. Thus the average ranks of the classifiers should be roughly equal, which is clearly not the case. To see if any differences are significant, a Friedman-test is performed. The probability of the null-hypothesis to be true for the observed results is 3.4×10^{-4}, therefore it can be rejected with high confidence. Two tailed Nemenyi-post-hoc tests are used to decide which classifiers differ significantly. For the given number of algorithms the difference between two classifiers has to be at least 0.91 for 95% confidence level and 1.53 for 99% confidence level. Thus

Boosted projections is significantly better than SVM on 95% level and for all remaining classifiers on 99% level. No significant differences are found between the SVM, RF, BS and BDT.

On average Boosted Projection needed a relatively low number of 285 Boosting rounds to achieve good performance. SVM, which gave second best performance, used on average 673 support vectors. Calculation of a projective feature should not be more expensive then evaluating the gaussian kernel of the SVM. This shows that Boosted Projections build effective classification ensembles. On the other hand, memory requirements of Boosted Projections are relatively high, since weights of spline bases need to be saved for each base classifier.

4.5. Conclusions

In this chapter the use of Boosted Projections for solving classification problems was investigated. Boosted Projections construct and ensemble of ridge-function base classifiers. For each base classifier, projection weights $\mathbf{w}$ are trained in order to minimize it's costs. Since each feature is trained separately to greedily minimize costs of current base classifier, only a low number of parameters are tuned in parallel. Boosting sample reweighting ensures the trained features to be both descriptive and independent.

The experimental results indicate, that the presented stage-wise feature generation approach is preferable to other schemes, which generate the entire feature set before training of the actual classifier starts. The performance of a classifier depends on all given features. This makes the optimization of a feature set extremely hard. In addition, many schemes for generating feature sets asses the fitness of features using measures only loosely related to the actual classifiers. Finally, ensuring independence of features in a feature set is hard.

By using a stage wise approach, Boosted Projections have a very simple structure. This has a number of advantages. Only two parameters have to be tuned to adjust Boosted Projections to the classification problem at hand. Therefore application is easy. Furthermore, extensions - like e.g. the use of a classifier cascade to improve efficiency- can be incorporated easily.

In the experimental section, Boosted Projections were compared to a set of state-of-the-art classifiers: SVM, Random Forest, Boosted Decision Trees and boosted Smoothing Splines. Performance was evaluated on a selection of 30 data sets from UCI machine learning repository. On average Boosted Projections achieved best performance of all analyzed classifiers. Statistical tests showed that observed differences of classifier performance are, with high probability, significant.

For some classification tasks, the achieved results will crucially depend on the

incorporation of prior knowledge of the problem domain. Though training of features in Boosted Projections helps to minimize the need for prior knowledge, incorporating prior knowledge leads to significant performance improvement in some domains. One example is image classification, where general classification techniques notoriously fail to achieve state-of-the-art performance. The adaption of Boosted Projections to deal with image processing problems is discussed in the next chapter.

A promising path for future extensions is the incorporation of automatic parameter selection schemes. This would remove the computational burden of performing cross validation in order to optimize generalization. In Boosted Projections, each base classifier solved a weighted least squares problem. For regression setting, efficient schemes for parameter selection (e.g. AIC, Sect. 2.1.1) exist. Application of these schemes is not straightforward and should be analyzed in future work.

Another direction of optimization of the Boosted Projections scheme is to reduce the amount of memory needed. For a naive implementation, depending on the number of support points used with each base function, high numbers of base function weights need to be stored. The property, that the number of effective degrees of freedom is significantly lower than the number of support points for penalized fits, can be used to find a more compact representation of the classifier. A possible solution is to use PCA to approximate the outputs of the base classifiers. Research on how this approximation influences the classification performance of Boosted Projections needs to be conducted.

data set	BP	SVM	BS	BDT	RF	BPR
anneal	**0.0011** ± 0.0035	0.0056 ± 0.0094	0.0022 ± 0.0047	**0.0011** ± 0.0035	0.0045 ± 0.0058	0.0011 ± 0.0035
audiology	**0.1593** ± 0.0766	0.1903 ± 0.0700	0.1595 ± 0.0740	0.1773 ± 0.0705	0.1862 ± 0.0695	0.1692 ± 0.1063
autos	0.1355 ± 0.0893	0.2095 ± 0.0940	**0.1217** ± 0.0613	0.1562 ± 0.0688	0.1752 ± 0.0715	0.1619 ± 0.1063
balance-scale	0.0432 ± 0.0302	**0.0000** ± 0.0000	0.0112 ± 0.0131	0.0288 ± 0.0209	0.1583 ± 0.0460	0.0400 ± 0.0243
breast-cancer	0.2170 ± 0.0951	**0.2165** ± 0.0574	0.2656 ± 0.0894	0.2477 ± 0.1014	0.5670 ± 0.1119	0.2304 ± 0.0888
breast-w	**0.0272** ± 0.0157	0.0286 ± 0.0166	0.0343 ± 0.0153	0.0286 ± 0.0202	0.0286 ± 0.0190	*0.0214* ± 0.0168
dermatology	**0.0135** ± 0.0263	0.0243 ± 0.0348	0.0216 ± 0.0279	0.0243 ± 0.0269	0.0217 ± 0.0279	0.0162 ± 0.0261
diabetes	**0.2162** ± 0.0564	0.2279 ± 0.0390	0.2253 ± 0.0506	0.2344 ± 0.0591	0.2318 ± 0.0498	0.2213 ± 0.0601
ecoli	**0.1165** ± 0.0776	0.1194 ± 0.0806	0.1285 ± 0.0769	0.1285 ± 0.0733	0.1226 ± 0.0773	*0.1137* ± 0.0830
glass	**0.1829** ± 0.0921	0.2810 ± 0.0893	0.2336 ± 0.0826	0.1926 ± 0.0864	0.1879 ± 0.0724	0.1905 ± 0.1135
heart-c	0.1452 ± 0.0383	**0.1422** ± 0.0527	0.1552 ± 0.0492	**0.1422** ± 0.0567	0.1817 ± 0.0668	0.2080 ± 0.0522
hepatitis	**0.1300** ± 0.0755	0.1558 ± 0.1005	0.1425 ± 0.0960	**0.1300** ± 0.0977	0.1550 ± 0.0971	0.2450 ± 0.1193
hypothyroid	0.0056 ± 0.0044	0.0186 ± 0.0061	0.0053 ± 0.0028	**0.0024** ± 0.0026	0.0040 ± 0.0034	0.0072 ± 0.0031
ionosphere	**0.0398** ± 0.0332	**0.0398** ± 0.0306	0.0655 ± 0.0447	0.0513 ± 0.0421	0.0569 ± 0.0354	0.0485 ± 0.0505
iris	**0.0267** ± 0.0644	**0.0267** ± 0.0466	0.0400 ± 0.0562	0.0400 ± 0.0644	0.0400 ± 0.0644	0.0267 ± 0.0344
led24	**0.2510** ± 0.0288	0.2620 ± 0.0278	0.2520 ± 0.0329	0.2590 ± 0.0354	0.2660 ± 0.0337	0.2590 ± 0.0264
letters	**0.0199** ± 0.0030	0.0211 ± 0.0024	0.1391 ± 0.0069	0.0316 ± 0.0051	0.0306 ± 0.0042	0.0297 ± 0.0020
lymph	**0.1010** ± 0.0464	0.1086 ± 0.0726	0.1424 ± 0.1086	**0.1010** ± 0.0574	0.1486 ± 0.0817	0.1143 ± 0.0776
optdigits-train	0.0112 ± 0.0030	**0.0099** ± 0.0030	0.0251 ± 0.0079	0.0123 ± 0.0037	0.0173 ± 0.0070	0.0118 ± 0.0038
page-blocks	0.0259 ± 0.0062	0.0307 ± 0.0075	0.0351 ± 0.0073	0.0259 ± 0.0037	**0.0247** ± 0.0068	0.0258 ± 0.0048
pendigits-train	**0.0025** ± 0.0013	0.0027 ± 0.0013	0.0145 ± 0.0047	0.0033 ± 0.0020	0.0073 ± 0.0044	0.0027 ± 0.0015
primary-tumor	**0.4894** ± 0.0897	0.4955 ± 0.0920	0.4924 ± 0.0888	0.5189 ± 0.1124	0.5250 ± 0.0956	0.4955 ± 0.0672
satimage	**0.0699** ± 0.0076	0.0730 ± 0.0078	0.0968 ± 0.0055	0.0751 ± 0.0063	0.0794 ± 0.0070	0.0720 ± 0.0062
segmentation-train	**0.0524** ± 0.0524	0.1095 ± 0.0637	0.0619 ± 0.0504	0.0667 ± 0.0643	0.0667 ± 0.0717	0.0762 ± 0.0681
soybean	**0.0454** ± 0.0176	0.0557 ± 0.0285	0.0513 ± 0.0304	0.0498 ± 0.0261	0.0557 ± 0.0248	0.0483 ± 0.0333
vehicle	0.1724 ± 0.0466	**0.1370** ± 0.0439	0.2114 ± 0.0474	0.1948 ± 0.0472	0.2457 ± 0.0434	0.1856 ± 0.0353
vote	**0.0274** ± 0.0208	0.0344 ± 0.0221	0.0320 ± 0.0267	0.0320 ± 0.0289	0.0390 ± 0.0263	*0.0160* ± 0.0189
vowel	0.0071 ± 0.0096	**0.0061** ± 0.0109	0.0990 ± 0.0362	0.0253 ± 0.0109	0.0212 ± 0.0130	0.0081 ± 0.0064
waveform-5000	**0.1298** ± 0.0119	0.1348 ± 0.0123	0.1320 ± 0.0099	0.1462 ± 0.0133	0.1422 ± 0.0123	*0.1298* ± 0.0117
yeast	**0.3767** ± 0.0401	0.3868 ± 0.0447	0.3882 ± 0.0400	0.3976 ± 0.0521	0.3808 ± 0.0503	*0.3754* ± 0.0334

Table 4.4.: Classification error rates and standard-deviation of error rates on UCI machine learning repository. BP - Boosted Projections, SVM - gaussian kernel Support Vector Machine, BS - boosted component-wise p-Splines, BDT - boosted decision trees, RF random forest, BPR - Boosted Projections without weight optimization. For data sets with names ending with "-train" only dedicated training data were used for cross validation.

5. Boosted Projections for Image Classification

5.1. Motivation

This chapter deals with classification of images of natural scenes. This problem formulation occurs in numerous practical applications. Examples are character recognition, face recognition, automatic detection of defects at production sites, and many more. What makes classification of natural images so challenging is the high number of inputs in relation with the relative low number of samples.

A large number of approaches to classification of image patches exist in the literature. In this section, the extension of the general classification approach Boosted Projections (Sect. 4) to image classification is discussed. Direct application to image classification doesn't give satisfying results. For the construction of Boosted Projections, independency of inputs was assumed. For image classification problems, the inputs usually exhibit significant spatial correlation. Also, separated image pixels carry only a low amount of information. In addition, the high number of inputs could lead to overfitting. Application of random subspaces to reduce the number of features trained in parallel (Sect. 4.2.2) will not improve performance since spatial relationships of inputs are not taken into account.

In the last chapter, no assumptions about the nature of the inputs for a classification problem were made. In this chapter, prior knowledge about images of natural scenes is used to help solving the classification problem and thus get better performance. In Sect. 5.3 regularization of features that can be used to prevent overfitting of trained features is discussed. Section 5.5 introduces a technique for training shift-invariant features in a boosting framework. Utilization of shift-invariance helps to reduce the number of samples needed to solve a given training problem. Experimental performance evaluation is conducted in Sect. 5.6 for the examples handwritten digit recognition and car side view classification.

5.2. Problem Formulation

The classifier is given the task to learn a decision rule $H(\mathbf{x})$ from a set of training samples $\mathbf{x}_i, y_i$ with $i = 1, \ldots, N$. For a given instance $\mathbf{x}$ the classification rule $H(\mathbf{x})$ utters a hypothesis about the corresponding class. The feature vector $\mathbf{x}$ is a vector representation of the image patch $\mathbf{X}$, such that $\mathbf{x} = \text{vect}(\mathbf{X})$. Dimensions of the input patch are denoted with $W \times H$ for patch width and height respectively. Therefore the length of the feature vector $\mathbf{x}$ equals $F = WH$.

In general all features used in this chapter will be local. Local features are restricted to have non-zero feature weights in a local neighborhood of the center pixel of the feature only. For application the use of a rectangular neighborhood of dimension $w \times h$ is common. Two types of image features are discussed in more detail: features performing linear weighting of the input features and features performing a convolution on the input patches. Restriction to local neighborhoods can be done for both feature types.

Linear Features are calculated as a scalar product of the feature-weight vector and the input vector: $f = \mathbf{w}^\mathsf{T}\mathbf{x}$.

Convolutional Features are denoted using matrix formulation $\mathbf{f} = \mathbf{W}\mathbf{x}$. The Output $\mathbf{f}$ is a vector representation of the output of the convolution $\mathbf{F} = \mathbf{X} * \mathbf{W}'$. Wherein $\mathbf{W}'$ is a convolution kernel with dimension $w \times h$. Matrix notation of the convolution is formed using convolution unrolling as depicted in Fig. 5.1. By unrolling the convolution:

$$\mathbf{X} * \mathbf{W}' = \text{reshape}(\mathbf{W}^\mathsf{T}\mathbf{x}) \ , \tag{5.1}$$

notations are simplified. In particular the formulation for back propagation training in Sect. 5.5.1 becomes much clearer. Also parallel implementations of the convolution can be derived from matrix multiplication easily.
Many elements of the convolution matrix $\mathbf{W}$ are constrained to be zero. Also, the weights of the convolution matrix representing a certain weight of the convolution kernel share the same value. Therefore the number of adjustable parameters for each formulation of the convolution is identical to wh. Note that linear features can be seen as a special case of convolutional features, where the input patch and the filter kernel have the same size.

A feature patch $\mathbf{X}$ represents a two-dimensional mapping of a scene. The feature values correspond to measurements on a discrete two-dimensional grid. This grid defines neighborhood relationships between feature values. When the

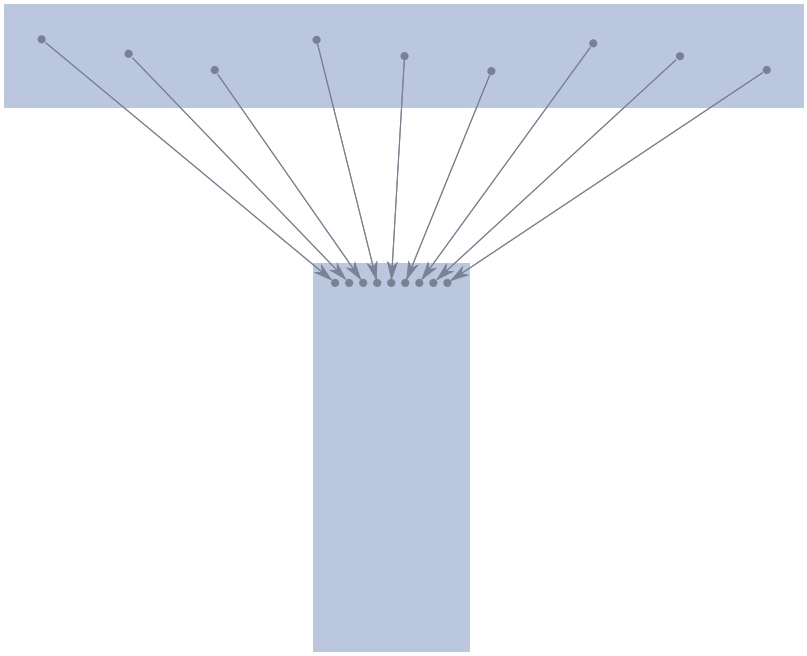

Figure 5.1.: Unrolled convolution. The upper row displays application of a 3×3 filter on a 5×5 patch. Only positions where the filters have full overlap are evaluated. The lower row shows the unrolled convolution in matrix notation.

sampling theorem is respected, neighboring feature values will exhibit a certain level of correlation. In contrast, correlation of feature values with high distance on the grid is low.

In most image classification tasks, the global appearance of objects is subject to strong variation. This may be caused by differing pose, illumination, intra class variance or other factors. Therefore modelling the object appearance using global models is difficult. On the other hand, strong local features exist (e.g. wheels for cars, busses, trucks, etc.). These local features have the advantage, that only coherent informations are combined. Furthermore, the number of features needed to describe a problem sufficiently is reduced by using local features. For example, when there are n_w variants of appearance of wheels in the car class and n_t variants of trunks, $n_w n_t$ global features are needed to describe all possible variants using univariate models[1]. In contrast, parts of objects can be described independently. Therefore only $n_w + n_t$ features are necessary to describe all possible variations in an additive way. Local models also give more flexibility for describing objects. For example, models can be made invariant against small variations of appearance easily by allowing uncertainty in the location of object parts. Local features are a widely adopted and used scheme for building powerful image classification models [e.g. Viola and Jones, 2002b; Hotta, 2002].

From a mathematical point of view, all measures for introducing prior knowledge into image classification problems either help to reduce the effective number of adjustable parameters or reduce the number of training samples needed to build a model. While using local features reduced the number of adjustable parameters directly, regularization techniques reduce the effective number of parameters. This is done by enforcing mathematical relations between feature values. Examples for commonly utilized relations to measure flexibility are smoothness, norm, symmetry of features and others.

5.3. Non Shift-Invariant Features

A very general approach for introducing prior knowledge into feature generation is to apply penalties to certain properties of the trained features. With this, the feature training learns a tradeoff between good class separation and low

[1] It would also be possible to model the appearance using global features, like e.g. principal component analysis (PCA). But with this type of models, all features need to be taken into account together. This work is mainly interested in features that can be used in a stage-wise manner in a boosting framework. Therefore each feature needs to be descriptive on it's own.

feature penalty:

$$f \leftarrow \underset{f \in (F)}{\arg\min} \left(\epsilon(f) + \lambda P(f) \right) \quad , \tag{5.2}$$

where λ adjust the influence of the feature penalization. By introducing the penalty, the number of effective degrees of freedom is reduced. This is especially helpful for image classification tasks, where a high number of adjustable parameters is combined with a relatively low number of samples.

In the following, the use of linear features $f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x}$ is assumed. In regression settings, often the norm of feature weights is penalized: $P(\mathbf{w}) = \|\mathbf{w}\|_d^d$, where d is the order of the utilized norm. Typical values of d are $d = 1$ (Lasso penalty) and $d = 2$ (Ridge penalty). However, this approach can not be applied directly to Boosted Projections, since the smoothing spline base classifiers are invariant against scaling of their inputs. Therefore penalties of the feature norm can be arbitrarily minimized by scaling the feature weights without altering the output of the base classifier.

Another approach uses the smoothness observed in many images of natural scenes [Hastie *et al.*, 1995]. Therefore it is very likely that high-frequency parts of trained features are due to overfitting instead of representing actual object structure. In order to enforce training of features exhibiting low pass character, roughness of the trained features is penalized:

$$P(\mathbf{w}) = \mathbf{w}^\mathsf{T}\mathbf{S}\mathbf{S}^\mathsf{T}\mathbf{w} \quad . \tag{5.3}$$

The matrix $\mathbf{S}$ penalizes the roughness of the trained feature patch and represents an unrolled convolution with Kernel $\mathbf{S}'$. Typically finite differences of second order are penalized using

$$\mathbf{S}' = \frac{1}{8} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad . \tag{5.4}$$

By penalizing second order derivatives, portions of the features up to first derivative are not penalized. This ensures that feature training is still flexible enough to find meaningful features for the problem at hand.

5.4. Shift-Invariant Features

As discussed above, natural images can be advantageously modeled as a combination of object parts. This reduces the variance that needs to be modelled by the individual features, since they only need to cope with local variations.

Many instances of a given image classification problem differ only in small variations of pose ore alignment. Examples are digits, where elastic deformations of examples of the same digit, written by the same subject, occur due to small unconscious oscillations of the hand. Another example are pictures of the same object taken under slightly different viewing angles, leading to affine transformations and occlusion in the produced images. Due to efficiency constraints and overfitting issues, it is not feasible to cover all these variations by individual features.

Shift-invariant features are able to deal with this small variations efficiently. Variations of the input features due to small local transformations will not alter the output of shift-invariant features. Therefore the outputs of one training sample will not only represent that particular instance—but also instances differing only in small local shifts of input features[2]. By that, the requirements to the number of training samples are relaxed, since only coarse poses need to be covered.

5.4.1. Key-Point Detector Based Approaches

These approaches are based on using keypoint detectors to find regions of interest in the image (blob-detection). Examples of keypoint detectors are Difference of Gaussians (DoG), Harris-Affine, Hessian and Maximally Stable Extremal Regions (MSER, Matas *et al.* [2002]). The first three approaches are discussed in Mikolajczyk and Schmid [2004]. A comparison of localization accuracy for all approaches can be found in Haja *et al.* [2008].

Keypoint detectors - depending on which particular is used - are able to identify keypoints independent of scale and affine transformations. Most approaches not only return the position of keypoints, but also their characteristic scale and rotation. This information can be used to build invariant features, representing the appearance of the objects at the keypoint locations.

To build classifiers, each keypoint is described by a feature descriptor. The feature descriptors can be used to find point correspondences between two images of the same object thereby allowing to draw conclusions about the change of pose of the depicted object. The feature descriptors can also be used to asses similarity between two different objects. Using this technique, example based classifiers can be build. The most popular feature descriptor is SIFT (Scale Invariant Feature Transform, Lowe [1999]). It's design together with the used keypoint detector attains invariance against scaling, lighting and affine trans-

[2]With growing level of shift-invariance, spatial relations of features become less important. In the extreme case, features are merely counted without taking their positions into account (see bag of features approach in Sect. 5.4.1)

formations. Other feature descriptors exist (e.g. SURF-Bay *et al.* [2006], Spin Images-Lazebnik *et al.* [2003]). Different approaches for combining the votes of the individual matched features to form a class decision can be found in the literature. Some examples are discussed briefly in the following.

Bags of Features

This approach [Dance *et al.*, 2004] completely ignores information about the spatial organization of the matched descriptors. Therefore it is intrinsically invariant against affine transformations. Descriptors found in the training set are clustered to build a set of descriptor prototypes used to represent image content. The feature vector used in this method is a histogram, representing the number of successful matches of descriptors from the current image to the trained descriptor prototypes. Classification can be performed by arbitrary techniques. Bags of Features is in particular helpful for categorization of photographs, where no prior knowledge about position, scale and orientation of object is available.

By completely ignoring spatial layout of features, a significant part of available information remains unused. Therefore hierarchical extensions to the Bags of Features model exist, taking feature positions into account. In Lazebnik *et al.* [2006], assignment of descriptors is done in a spatial pyramid structure. In the lowest level, global matching of descriptors similar to traditional Bag of Features is performed. Each higher level splits the image into smaller partitions. Matching of descriptors is performed separately for each partition, thereby achieving increasingly accurate localization.

Generalized Hough Transformation like Approaches

The approaches of this type [e.g. Mikolajczyk *et al.*, 2006] learn the relative position of invariant descriptor with respect to the center of the corresponding object. Similar to Bag of Features approaches, clustering of the descriptors is performed to build a appropriate representation of object parts. The positions of all descriptors, assigned to one cluster, are used to learn a probability distribution for the respective object part. In classification stage, a successfully mapped descriptor creates a vote for the class of the corresponding cluster center. In addition a spatial vote representing the location of object parts in the training stage is produced. If the accumulated votes for a spatial position exceed a given threshold, a object hypothesis is created for this position. Therefore simultaneous classification and localization is performed. Hierarchical models can be used to represent more complex classes [e.g. Bouchard and Triggs, 2005]. For these, each descriptor generates votes for object parts. By combining the object

part hypotheses, object hypotheses are build. Extensions for performing segmentation or depth reasoning using the trained models are available [e.g. Leibe *et al.*, 2008].

Drawbacks of keypoint-based approaches are the rather high demands they put on the resolution of the depicted objects. For low resolutions keypoint detectors have problems finding meaningful regions of interest. Sampling of descriptors on a equidistant grid are a workaround for low resolution problems. Typically used feature descriptors, however, are rather expensive for low resolution problems, where more efficient representations exist.

5.4.2. Tangent Distance

Tangent distance [e.g. Simard *et al.*, 1998] is an extension for introducing invariance to neighborhood-based classification techniques (e.g. k-Nearest-Neighbors, Radial Basis Networks). In order to achieve invariance against small shifts and local deformations of the depicted objects, neighborhood functions are adapted to the problem. These neighborhood functions are local in feature-space. Transformations likely representing intra class variance are assigned small distances. Transformations that are likely to be significant for class discrimination are assigned high distances. In consequence, the feature space is distorted, so that samples of the same class are mapped close to each other and samples of different classes are mapped further apart. Tangent distance models achieve good results for recognition of handwritten digit. A drawback of the approach is the need for large number of samples for training the neighborhood functions. Alternatively prior knowledge can be incorporated to reduce the required number of samples. While possible intra-class transformations for handwritten digits are well understood, it is hard to find relationships for general problems in cluttered background.

5.4.3. Deformable Models

Deformable models [e.g. Keysers and Gollan, 2007] use elastic transformations to produce allocations between trained class prototypes and instances to be classified. The approach is based on the intuition that allocation between the presented instance and the class prototype of the corresponding class is better than for arbitrary class combinations. The technique was successfully applied for example in handwritten digit recognition or face recognition. In most applications rather simple backgrounds are present and objects are well segmented.

5.4.4. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) where proposed in LeCun *et al.* [1990]. A number of extensions in comparison to standard MLPs (Sect. 2.3) where used to adapt them to image processing tasks:

- Reduction of number of network weights by using local features.
- The same local features are used for the whole image (weight sharing). In effect, the network performs a convolution.
- Subsampling layers achieve invariance against small deformations of the depicted objects.
- Hierarchical structure: complex features are formed by consecutive layers of convolution and subsampling. Lower convolution layers learn simple features. These are combined in the higher layers to form more complex features.

Figure 5.2 shows the structure of Convolutional Neural Networks. The first layer constructs features maps by performing a convolution of the input image patch with each trained convolution kernel. The feature maps are subsampled in the following layer, thereby removing redundancy and achieving invariance against small shifts. The resulting subsampled feature maps are combined additively using a fixed connection scheme, that has to be set manually before training. The resulting feature maps are used as input for another round of convolution and subsampling[3]. The outputs of the second layer of subsampling are fed into a fully connected multi-layer neural network performing the actual classification. Training of network weights (kernel weights and connections weights of the MLP) can be done using the backpropagation algorithm LeCun *et al.* [1998].

The special network architecture of CNNs allows training kernel weights that optimize class discrimination. The trained set of features adapts to the classification problem at hand. In particular, manual selection of a feature set for a given classification problem - which would require a high level of problem understanding - is not necessary. The convolution and subsampling layers can be seen as feature generators. After the weights of the convolution kernels have been trained, the fully connected MLP used for classification can be replaced by other classifiers [e.g. with Support Vector Machines in Huang and LeCun, 2006].

[3]Two rounds of convolution and subsampling is the most common choice. Other numbers could be used as well if suitable for the given image classification problem.

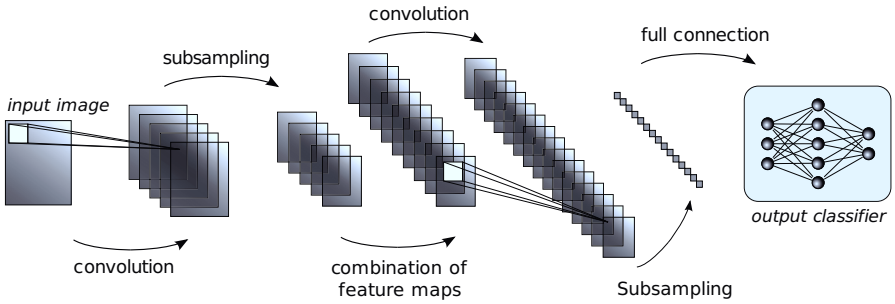

Figure 5.2.: Convolutional Neural Network. For feature training a fully connected MLP is used as output classifier. After features have been learned, arbitrary classifiers can be trained using their outputs.

While CNNs reach good performance for a number of image classification tasks, their design also results in a number of drawbacks. The complex architecture implies a large number of design parameters. Also, implementation of training is non-trivial [Simard *et al.*, 2003]. In Sect. 5.5 an algorithm for training features involving convolution and subsampling is proposed. By using a Boosting framework for training, the number of adjustable parameters can be greatly reduced.

5.4.5. Other Approaches Using Convolutional Features

Features based on convolution and subsampling are used in a number of related image classification approaches as well. While the network architectures are rather similar, the individual approaches on feature training differ. Biologically motivated systems [Serre *et al.*, 2007] use preset features, selected to model the primal visual cortex of mammals. Typically the convolution kernels of the input layer recreate oriented Gabor filters. The following layers use subsampling and radial basis functions to create an invariant representation of the input patches. The output is classified using off-the-shelf linear classifiers. The technique improves upon state of the art techniques on a wide range of image classification problems. In particular the system applied to problems involving many output classes and low numbers of samples. An advantage of the approach is the reduced number of kernel weights to be trained since the same fixed architecture is used for arbitrary classification problems. Therefore only a low number of design parameters need to be set. On the other hand, the architecture may be oversized for simple classification tasks. High numbers of convolution kernels are used in the input layer and a high number of examples need to be stored for the radial basis functions. This makes classification using this technique

computational expensive.

Another direction of using convolution and subsampling to build invariant features are generative approaches. Instead of training the features to optimize class discrimination, the features are trained to build models of the distribution of input samples. The quadratical error between input patches and the reproduction generated by the network is optimized in Ranzato *et al.* [2007]. In Lee *et al.* [2009], the likelihood of the trained model for the given input samples is optimized. The advantage of generative models is their lower susceptibility to overfitting. Also experimental results suggest that an universal set of features for representing natural images exists. Therefore arbitrary input patches - possibly not related to the given problem - can be used for feature training. This relaxes the need for labeled training samples for a given image classification problem. When sparseness constraints are applied during training of the model, the systems learn features representing object parts. The input layer typically learns features representing local edges and corners. Higher layers train more specialized features representing object parts and full objects. A drawback of the approach is that typically features with rather big local support are trained, resulting in expensive evaluation.

5.5. Boosting Shift-Invariant Features

The techniques proposed in this section is closely related to Convolutional Neural Networks. The goal is to keep the advantageous properties of CNNs like shift invariance and good classification performance. On the other hand, by using Boosting, the following improvements are achieved:

- Reduction of the number of design parameters. Most adjustable parameters can be left fixed to default values.
- Improved scaling to suit the problem at hand.
- Simple training.
- Simple architecture.

The proposed approach uses Boosted Projections, discussed in Sect. 4, to train local shift-invariant features. Shift-invariance is achieved using feature combining convolution and subsampling similar to CNNs. In each boosting stage only one convolution kernel is trained. This reduces the number of weights to be adjusted in parallel dramatically. In contrast, all weights in CNNs are trained simultaneously.

In CNNs the convolution operation is performed over the full input patch. In order to reduce the number of computations per feature, only local convolutions are used in the proposed approach[4]. Preliminary experiments suggest, that efficiency with respect to the ratio between error rate and number of multiplications needed can be improved using local features. Local patches are described by patch location $\mathbf{p} = [c_p, r_p, w_p, h_p]$, where c_p and r_p are the column and row of the center of the local patch. The width and height of the local patch are denoted with w_p and h_p respectively. The operator $\mathcal{P}(\mathbf{x}, \mathbf{p})$ is defined to extract patches of geometry $\mathbf{p}$ from feature vector $\mathbf{x}$. To extract discriminative information local-convolution features are used:

$$f(\mathbf{x}) = \text{sub}(\mathcal{P}(\mathbf{x}, \mathbf{p}) * \mathbf{K}) \ , \tag{5.5}$$

where $\mathbf{K}$ is the convolution kernel of size $w \times h$. The subsampling operation sub$(.)$ makes the result invariant to small shifts. It returns high values if a feature is present in a local neighborhood and low values if it isn't. For the experiments reported in Sec. 5.6 the subsampling operation sub$(.)$ returns the maximum absolute value of the filter response[5].

Using separated shift-invariant features without feature-interactions[6] leads to a simple architecture. Thereby a number of parameters necessary with CNNs - e.g. number of layers, number of neurons in each layer, etc. - can be dropped. Only the dimensions filter kernels and subsampling areas needs to be fixed. This corresponds to adjusting kernel size and subsampling step width necessary also with CNNs.

By using Boosted Projections (Sect. 4) output classifiers, complex output functions can be realized, despite the absence of hidden layers. Flexibility of the classifier ensemble can be adjusted conveniently by setting the roughness penalty of the smoothing spline base classifiers. Since smoothing splines are differentiable, gradient descent techniques can be used to train features. The specific training procedure is discussed in the following.

[4]Low number of computations of single features are particularly useful together when using boosted cascades of classifiers [Viola and Jones, 2002a].

[5]Note that this subsampling operator is non-differentiable. For the backpropagation training used in Sect. 5.5.1 a differentiable approximation needs to be used.

[6]For an extension allowing for simple feature-interactions, see Sect. 20.

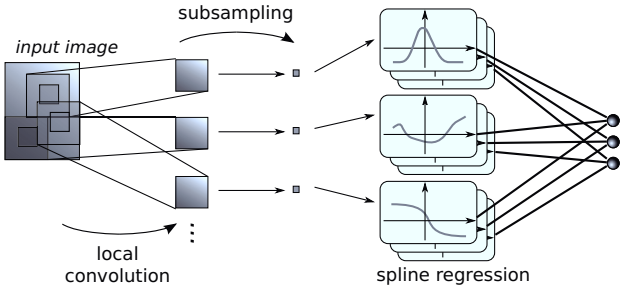

Figure 5.3.: Architecture for boosting shift-invariant features. The convolution kernels of the input layer have only local support on the input patch. The subsampling operation returns one value for each kernel feature. For multi-class problems, GentleBoost.MH (Sect. 3.5.2) is applied. One smoothing-spline base-classifier is trained for each binary subproblem of the 1-vs-all approach.

5.5.1. Training Shift-Invariant Features using Boosted Projections

In order to minimize the GentleBoost cost function, features are trained to optimized weighted squared fitting error:

$$f(\mathbf{x}) \leftarrow \min_{f(\mathbf{x})} \left(\epsilon(h(f(\mathbf{x}))\right) \stackrel{GB}{=} \min_{f(\mathbf{x})} \left(\sum_{i=1}^{N} c_i \left(h(f(\mathbf{x}_i)) - y_i\right)^2 \right) \quad , \tag{5.6}$$

where $h(f(\mathbf{x}))$ is the weighted least square fit to y over $f(\mathbf{x})$ using a penalized smoothing spline basis.

In order to improve clarity of notations, Eq. 5.5 for calculation of shift-invariant features is reformulated. Convolutions are unrolled and expressed as vector-matrix-multiplications. In addition, a differentiable approximation of the $\max \mathrm{abs}$-subsampling operator is used. This leads to

$$\begin{aligned} f(\mathbf{x}) &= f_{\text{sub}}(f_{\text{conv}}(\mathbf{x})) \\ f_{\text{conv}}(\mathbf{x}) &= \mathbf{u} = \mathbf{x}^{\mathsf{T}}\mathbf{L} \\ f_{\text{sub}}(\mathbf{u}) &= v = \|\mathbf{u}\|_{2n} \approx \max \mathbf{abs}(\mathbf{u}) \quad \text{with } n \in \mathbb{N} \ , \end{aligned} \tag{5.7}$$

where $\mathbf{u} \in \mathbb{R}^S$ is the output of the convolution operation in vector notation [7]. The matrix $\mathbf{L}$ represents the unrolled convolution. For approximation of

[7] S can be computed from the size of the input patch $w_p \times h_p$ and the size of the convolution kernel $w \times h$: $S = (w_p - w + 1)(h_p - h + 1)$.

$\max\text{abs}$-sampling an even norm of the convolution result is calculated. For $n \to \infty$ the output of the approximation tends towards the exact result.

Backpropagation Training of Kernel Weights

Calculation of kernel updates can be done comparable to training of neural networks. Backpropagation uses the errors of a layer to calculate the weight gradients for the preceding layer. Starting from the output errors, update for all network weights can be calculated. For the presented shift-invariant features with smoothing spline base classifiers, only the kernel weights $\mathbf{K}'$ need to be trained. The errors of the individual layers are calculated as follows:

Output Layer The error of the output layer can be calculated directly using the error function associated with the used boosting scheme. For GentleBoost the base classifier error is calculated by

$$\epsilon_i = (h(v_i) - y_i)^2 \quad . \tag{5.8}$$

The derivative of the error with respect to the outputs of the subsampling layer is

$$\Delta h_i = \frac{\partial \epsilon_i}{\partial v_i} = 2\,(h(v_i) - y_i)\,\frac{\partial h(v_i)}{\partial v_i} \quad . \tag{5.9}$$

The derivative of the base classifier can readily be calculated by

$$\frac{\partial h(v_i)}{\partial v_i} = \mathbf{a}^\mathsf{T}\mathbf{B}'(v_i) \quad , \tag{5.10}$$

where B' represents the derivative of the spline base $\mathbf{B}(v)$ with respect to v.

Subsampling Layer The derivative of the error in the subsampling layer with respect to the subsampling function can be calculated using the backpropagation rule:

$$\Delta f_{\text{sub}}(\mathbf{u}_i) = \Delta h_i \frac{\partial f_{\text{sub}}(\mathbf{u}_i)}{\partial \mathbf{u}_i^{(j)}} \quad . \tag{5.11}$$

The derivative of the subsampling function $f_{\text{sub}}(\mathbf{u}_i) = \|\mathbf{u}_i\|_{2n}$ is given by

$$\frac{\partial v_i}{\partial \mathbf{u}_i^{(j)}} = \frac{\partial f_{\text{sub}}(u_i)}{\partial \mathbf{u}_i^{(j)}} = \left(\mathbf{u}_i^{(1)} + \ldots + \mathbf{u}_i^{(S)}\right)^{-\frac{2n-1}{2n}} (\mathbf{u}_i^{(j)})^{2n-1} \quad . \tag{5.12}$$

This yields the delta-values $\Delta f_{\text{sub},i}$ for back propagation. Note that for high values of n, all but the strongest feature position are ignored:

$$\frac{\partial v_i}{\partial \mathbf{u}_i^{(j)}} \to \begin{cases} 1 & \text{if } j = \underset{m=1,\ldots,S}{\arg\max}(\mathbf{u}_i^{(m)}) \\ 0 & \text{else} \end{cases} \qquad \text{for } n \to \infty \quad . \tag{5.13}$$

Convolutional Layer The convolution layers holds the kernel weights the are to be updated. The unrolled convolution is split into the outputs associated with each kernel element:

$$f_{\text{conv}}(\mathbf{x}) = \mathbf{x}^{\mathsf{T}}\mathbf{L} = \mathbf{x}^{\mathsf{T}}\left(k_{11}\mathbf{L}_{11} + k_{12}\mathbf{L}_{12} + \ldots + k_{hw}\mathbf{L}_{hw}\right) \quad . \tag{5.14}$$

With the gradient of the individual kernel weights can be calculated by

$$\Delta k_{j_1 j_2} = \Delta f_{\text{sub}}(\mathbf{u}_i)\mathbf{L}^{\mathsf{T}}_{j_1 j_2}\mathbf{x} \tag{5.15}$$

The update of kernel weights can be done for example using the Levenberg-Marquardt technique. The complete scheme for building a classifier with local convolution features is shown in Alg. 13. Training time may be reduced, without deteriorating classification performance, by visiting only a limited number of random positions (line 5). Figure 5.3 shows a trained feature kernel and the associated output function for the example of digit recognition.

Combining Features.
In higher layers of hierarchical networks, basic features are combined to build more complex features [Ranzato *et al.*, 2007; Serre *et al.*, 2007]. This type of feature interaction cannot be modeled by using Alg. 13 directly. It is proposed to, rather than using hierarchical networks, build complex features as linear combinations of local convolution features: $z'_i = \mathbf{v}^{\mathsf{T}}\mathbf{z}_i$, where $\mathbf{z}_i = [f_1(\mathbf{x}_i), f_2(\mathbf{x}_i), \ldots]^{\mathsf{T}}$ represents the values of all convolutional features learned so far and $\mathbf{v}$ are their respective weights. While this approach may not be as powerful as using hierarchical networks, it comes at almost no extra costs.

Algorithm 13 is adapted by feeding linear combinations of features into the base classifier in line 18. The weights of the local convolution features $\mathbf{v}$ are trained to optimize class separation. Typically, only a small number of features, say two or three, need to be combined - depending on the problem at hand. In cases where the maximum number of convolutional features to be used is limited (e.g. due to computational resources), performance may be improved by adding Boosting stages using combinations of the already learned features. Calculation of local convolutional features is much more expensive than evaluation of base classifiers, so costs are negligible. Figure 5.4 shows the adapted architecture for combining filter outputs.

5.6. Experiments

In order to show the competitiveness of the presented approach, experiments on two well-known image classification databases are conducted. The data sets

Algorithm 13: Boosting of local convolution features

Input: Training samples $\{\mathbf{x}, y\}_i,\ i = 1, \ldots, N$
Input: Number of boosting rounds T
Input: Smoothing parameter λ
Input: Feature-geometry

$h_0(\mathbf{x}) = \overline{y}$
for $t = 1, \ldots, T$ **do**
  $c_i \leftarrow e^{-y_i H(\mathbf{x}_i)}$, $c_i \leftarrow c_i / (\sum_{i=1}^{N} c_i)$
  $\epsilon_{min} \leftarrow \infty$
  for *all positions* $\mathbf{p}$ **do**
    Initialize convolution kernel $\mathbf{K} \leftarrow \mathcal{N}(0, 1)$
    repeat
      $z_i = \text{sub}(\mathcal{P}(\mathbf{x}_i, \mathbf{p}) * \mathbf{K})$
      Fit base-classifier $h(z)$ to $\{z_i, y_i, c_i\}$
      Calculate kernel gradient $\Delta\mathbf{K}$ using back-prop
      Update kernel $\mathbf{K}$ (e.g. using Levenberg Marquardt)
    until *convergence or maximum number of rounds reached*
    $\epsilon \leftarrow \sum_{i=1}^{N} c_i \left(y_i - h(\text{sub}(\mathcal{P}(\mathbf{x}, \mathbf{p}) * \mathbf{K}))\right)^2$
    if $\epsilon < \epsilon_{min}$ **then**
      $\epsilon_{min} \leftarrow \epsilon,\ \mathbf{p}_t \leftarrow \mathbf{p},\ \mathbf{K}_t \leftarrow \mathbf{K}$
    end
  end
  Fit base-classifier $h_t(z)$ to $\{z_i, y_i, c_i\},\ z_i = \text{sub}(\mathcal{P}(\mathbf{x}_i, \mathbf{p}_t) * \mathbf{K}_t)$
  Add h_t to ensemble
end

Output: Classifier: $H(\mathbf{x}) = \sum_{t=0}^{T} h_t(\text{sub}(\mathcal{P}(\mathbf{x}, \mathbf{p}_t) * \mathbf{K}_t))$

are selected to have very different properties to illustrate the flexibility of the presented approach.

5.6.1. USPS Handwritten Digit Recognition

The first set of experiments was performed on the USPS handwritten digit recognition corpus. The database contains gray scale images of handwritten digits, normalized to have dimensions 16×16 leading to an input feature vector with 256 values. The training set includes $7,291$ samples, the test set $2,007$. Human error rate on this data set is approximately 2.5% [Simard *et al.*, 1998]. Classifiers specialized to optical character recognition also reach error rates of 2.5% [Keysers *et al.*, 2004]. These classifiers use two main techniques to take into account

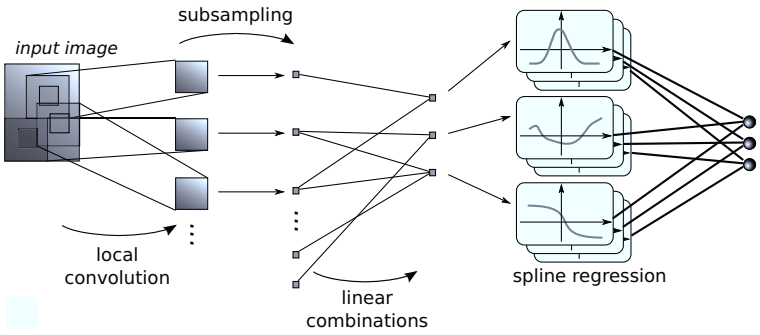

Figure 5.4.: Architecture of boosting shift-invariant features with linear feature combinations.

the special nature of the problem: (1) measures to be invariant against small local shifts are used (2) the training set is extended by distorted patterns in order to increase its variety.

Visual results

Figure 5.5 shows the first 25 features trained for classification of digit five vs digit eight from USPS zip corpus. Examples of the digits are depicted in Fig. 5.8. Convolution kernels of size 5×5 were trained in a 9×9 neighborhood. Many of the trained features correspond to line segments representing parts of the digits in the training set. Figure 5.6 shows an example of a trained feature with the corresponding smoothing spline base-classifier.

Figure 5.7 kernels corresponding to all positions of the 9×9 input patches for classification of digits zero vs one. As expected, the filters corresponding to the outer areas of the input patches focus on detecting properties of the digit zero. Filters corresponding the inner areas of the input patches detect vertical lines corresponding to the digit one. Note that filters differ significantly for different subpatch locations $\mathcal{P}$. Therefore using one kernel over the whole input patch, as used by Convolutional Neural Networks, will not give optimal outputs for all locations. This is one reason to use feature with local support.

Non Shift Invariant Features

In order to show that Boosted Projections (Chap. 4) can be adapted to specific problem domains easily, experiments with USPS handwritten digit corpus are conducted. The database contains images of handwritten digits at 16×16 resolution. Samples are split into a training set containing 7291 images and a test set containing 2007 images. Figure 5.8 shows some examples of training patterns.

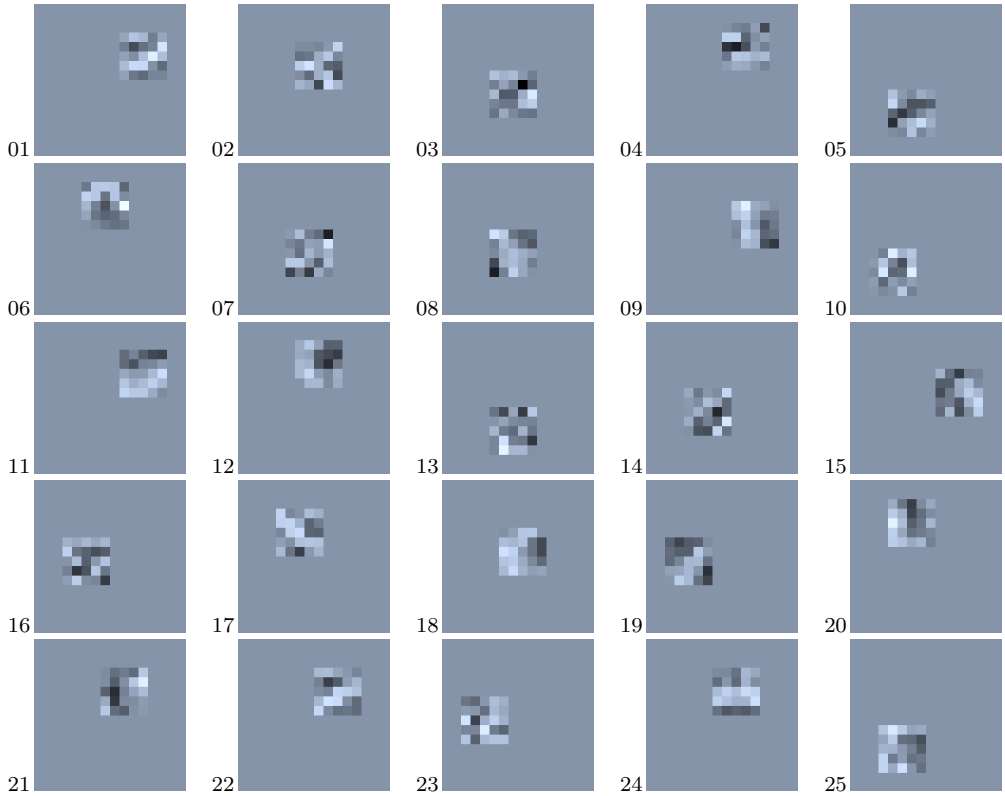

Figure 5.5.: Convolution kernels trained for classification of digits 5 vs 8.

Classifiers strongly tuned to the task of handwritten digit recognition exist. The goal is not to reach performance competitive to these specialized systems, but to show how Boosted Projections compare to off-the-shelf systems. The same classifiers as in the preceding experiments are examined. Classifiers building ensembles run for 500 rounds. Design parameters of classifiers were tuned using cross-validation on training set.

Results can not be expected to be competitive to best specialized classifiers for handwritten digits, since no special adaptions to take into account the specifics of the data set were made.

For the first experiment (*Inputs*) the classifiers work in input pixels directly. Spatial coherence of image pixel was ignored. Error rates are presented in Tab. 5.1. As expected, the achieved performances are relatively poor. SVM achieves the lowest error rate. It should be noted, that SVM uses more than 2300 support vectors and has thus much higher computational costs than the other classifiers.

For the second experiment (*Gabor*) an augmented feature set was generated using the Gabor filters depicted in Fig. 5.9. Outputs of all filters were concatenated resulting in a total of 2048 input features. The features were again used

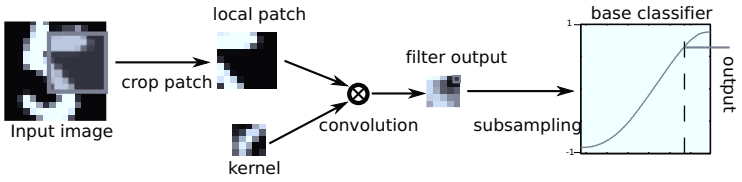

Figure 5.6.: Example of trained pooling feature (for handwritten digits 5 vs 8)

features	BP	SVM	BS	BDT	RF
Inputs	0.0583	0.0458	0.0802	0.0503	0.0606
Gabor	0.0379	0.0432	0.0700	0.0450	0.0541
Trained, local	0.0369	-	-	-	-
Trained, local, smooth	0.0342	-	-	-	-

Table 5.1.: Classification results on USPS data base: on input pixels, Gabor wavelets and trained features

without taking care of spatial coherence. Performance of all classifiers improves. Boosted Projections show the biggest improvement of performance.

Two more experiments were conducted using local, non shift-invariant features produced using Boosted Projections. In the first (*Trained, local*) features of size 5×5 were trained at random positions in the input patch. In the second experiment (*Trained, local, smooth*), the same approach was used, but in addition a roughness penalty on the second derivative of the trained features was used. The results are also shown in 5.1. The yielded performances indicate, that performance is improved by taking into account spatial coherence. Also the resulting classifiers of Boosted Projections are much more efficient than the classifiers using Gabor-filters.

The experiments in this section show that Boosted Projections can be adapted to specific problem domains very easily, since priors on the characteristics of the data to be classified can be incorporated easily. In order to achieve state-of-the-art error rates for handwritten digit recognition, more specific extensions - most importantly local shift-invariance - have to be incorporated.

Shift Invariant Features

Penalized cubic smoothing spline base-classifiers with 100 support points are used to approximate class distributions. Spline roughness penalty, as well as the size of the convolution kernel were determined using cross validation. Kernels of size 5×5 with a subsampling area of 5×5 gave best results - this means each

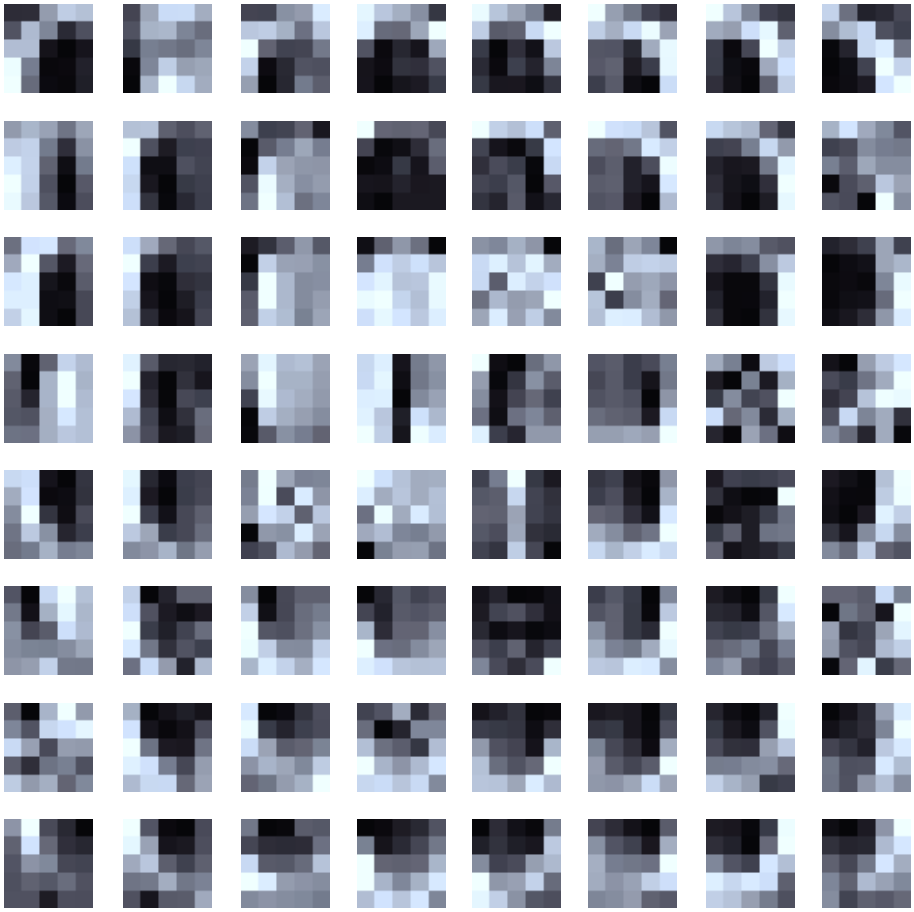

Figure 5.7.: Convolution kernels for classification of digits zero vs one. Candidates in first round of Boosting for all possible positions of the input patches.

pooling feature operates on a 9×9 patch. Pairs of convolutional features are combined to model feature-interactions. An ensemble of 1000 base classifiers was build. Features were added in rounds 1 to 500. The remaining boosting rounds combined already trained local convolutional features.

Experiments using an extended set of training patterns Keysers *et al.* [2004] suggest the original training set is to small to achieve optimal performance. In the literature different techniques are used to extend the training set. An extended training set is built by adding distorted versions of training patterns [Simard *et al.*, 2003], increasing the number of training samples by a factor of five. Note that we did not extend the test set in any way.

Figure 5.10 shows test error with respect to the number of features used. Experiments using the original training set yielded an error rate of 3.1%. On the

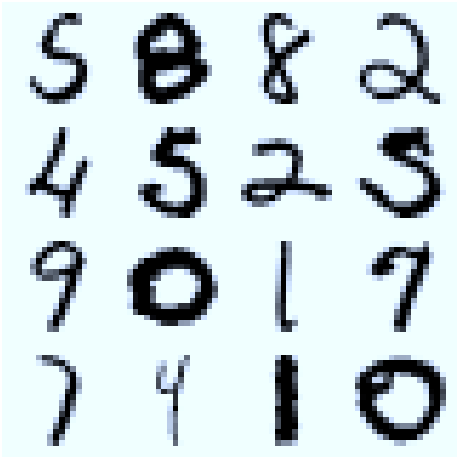

Figure 5.8.: Examples of handwritten digits from USPS zip handwritten digit corpus.

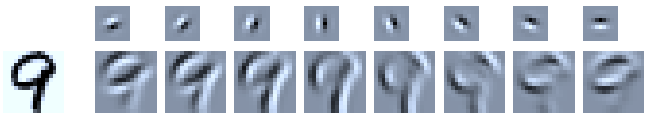

Figure 5.9.: Gabor filters and example of filter outputs.

extended training set an error rate of 2.6% was achieved. Note that the error rate of the extended feature set drops from 3.0% to 2.6% between round 500 and 1000 without adding new convolutional features. Table 5.2 compares the performance of the presented approach to other published results. The results of the presented scheme are competitive to other state-of-the art algorithms.

method	error [%]	error ext. [%]
human [Bottou *et al.*, 1994]	2.5	-
LeNet1 [Cun *et al.*, 1990]	4.2	-
tangent distance [Bottou *et al.*, 1994]	-	2.5
kernel densities [Keysers *et al.*, 2004]	3.3	2.4
this work	3.1	2.6

Table 5.2.: Test error rates on USPS database

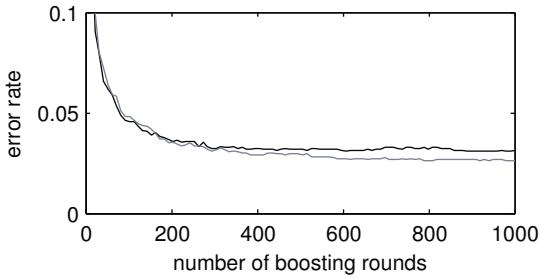

Figure 5.10.: Classification error on USPS depending on the number of boosting rounds. Note that features were trained until round 500. The remaining Boosting rounds add base classifiers combining already calculated features. The black curve shows the results when using the training samples directly, while the gray curve shows the results when using a training set extented with distorted examples.

5.6.2. UIUC Car Classification

A second set of experiments was conducted using the UIUC car side view database Agarwal *et al.* [2004]. The training set contains 550 images of cars and 500 images of background, each image of size 100×40. Again, cross validation was used to find good parameters. The best performance was achieved using convolution kernels of size 5×5 and a subsampling area of size 5×5.

The UIUC car database contains two test sets, both consist of natural images containing cars. The first set consists of 170 images containing 200 cars. The cars in this set have the same scale as the cars in the training set. The second test set consists of 107 images showing 139 cars. The dimensions of the cars range between 89×36 and 212×85.

A sliding window approach was used to generate candidates for the classifier. For multi-scale test images the sliding window classifier was applied to scaled versions of the images. The same scales as in Agarwal *et al.* [2004] ($s = 1.2^{-4,-3,\ldots,1}$) were used. Figure 5.11 shows some classification results on the single scale test set.

Performance evaluation was done in the same fashion as in the original paper Agarwal *et al.* [2004]. Table 5.3 compares the results yielded by the presented approach to state-of-the-art[8]. Results for single and multi-scale test set are among the best reported. In particular, the result produced by boosting shift-invariant features on the multi-scale test set are the best reported results using a sliding

[8]To show the effect of the randomness of our approach the results are given for multiple runs of the system.

method	error (single scale) [%]	error (multi-scale) [%]
Lampert *et al.* [2008]	1.5	1.5
Agarwal *et al.* [2004]	23.5	60.4
Leibe *et al.* [2008]	2.5	5.0
Fritz *et al.* [2005]	11.4	12.2
Mutch and Lowe [2006]	0.04	9.4
this work	(1.25) 1.55 (1.78)	(2.9) 3.6 (4.0)

Table 5.3.: Test error rates on UIUC cars (this work: min, mean, max over ten runs)

window approach.

The error rate with respect to the number of features on the single-scale test set is shown in Fig. 5.12. Errors drop to a competitive level quickly. For an average error of below 2% approximately 30 multiplications per pixel are used, giving a very efficient classifier.

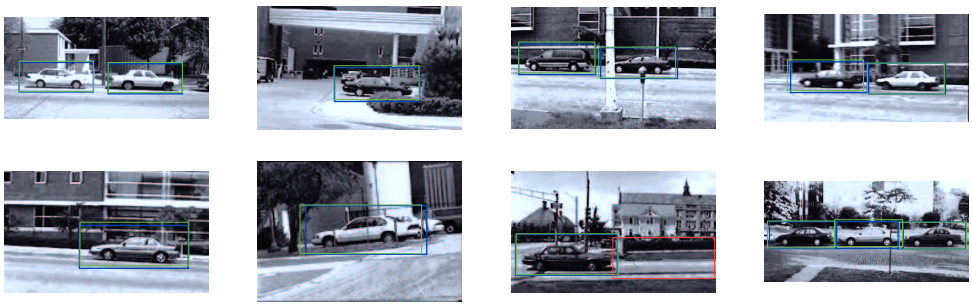

Figure 5.11.: Examples of classification on single-scale test set (ground truth: blue, true positives green, false positives red).

5.7. Discussion

In this chapter a novel approach for generating shift-invariant features was presented. By using Boosting to find meaningful features, the scheme is very simple and scalable. Performance, evaluated on USPS handwritten digit recognition database and UIUC car side views database, is competitive to state-of-the-art systems. The advantage of the presented method, when compared to other systems using similar features, is the low number of design parameters and its

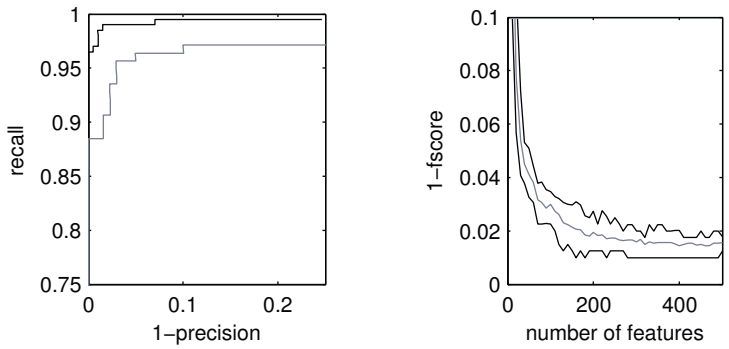

Figure 5.12.: Left: recall-precision curve for UIUC cars (black: single scale, gray: multi scale). Right: f-score on single scale test set (min, mean, max over 10 runs)

modularity. Boosting techniques like the use of cascades, can easily be incorporated.

A possible extension to the proposed scheme is utilization of hierarchical feature structures. In the present work, only the use of one feature-generation layer was discussed. It is, however, known that feature calculation in mammal's visual cortex uses hierarchical feature generation. Simple features are evaluated in the input layers, these simple features are then combined to form more complex features. Using hierarchical feature generation schemes is expected to improve classification performance for some problems. Using hierarchical schemes, however, contradicts the intention of designing a image classification scheme that is simple and scalable. A way to avoid that problem is to use generative feature generation approaches [e.g. Lee *et al.*, 2009] as input layers and use Boosted Projections to build the final classifier.

Another promising extension is to use multiple feature-scales. In the present work, features are generated on one fixed scale. While this is sufficient for classification of handwritten digits and related problems, for real world objects descriptive features will likely appear on multiple scales. In addition to improving classification accuracy, use of multiple scales could also help to improve efficiency by applying features on coarse scales first.

Finally, region-of-interest operators could be utilized to improve the efficiency of the presented scheme. Since regions of interest already are invariant against affine transformations, the costly convolution operation can be replaced by a sampling operation at the given interest points.

6. Conclusion and Outlook

6.1. Contents

In the present work, new techniques for training base classifiers in Boosting context, were proposed. Boosting is a classification approach that is particularly successful in image processing. Reasons for it's success are it's high computational efficiency and good classification results.

The major part of the scientific discussion on the topic of Boosting deals with the design of new Boosting "flavors"-that is, new ways for calculating of sample weights and weights of base classifiers. In the present work another route was taken. Instead of proposing new Boosting flavors, the emphasis was put on designing techniques for optimizing performance of the base classifiers used with Boosting.

The work consists of three main parts. The first part deals with multi-class Boosting techniques. The second part deals with feature generation in Boosting context. The final part discusses the extensions of the technique to image processing problems. The most important contributions of this work are the following:

Comparative Study on Multi-Class Boosting Techniques and Proposal of a New Scheme for Construction of Multi-Class Base Classifiers

A large number of possible techniques for extending Boosting for application to multi-class problems exists. However, many published studies are restricted to comparison of a specific group of approaches [e.g. Sun *et al.*, 2007]. Comparative studies including fundamentally different approaches are rare. In the present work, the following techniques for multi-class boosting were examined:

- Boosting of base classifiers using error-correcting-codes: One binary problem is solved in each stage of Boosting. The binary problems are defined by a code matrix which may be adapted during the course of training. Sample weights for the binary problems are derived from the multi-class sample weights associated with each sample and depend on how the classes are split up to form the binary problems.

- Boosting of base classifiers using one-vs-all approach: C binary problems are solved in each stage of Boosting, where C is the number of classes

involved in the classification problem. Each binary problem separates one class from all other classes. Each sample is associated with C sample weights corresponding to the possible outcomes.

- Boosting of base classifiers using one-vs-one approach: The number of base classifiers in each stage of Boosting depends on how the 1-vs-1 problems are defined. Using the naive approach, $1/2 \cdot C \cdot (C-1)$ base classifiers are necessary, which is clearly prohibitive for large Cs. In the literature, usually base classifiers are constructed using the same approach as for the one-vs-all technique. This is, however, not optimal for solving one-vs-one problems. In the present work, a new approach was proposed (Sect. 3.5.5), where one-vs-one base classifiers are constructed by solving $C-1$ coupled problems.

- In addition to dedicated multi-class Boosting schemes, a number of multi-class wrapper approaches was used for reference:
  - One-vs-All scheme.
  - One-vs-One scheme. Voting and Directed Decision Acyclic Graphs were used for generating the multi-class decision.
  - Error Correcting Output Codes (ECOC).

The different approaches were compared on a selection of 24 datasets from UCI machine learning repository using 10×cross-validation. The error rates show that the proposed approach - based on one vs one base classifiers - gives best performance of the examined techniques. However, the observed performance differences are not significant in a statistical sense. The technique that gave second best results is Boosting with base classifiers using error-correcting-codes. One result of the experiments is that dedicated multi-class schemes for Boosting outperform wrapper approaches. Another result is that performance of an individual scheme is largely affected by the choice of base classifier. For example, Boosting with error-correcting-codes gave good results for problems where the base classifiers were very flexible. On the other hand, it performs poorly using weak base classifiers like decision stumps. The conclusion that can be drawn from the experiments is that the choice of multi-class scheme has to be based on the problem at hand. Consequently, one should select between Boosting with error-correcting-codes and Boosting with one-vs-one base classifiers by using cross validation or similar techniques.

Possible extensions for the proposed multi-class scheme include, for example, the use of hierarchical approaches to deal with problems involving very high numbers of classes. For these problems even linear scaling of the complexity of the classifier with respective to the number of classes may be prohibitive. The

property, that confusion of some class combinations is very unlikely can be used to build clusters of classes. Using a multi-stage approach, the final decision of the classifier can be found by reducing the number of classes in the clusters in each stage. Since the number of clusters to handle in parallel is lower than the total number of involved classes, the complexity of the problem to be solved is reduced.

Also, due to the limited availability of appropriate data sets, feature selection was only simulated[1] in the reported experiments. In future work, a wider range of data sets should be used involving problems where the number of available features is much lower than the number of selected features in order to judge feasibility of the approaches for efficient feature selection. In particular, it should be studied which multi-class Boosting approach generates more efficient classifiers: the use of base classifiers utilizing error-correcting-codes [see. Torralba *et al.*, 2004] or one-vs-one comparisons (see Sect. 3.5.5).

Study of Utilization of Smoothing-Spline Base Classifiers

In most cases, Boosting is used together with decision tree base classifiers. These have the disadvantage, to be susceptible to overfitting. In addition, they produce non-continuous outputs of the classifier confidences in feature-space. This conflicts with the expectation that classes exhibit smooth distributions in feature space for real world problems. The present work uses smoothing spline (p-Splines, see Eilers and Marx [1996]) base classifiers. In contrast to other publications [e.g. Buhlmann and Yu, 2003] , cost functions that are tailored to classification problems are used.

Direct comparison of component-wise base classifiers using decision trees with base classifiers using smoothing splines show that performance of decision trees is slightly better than performance of component-wise smoothing splines (see Tab. 4.3). The observed performance differences are, however, not significant. Smoothing spline base classifiers outperform decision trees for problems where overfitting is an issue. Also the number of inputs for the problems needs to be sufficiently high for smoothing splines to achieve good performance. On datasets with very low numbers of inputs, performance of smoothing spline base classifiers is not competitive to decision trees. An explanation for this result is that for some problems, class separation is not possible using component-wise functions (e.g. XOR problem). In conclusion, component-wise smoothing splines are suitable for problems, involving a rich feature set. That is, the fea-

[1] The used data sets in most cases have a lower number of samples than the number of boosting rounds used in the experiments. Therefore the final classifier typically used all available input features and therefore no true feature selection was performed. The number of base-classifiers was used to estimate the complexity of the boosted ensemble.

ture set should include redundant features combining information from other features. For these problems, base classifiers using smoothing splines perform feature selection efficiently and show good resistance to overfitting.

Training of Linear Features in Boosting Context

As discussed above, component-wise smoothing splines will not show good performance for problems where class separation is not possible on the given input features. In order to solve these problems while maintaining the advantages, a scheme for training linear features, coined *Boosted Projections* was proposed. Derivability of smoothing-spline base classifiers was used to generate features optimizing the Boosting cost function. By using linear projections of the input data, the number of possible features is - in principle - infinite. Applying nonlinear functions to the scalar projections of the data produced in each stage of Boosting, allows for arbitrary class distributions to be modeled. Consequentially, Boosted Projections are not as dependent on the provided set of input features as other approaches[2]. Since only two parameters, namely the roughness penalty of the spline fit and the number of boosting rounds, need to be adjusted, adaption to a given problem is very simple.

Performance of Boosted Projections was evaluated empirically on a selection of 30 data sets from UCI machine learning repository (Sect. 4.4). State-of-the-art classifiers, namely SVM, boosted Decision Trees, boosted component-wise smoothing splines and Random Forests, were used for comparison. Boosted Projections achieved the best performance of the examined approaches. The observed performance difference between Boosted Projections and the other approaches were shown to be significant with a confidence level of 95%. Therefore, Boosted Projections represent a simple scheme fore building classifiers, giving competitive or better performance compared to state-of-the-art classifiers.

Possible future extensions of Boosted Projections include utilization of schemes for automatic selection of design parameters. Using these schemes would remove the need for expensive cross validation for parameter selection. With Boosted Projections, each base classifier solves a weighted least squares problem. For weighted least squares problems in regression settings, powerful schemes for parameter selection exist (e.g. AIC, Sect. 2.1.1). Preliminary experiments on using these approaches in classification settings did not lead to satisfactory performance. In order to build schemes for parameter selection in Boosting context, the interaction between automatic parameter selection and calculation of sample weights needs to be investigated.

Another direction of optimization of the Boosted Projections scheme is to re-

[2] Arbitrary, non-singular, dimensionality preserving projections can be applied to the input data without deteriorating classification performance.

duce the amount of memory needed for storage of the base classifiers. Since the number of effective degrees of freedom of the smoothing splines is lower than the utilized number of base functions, compression schemes can be used to build more compact representations. For example PCA could be used to reduce the number of weights needed to describe the base classifiers.

Training of Shift-Invariant Features in Boosting Context

Applying Boosted Projections to image processing problems directly, doesn't lead to competitive performance. The reasons are that the assumptions that were made during the design of Boosted Projections don't hold for image classification problems. For example, inputs of image classification problems, corresponding to neighboring pixels, show strong correlation, while Boosted Projections assumes the inputs to be independent. In addition, most image classification schemes use specialized techniques, that take into account invariances of natural images. By using these techniques, better utilization of available training data is possible.

In order to improve the applicability of Boosted Projections for image classification problems, an extension for training shift-invariant features was proposed. Shift-invariance is achieved by using features that perform a combination of convolution and subsampling operations. To be able to train these features, Boosted Projections were extended for training of non-linear features. Training of non-linear features is performed using the back-propagation algorithm. Consequently, all features that can be described by artificial neural network-like architectures can be used, leading to a very wide range of possible feature types.

In order to evaluate performance of the proposed system, experiments on two publicly available image classification data sets, namely USPS zip digit database and UIUC car side views, were conducted. Boosted Projections with shift-invariant features achieves competitive performance on both data sets. Advantages of the proposed scheme are the low number of design parameters and the flexibility to adjust to the problem at hand.

A possible extension to the proposed scheme is utilization of hierarchical feature structures. Biological evidence shows that hierarchical architectures are used in mammal's visual cortex. Using hierarchical feature generation schemes is expected to improve classification performance for some problems. Using hierarchical schemes, however, contradicts the intention of designing an image classification scheme that is simple and scalable. Incorporation of generative approaches could be used to resolve that problem.

More directions for extensions that were not covered in this work include incorporation of region-of-interest operators, use of multiple feature scales and incorporation of cascade structures to improve computational efficiency.

6.2. Discussion

This work presented extensions for improving the performance of boosting for image classification problems. Due to it's properties, Boosting is a good choice for wide range of classification problems. Most importantly, Boosting is efficient, modular and easily extensible. These properties make it very useful for solving image classification problems. The presented extensions for Boosting help to deal with problems commonly experienced when solving image classification tasks. By placing roughness constraints on the utilized base classifiers, the risk of overfitting is reduced. The use of shift-invariant features helps to improve generalization as well and reduces the amount of necessary training data. Finally, the proposed multi-class schemes reduce complexity by sharing features over the subproblems of the multi-class problem. The proposed extensions are, however, not limited to image classification problems, but may be used with tasks with similar characteristics as well - for example analysis of voice signals. More general, the proposed schemes are applicable to problems with number of inputs in the order of few thousands, especially if the inputs show significant correlation.

Boosted Projections are designed to be a general classification approach without limitations regarding the characteristics of input features. The complexity of the utilized smoothing spline base classifiers is significantly higher than the complexity of decision stumps. On the other hand, complexity of the smoothing splines is significantly lower than complexity of artificial neural networks [see Schwenk and Bengio, 2000]. By adjusting the roughness penalty, flexibility can be adjusted conveniently over a wide range. The good performance in the empirical evaluations (Sect. 4.4) to some degree contradicts the former believe, that Boosting will only produce good results when used with very weak base classifiers, like e.g. decision stumps. From the presented results, the conclusions can be drawn, that Boosting works well with arbitrary base classifiers, as long as they exhibit a certain level of variance and generalization can be adjusted effectively.

The shift-invariant features for image processing, presented in this work, performing a combination of convolution and subsampling should be viewed as one possible choice. Arbitrary feature types may be trained, as long as they can be modeled as neural network architectures. The particular type of feature used strongly depends on the problem at hand and the invariances the features should incorporate.

6.3. Outlook

The techniques, presented in this work, were evaluated on a selection of publicly available data sets and showed promising performance. The data sets were selected to represent a wide range of applications. Anyhow, a more exhaustive study of the presented techniques is necessary to assess the performance against other popular techniques and better understand the reasons for the good performance of the proposed techniques.

Naturally, not all aspects of the proposed technique could be covered by the present work. Two more extensions to Boosted Projections are proposed and briefly discussed here.

Training of Projective Subspaces.
Boosted Projections generate one direction of interest - or one-dimensional projection of the data - per stage. Using the resulting projections to induce reduced subspaces for classification is straightforward, since Boosted Projections focus on projections containing information important for classification. In order to control the number of output dimensions, a penalization scheme can be incorporated, that puts costs on the dimensionality of the subspace spanned by the features learned by Boosted Projections.
Low-dimensional subspaces are used in certain fields of image processing to reduce the requirements for number of training samples and reduce computational requirements. One typical example is face recognition [He *et al.*, 2005] - where usually only few images per object are available.

Multistage Classifiers.
Using Boosting with Cascade structures is a very successful technique [see Viola and Jones, 2002a]. Applying this technique for improving computational efficiency of the presented approaches was not discussed in the present work. It is, however, very likely that this technique will reduce computational costs significantly without deteriorating classification performance.

The author expects Boosted Projections to have an impact in fields where high precision of classification has to meet with high computational efficiency. Due to the modular design, Boosted Projections can be adapted easily to new application areas.

A. Notations

A.1. Formula symbols

formula symbol	description
N	Number of samples
y	Sample class
$\mathbf{y}$	Vector valued sample class in label space (Sect. 3)
$\mathbf{Y}$	Code matrix corresponding to set of vector valued samples classes
$\mathbf{y}_i$	Vector valued sample class of i-th instance
$\mathbf{x}$	Feature vector of a sample
F	Number of input features: $F = \|\mathbf{x}\|$
y_i	Class of i-th instance of a set of samples
$\mathbf{x}_i$	Feature vector of i-th instance of a set of samples
$\mathcal{Y}$	Set of possible sample classes
$\hat{y}$	Sample class hypothesis
$E[expression]$	Expectation value of $expression$
$\mathrm{I}_{(condition)}$	Indicator function. Evaluates to one if $condition$ is true and zero if otherwise.
$H(\mathbf{x})$	Classification rule for inducing class hypothesis $\hat{y}$ from feature vector $\mathbf{x}$.
$h(\mathbf{x})$	Weak classification rule (base classifier) for inducing class hypothesis $\hat{y}$ from feature vector $\mathbf{x}$.
$p(expression)$	Probability of $expression$
λ	Roughness penalty
$\mathbb{R}$	Real numbers
$W \times H$	Dimensions of an image patch in pixels (width $\times$ height).
$w \times h$	Dimensions of a filter kernel (width $\times$ height).

A.2. Utilized syntax

syntax	description
$\mathbf{x}^{(j)}$	j-th element of vector $\mathbf{x}$
$\mathbf{X}^{(j,k)}$	Element at j-th row and k-th column of matrix $\mathbf{X}$
$\mathbf{X}^{(j,:)}$	Vector containing j-th row of $\mathbf{X}$
$\lfloor z \rfloor$	Nearest integer with value smaller than z
n×cv	Cross-validation with n data partitions. For example 10×cv: ten-fold cross validation. See Appendix B.
$\mathbf{y}^{(k_1 k_2)}$	Label vector associated with the pairwise binary classification problem involving classes k_1 and k_2.
$H^{(k_1 k_2)}$	Pairwise classifier associated with the binary classification problem involving classes k_1 and k_2.
vect($\mathbf{X}$)	Transformation of Matrix to vector form by concatenation of columns.

B. Performance Evaluation

For assessing quality of classification algorithms, evaluation of performances plays the most important roll. Other aspects of the algorithms are very important for the practical application of the algorithms as well, for example computational complexity, scalability, complexity of implementation, etc. However, if a new classification algorithm fails to achieve competitive classification performance, it will not be used.

The classification performance is measured by the expected error rate on data that has not been used for training or tuning of the examined classifier, also termed generalization error. The problem is that the generalization error can't be calculated directly. To get a good estimation of generalization error, a large test set is necessary. However, for real-world problems usually the amount of labeled data is limited[1]. Factors limiting the number of samples for a problem are for example the costs associated with data markup or limited availability of instances (e.g. for classification of rare diseases). Therefore the available samples should be used as efficient as possible.

On the other hand, for many problems - e.g. tuning of parameters of a classifier - estimation of quantitative values of generalization error are not necessary. It is more important that qualitative comparison of classifiers is possible. In the following techniques for estimating performance of classifiers on a given set of samples are reviewed.

B.1. Error Estimation on Separated Test Set

One approach to estimation of generalization error splits the data into training and test set. While the training set is used for training of the classifier, including all parameter tuning, the test set is used to calculate the approximation of generalization error. The number of samples in the test set should be as high as possible in order to get a reliable estimation of generalization error. For a fixed number of available samples, there is a trade-off between size of training set and size of test set, since the two sets should not overlap. An advantage of using a separated test set is that training has to run only once. On the other hand, disadvantages are the lack of a feedback about the reliability of the estimated

[1] Synthetical data sets are an exception, where samples can be generated at will.

error rates and the poor utilization of training data. Examples, where separate test sets should be used for estimation of generalization error are synthetic problems, where test data can be generated easily and benchmark experiments on data bases that provide a dedicated test set.

B.2. Error Estimation on Training Data

There are different approaches for estimating generalization error on training data.

Cross-validation. For cross validation the data set is split into a number s of smaller sets of equal size. The performance is than calculated as the average over s folds ($s\times$cv). For each fold, one of the small sets is used to estimate the generalization error, while the classifier is trained using the remaining data sets. The variance of the estimated performance over the s runs can be used to asses the reliability of the estimated generalization performance[2]. Cross validation can be applied to arbitrary classifiers. The drawback is that the training effort is increased by roughly the factor s.

Cross validation can be seen as a technique to improve reliability of error estimation on an seperate test set. Note that techniques exist, where the test sets used in each round of cross-validation may be overlapping [Dietterich, 1998]. In this work, however, cross-validation always relates to using non-overlapping test sets.

Out-of-bag errors. Another approach, that can be used together with Bagging is out-of-bag error estimation. The property, that only a part of the samples[3] is used for training of each base classifier is used to estimate generalization error. For each sample the outputs of the base classifiers that didn't use that particular sample for training are accumulated. The outputs for each sample are compared to the true labels to estimate the performance of the samples. The estimated performance is somewhat pessimistic, since only a part of the classifiers in the ensemble is used to construct the outputs for each of the samples. The big advantage of out-of-bag error estimates is that they can be calculated in parallel with the normal training without additional effort. Unfortunately, out-of-bag error estimates are not available for general classifiers. While samples that are

[2]However, the results should be interpreted with caution, since the training sets are overlapping.

[3]For construction of the base classifiers roughly 37% of samples are not used for training, if the samples are drawn randomly with replacement and equal probability.

not used for training of one base classifier have no results on it's output, this would not be true for example for boosting techniques.

Theoretical limits of generalization error. For many classification techniques upper limits of the generalization error can be derived, using only mild assumptions about the distribution of the data. Similar to out-of-bag error estimates, no additional training effort is necessary. The training error and complexity of the trained classifier are combined to calculate the error estimate. Calculation of the training error is straightforward. The problem lies in the assessment of the complexity of a classifier. Different metrics are in use for that problem. Examples are the VC-dimension [Vapnik and Chervonenkis, 1971] or the effective number of degrees of freedom. Since derivations of theoretical limits of classification error for different classifiers use different assumptions about the underlying data, results for different classifier usually can't be compared.

In practice, theoretical limits of generalization error are utilized especially for deriving classification techniques and to explain their effectiveness. On the other side using these limit's for comparing classifiers hardly ever makes sense.

B.3. Performance Metrics

Different metrics for assessing the performance of classifiers are in use. Here only metrics for assessing the performance of the output classifier are discussed. Note that many classification approaches use other (continuous) cost functions internally. Due to better comparability, the focus here is on performance measures that return scalar values.

Error rate Fraction of samples that are assigned a wrong label by the classifier.

Misclassification costs The samples are assigned costs. The sum of the costs of misclassified samples gives the misclassification costs of a classifier. The utilized costs reflect that some misclassifications may lead to severe consequences, while others will only have mild effects. A common special case is that all samples of a class share the same costs.

Area under ROC curve The ROC (receiver operating characteristic) is used to asses the performance for binary problems. The error rates for both classes are plotted against each other for varying decision thresholds. By that, not only the performance at the working point is shown, but for all possible working points. The AUC (Area Under ROC Curve) summarizes of information of the ROC into one value.

F-score The F-Score is a statistical measure of a tests accuracy. It is defined as the harmonic mean of precision (fraction of correct results with respect to the number of all samples) and recall (fraction of returned samples of the retrieved class with respect to all samples of the retrieved class). The F-score is in particular useful for problem where the a-priori probability of classes is very asymmetric.

In this work the default performance measure is the error rate.

B.4. Significance Tests between Algorithms

In order to compare algorithms, their performance is evaluated on a set of data sets. There are different approaches to assessing whether an algorithm outperforms another.

Statistical Evaluation of Performance on One Dataset. The results of multiple run of the classifiers on one dataset are evaluated [Alpaydin, 1999; Hothorn *et al.*, 2005]. The use of multiple partitions of the training data is highly recommended in Hothorn *et al.* [2005]. Using one division into training and test set may give unreliable results, for example by favoring an overfitted classifier due to the particular selection of samples in the test set. Using multiple random divisions of the training samples, a permutation test can be constructed to test the null hypothesis, that the performance of all algorithms is equivalent.

A problem of using multiple runs of classifiers on one dataset to induce statistical results is that the training problems are not independent due to the overlap of training sets. Therefore statistical results have to be handled with care. Also, if the number of random divisions of the training data is high, the statistical tests may indicate statistical significance of performances with high power, while the average difference in performance is too small to be of practical relevance.

Usually the statistical significance of performance differences is tested on each dataset. The performance of classifiers is than compiled as number of significant win's, loss' and draw's of one algorithm against another [e.g. García-Pedrajas *et al.*, 2007].

Statistical Evaluation of Performance over Multiple Data Sets. A way to overcome the problem of statistical dependence of multiple divisions of the same dataset is to compare performance of classifiers over multiple data sets [Demšar, 2006]. A two stage test procedure is performed. The first stage performs a Friedman rank sum test to check if there is evidence for statistically significant difference of performances of the algorithms. If such evidence is

found, a post-hoc test is used to evaluate between which algorithms statistical significant differences can be proven. Demšar [2006] proposes to use a Nemeny test to perform this second stage.

Note that no tests for significance of performance differences on single data sets is performed. Statistical evaluation of performance over multiple data sets may incorporate cross-validation, but this is not mandatory. Cross-validation or related procedures can be used to get more reliable estimates of error rates. Also, the magnitude of performance difference of two algorithms on one data set will not be evaluated. Instead only the ranks of the algorithms are incorporated.

In performance evaluations conducted in this work, the test procedure proposed by Demšar [2006] is followed, using a Friedman rank sum test followed by a Nemenyi test. Tenfold cross validation is used for improved reliability and also to prevent overfitting to the test set when performing parameter selection.

C. Databases

C.1. Experiments with datasets from UCI repository

The selected 30 datasets from UCI machine learning repository [Asuncion and Newman, 2007] [1] were chosen to represent of wide range of problems with number of features between 4 and 69 and number of samples between 148 and 20000. The characteristics of the data sets are summarized in table C.1.

To be able to use all classifiers on the same inputs, data sets were preprocessed as follows: (1) categorical inputs with k levels were split into k binary variables (2) Missing values in categorical data are treated as a separate level (3) Missing values in continuous data are replaced by the mean value of the respective attribute (4) Inputs are scaled, so that max and min values are 0 and 1 respectively. The performance figures were calculated using ten-fold cross validation ($10 \times cv$). The same splits into training and test sets were used for all algorithms. For classification problems with dedicated training- and test-set, only the training set was used in cross validation.

C.2. UIUC Car Sideviews Database

The database contains images of car side views and was introduced in Agarwal *et al.* [2004]. The dataset contains:

- 1050 training images (550 cars, 500 non-cars). The training images all have a resolution of 100×40 pixels.
- 170 single-scale test images. The set contains 200 individual cars. The scale of the cars is approximately equal to the scale in the training images.
- 108 multi-scale test images. The set contains 139 cars at various scales.

Figure C.1 shows a selection of training and test images. Note that the negative examples, in many cases, don't resample typical background that is relevant for car classification.

[1] `http://archive.ics.uci.edu/ml/`

Figure C.1.: Examples of training and test images from UIUC cars database.

C.3. USPS Zip Digit Database

The United States Postal Service handwritten digits database consists of handwritten digits from mail envelopes. It contains 7291 samples for training and 2007 for testing. The individual digits are cropped and normalized to a size of 16×16 pixels. The input patches have 1000 grayscale levels. Examples of digits from the database are shown in Fig. 5.8.

name	# instances	# classes	attributes	
			cont	cat
aneal	898	5	6	32
audiology	226	24	-	69
autos	205	6	15	10
balance-scale	625	3	4	-
breast-cancer	286	2	-	9
breast-w	699	2	9	-
dermatology	366	6	34	-
diabetes	768	2	8	-
ecoli	336	8	7	-
glass	214	6	9	-
heart-c	303	2	6	7
hepatitis	155	2	6	13
hypothyroid	3772	4	7	21
ionosphere	351	2	34	-
iris	150	3	4	-
led24	1000	10	24	-
letters	20000	26	16	-
lymph	148	4	3	15
optdigits	5620/3823	10	64	-
page-blocks	5473	5	10	-
pendigits	7494/3498	10	16	-
primary-tumor	339	21	0	17
satimage	6435	6	36	-
segmentation	231/2100	7	19	-
soybean	683	2	-	35
vehicle	846	4	18	-
vote	435	2	-	16
vowel	990	11	10	-
waveform-5000	5000	3	40	-
yeast	1484	10	8	-

Table C.1.: Databases from UCI machine learning repository

D. Formulations of Additional Classifiers for MC-Classification in Simplex

In Sect. 3.5.5 a new approach for solving multi-class classification problems using boosted classifiers was derived. The technique is based on the 1-vs-1 approach, where pairwise comparisons of base classifiers are utilized. To optimize these pairwise comparisons efficiently, $C-1$-dimensional class prototypes $\mathbf{y}_k, k = 1, \ldots, C$ are used. The base classifiers are then optimized to minimize weighted squared error (Eq. (3.21)). Here the optimal outputs for neural networks and smoothing splines are derived. Each sample is assigned a weight vector $\mathbf{c}_i \in \mathbb{R}^{C\times 1}$.

D.1. Feed-forward Neural Networks

When using neural networks to solve multi-class classification problems, usually the 1-vs-all scheme is incorporated. Each output unit of the network attempts to solve one 1-vs-all problem. To apply the 1-vs-1 scheme from Sec. 3.5.5, slight modifications to the weight update between the last hidden layer and the outputs are necessary.

The squared error of the i-th sample can be calculated as follows:

$$\epsilon_i = \sum_k c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} (\mathbf{y}_i - \hat{\mathbf{y}}_i) \right)^2 \tag{D.1}$$

$$= (\mathbf{y}_i - \hat{\mathbf{y}}_i)^\mathsf{T} \mathbf{P}_i (\mathbf{y}_i - \hat{\mathbf{y}}_i) \tag{D.2}$$

$$\text{with } \mathbf{P}_i = \sum_k c_i^{(k)} (\mathbf{y}_i - \mathbf{y}_k)(\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \tag{D.3}$$

The output of the neural network with respect to the outputs of the last hidden layer $\mathbf{z}_i$ is

$$h(\mathbf{z}_i) = \hat{\mathbf{y}}_i = g(\mathbf{W}^\mathsf{T} \mathbf{z}_i) \ , \tag{D.4}$$

with the weight matrix $\mathbf{W}$ connecting the units of the last hidden layer with the units of the output layer and the output transfer function $g(.) \equiv \sigma_T(.)$. This gives the following relationship of the output error:

$$\epsilon_i = (\mathbf{y}_i - g(\mathbf{W}^\mathsf{T} \mathbf{z}_i))^\mathsf{T} \mathbf{P}_i (\mathbf{y}_i - g(\mathbf{W}^\mathsf{T} \mathbf{z}_i)) \tag{D.5}$$

The derivative with respect to the element at row r and column c of the weight matrix can be calculated as

$$\begin{aligned}\frac{\partial \epsilon_i}{\partial \mathbf{W}^{(r,c)}} = \sum_{k \in \mathcal{Y}} -2c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} (\mathbf{y}_i - g(\mathbf{W}^\mathsf{T} \mathbf{z}_i)) \right) \\ \times (\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} g'(\mathbf{W}^\mathsf{T} \mathbf{z}_i)) \mathbf{z}_i^{(q)}.\end{aligned} \tag{D.6}$$

This result can directly incorporated to update all network weights using the backpropagation algorithm.

D.2. Smoothing splines

The output of a set of smoothing splines used to solve the multi-class problem is expressed as a weighted sum of basis functions:

$$h(\mathbf{x}) = \mathbf{A}\mathbf{B}(\mathbf{x}) \ , \tag{D.7}$$

with $\mathbf{B}(\mathbf{x})$ returning the values of the spline bases evaluated at $\mathbf{x}$. The matrix $\mathbf{A} \in \mathbb{R}^{C-1 \times b}$, with b being the number of spline bases, builds the output for each dimension of label space. Therefore the objective to be minimized is

$$\min_{\mathbf{A}} h(\mathbf{A}) = \sum_{i,k} c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \mathbf{y}_i - (\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \mathbf{A}\mathbf{B}(\mathbf{x}_i) \right)^2 + \mathcal{P}(\mathbf{A}) \ . \tag{D.8}$$

The function $\mathcal{P}(\mathbf{A})$ returns the value of the roughness penalty. This value is added to the objective to enforce smoothness of the returned splines. The equation is rewritten using $\mathbf{a} = \text{vect}(\mathbf{A})$

$$\min_{\mathbf{a}} h(\mathbf{a}) = \sum_{i,k} c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \mathbf{y}_i - \left(\mathbf{B}(\mathbf{x}_i) \otimes (\mathbf{y}_i - \mathbf{y}_k) \right)^\mathsf{T} \mathbf{a} \right)^2 + \mathcal{P}(\mathbf{a}) \tag{D.9}$$

For improved readability, the short notation $\boldsymbol{\beta}_{ik} = (\mathbf{B}(\mathbf{x}_i) \otimes (\mathbf{y}_i - \mathbf{y}_k)$ is used in the following. Using quadratic penalties $\mathcal{P} = \mathbf{a}^\mathsf{T} \mathbf{D} \mathbf{a}$ and calculating the derivative with respect to $\mathbf{a}$ gives:

$$\frac{\partial h(\mathbf{a})}{\partial \mathbf{a}} = -2 \sum_{i,k} c_i^{(k)} \left((\mathbf{y}_i - \mathbf{y}_k)^\mathsf{T} \mathbf{y}_i - \boldsymbol{\beta}_{ik}^\mathsf{T} \mathbf{a} \right) \boldsymbol{\beta}_{ik} + \mathbf{D}\mathbf{a} \tag{D.10}$$

Setting to zero gives the optimal weighting of the spline bases:

$$\mathbf{a} = \left(\sum_{i,k} c_i^{(k)} \boldsymbol{\beta}_{ik} \boldsymbol{\beta}_{ik}^\mathsf{T} + \mathbf{D} \right)^{-1} \left(\sum_{i,k} c_i^{(k)} \mathbf{p}_k^\mathsf{T} \mathbf{y}_i \boldsymbol{\beta}_{ik} \right) \tag{D.11}$$

Therefore the optimal smoothing spline is defined by

$$h(\mathbf{x}) = \text{reshape}(\mathbf{a}, C - 1 \times b) \mathbf{B}(\mathbf{x}) \ . \tag{D.12}$$

D.3. SVM

SVMs (Sect. 2.2) are not usually used as base-classifiers for Boosting. However, to show the flexibility of the proposed 1-vs-1 classification scheme, the derivation of a formulation for a multi-class SVM, using the same approach is sketched. The following notations are used:

$\mathbf{W}$	Projection matrix: $\mathbf{W} \in \mathbb{R}^{C-1 \times F}$. The class hypotheses are calculated as $\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$.
$\mathbf{p}_m$	Connection between pair of class prototypes $\mathbf{p}_m = (\mathbf{y}_k - \mathbf{y}_l)\ \forall k, l$ with $k < l \leq C$. The order of the index m is arbitrary. The sum of the inner products of all vectors with themselves is equal to the Identity matrix: $a \cdot \sum_k \mathbf{p}_k \mathbf{p}_k^\mathsf{T} = I$, where a is an appropriate normalization factor.
C	Slack weight. Note that slack weights may be assigned on a per-sample base – with that the SVM could be used in a boosting scheme.
ξ_i	Slack penalty associated with the i-th sample.

The primal objective of the new multi-class SVM formulation is:

$$\min_{w,b,\xi} \quad \frac{1}{2} \sum_k \mathbf{p}_k^\mathsf{T} \mathbf{W}^\mathsf{T} \mathbf{W} \mathbf{p}_k + C \sum_i \xi_i \tag{D.13a}$$

$$\text{subject to} \quad \mathbf{p}_k^\mathsf{T} \mathbf{y}_i \mathbf{p}_k^\mathsf{T} \left(\mathbf{W}^\mathsf{T} \mathbf{z}_i + \mathbf{b} \right) \geq 1 - \xi_i \qquad \forall \quad k, i \tag{D.13b}$$

$$\xi_i \geq 0 \qquad \forall \quad i \tag{D.13c}$$

Where $\mathbf{z}_i$ is the representation of $\mathbf{x}_i$ in a kernel induced feature space. To find the dual representation, the Lagrangian of the problem is formulated:

$$\begin{aligned} L(\mathbf{W}, \mathbf{b}, \xi, \alpha) = \ & \frac{1}{2} \sum_k \mathbf{p}_k^\mathsf{T} \mathbf{W}^\mathsf{T} \mathbf{W} \mathbf{p}_k + C \sum_i \xi_i \\ & - \sum_{i,k} \alpha_{ik} \left(\mathbf{y}_i^\mathsf{T} \mathbf{p}_k \mathbf{p}_k^\mathsf{T} \left(\mathbf{W}^\mathsf{T} \mathbf{z}_i + \mathbf{b} \right) - 1 + \xi_i \right) \end{aligned} \tag{D.14}$$

Calculating the derivatives with respect to $\mathbf{W}$, ξ_i, $\mathbf{b}$ and setting to zero gives

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_k \mathbf{W} p_k p_k^\mathsf{T} - \sum_{i,k} \alpha_{i,k} \mathbf{p}_k \mathbf{p}_k^\mathsf{T} \mathbf{y}_i \mathbf{z}_i^\mathsf{T} \equiv \mathbf{0} \tag{D.15a}$$

$$\frac{\partial L}{\partial \xi_i} = C - \sum_k \alpha_{ik} \equiv 0 \tag{D.15b}$$

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i,k} \alpha_{ik} \mathbf{p}_k \mathbf{p}_k^\mathsf{T} \mathbf{y}_i \equiv \mathbf{0} \tag{D.15c}$$

Using the simplification

$$\sum_k \mathbf{W} p_k p_k^\mathsf{T} = d\mathbf{W} \ , \tag{D.16}$$

the weight matrix $\mathbf{W}$ is yielded:

$$\begin{aligned} \mathbf{W} &= \frac{1}{d} \sum_{i,k} \alpha_{ik} (\mathbf{p}_k^\mathsf{T} \mathbf{y}_i) \mathbf{z}_i \mathbf{p}_k^\mathsf{T} = \frac{1}{d} \sum_i \mathbf{z}_i \mathbf{a}_i^\mathsf{T} \\ \text{with} \quad \mathbf{a}_i &= \sum_k \alpha_{ik} \mathbf{p}_k^\mathsf{T} \mathbf{y}_i \mathbf{p}_k \end{aligned} \tag{D.17}$$

substitution into the first sum term of the primal objective gives

$$\begin{aligned} &\frac{1}{2} \sum_k \mathbf{p}_k^\mathsf{T} \mathbf{W}^\mathsf{T} \mathbf{W} \mathbf{p}_k \\ =&\frac{1}{2d^2} \sum_{i,j,k} \mathbf{p}_k^\mathsf{T} \mathbf{a}_i \mathbf{z}_i^\mathsf{T} \mathbf{z}_j \mathbf{a}_j^\mathsf{T} \mathbf{p}_k = \frac{1}{2d} \sum_{i,j} \mathbf{z}_i^\mathsf{T} \mathbf{z}_j \mathbf{a}_i^\mathsf{T} \mathbf{a}_j^\mathsf{T} \end{aligned} \tag{D.18}$$

substitution into the last term gives:

$$\begin{aligned} &\sum_{i,k} \alpha_{ik} \left(\mathbf{y}_i^\mathsf{T} \mathbf{p}_k \mathbf{p}_k^\mathsf{T} \left(\mathbf{W}^\mathsf{T} \mathbf{z}_i + \mathbf{b} \right) - 1 + \xi_i \right) \\ =&\frac{1}{d} \sum_{i,j} \mathbf{a}_i^\mathsf{T} \mathbf{a}_j \mathbf{z}_i^\mathsf{T} \mathbf{z}_j + \sum_{i,k} \alpha_{ik} \left(-1 - \xi_i \right) \end{aligned} \tag{D.19}$$

Putting everything together results in the dual Lagrangian:

$$L(\mathbf{W}, \mathbf{b}, \xi, \alpha) = \sum_{i,k} \alpha_{ik} - \frac{d}{2} \sum_{i,j} \mathbf{a}_i^\mathsf{T} \mathbf{a}_j \mathbf{z}_i^\mathsf{T} \mathbf{z}_j \ , \tag{D.20}$$

which, together with the constraints from above, gives the dual optimization problem:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2d}\sum_{i,j}\mathbf{a}_i^\mathsf{T}\mathbf{a}_j\mathbf{z}_i^\mathsf{T}\mathbf{z}_j - \sum_{i,k}\alpha_{ik} \tag{D.21a}$$

$$\text{subject to} \quad \sum_{i,k}\alpha_{ik}\mathbf{p}_k\mathbf{p}_k^\mathsf{T}\mathbf{y}_i = 0 \tag{D.21b}$$

$$\sum_k \alpha_{ik} \leq C \qquad \forall i \tag{D.21c}$$

$$\alpha_{ik} > 0 \qquad \forall i,k \tag{D.21d}$$

$$\mathbf{a}_i = \sum_k \alpha_{ik}\mathbf{p}_k^\mathsf{T}\mathbf{y}_i\mathbf{p}_k \tag{D.21e}$$

By defining the projection matrix $\mathbf{P}_i$ for each sample as

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{p}_{i,1}^\mathsf{T}\mathbf{y}_i\mathbf{p}_{i,1} & \dots & \mathbf{p}_{i,d}^\mathsf{T}\mathbf{y}_i\mathbf{p}_{i,d} \end{bmatrix} \tag{D.22}$$

with $\mathbf{p}_{i\dots}$ being the relevant projections for the ith sample and using

$$\mathbf{a}_i = \mathbf{P}\boldsymbol{\alpha}_i \qquad \text{or} \qquad \boldsymbol{\alpha}_i = \mathbf{P}_i^{-1}\mathbf{a}_i = \mathbf{Q}_i\mathbf{a}_i \tag{D.23}$$

the α can be removed from the optimization problem:

$$\min_{\mathbf{a}} \quad \frac{1}{2d}\sum_{i,j}\mathbf{a}_i^\mathsf{T}\mathbf{a}_j\mathbf{z}_i^\mathsf{T}\mathbf{z}_j - \sum_i \mathbf{e}^\mathsf{T}\mathbf{P}^{-1}\mathbf{a}_i \tag{D.24a}$$

$$\text{subject to} \quad \sum_i \mathbf{a}_i = \mathbf{0} \tag{D.24b}$$

$$\mathbf{e}^\mathsf{T}\mathbf{Q}_i\mathbf{a}_i \leq C, \quad \mathbf{Q}_i\mathbf{a}_i > \mathbf{0} \qquad \forall i \tag{D.24c}$$

The output function of the multi-class SVM can now be rewritten in dual variables

$$\begin{aligned} f(\mathbf{z}) =& \mathbf{W}^\mathsf{T}\mathbf{z} + \mathbf{b} \\ =& \frac{1}{d}\sum_i \mathbf{a}_i\mathbf{z}_i^\mathsf{T}\mathbf{z} + \mathbf{b} \end{aligned} \tag{D.25}$$

In Order to calculate the bias $\mathbf{b}$ the dual Lagrangian is used:

$$\begin{aligned} L(\mathbf{a}) =& \frac{1}{2d}\sum_{i,j}\mathbf{a}_i^\mathsf{T}\mathbf{a}_j\mathbf{z}_i^\mathsf{T}\mathbf{z}_j - \sum_i \mathbf{e}^\mathsf{T}\mathbf{Q}_i\mathbf{a}_i + \mathbf{b}^\mathsf{T}\sum_i \mathbf{a}_i \\ & - \sum_i \boldsymbol{\lambda}_i^\mathsf{T}\mathbf{Q}_i\mathbf{a}_i - \sum_i \mu_i\left(C - \mathbf{e}^\mathsf{T}\mathbf{Q}_i\mathbf{a}_i\right) \end{aligned} \tag{D.26}$$

The derivative with respect to $\mathbf{a}_i$ is

$$\begin{aligned}\frac{\partial L(\mathbf{a})}{\partial a_i^{(k)}} =& \frac{1}{d}\sum_j a_j^{(k)}\mathbf{z}_i^\mathsf{T}\mathbf{z}_j - \mathbf{e}^\mathsf{T}\mathbf{q}_i^{(:,k)} + b^{(k)} \\ & - \boldsymbol{\lambda}_i^\mathsf{T}\mathbf{q}_i^{(:,k)} + \mu_i\mathbf{e}^\mathsf{T}\mathbf{q_i}^{(:,k)} \equiv 0 \\ \text{with} \qquad & \lambda_i^{(k)} \geq 0, \qquad \mu_i \geq 0 \qquad \forall i,k\end{aligned} \tag{D.27}$$

where $a_i^{(k)}$ is the kth element of vector $\mathbf{a}_i$ and $\mathbf{q}_i^{(:,k)}$ is the respective column of the matrix $\mathbf{Q}_i$. The formulation for the upper and lower bounds of $b_i^{(k)}$ is as follows:

$$b^{(k)} \geq -\frac{1}{d}\sum_j a_j^{(k)}\mathbf{z}_i^\mathsf{T}\mathbf{z}_j + \mathbf{e}^\mathsf{T}\mathbf{q}_i^{(:,k)} - \mu_i\mathbf{e}^\mathsf{T}\mathbf{q}_i^{(:,k)} \tag{D.28}$$

$$\text{for} \qquad \mathbf{q}_i^{(k,:)}\mathbf{a}_i = 0$$

$$b^{(k)} \geq -\frac{1}{d}\sum_j a_j^{(k)}\mathbf{z}_i^\mathsf{T}\mathbf{z}_j + \mathbf{e}^\mathsf{T}\mathbf{q}_i^{(:,k)} + \boldsymbol{\lambda}_i^\mathsf{T}\mathbf{q}_i^{(:,i)} \tag{D.29}$$

$$\text{for} \qquad \mathbf{e}^\mathsf{T}\mathbf{Q}_i\mathbf{a}_i = C$$

With that the solution of the 1-vs-1 multi-class SVM is complete. In preliminary experiments, the formulation resulted in the same error rates as achieved with state-of-the-art SVM formulations [Chang and Lin, 2001] using wrapper approaches to build multi-class SVMs. The new formulation, however, has the disadvantage that the training effort scales quadratical with the number of classes C when using naive implementations. In comparison, the training effort for SVMs using wrapper techniques does not depend on the number of classes involved.

Bibliography

Shivani Agarwal, Aatif Awan, and Dan Roth. Learning to detect objects in images via a sparse, part-based representation. In *IEEE Transactions on Pattern Analysis and Matchine Intelligence*, volume 26, 2004.

Ethem Alpaydin. Combined 5 x 2 cv f test for comparing supervised classification learning algorithms. *Neural computation*, 11(8):1885–1892, November 1999.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Proceedings of the ninth European Conference on Computer Vision*, May 2006.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. *Pattern Recognition, 1994. Vol. 2 - Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, 2:77–82 vol.2, Oct 1994.

G. Bouchard and B. Triggs. Hierarchical part-based visual object categorization. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:710–715 vol. 1, June 2005.

Leo Breiman. Random forests. *Machine Learning*, V45(1):5–32, October 2001.

Peter Buehlmann and Torsten Hothorn. Boosting algorithms: Regularization, prediction and model fitting, Apr 2008.

P. Buhlmann and B. Yu. Boosting with the l2 loss: Regression and classification. *Journal of the American Statistical Association*, 98:324–339, January 2003.

Chih C. Chang and Chih J. Lin. *LIBSVM: a library for support vector machines*, 2001.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. 1 - 3.

Y. Le Cun, B. Boser, J. S Denker, D. Henderson, R. E.. Howard, W. Howard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II (Denver 1989)*, pages 396–404. Morgan Kaufmann, San Mateo, CA, 1990.

Chris Dance, Jutta Willamowski, Lixin Fan, Cedric Bray, and Gabriela Csurka. Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, 2004.

Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Mach. Learn.*, 46(1-3):225–254, 2002.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, January 2006.

Persi Diaconis and Mehrdad Shahshahani. On nonlinear functions of linear combinations. *SIAM Journal on Scientific and Statistical Computing*, 5(1):175–191, 1984.

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

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

P. Eilers and B. Marx. Flexible smoothing with b-splines and penalties, 1996.

Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522, New York, NY, USA, 2003. ACM.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76:817–823, 1981.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting, 1998.

Jerome H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996.

Mario Fritz, Bastian Leibe, Barbara Caputo, and Bernt Schiele. Integrating representative and discriminative models for object category detection. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1363–1370, Washington, DC, USA, 2005. IEEE Computer Society.

N. Garcia-Pedrajas and D. Ortiz-Boyer. Improving multiclass pattern recognition by the combination of two strategies. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(6):1001–1006, June 2006.

Nicolás García-Pedrajas, César García-Osorio, and Colin Fyfe. Nonlinear boosting projections for ensemble construction. *J. Mach. Learn. Res.*, 8:1–33, 2007.

Venkatesan Guruswami and Amit Sahai. Multiclass learning, boosting, and error-correcting codes. In *COLT '99: Proceedings of the twelfth annual conference on Computational learning theory*, pages 145–155, New York, NY, USA, 1999. ACM.

Andreas Haja, Bernd Jähne, and Steffen Abraham. Localization accuracy of region detectors. In *CVPR*, 2008.

T. Hastie, A. Buja, and R. Tibshirani. Penalized discriminant analysis. *Annals of Statistics*, 23:73–102, 1995.

Xiaofei He, Shuicheng Yan, Yuxiao Hu, P. Niyogi, and Hong-Jiang Zhang. Face recognition using laplacianfaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(3):328–340, March 2005.

Chinmay Hegde, Michael Wakin, and Richard Baraniuk. Random projections for manifold learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 641–648. MIT Press, Cambridge, MA, 2008.

Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, Aug 1998.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Torsten Hothorn, Friedrich Leisch, Kurt Hornik, and Achim Zeileis. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14:675–699, 2005.

Kazuhiro Hotta. Object detection method based on local kernels and automatic kernel selection by kullback-leibler divergence. In *WACV '02: Proceedings of the Sixth IEEE Workshop on Applications of Computer Vision*, page 105, Washington, DC, USA, 2002. IEEE Computer Society. 3 - 3.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, Mar 2002.

Fu-Jie Huang and Yann LeCun. Large-scale learning with svm and convolutional nets for generic object categorization. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press, 2006.

Daniel Keysers and Christian Gollan. Deformation models for image recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(8):1422–1435, 2007. Student Member-Deselaers, Thomas and Member-Ney, Hermann.

Daniel Keysers, Wolfgang Macherey, Hermann Ney, and Jï¿½rg Dahmen. Adaptation in statistical pattern recognition using tangent vectors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):269–274, 2004.

Christoph H. Lampert and Matthew B. Blaschko. A multiple kernel learning approach to joint multi-class object detection. In *Proceedings of the 30th DAGM symposium on Pattern Recognition*, pages 31–40, Berlin, Heidelberg, 2008. Springer-Verlag.

C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, pages 1–8, 06 2008. Best paper award.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A sparse texture representation using affine-invariant regions. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:319, 2003.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:2169–2178, 2006.

Y. LeCun, B. Boser, J. S Denker, D. Henderson, R. E.. Howard, W. Howard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II (Denver 1989)*, pages 396–404. Morgan Kaufmann, San Mateo, CA, 1990.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, pages 9–50, London, UK, 1998. Springer-Verlag.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616, New York, NY, USA, 2009. ACM.

Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vision*, 77(1-3):259–289, 2008.

Stan Z. Li, ZhenQiu Zhang, Heung-Yeung Shum, and HongJiang Zhang. Floatboost learning for classification. In *NIPS*, pages 993–1000, 2002.

Ole C. Lingjærde and Knut Liestøl. Generalized projection pursuit regression. *SIAM J. Sci. Comput.*, 20(3):844–857, 1999.

Ce Liu and Hueng-Yeung Shum. Kullback-leibler boosting. *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 1:I–587–I–594 vol.1, June 2003.

David G. Lowe. Object recognition from local scale-invariant features. *Computer Vision, IEEE International Conference on*, 2:1150–1157 vol.2, 1999.

J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conf. (BMVC)*, pages 384–393, 2002.

Krystian Mikolajczyk and Cordelia Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, October 2004.

Krystian Mikolajczyk, Bastian Leibe, and Bernt Schiele. Multiple object class detection with a generative model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006*, volume 1, pages 26– 36, June 2006. 3 - 3.

Alejandro Murua. Upper bounds for error rates of linear combinations of classifiers. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 5, May 2002*, 2002.

J. Mutch and D. G. Lowe. Multiclass object recognition with sparse, localized features. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, 1:11–18, 2006.

T. Phetkaew, B. Kijsirikul, and W. Rivepiboon. Reordering adaptive directed acyclic graphs: an improved algorithm for multiclass support vector machines. *Neural Networks, 2003. Proceedings of the International Joint Conference on*, 2:1605–1610 vol.2, July 2003.

Allan Pinkus. Fundamentality of ridge functions. *J. Approx. Theory*, 75:295–311, 1993.

John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems*, pages 547–553. MIT Press, 2000.

J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

J. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

Lev Reyzin and Robert E. Schapire. How boosting the margin can also boost classifier complexity. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 753–760, New York, NY, USA, 2006. ACM.

Charles B. Roosen and Trevor J. Hastie. Automatic smoothing spline projection pursuit. *Journal of Computational and Graphical Statistics*, 3:235–248, 1994.

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

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In J. L. McClelland, D. E. Rumelhart, others, and eder, editors, *Parallel Distributed Processing: Volume 2: Psychological and Biological Models*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

Robert E. Schapire and Yoram Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

Robert E. Schapire. Using output codes to boost multiclass learning problems. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 313–321, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

Matthias Schmid and Torsten Hothorn. Boosting additive models using component-wise p-splines. *Comput. Stat. Data Anal.*, 53(2):298–311, 2008.

Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural Comput.*, 12(8):1869–1887, 2000.

Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):411–426, March 2007.

Patrice Simard, Yann Lecun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, pages 239–27, London, UK, 1998. Springer-Verlag.

Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 958, Washington, DC, USA, 2003. Microsoft Research, IEEE Computer Society. 1 - 3.

Yijun Sun, Sinisa Todorovic, and Jian Li. Unifying multi-class adaboost algorithms with binary base learners under the margin framework. *Pattern Recogn. Lett.*, 28(5):631–643, 2007.

A. Torralba, K.P. Murphy, and W.T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2:II–762–II–769 Vol.2, June-2 July 2004.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001. 1 - 3.

P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade, 2002. 1 - 3.

Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision - to appear*, 2002. 1 - 3.

Manfred K Warmuth, Karen Glocer, and Gunnar Rätsch. Boosting algorithms for maximizing the soft margin. In *Advances in Neural Information Processing Systems (NIPS'08)*. MIT Press, 2008. In press.

Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics And Its Interface*, 2:349–360, 2009.