Boosting Regarding Multiclass Problems and Probability Estimations

Guenther Eibl

June 3, 2002
Contents

Abstract 1

Introduction 1

1 Fundamentals 6
  1.1 Basics .................................................. 6
    1.1.1 Decision stumps .................................. 6
    1.1.2 Bagging and the basic boosting algorithm ......... 7
    1.1.3 The roots of boosting algorithms ................. 9
  1.2 AdaBoost for two groups .................................. 10
    1.2.1 Description of the algorithm ..................... 10
    1.2.2 Choice of $\alpha_t$ and stopping criterion ........ 11
    1.2.3 Confidence-rated boosting ......................... 13
  1.3 AdaBoost for multiclass problems ....................... 15
    1.3.1 Description of AdaBoost.M1 ...................... 15
    1.3.2 Decrease of learning error for AdaBoost.M1 ....... 15
    1.3.3 AdaBoost.M2 ..................................... 17
    1.3.4 Error-correcting output codes .................... 20
  1.4 Boosting as gradient descent in function space .......... 22
    1.4.1 Gradient descent for numerical optimization ....... 22
    1.4.2 Gradient descent in function space ............... 23
    1.4.3 AdaBoost as gradient descent in function space ... 24
    1.4.4 Convergence results ................................ 27

2 GrPloss 31
  2.1 Generalization of gradient descent for multiclass problems ... 31
    2.1.1 Inner product .................................... 31
    2.1.2 Exponential loss-functions ....................... 32
  2.2 Gradient descent for the exponential pseudo-loss .......... 33
    2.2.1 Gradient descent for exponential loss-functions .... 33
    2.2.2 Choice of $\alpha_t$ and resulting algorithm GrPloss ... 35
    2.2.3 GrPloss for decision stumps ..................... 40

3 BoostMA 43
  3.1 Original ad-hoc derivation: AdaBoost.M1W ................ 43
  3.2 BoostMA ............................................. 45

4 Probability estimation and Postrang 51
  4.1 Problog1 ............................................ 51
  4.2 Problog2 ............................................ 54
CONTENTS

4.3 Generalization and connection to error-correcting codes . . . . . 56
4.4 Postrank . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

5 Experiments 60
5.1 Experimental setup . . . . . . . . . . . . . . . . . . . . . . . . 61
  5.1.1 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
    5.1.1.1 Choice of the 12 datasets . . . . . . . . . . . . . . . 61
    5.1.1.2 Short problem description and summary . . . . . . 61
  5.1.2 Experimental setup and remarks about the implementation 63
    5.1.2.1 Error estimation . . . . . . . . . . . . . . . . . . . . . 63
    5.1.2.2 Implementation of the base classifiers . . . . . . . . 63
    5.1.2.3 Treatment of the rounding problem . . . . . . . . . 64
    5.1.2.4 The stopping criterion . . . . . . . . . . . . . . . . 64
  5.2 Comparison of algorithms . . . . . . . . . . . . . . . . . . . . 66
    5.2.1 Performance of boosting with reweighting . . . . . . . . 66
    5.2.2 Comparing reweighting and resampling . . . . . . . . . . 71
    5.2.3 Results for boosting with resampling . . . . . . . . . . . 72
      5.2.3.1 Comparison of the test error . . . . . . . . . . . . 72
      5.2.3.2 Generalization and overfit . . . . . . . . . . . . . . 76
      5.2.3.3 Comparison with theory . . . . . . . . . . . . . . . 78
    5.2.4 Stopping criterion . . . . . . . . . . . . . . . . . . . . . . 85
    5.2.5 Evolution of the margin and the group proportions 86
  5.3 Assessment of postprocessing methods . . . . . . . . . . . . . . 94
    5.3.1 Assessment of the probability estimation methods . . . . . .94
    5.3.2 Performance with probability voting . . . . . . . . . . . . 95
    5.3.3 Performance of Postrank . . . . . . . . . . . . . . . . . . 98

6 Summary 100
6.1 Common state of the art and active research areas . . . . . . . . 100
6.2 Algorithms derived in this dissertation . . . . . . . . . . . . . 101
  6.2.1 GrPloss . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
  6.2.2 AdaBoost.M1W . . . . . . . . . . . . . . . . . . . . . . . . 102
  6.2.3 BoostMA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
  6.2.4 Problog1, Problog2 and Postrank . . . . . . . . . . . . . . 103
6.3 Results of the experiments . . . . . . . . . . . . . . . . . . . . . 103

List of symbols 105
List of figures 106
List of tables 109
Bibliography 110
Curriculum Vitae 112
Abstract

In this dissertation two new boosting algorithms, called GrPloss and BoostMA, are developed. They are designed for multiclass classification problems with weak confidence-rated base classifiers such as decision stumps. Two new performance measures are introduced, called pseudo-loss error and maxgroup error. It is proved, that GrPloss exponentially decreases an upper bound for the pseudo-loss error and BoostMA exponentially decreases an upper bound for the maxgroup error, respectively. The algorithms were validated experimentally with 12 big multiclass datasets using decision stumps as base classifiers. The performance of GrPloss, BoostMA and the well known algorithm AdaBoost.M2 turned out to be comparable. Additionally a method to provide probability estimates for the assignment confidence was developed. This method showed good results in the experiments.
Introduction

In classification problems objects belong to exactly one of several predefined groups \( g \in G \). The objects are described by properties \( x \in X \), which can be measured. The goal is to find rules, which predict the group membership from the knowledge of the measurements. All the available information is contained in a learning set or training set \( L = \{(x_1, g_1), \ldots, (x_N, g_N)\} \) of identically, independently sampled objects, whose properties and group memberships are known. The aim is to design algorithms, where the computer learns the classification rules from experience.

Typical examples of applications are to predict, if a person has a disease or not from laboratory data, digit and speech recognition or generally pattern recognition.

The classification rules are maps \( H : X \rightarrow G \) which should have a low expected error rate \( E_{(X,G)}[H(x) \neq g] \). The expected error rate is usually approximated by the training error \( \frac{1}{N} \left| \{(x_i, g_i) \in L; H(x_i) \neq g_i\} \right| \), the proportion of wrongly classified objects of the learning set. Since the training error is usually too optimistic the expected error can be approximated better by a test error or by a crossvalidation error rate (see also [20]).

Boosting algorithms are algorithms, which should turn a weak classifier with high error rate (like decision stumps) into a strong classifier with low error rate. Boosting is very young, the most important and best known boosting algorithm, AdaBoost ([10]), was introduced in 1996. This algorithm and most of the theory is designed for two group problems. We focus on multiclass classification problems with decision stumps as weak classifiers. The straightforward multiclass extension, AdaBoost.M1 ([11]), often doesn’t work here, because the decision stumps are too weak. Another multiclass extension, AdaBoost.M2 ([11]), can overcome this problem by introducing the so-called pseudo-loss. The main goal of this work was to design new algorithms for multiclass problems with weak base classifiers.

In sections 1.1 and 1.2 boosting is introduced, section 1.3 describes the common approaches to extend boosting to multiclass problems. Section 1.4 introduces the gradient descent framework [19], which is an important basis for the algorithm GrPloss. Overall chapter 1 can be seen as a summary of known facts, the subsequent three chapters describe the algorithms and methods, which were developed in this thesis.
I propose two new algorithms, GrPloss and BoostMA, and compare them experimentally with AdaBoost.M2 in chapter 5. GrPloss is derived in chapter 2 from a multiclass generalization of the framework in [19] for a special loss function. BoostMA comes from an idea in section 3.1 and is systematically derived in section 3.2.

Another goal is to get additional information by analyzing the output of the boosting algorithm. The goal is not only to assign an object to a group, but also to provide a measure of confidence in this assignment, i.e. to provide estimates for the probability $p(g|x)$, that an object with measurements $x$ belongs to the group $g$. My approach is to apply logistic regressions to the output as described in chapter 4. Postrank of chapter 4.4 is a simple and fast ad-hoc method to improve the voting step, which worked well in preliminary trials. I therefore decided to investigate it experimentally, too.

Chapter 5 contains experiments with 12 benchmark datasets for multiclass problems in order to compare the two new algorithms with AdaBoost.M2 and to assess the quality of the probability estimation.

Finally chapter 6 contains a summary and a critical discussion of the results.
Chapter 1

Fundamentals

1.1 Basics

1.1.1 Decision stumps

Since we use decision stumps as base classifiers we introduce some notation here. A decision stump \( h \) is a classification tree with 2 leaves. For the digit example this means that the decision stump chooses a variable \( j \) and divides the learning set \( \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathbb{X}, g_i \in \mathbb{G}\} \) into two disjoint subsets or leaves \( l_0 \) and \( l_1 \) by the use of this variable.

If \( \mathbb{X}_j \) is ordered then it chooses \( c \in \mathbb{X}_j \) and sets

\[
\begin{align*}
l_0 &= \{(x_i, g_i) \in \mathcal{L} \mid x_{ij} \leq c\} \\
l_1 &= \{(x_i, g_i) \in \mathcal{L} \mid x_{ij} > c\}
\end{align*}
\]

If \( \mathbb{X}_j \) is unordered then it chooses \( S \subseteq \mathbb{X}_j \):

\[
\begin{align*}
l_0 &= \{(x_i, g_i) \in \mathcal{L} \mid x_{ij} \notin S\} \\
l_1 &= \{(x_i, g_i) \in \mathcal{L} \mid x_{ij} \in S\}
\end{align*}
\]

Now we call \( N_k \) the number of elements of leave \( l_k \), \( N_k(g) \) the number of elements of group \( g \) in leave \( l_k \) and the corresponding group proportion \( \pi_k(g) := N_k(g) \setminus N_k \). The decision stump \( h \), given by \((j, c)\) and \((j, S)\) is then defined by

\[
\begin{align*}
h : \mathbb{X} \times \mathbb{G} &\to [0, 1] : \quad h(x, g) := \pi_0(g)[x_j \leq c] + \pi_1(g)[x_j > c] \\
h : \mathbb{X} \times \mathbb{G} &\to [0, 1] : \quad h(x, g) := \pi_0(g)[x_j \notin S] + \pi_1(g)[x_j \in S]
\end{align*}
\]

respectively, where \([\cdot]\) is a function of boolean values with \([true] = 1 \) and \([false] = 0\). Decision stumps are very simple and weak classifiers, for example if one uses them to classify the digitbreiman example (section 5.1.1.2) one only achieves an error rate of about 80\%.
1.1.2 Bagging and the basic boosting algorithm

Bagging is a method which is closely related to boosting. The bagging method draws many bootstrap samples from the training set, gets a base classifier from each bootstrap sample and then combines all base classifiers by unweighted majority voting.

Definitions: Let $M$ be a set with $|M| = N$. A bootstrap sample $B(M)$ is the result of sampling $N$ times from $M$ where every element of $M$ gets sampled with equal probability $\frac{1}{N}$ at each of the $N$ sampling rounds.

To ease notation we define, that $\tilde{u} = \arg\max_u f(u)$, if $\tilde{u}$ maximizes $f$. If $\tilde{u}$ is not unique one takes a random element.

The bagging algorithm is given in Fig.1.2.

---

**Input:** learning set $\mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathcal{X}, g_i \in \mathcal{G}\}$

- weak classifier of the form $h: \mathcal{X} \rightarrow \mathcal{G}$
- $T$: number of base classifiers

**For** $t = 1, \ldots, T$:

- Get a bootstrap-sample from $\mathcal{L}$ and use this bootstrap sample to get the weak classifier $h_t$.

**Output:** final classifier $H(x)$:

$$H(x) = \arg\max_{g \in \mathcal{G}} |\{t; h_t(x) = g\}|$$

---

When does bagging function?

1.) A trivial answer is that $h$ must be different for different learning sets. This property of $h$ can also be called instability of $h$ or more precisely instability against resampling. In a bias-variance analysis the success of bagging is explained by variance reduction.

2.) On average bootstrap samples contain only about 63% of the cases of the original learning sample. Therefore another condition is that this lack of data
shouldn’t deteriorate the classifier too much.

Boosting can be seen as a generalization of bagging. The general basic boosting algorithm is given in Fig. 1.3.

\[ \text{Input: learning set } L = \{ (x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathcal{X}, g_i \in \mathcal{G} \} \]

\[ \text{weak classifier of the form } h : \mathcal{X} \rightarrow \mathcal{G} \]

\[ T: \text{number of base classifiers} \]

Initialization: \( D_1(i) = \frac{1}{N} \)

For \( t = 1, \ldots, T \):

\begin{itemize}
  \item Take a weighted sample from \( L \) with weights \( D_t \), use this sample to get the weak classifier \( h_t \) and calculate some measure \( \epsilon_t \) for the performance of \( h_t \).
  \item Check a stopping criterion, which usually depends on \( \epsilon_t \).
  \item Choose \( \alpha_t \). Usually \( \alpha_t \) is some fixed function of \( \epsilon_t \).
  \item Update \( D \). Usually \( D \) gets bigger for examples, which are hard to classify.
\end{itemize}

Output: final classifier \( H(x) \):

\[ H(x) = \arg \max_{g \in \mathcal{G}} \sum_{t; h_t(x) = g} \alpha_t \]

Figure 1.3: General boosting algorithm

A formulation of bagging more similar to the formulation of boosting would be to say "draw a weighted sample from \( L \) with weights \( D(i) = 1/N \ \forall i \)" and

\[ H(x) = \arg \max_{g \in \mathcal{G}} \sum_{t; h_t(x) = g} 1. \]

A difference from bagging to boosting is that all training samples have the same probability to be chosen at each sampling step, where boosting changes the sampling probability after the next base classifier was constructed. The idea behind the update is to force the base classifiers to concentrate on hard-to-classify points. One consequence of this iterative update is that boosting works sequentially while for bagging all base classifiers can also be obtained parallel.

The final bagging classifier is an unweighted voting of the base classifiers, for boosting the final classifier consists of weighted votings. Usually the weight of a good base classifier is higher than for a bad one. The complexity of the resulting classifiers is quite similar, which leads to similar error bounds for the generalization error [25].

The algorithm above takes samples from the original learning set \( L \). For some base classifiers, as decision stumps, one can directly feed the weights to the classifier. This is called boosting by reweighting is a more exact use of the
weights $D_t(i)$, which we will call sampling distribution anyway. In 5.2.2 we will compare boosting with resampling and boosting with reweighting.

### 1.1.3 The roots of boosting algorithms

The roots of boosting lie in the PAC (Probably Approximately Correct) learning model:

A concept $c$ is a map $c : X \rightarrow \{0, 1\}$ and $X$ is called domain. A concept class $C$ is a class of concepts and a distribution $D : X \rightarrow [0, 1]$. As in the classification setting a learning set $\mathcal{L}$ is given.

The goal is to find a hypothesis $h : X \rightarrow [0, 1]$ with small error $E_{X \sim D}(|h(x) - c(x)|)$.

Note that the difference to the classification problem is that there exists a deterministic process $c$ instead of a probability distribution $P$ on $(X \times \mathbb{G})$.

**Definition:** weak and strong PAC learning algorithms:

1.) A strong learning algorithm is a learning algorithm which can, given access to independently drawn learning samples, suffice the following 2 conditions:
   (i) For all $\varepsilon < 0$ and $\delta > 0$ it outputs a hypothesis $h$ that has with probability $1 - \delta$ an error less than $\varepsilon$.
   (ii) The running time is polynomial in $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$ and all other relevant parameters (e.g. $N$).

2.) A weak learning algorithm is an algorithm that fulfills conditions (i) and (ii) but only for $\varepsilon \geq \frac{1}{2} - \gamma$ for $\gamma > 0$. That means that a weak learning algorithm only has to be better than guessing.

The original aim of a boosting algorithm was to convert a weak learning algorithm into a strong learning algorithm. In the next two sections we will see, that boosting algorithms deserve their name.

Note: condition (i) can only be fulfilled for deterministic problems, i.e. if the Bayes-error is zero.
1.2 AdaBoost for two groups

1.2.1 Description of the algorithm

In this chapter we introduce the original algorithm AdaBoost [11] for the two class case \(|G| = 2\) with \(G = \{-1, +1\}\) and \(h : \mathbb{X} \rightarrow \{-1, +1\}\) which will lead to a simpler notation. Note that the original algorithm can permit also confidence-rated classifiers of the form \(h : \mathbb{X} \rightarrow [-1, +1]\), which we do not permit here for sake of simplicity.

**Note:** We will often use the product \(g_i h(x_i)\) which is 1, if \(g_i = h(x_i) = 1\) or \(g_i = h(x_i) = -1\) and 0, if \(1 = g_i \neq h(x_i) = -1\) or \(-1 = g_i \neq h(x_i) = 1\). Therefore \(g_i h(x_i) = \begin{cases} +1 & g_i = h(x_i) \\ -1 & g_i \neq h(x_i) \end{cases}\)

---

**Input:** learning set \(\mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N) ; x_i \in \mathbb{X}, g_i \in G\}\),
\(G = \{-1, +1\}\) weak classifier of the form \(h : \mathbb{X} \rightarrow G\)
\(T\): number of boosting rounds

**Initialization:** \(D_1(i) = \frac{1}{N}\)

For \(t = 1, \ldots, T\):

- Train the weak classifier with distribution \(D_t\) where \(h_t\) should minimize
  \[ \epsilon_t := \sum_i D_t(i) g_i h_t(x_i) \]

- Choose \(\alpha_t \geq 0\)

- Update D:
  \[ D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t g_i h_t(x_i)}}{Z_t} \]
  where \(Z_t\) is a normalization factor (chosen so that \(D_{t+1}\) is a distribution)

**Output:** final classifier \(H(x)\):

\[ H(x) = \text{sign}(f(x)) \text{ with } f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \]

---

Figure 1.4: Algorithm AdaBoost for 2 groups

In the algorithm \(\alpha_t\) is unspecified. We just need to know now that \(\alpha_t\) has the following 2 properties:
(i) \(\alpha_t \geq 0\).
(ii) The better \(h_t\) the bigger \(\alpha_t\).

We will derive a good choice for \(\alpha_t\) in the next subsection.
CHAPTER 1. FUNDAMENTALS

Cases wrongly classified by \( h_t \) are sampled with higher probability at the next round and right classified cases with lower probability, because for positive \( \alpha_t \), we have

\[
e^{-\alpha_t g_i f(x_i)} = \begin{cases} 
  \geq 1 & \text{if } h_t(x_i) \neq g_i \\
  \leq 1 & \text{if } h_t(x_i) = g_i
\end{cases}
\]

In the last step the algorithm combines the hypotheses \( h_1, \ldots, h_T \) with different weights to the final classifier \( H \). The better the hypothesis the higher the weight \( \alpha_t \).

The value \( |f(x_i)| \) can be regarded as a measure of confidence in the classification of \( x_i \) with the boosting classifier \( H \). In the twoclass case \( g_i f(x_i) \) is just identical to the margin of example \( x_i \). For the multiclass case the margin of example \( x_i \) is defined as \( f(x_i, g_i) - \max_{g \neq g_i} f(x_i, g) \). The margin \( \rho \) of the classifier \( H \) is defined as the minimum margin over all examples of the training set. It plays an important role in the theory about generalization error [18, 25] and provides some ways to make boosting more resistant against noise [22, 23, 13, 27].

In the current version the algorithm needs the number of boosting rounds \( T \) as an input, because we do not have a stopping criterion yet. From the analysis of the next section we will get a stopping criterion for the algorithm so that \( T \) won’t be required any more.

1.2.2 Choice of \( \alpha_t \) and stopping criterion

In this chapter we demand that the learning error \( \frac{1}{N} |\{i; H(x_i) \neq g_i\}| \) declines as fast as possible with the boosting round \( t \) which leads to the specific choice of \( \alpha_t \) and also to a stopping criterion.

Lemma: For the boosting algorithm above the following holds:

\[
\frac{1}{N} |\{i; H(x_i) \neq g_i\}| \leq \prod_{t=1}^{T} Z_t
\]

Proof: We can rewrite the training error as

\[
\text{error} := \frac{1}{N} |\{i; H(x_i) \neq g_i\}| = \frac{1}{N} \sum_{i=1}^{N} [H(x_i) \neq g_i]
\] (1.1)

If \( H(x_i) \neq g_i \) then \( g_i f(x_i) \leq 0 \) and \( e^{-g_i f(x_i)} \geq 1 = [H(x_i) \neq g_i] \).
If \( H(x_i) = g_i \) we have \( e^{-g_i f(x_i)} \geq 0 = [H(x_i) \neq g_i] \).
This leads to

\[
[H(x_i) \neq g_i] \leq e^{-g_i f(x_i)}
\] (1.2)

Substituting (1.2) in (1.1) we get

\[
\text{error} \leq \frac{1}{N} \sum_{i=1}^{N} e^{-g_i f(x_i)}
\] (1.3)
Now we have to unravel the update rule of $D_{t+1}$:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t g_t(x_i)}}{Z_t} = \frac{D_{t-1}(i)e^{-\alpha_{t-1} g_{t-1}(x_i)}e^{-\alpha_t g_t(x_i)}}{Z_{t-1}} = \cdots =$$

$$= D_1(i) \prod_{s=1}^{t} \frac{e^{-\alpha_s g_s(x_i)}}{Z_s}$$

$$= \frac{1}{N} e^{-g_t \sum_{s=1}^{t} \alpha_s h_s(x_i)} \prod_{s=1}^{t} \frac{1}{Z_s}$$

Combining the last equation with (1.3) we get

$$\text{error} \leq \sum_{i=1}^{N} \prod_{s=1}^{t} Z_s D_{t+1}(i) = \prod_{s=1}^{t} Z_s \sum_{i=1}^{N} D_{t+1}(i) = \prod_{s=1}^{t} Z_s$$

Now we will use the weighted training error

$$\epsilon_t := \sum_{i=1}^{N} D_t(i)[h_t(x_i) \neq g_i]$$

of the hypothesis $h_t$ to choose $\alpha_t$ and upper bound the training error of the boosting classifier.

**Theorem:** Let $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$ and $\gamma_t := \frac{1}{2} - \epsilon_t \geq \gamma > 0$. Then

$$\frac{1}{N} \left| \{i; \ H(x_i) \neq g_i\} \right| \leq 2^T \prod_{i=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)} \leq e^{-2\gamma^2 T}.$$

**Notes:**

1.) The theorem states that the training error decreases exponentially fast, if the base classifier $h_t$ is at every round a little bit better than random guessing.

2.) Another formulation would be: AdaBoost turns a weak learning algorithm $h$ into a strong learning algorithm, so AdaBoost is really a boosting algorithm.

3.) In practice $\epsilon_t$ increases with $t$, so the analysis provides the stopping criterion "stop, if $\epsilon_t \geq \frac{1}{2}".$

4.) The choice for $\alpha_t$ has the following properties:

$$\alpha_t = \begin{cases} 
\geq 0 & \text{if } \epsilon_t \leq \frac{1}{2} \\
\leq 0 & \text{if } \epsilon_t \geq \frac{1}{2}
\end{cases}$$

and the smaller $\epsilon_t$ the bigger $\alpha_t$ for the combination step.
Proof: From the lemma we have

\[
\text{error} = \frac{1}{N} |\{i; H(x_i) \neq g_i\}| \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left( \sum_{i} D_t(i) e^{-\frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) g_i h_t(x_i)} \right)
\]

Using \(g_i h_t(x_i) = \begin{cases} +1 & g_i = h_t(x_i) \\ -1 & g_i \neq h_t(x_i) \end{cases}\) we get

\[
\text{error} \leq \prod_{t=1}^{T} \left( \sqrt{\frac{\epsilon_t}{1 - \epsilon_t}} \sum_{i; h_t(x_i) = g_i} D_t(i) + \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \sum_{i; h_t(x_i) \neq g_i} D_t(i) \right)
\]

\[
= \prod_{t=1}^{T} \left( \sqrt{\frac{\epsilon_t}{1 - \epsilon_t}} (1 - \epsilon_t) + \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} \epsilon_t \right)
\]

\[
= 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)}
\]

\[
\leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2}
\]

Now using \(\sqrt{1 + x} \leq 1 + \frac{1}{2} x\) for \(x \leq 0\) (which can be seen by Taylor expansion of \((1 + x)^k\)) we get

\[
\text{error} \leq \prod_{t=1}^{T} \left( 1 - 2\gamma_t^2 \right)
\]

With use of \(e^x \geq 1 + x\) for \(x \leq 0\) (look at Taylor expansion of \(e^x\)) we come to

\[
\text{error} \leq \prod_{t=1}^{T} e^{-2\gamma_t^2} = e^{-2 \sum_{t=1}^{T} \gamma_t^2} \leq e^{-2\gamma^2 T}.
\]

1.2.3 Confidence-rated boosting

In most boosting algorithms the base classifier only assigns an object to a group without giving any information about the confidence in this assignment. As we use decision stumps which are able to output this confidence, it is natural to use this information where possible. In the previous algorithm the information about the confidence was not incorporated in the error bound and the choice of \(\alpha_t\), because they use only the crude error rate \(\epsilon_t\), which does not use the information about the confidences.

Schapire and Singer [26] gave a more general analysis of AdaBoost and obtained an error bound and a choice of \(\alpha_t\), which also uses the confidence-rated prediction there.

Theorem: Assume each \(h_t\) has range \([-1, +1]\) and that we choose

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 + r_t}{1 - r_t} \right)
\]
where
\[ r_t = \sum_{i=1}^{N} D_t(i) g_i h_t(x_i) = E_{i \sim D_t}[g_i h_t(x_i)]. \]

Then the training error is at most
\[ \prod_{t=1}^{T} \sqrt{1 - r_t^2}. \]

**Note:** If \( h_t \) has range \( \pm 1 \) then
\[ r_t = - \sum_{i; h_t(x_i) \neq g_i} D_t(i) + \sum_{i; h_t(x_i) = g_i} D_t(i) = -2 \sum_{i; h_t(x_i) \neq g_i} D_t(i) + \sum_{i=1}^{N} D_t(i) = -2 \epsilon_t + 1. \]

Therefore maximizing \( r_t \) is equivalent to minimizing \( \epsilon_t \) and substitution in the formula for \( \alpha_t \) gives
\[ \alpha_t = \frac{1}{2} \ln \left( \frac{1 - 2 \epsilon_t + 1}{1 + 2 \epsilon_t - 1} \right) = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \]

which is the same \( \alpha_t \) as found from the original error bound.

Both in the section about GrPloss and in the section about BoostMA we use modifications of the proof of this theorem to get bounds for measures of performance and good choices for alpha for these algorithms.
1.3 AdaBoost for multiclass problems

This section describes the common boosting algorithms for the multiclass case: AdaBoost.M1 (which does not work for weak classifiers) in sections 1.3.1 and 1.3.2, AdaBoost.M2 in section 1.3.3 and the reduction to two-class problems with error-correcting output codes in section 1.3.4.

1.3.1 Description of AdaBoost.M1

In this chapter we will make the straightforward extension of AdaBoost to the multiclass case. The differences to AdaBoost for 2 groups are:

1.) Now we need a base classifiers \( h : X \to G = \{1, \ldots, |G|\} \).

2.) We replace the term \( g_i h_t(x_i) \) by a similar term which can be extended to more than two classes. We will use

\[
2([h_t(x_i) = g_i]) - \frac{1}{2} = \begin{cases} +1 & g_i = h(x_i) \\ -1 & g_i \neq h(x_i) \end{cases}
\]

So instead of \( e^{-\alpha_t g_i h_t(x_i)} \) we use \( e^{-2\alpha_t [h_t(x_i) = g_i]} \) by throwing away the term \( e^\alpha \) which is the same for all \( i \) and therefore superfluous because of the normalization in the update step.

3.) In the combination step we must replace the sign. We can use

\[
H(x) = \arg\max_{g \in G} \left( \sum_{t=1}^{T} \alpha_t [h_t(x) = g] \right)
\]

The resulting algorithm is given in Fig.1.5

1.3.2 Decrease of learning error for AdaBoost.M1

**Theorem:** Consider AdaBoost.M1 with \( \epsilon_t \leq \frac{1}{2} - \gamma \). Then

\[
\frac{1}{N} | \{ i; H(x_i) \neq g_i \} | \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)} \leq e^{-2\gamma T}.
\]

**Proof:** The proof reduces the situation of AdaBoost.M1 to a situation of AdaBoost for 2 groups and then applies the theorem of section 1.2.2 leading to the result above. We will mark all variables of the two group algorithm with a tilde. Parallel to AdaBoost.M1 we consider AdaBoost for two groups applied to the special two group problem \( \hat{\mathcal{L}} = \{(i, -1); i = 1, \ldots, N\} \), the base classifier \( \hat{h}_t(i) := 2([h_t(x_i) \neq g_i]) - \frac{1}{2} \) and \( \hat{D}_1(i) = \frac{1}{N} \).

1.) First we prove that \( \hat{D}_t(i) = D_t(i), \hat{\epsilon}_t = \epsilon_t \) and \( \hat{\alpha}_t = \alpha_t \) by induction over \( t \):

\[
t = 1: \hat{D}_1(i) = D_1(i) = \frac{1}{N}
\]

\[
\epsilon_1 := \sum_i D_1(i)[h_t(x_i) \neq g_i]
\]
**Input:** learning set $L = \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in X, g_i \in G\}$, $G = \{1, \ldots, |G|\}$, weak classifier of the form $h: X \rightarrow G$

Optionally $T$: number of boosting rounds

**Initialization:** $D_1(i) = \frac{1}{N}$

For $t = 1, \ldots, T$:

- Train the weak classifier with distribution $D_t$ where $h_t$ should minimize
  $$\epsilon_t := \sum_i D_t(i)[[h_t(x_i) \neq g_i]]$$

- If $\epsilon_t \geq \frac{1}{2}$: **goto** output with $T := t - 1$

- Set
  $$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

- Update $D$:
  $$D_{t+1}(i) = \frac{D_t(i)e^{-2\alpha_t[[h_t(x_i) = g_i]]}}{Z_t}$$
  where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ is a distribution)

**Output:** final classifier $H(x)$:

$$H(x) = \arg \max_{g \in G} \left( \sum_{t=1}^T \alpha_t[[h_t(x) = g]] \right)$$

---

Figure 1.5: Algorithm AdaBoost.M1

$$\tilde{h}_t(i) = 2([[h_t(x_i) \neq g_i]] - \frac{1}{2})$$

$$\Rightarrow [[h_t(x_i) \neq g_i]] = \frac{1}{2} \tilde{h}_t(i) + \frac{1}{2}$$

$$= \begin{cases} 0 & \tilde{h}_t(i) = -1 \\ 1 & \tilde{h}_t(i) = +1 \end{cases}$$

$$= [[\tilde{h}_t(i) = 1]]$$

Therefore $\epsilon_1 = \sum_i \tilde{D}_1(i)[[\tilde{h}_1(i) = 1]]$ which is just $\tilde{\epsilon}_1$, since all cases in $\tilde{L}$ are in group -1.

$$\tilde{\alpha}_1 = \frac{1}{2} \ln \left( \frac{1 - \tilde{\epsilon}_1}{\tilde{\epsilon}_1} \right) = \frac{1}{2} \ln \left( \frac{1 - \epsilon_1}{\epsilon_1} \right) = \alpha_1.$$
CHAPTER 1. FUNDAMENTALS

1. FUNDAMENTALS

\[ t \to t + 1: \]
\[ D_{t+1}(i) \propto D_t(i) e^{-\tilde{\alpha}_t \tilde{h}_t(i)} = D_t(i) e^{-\tilde{\alpha}_t (-2[[h_t(x_i) \neq g_i]] - \frac{1}{2})} = D_t(i) e^{2\alpha_t (\frac{1}{2} - [[h_t(x_i) = g_i]])} = D_{t+1}(i) e^{\alpha_t} \]

by using the induction assumption. Because both \( \tilde{D}_{t+1}(i) \) and \( D_{t+1}(i) \) are distributions, the constant factors vanish during normalization, so \( \tilde{D}_{t+1}(i) \) and \( D_{t+1}(i) \) are the same.

The proof that \( \tilde{\epsilon}_{t+1} = \epsilon_{t+1} \) and \( \tilde{\alpha}_{t+1} = \alpha_{t+1} \) is analogous to \( t = 1 \).

2.) Now we want to show: \( H(x_i) \neq g_i \Rightarrow \tilde{H}(i) = 1 \).

When we have shown this we are at the end of the proof, because then we have for the errors of \( H \) and \( \tilde{H} \)
\[
\text{error}(H) = Pr_{i \sim D}(H(x_i) \neq g_i) \leq Pr_{i \sim \tilde{D}}(\tilde{H}(i) = 1) = \text{error}(\tilde{H}) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)}
\]

where we used the theorem of 1.2.2 to get the second inequality.

\[ H(x_i) \neq g_i \Leftrightarrow \sum \alpha_i [[h_t(x_i) = g_i]] \leq \sum \alpha_i [[h_t(x_i) = H(x_i)]] \]

Since also \( \sum \alpha_i [[h_t(x_i) = g_i]] + [[h_t(x_i) = H(x_i)]] \leq \sum \alpha_i \) and \( \alpha_i \geq 0 \) hold, we get by using

\[ (a + b \leq 1) \land (a, b \geq 0) \land (a \leq b) \Rightarrow a \leq \frac{1}{2} \]
\[ \sum_i \alpha_i [[h_t(x_i) = g_i]] \leq \frac{1}{2} \sum_i \alpha_i. \]

Thus \( \sum_i \tilde{\alpha}_i (\frac{1}{2} - \frac{h_t(x_i)}{2}) \leq \frac{1}{2} \sum_i \tilde{\alpha}_i, \)
so \( \sum_i \tilde{\alpha}_i \tilde{h}_t(x_i) \geq 0 \), which means that \( \tilde{H}(i) = 1 \).

Note: The condition \( \epsilon_t \leq \frac{1}{2} \) for more than 2 groups is not trivial. For example the best decision stump for the digit example has an error rate of more than 80%, so AdaBoost.M1 doesn’t work there, one needs a stronger base classifier than decision stumps for the digit example. AdaBoost.M2 is an algorithm, that also works with such weak classifiers. Also the algorithms GrPloss and BoostMA are designed to work with weak base classifiers.

1.3.3 AdaBoost.M2

A multiclass extension for weak classifiers that is more sophisticated than AdaBoost.M1 is AdaBoost.M2 (figure 1.6) published by Freund and Schapire ([11]).

AdaBoost.M2 changes the goal of the weak classifiers to predict a set of plausible groups and evaluates the weak classifiers using the pseudo-loss \( \epsilon_t \) (step 2)
Input: learning set \( \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathbb{X}, g_i \in \mathbb{G}\} \), weak classifier of the form \( h: \mathbb{X} \times \mathbb{G} \rightarrow [0, 1] \), \( T \): maximum number of boosting rounds

Initialization: \( D_1(i) = 1/N \), weight vector \( w_{i,g}^1 = D_1(i)/(|\mathbb{G}| - 1) \)

For \( t = 1, \ldots, T \):
1. Set \( W_t^i = \sum_{g \neq g_i} w_{i,g}^t \)
   for \( g \neq g_i \): \( q_{i,g}^t = w_{i,g}^t/W_t^i \)
   and
   \( D_t(i) = W_t^i/N \sum_{i=1}^N W_t^i \)
2. Call the weak classifier \( h_t \) with distribution \( D_t \)
   and label weighting function \( q_t \); \( h_t \) should minimize the pseudo-loss \( \epsilon_t \):
   \[
   \epsilon_t = \frac{1}{2} \sum_{i=1}^N D_t(i) \left( 1 - h_t(x_i, g_i) + \sum_{g \neq g_i} q_{i,g}^t h_t(x_i, g) \right)
   \]
3. Set \( \beta_t = 1 - \epsilon_t \)
4. For \( i = 1, \ldots, N \) and \( g \in \mathbb{G} \setminus \{g_i\} \) set the new weight vectors to be
   \[
   w_{i,g}^{t+1} = w_{i,g}^t \beta_t (1 + h_t(x_i, g) - h_t(x_i, g_i))
   \]

Output: final classifier \( H \):
\[
H(x) = \arg \max_{g \in \mathbb{G}} \sum_{t=1}^T \ln \left( \frac{1}{\beta_t} \right) h_t(x, g)
\]

Figure 1.6: Algorithm AdaBoost.M2

which penalizes the weak classifiers for failing to include the correct group in the predicted plausible group set and for each incorrect label in the predicted plausible group set. The exact form of the pseudo-loss is under control of the algorithm so that the weak classifier can focus also on the groups which are hardest to distinguish from the correct group by changing the matrix \( q \). The following theorem guarantees the decrease of the training error as long as the pseudo-loss is less than \( 1/2 \), which is much more easier to achieve than training error less than 50%.

**Theorem:** Let \( \epsilon_t \leq \frac{1}{2} - \gamma \) for \( \gamma \in (0, 1) \). Then the training error of AdaBoost.M2 is upper bounded by
\[
(|\mathbb{G}| - 1) 2^T \prod_{t=1}^T \sqrt{\epsilon_t(1 - \epsilon_t)} \leq \exp(-2\gamma^2 T).
\]

For this algorithm the pseudo-loss plays the same role as the error rate in the previous boosting algorithms. In every boosting round \( t \) the algorithm gets a weak classifier by minimizing the weighted pseudo-loss. The pseudo-loss is used for updating the sampling weights \( D(i) \) (and the matrix \( q \)) for the next boosting round (steps 3, 4 and 1). As the previous ones this algorithm focuses more on cases that were difficult to classify properly at the previous boosting round. The pseudo-loss also implicitly occurs in the weighted majority-vote of the \( T \) base
classifiers (step 3 and output).

For the sake of a better comparison with the other algorithms we introduce a more modern notation and make some simple rearrangements:
The update of the sampling distribution of AdaBoost.M2 consists of updating w, W and D, which can be found in steps 4 and 1: we omitted the definition of W and put all updates together at the end of the boosting loop. This includes the update of q. Note that q doesn’t occur explicitly in the update for the sampling distribution, it only plays a role in the calculation of $\epsilon_t$.
Instead of $\beta_t$ we will use $\alpha_t$ as before and we also built in the stopping criterion.
Note, that in the output step an object x is assigned to the group with the highest score $f(x, g)$. The resulting algorithm is given in Fig.1.7

---

**Input:** learning set $\{(x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathbb{X}, g_i \in \mathbb{G}\}, \mathbb{G} = \{1, \ldots, |\mathbb{G}|\}$

weak classifier of the form $h : \mathbb{X} \times \mathbb{G} \rightarrow [0, 1]$

Optionally $T$: maximum number of boosting rounds

**Initialization:** $D_1(i) = 1/N, w_{1,i,g}^1 = D_1(i)/(|\mathbb{G}| - 1), q_{1,i,g}^1 = \frac{1}{|\mathbb{G}| - 1}$

For $t = 1, \ldots, T$

- Train the weak classifier $h_t$ with distribution $D_t$ and matrix $q_t$ where $h_t$ should minimize
  $$\epsilon_t = \frac{1}{2} \sum_{i=1}^{N} D_t(i) \left( 1 - h_t(x_i, g_i) + \sum_{g \neq g_i} q_{t,i,g}^t h_t(x_i, g) \right)$$

- if $\epsilon_t \geq \frac{1}{2}$: $T := t - 1$, goto output step

- Set
  $$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

- Update of $D(i)$, $w_{i,g}$ and $q$:
  $$w_{t+1,i,g} + 1 = \frac{w_t^{t+1,i,g} e^{-\alpha_t(1-h_t(x_i, g_i) + h_t(x_i, g))}}{Z_t}$$
  $$D_{t+1}(i) = \frac{1}{Z_t} \sum_{g \neq g_i} w_{t,i,g}^{t+1}$$
  with normalization constant $Z_t$
  $$q_{t+1,i,g} = \frac{w_{t+1,i,g}}{\sum_{g \neq g_i} w_{t+1,i,g}}$$

**Output:** final classifier $H = h_f$: Normalize $(\alpha_t)_{t=1,\ldots,T}$ and set

$$H(x) = h_f(x) = \arg \max_{g \in \mathbb{G}} f(x, g) := \arg \max_{g \in \mathbb{G}} \sum_{t=1}^{T} \alpha_t h_t(x, g)$$

---

Figure 1.7: Algorithm AdaBoost.M2, modern notation
1.3.4 Error-correcting output codes

Using output codes is a method for reducing multiclass classification problems to twoclass problems ([1, 5, 6, 7, 14, 16, 17, 24]). The aim is to improve the performance of inductive learning with respect to generalization error and resistance to noise. The idea of the algorithm is to present each class \( g \) by a binary codeword

\[
a_g = (a_{1g}, \ldots, a_{Sg}) \text{ with } a_{sg} \in \{\pm 1\}
\]

of a given length \( S \). We can summarize the coding information for all \(|G|\) groups in the codematrix

\[
a := (a_{sg})_{s=1,\ldots,S, g=1,\ldots,|G|}.
\]

Each bit of the codewords divides the learning set in two groups

\[
G_{-1} := \{(x_i, g_i) \in L; a_{sg_i} = -1\} \quad \text{and} \quad G_{+1} := \{(x_i, g_i) \in L; a_{sg_i} = +1\}.
\]

Therefore for each bit \( s \) a twoclass learning algorithm \( r_s \) can be constructed to learn this partition. An unseen object \( x \) which should be classified gets analyzed by the \( S \) twoclass rules \( r_s \) and the \( S \) resulting bits are concatenated to the codeword \( r(x) \) of the object

\[
r(x) := (r_1(x), \ldots, r_S(x)) \in \{\pm 1\}^S
\]

Object \( x \) is then assigned to the group \( g \) with the most similar codeword, where this similarity is most often defined by the Hamming distance

\[
g := \arg \min_{\tilde{g} \in G} d_H(r(x), a_{s \tilde{g}})
\]

with

\[
d_H(u, v) := |\{j; u_j \neq v_j\}|
\]

If the codewords are very different, then some twoclass rules can make mistakes, but the most similar codeword is still the right one. Therefore some errors of twoclass rules can be corrected.

The algorithm of the basic method is given in figure 1.8.

---

**Input:** Learning set \( L \), codematrix \( a \) and a base inducer \( r \).

**For** \( s = 1, \ldots, S \): Make binary classifications

- Set training set \( T_s = \{(x_i, a_{sg_i}); (x_i, g_i) \in L\} \).
- Get classifier \( r_s : X \to \{\pm 1\} \) from this training set.

**Output:** For each \( x \) to classify set

\[
g := \arg \min_{\tilde{g} \in G} d_H(r(x), a_{s \tilde{g}})
\]

---

Figure 1.8: Basic reduction to twoclass problems with error-correcting output codes
A good code has 2 properties:

a) Column separation: Each codeword should be well-separated from the other codewords with respect to the distance $d$, so that many two-class classification errors can be corrected.

b) Row separation: The two-class classifiers $r_s$ should be uncorrelated with each other. That means that the distance between rows should also be large.

There are also some minimal requirements: no rows consisting of all ones or all zeros, and remove complements of rows.

Example for standard codes are

a) OPC-codes (one per class): $a$ is a $|G| \times |G|$ matrix with

$$a_{sg} = \begin{cases} 
+1 & g = s \\
-1 & g \neq s 
\end{cases}$$

This code has only Hamming distance two between codewords.

b) ”Random codes“: $a_{sg}$ is chosen at random with equal probability from $\{\pm 1\}$. The expected Hamming distance is $|G|/2$ between two codewords, and they also have expected Hamming distance $|G|/2$ between two columns.

There are several extensions of this basic approach

a) In the version above the group space $G$ gets divided into a partition of two group spaces $G_{-1}$ and $G_{+1}$, where every original group $g$ belongs to exactly one of this subspaces. The first extension just takes two disjoint subspaces $G_{-1}$ and $G_{+1}$ from $G$ and compares them. There can be groups $g$, which do not belong either to $G_{-1}$ or $G_{+1}$. We can make this extension by allowing zeros as values in the codematrix

$$a_{sg} \in \{-1, 0, +1\}$$

where $a_{sg} = 0$ means that group $g$ does not belong to the training set $T_s$ ([1]).

b) The two-class classifier $r_s$ in the algorithm just outputs $+1$ or -1. One can of course let the classifier output some measure of confidence in the prediction ([16]).

c) There is also a method to get probability estimates from error-correcting output codes ([16]), which leads to a system of $S$ linear equations for $p_{|x} = (p(1|x), \ldots, p(|G||x))$

$$ap_{|x} = r(x)$$

together with the constraints $p(g|x) \geq 0$ and $\|p_{|x}\|_1 = 1$. 
1.4 Boosting as gradient descent in function space

This chapter mainly contains results from [19], where a general framework was developed which sees boosting as gradient descent in function space. This framework will be the basis for the construction of the algorithm GrPloss in the next chapter.

1.4.1 Gradient descent for numerical optimization

**Task:** Let \( L : \mathbb{R}^n \to \mathbb{R} \). Now we search a point \( x^* := \arg \min_{x \in \mathbb{R}^n} L(x) \). The standard example is a wanderer on a mountain, who wants to get down to the valley: for this example \( x \in \mathbb{R}^2 \) and \( L(x) \) has the interpretation as the height at position \( x \). \( L \) is called loss function.

Numerical optimization methods often start at a starting point \( x_0 \) and approach to \( x^* \) with small steps \( \Delta x: x_{\text{new}} = x_{\text{old}} + \Delta x_{\text{new}} \). When we make \( T \) steps we get

\[
x_T = x_0 + \Delta x_1 + \ldots + \Delta x_T = x_0 + \sum_{t=1}^{T} \Delta x_t
\]

We want to have \( x_T \approx x^* \) and \( L(x_T) \approx L(x^*) = \min_{x \in \mathbb{R}^n} L(x) \).

When a wanderer wants to get down as quickly as possible he usually makes a step in the direction of the steepest descent which is just a step in the direction opposite to the gradient of \( L \) at his position:

\[
\Delta x_t \sim - \frac{\partial L}{\partial x}(x_{t-1}) =: -\nabla L(x_{t-1})
\]

By choosing a stepsize \( \alpha_t \) we get one step of the method:

\[
x_t = x_{t-1} + \alpha_t \Delta x_t = x_{t-1} + \alpha_t (-\nabla L(x_{t-1}))
\]

A standard choice of \( \alpha_t \), usually called line search, is

\[
\alpha_t = \arg \min_{\alpha} L(x_{t-1} + \alpha (-\nabla L(x_{t-1})))
\]

The standard stopping criterion stops when the decrease of \( L \) doesn’t exceed a given value \( TOL \). The algorithm is given in Fig. 1.9

Gradient descent finds local minima, so there is the possibility that the algorithm gets stuck at a local minimum \( \tilde{x} \) with high \( L(\tilde{x}) \). By altering the starting value one can try to find other local minima with lower \( L \) value. Generally descent is a simple, but not very satisfying method. But for convex loss functions gradient descent is able to find the global minimum, because it’s also the only local minimum.
CHAPTER 1. FUNDAMENTALS

Input: \( L : \mathbb{R}^n \rightarrow \mathbb{R} \), starting value \( x_0 \) and minimum decrease \( TOL \). \( t := 1 \)

Loop: while \( \Delta L \geq TOL \)

- \( \alpha_t = \arg \min_{\alpha} (L(x_{t-1} + \alpha(-\nabla L(x_{t-1})))) \)
- \( x_t = x_{t-1} + \alpha_t(-\nabla L(x_{t-1})) \)
- \( \Delta L := L(x_{t-1}) - L(x_t) \)

Output: \( x_t, L(x_t) \)

Figure 1.9: Algorithm numerical gradient descent

1.4.2 Gradient descent in function space

Task: We have a function space \( F = \text{lin}(\mathcal{H}) \) consisting of functions of the form

\[
 f(x; \alpha, \beta) = \sum_{t=1}^{T} \alpha_t h_t(x; \beta_t), \quad h_t : X \rightarrow \{\pm 1\}
\]

with \( \alpha = (\alpha_1, \ldots, \alpha_T), \beta = (\beta_1, \ldots, \beta_T) \) and \( h_t \in \mathcal{H} \).

We also have a learning set \( L = \{(x_i, g_i), \ldots, (x_N, g_N)\} \) and a loss function

\[
 L(f) = E_{g,x}[l(f(x), g)] = E_{x}[E_{g}[l(f(x), g)]], \quad l : \mathbb{R} \times \mathbb{G} \rightarrow \mathbb{R}_{\geq 0}
\]

The goal is to find \( f^* = \arg \min_{f \in F} L(f) \).

Notes: 1.) For decision stumps \( \beta_t = (j, c) \) and \( \beta_t = (j, S) \) for ordered and unordered \( X_j \) respectively.
2.) We define the gradient in a function space as follows:

\[
 \nabla L(f)(x) := \frac{\partial L(f + e1_x)}{\partial e} \bigg|_{e=0} = \lim_{e \rightarrow 0} \frac{L(f + e1_x) - L(f)}{e}
\]

with

\[
 1_x(y) = \begin{cases} 
    1 & y = x \\
    0 & y \neq x 
  \end{cases}
\]

Now we have the problem that \( -\nabla L(f_t)(x) \) is not necessarily an element of \( F \), so we replace it by an element \( h_t \) of \( F \) which is as parallel to it as possible. Therefore we need an inner product \( \langle , \rangle \) and set:

\[
 \beta_t := \arg \max_{\beta} (-\nabla L(f_{t-1}), h(\beta))
\]

and \( h_t := h(\beta_t) \). The inequality \( \langle -\nabla L(f_{t-1}), h(\beta_t) \rangle \leq 0 \) means, that we can’t find a good ”direction” \( h(\beta_t) \), so the algorithm stops.

The resulting algorithm is given in Fig. 1.10
CHAPTER 1. FUNDAMENTALS

Input: $l$, starting value $f_0$. $t := 1$

Loop: while $\langle -\nabla L(f_{t-1}), h(\beta) \rangle > 0$

- $\beta_t := \arg \max_{\beta} \langle -\nabla L(f_{t-1}), h(\beta) \rangle$
- $\alpha_t := \arg \min_{\alpha} (L(f_{t-1} + \alpha h_t(\beta_t)))$
- $f_t = f_{t-1} + \alpha_t h_t(\beta_t)$

Output: $f_t, L(f_t)$

Figure 1.10: Algorithm gradient descent in function space

1.4.3 AdaBoost as gradient descent in function space

Theorem: AdaBoost for $|G| = 2$ is a gradient descent algorithm with

$$l(f(x), g) = l(gf(x)) = e^{-gf(x)},$$

so that $L(f) = \frac{1}{N} \sum_{i=1}^{N} e^{-g_if(x_i)}$

and $\langle u, v \rangle := \frac{1}{N} \sum_{i=1}^{N} u(x_i)v(x_i)$

Note: For $|G| = 2$ the term $gf(x)$ is called margin of object $x$. A cost function $L$ with $l$ of the form $l(f(x), g) = l(gf(x))$ is called margin cost function.

Proof: (i) The first thing we prove is that

$$\nabla L(f)(x) = \begin{cases} 0 & x \neq x_i \\ \frac{1}{N} g_i l'(g_i f(x_i)) & x = x_i \end{cases}$$

$$\nabla L(f)(x) = \lim_{e \to 0} \frac{L(f + e1_x) - L(f)}{e}$$

$$= \lim_{e \to 0} \frac{1}{N} \sum_{i=1}^{N} (l(g_i(f + e1_x)(x_i)) - l(g_i f(x_i)))$$

$$= \begin{cases} 0 & x \neq x_i \\ \frac{1}{N} \lim_{e \to 0} \frac{l(g_i f(x_i) + g_i e) - l(g_i f(x_i))}{e} & x = x_i \end{cases}$$

Since

$$l'(g_i f(x_i)) = \lim_{e \to 0} \frac{l(g_i f(x_i) + e) - l(g_i f(x_i))}{e}$$

$$= \lim_{g_i e \to 0} \frac{l(g_i f(x_i) + g_i e) - l(g_i f(x_i))}{g_i e}$$

$$= \frac{1}{g_i} \lim_{e \to 0} \frac{l(g_i f(x_i) + g_i e) - l(g_i f(x_i))}{e}$$
we get statement (i).

(ii) If $L$ is a margin cost function with $l'(z) < 0$ (any sensible cost function and of course also $l(z) = e^{-z}$ should suffice that) then the following holds:

$$
\hat{h}_t = \arg \max_h \langle -\nabla L(f_{t-1}), h \rangle \iff h_t = \arg \min_h \sum_{i, h(x_i) \neq g_i} \tilde{D}_t(i)
$$

with \( \tilde{D}_t(i) = \frac{l'(g_if_{t-1}(x_i))}{\sum_i l'(g_if_{t-1}(x_i))} \)

because of (i)

$$
\hat{h}_t = \arg \max_h \langle -\nabla L(f_{t-1}), h \rangle
$$

\iff \frac{1}{N^2} \sum_i g_i l'(g_if_{t-1}(x_i)) h(x_i) \text{ is maximal}

Because $l'(z) < 0$ we can divide it by $\sum_i l'(g_if_{t-1}(x_i)) > 0$ and obtain

$$
\iff \frac{1}{N^2} \sum_i g_i h(x_i) \frac{l'(g_if_{t-1}(x_i))}{\sum_i l'(g_if_{t-1}(x_i))} \text{ is maximal}
$$

The fraction is just $\tilde{D}(i)$ and we also omit \( \frac{1}{N^2} \)

$$
\iff \sum_{i, h(x_i) = g_i} \tilde{D}(i) - \sum_{i, h(x_i) \neq g_i} \tilde{D}(i) = 1 - 2 \sum_{i, h(x_i) \neq g_i} \tilde{D}(i) \text{ is maximal}
$$

\iff \hat{h}_t = \arg \min_h \sum_{i, h(x_i) \neq g_i} \tilde{D}(i)

(iii) Now we show that $D_{t+1}(i) = \tilde{D}_{t+1}(i)$.

$$
D_{t+1}(i) = \frac{\frac{1}{N^t} \prod_{s=1}^{t} e^{-\alpha_s g_i h_s(x_i)}}{\prod_{s=1}^{t} Z_s} = \frac{\frac{1}{N^t} e^{-g_i f_t(x_i)}}{\prod_{s=1}^{t} Z_s}
$$

Since $D_{t+1}$ is a distribution we get

$$
D_{t+1}(i) = \frac{e^{-g_i f_t(x_i)}}{\sum_i e^{-g_i f_t(x_i)}} = \frac{l'(g_i f_t(x_i))}{\sum_i l'(g_i f_t(x_i))} = \tilde{D}_{t+1}(i).
$$

(iv) Now we show that the stopping criteria are the same: From

$$
\epsilon_t = \sum_{i, h(x_i) \neq g_i} \tilde{D}_t(i) \geq \frac{1}{2}
$$

we find by using that $D_t$ is a distribution and using

$$
(a + b = 1) \land (a \geq \frac{1}{2}) \Rightarrow b - a \leq 0
$$

that

$$
\sum_{i, h(x_i) = g_i} \tilde{D}_t(i) - \sum_{i, h(x_i) \neq g_i} \tilde{D}_t(i) \leq 0
$$
Using a part from the proof of (ii) starting from the left side of the previous inequality but in the opposite direction we get

\[
\frac{1}{N^2} \langle -\nabla L(f_{t-1}), h_t \rangle \leq 0
\]

(v) At last we prove that the optimal step sizes are the same: \( \tilde{\alpha}_t = \alpha_t \).

\[
\tilde{\alpha}_t = \arg \min_{\tilde{\alpha}} \left( L(f_t - 1 + \tilde{\alpha} h_t(\beta_t)) \right) = \arg \min_{\tilde{\alpha}} \left( \sum_i e^{-g_i(f_{t-1}(x_i) + \tilde{\alpha} h_t(x_i))} \right)
\]

\[
\Rightarrow 0 = \frac{\partial L(f_{t-1} + \tilde{\alpha} h_t)}{\tilde{\alpha}} = \sum_i e^{-g_i(f_{t-1}(x_i) + \tilde{\alpha} h_t(x_i))} (\sum_{i, h(x_i) \neq g_i} e^{-g_i(f_{t-1}(x_i))}) e^{\tilde{\alpha}_t} - \sum_{i, h(x_i) = g_i} e^{-g_i(f_{t-1}(x_i))} e^{-\tilde{\alpha}_t}
\]

In (iii) we got, that \( e^{-g_i(f_{t-1}(x_i))} \) is proportional to \( D_t(i) \), so we get

\[
0 = \epsilon_t e^{\tilde{\alpha}_t} - (1 - \epsilon_t) e^{-\tilde{\alpha}_t}
\]

\[
\Rightarrow \tilde{\alpha}_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) = \alpha_t
\]

Notes:
1.) One wants to minimize the expected error rate \( \text{error}(h_f) \) for \( h_f(x) := \arg \max_{g \in \mathbb{G}} f(x) \). For \( \mathbb{G} = \{ \pm 1 \} \) \( \text{err}(h_f) \) is

\[
\text{err}(h_f) = E_X (1 - P(h_f(x)|x))
\]

\[
= \int ([h_f(x) = 1]P(+1|x) + [h_f(x) = -1]P(-1|x))dP(x)
\]

But this functional is not a pleasant one to minimize, therefore we try to minimize

\[
\tilde{L}(f) := \int (e^{-f(x)} P(+1|x) + e^{+f(x)} P(-1|x))dP(x)
\]

Because of \( [[f(x) \leq 0]] \leq e^{-f(x)} \) and \( [[f(x) \geq 0]] \leq e^{f(x)} \) we have

\[
\text{error}(h_f) \leq \tilde{L}(f)
\]

2.) Even when \( \text{error}(h_f) = 0 \) (\( \Leftrightarrow \forall i : g_i f(x_i) > 0 \)) we have that

\[
L(f) = \frac{1}{N} \sum_i e^{-g_i f(x_i)} \neq 0
\]
so boosting doesn’t have to stop, if error($h_f$) = 0, because $L(f)$ could still be decreased further. In some experiments ([25]) boosting wasn’t stopped when training error zero was reached. There the test error could be lowered further, because the data were classified correctly with greater confidence.

3.) The formulation of AdaBoost as minimization of a functional, which is closely related to classification error, by gradient descent provides many generalization possibilities:

a) Consider other loss functions (Fig. 1.11). For example for noisy data AdaBoost penalizes hard-to-classify points too much, so one could choose loss functions $l(z)$, for $z$ denoting the margin, that are not so high near $z = -1$. Note, that the sigmoid function is monotonously increasing but not convex.

b) Another possibility to make boosting more resistant against noise is to change the loss function replacing the margin by a ”soft” margin by adding a regulation term ([23]):

$$L(f_t) = \frac{1}{N} \sum_i e^{-g_if_t(x_i)} + C\zeta_t(i)$$

where $C > 0$ and $\zeta_t(i)$ is a measure of mistrust in object $(x_i, g_i)$, e.g.

$$\zeta_t(i) = \sum_{s=1}^t \alpha_tD_t(i)$$

Here $\zeta_t(i)$ measures the influence of object $i$ to the final classification, which is the higher the ”noisier” object $i$ is.

c) Other methods consider noise by regarding boosting as optimization problems in terms of the margins and introduce slack variables similar as it is done with support vector machines (SVMs) ([4, 22, 15, 13, 27]).

**1.4.4 Convergence results**

This section provides some convergence results for general gradient descent in function space.

**Theorem:** (Convergence of gradient descent)
Set $h_t \in \mathcal{H}$. If $L : \mathcal{F} \to \mathbb{R}$ is
(i) lower bounded
(ii) and Lipschitz differentiable:
\[ \forall f, f' \in F : \| \nabla L(f) - \nabla L(f') \| \leq K \| f - f' \| \text{ for } \| f \| := \sqrt{\langle f, f \rangle} \]
(iii) and for the step size \( \alpha_t \) of the sequence \( f_0, f_1, \ldots \) we choose
\[ \alpha_t := \frac{-\nabla L(f_{t-1}), h_t}{K \| h_t \|^2} \]

Then either:
(i) The algorithm stops at round \( T + 1 \) with \( \langle -\nabla L(f_T), h_{T+1} \rangle \leq 0 \)
or
(ii) \( L(f_t) \) converges to a finite value \( L^* \):
\[ \lim_{t \to \infty} L(f_t) = L^* \]
and
\[ \lim_{t \to \infty} \langle \nabla L(f_t), h_{t+1} \rangle = 0. \]

Note: The theorem above just states that the algorithm converges or stops but nothing about the quality of the reached limit \( L^* \).

To prove the theorem we first need a general lemma:

**Lemma:** Let \( (U, \langle \cdot, \cdot \rangle) \) be an inner product space with norm \( \| f \| := \sqrt{\langle f, f \rangle} \) and \( L : U \to \mathbb{R} \) be differentiable and Lipschitz differentiable with Lipschitz constant \( K \).

Then:
\[ \forall \alpha > 0 \forall f, u \in U : L(f + \alpha u) \leq L(f) + \alpha \langle \nabla L(f), u \rangle + \frac{1}{2} K \alpha^2 \| u \|^2. \]
(Compare the right side with the Taylor expansion of order 2)

**Proof of the Lemma:** Using the chain rule we get
\[ L(f + \alpha u) = L(f) + \int_0^\alpha \langle \nabla L(f + \alpha'u), u \rangle \, d\alpha' \]
\[ \langle \nabla (L(f + \alpha'u), u) \rangle = \langle \nabla L(f + \alpha'u) - \nabla L(f), u \rangle + \langle \nabla L(f), u \rangle \]
\[ \leq \| \nabla L(f + \alpha'u) - \nabla L(f) \| \| u \| + (\nabla L(f), u) \text{ by Cauchy – Schwartz} \]
\[ \leq K \alpha' \| u \|^2 + \langle \nabla L(f), u \rangle \text{ by using the Lipschitz property} \]

Substituting in the integral above and integration leads to
\[ L(f + \alpha u) \leq L(f) + \alpha \langle \nabla L(f), u \rangle + \frac{1}{2} K \alpha^2 \| u \|^2. \]
\[ \square \]
Proof of the theorem:
(i) By definition the algorithm stops at round \( t \), if \( \langle -\nabla L(f_{t-1}), h_t \rangle \leq 0 \).
(ii) Suppose this is never the case. Then \( \alpha_t \geq 0 \) (assumption (iii)), which lets us (together with assumption (ii)) apply the lemma above:
\[
L(f_{t-1}) - L(f_t) = L(f_{t-1}) - L(f_{t-1} + \alpha_t h_t)
\geq -\alpha_t \langle \nabla L(f_{t-1}), h_t \rangle - \frac{1}{2} K \|h_t\|^2 \alpha_t^2
\]
Note: The greatest reduction \( L(f_{t-1}) - L(f_t) \) is achieved for our choice of \( \alpha_t \).
Substituting for \( \alpha_t \) above yields to
\[
L(f_{t-1}) - L(f_t) \geq \langle \nabla L(f_{t-1}), h_t \rangle - \frac{1}{2} K \|h_t\|^2 \alpha_t^2
\]
Therefore \( L(f_t) \leq L(f_{t-1}) \), therefore \( L(f_1), L(f_2), \ldots \) is a decreasing sequence of losses which must converge to a loss \( L^* \), because \( L \) is bounded below (assumption (i)). Thus also \( \lim_{t \to \infty} \langle \nabla L(f_t), h_{t+1} \rangle = 0 \).

Now we show conditions which ensure optimality of \( L^* \):

**Theorem:** (Conditions for optimality of gradient descent)
Assume the setting of the previous theorem and additionally:
(i) \( L \) is convex
(ii) The set \( \mathcal{H} \) of weak hypotheses is negation closed, i.e. 
\[
h \in \mathcal{H} \rightarrow -h \in \mathcal{H}
\]
(iii) On each round the algorithm finds a function \( h_t \in \mathcal{H} \) maximizing \( \langle -\nabla L(f_{t-1}), h_t \rangle \)
Then any accumulation point \( f^* \) of \( \{f_t\}_{t \geq 1} \) satisfies \( L(f^*) = \inf_{f \in \mathcal{F}} L(f) \).

To prove this theorem we use again a general lemma:

**Lemma:** Let \( (\mathcal{F}, \langle \cdot, \cdot \rangle) \) be an inner product space with norm \( \|f\| := \sqrt{\langle f, f \rangle} \) and \( L : \mathcal{F} \to \mathbb{R} \) be a differentiable, convex functional on \( \mathcal{F} \). Let \( M \) be any linear subspace of \( \mathcal{F} \) and denote \( M^\perp := \{v \in \mathcal{F} : \forall u \in M \langle v, u \rangle = 0\} \).
Then:
\[
f \in M \text{ and } \nabla L(f) \in M^\perp \Rightarrow L(f) = \inf_{u \in M} L(u).
\]
In words this means: if the gradient of a convex functional at a point of a linear subspace points out of it, then this point minimizes the functional.

**Proof of the lemma:** Choose an arbitrary \( u \in M \) and look at the convexity of \( L \):
\[
\forall \epsilon > 0 : L(f + \epsilon(u - f)) \leq (1 - \epsilon)L(f) + \epsilon L(u)
\]
\[ \forall \epsilon > 0 : \frac{L(f + \epsilon(u - f)) - L(f)}{\epsilon} \leq L(u) - L(f) \]

Now we take the limit \( \epsilon \to 0 \) and get

\[ \lim_{\epsilon \to 0} \frac{L(f + \epsilon(u - f)) - L(f)}{\epsilon} = \langle \nabla L(f), u - f \rangle \leq L(u) - L(f) \]

Now since both \( f \) and \( u \) are in \( M \) and \( M \) is a linear subspace, we have

\[ u - f \in M \]

Thus

\[ \nabla L(f) \in M^\perp \Rightarrow L(u) - L(f) \leq \langle \nabla L(f), u - f \rangle = 0 \]

Therefore for arbitrary \( u \in M \) we have \( L(u) - L(f) \leq 0 \) which means that

\[ L(f) = \inf_{u \in M} L(u). \]

\[ \square \]

**Proof of the theorem:** Let \( f^* \) be an accumulation point of \( \{f_t\}_{t \leq 1} \). Because of the theorem above

\[ \sup_{h \in \mathcal{H}} \langle \nabla L(f^*), h \rangle = 0. \]

Because of the negation closure we get

\[ \sup_{h \in \mathcal{H}} \langle \nabla L(f^*), -h \rangle = \sup_{h \in \mathcal{H}} \langle -\nabla L(f^*), h \rangle = 0. \]

Since the algorithm can always find the best \( h \in \mathcal{F} \) we get

\[ \nabla L(f^*) \in \mathcal{H}^\perp = \text{lin}(\mathcal{H})^\perp = \mathcal{F}^\perp. \]

Since \( f^* \in \mathcal{F} = \text{lin}(\mathcal{H}) \) we can apply the previous lemma and get

\[ L(f^*) = \inf_{f \in \mathcal{F}} L(f). \]

\[ \square \]

**Note:** The second theorem provides a choice of the step size, which - in contrast to \( \alpha_t \) for AdaBoost - is a minimum step size and therefore not adaptive. For exponential losses as used for AdaBoost and GrPloss these theorems give useful insights, but are not globally applicable, because the Lipschitz differentiability condition is only locally fulfilled and not globally on \( \mathcal{F} \) as it is required in the proofs. Therefore to get a choice for \( \alpha_t \) we will use a bound for a measure for the performance, similar as for AdaBoost. The minimizer of this bound will then be our step size \( \alpha_t \).
Chapter 2

GrPloss

In this chapter we make a generalization of the gradient-descent framework to the multiclass case. Then we apply this framework to minimize the exponential pseudo-loss. The choice for the step size $\alpha_t$ comes from a bound for a measure of performance, which we derive similar as it was done by Schapire and Singer in [26]. The algorithm is then simplified for the special case of decision stumps as base classifiers. The resulting algorithm is called GrPloss.

2.1 Generalization of gradient descent for multiclass problems

In this section we want to generalize the gradient descent framework in order to treat the multiclass case with confidence-rated base classifiers. More accurately we want to treat classifiers $f$ of the form $f : X \times G \to \mathbb{R}$. We denote the set of possible classifiers with $\mathcal{F}$.

2.1.1 Inner product

The first thing to do is to adapt the inner product:

First we state it in a general form: for two functions $f, h : X \times G \to \mathbb{R}$ an inner product $\langle \cdot, \cdot \rangle$ on $\mathbb{R}^{|G|}$ induces an inner product

$$
\langle f, h \rangle := \frac{1}{N} \sum_{i=1}^{N} \langle f(x_i), h(x_i) \rangle \text{ on } \mathcal{F}
$$

where

$$
f(x_i) = (f(x_i, 1), \ldots, f(x_i, |G|)) \text{ and } h(x_i) = (h(x_i, 1), \ldots, h(x_i, |G|)).
$$
We will use the square of the usual scalar product leading to the inner product \( \langle f, h \rangle := \frac{1}{N} \sum_{i=1}^{N} \sum_{g=1}^{G}|f(x_i, g)|h(x_i, g) \) on \( \mathcal{F} \).

Another plausible inner product would be \( \langle f, h \rangle = \frac{1}{N} \sum_i f(x_i, g_i)h(x_i, g_i) \). But this choice is only an inner product on \( \mathcal{L} \), because it ignores differences in the confidences of the wrong groups. Therefore it doesn’t seem to be a good choice.

### 2.1.2 Exponential loss-functions

We also have to choose a loss-function on \( \mathcal{F} \):

Similar to the previous chapter we define a cost function \( l : \mathbb{R}^{[G]} \times \mathbb{G} \to \mathbb{R}_{\geq 0} \) and define

\[
L(f) := E_{x,g}l(f(x), g) = E_x[E_g[l(f(x), g)] \quad \text{for} \quad f(x) = (f(x, 1), \ldots, f(x, |\mathbb{G}|))
\]

A loss-function corresponding to AdaBoost.M1 would be

\[
L(f) = \frac{1}{N} \sum_i e^{-f(x_i, g_i)}.
\]

For the algorithm GrPloss we choose the loss-function

\[
L(f) := \frac{1}{N} \sum_i e^{\text{pseudo-loss}(i)}
\]

where as for AdaBoost.M2

\[
\text{pseudo-loss}(i) := \frac{1}{2} \left( 1 - f(x_i, g_i) + \sum_{g \neq g_i} \frac{1}{|\mathbb{G}| - 1} f(x_i, g) \right)
\]

For notational reasons we define

\[
z_i := (x_i, g_i)
\]

and

\[
l(f, z_i) := \exp \left[ \frac{1}{2}(1 - f(x_i, g_i) + \sum_{g \neq g_i} \frac{1}{|\mathbb{G}| - 1} f(x_i, g)) \right].
\]

So we have

\[
L(f) = \frac{1}{N} \sum_i l(f, z_i)
\]

In the next section we will make the calculations not for the special choice of the pseudo-loss in the exponent but for cost functions of the following form:

\[
l(f, z_i) := \exp[v(f, z_i)] \quad \text{with} \quad v(f, z_i) = v_0 + \sum_g v_g(i)f(x_i, g)
\]

with \( v_g(i) \in \mathbb{R} \).

In the case of the exponential pseudo-loss we have

\[
v_0 = \frac{1}{2} \quad \text{and} \quad v_g = \begin{cases} -1/2 & \text{for } g = g_i \\ 1/(2(|\mathbb{G}| - 1)) & \text{for } g \neq g_i \end{cases}
\]
2.2 Gradient descent for the exponential pseudo-loss

In this section we will derive the algorithm GrPloss. We first consider slightly more general exponential loss-functions. Based on the gradient descent framework we derive a gradient descent algorithm for these loss-functions in a straightforward way in subsection 2.2.1. In contrast to the general framework we can additionally derive a simple update-rule for the sampling distribution as it exists for the original boosting algorithms. Gradient descent does not provide a special choice for $\alpha_t$. Therefore in subsection 2.2.2 we define the pseudo-loss error and derive $\alpha_t$ by minimization of a bound for the pseudo-loss error. Finally we state the algorithm for decision-stumps as base classifiers in subsection 2.2.3.

2.2.1 Gradient descent for exponential loss-functions

Theorem: For the inner product

$$\langle f, h \rangle := \frac{1}{N} \sum_{i=1}^{N} \sum_{g=1}^{G} f(x_i, g) h(x_i, g)$$

on $\mathcal{F}$ and loss-functions of the form

$$L(f) = \frac{1}{N} \sum_i l(f, z_i) = \frac{1}{N} \sum_i \exp \left[ v(f, z_i) \right] \quad \text{with} \quad v(f, z_i) = v_0 + \sum_g v_g(i) f(x_i, g)$$

gradient descent leads to the algorithm in figure 2.1.

---

**Input:** Learning set $\mathcal{L}$, maximum number of boosting rounds $T$

**Initialisation:** $f_0 := 0$, $t := 1$, $\forall i : D_1(i) := \frac{1}{N}$.

**Loop:** For $t = 1, \ldots, T$ do

- $h_t = \arg \min_h \sum_i D_t(i) v(h, z_i)$
- If $\sum_i D_t(i) v(h_t, z_i) \geq v_0 : T := t - 1$, goto output.
- Choose $\alpha_t$.
- Update $f_t = f_{t-1} + \alpha_t h_t$ and $D_{t+1}(i) = \frac{1}{Z_t} D_t(i) l(\alpha_t h_t, z_i)$

**Output:** $f_T$, $L(f_T)$

---

**Figure 2.1:** Gradient descent for exponential loss functions

**Proof:** The proof basically consists of 3 steps: the calculation of the gradient, the choice for $h_t$ together with the stopping criterion and the update rule for the sampling distribution.
(i) Calculation of the gradient:
The gradient is now a function
\[ \nabla L(f) : \mathbb{X} \times \mathbb{G} \to \mathbb{R} \]
defined by
\[
\nabla L(f)(x, g) := \lim_{k \to 0} \frac{L(f + k1_{x,g}) - L(f)}{k}
\]
for
\[
1_{x,g}(x', g') = \begin{cases} 
1 & (x, g) = (x', g') \\
0 & (x, g) \neq (x', g')
\end{cases}
\]
Case 1: for \( x \neq x_i \):
\[
\nabla L(f)(x, g) = \lim_{k \to 0} \frac{L(f + k1_{x_i,g}) - L(f)}{k} = 0.
\]
Case 2: \( x = x_i \):
\[
L(f + k1_{x_i,g}) = \frac{1}{N} \sum_i \exp \left[ v_0 + \sum_{g'} v_{g'}(i)(f(x_i, g') + k1_{x_i,g}(x_i, g')) \right]
= \frac{1}{N} \exp \left[ v_0 + \sum_{g'} v_{g'}(i)f(x_i, g') + kv_g(i) \right]
= \frac{1}{N} l(f, z_i)e^{kv_g(i)}
\]
Substitution in the definition of \( \nabla L(f) \) leads to
\[
\nabla L(f)(x_i, g) = \lim_{k \to 0} \frac{l(f, z_i)e^{kv_g(i)} - l(f, z_i)}{k} = l(f, z_i)v_g(i)
\]
Thus
\[
\nabla L(f)(x, g) = \begin{cases} 
0 & x \neq x_i \\
l(f, z_i)v_g(i) & x = x_i
\end{cases} \tag{2.1}
\]

(ii) Now we proof the choice of the base classifier \( h_t \) and the stopping criterion:
\[
h_t = \arg \max_h (-\nabla L(f_{t-1}), h)
= \arg \min_h \nabla L(f_{t-1}), h)
\]
\[
= \arg \min_h \sum_i \sum_g \nabla L(f_{t-1})(x_i, g)h(x_i, g)
\]
\[
= \arg \min_h \sum_i \sum_g l(f_{t-1}, z_i)v_g(i)h(x_i, g)
\]
\[
= \arg \min_h \sum_i \left( l(f_{t-1}, z_i)\left( \sum_g v_g(i)h(x_i, g) + v_0 \right) - v_0 \right)
\]
\[
= \arg \min_h \sum_i l(f_{t-1}, z_i)v(h, z_i)
\]
where we used equation 2.1 for the fourth equality. Thus
\[
h_t = \arg \min_h \sum_i D_t(i)v(h, z_i) \tag{2.2}
\]
for
\[ D_t(i) := \frac{l(f_{t-1}, z_i)}{\sum l(f_{t-1}, z'_i)} \]
which is just the choice of \( h_t \) of the algorithm. And also the stopping criterion is the same, because from the previous equations we can see, that
\[ \langle -\nabla L(f_{t-1}), h_t \rangle \leq 0 \iff \sum_i D_t(i)v(h, z_i) \geq \sum_i D_t(i)v_0 = v_0. \]

### (iii) Update rule:

Now we show that we can calculate an update rule for the sampling distribution \( D_t \). Note that in the gradient descent framework this is in general not possible.

\[
D_{t+1}(i) = \frac{l(f_t, z_i)}{Z'_t} = \frac{l((f_{t-1} + \alpha_t h_t)(z_i))}{Z'_t} = \frac{1}{Z'_t} \exp \left[ v(f, z_i) + v(\alpha_t h_t, z_i) \right] = \frac{1}{Z'_t} l(f_t, z_i) l(\alpha_t h_t, z_i) = \frac{1}{Z'_t Z'_{t-1}} D_t(i) l(\alpha_t h_t, z_i)
\]

where we used 2.3 for the first and the last step. Therefore we get
\[
D_{t+1}(i) = \frac{1}{Z_t} D_t(i) l(\alpha_t h_t, z_i)
\]

Note, that the stopping criterion came from the convergence results of subsection 1.4.4, which used the global Lipschitz differentiability, which is not fulfilled in this case. In the next subsection we will specify the choice of \( \alpha_t \) and also get an appropriate stopping criterion, which will appear to be the same criterion as this one.

### 2.2.2 Choice of \( \alpha_t \) and resulting algorithm GrPloss

Now we consider the special case of the exponential pseudo-loss. To simplify notation we omit the constant term of the pseudo-loss and define
\[
u(f, z_i) := \frac{1}{2} (f(x_i, g_i) - \sum_{g \neq g_i} \frac{1}{|G| - 1} f(x_i, g))
\]
The update-rule 2.4 has the form
\[
D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp \left[ \frac{\alpha_t}{2} (1 - h_t(x_i, g_i) + \sum_{g \neq g_i} \frac{1}{|G| - 1} h_t(x_i, g)) \right].
\]

Of course, this is not the same update rule as the one used in section 1.4.4, because we used the exponential pseudo-loss instead of the exponential loss. However, we can still compare the two update rules, as they are very similar.

\[
D_{t+1}(i) = \frac{1}{Z_t} D_t(i) e^{-\alpha_t u(h_t, z_i)} = \frac{1}{Z_t} D_t(i) \exp \left[ -\frac{\alpha_t h_t(x_i, g_i)}{2} \prod_{g \neq g_i} \exp \left[ \frac{\alpha_t h_t(x_i, g)}{2(|G| - 1)^t} \right] \right]
\]
The last line shows a big difference to AdaBoost.M2. Here the weights of the group terms are multiplied, where for AdaBoost.M2 the group terms are summed up. We get for 2.2:

$$h_t = \arg \min_h \sum_i D_t(i) \frac{1}{2} \left( 1 - h(x_i, g_i) + \sum_{g \neq g_i} \frac{1}{|G| - 1} h(x_i, g) \right)$$

As for AdaBoost.M2 the base classifier should minimize the weighted pseudo-loss. Now we define a new error measure:

**Definition:** A classifier $f : X \times G \rightarrow \mathbb{R}$ makes a pseudo-loss error in classifying an object $x$ coming from group $k$, if

$$f(x, k) < \frac{1}{|G| - 1} \sum_{g \neq k} f(x, g).$$

The corresponding training error rate is denoted by $plerr$:

$$plerr := \frac{1}{N} \sum_{i=1}^{N} \left[ f(x_i, g_i) < \frac{1}{|G| - 1} \sum_{g \neq g_i} f(x_i, g) \right].$$

The following theorem states, that the algorithm GrPloss minimizes the pseudo-loss error under some weak conditions.

**Theorem:** If for all base classifiers $h_t$ of the algorithm given in figure 2.2.2

$$U_t := \sum_i D_t(i) \left( h_t(x_i, g_i) - \frac{1}{|G| - 1} \sum_{g \neq g_i} h_t(x_i, g) \right) > \delta$$

holds for $\delta > 0$ then the pseudo-loss error of the training set fulfills

$$plerr \leq \prod_{t=1}^{T} \sqrt{1 - U_t^2} \leq e^{-\delta T}.$$ 

The theorem provides a natural stopping criterion, which stops the algorithm, if $U_t \leq 0$ holds.

**Proof:**
(i) Similar as in [26] we first **bound $plerr$ by the product of the normalization constants**

$$plerr \leq \prod_{t=1}^{T} Z_t \quad \text{(2.5)}$$

To prove 2.5 we first notice, that

$$f(x_i, g_i) \geq \frac{1}{|G| - 1} \sum_{g \neq g_i} f(x_i, g) \iff e^{-u(f, z_i)} \in (0, 1]$$

$$f(x_i, g_i) \leq \frac{1}{|G| - 1} \sum_{g \neq g_i} f(x_i, g) \iff e^{-u(f, z_i)} \geq 1$$
**Input:** learning set $\mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in X, g_i \in G\}$, weak classifier of the form $h : X \times G \rightarrow [0, 1]$. Optionally $T$: maximal number of boosting rounds

**Initialization:** $D_1(i) = \frac{1}{N}$.

For $t = 1, \ldots, T$:

- Train the weak classifier $h_t$ with distribution $D_t$, where $h_t$ should maximize
  \[ U_t = \sum_i D_t(i) \left( h_t(x_i, g_i) - \frac{1}{|G| - 1} \sum_{g \neq g_i} h_t(x_i, g) \right) \]
- If $U_t \leq 0$: goto output with $T := t - 1$
- Set $\alpha_t = \ln \left( \frac{1 + U_t}{1 - U_t} \right)$.
- Update $D$:
  \[ D_{t+1}(i) = \frac{1}{Z_t} D_t(i) e^{-\alpha_t u(h_t, z_i)} \]
  where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ is a distribution)

**Output:** final classifier $H(x)$:

\[ H(x) = \arg \max_{g \in G} f(x, g) = \arg \max_{g \in G} \left( \sum_{t=1}^T \alpha_t h_t(x, g) \right) \]

---

**Figure 2.2:** Algorithm GrPloss

Therefore

\[ plerr \leq \frac{1}{N} \sum_i e^{-u(f_T, z_i)}. \] (2.6)
Now we unravel the update-rule

\[
D_{T+1}(i) = \frac{1}{Z_T} e^{-\alpha_T u(h_T, z_i)} D_T(i)
\]

\[
= \frac{1}{Z_T Z_{T-1}} e^{-\alpha_T u(h_T, z_i)} e^{-\alpha_{T-1} u(h_{T-1}, z_i)} D_{T-1}(i)
\]

\[
= \ldots = D_1(i) \prod_{t=1}^{T} e^{-\alpha_t u(h_t, z_i)} \frac{1}{Z_t}
\]

\[
= \frac{1}{N} \exp \left( - \sum_{t=1}^{T} \alpha_t u(h_t, z_i) \right) \prod_{t=1}^{T} \frac{1}{Z_t}
\]

\[
= \frac{1}{N} e^{-u(f_T, z_i)} \prod_{t=1}^{T} \frac{1}{Z_t}
\]

where the last equation uses the linearity of \( u \). Since

\[
1 = \sum_i D_{T+1}(i) = \sum_i \frac{1}{N} e^{-u(f_T, z_i)} \prod_{t=1}^{T} \frac{1}{Z_t}
\]

we get 2.5 by using 2.6 and the equation above

\[
\text{plerr} \leq \frac{1}{N} \sum_i e^{-u(f_T, z_i)} = \prod_{t=1}^{T} Z_t
\]

(ii) Derivation of \( \alpha_t \):

Now we derive \( \alpha_t \) by minimizing the upper bound 2.5. Now we plug the definition of \( Z_t \) in 2.5

\[
\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left( \sum_i D_t(i) e^{-\alpha_t u(h_t, z_i)} \right)
\]

Now we get an upper bound for this product using the convexity of the function \( e^{-\alpha_t u} \) between -1/2 and +1/2 (since \( u \in [-1/2, +1/2] \)) for positive \( \alpha_t \).

\[
\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} \left( \sum_i D_t(i) \left( \left( \frac{1}{2} + u(h_t, z_i) \right) e^{-\frac{\alpha_t}{2}} + \left( \frac{1}{2} - u(h_t, z_i) \right) e^{\frac{\alpha_t}{2}} \right) \right) \quad (2.7)
\]

Since each \( \alpha_t \) occurs in exactly one factor of this product we can minimize each factor separately by setting the first derivative with respect to \( \alpha_t \) to zero.

\[
\sum_i D_t(i) \left( \frac{1}{2} + u(h_t, z_i) \right) e^{-\frac{\alpha_t}{2}} = \sum_i D_t(i) \left( \frac{1}{2} - u(h_t, z_i) \right) e^{\frac{\alpha_t}{2}}
\]

\[
\Leftrightarrow \alpha_t = \ln \left( \frac{1 + U_t}{1 - U_t} \right) \quad \text{with} \quad U_t := 2 \sum_i D_t(i) u(h_t, z_i) \in [-1, 1].
\]

(iii) Plugging \( \alpha_t \) in:
Now we substitute $\alpha_t$ back in 2.7.

\[
\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} \left( \sum_i D_t(i) \left( \frac{1}{2} + u(h_t, z_i) \right) \sqrt{\frac{1 - U_t}{1 + U_t}} + \left( \frac{1}{2} - u(h_t, z_i) \right) \sqrt{\frac{1 + U_t}{1 - U_t}} \right)
\]

\[
= \prod_{t=1}^{T} \frac{1}{2} \left( \sqrt{\frac{1 - U_t}{1 + U_t}} + \sqrt{\frac{1 + U_t}{1 - U_t}} \right) + U_t \left( \sqrt{\frac{1 - U_t}{1 + U_t}} - \sqrt{\frac{1 + U_t}{1 - U_t}} \right)
\]

\[
= \prod_{t=1}^{T} \frac{1}{2} \sqrt{\frac{1 - U_t^2}{1 + U_t^2}} \left( [1 - U_t + 1 + U_t] + U_t[1 - U_t - (1 + U_t)] \right)
\]

\[
= \prod_{t=1}^{T} \sqrt{1 - U_t^2}
\]

\[
\leq \exp \left( \sum_{t=1}^{T} -U_t \right)
\]

where for the last inequality we used $\sqrt{1+x} \leq (1+x) \leq e^x$ for $x \leq 0$.

Now if each classifier $h_t$ is strong enough that $U_t$ is greater than zero, i.e. if $\forall t: U_t \geq \delta$, then together with 2.5 we finally get

\[
\text{perr} \leq \prod_{t=1}^{T} \sqrt{1 - U_t^2} \leq e^{-\delta T}.
\]

\[\square\]

**Note:** 1.) Now we want to investigate our choice of $\alpha_t$ for classifiers which do not provide confidences, i.e. $h_t(x_i, g)$ is 1 for exactly one group $g$ and zero otherwise. Then we get with $err_t$ being the weighted training error of $h_t$:

\[
\alpha_t = \ln \left( \frac{1 + \sum_i D_t(i) \left( h_t(x_i, g_i) - \frac{1}{|G| - 1} \sum_{g \neq g_i} h_t(x_i, g) \right) }{1 - \sum_i D_t(i) \left( h_t(x_i, g_i) - \frac{1}{|G| - 1} \sum_{g \neq g_i} h_t(x_i, g) \right) } \right)
\]

\[
= \ln \left( \frac{1 + (1 - err_t) - \frac{err_t}{|G| - 1} }{1 - (1 - err_t) + \frac{err_t}{|G| - 1} } \right)
\]

\[
= \ln \left( -1 + 2 \frac{(|G| - 1)}{err_t |G|} \right)
\]

Now $\alpha_t$ has the following 3 properties:

\[
err_t = \frac{|G| - 1}{|G|} \Rightarrow \alpha_t = 0
\]

\[
err_t \rightarrow 0 \Rightarrow \alpha_t \rightarrow +\infty
\]

\[
err_t \rightarrow 1 \Rightarrow \alpha_t \rightarrow \ln \left( \frac{|G| - 1}{|G|} \right)
\]

Therefore $\alpha_t$ is lower bounded for $|G| > 2$, if the training error reaches 1, but not for $|G| = 2$. Since we stop, if $U_t \leq 0 \Leftrightarrow err_t \leq (|G| - 1)/|G| \Leftrightarrow \alpha_t \leq 0$ and
we also assumed \( \alpha_t > 0 \) for the derivation the last property is interesting but doesn’t matter in practice.

2.) There is quite a difference in the update of the sampling distributions between AdaBoost.M2 and GrPloss. For GrPloss it has a multiplicative form

\[
D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp \left( \frac{\alpha_t h_t(x_i, g_i)}{2} \right) \prod_{g \neq g_i} \exp \left( \frac{\alpha_t h_t(x_i, g) - \alpha_t h_t(x_i, g_i)}{2(|G| - 1)} \right)
\]

where for AdaBoost.M2 it is more of an additive form

\[
D_{t+1}(i) = \frac{1}{Z_t} \sum_{g \neq g_i} w_{t,g} e^{-\alpha_t (1 - h_t(x_i, g_i) + h_t(x_i, g))}
\]

### 2.2.3 GrPloss for decision stumps

So far we have considered classifiers of the form \( h : \mathbb{X} \times G \to [0, 1] \), which also include decision stumps. However decision stumps \( h \) have the additional property

\[
\sum_{g \in G} h(x, g) = 1
\]

which we did not use in the previous section for the derivation of \( \alpha_t \). For data from the training set this property can also be written as

\[
\sum_{g \neq g_i} \frac{1}{|G| - 1} h(x_i, g) = \frac{1}{|G| - 1} - \frac{1}{|G| - 1} h(x_i, g_i)
\]

(2.8)

Now we want to use the normalization property to minimize a tighter bound for the pseudo-loss error.

**Bounding the pseudo-loss error and choice of \( \alpha_t \):** To get \( \alpha_t \) we can start with

\[
p_{\text{err}} \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left( \sum_{i} D_t(i) e^{-\alpha_t u(h_t, z_i)} \right)
\]

which was derived in part (i) of the proof of the previous subsection. First we simplify \( u(h, z_i) \) using 2.8 and get

\[
u(h, z_i) = \frac{1}{2} (h(x_i, g_i) - \sum_{g \neq g_i} \frac{1}{|G| - 1} h(x_i, g)) = \frac{1}{2} \left( \frac{|G|}{|G| - 1} h(x_i, g_i) - \frac{1}{|G| - 1} \right).
\]

In contrast to the previous section \( u(h, z_i) \in \left[ -\frac{1}{2(|G| - 1)}, \frac{1}{2} \right] \) for \( h(x_i, g_i) \in [0, 1] \), which we will take into account for the convexity argument:

\[
p_{\text{err}} \leq \prod_{t=1}^{T} \sum_{i=1}^{N} D_t(i) \left( h(x_i, g_i) e^{-\alpha_t/2} + (1 - h_t(x_i, g_i)) e^{\alpha_t/(2(|G| - 1))} \right)
\]

(2.9)

Setting the first derivative with respect to \( \alpha_t \) to zero leads us to

\[
\sum_{i=1}^{N} D_t(i) \frac{h(x_i, g_i)}{2} e^{-\alpha_t/2} = \sum_{i=1}^{N} D_t(i) \frac{1 - h_t(x_i, g_i)}{2(|G| - 1)} e^{\alpha_t/(2(|G| - 1))}
\]
\( \Rightarrow \alpha_t = \frac{2(|G| - 1)}{|G|} \ln \left( \frac{(|G| - 1) r_t}{1 - r_t} \right) \)

where we defined

\[ r_t := \sum_{i=1}^{N} D_t(i) h_t(x_i, g_i) \]

Now we plug \( \alpha_t \) in 2.9 and get

\[ plerr \leq \prod_{t=1}^{T} \left( r_t \left( \frac{1 - r_t}{r_t(|G| - 1)} \right)^{(|G| - 1)/|G|} + (1 - r_t) \left( \frac{r_t(|G| - 1)}{1 - r_t} \right)^{(1)/|G|} \right) \]

**Stopping criterion:** For \( r_t = 1/|G| \) the corresponding factor is 1. Looking at the first and second derivative of the bound one can easily verify that it has a unique maximum at \( r_t = 1/|G| \). Therefore the bound drops as long as \( r_t > 1/|G| \), so we got a natural stopping criterion, which stops, if \( r_t \leq 1/|G| \).

One can easily verify, that it is equivalent to the old one.

\[ \sum_{g \in G} h(x, g) = 1 \Rightarrow \left( r_t \leq \frac{1}{|G|} \Leftrightarrow U_t \leq 0 \right) \]

Note that we have assumed \( \alpha_t > 0 \), so the case \( r_t < 1/|G| \) is not allowed anyway.

**Update rule:** Using \( \sum_{g} h(x, g) = 1 \) leads to the simple update rule

\[ D_{t+1}(i) = \frac{D_t(i)}{Z_t} \exp \left[ -\alpha_t \frac{1}{2} \left( \frac{|G|}{|G| - 1} h_t(x_i, g_i) - \frac{1}{|G| - 1} \right) \right] \]

\[ = \frac{D_t(i)}{Z_t} \left( \frac{1 - r_t}{r_t(|G| - 1)} \right)^{h_t(x_i, g_i) - 1/|G|} \]

\[ = \frac{D_t(i)}{Z_t} e^{-\tilde{\alpha}_t (h_t(x_i, g_i) - 1/|G|)} \]

for

\[ \tilde{\alpha}_t := \ln \left( \frac{(|G| - 1) r_t}{1 - r_t} \right) \]

Also the goal of the base classifier can be simplified, because maximizing \( U_t \) is equivalent to maximizing \( r_t \).

The resulting algorithm is shown in figure 2.3.

**Notes:** 1.) We will see later, that the resulting algorithm is a special case of the algorithm BoostMA of the next chapter (with \( \pi_g = 1/|G| \)).

2.) The new choice for \( \alpha_t \), which we obtained with another convexity bound is different to the old one. To investigate this we can use

\[ U_t = \sum_{i=1}^{N} D_t(i) \left( \frac{|G|}{|G| - 1} h_t(x_i, g_i) - \frac{1}{|G| - 1} \right) = \frac{1}{|G| - 1} (|G| r_t - 1). \]

and simplify

\[ \alpha_{t,old} = \frac{1}{2} \ln \left( \frac{1 + U_t}{1 - U_t} \right) = \frac{1}{2} \ln \left( \frac{1 + r_t - |G|/2}{1 - r_t} \right). \]
Input: learning set \( \mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in \mathcal{X}, g_i \in \mathcal{G}\} \),
\( \mathcal{G} = \{1, \ldots, \lvert \mathcal{G} \rvert\} \) weak classifier of the form \( h : \mathcal{X} \times \mathcal{G} \rightarrow [0, 1] \).
Optionally \( T \): number of boosting rounds

Initialization: \( D_1(i) = \frac{1}{N} \).

For \( t = 1, \ldots, T \):
- Train the weak classifier \( h_t \) with distribution \( D_t \), where \( h_t \) should maximize
  \[ r_t = \sum_i D_t(i)h_t(x_i, g_i) \]
- If \( r_t \leq \frac{1}{\lvert \mathcal{G} \rvert} \): goto output with \( T := t - 1 \)
- Set
  \[ \alpha_t = \ln \left( \frac{(\lvert \mathcal{G} \rvert - 1)r_t}{1 - r_t} \right) . \]
- Update \( D \):
  \[ D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\alpha_t(h_t(x_i, g_i) - 1/\lvert \mathcal{G} \rvert)} . \]
  where \( Z_t \) is a normalization factor (chosen so that \( D_{t+1} \) is a distribution)

Output: Normalize \( \alpha_1, \ldots, \alpha_T \) and set the final classifier \( H(x) \):

\[ H(x) = \arg \max_{g \in \mathcal{G}} f(x, g) = \arg \max_{g \in \mathcal{G}} \left( \sum_{t=1}^{T} \alpha_t h_t(x, g) \right) \]

Figure 2.3: Algorithm GrPloss for decision stumps
Chapter 3

BoostMA

The algorithm BoostMA comes from the idea to modify the choice for \( \alpha_t \) in AdaBoost.M1 in order to make it work for weak base classifiers. It starts with a simple ad-hoc choice for \( \alpha_t \) in section 3.1. In section 3.2 we improve the analysis by building in confidence-rated predictions and choosing \( \alpha_t \) by the minimization of a bound for a measure of performance, which we call the maxgroup error.

3.1 Original ad-hoc derivation: AdaBoost.M1W

As already shown in section 1.2.2 AdaBoost.M1 does not work for weak base classifiers as decision stumps. AdaBoost.M2 and GrPloss solved this problem, but in a quite complicated way by introducing the pseudo-loss, which should be minimized by the base classifier. We instead wanted to find a simple modification of AdaBoost.M1 which makes it work for weak base classifiers.

We start by looking at the combination step

\[
H(x) = \arg\max_{g \in G} \left( \sum_{t=1}^{T} \alpha_t \left[ [h_t(x) = g] \right] \right)
\]

There each base classifier \( h_t \) gives a vote for the group \( h_t(x) \), to which it would assign \( x \). The votings are weighted by the factor \( \alpha_t \), which is bigger, if the base classifier is better. The key point for the modification is the property of the algorithm AdaBoost.M1, that

\[
\alpha_t = \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \geq 0 \Leftrightarrow \epsilon_t \leq \frac{1}{2}.
\] (3.1)

If the error rate is bigger than 1/2, the weight \( \alpha_t \) gets negative and the ensemble classifier \( H \) does the opposite of what the base classifier \( h_t \) proposes. In AdaBoost.M1 \( \alpha_t \) does not get negative, because in the derivation of the bound for the training error of AdaBoost.M1 \( \alpha_t \) is assumed to be positive and AdaBoost.M1 stops, if \( \epsilon_t \geq \frac{1}{2} \) (and therefore \( \alpha_t \leq 0 \) because of (3.1)). Nevertheless this was the starting point for our modification.
If one has a base classifier \( h \) with error rate greater than \( 1/2 \) but better than random guessing (which has an expected error rate of \( 1 - 1/|G| \)), the ensemble classifier \( H \) should not do the opposite of what the base classifier \( h \) proposes. Therefore we wanted to find a choice for \( \alpha_t(\epsilon_t) \) such that
\[
\alpha_t \geq 0 \iff \epsilon_t \leq 1 - \frac{1}{|G|}.
\] (3.2)

To derive \( \alpha_t(\epsilon_t) \) we assumed \( \alpha_t(\epsilon_t) \) to be basically of the same form as \( \alpha_t \) in AdaBoost.M1, so we set
\[
\alpha_t(\epsilon_t) = \ln \left( \frac{a_n \epsilon_t + b_n}{a_d \epsilon_t + b_d} \right) =: \ln (z(\epsilon_t))
\] (3.3)
where \( n \) and \( d \) are subscripts for the nominator and denominator respectively. Then we wanted \( \alpha_t \) to fulfill condition (3.2) and two additional conditions, which are also fulfilled by \( \alpha_t(\epsilon_t) \) of AdaBoost.M1:
\[
\begin{align*}
\alpha_t(1 - \frac{1}{G}) &= 0 \\
\alpha_t &\to -\infty \text{ for } \epsilon_t \to 1 \\
\alpha_t &\to \infty \text{ for } \epsilon_t \to 0
\end{align*}
\]
For \( z(\epsilon_t) \) this means that
\[
\begin{align*}
z(1 - \frac{1}{G}) &= 1 \\
z(1) &= 0 \\
\epsilon_t = 0 &\Rightarrow \text{ Denominator of } z = 0.
\end{align*}
\]
These conditions directly results in the following conditions for the 4 constants
\[
\begin{align*}
a_n \left(1 - \frac{1}{G}\right) + b_n &= a_d \left(1 - \frac{1}{G}\right) + b_d \\
b_n &= -a_n \\
b_d &= 0
\end{align*}
\]
Substitution for \( b_n \) and \( b_d \) in the first equation and solving for \( a_d \) leads to
\[
a_n = a_d(1 - G).
\]
Now we substitute the constants \( a_n, b_n \) and \( b_d \) in (3.3), \( a_d \) gets cancelled, and we get
\[
\alpha_t = \ln \left( \frac{(G - 1)(1 - \epsilon_t)}{\epsilon_t} \right). \tag{3.4}
\]
Since the rest of the algorithm AdaBoost.M1 is left untouched we can already write down the algorithm, which we call AdaBoost.M1W (W..Weak), in figure 3.1.

Note, that this is just an ad-hoc modification without any proof for a decrease in the error rate. We also don’t have a stopping criterion any more. An intuitive ad-hoc stopping criterion would stop at the first boosting round \( t_{stop} \), where
\[
\epsilon_{t_{stop}} \geq 1 - \frac{1}{|G|}.
\]
CHAPTER 3. BOOSTMA

**Input:** learning set $L = \{(x_1, g_1), \ldots, (x_N, g_N); x_i \in X, g_i \in G\}$, 
$G = \{1, \ldots, |G|\}$, classifier of the form $h : X \rightarrow G$. 
$T$: number of boosting rounds

**Initialization:** $D_1(i) = \frac{1}{N}$.

For $t = 1, \ldots, T$:

- Train the weak classifier $h_t$ with distribution $D_t$, where $h_t$ should minimize the weighted error rate
  \[ \epsilon_t = \sum_i D_t(i)[h_t(x_i) \neq g_i] \]

- Set
  \[ \alpha_t = \ln \left( \frac{(G-1)(1-\epsilon_t)}{\epsilon_t} \right). \]

- Update D:
  \[ D_{t+1}(i) = D_t(i)e^{-\alpha_t(1-[h_t(x_i)\neq g_i])}/Z_t. \]
  where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ is a distribution)

**Output:** Set the final classifier $H(x)$:

\[ H(x) = \arg \max_{g \in G} \left( \sum_{t=1}^T \alpha_t[h_t(x) = g] \right) \]

---

Figure 3.1: Algorithm AdaBoost.M1W

We made some experiments with data sets described in chapter 5, where we stopped after 2000 boosting rounds. The main question to be answered by the experiments is, if AdaBoost.M1W is able to boost base classifiers with error rates greater than 1/2. The answer to this question is yes. For the 8 datasets, where the error rate of a single decision stump exceeds 1/2, AdaBoost.M1 failed, because for all 8 datasets it couldn’t decrease the training error rate at all, whereas AdaBoost.M1W worked for all 8 datasets (Table 5.1 and Fig.3.2). One can also see, that the stopping criterion is reasonable ([9]).

### 3.2 BoostMA

In this section we want to improve the analysis by building in confidence-rated predictions and group proportions in a more systematic way. Now we don’t compare the base classifier with random guessing but with the maxgroup rule that assigns an object to the most frequent group $g$ in the training set

\[ g = \arg \max_{\hat{g} \in \hat{G}} \frac{N_{\hat{g}}}{N}. \]
Figure 3.2: Training (solid) and test error (dash-dotted) of AdaBoost.M1W dependent on the number of boosting rounds. The vertical line denotes $t_{stop}$.

This rule has a training error rate of $1 - \frac{N_g}{N}$, which can be much lower than random guessing, if there is one very frequent group. Since the maxgroup-rule is a better rule than random guessing and also doesn’t use information about the measurements of an object it is a more natural rule to compare with. In contrast to AdaBoost.M1 and the approach in the previous section we also want to use the confidences of the base classifiers. Therefore it seems natural to investigate a modification, where the update of the sampling distribution has the form

$$D_{t+1}(i) = D_t(i) e^{-\alpha_t(h_t(x_i,g_i)-\pi_g)} Z_t.$$

where $\pi_g$ is the performance of the maxgroup-rule. The original idea was to choose $\frac{N_g}{N}$ for $\pi_g$. At the end of this chapter we will set

$$\pi_g := \sum_{g \in G} \left( \frac{N_g}{N} \right)^2$$

and justify this setting. But up to that point we let the choice of $\pi_g$ open and just require, that $\pi_g \in (0, 1)$. As for GrPloss the modus operandi consists of finding an upper bound for a measure of performance for the training set and minimizing the bound with respect to $\alpha$. This measure of performance is the maxgroup error $\text{maxerr}$, which is the proportion of examples, where the classifier performs worse than the maxgroup rule.
**Definition:** A classifier $f : \mathbb{X} \times G \rightarrow [0, 1]$ makes a maxgroup error in classifying an object $x$ coming from group $k$, if

$$f(x, k) < \pi_g.$$  

The corresponding estimate of the expected maxgroup error using the training set is called $mxerr$:

$$mxerr := \sum_{i=1}^{N} \mathbb{I}[f(x_i, g_i) < \pi_g]$$

**Note:** When we use decision stumps as base classifiers we have the property $h(x, g) \in [0, 1]$. By normalizing $\alpha_1, \ldots, \alpha_T$, so that they sum to one, we ensure $f(x, g) \in [0, 1]$.

1.) **Bounding $mxerr$ in terms of the normalization constants $Z_t$:** Similar to the calculations used to bound the error rate of AdaBoost we begin by bounding $mxerr$ in terms of the normalization constants $Z_t$: We have

$$1 = \sum_i D_{t+1}(i) = \sum_i D_t(i) \frac{e^{-\alpha_t(h_t(x_i, g_i) - \pi_g)}}{Z_t} = \ldots$$

$$= \prod_s Z_s \frac{1}{N} \sum_1^t \prod_{s=1}^t e^{-\alpha_s(h_s(x_i, g_i) - \pi_g)} = \prod_s Z_s \frac{1}{N} \sum_i e^{-(f(x_i, g_i) - \pi_g \sum \alpha_s)}$$

So we get

$$\prod_t Z_t = \frac{1}{N} \sum I \frac{e^{-(f(x_i, g_i) - \pi_g \sum \alpha_t)}}{Z_t}$$

Using

$$\frac{f(x_i, g_i) - \pi_g}{\sum \alpha_t} < \pi_g \Rightarrow e^{-(f(x_i, g_i) - \pi_g \sum \alpha_t)} > 1$$

we get

$$mxerr \leq \prod_t Z_t$$  \(3.5\)

2.) **Choice of $\alpha_t$:** Now we bound $\prod Z_t$ and then we minimize it, which leads us to the choice of $\alpha_t$. First we use the definition of $Z_t$ and get

$$\prod Z_t = \prod \left( \sum_i D_t(i) e^{-\alpha_t(h_t(x_i, g_i) - \pi_g)} \right).$$  \(3.6\)

Now we use the convexity of $e^{-\alpha_t(h_t(x_i, g_i) - \pi_g)}$ for $h_t(x_i, g_i)$ between 0 and 1 (so $h_t(x_i, g_i) - \pi_g \in [-\pi_g, 1 - \pi_g]$) and get

$$mxerr \leq \prod_t \sum D_t(i) \left( h_t(x_i, g_i) e^{-\alpha_t(1 - \pi_g)} + (1 - h_t(x_i, g_i)) e^{\alpha_t \pi_g} \right)$$

$$= \prod_t \left( r_t e^{-\alpha_t(1 - \pi_g)} + (1 - r_t) e^{\alpha_t \pi_g} \right)$$
for
\[ r_t = \sum_i D_t(i) h_t(x_i, g_i) \]
as in the previous section. Since each factor of the bound above depends on only one \( \alpha_t \), we can minimize each factor separately by setting the first derivative with respect to \( \alpha_t \) to zero.

\[ 0 = \frac{\partial}{\partial \alpha_t} \left( r_t e^{-\alpha_t(1-\pi_g)} + (1-r_t)e^{\alpha_t \pi_g} \right) \]

\[ \Leftrightarrow -(1-\pi_g)r_t e^{-\alpha_t(1-\pi_g)} + \pi_g(1-r_t)e^{\alpha_t \pi_g} = 0 \]

\[ \Leftrightarrow (1-\pi_g)r_t = \pi_g(r_t) \]

\[ \Leftrightarrow \alpha_t = \ln \left( \frac{(1-\pi_g)r_t}{\pi_g(1-r_t)} \right) \]

Note: Again \( \alpha_t \) is greater for better classifiers and fulfills
\[ \alpha_t = 0 \Leftrightarrow (1-\pi_g)r_t = \pi_g(1-r_t) \Leftrightarrow r_t = \pi_g. \]

3.) **Bound for \( \text{mxerr} \):** To get the bound for \( \text{mxerr} \) we substitute our choice for \( \alpha_t \) in 3.6 and get

\[ \text{mxerr} \leq \prod_t \left( \frac{(1-\pi_g)r_t}{\pi_g(1-r_t)} \right)^{\pi_g} \sum_i D_t(i) \left( \frac{\pi_g(1-r_t)}{(1-\pi_g)r_t} h_t(x_i, g_i) \right)^{\pi_g} \]

(3.7)

Now we bound the term \( \left( \frac{\pi_g(1-r_t)}{(1-\pi_g)r_t} h_t(x_i, g_i) \right) \) by use of the inequality

\[ x^a \leq 1 - a + ax \] for \( x \geq 0 \) and \( a \in [0,1] \)

\( (x^a \text{ convex for } a \in [0,1] \text{ means } x^a \leq ax^1 + (1-a)x^0 = 1 - a + ax) \) and get

\[ \left( \frac{\pi_g(1-r_t)}{(1-\pi_g)r_t} \right)^{h_t(x_i, g_i)} \leq 1 - h_t(x_i, g_i) + h_t(x_i, g_i) \frac{\pi_g(1-r_t)}{(1-\pi_g)r_t}. \]

Taking the weighted sum and simplifications lead to

\[ \sum_i D_t(i) \left( \frac{\pi_g(1-r_t)}{(1-\pi_g)r_t} h_t(x_i, g_i) \right) \leq \frac{1-r_t}{1-\pi_g} \]

Substitution in 3.7 leads to

\[ \text{mxerr} \leq \prod_t \left( \frac{r_t^\pi_g}{\pi_g^{1-\pi_g}} \right)^{\pi_g} \]

(3.8)

Looking at this bound we see that the factors of \( \text{mxerr}(r_t) \) are symmetric around \( r_t = \pi_g \) and take their maximum of 1 there. Therefore if \( r_t > \pi_g \) is valid the bound for \( \text{mxerr} \) decreases.
4.) **Exponential decrease of the bound:** As for AdaBoost we want to know, if the bound for \( m_{xerr} \) decreases exponentially, if \( r_t = \pi_g + \delta \) with \( \delta \in (0, 1 - \pi_g) \) for all \( t \). We can rewrite 3.8 as

\[
m_{xerr} \leq \prod_t \left( 1 - \frac{\delta}{1 - \pi_g} \right)^{1 - \pi_g} \left( 1 + \frac{\delta}{\pi_g} \right)^{\pi_g}
\]

and bound both terms using the binomial series: the series of the first term have only negative terms, we stop after the terms of second order and get

\[
\left( 1 - \frac{\delta}{1 - \pi_g} \right)^{1 - \pi_g} \leq 1 - \delta + \delta^2 \frac{(1 - \pi_g)(-\pi_g)}{2(1 - \pi_g)^2}.
\]

The series of the second term have both positive and negative terms, we stop after the positive term of first order and get

\[
\left( 1 + \frac{\delta}{\pi_g} \right)^{\pi_g} \leq 1 + \delta.
\]

Thus

\[
m_{xerr} \leq \prod_t \left( 1 - \delta^2 \left( 1 + \frac{\pi_g}{2(1 - \pi_g)} \right) - \delta^3 \left( \frac{\pi_g}{2(1 - \pi_g)} \right) \right).
\]

Using \( 1 + x \leq e^x \) for \( x \leq 0 \) leads to

\[
m_{xerr} \leq \exp \left[ -T \left( \delta^2(1 + \frac{\pi_g}{2(1 - \pi_g)}) + \delta^3(\frac{\pi_g}{2(1 - \pi_g)}) \right) \right].
\]

Using 1.

\[
\text{Note: 1.) In contrast to GrPloss the algorithm doesn’t change when we add the property } \sum_g h_t(x, g) = 1, \text{ because the algorithm only uses } h_t(x_i, g_i).
\]

2.) For the special case \( \pi_g = 1/|G| \) we get

\[
D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\alpha_t(h_t(x_i, g_i) - 1/|G|)} \quad \text{for} \quad \alpha_t = \ln \left( \frac{(|G| - 1) r_t}{1 - r_t} \right),
\]

which is just the same as GrPloss for decision stumps. For this choice the pseudo-loss error and the maxgroup error are the same too.

3.) **Choice of \( \pi_g \):** Since we use decision stumps whose output are group proportions, it is reasonable to represent the maxgroup rule in the form

\[
\text{maxgroup rule } : X \times G \rightarrow [0, 1] : h(x, g) = \frac{N_g}{N}.
\]
CHAPTER 3. BOOSTMA

Since we compare \( r_t \) of the decision stumps with \( \pi_g \) it is natural to choose \( r_1 \) of the maxgroup rule for \( \pi_g \), which leads to

\[
\pi_g := \sum_{g \in G} \left( \frac{N_g}{N} \right)^2.
\]

Algorithm Boost.MA:

**Input:** learning set \( \mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N)\}; x_i \in \mathcal{X}, g_i \in \mathcal{G} \}, \)

\( \mathcal{G} = \{1, \ldots, |\mathcal{G}|\} \), weak classifier of the form \( h: \mathcal{X} \times \mathcal{G} \rightarrow [0, 1] \).

Optionally \( T \): number of boosting rounds

**Initialization:** \( D_1(i) = \frac{1}{N} \).

For \( t = 1, \ldots, T \):

- Train the weak classifier \( h_t \) with distribution \( D_t \), where \( h_t \) should maximize

\[
r_t = \sum_i D_t(i) h_t(x_i, g_i)
\]

- If \( r_t \leq \pi_g \): goto output with \( T := t - 1 \)

- Set

\[
\alpha_t = \ln \left( \frac{(1 - \pi_g)r_t}{\pi_g(1 - r_t)} \right).
\]

- Update \( D \):

\[
D_{t+1}(i) = D_t(i) e^{-\alpha_t (h_t(x_i, g_i) - \pi_g)} / Z_t.
\]

where \( Z_t \) is a normalization factor (chosen so that \( D_{t+1} \) is a distribution)

**Output:** Normalize \( \alpha_1, \ldots, \alpha_T \) and set the final classifier \( H(x) \):

\[
H(x) = \arg\max_{g \in \mathcal{G}} f(x, g) = \arg\max_{g \in \mathcal{G}} \left( \sum_{t=1}^{T} \alpha_t h_t(x, g) \right)
\]

Figure 3.3: Algorithm BoostMA
Chapter 4

Postprocessing: Probability estimation and postrang

The goal of this chapter is to use the output of AdaBoost.M2 to get a probability estimate \( p_x = (p(1|x), \ldots, p(|G||x)) \) for \( (P(1|x), \ldots, P(|G||x)) \) by an additional analysis of the distributions of the \(|G|\) possible scores \( f(x, g)\).

The first section contains Problog1, a method for the probability estimation, which is very simple and easy to program and compute. The drawback is, that it uses an ad-hoc normalization, which is not very satisfying. The second section contains a cleaner way to normalize at the cost of solving a system of linear equations. The corresponding method is called Problog2. The third section generalizes the probability estimation by making a connection to error-correcting output codes and making multivariate density estimations. The fourth and last section contains an ad-hoc modification, where the scores \( f(x, k) \) are converted to ranks before majority voting.

4.1 Problog1 (Probability estimation with logistic regressions 1)

The intuitive idea of the probability estimation is that score \( f(x, k) \) should be highest for examples from group \( k \) and lower for examples of other groups. By applying a binary logistic regression with score \( f(x, k) \) as input variable one not only distinguishes group \( k \) from the other groups, but also gets an estimate for the probability \( p(k|x) \) to be in group \( k \) (see Fig. 4.1).

\[
p(k|x) = \frac{1}{1 - \exp[b_k + w_k f(x, k)]}
\]

The question remains how to combine the \(|G|\) probability estimates obtained from logistic regression: an obvious way is to normalize the \(|G|\) estimates so
Figure 4.1: Distribution of score $f(x,1)$ for cases in group 1 (dark grey) and cases in other groups (light grey) and $p(1|f(x,1))$, estimated by logistic regression that they sum to one

$$p(k|x) = \frac{p(k|x)}{\sum_g p(g|x)}.$$  

The resulting algorithm is given in Fig. 4.2.

The computational costs of the probability estimation are low. One makes $|G|$ univariate logistic regressions after boosting, and for each object to classify there is only the evaluation of the logistic regression and the division for the normalization.
**Input:** Learning set $\mathcal{L}$ and from AdaBoost $\alpha_1, \ldots, \alpha_T$ and $h_1, \ldots, h_T$.

For $k = 1, \ldots, |G|$: Make binary logistic regressions:

- Set training set
  $$T_k = \{(f(x_1, k), y_k(g_1)), \ldots, (f(x_N, k), y_k(g_N)); (x_i, g_i) \in \mathcal{L}\}$$
  where
  $$f(x, k) = \sum_t \alpha_t h_t(x, k) \quad \text{and} \quad y_k(g) = \begin{cases} +1 & g = k \\ -1 & g \neq k \end{cases}$$

- Learn logistic regression, given by $b_k$ and $w_k$ from this dataset
  $$\ln \left( \frac{p(+1|f_k)}{p(-1|f_k)} \right) = \ln \left( \frac{p(k|f_k)}{p(G \setminus k|f_k)} \right) = b_k + w_k f_k$$

**Output:** For each $x$ to classify set

$$p(k|x) = \frac{1}{1 - \exp[b_k + w_k \sum\alpha_t h_t(x, k)]}$$

and then

$$p(k|x) = \frac{p(k|x)}{\sum_g p(g|x)}.$$

---

Figure 4.2: Algorithm Problog1
4.2 Problog2

Since an ad-hoc normalization is not a satisfying solution we tackled the combination problem more systematically: For each score \( k \) we have the following equation from the logistic regression

\[
\log \left( \frac{p(k|x)}{p(G \setminus k|x)} \right) = b_k + w_k f(x, k) =: c_k(x)
\]

for \( p(G \setminus k|x) = \sum_{g \neq k} p(g|x) \). Now the idea was to fulfill these equations as well as possible under the normalization constraint \( \sum_g p(g|x) = 1 \). Basic transformations yield to

\[
p(k|x) - e^{c_k(x)} \sum_{g \neq k} p(g|x) = \sum_{g=1}^{\mid G \mid} \left( \left[ [g = k] \right] - e^{c_k(x)} \left[ [g \neq k] \right] \right) p(g|x) = 0
\]

Thus we have obtained a system of linear equations

\[ Ap|x = 0 \]

with

\[ A_{kg} = \left[ [g = k] \right] - e^{c_k(x)} \left[ [g \neq k] \right] \]

for \( p|x = p(1|x), \ldots, p(\mid G \mid|x) \) together with the additional normalization constraint

\[ \sum_g p(g|x) = 1. \]

By specifying, that we satisfy the system of linear equations as much as possible, if

\[ p|x = \arg \min_{\hat{p}|x} \| A\hat{p}|x \|_2 \]

we reach a formulation of the problem, for which a solution in Matlab (function lsqlin.m) is available. Note, that the basic transformations above consist of taking the exponent and subtracting, we do not multiply the whole equation with a constant. For an exact solution of the system multiplications of the equations with different constants would make no difference, but it does for an approximate solution, which we have here.

The resulting algorithm is given in Fig. 4.3.

As Problog1 does, Problog2 performs \( \mid G \mid \) logistic regressions after boosting has finished. But to classify a particular object \( x \) one additionally has to approximately solve a system of \( \mid G \mid \) equations in \( \mid G \mid \) variables under the normalization constraint for the \( \mid G \mid \) variables.
Input: Learning set $\mathcal{L}$ and from AdaBoost $\alpha_1, \ldots, \alpha_T$ and $h_1, \ldots, h_T$.

For $k = 1, \ldots, |G|$: Make binary logistic regressions:

- Set training set $T_k = \{(f(x_1, k), y_k(g_1)), \ldots, (f(x_N, k), y_k(g_N)) ; (x_i, g_i) \in \mathcal{L}\}$

  where
  $$f(x, k) = \sum_t \alpha_t h_t(x, k) \quad \text{and} \quad y_k(g) = \begin{cases} +1 & g = k \\ -1 & g \neq k \end{cases}$$

- Learn logistic regression, given by $b_k$ and $w_k$ from this dataset

  $$\ln \left( \frac{p(+1|f_k)}{p(-1|f_k)} \right) = p(k|f_k) \ln \left( \frac{p(G \setminus k|f_k)}{p(f_k)} \right) = b_k + w_k f_k$$

Output: For each $x$ to classify:

- For all $k$ set $c_s(x) = b_s + w_s f(x)$

  and for $g, k = 1, \ldots, |G|$ set

  $$A_{kg} = \left[ [k = g] - c_k(x) [k \neq g] \right]$$

- Calculate

  $$p_{|x|} = \arg \min_{p_{|x|}} \|A \hat{p}_{|x|}\|_2$$

  under the constraint $\sum_g p(g|x) = 1$
4.3 Generalization and connection to error-correcting codes

Problog1 and Problog2 both use one single score \( k \) to distinguish group \( k \) from the other groups. There are two natural generalization possibilities of this method.

The first one is to use multivariate instead of univariate logistic regression, that means we use

\[
      f(x) := (f(x, 1), \ldots, f(x, |\mathcal{G}|))
\]

as independent variables for the logistic regression instead of just one component \( f(x, k) \).

The second one is not to distinguish one group from the others but to distinguish cases from a set of groups from another set of groups. We also permit, that there are groups \( g \) (with \( a_{s,g} = 0 \)), whose cases are not considered for the logistic regression for bits of the \( |\mathcal{G}| \) codewords. More precisely one defines a codeword \( a_{s,-} = (a_{s,1}, \ldots, a_{s,|\mathcal{G}|}) \in \{-1, 0, +1\}^{|\mathcal{G}|} \) and a training set \( \mathcal{T}_s = \{(f(x_i), a_{s,g_i}) : a_{s,g_i} \neq 0, (x_i, g_i) \in \mathcal{L}\} \). Then one uses logistic regression with this training set leading to coefficients \( b_s \) and \( w_s = (w_s, 1, \ldots, w_s,p) \) and get

\[
      \log \left( \frac{\sum_{g: a_{s,g}=+1} p(g|x)}{\sum_{g: a_{s,g}=-1} p(g|x)} \right) = b_s + \sum_{j=1}^p w_{s,j} f(x,j) =: c_s(x)
\]

Again we can make some simple transformations without scalar multiplications and get

\[
      \sum_{g: a_{s,g}=+1} p(g|x) - \sum_{g: a_{s,g}=-1} e^{c_s(x)} p(g|x) = 0
\]

leading to a system of equations

\[
      A p_{|x} = 0
\]

with

\[
      A_{sg} = [(a_{s,g} = +1)] - e^{c_s(x)}[(a_{s,g} = -1)]
\]

for \( p_{|x} = p(1|x), \ldots, p(|\mathcal{G}||x) \) together with the additional normalization constraint

\[
      \sum_g p(g|x) = 1.
\]

The resulting algorithm is given in Fig. 4.4.

Now we must perform \( S \) logistic regressions after boosting has finished. To classify a particular object \( x \) one has to approximately solve a system of \( S \) equations in \( |\mathcal{G}| \) variables under the normalization constraint for the \( |\mathcal{G}| \) variables.

Note: This generalization is similar to the method for the probability estimation in section 1.3.4. One difference is, that here we already consider scores instead of the measurements \( x \). Since each score belongs to a group the situation here is different. This is the cause, why we came on another way to the
error-correcting approach. This is probably also the cause, why in preliminary trials Problog1 and Problog2 always outperformed the error-correcting approach with random matrices. The poor results were one reason, why we didn’t make experiments with this generalization.

Another difference is, that in our case the twoclass classifier is logistic regression and our set of linear equations is special for logistic regression.

Our probability estimation also permits zeros to be in the codematrix. The proportion of zeros in the codematrix would be an additional parameter to investigate, so this extends the problem of finding a good code, so that an additional validation set would be required.

---

**Input:** Learning set \( \mathcal{L} \), codematrix \( a = (a_{s,g})_{s=1,...,S; g=1,...,|G|} \) and from AdaBoost \( \alpha_1, \ldots, \alpha_T \) and \( h_1, \ldots, h_T \).

**For** \( s = 1, \ldots, S \): Make binary logistic regressions:
- Set training set \( \mathcal{T}_s = \{(f_i, a_{s,g_i}) : a_{s,g_i} \neq 0, (x_i, g_i) \in \mathcal{L}\} \)
  
  where \( f_i = f(x_i) = (f(x_i, 1), \ldots, f(x_i, |G|)) \).

- Learn logistic regression, given by \( b_s \) and \( w_{s,j} \) from this dataset

\[
\log \left( \frac{\sum_{g : a_{s,g} = +1} p(g|x)}{\sum_{g : a_{s,g} = -1} p(g|x)} \right) = b_s + \sum_{j=1}^p w_{s,j} f(x, j)
\]

**Output:** For each \( x \) to classify:
- For \( s = 1, \ldots, S \) set

\[
c_s(x) = b_s + \sum_{j=1}^p w_{s,j} f(x, j)
\]

and for \( s = 1, \ldots, S \) and \( g = 1, \ldots, |G| \) set

\[
A_{s,g} = [[a_{s,g} = +1]] - e^{c_s(x)}[[a_{s,g} = -1]]
\]

- Calculate

\[
p_{\|x} = \arg \min_{\tilde{p}_{\|x}} \|\tilde{A}_{\|x}\|_2
\]

under the constraint \( \sum_g \tilde{p}(g|x) = 1 \)

---

Figure 4.4: Algorithm Probecoc
4.4 Postrank

In this section we substitute the scores for the groups by ranks. An object \( x \) will then be assigned to the group with the highest rank instead of the group with the highest score. In previous trials with a smaller digit data set AdaBoost.M2 concentrated on examples of the groups 1 and 7. As a consequence the scores for groups 1 and 7 could get higher than the scores for the other groups. The final classifier then made many mistakes by assigning objects wrongly to groups 1 and 7. The substitution by ranks largely corrected this distortion and diminished the mean final training error from 28.7% down to 26.7% and the mean test error rate from 34.3% down to 26.9% (Bayes-error 26.0%).

Now we describe the method more precisely: recalling the voting step of AdaBoost.M2 an object \( x \) gets assigned to the group

\[
\arg \max_{g \in G} f(x, g) = \arg \max_{g \in G} \sum_{t=1}^T \alpha_t h(x, g)
\]

For each group \( g \) we order the scores for this group for all training samples, i.e. we find a permutation \( r_g : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\} \) with

\[
f(x_{r_g(1)}(x), g) \leq f(x_{r_g(2)}(x), g) \leq \ldots \leq f(x_{r_g(N)}(x), g)
\]

Then we substitute the scores \( f(x, g) \) by \( r_g^{-1}(i) \) and assign \( x_i \) to the group

\[
\arg \max_{g \in G} r_g^{-1}(i)
\]

So far this method works for all cases in the training set but not for a new object \( x \). We can easily accomplish this by computing \( f(x, g) \), finding the example \( i_g \) with the most similar score for this group in the training set

\[
i_g(x) := \arg \min_{i \in \{1, \ldots, N\}} |f(x, g) - f(x, g)|
\]

and taking \( r_g^{-1}(i_g(x)) \) instead of \( f(x, g) \). If \( i_g(x) \) is not uniquely defined we choose one of these nearest neighbors at random.

We can summarize this method in algorithm 4.5

**Note:**
1. If we have several examples in the training set with equal score \( f(x, g) \) then \( r_g \) is not uniquely defined. But we can arbitrarily choose one possible \( r_g \), because we have the randomness built in at finding the nearest neighbor, if \( |i_g(x)| \geq 2 \).
2. This method has no theoretical justification, nevertheless we decided to investigate it further, because its computation costs are very low and it can be programmed easily.
Input: From the boosting algorithm score matrix \( f = f(x_i, g)_{i=1,\ldots,N, g=1,\ldots,|G|} \)

For \( g = 1, \ldots, |G| \): sort scores for group \( g \)

- Find \( r_g : \{1, \ldots, N\} \to \{1, \ldots, N\} \) with
  \[
  f(x_{r_g(1)}, g) \leq f(x_{r_g(2)}, g) \leq \cdots \leq f(x_{r_g(N)}, g)
  \]

Output: For each \( x \) to classify:

- Compute \( f(x) = (f(x, 1), \ldots, f(x, |G|)) \).
- For \( g = 1, \ldots, |G| \)
  \[
  i_g(x) := \arg \min_{i \in \{1, \ldots, N\}} |f(x_i, g) - f(x, g)|
  \]

- If \( |i_g| \geq 2 \): choose one element of \( i_g(x) \) at random and call it \( i_g(x) \).
- \( H(x) := \arg \max_{g \in G} r_g^{-1}(i_g(x)) \)

---

Figure 4.5: Algorithm Postrank
Chapter 5

Experiments

There are two central questions that should be answered by the experiments. The first one concerns the two new algorithms GrPloss and BoostMA: how efficiently can they minimize their performance measures and how does this minimization influence the training and test error? This question will be answered in section 5.2. The second one concerns the probability estimation: how well does the probability estimation work and how does the probability estimation influence the error rates? This question will be answered in section 5.3. We answer the questions by means of experiments with 12 datasets, therefore we start with making some remarks about the experimental setup in section 5.1.

The original plan for the comparison of the algorithms was to compare 3 algorithms: AdaBoost.M2 as a reference algorithm, GrPloss and BoostMA. We also studied a version of AdaBoost.M2, where \( q \) is held constant, because it has the same structure as AdaBoost.M2, but uses the same pseudo-loss as GrPloss does.

We first preferred boosting with reweighting to boosting with resampling, because it is a more direct and deterministic method in the sense, that every run on the same data is exactly the same. But the results of the experiments were not satisfying. AdaBoost.M2 had clearly the best performance, the other 3 algorithms were partly extremely bad. There are some reasons, why boosting with resampling should perform better, so we repeated the experiments with resampling. As expected the results were better with resampling. All algorithms had comparable performance, but the 2 new algorithms converged a little bit faster than AdaBoost.M2 and its modification.
5.1 Experimental setup

5.1.1 Datasets

5.1.1.1 Choice of the 12 datasets

Since the focus of research lies on multiclass problems only datasets with three or more groups were considered. To get a reliable estimate for the expected error the dataset should contain about 1000 cases, because in previous trials with the iris dataset (150 cases) we had problems with the error estimation with 10-fold crossvalidation. There the variation of the 10 error rates was too high to provide a reliable estimate. The vehicle dataset is the smallest one with 846 cases. We chose datasets, which were used previously in other publications before to get an idea of the error, which can be reached with other methods and to check the results for consistency with results from others. Most files are available from the UCI repository (http://www1.ics.uci.edu/ mlearn/MLRepository). Another restriction was to use only datasets without missing values. Missing values would have required much computational effort and could have eventually distorted the results. As a result of these considerations we used 12 datasets. Most of them have balanced group proportions, only four of twelve have unequal group proportions (see table 5.2). This is important, because only for these datasets the algorithms GrPloss and BoostMA are different, because

$$\pi_g = \sum_{g \in G} \left( \frac{N_g}{N} \right)^2 \neq \frac{1}{|G|}.$$ 

5.1.1.2 Short problem description and summary

Since the datasets are not of primary importance only a short summary is given. The most important facts are given in table 5.1 and table 5.2.

The datasets digitbreiman, optdigits and pendigits all concern digit recognition. For digitbreiman each of the 10 digits consists of 7 lights which can be lighted or not as used for digital displays. It is a simulated classification problem described in [3] with a Bayes error rate of 26%. Optdigits and pendigits regard handwritten digits, where for optdigits a digit is represented by 32x32 bitmaps and for pendigits a digit is represented by a space-time curve on a pressure sensitive tablet.

In the dataset cars offers for cars are divided into the 4 categories unacceptable, acceptable, good and very good by knowledge about the price, technology and comfort of the car.

The goal of the letter dataset is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters, where each pixel display is described by some statistical measures.

For nursery the task is to rank applications for nursery schools into the 5 groups not recommended, recommended, very recommended, priority and special priority by knowledge about occupation of parents and child’s nursery, family structure and financial standing, and social and health picture of the family.
Table 5.1: Properties of the databases

<table>
<thead>
<tr>
<th>database</th>
<th>N</th>
<th># groups</th>
<th># variables</th>
<th>type of variables</th>
<th>groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>1728</td>
<td>4</td>
<td>6</td>
<td>ordered</td>
<td>unbalanced</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>5000</td>
<td>10</td>
<td>7</td>
<td>binary</td>
<td>balanced</td>
</tr>
<tr>
<td>letter</td>
<td>20000</td>
<td>26</td>
<td>16</td>
<td>integer</td>
<td>balanced</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>4</td>
<td>8</td>
<td>ordered</td>
<td>unbalanced</td>
</tr>
<tr>
<td>optdigits</td>
<td>5620</td>
<td>10</td>
<td>64</td>
<td>integer</td>
<td>balanced</td>
</tr>
<tr>
<td>pendigits</td>
<td>10992</td>
<td>10</td>
<td>16</td>
<td>integer</td>
<td>balanced</td>
</tr>
<tr>
<td>satimage</td>
<td>6435</td>
<td>6</td>
<td>34</td>
<td>integer</td>
<td>unbalanced</td>
</tr>
<tr>
<td>segmentation</td>
<td>2310</td>
<td>7</td>
<td>19</td>
<td>continuous</td>
<td>balanced</td>
</tr>
<tr>
<td>waveform</td>
<td>5000</td>
<td>3</td>
<td>21</td>
<td>continuous</td>
<td>balanced</td>
</tr>
<tr>
<td>vehicle</td>
<td>846</td>
<td>4</td>
<td>18</td>
<td>integer</td>
<td>balanced</td>
</tr>
<tr>
<td>vowel</td>
<td>990</td>
<td>11</td>
<td>10</td>
<td>continuous</td>
<td>balanced</td>
</tr>
<tr>
<td>yeast</td>
<td>1484</td>
<td>10</td>
<td>9</td>
<td>cont+categorical</td>
<td>unbalanced</td>
</tr>
</tbody>
</table>

Table 5.2: Group frequencies for databases with unbalanced groups

<table>
<thead>
<tr>
<th>database</th>
<th>1 group</th>
<th>2 group</th>
<th>3 group</th>
<th>4 group</th>
<th>5 group</th>
<th>6 group</th>
<th>7 group</th>
<th>8 group</th>
<th>9 group</th>
<th>10 group</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>1210</td>
<td>384</td>
<td>69</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nursery</td>
<td>4320</td>
<td>330</td>
<td>4266</td>
<td>4044</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>satimage</td>
<td>1072</td>
<td>479</td>
<td>961</td>
<td>415</td>
<td>470</td>
<td>1038</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yeast</td>
<td>463</td>
<td>429</td>
<td>244</td>
<td>163</td>
<td>51</td>
<td>44</td>
<td>37</td>
<td>30</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>

The purpose of satimage is to predict the type of soil by multi-spectral values of pixels in 3x3 neighborhoods in a satellite image.
The purpose of segmentation is to predict the type of a 3x3 region of color outdoor pictures into the 7 classes brickface, sky, foliage, cement, window, path, grass.
The aim of the vehicle dataset is to classify a given silhouette as one of 4 types of vehicle (Opel, Saab, bus, van) using a set of features extracted from the silhouette.
Vowel aims at recognizing 10 not nearer specified vowels by some undescribed features.
As the digit example the waveform data are simulated and described in [3]. The data should simulate silhouettes of ships, and the Bayes error rate is 14%.
The goal of yeast is to predict the right one of the following 10 cellular localization sites of proteins: cytosolic or cytoskeletal, nuclear, mitochondrial, membrane protein - no terminal signal, membrane protein - uncleaved signal, membrane protein - cleaved signal, extracellular, vacuolar, peroxisomal, endoplasmic reticulum lumen. This is the only example where the features are partly scores of previously performed discriminant analyses.
5.1.2 Experimental setup and remarks about the implementation

5.1.2.1 Error estimation

To estimate the expected classification error we used either test error or 10-fold crossvalidation. We used a test error estimate for big datasets (N≥5000), which required much computing time in preliminary trials. There the removal of a large test set (32% of the data) should not worsen the resulting classifier. We also used a test error estimate for the vowel dataset, because it was already divided into a training and a test set by the providers of the dataset. For datasets with less than 5000 instances, where a division into a training and a test set was not specified, we used 10-fold crossvalidation for the error estimation. Table 5.3 gives a quick overview over the error estimation, where the numbers are the number of instances in the training and test set respectively.

<table>
<thead>
<tr>
<th>database</th>
<th>type of error estimation</th>
<th>chosen base classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>10-CV</td>
<td>pure integer</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>3437/1563</td>
<td>pure integer</td>
</tr>
<tr>
<td>letter</td>
<td>16000/4000</td>
<td>pure continuous</td>
</tr>
<tr>
<td>nursery</td>
<td>10-CV</td>
<td>pure continuous</td>
</tr>
<tr>
<td>optdigits</td>
<td>3823/1797</td>
<td>pure continuous</td>
</tr>
<tr>
<td>pendigits</td>
<td>7494/3498</td>
<td>pure continuous</td>
</tr>
<tr>
<td>satimage</td>
<td>4435/2000</td>
<td>pure continuous</td>
</tr>
<tr>
<td>segmentation</td>
<td>10-CV</td>
<td>pure continuous</td>
</tr>
<tr>
<td>waveform</td>
<td>3437/1563</td>
<td>pure continuous</td>
</tr>
<tr>
<td>vehicle</td>
<td>10-CV</td>
<td>pure continuous</td>
</tr>
<tr>
<td>vowel</td>
<td>528/462</td>
<td>pure continuous</td>
</tr>
<tr>
<td>yeast</td>
<td>10-CV</td>
<td>continuous+integer</td>
</tr>
</tbody>
</table>

5.1.2.2 Implementation of the base classifiers

Nearly the whole algorithm was implemented in Matlab. The base classifier for databases with integer variables was totally programmed in Matlab using the find command. Note, that ordered and dichotomous variables were coded and treated as integers.

Computation time is a critical point, because the datasets and the number of boosting rounds are big. To minimize computation time the search loop of the base classifier for continuous variables was implemented in C. For the vowel dataset this reduced the computation time of a stump from 35.9 seconds programmed in Matlab as a whole to 0.61 seconds. For most datasets with integer variables this base classifier was even faster than the one implemented in Matlab, so it was also used there. Only for the digitbreiman and car datasets
with few different integer values the Matlab version remained faster.

The yeast dataset is the only dataset including a categorical variable, which is not dichotomous. This categorical variable has 3 categories. Most cases are from category one, only 4 from category 2 and 11 from category 3. The two small categories were treated as one, so that the base classifier was then a mixture of base classifiers for integer and classifiers for continuous variables. Table 5.3 includes a summary of the used base classifier types.

For the 3 algorithms AdaBoost.M2 with constant $q$, GrPloss and BoostMA the goal of the base classifier is to maximize $r_t$, but for AdaBoost.M2 the goal of the base classifier is to minimize the pseudo-loss. This results in a total of $3 \cdot 2 = 6$ different types of base classifiers. Note, that the minimization of the pseudo-loss requires more time than the minimization of $r_t$, which results in a higher computation time.

The minimum number of cases in a leave was set to 10.

5.1.2.3 Treatment of the rounding problem

As discussed in [2] the individual sampling weights $D_t(i)$ can get very small. Since decision stumps are very weak classifiers the number of boosting rounds was expected to be very high, which increases the chance, that the sampling weights get very low. In preliminary trials sampling weights even fell below the unit roundoff of Matlab ($2.2204 \cdot 10^{-16}$). Similar as it was done in [2] we set a lower bound $D_{min}$ to $10^{-10}$ and set the weights of instances, which were below this threshold, to $D_{min}$. Note, that these low weights would be ideal for weight trimming as described in [12], which would decrease computation time. To keep things clear without uncertain effects we omitted weight trimming.

5.1.2.4 The stopping criterion

For the trials undertaken so far with AdaBoost.M2 with resampling the stopping criterion $\epsilon_t > \frac{1}{2}$ never appeared to be a problem (see figure 5.1, right panel, for a typical picture). However for AdaBoost.M2 with reweighting the curve gets much smoother and doesn’t seem to reach $\frac{1}{2}$ (see figure 5.1, left panel). The idea is to stop the algorithm, if the pseudo-loss is near enough to $\frac{1}{2}$. As the pseudo-loss reaches $\frac{1}{2}$ the weight $\alpha$ of the base classifier reaches 0, so we used the following stopping criterion. Set $\Lambda = \text{mean}(\alpha_1, \ldots, \alpha_{10})$ and a stopping constant $\text{STOPCONST}$. Stop the algorithm at round $t$, if

$$\frac{\text{mean}(\alpha_{t-9}, \ldots, \alpha_t)}{\Lambda} < \text{STOPCONST}$$

We used the mean of 10 values for $\alpha$ to compensate for the fluctuations of $\alpha$. The advantage of the formulation with $\alpha$ is, that it is the same for all algorithms. It has also the simple interpretation that the algorithm should stop, if the change of the final classifier gets small enough. The choice for $\text{STOPCONST}$ in our implementation was 0.02.
We also set a maximum number of boosting rounds to stop the algorithm, if the stopping criterion does not work. After previous trials with the digitbreiman and the waveform database with 1000 examples in the learning set and time estimations for the datasets we chose to set maximum number of boosting rounds to 2000. With this choice we had kept a reserve for the digitbreiman and the waveform database and also kept the estimated computation time for all databases feasible.

![Figure 5.1: Pseudo-loss for AdaBoost.M2 dependent on the boosting round for reweighting (left) and resampling (right)](image-url)
5.2 Comparison of algorithms

Following the historical proceeding we will make a short look at the results of boosting with reweighting in 5.2.1. In 5.2.2 we will see, that reweighting is clearly inferior to resampling. Subsection 5.2.3 contains the main results of the work. There we will look at the results of boosting with resampling in detail.

5.2.1 Performance of boosting with reweighting

Good indicators for the performance of the algorithms turned out to be error curves, where the error is plotted against the number of boosting rounds (see figure 5.3). Note that for only 4 databases GrPloss and BoostMA are different. We will discuss their differences in the next section and only talk about GrPloss here.

If one wants to get the best classifier one would choose the round with the minimal training error and take the corresponding classifier [8]. The corresponding estimate of the expected error, estimated either by crossvalidation or by a test error, is our primary measure of performance. For sake of simplicity we will make no difference between the two methods to estimate the expected error and call them both test error. There AdaBoost.M2 clearly performs best (see figure 5.2). GrPloss and AdaBoost.M2 with constant q have comparable performance. For some databases these two algorithms even fail to work properly with respect to the training error.

A point in favour of GrPloss is, that it decreases the training error more quickly than AdaBoost.M2.

Since GrPloss and AdaBoost.M2 with constant q have comparable performance, the modification of q seems to offer more possibilities for the algorithm to diminish the training error further. GrPloss and AdaBoost.M2 with constant q seem to get stuck in a local minimum faster than AdaBoost.M2. With resampling the algorithm gets more possibilities to avoid these local minima. Using resampling also makes the base hypothesis space bigger, because with reweighting a base classifier must satisfy the following condition for \( \pi_0(g) \) and \( \pi_1(g) \):

\[
N_g = N_0(g) + N_1(g) = N_0\pi_0(g) + N_1\pi_1(g),
\]

because \( N_g \) (the number of instances from group \( g \) in the learning sample) is constant. Now we define \( D_t(g) \) the sum of the weights in round \( t \) for all instances from group \( g \):

\[
D_t(g) = \sum_{i: y_i = g} D_t(i).
\]

Since \( D_t(g) \) can have big changes during boosting (see subsection 3) \( N_g \) also changes for boosting with resampling but not for boosting with reweighting. This is an effect, that really matters and is important especially for the discrimination of groups, which are not so frequent in the training set.

The fact, that AdaBoost.M2 differed substantially from the other algorithms, can also be confirmed by figure 5.4, where \( r_t \) is plotted against \( t \). AdaBoost.M2
Figure 5.2: Pairwise comparison of the algorithms with reweighting with respect to the test error at the round with minimal training error

is the only algorithm, where $r_t$ (and subsequently also the pseudo-loss without $q$) shows an oscillating behaviour.

Another important point to notice is, that GrPloss was best in minimizing the pseudo-loss error rate, which can be seen from the pseudo-loss error curve (figure 5.5).
Figure 5.3: Training error curves for boosting with reweighting: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA.
Figure 5.4: Curves for $r_t$ for boosting with reweighting: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant $q$, dash dotted: GrPloss, dotted: BoostMA
Figure 5.5: Pseudo-loss error curves for boosting with reweighting: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA.
5.2.2 Comparing reweighting and resampling

Boosting with resampling brought a substantial improvement for the algorithms AdaBoost.M2 with constant q, GrPloss and BoostMA. The performance for AdaBoost.M2 was also better for boosting with reweighting. This can be seen by looking at our main performance measure, the test error at the boosting round with the minimal training error (figure 5.6). Since boosting with resampling was superior to boosting with reweighting we concentrate on the results for resampling in the following subsection.

Figure 5.6: Comparison between reweighting and resampling for all 4 algorithms. Each point represents the test error at the boosting round with the minimal training error for a dataset.
5.2.3 Results for boosting with resampling

This subsection contains the main results and descriptions of the boosting performance.

5.2.3.1 Comparison of the test error

Since all four algorithms had a better performance with resampling, but AdaBoost.M2 had the smallest improvements, we expect smaller differences between the algorithms. And in fact pairwise comparisons of the test error at the round with minimal training error show, that there doesn’t exist a clear winner as it is the case for reweighting (figure 5.7 and table 5.4).

Differences greater than 5% only occur for the car, letter, pendigits and vowel database. However one has to be cautious in interpreting the results, because for the letter dataset AdaBoost.M2 and AdaBoost.M2 with constant q may not have reached the end of the boosting process at the maximum number of boosting rounds of 2000 (figure 5.9). This leads to the smaller test error at the round with minimal training error for GrPloss. One also has to be cautious with the vowel dataset, where a big part of the test error (about 20 %) comes from generalization.

Therefore a cautious conclusion assigns about equal performance to the 4 algorithms with AdaBoost.M2 with constant q having maybe the worst overall performance.

The digitbreiman and the waveform datasets are simulated problems for which the Bayes error rate is known to be 26% and 14% respectively ([3]). For these two databases we can compare the test error at the round with minimal training error with the Bayes error. For the digitbreiman database the difference is less than 1.5% for all four algorithms, which is quite good, for the waveform database the differences vary from 2.6% to 5.6%. One can also see this as a simple check, that the algorithms work properly.
Figure 5.7: Pairwise comparison of the algorithms with resampling with respect to the test error at the round with minimal training error

Table 5.4: Test error for boosting with resampling at the round with minimal training error; bold and italic numbers correspond to high (>5%) and medium (>1.5%) differences to the minimal of the four error rates

<table>
<thead>
<tr>
<th>database</th>
<th>AdaM2</th>
<th>AdaM2oq</th>
<th>GrPloss</th>
<th>BoostMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.75</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>27.51</td>
<td>26.68</td>
<td>27.13</td>
<td>27.38</td>
</tr>
<tr>
<td>letter</td>
<td>47.18</td>
<td>53.13</td>
<td>41.70</td>
<td>41.70</td>
</tr>
<tr>
<td>nursery</td>
<td>14.27</td>
<td>12.51</td>
<td>12.35</td>
<td>12.67</td>
</tr>
<tr>
<td>optdigits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pendigits</td>
<td>18.61</td>
<td>26.50</td>
<td>20.44</td>
<td>20.75</td>
</tr>
<tr>
<td>satimage</td>
<td>18.25</td>
<td>18.95</td>
<td>17.80</td>
<td>18.90</td>
</tr>
<tr>
<td>segmentation</td>
<td>8.40</td>
<td>7.45</td>
<td>9.31</td>
<td>9.48</td>
</tr>
<tr>
<td>vehicle</td>
<td>35.34</td>
<td>38.53</td>
<td>38.16</td>
<td>36.87</td>
</tr>
<tr>
<td>vowel</td>
<td>54.33</td>
<td>57.79</td>
<td>67.32</td>
<td>67.32</td>
</tr>
<tr>
<td>waveform</td>
<td>16.63</td>
<td>19.64</td>
<td>18.17</td>
<td>17.72</td>
</tr>
<tr>
<td>yeast</td>
<td>60.65</td>
<td>61.52</td>
<td>61.99</td>
<td>62.47</td>
</tr>
</tbody>
</table>
Figure 5.8: Test error curves for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA
CHAPTER 5. EXPERIMENTS

Figure 5.9: Training error curves for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA.
5.2.3.2 Generalization error and overfit

The generalization error is defined as the difference between the test error and the training error. Note, that we took the difference between the test and the training error at the round with minimal training error and not at the last boosting round. Since the boosting algorithms should lead to rather equally complex classifiers the generalization error is expected to be of same size for all four algorithms. Table 5.5 shows, that the obtained differences are in fact very small. For the vowel dataset the generalization error is very big. This may be due to the fact, that the vowel data set was already divided into a training and a test set. However the training set contains samples of 8 speakers and the test set contains samples of 7 different speakers, which maybe an explanation for the high generalization error.

Table 5.5: Generalization error for boosting with resampling; bold and italic numbers correspond to high(>3%) and medium(>1%) differences to the minimal of the 4 generalization errors

<table>
<thead>
<tr>
<th>database</th>
<th>AdaM2</th>
<th>AdaM2sq</th>
<th>GrPloss</th>
<th>BoostMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>4.81</td>
<td>4.42</td>
<td>3.11</td>
<td>3.58</td>
</tr>
<tr>
<td>letter</td>
<td>1.19</td>
<td>0.91</td>
<td>1.15</td>
<td>1.17</td>
</tr>
<tr>
<td>nursery</td>
<td>0.03</td>
<td>0.07</td>
<td>-0.06</td>
<td>-0.01</td>
</tr>
<tr>
<td>optdigits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pendigits</td>
<td>1.91</td>
<td>1.59</td>
<td>1.82</td>
<td>1.75</td>
</tr>
<tr>
<td>satimage</td>
<td>8.36</td>
<td>6.67</td>
<td>6.10</td>
<td>6.25</td>
</tr>
<tr>
<td>segmentation</td>
<td>22.65</td>
<td>16.86</td>
<td>24.78</td>
<td>24.00</td>
</tr>
<tr>
<td>vehicle</td>
<td>4.19</td>
<td>2.69</td>
<td>3.02</td>
<td>2.81</td>
</tr>
<tr>
<td>vowel</td>
<td>0.40</td>
<td>0.76</td>
<td>2.42</td>
<td>1.31</td>
</tr>
</tbody>
</table>

We define the overfit as the difference between the training error at the last boosting round and the minimal training error. Generally the overfit is low, but notice, that often the overfit can not really be estimated, because the algorithm stopped at the maximum number of boosting rounds and not because of the stopping criterion (table 5.8). Table 5.7 suggests, that GrPloss and BoostMA suffer more from overfit than the other two algorithms. However a look at the training error curves shows, that GrPloss and BoostMA are faster in minimizing the training error. Therefore they reach the minimal training error faster and have more "time" to overfit. Therefore such a comparison of the overfit favours AdaBoost.M2 and AdaBoost.M2 with constant q.
Table 5.6: Minimal training error for boosting with resampling; bold and italic numbers correspond to high(>5%) and medium(>1.5%) differences to the minimal of the 4 error rates

<table>
<thead>
<tr>
<th>database</th>
<th>AdaM2</th>
<th>AdaM2oq</th>
<th>GrPloss</th>
<th>BoostMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>25.49</td>
<td>25.52</td>
<td>25.63</td>
<td>25.63</td>
</tr>
<tr>
<td>letter</td>
<td><strong>46.07</strong></td>
<td><strong>51.89</strong></td>
<td>40.02</td>
<td>40.14</td>
</tr>
<tr>
<td>nursery</td>
<td><strong>14.16</strong></td>
<td>12.50</td>
<td>12.37</td>
<td>12.63</td>
</tr>
<tr>
<td>optdigits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pendigits</td>
<td>13.82</td>
<td><strong>23.38</strong></td>
<td><strong>17.17</strong></td>
<td><strong>17.20</strong></td>
</tr>
<tr>
<td>satimage</td>
<td>15.85</td>
<td>16.82</td>
<td>15.69</td>
<td>16.87</td>
</tr>
<tr>
<td>segmentation</td>
<td>7.49</td>
<td>6.65</td>
<td>9.05</td>
<td>8.90</td>
</tr>
<tr>
<td>vehicle</td>
<td>26.46</td>
<td><strong>31.32</strong></td>
<td><strong>30.15</strong></td>
<td><strong>30.19</strong></td>
</tr>
<tr>
<td>vowel</td>
<td>30.87</td>
<td><strong>41.29</strong></td>
<td><strong>41.67</strong></td>
<td><strong>42.23</strong></td>
</tr>
<tr>
<td>waveform</td>
<td>12.45</td>
<td>16.41</td>
<td>14.55</td>
<td>14.49</td>
</tr>
<tr>
<td>yeast</td>
<td>60.18</td>
<td>60.01</td>
<td>59.31</td>
<td>60.61</td>
</tr>
</tbody>
</table>

Table 5.7: Overfit for boosting with resampling; bold numbers correspond to an overfit that is more than 1% higher than the minimal of the 4 overfits

<table>
<thead>
<tr>
<th>database</th>
<th>AdaM2</th>
<th>AdaM2oq</th>
<th>GrPloss</th>
<th>BoostMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>0.32</td>
<td>0.38</td>
<td>0.35</td>
<td>0.32</td>
</tr>
<tr>
<td>letter</td>
<td>0</td>
<td>0.11</td>
<td>0.41</td>
<td>0.37</td>
</tr>
<tr>
<td>nursery</td>
<td>0.24</td>
<td>0.14</td>
<td><strong>1.94</strong></td>
<td><strong>2.10</strong></td>
</tr>
<tr>
<td>optdigits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pendigits</td>
<td>0.07</td>
<td>0.71</td>
<td>0.21</td>
<td>0.12</td>
</tr>
<tr>
<td>satimage</td>
<td>0.09</td>
<td>0.99</td>
<td><strong>2.68</strong></td>
<td>0.18</td>
</tr>
<tr>
<td>segmentation</td>
<td>0.05</td>
<td>0.25</td>
<td>1.03</td>
<td><strong>1.36</strong></td>
</tr>
<tr>
<td>vehicle</td>
<td>0.28</td>
<td>0.66</td>
<td>1.08</td>
<td>0.89</td>
</tr>
<tr>
<td>vowel</td>
<td>0.38</td>
<td>0.95</td>
<td><strong>1.52</strong></td>
<td><strong>1.52</strong></td>
</tr>
<tr>
<td>waveform</td>
<td>0.06</td>
<td>0.29</td>
<td>0.35</td>
<td>0.23</td>
</tr>
<tr>
<td>yeast</td>
<td>0</td>
<td>0.22</td>
<td><strong>2.15</strong></td>
<td>0.08</td>
</tr>
</tbody>
</table>
5.2.3.3 Comparison with theory

First we look, if the derived bounds for the error measures hold. More particularly we check, if the bound for the training error for AdaBoost.M2 from the theorem in section 1.3.3

\[
\text{err} \leq (|G| - 1) 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)}
\]

, the bounds for the pseudo-loss error for GrPloss from the theorem in 2.2.2

\[
\text{plerr} \leq \prod_{t=1}^{T} \sqrt{1 - U_t^2}
\]

and the bounds for the maxgroup error for BoostMA from the theorem in 3.2

\[
\text{mxerr} \leq \prod_{t=1}^{T} \left( \frac{r_t^{\pi_g} (1 - r_t)^{1 - \pi_g}}{(1 - \pi_g)^{1 - \pi_g} \pi_g^{\pi_g}} \right)
\]

hold (see figures 5.10, 5.11 and 5.12). The looser bound for \(\text{plerr}\) and \(\text{mxerr}\) is the one of the theorem and the tighter bound is the product of the normalizing constants \(\prod_t Z_t\), which occurs in the corresponding proofs. Indeed all bounds hold. The bounds for the training error are not useful except for the car example. Because of the factor \(|G| - 1\) they most often remain above 1, and so they are senseless. The bounds for \(\text{plerr}\) and \(\text{mxerr}\) are much tighter.

From the point of theory AdaBoost.M2 minimizes a bound for the training error, where GrPloss and BoostMA minimize bounds for other measures of performance. Therefore AdaBoost.M2 should be the best minimizer of the training error. However the bound for the training error is very loose. In contrast to AdaBoost.M2 the algorithms GrPloss and BoostMA minimize other sorts of error rates, but the bounds for these error rates are much tighter. This might be the cause, why their training error rate is comparable to that of AdaBoost.M2 (figure 5.9). Minimizing a looser bound for a more direct performance measure and minimizing a tighter bound for a more indirect performance measure appear to lead to equal performance.
Figure 5.10: Training error and its bound for AdaBoost.M2 with resampling.
Figure 5.11: Pseudo-loss error and its bounds for GrPloss with resampling
CHAPTER 5. EXPERIMENTS

Figure 5.12: Maxgroup error and its bounds for BoostMA with resampling
Now we look at the maxgroup error: Only for 4 databases the group proportions are clearly not balanced. For the other 8 databases group proportions were approximately the same, and there GrPloss and BoostMA did not differ. BoostMA clearly performs best with respect to the maxgroup error (figure 5.13). For the car and yeast database BoostMA only stops early, but the other 3 algorithms even increase the maxgroup error for these two datasets.

![Figure 5.13: Maxgroup error for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA](image)

Also in agreement with theory GrPloss is the best algorithm in the minimization of the pseudo-loss error (figure 5.14) and the exponential pseudo-loss (figure 5.15).
Figure 5.14: Pseudo-loss error curves for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA.
Figure 5.15: Exponential pseudo-loss for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA
5.2.3.4 Stopping criterion

For boosting with reweighting the stopping criterion

\[
\text{Stop, if } \frac{\text{mean}(\alpha_{t-9}, \ldots, \alpha_t)}{\text{mean}(\alpha_1, \ldots, \alpha_{10})} < 0.02.
\]

often worked, because \(\alpha_t\) came near enough to zero. But for boosting with resampling often \(r_t\) and therefore \(\alpha_t\) seemed to level off too early (figure 5.16), so the algorithm often did not stop (table 5.8). Recalling theory the algorithm

![Figure 5.16: Typical picture for \(\alpha\) depending on the boosting round](image)

...will not decrease the bound for the performance measure further, if \(r_t \leq 0\) and \(\epsilon_t \geq \frac{1}{2}\) respectively. But the theory will also not suggest benefits, if the bound is already small enough, i.e. smaller than \(1/N\). Looking at figures 5.10, 5.11 and 5.12 this can really happen. Therefore we propose a new stopping criterion with an additional stopping possibility:

For AdaBoost.M2 we get

\[
\text{Stop, if } \frac{\text{mean}(\alpha_{t-9}, \ldots, \alpha_t)}{\text{mean}(\alpha_1, \ldots, \alpha_{10})} < 0.02 \quad \text{or} \quad (|G|-1) \frac{2}\prod_{s=1}^{t} \sqrt{\epsilon_s} (1-\epsilon_s) \leq \frac{1}{N}.
\]

For GrPloss and BoostMA we use the tighter bound and get

\[
\text{Stop, if } \frac{\text{mean}(\alpha_{t-9}, \ldots, \alpha_t)}{\text{mean}(\alpha_1, \ldots, \alpha_{10})} < 0.02 \quad \text{or} \quad \prod_{s=1}^{t} Z_s \leq \frac{1}{N}.
\]

Since for AdaBoost.M2 the bound for the training error is so loose, this only leads to one earlier stop, which occurs for the car dataset. The bounds for the pseudo-loss error for GrPloss and the maxgroup error for BoostMA are much tighter, and so the new stopping criterion will additionally stop for 8 more and 6 more databases respectively. The original and new resulting boosting rounds are listed in table 5.8

A look at the training error curves (figure 5.3) confirms, that this additional stopping possibility is useful. For the datasets, where the training error curve is getting constant, it stops in the flat part of the training error curve. For the datasets with overfit (table 5.7) it stops in the part, where the training error curve begins to rise.
Table 5.8: Number of boosting rounds resulting from the old and the new stopping criterion for boosting with resampling. Bold numbers indicate earlier stops with the new criterion.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>AdaM2 stop</th>
<th>AdaM2 stop</th>
<th>GrPloss old</th>
<th>GrPloss new</th>
<th>BoostMA old</th>
<th>BoostMA new</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>2000</td>
<td>102</td>
<td>2000</td>
<td>19</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
<td>100</td>
<td>2000</td>
<td>100</td>
</tr>
<tr>
<td>satimage</td>
<td>1438</td>
<td>1438</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
<td>117</td>
</tr>
<tr>
<td>segmentation</td>
<td>891</td>
<td>906</td>
<td>2000</td>
<td>117</td>
<td>2000</td>
<td>117</td>
</tr>
<tr>
<td>yeast</td>
<td>463</td>
<td>458</td>
<td>2000</td>
<td>2000</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

5.2.3.5 Evolution of the margin and the group proportions

The margin of a classifier plays an important role in theory [19, 22, 23, 25, 27]. It is widely believed, that AdaBoost.M2 maximizes the margin, therefore we also recorded the margin of the resulting ensemble classifier during boosting (figure 5.17). All 4 algorithms increase the margin and as for the training error GrPloss and BoostMA seem to be faster. Plotting the test error against the margin for subsequent boosting rounds (figure 5.18) shows, that the test error is generally decreasing with increasing margin. But we can also see, that this dependence is too weak to explain differences between the algorithms. We thought, that particularly for the vowel dataset, where algorithms suffer from high generalization errors, the margin could explain the differences in the test error rates. But that was not the case, because all algorithms reach the same margin but have different test errors.

Another measure we investigated was the mean margin of the examples of the training set (figure 5.19). We thought, that the mean margin could also be a good measure of performance. This turned out to be wrong, because for many datasets the mean margin increased, although the test error increased too (figure 5.20).

We also investigated, how the algorithms concentrate on examples of certain groups. Only in [21] there is a short note about this concentration. In figures 5.21, 5.22 and 5.23 we plot the group proportions $D_t(g) = \sum_{i:g_i=g} D_t(i)$ dependent on the boosting round. One can clearly see, that the group proportions change and that the algorithms converge to a final group distribution, which is specific for the algorithm. The differences in the final group proportions can be quite big, but the ranking of the final group proportions is similar for all algorithms.
Figure 5.17: Margin for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA
Figure 5.18: Test error dependent on the margin for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: Gr-
Figure 5.19: Mean margin for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA
Figure 5.20: Test error dependent on the mean margin for boosting with resampling: solid: AdaBoost.M2, dashed: AdaBoost.M2 with constant q, dash dotted: GrPloss, dotted: BoostMA.
Figure 5.21: $D_t(g)$ dependent on the boosting round for car, digitbreiman, letter and nursery.
Figure 5.22: $D_t(g)$ dependent on the boosting round for optdigits, pendigits, satimage and segmentation.
Figure 5.23: \( D_t(g) \) dependent on the boosting round for vehicle, vowel, waveform and yeast.
5.3 Assessment of postprocessing methods

5.3.1 Assessment of the probability estimation methods

For the experiments we implemented both Problog1 and Problog2. For some datasets we had problems with the numerical solution of the system of linear equations under the normalization constraint occurring in the calculation of Problog2. For the datasets, where we didn’t have these problems, the differences between Problog2 and Problog1 were negligible. Therefore we will only talk about Problog1 for the rest of this section.

A simple but effective way to look, if the probability estimates are good, is to use slightly modified bar plots (figure 5.24). Each bar is defined by an integer $m$ and represents all instances in the training set, for which the probability estimate for being in the assigned group lies between $(m - 1)/10$ and $m/10$ for $m \in \{1, \ldots, 10\}$. The height of the bar corresponds to the error rate for these instances. The height of the symbol + corresponds to the mean probability for being in the assigned group for these instances. As it is done for mosaic plots the width of the bar is proportional to the number of instances represented by the bar. The probability estimation works well, if the height of the symbol + and the height of the bar are similar.

Figure 5.24: Bar plots for the probability estimation with Problog1 using the scores resulting from GrPloss with resampling at the final boosting round.
A look at the resulting figure for Problog1 (figure 5.24) shows, that the estimated probabilities for being in the assigned group are often too low, which means, that Problog1 is too pessimistic. Note, that in the figure the car and the optdigits dataset are excluded, because for them the error rate is zero. Therefore also the total mean estimated probability for being in the assigned group is lower than the accuracy of the classifier.

Revisiting Problog1 the normalization step had the following form

\[ p(k|x) = \frac{p(k|x)}{\sum_g p(g|x)} \]

Similar to the definition of the margin we modify the probability estimation by omitting some groups \( g \) with low \( p(g|x) \). For each \( x \) we first order the probabilities \( p(k|x) = p(g(1)|x) \geq p(g(2)|x) \geq \ldots \geq p(g(|G|)|x) \) and then we use the following normalization

\[ p(k|x) = \frac{p(k|x)}{\sum_{i=1}^n p(g(i)|x)} \]

We call \( n - 1 \) the effective number of groups, which should stand for the number of groups, which are wrong, but plausible. The case \( n = |G| \) is the original normalization step for Problog1. Making \( n \) smaller increases the estimated probability. We chose the effective number of groups so, that the resulting mean estimated probability for being in the assigned group is nearest to the accuracy of the classifier with majority voting, which is estimated by 1 minus the training error rate. The resulting bar plot is shown in figure 5.25. This modified method for the probability estimation works quite well.

### 5.3.2 Performance with probability voting

Of course one can use the probability estimates to assign a case to a group. We will say, that we assign a case \( x \) to a group with probability voting, if

\[ H(x) = \arg \max_{g \in G} p(g|x). \]

To get good probability estimates it is important, that we don’t loose discriminating power with this process. Therefore we check, if the test error with probability voting is comparable to the test error with majority voting (figure 5.26 and table 5.9). It looks, as if the performance is unchanged for AdaBoost.M2, worse for AdaBoost.M2 with constant \( q \) and even a bit better for GrPloss. Note also, that Problog1 and the improved version from the previous section assign an instance to the same group.
Figure 5.25: Bar plots for the improved probability estimation using the scores resulting from GrPloss with resampling at the final boosting round.

Table 5.9: Comparison of the majority voting with probability voting by consideration of the test error; bold and italic numbers correspond to high (>5%) and medium (>1.5%) disadvantage to the other voting method

<table>
<thead>
<tr>
<th>algorithm</th>
<th>voting method</th>
<th>AdaM2</th>
<th>AdaM2q</th>
<th>AdaM2oq</th>
<th>AdaM2oq</th>
<th>GrPloss</th>
<th>GrPloss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AdaM2</td>
<td>AdaM2q</td>
<td>AdaM2oq</td>
<td>AdaM2oq</td>
<td>AdaM2oq</td>
<td>GrPloss</td>
<td>GrPloss</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>majority</td>
<td>probability</td>
<td></td>
<td>probability</td>
<td></td>
<td>GrPloss</td>
<td>GrPloss</td>
</tr>
<tr>
<td>letter</td>
<td>27.00</td>
<td>26.94</td>
<td>26.81</td>
<td>27.06</td>
<td>27.13</td>
<td>27.83</td>
<td></td>
</tr>
<tr>
<td>nursery</td>
<td>47.18</td>
<td>48.63</td>
<td>53.33</td>
<td>53.65</td>
<td>42.38</td>
<td>41.28</td>
<td></td>
</tr>
<tr>
<td>pendigits</td>
<td>14.43</td>
<td>13.97</td>
<td>12.71</td>
<td>29.23</td>
<td>14.25</td>
<td>13.10</td>
<td></td>
</tr>
<tr>
<td>satimage</td>
<td>18.70</td>
<td>20.87</td>
<td>28.50</td>
<td>26.5</td>
<td>20.50</td>
<td>21.93</td>
<td></td>
</tr>
<tr>
<td>segmentation</td>
<td>17.85</td>
<td>17.40</td>
<td>19.40</td>
<td>19.45</td>
<td>20.20</td>
<td>17.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.27</td>
<td>7.36</td>
<td>7.49</td>
<td>18.53</td>
<td>11.04</td>
<td>8.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35.10</td>
<td>32.27</td>
<td>38.65</td>
<td>40.90</td>
<td>37.33</td>
<td>33.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>53.90</td>
<td>56.28</td>
<td>59.09</td>
<td>58.87</td>
<td>67.97</td>
<td>62.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16.70</td>
<td>16.95</td>
<td>19.39</td>
<td>32.44</td>
<td>17.91</td>
<td>17.27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60.58</td>
<td>61.32</td>
<td>60.98</td>
<td>60.71</td>
<td>63.88</td>
<td>62.53</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.26: Comparison of majority voting with probability voting for all 4 algorithms. Each point represents the error for the test data of a dataset.
5.3.3 Performance of Postrank

Postrank (section 4.4) is an ad-hoc method without theoretical confirmation, that had good performance in preliminary trials, where we analyzed smaller versions of the digitbreiman and the waveform dataset with AdaBoost.M2 and an earlier version of BoostMA similar as in 3.1.

The results of the preliminary trials could be confirmed, but for most other databases the performance of Postrank was very bad. For the nursery and yeast datasets Postrank applied to the scores from AdaBoost.M2 even leads to error rates more than 20% higher than the error rates of simple majority voting. In table 5.10 and figure 5.27 one can clearly see, that Postrank overall worsens the algorithms.

Table 5.10: Comparison of majority voting with Postrank by consideration of the test error; bold and italic numbers correspond to high(>5%) and medium(>1.5%) disadvantages to the other voting method

<table>
<thead>
<tr>
<th>algorithm</th>
<th>AdaM2</th>
<th>AdaM2q</th>
<th>GrPloss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>majority</td>
<td>Postrank</td>
<td>majority</td>
</tr>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digitbreiman</td>
<td>27.00</td>
<td>27.26</td>
<td>26.81</td>
</tr>
<tr>
<td>letter</td>
<td>47.18</td>
<td>49.79</td>
<td>53.33</td>
</tr>
<tr>
<td>nursery</td>
<td>14.43</td>
<td><strong>34.81</strong></td>
<td>12.71</td>
</tr>
<tr>
<td>optdigits</td>
<td>0</td>
<td>0.17</td>
<td>0</td>
</tr>
<tr>
<td>pendigits</td>
<td>18.70</td>
<td>22.87</td>
<td>28.50</td>
</tr>
<tr>
<td>satimage</td>
<td>17.85</td>
<td>19.95</td>
<td>19.40</td>
</tr>
<tr>
<td>segmentation</td>
<td>8.27</td>
<td>12.47</td>
<td>7.49</td>
</tr>
<tr>
<td>vehicle</td>
<td><strong>35.10</strong></td>
<td>33.57</td>
<td>38.65</td>
</tr>
<tr>
<td>vowel</td>
<td>53.90</td>
<td><strong>61.04</strong></td>
<td>59.09</td>
</tr>
<tr>
<td>waveform</td>
<td>16.70</td>
<td>17.02</td>
<td>19.39</td>
</tr>
<tr>
<td>yeast</td>
<td>60.58</td>
<td><strong>84.38</strong></td>
<td>60.98</td>
</tr>
</tbody>
</table>
Figure 5.27: Comparison of majority voting with Postrank for all 4 algorithms. Each point represents the error for the test data of a dataset.
Chapter 6

Summary

6.1 Common state of the art and active research areas

The main goal of this dissertation was to find and explore new boosting algorithms for multiclass problems using decision stumps as base classifiers. There are two general approaches in the literature of constructing boosting classifiers for multiclass problems.

The first one reduces one multiclass problem to several twoclass problems with the use of error-correcting codes. This approach was briefly introduced in section 1.3.4. Boosting is a time consuming, iterative process and the resulting twoclass problems can easily be solved in parallel. However it is required, that the user has the resources for parallel computing, otherwise the problem of high computing time is intensified for error-correcting codes. There is also not a strong theory behind error-correcting codes. At its current state it is more an ad-hoc method without strong theory behind. But one must admit, that it works well in experiments with benchmark datasets.

The other approach addresses the multiclass problem more directly. The most frequently used algorithm in the literature is AdaBoost.M1 (sections 1.3.1 and 1.3.2), which is a straightforward generalization of AdaBoost for 2 groups (section 1.2). Since most often classification trees are used as base learners, the requirement, that the error rate of a base learner is below $1/2$, is normally easily fulfilled. But for decision stumps this condition can be hard to fulfill for multiclass problems (e.g. for 9 of the 12 datasets used in this dissertation the condition is not fulfilled).

The algorithm AdaBoost.M2 (section 1.3.3) overcomes this problem with the introduction of the pseudo-loss, which should be minimized instead of the error rate by the base learner. A reduction of an upper bound for the training error is guaranteed as long as the pseudo-loss of the base learner is less than $1/2$, which is easy to achieve also for weak classifiers as decision stumps. The disadvantage
of this algorithm is, that the base learner has to be redesigned, which may be laborious or even impossible, e.g. if one wants to use commercial programs as base learners. It also turned out, that the time for calculating a base classifier, which minimizes the pseudo-loss, is usually greater than for a base classifier, which minimizes the error rate. If one calculates for many boosting rounds this can be an important factor.

In the last years a theoretical framework, which sees boosting as gradient descent in function space, was developed (section 1.4). Many special algorithms of boosting can be unified (more or less exactly) by this framework. It can be also used for the design of new algorithms as it was done with GrPloss.

Until now the algorithms were discussed, which are the basis of this dissertation and which are all described in chapter 1. Now the two newest and most active boosting research areas will be briefly discussed.

The gradient descent framework is also used to find other algorithms (e.g. DOOM), which should minimize new loss functions. These loss functions are usually dependent on the margin (section 1.2). The goal is, that the resulting algorithms are less sensible to noise. Another approach for the design of more robust algorithms combines boosting with the theory about support vector machines (SVM) by the formulation of a minimization problem. For SVMs robustness is achieved by the introduction of so-called "slack-variables". However sensitivity with respect to noise is not the focus of this dissertation.

Another very young research direction builds "random forests" from individual trees. These trees are different, because randomness is injected, either by selection of random input variables or random features (most commonly linear combinations of input variables). A big advantage is, that the individual trees can be computed in parallel. Also an asymptotic theory is already developed.

### 6.2 Algorithms derived in this dissertation

The most important information from the literature, which was relevant for this dissertation is contained in chapter 1, the subsequent chapters contain the derivation of the new algorithms.

#### 6.2.1 GrPloss

The basis for this algorithm is the gradient descent framework of section 1.4. The exponential pseudo-loss was chosen as the loss function, which should be minimized.

The specialization of the framework for this pseudo-loss was not straightforward. First a reformulation of the usual formulation of the framework for multiclass problems was made in order to make everything more similar to the other algorithms. But this is just a minor point. Then the algorithm for exponential loss-functions, which are little more general than exponential pseudo-loss-functions,
was derived. This derivation was quite similar to the original one with the big difference, that additionally an update rule for the sampling distribution could be derived. The general algorithm doesn’t contain such an update rule. It also does not provide a step size $\alpha_t$. Then the pseudo-loss error was introduced, which lead to the choice for $\alpha_t$ by the minimization of the pseudo-loss error. The derivation of step size is similar to the derivation of the step size for confidence-rated boosting. GrPloss is guaranteed to minimize an upper bound for the pseudo-loss error exponentially as long as the base classifier is strong enough. The condition for the base classifier is very weak, so that the algorithm is suited for weak base classifiers.

For the special case of decision stumps a slightly different step size was derived. Additional simplifications for this case lead to the final algorithm. Interestingly GrPloss for decision stumps and BoostMA turned out to be extremely similar, they are exactly the same, if all group proportions are equal. This was the case for 8 of the 12 datasets, which were investigated. Therefore the comparison between GrPloss and BoostMA was limited to 4 datasets, where the two algorithms differed. There GrPloss had a big advantage for the car dataset, for the other 3 datasets the performance was equal.

6.2.2 AdaBoost.M1W

The algorithm AdaBoost.M1W is an ad-hoc algorithm, which evolves from a simple modification of just one line of AdaBoost.M1. However in experiments with datasets this modification made AdaBoost.M1 work for weak base classifiers. The big advantage of this algorithm is, that it is applicable, if one has a base classifier, whose output is just the group without confidences. Therefore it has a wider applicability. However it is just an ad-hoc method without any theoretical fundament in the form of a guaranteed decrease of an upper bound for the training error. The question about the performance of AdaBoost.M1W compared to the performance of AdaBoost.M1 for datasets, where the conditions for AdaBoost.M1 are fulfilled, is still open and should be investigated further.

6.2.3 BoostMA

BoostMA is the successor of AdaBoost.M1W. The base classifier is additionally assumed to give confidences. The maxgroup-error was introduced and a choice for the step size was derived by the minimization of a bound for the maxgroup error similar as for GrPloss. BoostMA is guaranteed to minimize this upper bound exponentially as long as the base classifier is strong enough. The condition for the base classifier is very weak, so that the algorithm is suited for weak base classifiers.
6.2.4 Problog1, Problog2 and Postrank

A second goal of this dissertation was not only to assign an object to a group, but also to provide estimates for the probability of being in a group. The output of the algorithms consists of a score for each of the $|G|$ groups. An object gets assigned to the group with the highest score. The probability estimates are got by a re-analysis of the $|G|$ scores with binary logistic regressions.

Problog 1 combines these $|G|$ probabilities with a simple normalization step.

Problog 2 combines them more systematically, which leads to a system of linear equations with a normalization constraint.

The probability estimation method could be additionally generalized by introducing multivariate logistic regressions and building a connection to error-correcting codes.

Postrank is a simple ad-hoc method, which substitutes the scores by ranks and uses majority voting with these ranks.

6.3 Results of the experiments

The experimental part consisted of the application of AdaBoost.M2, GrPloss, BoostMA and a simplified version of AdaBoost.M2 to 12 multiclass datasets. Both boosting by reweighting and boosting with resampling were implemented for all algorithms. Boosting with resampling was clearly superior to boosting with reweighting for all algorithms. With resampling the base hypothesis space is bigger, which results in more flexible base classifiers leading to better results. Therefore boosting with resampling was investigated further.

An important investigation is to check, that the bounds for the training error of AdaBoost.M2, the bounds for the pseudo-loss error of GrPloss and the bounds for the maxgroup error of BoostMA hold. This check also shows, that the bound for the training error of AdaBoost.M2 is so loose, that it is surprising, that AdaBoost.M2 functions. The bounds for the performance measures of GrPloss and BoostMA are much tighter, but these performance measures are not bounds for the classification error. From point of theory one therefore has to choose between an algorithm, which minimizes a loose bound for a direct performance measure, and two algorithms, which minimize tighter bounds for a more indirect performance measure. All three algorithms have comparable performance, measured by test and crossvalidation error respectively.

The margin of a classifier plays an important role in boosting theory, therefore the evolution of the margin dependent on the boosting round was recorded. All algorithms increase the margin. The test error is generally decreasing with increasing margin, so the margin is a reasonable performance measure. But the dependence between margin and the test error is too weak to explain differences between the algorithms. Also the mean margin of the training examples was recorded, but it turned out to be a very bad performance measure.
The results of the experiments lead to the proposal of a new additional possibility to stop the algorithm, which surprisingly has never been proposed in any paper before. It simply stops, when theory doesn’t suggest improvements of the performance measures, i.e. when the bounds of the performance measures get smaller than $1/N$. This additional stopping possibility turned out to be useful. A cause for the lack of this criterion in the literature may be, that for many experiments in the literature the number of boosting rounds is set to quite a low, fixed number, because the base classifiers are usually trees, and not so many boosting rounds are needed there. Another cause may be the looseness of the bound for the training error.

Additionally the generalization error, the overfit, the exponential pseudo-loss and the concentration of the algorithms on certain groups were described.

Despite promising results in preliminary trials with the waveform and the digit example Postrank had clearly worse results compared to majority voting. The difference between Problog1 and Problog2 were negligible, but for some datasets numerical problems occured for Problog2.

The probability estimation with Problog1 worked well, however there was often an underestimation of the probability of being in the assigned group. The result could be improved with a little modification of the normalization step. Using the probability estimates for voting leads to error rates comparable to majority voting, so there is no loss in accuracy due to probability estimation.
List of Symbols

\( \mathcal{L} \) learning set: \( \mathcal{L} = \{(x_1, g_1), \ldots, (x_N, g_N) \}; \ x_i \in \mathbb{X}, \ g_i \in \mathbb{G} \)

\( \mathbb{G} \) space of groups: \( \mathbb{G} = \{1, \ldots, |\mathbb{G}|\} \text{ or } \mathbb{G} = \{-1, +1\} \)

\( \mathbb{X} \) space of attributes: \( \mathbb{X} = \mathbb{X}_1 \times \ldots \times \mathbb{X}_p \)

\( N \) number of instances in the learning set

\( N_g \) number of instances in the learning set from group \( g \)

\(||[]\||\) useful function on boolean values: \( ||true|| = 1, ||false|| = 0 \)

\( \arg \max_{m \in M} u(m) \) maximizer of function \( u : u(\arg \max_{m \in M} u(m)) = \max_{m \in M} u(m) \)

\( \mathcal{H} \) space of base classifiers

\( h \) single base classifier: \( h : \mathbb{X} \to \mathbb{G} \) or confidence-rated \( h : \mathbb{X} \times \mathbb{G} \to [0, 1] \)

\( \mathcal{F} \) space of final classifiers: \( \mathcal{F} = \text{lin}(\mathcal{H}) \)

\( f \) single final classifier: \( f(x) = \sum_t \alpha_t h_t(x) \) for \( \mathbb{G} = \{\pm 1\} \) or \( f(x, g) = \sum_t \alpha_t h_t(x, g) \)

\( H \) single final classifier: \( H(x) = \text{sign}(f(x)) \) or \( H(x) = \arg \max_{g \in \mathbb{G}} f(x, g) \)

\( z_i \) \( i \)-th element of the learning set: \( z_i := (x_i, g_i) \)

\( l(f, z_i) \) loss function \( l : \mathbb{R}^{|\mathbb{G}|} \times \mathbb{G} \to \mathbb{R}_{\geq 0} \)

\( L(f) \) loss function \( L(f) = \frac{1}{N} \sum_{i=1}^N l(f, z_i) \)

\( \nabla L(f) \) gradient of loss function \( \nabla L(f) = \left( \frac{\partial L(f)}{\partial f(x, g)} \right) \)

\( 1_x, 1_{xg} \)

\( 1_x(y) = \begin{cases} 1 & y = x, 0 \end{cases} \quad 1_{xg}(x', g') = \begin{cases} 1 & (x, g) = (x', g') \end{cases} \)

\( \langle \cdot, \cdot \rangle \) inner product on \( \mathcal{F} : \langle f, h \rangle = \frac{1}{N} \sum_{i=1}^N f(x_i)h(x_i) \) or \( \langle f, h \rangle = \frac{1}{N} \sum_{i=1}^N \sum_{g \in \mathbb{G}} f(x_i, g)h(x_i, g) \)

\( \text{pseudo-loss} \)

\( \text{pseudo-loss}(i) = \frac{1}{2} (1 - f(x_i, g_i)) + \sum_{g \neq g_i} \frac{1}{|\mathbb{G}| - 1} f(x_i, g) \)

\( v(f, z_i) \)

\( v(f, z_i) = v_0 + \sum_g v_g(f(x_i, g), g) \), exponential loss \( l(f, z_i) = \exp[v(f, z_i)] \)

\( u(f, z_i) \)

\( u(f, z_i) = \frac{1}{2} (f(x_i, g_i) - \sum_{g \neq g_i} \frac{1}{|\mathbb{G}| - 1} f(x_i, g)) \)

\( \text{perr} \)

\( \text{pseudo-loss error: perr} = \frac{1}{N} \sum_{i=1}^N [f(x_i, g_i) < \frac{1}{|\mathbb{G}| - 1} \sum_{g \neq g_i} f(x_i, g)] \)

\( U_t \)

\( U_t = 2 \sum_{i=1}^N D_t(i)u(h_t, z_i) \)

\( r_t \)

\( r_t = \sum_{i=1}^N D_t(i)h_t(x_i, g_i) \)

\( \text{mxerr} \)

\( \text{maxgroup error: mxerr} = \sum_{i=1}^N [f(x_i, g_i) < \pi_g] \) for \( \pi_g \in (0, 1) \)

\( p_i \)

\( \text{estimated conditional probability for object } x: p_i = (p(1|x), \ldots, p(|\mathbb{G}||x)) \)

\( a_x,g \)

\( \text{codematrix for error-correcting codes: } a_{x,g} : \text{codeword for group } g \)

\( N_{m}(g) \) number of instances in leave \( m \) from group \( g \)
List of Figures

1.1 Decision stump ............................................. 7
1.2 Algorithm Bagging ......................................... 7
1.3 General boosting algorithm ................................. 8
1.4 Algorithm AdaBoost for 2 groups .......................... 10
1.5 Algorithm AdaBoost.M1 ................................. 16
1.6 Algorithm AdaBoost.M2 ................................. 18
1.7 Algorithm AdaBoost.M2, modern notation .................. 19
1.8 Algorithm error-correcting output codes .................. 20
1.9 Algorithm numerical gradient descent .................... 23
1.10 Algorithm gradient descent in function space ............... 24
1.11 Examples for margin cost functions ........................ 27
2.1 Gradient descent for exponential loss functions .......... 33
2.2 Algorithm GrPloss ......................................... 37
2.3 Algorithm GrPloss for decision stumps ..................... 42
3.1 Algorithm AdaBoost.M1W .................................. 45
3.2 Training (solid) and test error (dash-dotted) of AdaBoost.M1W dependent on the number of boosting rounds. The vertical line denotes $t_{stop}$ ......................... 46
3.3 Algorithm BoostMA ......................................... 50
4.1 Distribution of score $f(x, 1)$ for cases in group 1 and other groups 52
LIST OF FIGURES

4.2 Algorithm Problog1 .................................................. 53
4.3 Algorithm Problog2 .................................................. 55
4.4 Algorithm Probecoc .................................................. 57
4.5 Algorithm Postrank .................................................. 59

5.1 Pseudo-loss curve for AdaBoost.M2 for reweighting and resampling 65
5.2 Test error for boosting with reweighting ............................... 67
5.3 Training error curves for boosting with reweighting ................... 68
5.4 Curves for \( r_t \) for boosting with reweighting ....................... 69
5.5 Pseudo-loss error curves for boosting with reweighting ............... 70
5.6 Comparison between reweighting and resampling ..................... 71
5.7 Test error for boosting with resampling ............................... 73
5.8 Test error curves for boosting with resampling ....................... 74
5.9 Training error curves for boosting with resampling ................... 75
5.10 Training error and its bound for AdaBoost.M2 with resampling ... 79
5.11 Pseudo-loss error and its bounds for GrPloss with resampling ..... 80
5.12 Maxgroup error and its bounds for BoostMA with resampling ... 81
5.13 Maxgroup error curves for boosting with resampling ............... 82
5.14 Pseudo-loss error curves for boosting with resampling ............... 83
5.15 Exponential pseudo-loss curves for boosting with resampling .... 84
5.16 Stopping criterion: \( \alpha \) levels off ................................ 85
5.17 Margin for boosting with resampling ................................ 87
5.18 Test error dependent on the margin .................................. 88
5.19 Mean margin for boosting with resampling ............................ 89
5.20 Test error dependent on the mean margin ............................. 90
5.21 \( D_t(g) \) dependent on the boosting round for car, digitbreiman, letter and nursery .................................................. 91
5.22 \( D_t(g) \) dependent on the boosting round for optdigits, pendigits, satimage and segmentation. .................. 92
\section*{LIST OF FIGURES}

5.23 $D_t(g)$ dependent on the boosting round for vehicle, vowel, waveform and yeast. ...................................................... 93

5.24 Bar plots for the probability estimation with Problog1 .............. 94

5.25 Bar plots for the improved probability estimation ..................... 96

5.26 Comparison of majority voting with probability voting .............. 97

5.27 Comparison of majority voting with Postrank .......................... 99
List of Tables

5.1 Properties of the databases . . . . . . . . . . . . . . . . . . . . . . . . . . 62
5.2 Group frequencies for databases with unbalanced groups . . . . . . 62
5.3 Experimental setup for databases . . . . . . . . . . . . . . . . . . . . . . . 63
5.4 Test error for boosting with resampling . . . . . . . . . . . . . . . . . . . 73
5.5 Generalization error for boosting with resampling . . . . . . . . . . 76
5.6 Minimal training error for boosting with resampling . . . . . . . . . 77
5.7 Overfit for boosting with resampling . . . . . . . . . . . . . . . . . . . . 77
5.8 New stopping criterion for boosting with resampling: new T . . . 86
5.9 Comparison of majority voting with probability voting . . . . . . 96
5.10 Comparison of majority voting with Postrank . . . . . . . . . . . . 98
Bibliography


Curriculum Vitae

Personal Data

name Guenther Eibl

date and place of birth 14.5.1971 in Salzburg

parents Hedda and Prof. Mag. Hans Eibl

nationality Austria

family status single

Education

1977-1981 Volksschule Pradl Ost I in Innsbruck

1981-1989 Bundesgymnasium and Bundesrealgymnasium
Reithmannstrasse in Innsbruck

30.5.1989 school leaving examination

Oktober 89 - September 90 military service

Oktober 1990 begin of the studies mathematics and physics

WS 94/95 - WS 95/96 tutorials in mathematics

WS 95/96 - SS 97 member of the "Studienrichtungsvertretung Mathematik"

july 97 final exam of the mathematics study

SS 98 - 1.11.1998 work for diploma thesis in physics

1.11.1998 - present university assistant at the institute of biostatistics in Innsbruck, Austria