Ensemble-Lernmethoden zur Klassifikation

Eine Analyse von Boosting-Algorithmen

Diplomarbeit



vorgelegt von
Gunnar Rätsch
geboren in Neuruppin

am
Institut für Informatik
der
Universität Potsdam

Betreuung
Prof. Dr. rer. nat. Peter Behr
Dr. rer. nat. Klaus-Robert Müller

21. April 1998

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Quellen angefertigt zu haben.

Ich erkläre mein Einverständnis mit der öffentlichen Aufstellung dieser Arbeit in der Bibliothek des Instituts für Informatik der Universität Potsdam.

Berlin, den 21. April 1998

Danksagung

Die vorliegende Arbeit wurde im Rahmen des Forschungsprojektes CANDY am Institut FIRST der GMD Berlin angefertigt. An dieser Stelle möchte ich allen danken, die mir während dieser Zeit hilfreich zur Seite standen.
Besonders gilt mein Dank Herrn Dr. rer. nat. Klaus-Robert Müller vom Institut FIRST der GMD und Herrn Prof. Dr. rer. nat. Peter Behr vom Institut für Informatik der Universität Potsdam für die engagierte Betreuung dieser Arbeit.
Außerdem sei Takashi Onoda, Bernhard Schölkopf, Steven Lemm, Alexander Smola, Sebastian Mika, Andreas Ziehe und nicht zuletzt Jens Kohlmorgen für zahlreiche Diskussionen und Hinweise herzlich gedankt.
Auch danke ich der GMD FIRST Berlin für die Bereitstellung von Rechentechnik und Software.
Last but not least möchte ich auch meiner Freundin Anja danken, daß sie mich bei meiner Arbeit unterstützt hat und sie so verständnisvoll und geduldig war.

Zusammenfassung

In dieser Arbeit werden *Ensemble-Lernmethoden zur Klassifikation* untersucht. Dazu wird insbesondere auf *AdaBoost* und *Arcing* eingegangen. Unter welchen Bedingungen diese Algorithmen gute Ergebnisse liefern und wie robust sie gegen die *Verrauschung der Trainingsdaten* sind, ist die Grundfrage dieser Arbeit. Dazu wird theoretisch und intuitiv erklärt, warum diese Algorithmen funktionieren und gezeigt, welche *Zielfunktion* von *AdaBoost* optimiert wird. Es wird herausgearbeitet, daß *AdaBoost* einem *Annealing-Prozeß* gleicht und asymptotisch ein *lineares Optimierungsproblem* approximativ löst. Dabei wird herausgestellt, daß AdaBoost asymptotisch *nicht robust* gegen die Verrauschung der Trainingsdaten ist. Um die gewünschte Robustheit zu erreichen, werden aus den theoretischen Überlegungen verschiedene Verbesserungen abgeleitet. Dazu wird das Konzept der *Slack-Variablen* in *AdaBoost* eingearbeitet und aus der Sicht der *Margin-Verteilungen* motiviert.
Die theoretischen Überlegungen werden durch Experimente mit künstlichen und realen Datensätzen untermauert. In einem experimentellen Vergleich wird gezeigt, daß die Verbesserungen für *AdaBoost* sehr oft bessere Generalisierungsergebnisse erreichen als der originale AdaBoost-Algorithmus. Es werden bessere durchschnittliche Ergebnisse erreicht, als für *Support-Vektor-Algorithmus*, von dem angenommen wird, daß er hervorragende Klassifikatoren erzeugt.

Abstract

In this thesis *ensemble learning methods for classification* are presented, especially *AdaBoost* and *Arcing* algorithms are analyzed. It is theoretically shown why *AdaBoost* and *Arcing* are working well and explained what kind of *target function* is optimized. It is pointed out that AdaBoost's optimization is similar to a *annealing process* and *linear programming*. From an asymptotic analysis of AdaBoost's learning process it is shown that those algorithms are *not noise robust* asymptotically.
In order to get a noise robustness, several improvements of *AdaBoost* are proposed. The concept of *slack variables* is introduced into AdaBoost and is motivated from the margin's point of view.
These improvements are experimentally compared for artificial and real-world datasets. They generally show much better results than the original AdaBoost algorithm and also a better overall performance than the *Support Vector algorithm*, which is considered to be an excellent classifier.

Inhaltsverzeichnis

1 Überblick **1**

2 Ensemble-Lernmethoden **3**
2.1 Bagging 4
2.2 AdaBoost 5
2.3 Arcing 8
2.4 Bias und Varianz 11
2.5 Margin-Verteilungen 14

3 Warum AdaBoost oft gute Ergebnisse liefert **16**
3.1 Experimentelle Ergebnisse 16
3.2 AdaBoost gleicht einem Gradienten-Verfahren 21
3.3 AdaBoost ist ein Annealing-Prozeß 26
3.4 Wie groß ist der Margin? 30

4 Warum AdaBoost verbessert werden sollte **33**
4.1 AdaBoost ist nicht noise-robust 33
4.2 Marginmaximierung vs. Overfitting 35

5 Ansätze zu Verbesserungen des AdaBoost-Algorithmus **39**
5.1 Optimierung von ϕ 39
5.2 Regularisierung durch veränderte Kostenfunktion 41
5.3 Lineare Programmierung zur Maximierung der Stabilitäten 45
5.4 Quadratische Programmierung zur Maximierung der Stabilitäten 48

6 Experimenteller Vergleich der Methoden **51**
6.1 Datensätze und Modellselektion 51
6.2 Simulationen 55

7 Zusammenfassung **62**

Anhang **64**
A.1 Klassifikation mit RBF-Netzen 64
A.2 Boosting 71
A.3 Datensätze 73
A.4 Implementation 75

Kapitel 1

Überblick

In den letzten Jahren wuchs das Interesse an Lern-Algortihmen zur Klassifkation, die niedrige Generalisierungsfehler durch Kombination vieler Einzelhypothesen (Ensemble von Klassifikatoren) erreichen. Mit *Ensemble-Lernmethoden* wurden für *Decision-Tree-Lernalgorithmen* wie z.B. C4.5 und CART große Verbesserungen erreicht [5, 7, 16, 22, 39].

Mit *Boosting* [20] als eine spezielle Art einer Ensemble-Lernmethode wurden die bisher besten – bis heute ungeschlagenen – Ergebnisse in der Handschrifterkennung erreicht [39]. Vor nur drei Jahren wurde mit *AdaBoost* [21] eine Methode entwickelt, mit der man viele schwache Hypothesen zu Endhypothesen kombinieren kann, die in vielen Fällen gute Ergebnisse im Vergleich zu den Einzelhypothesen liefern [4, 6, 8, 16, 17, 20, 39, 53].

Bis heute wurde der AdaBoost-Algorithmus auf viele Arten verschiedener Einzelhypothesen (Neuronale Netze, Decision Trees usw.) angewendet – ebenfalls mit gutem Erfolg. Warum AdaBoost in einigen Fällen, insbesondere bei Vorhandensein von *Rauschen in den Trainingsdaten*, schlechte Ergebnisse liefert, konnte jedoch nicht erklärt werden.

Als ersten Schritt zum theoretischen Verständnis wurde im letzten Jahr in Schapire et al. [53] versucht, eine Erklärung für die gute Generalisierungsfähigkeit von AdaBoost anhand der Verteilung der Abstände der Beispiele von der Entscheidungslinie (*Margin-Verteilungen*) zu finden. Diese Arbeiten wurden in Breiman [7, 6, 8] fortgesetzt und es entstand eine Verallgemeinerung von AdaBoost: das *Arcing*. Dabei wurde festgestellt, daß das Ensemble bessere Generalisierungseigenschaften hat, wenn die Abstände der Beispiele von der Entscheidungslinie (*Margin*) möglichst groß sind.

Erst vor einigen Monaten wurde diese Idee in Grove et al. [27] aufgegriffen und versucht, mit Hilfe der linearen Optimierung Klassifikatoren zu finden, die einen größeren Margin haben. Dabei wurde festgestellt, daß diese Art der Veränderungen des Ensembles meist zu einer Verschlechterung der Generalisierungsfähigkeit führt.

Davon ausgehend wird in dieser Arbeit versucht, die folgenden Fragen zu beantworten:

1. Wie funktioniert AdaBoost? Welche Zielfunktion wird bei AdaBoost optimiert?
2. Wie robust ist der AdaBoost gegen die Verrauschung der Trainingsdaten?
3. Welchen Zusammenhang gibt es zwischen AdaBoost, der *Maximum-Margin-Klassifikation* und dem *Support-Vektor-Algorithmus* [3, 14, 64] ?
4. Was kann man an AdaBoost noch verbessern?

Dazu wird im Kapitel 2 ein kurzer Überblick über die gebräuchlichsten Ensemble-Lernmethoden (*Bagging*, AdaBoost, Arcing) gegeben. Außerdem werden wichtige Sätze und Definitionen gegeben und erläutert.
In Kapitel 3 werden zunächst die Vorteile des Lernens im Ensemble experimentell dargestellt. Dann wird der *AdaBoost-ähnliche Algorithmus* definiert und analysiert. Es wird gezeigt, welche Zielfunktion bei AdaBoost optimiert wird und außerdem, daß AdaBoost asymptotisch die Klassifikation mit einem großen Margin anstrebt.

Im folgenden Kapitel 4 werden die herausgearbeiteten Eigenschaften von AdaBoost bewertet und gezeigt, daß AdaBoost zu Overfitting fähig und nicht noise-robust ist. Daran schließt sich eine Diskussion über die *Klassifikation mit einem großen Margin* an. Es werden die Probleme von AdaBoost herausgestellt.

Als Ergebnis werden in Kapitel 5 einige Verbesserungen für AdaBoost vorgeschlagen. Diese werden aus den vorherigen Betrachtungen abgeleitet. Dafür wird einerseits versucht, AdaBoost so zu verändern, daß es robuster gegen die Einflüsse von Rauschen ist; andererseits wird (wie in Grove et al. [27]) nach Möglichkeiten gesucht, das Ensemble nachträglich durch *lineare und quadratische Optimierung* zu verbessern.

In einem experimentellem Vergleich der Methoden wird gezeigt, daß diese Modifikationen von AdaBoost zu signifikanten Verbesserungen führen. Die quantitativen Verbesserungen geben Aufschluß darüber, wie wichtig die in den vorherigen Kapiteln gemachten Betrachtungen im einzelnen sind.

Kapitel 2

Ensemble-Lernmethoden

In diesem Kapitel wird auf Ensemble-Lernmethoden, insbesondere auf *Bagging* (Breiman [5]), *Arcing* (Breiman [7]) und *AdaBoost* (Freund et al. [21]) eingegangen. Es werden die für die nächsten Kapitel notwendigen Sätze und Begriffe eingeführt.
Als *Ensemble-Lernmethoden* bezeichnet man Lernalgorithmen, die auf dem Prinzip der Kombination von mehreren Hypothesen beruhen. Sie benutzen einen Lernalgorithmus $\mathcal{L}$ zum Trainieren von *Basishypothesen*. Außerdem bestimmen sie, wie die Basishypothesen zu einer *Gesamthypothese* (*kombinierte Hypothese*, *Endhypothese*) kombiniert werden sollen.
Bei den Betrachtungen in dieser Arbeit wird nur die Klassifikation in zwei Klassen betrachtet. Die Analyse der Ensemble-Lernmethoden wird dadurch vereinfacht. Die Ergebnisse lassen sich größtenteils auf die Klassifikation mit mehreren Klassen übertragen.
Die Menge der möglichen Basishypothesen[1] wird durch den Lernalgorithmus für die Basishypothesen $\mathcal{L} : \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle =: \mathbf{Z} \mapsto h^{\mathbf{Z}}$ festgelegt und definiert eine Funktionenklasse

$$\mathcal{H}^{\mathcal{L}} := \{h : x \mapsto \{-1, 1\} \text{ , wobei } h \text{ Ausgabe von } \mathcal{L} \text{ ist}\} \ . \tag{2.1}$$

Die Basishypothesen h_t werden bei den hier betrachteten Methoden *konvex* zur Endhypothese f kombiniert:

$$f \in \mathcal{C} := \left\{ g : x \mapsto \sum_{t=1}^{T} c_t h_t(x) \text{ , wobei } c_t > 0, \ \sum_{t=1}^{T} c_t = 1, \ h_t \in \mathcal{H}^{\mathcal{L}} \right\} \tag{2.2}$$

wobei T die Anzahl der kombinierten Hypothesen und $\{c_t\}_{t=1}^{T}$ die *Gewichte* der einzelnen Hypothesen sind. $\mathcal{C}$ ist die *konvexe Hülle* von $\mathcal{H}^{\mathcal{L}}$. Damit ist klar, daß $f(\mathbf{x}) \in [-1, 1]$.
Welche Basishypothesenklasse $\mathcal{H}^{\mathcal{L}}$ verwendet wird, spielt eine zunächst untergeordnete Rolle. Die verschiedenen Ensemble-Lernmethoden unterscheiden sich in der Art der Kombination der Hypothesen, d.h. wie die Gewichte $\{c_t\}_{t=1}^{T}$ gewählt werden, und mit Hilfe welcher Trainingsdaten der Algorithmus $\mathcal{L}$ die Hypothese h_t findet.
In Abbildung 2.1 findet man den allgemeinen, einer Ensemble-Lernmethode zugrunde liegenden Algorithmus. Dabei ist zu beachten, daß die in Zeile 1 erzeugte Trainingsmenge durchaus gewichtet sein kann (vgl. AdaBoost und Arcing).

[1] In dieser Arbeit werden nur Basishypothesen der Form $h : x \mapsto \{-1, 1\}$ betrachtet, wobei meist $x \in \mathbb{R}^a$ und a die Dimension der Eingabedaten ist. Diese sind einfacher zu behandeln. Viele Ergebnisse lassen sich

Algorithm Ensemble-Lernmethode

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$

Basislernalgorithmus $\mathcal{L}$

Anzahl der zu kombinierenden Hypothesen T

Do for $t = 1{:}T$,

1. Erzeuge Trainingsmenge $\mathbf{Z}_t$ der Größe l aus $\mathbf{Z}$
2. Starte $\mathcal{L}$ mit $\mathbf{Z}_t$ und erhalte $h_t : \mathbf{x} \mapsto \{-1, 1\}$
3. Bestimme c_t

Output:

kombinierte Hypothese

$$f(\mathbf{x}) = \sum_{t=1}^{T} c_t h_t(\mathbf{x})$$

Abbildung 2.1: Die allgemeine Form einer Ensemble-Lernmethode

Wie üblich wird bei der Klassifikation ein Schwellenwert (*threshold*) benutzt, um die kontinuierliche Funktion $f : x \mapsto [-1, 1]$ für die Klassifikation verwenden zu können. Naheliegend ist die signum-Funktion:

$$\begin{aligned} \tilde{f}(\mathbf{x}) &= \operatorname{sgn}(f(\mathbf{x})) && (2.3) \\ &= \operatorname{sgn}\left(\sum_{t=1}^{T} c_t h_t(\mathbf{x})\right) \\ &= \begin{cases} 1 & \text{wenn } \sum_{t=1}^{T} c_t I(h_t(\mathbf{x}) = 1) \geq \frac{1}{2} \\ -1 & \text{sonst} \end{cases}, && (2.4) \end{aligned}$$

wobei $I(true) = 1$ und $I(false) = 0$ ist.

Für die Analyse ist jedoch oft die Betrachtung von f selbst notwendig, da z. B. $|f(\mathbf{x})|$ etwas über die *Sicherheit der Klassifikation* aussagt [53].

2.1 Bagging

Bagging ist eine der einfachsten Ensemble-Lernmethoden. Der Name leitet sich von **b**ootstrap **agg**regat**ing** ab [5].
Das Trainingsset $\mathbf{Z}_t$ wird in jedem Schritt durch *zufällige Auswahl mit Zurücklegen* (l mal)

auf $h : x \mapsto [-1, 1]$ verallgemeinern.

aus $\mathbf{Z}$ erhalten (man bezeichnet $\mathbf{Z}_t$ als ein *Bootstrap-Replikat von* $\mathbf{Z}$). Bei diesem Vorgehen sind durchschnittlich nur ca. 63% aller Trainingsbeispiele im Replikat zu finden [5]. Die restlichen Trainingsdaten treten nicht auf.
Die Gewichte der Hypothesen sind alle gleich, d.h. $c_t = 1/T$, $t = 1, \ldots, T$. Man erhält die in Abbildung 2.2 dargestellte einfache Form.

Algorithm Bagging

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle (\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)}) \rangle$

Basislernalgorithmus $\mathcal{L}$

Anzahl der zu kombinierenden Hypothesen T

Do for $t = 1{:}T$,

1. Erzeuge Bootstrap-Replikat $\mathbf{Z}_t$ aus $\mathbf{Z}$
2. Starte $\mathcal{L}$ mit $\mathbf{Z}_t$ und erhalte $h_t : \mathbf{x} \mapsto \{-1, 1\}$

Output:

kombinierte Hypothese

$$f(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x})$$

Abbildung 2.2: Der Bagging-Algorithmus

Mit (2.4) findet man

$$\tilde{f}(\mathbf{x}) = \begin{cases} 1 & \text{wenn } \mathrm{card}\{t \mid h_t(\mathbf{x}) = 1\} \geq \lfloor T/2 \rfloor \\ -1 & \text{sonst} \end{cases} .$$

Bei der Auswertung der kombinierten Hypothese findet also eine (ungewichtete) Abstimmung zwischen den T Hypothesen statt. Ist die Zahl der Hypothesen, die für die Klasse mit Label 1 entscheiden, größer als die Zahl derer, die für Label -1 entscheiden, so wird Label 1 gewählt (und umgekehrt).

2.2 AdaBoost

Der *AdaBoost*-Algorithmus ist eine Weiterentwicklung von *Boosting*. Boosting hat seinen Ursprung in der *Spieltheorie* [23, 8], in der *PAC-Lern-Theorie* [7, 21, 28] und im *Online Learning* (Freund et al. [21]). AdaBoost wurde von Freund und Schapire ursprünglich für die Klassifikation entwickelt und basiert auf der Kombination von Hypothesen, die auf gezielt veränderten Trainingsmengen trainiert wurden. Jedes Beispiel $(\mathbf{x}^{(i)}, y^{(i)})$ erhält ein Gewicht p_i. Je höher das Gewicht ist, desto wichtiger ist das Beispiel beim Training der

nächsten Hypothese.
Die Gewichte hängen von den vorherigen Hypothesen ab (das bedeutet, daß die Basishypothesen sequentiell trainiert werden müssen). Das Gewicht p_i wird größer, wenn die letzte Hypothese h das Muster $\mathbf{z}^{(i)} = (\mathbf{x}^{(i)}, y^{(i)})$ falsch klassifiziert hat, d.h. wenn $\tilde{f}(\mathbf{x}^{(i)}) \neq y^{(i)}$ ist. Die Erhöhung der Gewichte p_i geschieht durch Multiplikation mit einem Faktor β_t. Dieser Faktor wird aus dem mit $\mathbf{p} = [p_1, \ldots, p_l]^\top$ gewichteten Klassifikationsfehler ϵ_t der letzten Hypothese bestimmt:

$$\epsilon_t = \sum_{i=1}^{l} p_i^t I(h_t(\mathbf{x}^{(i)}) \neq y^{(i)}) \tag{2.5}$$

$$\beta_t = \frac{\epsilon_t}{1-\epsilon_t} \quad . \tag{2.6}$$

Gilt $\epsilon_t < \frac{1}{2} - \gamma$ ($\gamma > 0$), so spricht man von einem γ-*Weak-PAC-Learner* [21, 28]. Für AdaBoost (und für Arcing) benötigt man einen Algorithmus $\mathcal{L}$, der für jede gewichtete Trainingsmenge eine Hypothese h_t erzeugt, für die $\epsilon_t < \frac{1}{2}$ gilt[2]. Frühere Versionen von Boosting forderten sogar, daß in jedem Schritt der Trainingsfehler ϵ_t kleiner als $\frac{1}{2} - \gamma$ ist.

Definition 2.1 (Gewichtete Minimierung) *Für eine gegebene Wahrscheinlichkeitswichtung* $\mathbf{p}$ *für die Trainingsmenge* $\mathbf{Z}$ *finde die Hypothese* $h \in \mathcal{H}$*, die*

$$\epsilon(h) = \sum_{i=1}^{l} p_i I(h(\mathbf{x}^{(i)}) \neq y^{(i)})$$

minimiert.

In der Praxis findet man oft nicht die Hypothese, die ϵ minimiert, sondern nur eine gute Näherung. Das hat meist keine negativen Auswirkungen.
Außerdem kann es sein, daß es keine Basishypothese in $\mathcal{H}$ gibt, für die $\epsilon_t < \frac{1}{2}$ gilt[3], dann kommt der AdaBoost-Algorithmus zum Stoppen. In Abbildung 2.3 ist der vollständige AdaBoost-Algorithmus beschrieben. Es gibt zwei Methoden, wie man die Gewichte $\mathbf{p}$ beim Lernen der Basishypothesen (Schritt 1b.-2a.) berücksichtigen kann:

- durch Erzeugung eines Bootstrap-Replikats, wobei die Gewichte $\mathbf{p}$ bei der zufälligen Auswahl die Wahrscheinlichkeit bestimmen, daß ein Beispiel gewählt wird, und
- durch direkte Benutzung der Gewichte $\mathbf{p}$ im Lernalgorithmus $\mathcal{L}$ und Veränderung der Fehlerfunktion, wie es beispielsweise für RBF-Netze im Anhang A.1 beschrieben ist.

NB: Fast alle Fehlerfunktionen haben die Form $\sum_{i=1}^{l} c(h(\mathbf{x}^{(i)}), y^{(i)})$. Man kann die Gewichte p_i direkt benutzen und erhält eine gewichtete Fehlerfunktion $\sum_{i=1}^{l} p_i c(h(\mathbf{x}^{(i)}), y^{(i)})$.

[2] $\epsilon_t > \frac{1}{2}$ wird nicht erlaubt, da dann das Gewicht der Hypothese negativ sein würde.
[3] Dann gäbe es nur Hypothesen mit $\epsilon_t = \frac{1}{2}$ und es wäre $b_t = 0$.

Algorithm AdaBoost

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$

Basislernalgorithmus $\mathcal{L}$

Anzahl der zu kombinierenden Hypothesen T

Initialize:

Gewichte für Trainingsbeispiele: $w_i^1 = 1/l$

Do for $t = 1 : T$,

1a. Setze $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{l} w_i^t}$

1b. Erzeuge Trainingsbeispiele $\mathbf{Z}_t$ aus $\mathbf{Z}$ entsprechend der Verteilung $\mathbf{p}^t$

2a. Starte $\mathcal{L}$ mit $\mathbf{Z}_t$ und erhalte $h_t : \mathbf{x} \mapsto \{-1, 1\}$

3a. Bestimme den Fehler von h_t : $\epsilon_t = \sum_{i=1}^{l} p_i^t I(h_t(\mathbf{x}^{(i)}) \neq y^{(i)})$.

3b. Setze $\beta_t = \epsilon_t/(1 - \epsilon_t)$.

3c. Berechne die neuen Gewichte für die Trainingsbeispiele

$$w_i^{t+1} = w_i^t \beta_t^{1-I(h_t(\mathbf{x}^{(i)}) \neq y^{(i)})}$$

Output:

Endhypothese

$$f(\mathbf{x}) = \sum_{t=1}^{T} c_t h_t(\mathbf{x}) \ ,$$

wobei

$$\alpha_t = \frac{\log(\beta_t)}{\sum_{t=1}^{T} \log(\beta_t)} \ .$$

Abbildung 2.3: Der AdaBoost-Algorithmus

Satz 2.2 (Freund et al. [53]) *Angenommen, der Lernalgorithmus $\mathcal{L}$ erzeugt beim Aufruf durch AdaBoost Hypothesen, deren gewichtete Klassifikationsfehler $\epsilon_1, \ldots, \epsilon_T$ seien. Dann gilt für jedes $\theta \in [-1, 1]$:*

$$\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] = \frac{1}{l} \sum_{i=1}^{l} I(y^{(i)} f(\mathbf{x}^{(i)}) \leq \theta) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1 - \epsilon_t)^{1+\theta}} \ .$$

Satz 2.2 besagt, daß die Fehlerwahrscheinlichkeit für Beispiele $\mathbf{z} = (\mathbf{x}, y) \in \mathbf{Z}$ exponentiell mit der Anzahl der Iterationen gegen Null strebt (falls $\epsilon_t < \frac{1}{2} - \gamma$). Hat man also einen

γ-Weak-PAC-Learner und eine Trainingsmenge der Größe l, so gilt:

$$\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}\left[yf(\mathbf{x}) \leq 0\right] \leq 2^T \left(\frac{1}{2}-\gamma\right)^{T/2} \left(\frac{1}{2}+\gamma\right)^{T/2} = \left(1-4\gamma^2\right)^{T/2} \ .$$

Da aus $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[yf(\mathbf{x}) \leq 0] < 1/l$ folgt, daß $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[yf(\mathbf{x}) \leq 0] = 0$, ist für

$$T > 2\frac{\ln(1/l)}{\ln(1-4\gamma^2)}$$

der Trainingsfehler $E_{\text{emp}} = \mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}\left[yf(\mathbf{x}) \leq 0\right] = 0$.

Beispiel 2.3 *Angenommen, wir hätten $l = 100$ und $\gamma = 0.25$; dann würde der Trainingsfehler nach höchstens* 33 *Iterationen Null sein.*

In der Sprache der PAC-Theorie bedeutet das, daß AdaBoost durch Kombination eines Ensembles von Weak-PAC-Learner einen *Strong-PAC-Learner* erzeugt.
Interessant ist auch die empirisch festgestellte Tatsache, daß der Generalisierungsfehler E_{gen} auch nach Erreichen von $E_{\text{emp}} = 0$ weiter sinkt. Dieses Phänomen wurde in Freund et al. [53] im Zusammenhang mit *Margin-Verteilungen* untersucht (siehe Abschnitt 2.5).

2.3 Arcing

Relativ kurz nach dem Erscheinen des AdaBoost-Algorithmus stellte Breiman [7] einen *Arcing-Algorithmus* vor (**a**daptive **r**esampling and **c**ombin**ing**). Zunächst veränderte er AdaBoost und erhielt den Algorithmus *Arc-X4* [7]. Dann definierte er eine Klasse von Algorithmen, die er unter ,,Arcing-Algorithmen" zusammenfaßte [8]. Interessanterweise ist AdaBoost eine spezielle Form eines Arcing-Algorith- mus und Satz 2.2 gilt (in leicht veränderter Form) für alle Arcing-Algorithmen.

2.3.1 Nichtnormalisiertes Arcing

Der nichtnormalisierte Arcing-Algorithmus hat zwei Parameter. Der erste ist eine Funktion $s : \mathbb{R} \to [0, \infty)$, an die die folgenden Bedingungen gestellt werden:

$$\lim_{x\to-\infty} s(x) = 0 \quad \text{und} \quad \lim_{x\to\infty} s(x) = \infty \tag{2.7}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}s(x) > 0 \quad \text{und} \quad \frac{\mathrm{d}^2}{\mathrm{d}x^2}s(x) > 0 \quad \text{für alle} \quad x \in \mathbb{R} \ . \tag{2.8}$$

Der zweite Parameter ist ϕ mit der Bedingung $\epsilon_t < \phi < 1$ $(t = 1, \ldots, T)$, wobei ϵ_t wieder der gewichtete Klassifikationsfehler der Einzelhypothese ist. Wie klein ϕ gewählt werden kann, hängt also von der Komplexität der Basishypothesen ab. Zur Vereinfachung definiert man:

$$f_t^u(\mathbf{x}^{(i)}) := \sum_{r=1}^{t} b_r h_r(\mathbf{x}^{(i)}) \tag{2.9}$$

als die *nichtnormalisierte Gesamthypothese* und

$$f_t(\mathbf{x}^{(i)}) := \frac{1}{|\mathbf{b}|} \sum_{r=1}^{t} b_r h_r(\mathbf{x}^{(i)}) \tag{2.10}$$

als die *normalisierte Gesamthypothese* bis zur Iteration t ist. Außerdem sei[4]

$$\begin{aligned} \mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b}) &:= \frac{|\mathbf{b}| - y^{(i)} f_t^u(\mathbf{x}^{(i)})}{2} \\ &= \frac{|\mathbf{b}|}{2}(1 - y^{(i)} f_t(\mathbf{x}^{(i)})) \ , \end{aligned} \tag{2.11}$$

wobei $\mathbf{b}$ der Hypothesengewichtsvektor und $|\mathbf{b}| = \sum_{t=1}^{T} b_r$ ist. Leicht zu sehen ist, daß $\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b})/|\mathbf{b}| \in [0, 1]$.
Der Algorithmus arbeitet ähnlich wie AdaBoost (Abbildung 2.3). Es ändern sich die Zeilen 3a. bis 3c zu:

3a. Bestimme $\beta_t \in (0, 1)$ durch Minimierung der Funktion

$$G(\beta_t) = \sum_{i=1}^{l} s\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b}) - \phi|\mathbf{b}|\right) \ , \tag{2.12}$$

wobei $\mathbf{b} = [\log(1/\beta_1), \ldots, \log(1/\beta_t)]^\top$ ist.

3b. Berechne die neuen Gewichte[5]

$$w_i^{t+1} = s'\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b}) - \phi|\mathbf{b}|\right) \ . \tag{2.13}$$

Satz 2.4 (Breiman [8]) *Seien $\mathbf{b}^t$ die Gewichte des nichtnormalisierten Arcing-Algorithmus, und sei $\mathbf{c}^t = \mathbf{b}^t/|\mathbf{b}^t|$, dann gilt*

$$\lim_{t\to\infty} \max_{i=1,\ldots,l} \mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{c}^t) \leq \phi \ .$$

Es wird damit gesagt, daß der Fehler[6] für jedes Beispiel kleiner als ϕ wird. Überraschend ist die

Proposition 2.5 (Breiman [8]) *AdaBoost ist ein nichtnormalisierter Arcing-Algorithmus mit $s(x) = \exp(x)$ und $\phi = \frac{1}{2}$.*

Beweis: Mit $s(x) = s'(x) = \exp(x)$ und (2.9) erhält man aus (2.13):

$$\begin{aligned} s'\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b}) - \phi|\mathbf{b}|\right) &= \exp\left\{-\phi|\mathbf{b}| + \frac{1}{2}\left(|\mathbf{b}| - y^{(i)} f_u^t(\mathbf{x}^{(i)})\right)\right\} \\ &= e^{-\phi|\mathbf{b}|}\pi_i^t \end{aligned}$$

[4] Die Funktion err wurde auch in Breiman [6] benutzt, um einige Eigenschaften von Arcing-Algorithmen zu erklären und wurde dort *Edge* genannt.

[6] Als Fehlermaß wird die negative Stabilität (vgl. Abschnitt 2.5) eines Beispiels angesehen.

wobei

$$\begin{aligned}\pi_i^t &= \prod_{r=1}^{t} \exp\left\{\frac{1}{2}\log\frac{1}{\beta_r}\left(1 - y^{(i)}h_r(\mathbf{x}^{(i)})\right)\right\} \\ &= \prod_{r=1}^{t}(1/\beta_r)^{I(h_r(\mathbf{x}^{(i)})\neq y^{(i)})} \\ &= \pi_i^{t-1}(1/\beta_t)^{I(h_t(\mathbf{x}^{(i)})\neq y^{(i)})} \\ &= \pi_i^{t-1}(1/\beta_t)\beta_t^{1-I(h_t(\mathbf{x}^{(i)})\neq y^{(i)})} \ . \end{aligned} \tag{2.14}$$

Mit $p_i^t = \frac{\pi_i^t}{\sum_{j=1}^{l}\pi_j^t}$ kürzt sich $1/\beta_t$ und man erhält die in Abschnitt 2.2 beschriebenen Updateregeln für AdaBoost.
Die Minimierung von G in Zeile 3a. kann in diesem Fall auch analytisch erfolgen und man erhält (siehe Anhang A.2.1)

$$\beta_t = \frac{(1-\phi)\epsilon_t}{\phi(1-\epsilon_t)} \ . \tag{2.15}$$

Für $\phi = 1/2$ ist das die Berechnung von β_t bei AdaBoost. ■

2.3.2 Normalisiertes Arcing

Das Normalisierte Arcing hat nur einen Parameter, die Funktion s, an die ebenfalls die Bedingungen (2.7) und (2.8) gestellt werden.
Der Algorithmus arbeitet ähnlich wie AdaBoost (Abbildung 2.3). Es ändern sich die Zeilen 3a. bis 3c:

3a. Setze $b_t = 1$, d.h. nach T Iterationen gilt $c_t = 1/T$.

3b. Berechne die neuen Gewichte

$$w_i^{t+1} = s'\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b})\right) \ .$$

3c. Breche ab, wenn $\epsilon_t \geq \sum_{i=1}^{l} p_i^t I(\tilde{f}_t(\mathbf{x}^{(i)}) \neq y^{(i)})$ ist.

Satz 2.6 (Breiman [8]) *Angenommen, beim normalisierten Arcing werden die Hypothesen wie in Definition 2.1 bestimmt. Dann gilt für $T \to \infty$ oder für den Fall des Abbruchs (in Zeile 3c.), daß die gefundenen Hypothesen $\{h_t\}_{t=1}^{T}$ das Funktional*

$$G = \sum_{i=1}^{l} s\left(\mathrm{err}_T(\mathbf{z}^{(i)}, \mathbf{b})\right)$$

global bezüglich

$$\widetilde{\mathcal{C}} = \bigcup_{T=1}^{\infty}\left\{f : x \mapsto \frac{1}{T}\sum_{t=1}^{T} h_t(x) \mid h_t \in \mathcal{H}^{\mathcal{L}}\right\}$$

minimieren, d.h. $f = \mathrm{argmin}_{g\in\widetilde{\mathcal{C}}} G$.

2.4 Bias und Varianz

Wann und warum liefern Ensemble-Lernmethoden gute Ergebnisse? Diese Frage wurde vielfach im Zusammenhang von Bias-Varianz-Zerlegungen des Fehlers untersucht [32, 24, 7, 37]. Für die Regression hat man einige Antworten darauf gefunden.

2.4.1 Regression

Bei der Regression werden Basishypothesen der Form $h : \mathbf{x} \mapsto \mathbb{R}$ benutzt (damit ist $f(\mathbf{x}) \in \mathbb{R}$). Krogh et al. [37] (vgl. auch Breiman [5] und Sollich et al. [62]) bewiesen einige Eigenschaften für Ensemble-Lernmethoden zur Funktionsapproximation. Es wurde u.a. gezeigt, daß für ein Ensemble von Funktionsapproximatoren, die gleiche Generalisierungsfehler haben, sich dieser im Ensemble nur verkleinern kann.
Den Ausführungen in [37] folgend sei $\mathcal{D}$ eine *Wahrscheinlichkeitsverteilung* über $\mathbb{R}^a \times \mathbb{R}$ (a ist die Dimension des Eingaberaums) und $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$ eine Menge von l Trainingsbeispielen, die bzgl. $\mathcal{D}$ erzeugt wurde. Außerdem sei $f^*(\mathbf{x}^{(i)}) = y^{(i)}$ die zu approximierende reelwertige Funktion.
Das Ensemble besteht aus T Basishypothesen $h : \mathbf{x} \mapsto \mathbb{R}$. Die Ausgabe des Ensembles berechnet sich durch

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \ . \tag{2.16}$$

Der *Grad der Abweichung* der Basishypothese $h_t(\mathbf{x})$ von der Ausgabe des Ensembles $f(\mathbf{x})$ für ein Trainingsbeispiel $\mathbf{x}$ wird *Ambiguity* genannt und ist durch $a^t(\mathbf{x}) = (h_t(\mathbf{x}) - f(\mathbf{x}))^2$ definiert. Die *Ensemble-Ambiguity* läßt sich dann als

$$\overline{a}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t a^t(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t (h_t(\mathbf{x}) - f(\mathbf{x}))^2 \tag{2.17}$$

definieren, was der gewichteten Varianz der Ausgaben der Hypothesen im Ensemble entspricht.
Der *quadratische Fehler* der Hypothese h_t und der Gesamthypothese f ist

$$\begin{aligned} \epsilon^t(\mathbf{x}) &= (f^*(\mathbf{x}) - h_t(\mathbf{x}))^2 \\ e(\mathbf{x}) &= (f^*(\mathbf{x}) - f(\mathbf{x}))^2 \ . \end{aligned}$$

Mit (2.17) erhält man

$$\overline{a}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \epsilon^t - e(\mathbf{x}) \ ,$$

wobei $\sum_{t=1}^{T} \alpha_t = 1$ benutzt wurde. Setzt man $\overline{\epsilon}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \epsilon^t(\mathbf{x})$, erhält man

$$e(\mathbf{x}) = \overline{\epsilon}(\mathbf{x}) - \overline{a}(\mathbf{x}) \ . \tag{2.18}$$

Mittelt, d.h. integriert man nun ϵ^t, a^t und e über der Verteilung $\mathcal{D}$ im Eingaberaum, erhält man

$$E^t = \int \epsilon^t(\mathbf{x}) \ \mathrm{d}\mathcal{D}(\mathbf{x}) \ , \quad A^t = \int a^t(\mathbf{x}) \ \mathrm{d}\mathcal{D}(\mathbf{x}) \ , \quad E = \int e(\mathbf{x}) \ \mathrm{d}\mathcal{D}(\mathbf{x}) \ ,$$

wobei E der Generalisierungsfehler von f ist. A^t und E^t sind Ambiguity und Generalisierungsfehler der Einzelhypothesen. Mit (2.18) erhält man

$$E = \overline{E} - \overline{A} \ , \tag{2.19}$$

wobei $\overline{E} = \sum_{t=1}^{T} \alpha_t E^t$ der gewichtete Mittelwert der Generalisierungsfehler und $\overline{A} = \sum_{t=1}^{T} \alpha_t A^t$ das gewichtete Mittel der Ambiguity der Einzelhypothesen ist.
Gleichung (2.19) gibt Auskunft darüber, wie das Ensemble beschaffen sein muß, damit es möglichst geringe Generalisierungsfehler liefert. Da $\overline{A}$ nichtnegativ ist, wird der Generalisierungsfehler der Gesamthypothese immer kleiner als der gewichtete Durchschnitt der Generalisierungsfehler der Einzelhypothesen sein. Für Bagging ist damit gezeigt, daß

$$E \leq \frac{1}{T} \sum_{t=1}^{T} E^t \ .$$

Man kann aus (2.19) auch erkennen, daß E dann am kleinsten wird (vorrausgesetzt $\overline{E} = \text{const.}$), wenn die Ambiguity am größten ist. Es ist für den Lernprozeß wichtig, daß die Basishypothesen möglichst unterschiedlich sind[7].
Mit (2.19) wird ein Zusammenhang zwischen *Bias*, *Varianz* und Generalisierungsfehler hergestellt: Wenn einerseits das Ensemble einen großen Bias hat, so ist die Ambiguity klein und die Basishypothesen werden nahezu identisch sein (auch außerhalb der Trainingsdaten). Der Generalisierungsfehler E wird sich nicht deutlich von $\overline{E}$ unterscheiden. Ist andererseits die Ambiguity groß, so wird E deutlich kleiner als $\overline{E}$ sein. Dabei ist aber zu bedenken, daß $\overline{E}$ bei Ensembles mit hoher Ambiguity möglicherweise größer ist, insbesondere dann, wenn die Erhöhung der Ambiguity durch Erhöhung der Komplexität der Basishypothesen geschehen ist. Dieser Zusammenhang ist zum *Bias-Varianz-Dilemma* (Bienenstock et al. [26]) äquivalent.
Es ist anzunehmen, daß sich eine hohe Ambiguity auf den Trainingsbeispielen auch auf dem übrigen Eingaberaum fortsetzt. In Heskes [31] wurde durch Veränderung der Gewichte $\{\alpha_t\}_{t=1}^{T}$ die Ambiguity auf den Trainingsbeispielen maximiert. Mit dieser Methode wurden deutlich bessere Ergebnisse als mit Bagging erzielt.
Zum Beispiel in Perrone [49] findet man eine additive Zerlegung des Generalisierungsfehlers in Bias und Varianz:

$$E = E^* + \text{Bias}(f) + \text{Varianz}(f) \ , \tag{2.20}$$

Wobei E^* der Generalisierungsfehler des besten Klassifikators (*Bayes-Klassifikator*) ist. Außerdem wurde gezeigt, daß sich die Varianz beim Bagging durch die Mittelung verringert und $\text{Varianz}(f) \to 0$ für $t \to \infty$, wobei $\text{Bias}(f) = \text{const}$.

2.4.2 Klassifikation

Die Definition von Bias und Varianz für den Fall der Klassifikation ist leider nicht so einfach. Die Zahl der Versuche (d.h. die Zahl der Definitionen von Bias und Varianz) den Generalisierungsfehler additiv zu zerlegen, ist beträchtlich (siehe z.B. Breiman [7], Heskes [32], Dietterich et al. [15], Kohavi et al. [35] und Tibshirani [63]). Friedman [24] zeigt sogar, daß die Zerlegung eher multiplikativ zu erfolgen hat.

[7] Würde man gleiche Hypothesen kombinieren, würde sich nichts an E ändern.

Eine Definition

Breiman [7] folgend kann man Bias und Varianz für die Klassifikation mit zwei Klassen wie folgt definieren: Sei $\mathcal{D}$ eine Verteilung über $X \times \{\pm 1\}$ und $\mathbf{Z} = \langle (\mathbf{x}^{(1)}, y^{(1)}), \ldots, \mathbf{x}^{(l)}, y^{(l)}) \rangle$ eine Trainingsmenge der Größe l, die bezüglich $\mathcal{D}$ gewählt wurde. Außerdem sei $\mathcal{L}(\cdot, \mathbf{Z})$ die Hypothese, die vom Lernalgorithmus $\mathcal{L}$ aus den Trainingsbeispielen $\mathbf{Z}$ erzeugt wird. Dann ist der Klassifikationsfehler auf ungesehenen Daten

$$\mathrm{PE}(\mathcal{L}(\cdot, T)) = \mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}[\mathcal{L}(\mathbf{x}, T) \neq y]$$

und der Erwartungswert über alle Trainingsmengen dafür sei $\mathrm{PE}(\mathcal{L}) = \mathrm{E}_{\mathbf{Z}}\mathrm{PE}(\mathcal{L}(\cdot, T))$. Die kleinste Fehlerrate wird durch den *Bayes-Klassifikator*[8] $\mathcal{B}$ erreicht [2]:

$$\mathcal{B}(\mathbf{x}) = \operatorname*{argmax}_{j\in\{\pm 1\}} P(j|\mathbf{x}) \ .$$

wobei $P(y|\mathbf{x}) = \mathbf{P}_{(\hat{\mathbf{x}},\hat{y})\sim\mathcal{D}}[y = \hat{y}, \mathbf{x} = \hat{\mathbf{x}}]$ ist. Die Fehlklassifikationsrate dieses Klassifikators ist

$$\mathrm{PE}(\mathcal{B}) = 1 - \int \operatorname*{argmax}_{j\in\{\pm 1\}} P(j|\mathbf{x})\mathrm{d}P(\mathbf{x}) \ .$$

Außer dem sei

$$Q(j|\mathbf{x}) = \mathrm{E}_{\mathbf{Z}}(\mathcal{L}(\mathbf{x}, \mathbf{Z})) \ .$$

Der durch die Kombination unendlich vieler Hypothesen, die mit unabhängigen Trainingsmengen trainiert worden sind, entstandene – aggregierte – Klassifikator[9] ist [7]:

$$\mathcal{L}_A(\mathbf{x}) = \operatorname*{argmax}_{j\in\{\pm 1\}} Q(j|\mathbf{x}) \ .$$

Man teilt nun den Eingaberaum in zwei disjunkte Teile [7]:
In ein *Unbiased-Set* U:

$$U = \{\mathbf{x} \in X | \mathcal{L}_A(\mathbf{x}) = \mathcal{B}(\mathbf{x})\}$$

und ein *Biased-Set* B:

$$B = \{\mathbf{x} \in X | \mathcal{L}_A(\mathbf{x}) \neq \mathcal{B}(\mathbf{x})\} \ .$$

Die Idee dabei ist, daß auf dem Unbiased-Set das Voting gut arbeitet und auf dem Biased-Set nicht.

Definition 2.7 (Breiman [7]) *Der Bias eines Klassifikators $\mathcal{L}$ ist*

$$\mathrm{Bias}(\mathcal{L}) = \mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}(\mathcal{B}(\mathbf{x}) = y, \mathbf{x} \in B) - \mathrm{E}_{\mathbf{Z}}\mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}(\mathcal{L}(\mathbf{x}, \mathbf{Z}) = y, \mathbf{x} \in B)$$

und seine Varianz ist

$$\mathrm{Var}(\mathcal{L}) = \mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}(\mathcal{B}(\mathbf{x}) = y, \mathbf{x} \in U) - \mathrm{E}_{\mathbf{Z}}\mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}(\mathcal{L}(\mathbf{x}, \mathbf{Z}) = y, \mathbf{x} \in U) \ .$$

[8] Hier ist der Klassifikator gemeint, der aus der genauesten Kenntnis von $\mathcal{D}$ konstruiert werden kann und die kleinstmöglichen Klassifikationsfehler erreicht.

[9] Hier ist der Klassifikator gemeint, der durch (ungewichtetes) Voting von unendlich vielen Hypothesen entsteht (analog zum Bagging in der Asymptotik).

Das führt zu einer *additiven Zerlegung* des Klassifikationsfehlers für ungesehene Beispiele in den Bayes-Fehler, den Bias und die Varianz des Klassifikators:

$$\mathrm{PE}(\mathcal{L}) = \mathrm{PE}(\mathcal{B}) + \mathrm{Bias}(\mathcal{L}) + \mathrm{Var}(\mathcal{L}) \ . \tag{2.21}$$

Definition 2.7 hat – im Gegensatz zu anderen Definitionen – die folgenden (guten) Eigenschaften hat [7]:

1. Bias und Varianz sind immer nicht-negativ.
2. Die Varianz des aggregierten Klassifikators ist Null.
3. Der Bias von $\mathcal{B}$ ist Null.

Diese Bedeutung von Bias und Varianz mag irreführend sein, wenn man an die übliche Bedeutung von Bias und Varianz (in der Regression [26]) denkt. Das wichtige an dieser Zerlegung ist jedoch, daß die Varianz so definiert ist, daß sie durch Aggregation vieler Klassifikatoren (wie z.B. bei Bagging) eliminiert werden kann. Der Bias des Ensembles kann natürlich größer sein, als der des Einzelklassifikators[10].

2.5 Margin-Verteilungen

In Schapire et al. [53] versuchte man die Effektivität von Ensemble-Methoden anhand von *Margin-Verteilungen* zu erklären.
Untersucht man die Ausgaben einer Gesamthypothese f, so kann man die *Stabilitäten* (den *Margin*) von Trainingsbeispielen messen:

$$\mathrm{stab}_t(\mathbf{z}^{(i)}) := f_t(\mathbf{x}^{(i)})y^{(i)} \ , \tag{2.22}$$

wobei $\mathrm{stab}(\mathbf{z}^{(i)}) := \mathrm{stab}_T(\mathbf{z}^{(i)})$. Ein Trainingsbeispiel $\mathbf{z}^{(i)}$ wird von f richtig klassifiziert, genau dann wenn $\mathrm{stab}_T(\mathbf{z}^{(i)}) \geq 0$.
Als einen *Margin-Verteilungs-Graphen* [53] bezeichnet man einen Graphen, der die kumulative Verteilung $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}(\mathrm{stab}(\mathbf{z}) \leq \theta)$ über $\theta \in [-1, 1]$ darstellt. Der Graph zeigt für eine Stabilität θ den relativen Anteil der Beispiele, die eine kleinere Stabilität als θ haben.

Beispiel 2.8 *In Abbildung 2.4 sind typische Margin-Verteilungen für Bagging und AdaBoost dargestellt. Diese wurden für den Datensatz #1 (siehe A.3.1) mit 13 Zentren und jeweils 100 Iterationen erzeugt.*

Ziel der Betrachtungen in Schapire et al. [53] war es herauszustellen, warum AdaBoost im Vergleich zu Bagging bessere Generalisierungsfehler $\mathbf{P}_{\mathbf{z}\sim\mathcal{D}}[\mathrm{stab}(\mathbf{z}) \leq 0]$ liefert. Dies wurde mit der Eigenschaft der Stabilitätsmaximierung (Satz 2.2) des AdaBoost-Algorithmus erklärt.

Satz 2.9 (Schapire et al. [53]) *Sei $\mathcal{D}$ eine Verteilung über $X \times \{\pm 1\}$ und $\mathbf{Z}$ eine Trainingsmenge der Größe l, die bezüglich $\mathcal{D}$ gewählt wurde. Weiterhin sei $\mathcal{H}$ der Raum der Basishypothesen mit einer VC-Dimension d [65]. Angenommen es sei $\delta > 0$ und $l \geq d \geq 1$.*

[10] So bleibt also das Bias-Varianz-Dilemma erhalten.

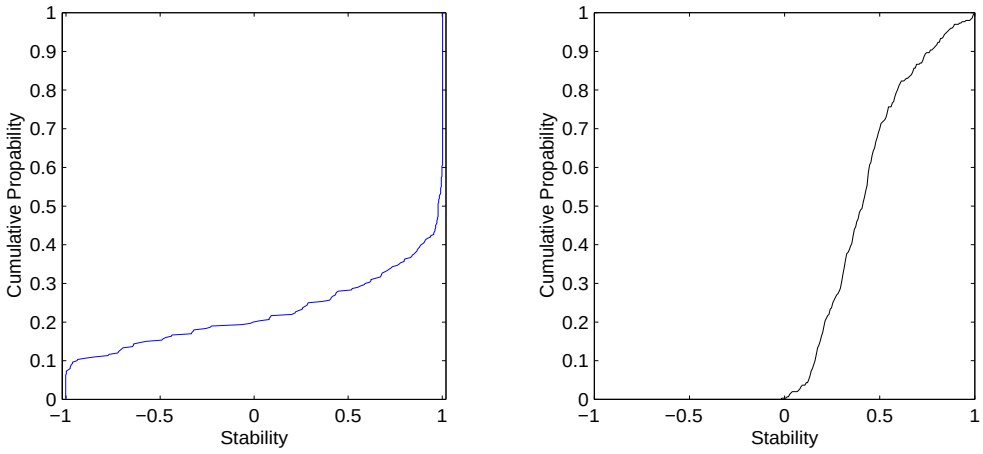


Abbildung 2.4: Typische Margin-Verteilungs-Graphen für Bagging (links) und AdaBoost (rechts). AdaBoost weist deutlich weniger Beispiele mit negativer Stabilität auf (d.h. der Trainingsfehler ist kleiner).

Dann gilt mit einer Wahrscheinlichkeit von mindestens $1-\delta$ *über der zufälligen Wahl der Trainingsmenge* $\mathbf{Z}$*, daß jede gewichtete Funktion* $f \in \mathcal{C}$ *(Konvexe Hülle von* $\mathcal{H}$*; vgl. (2.2)) die folgende Ungleichung für alle* $\theta > 0$ *erfüllt:*

$$\mathbf{P}_{\mathbf{z}\sim\mathcal{D}}[\mathrm{stab}(\mathbf{z}) \leq 0] \leq \mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[\mathrm{stab}(\mathbf{z}) \leq \theta] + \mathcal{O}\left(\frac{1}{\sqrt{l}}\left(\frac{d\log^2(l/d)}{\theta^2} + \log(1/\delta)\right)\right) . \quad (2.23)$$

Benutzt man Satz 2.2, sieht man, daß der erste Term der rechten Seite von (2.23) bei AdaBoost (im Gegensatz zu Bagging) minimiert wird. AdaBoost ist insbesondere dazu geeignet, $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[\mathrm{stab}(\mathbf{z}) \leq \theta]$ klein zu machen. Da der zweite Term der rechten Seite für Bagging und AdaBoost gleich ist, wird die rechte Seite für AdaBoost kleiner.
Unklar ist noch, wie eng die in Satz 2.9 angebene obere Schranke für den Generalisierungsfehler ist und ob ein Minimum der rechten Seite auch in der Nähe des Minimums der linken Seite liegt. An fiktiven Beispielen (z.B. $d = 2$ und $l = 10000$) kann man die rechte Seite der Schranke ausrechnen (angenommen es sei $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[\mathrm{stab}(\mathbf{z}) < \theta] = 0$, $\theta = 1$) und mit den Ergebnissen für existierende Datensätze vergleichen. Es zeigt sich, daß (zumindest für einige Beispiele) diese Schranke sehr weit von der Wirklichkeit entfernt sind.
Auch wenn dieser Satz eher eine qualitative Aussage über den Generalisierungsfehler macht, ist klar, daß man bei fest gewählter Komplexität und bei größeren Stabilitäten der Mehrzahl der Trainingsbeispiele zu kleineren Generalisierungsfehlern kommt [64].

Kapitel 3

Warum AdaBoost oft gute Ergebnisse liefert

3.1 Experimentelle Ergebnisse

3.1.1 Vergleiche

In diesem Abschnitt sollen einige Erfolge von AdaBoost insbesondere im Bereich der Handschrifterkennung herausgestellt werden. AdaBoost liefert im Vergleich zu anderen Ensemble-Lernmethoden und zu Einzelklassifikatoren sehr gute Ergebnisse [4, 6, 8, 16, 17, 20, 39, 53].

NIST-Datensatz

In LeCun et al. [39] wurden die besten zu dieser Zeit verfügbaren Klassifikationsalgorithmen zur Handschrifterkennung auf dem NIST-Datensatz[1] verglichen. Es wurden u.a. LeNet 1, LeNet 4, geboostetes LeNet 4 und LeNet 5 verglichen. LeNet 1, 4 und 5 sind auf die Handschrifterkennung hochspezialisierte neuronale Netze (mit mehreren konvolutiven Schichten), die zur Erkennung von Postleitzahlen benutzt werden [4, 39]. Tabelle 3.1 zeigt

[1] Dieser Datensatz ist 784-dimensional, enthält insgesamt etwa 120.000 Beispiele, und wurde vom *National Intitute for Standardization* für Benchmarks zur Verfügung gestellt.

Architektur	Testfehler
LeNet 1	1.7%
LeNet 4	1.1%
LeNet 5	0.9%
SVM mit polynom. Kern	1.4%
SVM mit virt. SV	0.8%
boosted LeNet 4	**0.7%**

Tabelle 3.1: Vergleich zwischen verschiedener Arten von Klassifikatoren. Boosted LeNet 4 ist bis heute ungeschlagen.

die Ergebnisse des Vergleichs. Die Ergebnisse des *Support-Vektor-Algorithmus* [3, 14, 64] wurden hinzugefügt [12, 55]. Das geboostete[2] LeNet 4 erreicht die besten Ergebnisse und ist bis heute ungeschlagen. Erstaunlich ist, daß dabei nur drei Boosting-Iterationen benutzt wurden und trotzdem solch ein gutes Ergebnis erzielt wurde.

UCI-Repository

In Quinlan [51] wurden die Ergebnisse von AdaBoost auf *Descision Trees* (Version C4.5) mit den Ergebnissen der Einzelklassifikatoren verglichen. Abbildung 3.1 zeigt eine Zusammenfassung der Ergebnisse dieses Vergleiches. In fast allen Fällen (23 von 27) brachte die Verwendung des AdaBoost-Algorithmus eine Verkleinerung des Generalisierungsfehlers. Es wurden keine Angaben zur Varianz dieser Bestimmung gemacht.
Für den Vergleich wurde das UCI-Repository von Benchmark-Datensätzen benutzt, das ein breites Spektrum von Lernproblemen (sehr hoch- bzw. niedrig-dimensionale und viele bzw. wenige Beispiele) bietet. Eine genaue Beschreibung der Datensätze findet man im Internet unter http://www.ics.uci.edu/mlearn/MLSummary.html.

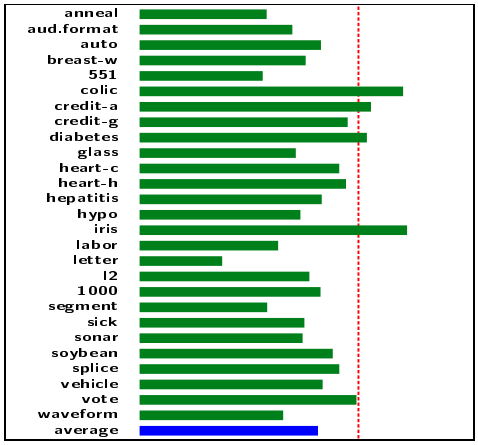


Abbildung 3.1: Vergleich Einzelklassifikator – AdaBoost: Zusammenfassung der Ergebnisse von Quinlan [51]. Die rote Linie steht für den Generalisierungsfehler der Einzelhypothese und die grünen Balken zeigen den Fehler von AdaBoost relativ zur Einzelhypothese.

Datensätze #1 und #2

Im Anhang A.3.1 sind zwei künstliche Datensätze beschrieben. Auch für diese Datensätze zeigt AdaBoost auf RBF-Netzen (siehe A.1) relativ gute Leistungen im Vergleich zu Einzelklassifikatoren (RBF-Netz) und zu Bagging auf RBF-Netzen. Die experimentellen Ergebnisse findet man in Tabelle 3.2.
Bei Datensatz #1 ist AdaBoost in jedem Fall besser als die Basishypothese und in fast jedem Fall besser als Bagging (außer bei sehr hoher Komplexität der Basishypothese). Bei Datensatz #2 ist AdaBoost in fast jedem Fall besser als die Einzelhypothese, aber in fast jedem Fall schlechter als Bagging. Ein möglicher Grund dafür ist, daß für Datensatz #2 nur eine sehr kleine Komplexität notwendig ist (Optimum bei vier RBF-Zentren) und daß es durch die Kombination der Hypothesen zu Overfitting kommt. Weitere Gründe für die im Vergleich zu Bagging schlechten Ergebnisse werden in Kapitel 4 gesucht.
Eine weitere positive Eigenschaft fällt auf: Während bei der Einzelhypothese und bei Bagging der Generalisierungsfehler relativ stark von der Komplexität der Basishypothesen abhängt (bei Bagging nur, wenn die Basishypothese zu schwach ist), für AdaBoost findet man diese Abhängigkeit in diesen Experimenten nicht.

[2] Damals wurde ein Vorläufer von AdaBoost benutzt: es wurde keine Gewichtung der Trainingsbeispiele verwendet, sondern auf eine relativ ähnliche Art Trainingsmengen erzeugt [39].

	Datensatz #1							Datensatz #2					
#Zentren	4	7	10	13	20	30	40	2	3	4	5	7	10
RBF-Netz	16.3	9.2	6.5	6.2	5.7	5.9	6.3	32.7	11.5	10.3	10.9	11.4	11.4
Bagging	9.9	7.0	6.2	5.8	5.7	5.5	5.7	32.7	10.5	9.8	10.1	10.3	10.4
AdaBoost	5.6	5.4	5.4	5.4	5.7	6.0	6.0	32.7	10.8	10.8	10.8	11.1	11.0

Tabelle 3.2: Vergleich Einzelhypothese – Bagging – AdaBoost: Für die Datensätze #1 und #2 (ohne Rauschen) wurden die Generalisierungsfehler (in %) für die drei Fälle bestimmt. Bei Bagging und AdaBoost wurden jeweils 100 Hypothesen kombiniert und über verschiedene Trainingsmengen gemittelt (Standardabweichung etwa 0.2 – 0.8%).

3.1.2 Entscheidungslinien werden glatter

Bei der Analyse des AdaBoost-Algorithmus stellt sich sehr oft die Frage, warum der Testfehler trotz steigender Zahl der kombinierten Klassifikatoren (und damit wachsender Komplexität) sinkt. Eine Antwort erhält man, wenn man sich an einem einfachen Beispiel die Entwicklung der Entscheidungslinien verdeutlicht. In Abbildung 3.2 findet man solche Entscheidungslinien nach verschiedenen Anzahlen von Iterationen für Datensatz #1[3]. Dargestellt sind die Entscheidungsgrenzen des (bis dahin) kombinierten Klassifikators und des jeweiligen neuen Einzelnetzes. Man erkennt die Entwicklung und auch, daß die Entscheidungslinien in den ersten Schritten viele ,,Ecken und Kanten" enthalten. Außerdem ist die Approximation[4] der Bayes-Entscheidungslinie dargestellt (oben links). Da diese sehr glatt (global und lokal) ist, bringt jede globale[5] Nichtglattheit eine Erhöhung des Generalisierungsfehlers.

Die Klassengrenzen werden mit fortschreitendem Prozess glatter. Man kann auch sehen, daß bei einer hohen Anzahl der Iterationen die Entscheidungslinie nicht mehr so glatt ist. Dieses Phänomen wird näher in Kapitel 4 betrachtet.

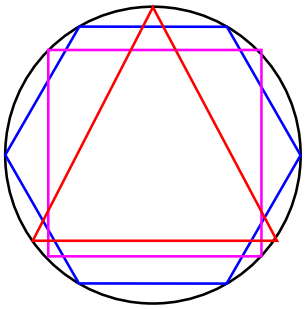


Abbildung 3.3: Viele Ecken ergeben einen Kreis

Da die wahren Entscheidunglinien (in vielen Fällen) ebenfalls keine Ecken oder Kanten haben, sinkt der Generalisierungsfehler mit global glatter werdenden Grenzen. An Beispielen kann man sich verdeutlichen, daß sie auch bei Benutzung von Basishypothesen, die selbst keine glatten oder runden Entscheidungslinien erzeugen, glatter werden: In Abbildung 3.3 findet man ein Beispiel dafür, daß die Kombination von vielen Ecken durchaus eine Rundung ergeben kann.

In den ersten Iterationen des Lernprozesses werden Ecken erzeugt, die dann durch Hin-

[3] Es wurden bewußt RBF-Netze mit nur vier Zentren benutzt, so daß man die Bildung der Entscheidungslinie verfolgen kann.

[4] Es wurde ein RBF-Netz mit 100 Zentren auf einem Datensatz mit 10^4 Beispielen trainiert. Die hohe Komplexität des Netzes sichert die Fähigkeit zur Approximation der Entscheidungslinie und die hohe Anzahl der Beispiele verhindert Overfitting.

[5] Mit global nichtglatt sind insbesondere die Ecken und Kanten wie z.B. nach der 4. Iteration gemeint. Lokale Nichtglattheit bedeutet, daß die Entscheidungslinie lokal ,,hochfrequent" um einen Mittelwert schwankt (große Ableitungen). Lokale Nichtglattheit beeinflußt den Generalisierungsfehler in der Regel kaum.

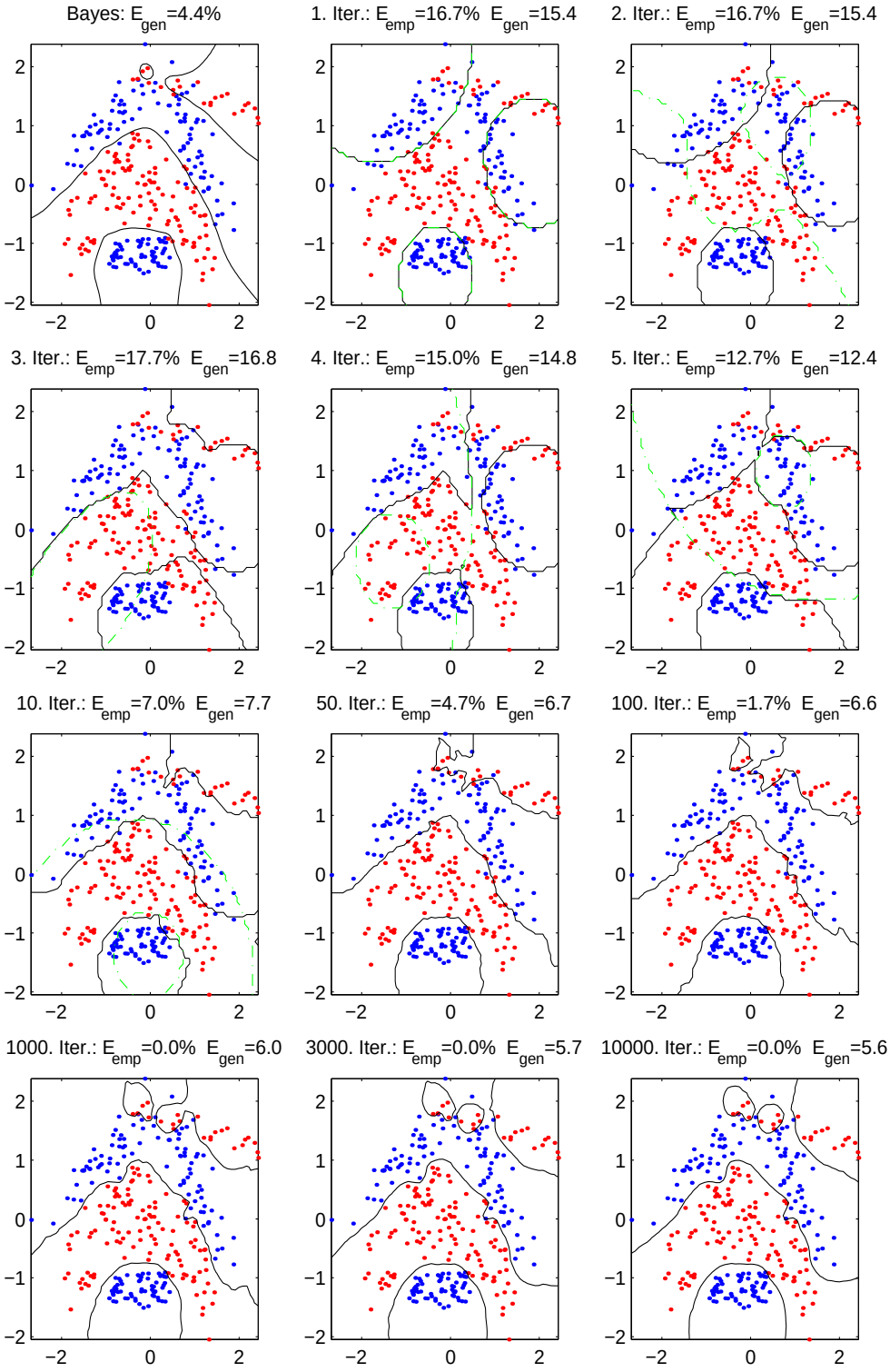


Abbildung 3.2: Entstehende Entscheidungslinien (schwarz) auf Datensatz #1 (rote und blaue Punkte) für AdaBoost mit RBF-Netzen (vier Zentren) nach $1, \ldots, 5, 10, 50, 100, 1000, 3000$ und 10000 Iterationen. Bei den ersten fünf wird auch die Entscheidungslinie des Einzelklassifikators (grün) dargestellt.

zufügen weiterer Hypothesen wieder geglättet werden; unabhängig davon, wie glatt die Basishypothesen sind. Nicht verwunderlich ist der Erfolg von Decision Trees in Verbindung mit Bagging und AdaBoost [5, 16, 22].

3.1.3 Ensemble-Lernmethoden verringern die Varianz

Eine andere Antwort auf die im letzten Abschnitt gestellte Frage erhält man, wenn man die Varianz der Klassifikatoren betrachtet. In Breiman [7] wurde gezeigt, daß Bagging asymptotisch die Varianz (vgl. Abschnitt 2.4) der Klassifikation stark reduziert. In Friedman [24] wird gezeigt, daß Varianz-Verkleinerung (auf den richtig klassifizierten Beispielen) zur Verringerung des Generalisierungsfehlers führt.

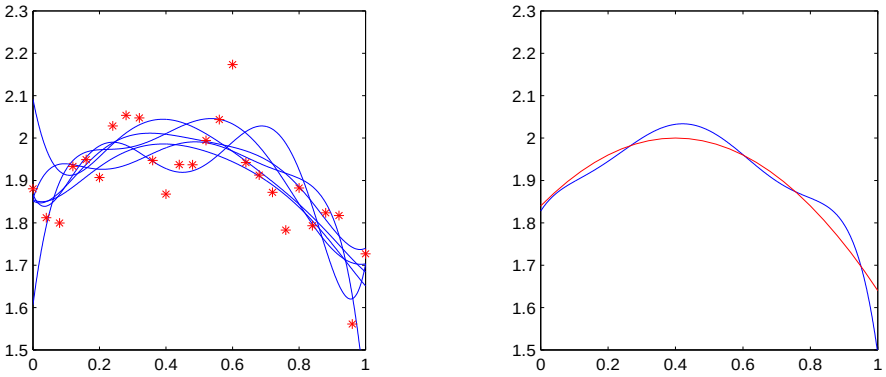


Abbildung 3.4: Analogie zu den Entscheidungslinien bei Ensemble-Lernmethoden: fünf Polynome 6. Grades werden an Daten, die mit einem Polynom 2. Grades erzeugt werden (+ Gaußsches Rauschen), gefittet (links). Dabei werden jeweils unterschiedliche Gewichtungen der Trainingsmenge benutzt. Man kann eine hohe Varianz der Approximation erkennen. Rechts ist die wahre Kurve und die sich aus der Mittelung der fünf Einzelkurven ergebende Kurve abgebildet. Die Abweichung von der wahren Kurve wird deutlich verringert.

Beispiel 3.1 *In Abbildung 3.4 findet man ein Beispiel dafür, daß Bagging Overfitting vermeiden und die Varianz der Entscheidungslinie verringern kann. Das wird anhand von Daten, die mit einem Polynom 2. Grades erzeugt (plus Gaußsches Rauschen) und dann mit Polynomen 6. Grades appoximiert werden, gezeigt. Beim Bagging wird in jeder Iteration ein Boostrap-Replikat erzeugt und so das Trainingsset variiert. In diesem Beispiel wurde ein gewichteter Trainingsdatensatz benutzt, dessen Gewichtungen sich in jeder Iteration zufällig verändern. Mittelt man die Ausgaben von 5 Polynomen 6. Grades, erhält man eine deutlich bessere Approximation zur wahren Funktion*[6]*. Intuitiv versteht man damit, daß Bagging Overfitting vermeiden bzw. reduzieren kann.*

Wie sich in den nächsten Abschnitten noch herausstellen wird, optimiert AdaBoost eine spezielle Fehlerfunktion (Abschnitt 3.2). Die Minimierung dieser Fehlerfunktion ist oft

[6]Diese Analogiebetrachtungen zur Regression durch Polynome dient lediglich dem besseren Verständnis

Algorithmus	Anzahl der Zentren 4	7	10	13	20
RBF-Netz	17.8 ± 6.6	8.8 ± 1.9	6.7 ± 0.4	6.1 ± 0.5	5.8 ± 0.5
Bagging	10.4 ± 1.1	7.2 ± 0.5	6.2 ± 0.1	5.7 ± 0.3	5.7 ± 0.4
AdaBoost	6.0 ± 0.4	5.4 ± 0.5	5.4 ± 0.3	5.5 ± 0.3	5.7 ± 0.2

Tabelle 3.3: Für die drei Algorithmen RBF-Netz, Bagging und AdaBoost wurde die Standardabweichung des Generalisierungsfehlers bestimmt (alle Angaben in %). Dazu wurden für Datensatz #1 jeweils 100 Hypothesen erzeugt und die Standardabweichung der Fehler berechnet. Die Einzelhypothese zeigt deutlich höhere Varianzen als die Ensemble-Lernmethoden. Bagging und AdaBoost haben ähnliche Varianzen.

nicht optimal (vgl. Propostion 3.6). Aber gerade durch die Nichtoptimalität der Minimierung entstehen Hypothesen, die in „ähnlicher Form" schon im Ensemble existieren[7]. Dadurch wird auch bei AdaBoost, wie beim Bagging, die Varianz verkleinert (insbesondere die Varianz, die auf den Basislernalgorithmus zurückzuführen ist). In Breiman [7] wird vermutet, daß allein schon die Kombination von Hypothesen, die auf unterschiedlichen Trainingsmengen trainiert wurden, eine Verringerung der Varianz impliziert.
Das wird beispielhaft für Datensatz #1 in Tabelle 3.3 gezeigt[8].
Daß AdaBoost einen Bagging-Prozeß beinhaltet, ist sehr schwierig zu zeigen, da AdaBoost im Lernprozeß deutlich komplizierter als Bagging ist. In Abbildung 3.5 wird dennoch versucht deutlich zu machen, daß Bagging ein Bestandteil von AdaBoost ist. Für diese Abbildung wurde ein Ensemble mit AdaBoost auf Datensatz #1 ($\sigma = 0$) trainiert. Dann wurden die Ausgaben der einzelnen Hypothesen (hier 200) mit einander verglichen. Jeder Punkt in Abbildung 3.5 stellt die κ-Korrelation [43, 1, 13] zwischen den Ausgaben zweier Hypothesen dar. Man erkennt sich wiederholende Muster und auch, daß sich die Korrelation nur langsam mit der Anzahl der Iterationen verändert. Es werden also Hypothesen erzeugt, die auf vielen Beispielen ähnlich klassifizieren, wobei sich aber die Entscheidungslinien unterscheiden (Analogie zur Ambiguity). Es kommt dadurch zu einer Mittelung (Analogie zum Begriff der Ambiguity in Abschnitt 2.4.1).

3.2 AdaBoost gleicht einem Gradienten-Verfahren

Ausgehend vom Arcing-Algorithmus und von Proposition 2.5 werden nun *AdaBoost-ähnliche Algorithmen* untersucht.

Definition 3.2 *Ein AdaBoost-ähnlicher Algorithmus ist ein nichtnormalisierter Arcing-Algorithmus mit* $s(x) = \exp(x)$ *und* $\phi \in (0, 1)$*, wobei für die Trainingsfehler* ϵ_t

$$\phi > \epsilon_t + \Delta, \text{ für alle } t = 1, \ldots, \infty \ . \tag{3.1}$$

[7] Auch beim Bagging sind die Einzelhypothesen relativ gleich, da sie aus ähnlichen Trainingssets entstanden sind

[8] Vergleicht man die Ergebnisse von Tabelle 3.2 und 3.3, so stellt man leichte Unterschiede fest. Diese sind darauf zurückzuführen, daß in 3.2 über mehrere Datensätze und in 3.3 über mehrere Klassifikatoren auf dem gleichen Datensatz gemittelt wurde.

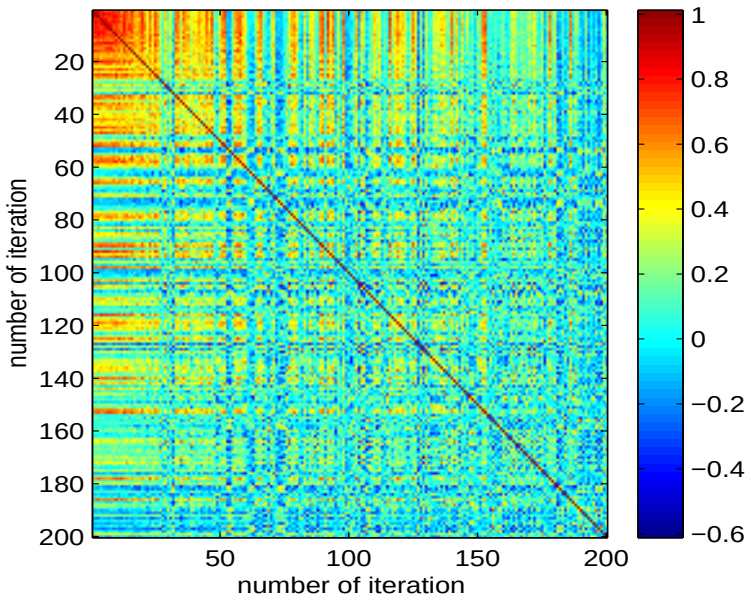


Abbildung 3.5: κ-Korrelation zwischen den Ausgaben der im AdaBoost-Prozeß entstandenen Einzelhypothesen

gelten soll. $\Delta > 0$ *sei beliebig, aber fest für alle* t*.*

NB: Für $\phi = \frac{1}{2}$ hat man wieder einen Δ-Weak-Learner. Man könnte hier also von einem Δ-ϕ-Weak-Learner sprechen.

Wie bereits in Gleichung (2.9) und Gleichung (2.10) seien

$$\begin{aligned} f_t^u(\mathbf{x}) &= \sum_{r=1}^{t} b_r^t h_r^t(\mathbf{x}) \\ f_t(\mathbf{x}) &= \frac{1}{|\mathbf{b}^t|} \sum_{r=1}^{t} b_r^t h_r^t(\mathbf{x}) \ , \end{aligned}$$

wobei

$$\begin{aligned} \mathbf{b}^t &= [\log(1/\beta_1), \ldots, \log(1/\beta_t), 0, \ldots, 0]^\top \qquad \text{und} \\ \mathbf{h}^t &= \{h_r\}_{r=1}^T, h_r = \mathbf{0}, \text{ für } r > t \end{aligned}$$

ist. Mit (2.11) und (2.22) findet man

$$\mathrm{err}_t(\mathbf{z}, \mathbf{b}^t) = \frac{1}{2}|\mathbf{b}^t|(1 - \mathrm{stab}_t(\mathbf{z})) \ . \tag{3.2}$$

Folgendes Lemma ist eine Erweiterung des Satzes 2.4:

Lemma 3.3 *AdaBoost-ähnliche Algorithmen minimieren asymptotisch das Funktional*

$$\widetilde{G}(\mathbf{b}^t, \mathbf{h}^t) = \sum_{i=1}^{l} \exp\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{b}^t) - \phi|\mathbf{b}^t|\right) \ . \tag{3.3}$$

Es gilt:

$$\lim_{t\to\infty} \widetilde{G}(\mathbf{b}^t, \mathbf{h}^t) \ = \ 0 \tag{3.4}$$

$$\textit{und} \qquad \lim_{t\to\infty} \min_{i=1,\ldots,l} \mathrm{stab}_t(\mathbf{z}^{(i)}) \ \geq \ 1 - 2\phi \ . \tag{3.5}$$

Beweis: Mit (2.15) und (3.1) findet man $\lim_{t\to\infty} |\mathbf{b}^t| = \infty$. Als untere Abschätzung für $|\mathbf{b}|$ hat man sogar $|\mathbf{b}^t| \geq O\left(t \log \frac{\phi(1+\Delta-\phi)}{(1-\phi)(\phi-\Delta)}\right)$. Mit Satz 2.4 gilt

$$\begin{aligned} \widetilde{G}(\mathbf{b}^t, \mathbf{h}^t) \ &= \ \sum_{i=1}^{l} \exp\left(\left(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{c}^t) - \phi\right)|\mathbf{b}^t|\right) \\ &\leq \ l \exp\left(\left(\max_{i=1,\ldots,l}(\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{c}^t)) - \phi\right)|\mathbf{b}^t|\right) \end{aligned} \tag{3.6}$$

und außerdem $\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{c}^t) - \phi \leq 0$, für t groß genug[9] Dann folgt mit (3.6), $|\mathbf{b}^t| \to \infty$ und $\exp(-\infty) = 0$ die Behauptung (3.4), und mit Gleichung (3.2) folgt (3.5). ■

NB: Mit (3.2) und $\phi = \frac{1}{2}$ findet man für AdaBoost die besonders einfache Form

$$\widetilde{G}(\mathbf{b}^t, \mathbf{h}^t) = \sum_{i=1}^{l} e^{-\frac{1}{2}|\mathbf{b}^t|\mathrm{stab}_t(\mathbf{z}^{(i)})} \ .$$

Die in Lemma 3.3 genannte Minimierung kann auf zwei Arten erfolgen: Zum einen kann man das Funktional durch Hinzufügen neuer Hypothesen beliebig verkleinern (da $|\mathbf{b}^t|$ damit steigt); zum anderen hat die Wahl der Hypothesen $\mathbf{h}$ und der Gewichte $\mathbf{c}$ Einfluß auf die *Konvergenzgeschwindigkeit*. Gleichung (3.3) läßt sich mit (2.11) umschreiben zu

$$\widetilde{G}(\mathbf{b}^t, \mathbf{h}^t) = \sum_{i=1}^{l} e^{-\phi|\mathbf{b}^t|} \exp\left(\sum_{r=1}^{T} b_r^t l_r(\mathbf{z}^{(i)})\right) \ , \tag{3.7}$$

wobei $l_r(\mathbf{z}^{(i)}) = I(y^{(i)} \neq h_r(\mathbf{x}^{(i)}))$ ist [8].
Angenommen, in der t-ten Iteration würde man fragen, für welche Trainingsbeispiele es besonders wichtig ist, von der nächsten Hypothese richtig klassifiziert zu werden, um $\widetilde{G}$ möglichst klein zu machen. Dazu berechnet man die Ableitungen bzgl. des Fehlers $l_t(\mathbf{z})$ der nächsten Hypothese. Mit (3.2) erhält man aus (3.7):

$$\widetilde{G}^i_{l_t} \equiv \frac{\partial \widetilde{G}(\mathbf{b}^{t-1}, \mathbf{h}^{t-1})}{\partial l_t(\mathbf{z}^{(i)})} = b_t \left(e^{1/2-\phi} e^{-\frac{1}{2}\mathrm{stab}_{t-1}(\mathbf{z}^{(i)})}\right)^{|\mathbf{b}^{t-1}|} \ . \tag{3.8}$$

[9] Eigentlich muß hier gezeigt werden, daß $\mathrm{err}_t(\mathbf{z}^{(i)}, \mathbf{c}^t) - \phi$ *echt kleiner* als Null ist. Mit den bisherigen Sätzen ist das nicht so einfach möglich. Mit Korollar 3.11 kann man dann auch die echte Relation zeigen.

Für Trainingsbeispiele $\mathbf{z}^{(i)}$, für die die Vergrößerung der Stabilität besonders wichtig ist, um $\widetilde{G}$ zu minimieren (für die es also besonders notwendig ist, von der nächsten Hypothese richtig klassifiziert zu werden), ist $\widetilde{G}^i_{l_t}$ groß – sonst relativ klein. Damit kann man $\widetilde{G}^i_{l_t}$ benutzen, um die Gewichtung der Beispiele für das Training der nächsten Hypothese zu bestimmen.

Lemma 3.4 *Bei AdaBoost-ähnlichen Algorithmen werden die Gewichte der Trainingsbeispiele* $\mathbf{p}^t$ *in der* t*-ten Iteration durch*

$$p_i^t = \frac{\frac{\partial \widetilde{G}(\mathbf{b}^{t-1},\mathbf{h}^{t-1})}{\partial l_t(\mathbf{z}^{(i)})}}{\sum_{j=1}^{l} \frac{\partial \widetilde{G}(\mathbf{b}^{t-1},\mathbf{h}^{t-1})}{\partial l_t(\mathbf{z}^{(j)})}} \tag{3.9}$$

bestimmt.

Beweis: Aus (3.9) mit (3.8) erhält man

$$p_i^t = \frac{\exp\left(-\frac{1}{2}\mathrm{stab}_{t-1}(\mathbf{z}^{(i)})|\mathbf{b}^{t-1}|\right)}{\sum_{j=1}^{l}\exp\left(-\frac{1}{2}\mathrm{stab}_{t-1}(\mathbf{z}^{(j)})|\mathbf{b}^{t-1}|\right)} \; . \tag{3.10}$$

Verwendet man außerdem (2.14) im Beweis von Proposition 2.5 und $p_i^t = \frac{\pi_i^t}{\sum_{j=1}^{l}\pi_j^t}$, ergibt sich die (in den nichtnormalisierten Arcing-Algorithmen benutzte) Formel für $\mathbf{p}^t$ (vgl. Zeile 1a. in Abbildung 2.3 und Gleichung (2.13)). ■

Die Normierung $|\mathbf{p}| = 1$ in (3.9) entkoppelt die Optimierung von h_t von der Optimierung von b_t (vgl. (3.8)). Es kann zunächst die Hypothese und dannach das optimale Gewicht mit Gleichung (2.15) durch

$$b_t = -\log\beta_t = \log\frac{1-\epsilon_t}{\epsilon_t} - \log\frac{1-\phi}{\phi} \tag{3.11}$$

bestimmt werden (Herleitung im Anhang A.2.1). Wenn die Kenntnis von $\mathbf{p}^t$ ausreicht, könnte in jedem Schritt eine *optimale Kombination* (h_t, b_t) bzgl. $G(\mathbf{b}^{t-1},\mathbf{h}^{t-1})$ gefunden werden.
Damit kommt die Frage auf, wie die Hypothese h_t aus der Gewichtung $\mathbf{p}^t$ bestimmt werden muß, damit sie das Funktional bestmöglich minimiert. Breiman [8] schlägt die gewichtete Minimierung (Definition 2.1) vor. Wie man sich an Beispielen verdeutlichen kann, ist die Minimierung des gewichteten Klassifikationsfehlers ϵ leider nicht gleichbedeutend mit der Minimierung von $\widetilde{G}$.
Eine andere Möglichkeit $\widetilde{G}$ zu optimieren, ist die Parameter von h_t direkt mit einem nichtlinearen Optimierungsverfahren zu bestimmen. Das würde die bestmögliche Hypothese (oder wegen lokaler Minima nur eine gute Approximation davon) ergeben. Es wäre der prinzipiell richtige Weg zur Minimierung von $\widetilde{G}$, der aber relativ aufwendig ist[10].
Unklar ist noch, wie man nur durch Kenntnis von $\mathbf{p}^t$, d.h. unabhängig von $|\mathbf{b}^t|$ die optimale Hypothese h_t bzgl. $\widetilde{G}(\mathbf{b}^{t-1},\mathbf{h}^{t-1})$ finden kann. Eine Möglichkeit ist die gewichtete Minimierung, die nichtoptimale Ergebnisse liefern kann. Eine andere Möglichkeit ist, die Beispiele in sortierter (absteigender) Reihenfolge von der Basishypothese richtig klassifizieren zu lassen. Je mehr Beispiele (in dieser Ordnung) korrekt sind, desto besser. Formal ausgedrückt heißt das:

[10] Eine Kombination von linearer und nichtlinearer Optimierung wäre notwendig.

Definition 3.5 (Sortierte Minimierung) *Gegeben sei eine Wahrscheinlichkeitswichtung* $\mathbf{p}$ *für die Trainingsmenge* $\mathbf{Z}$. *Sortiere* $\mathbf{p}$ *durch die Permutation* S, *so daß* $p_{S(i)} \leq p_{S(j)}$ *für* $1 \leq i < j \leq l$. *Die sortierte Minimierung findet eine Hypothese* $h \in \mathcal{H}$, *die*

$$J^h = \max \left\{ j \mid I(h(\mathbf{x}^{(S(j))}) \neq y^{(S(j))}) \right\} \tag{3.12}$$

minimiert.

Es gibt leider auch für diese Art der Minimierung Situationen, in denen die Ergebnisse nicht optimal sind. Man kann aber beweisen, daß eine Hypothese h^*, die ein kleineres $\epsilon(h^*)$ und ein größeres J^{h^*} erzeugt als eine andere Hypothese h, das Funktional $\widetilde{G}$ besser optimiert. Möglicherweise ist eine Kombination von sortierter und gewichteter Minimierung notwendig, um bessere Hypothesen zu finden. Wie diese Kombination genau arbeitet ist noch nicht klar.
Die folgende Anmerkung zeigt, daß AdaBoost-ähnliche Algorithmen einem Gradientenverfahren gleichen. Dabei werden die Hypothesen als Gradienten aufgefaßt. Damit ist klar, daß auch schlechte Approximationen der Hypothese h_t^* zur Minimierung von $\widetilde{G}$ führen. Die Konvergenz verlangsamt sich aber dadurch.

Anmerkung 3.6 *AdaBoost-ähnliche Algorithmen approximieren ein Gradientenabstiegs-Verfahren auf dem Raum* $\mathcal{C}$ *(Funktionenklasse der Konvex-Kombinationen von* $\mathcal{H}$*), um das Funktional* $\widetilde{G}$ *schrittweise zu minimieren. Dabei werden nur Startwerte* h_1 *und Gradienten* $h_2, \ldots, h_T$ *aus einem Teilraum* $\mathcal{C}^{\mathcal{L}} \subset \mathcal{C}$ *benutzt, der zu* $\mathcal{H}$ *(Klasse der Basishypothesen) isomorph ist.*

Begründung: Analog zu einem Gradientenverfahren in einem Vektorraum ist ein Gradientenverfahren auf dem Hypothesenraum $\mathcal{C}$ bzgl. $\widetilde{G}$ ein Verfahren, das iterativ zu einem Startwert (Hypothese $\widetilde{h}_1 \in \mathcal{C}$) *Gradienten-Hypothesen* $\widetilde{h}_2, \ldots, \widetilde{h}_T \in \mathcal{C}$ addiert, um $\widetilde{G}$ zu minimieren. Dabei ist jedes $\widetilde{h}_i$ $(i \geq 2)$ eine Richtung im Raum $\mathcal{C}$.
Um die Gradienten-Hypothesen zu bestimmen, müßte man den Gradient von $\widetilde{G}$ bzgl. $\mathcal{C}$ bestimmen. Das ist praktisch kaum möglich und auch nicht notwendig, da nur die Trainingsfehler der Hypothesen auf $\widetilde{G}$ eine Auswirkung haben. Deshalb sucht man Hypothesen, die fehlerhaft auf solchen Trainingsbeispielen sind, die $\widetilde{G}$ maximieren. Die Anforderung an eine ,,gute" Hypothese kann also durch die Wichtigkeit der Trainingsbeispiele formuliert werden. Es ist jedoch schwierig, eine optimale Hypothese $\widetilde{h} \in \mathcal{C}$ zu finden, die diesem Kriterium genügt. Deshalb schränkt man den Raum der möglichen Gradienten auf $\widetilde{h} \in \mathcal{C}^{\mathcal{L}}$ ein[11].
Man findet durch gewichtete Minimierung, sortierte Minimierung o.ä. die Hypothese h_t, die eine gute Approximation zur besten Hypothese in $\mathcal{C}^{\mathcal{L}}$ ist, und durch Gleichung (3.11) das optimale Gewicht für diese Hypothese b_t (die Schrittweite des Gradienten). Man erhält also die fast optimale Gradienten-Hypothese $\widetilde{h}_t \in \mathcal{C}^{\mathcal{L}}$ mit der optimalen Schrittlänge.
Auch ein Optimierungsverfahren, das keine optimalen Richtungen benutzt (oder sogar Zufalls-Richtungen) wird das Funktional minimieren (Modulo der lokalen Minima[12]), wenn die Schrittweiten optimal bestimmt werden [33, 42]. ■

[11] Das entspricht dann einem Gradienten-Abstieg, bei dem in einem Schritt immer parallel zu den Achsen optimiert wird (*Coordinate Descent*). Als Achsen werden hier die Basishypothesen angesehen.

[12] Es kann natürlich sein, daß $\widetilde{G}$ lokale Minima hat, nicht das beste Minimum erreicht wird. Aber das kann auch bei einem normalen Gradiente-Abstiegs-Verfahren passiern.

3.3 AdaBoost ist ein Annealing-Prozeß

Wie festgestellt wurde, arbeiten AdaBoost-ähnliche Algorithmen mit dem Fehlerfunktional $\widetilde{G}$ in (3.3). Eine wichtige Rolle dabei spielt die Größe von $|\mathbf{b}|$. Untersucht man die Bestimmung der Gewichte der Trainingsbeispiele in der t-ten Iteration, so wird die Abhängigkeit von $|\mathbf{b}|$ besonders klar. Gleichung (3.10) ergibt

$$p_i^t = \frac{\exp\left(-\frac{1}{2}\mathrm{stab}_{t-1}(\mathbf{z}^{(i)})\right)^{|\mathbf{b}^{t-1}|}}{\sum_{j=1}^{l}\exp\left(-\frac{1}{2}\mathrm{stab}_{t-1}(\mathbf{z}^{(j)})\right)^{|\mathbf{b}^{t-1}|}} \ . \tag{3.13}$$

Wenn $|\mathbf{b}| = 1$ ist, dann handelt es sich dabei um die *Softmax-Funktion*. Diese Funktion ist die ,,sanfte" Version der *Winner-Takes-All-Funktion* [2]. Wird $|\mathbf{b}|$ erhöht, so verschärft sich der Wettbewerb und für $|\mathbf{b}| \to \infty$ konvergiert (3.13) gegen die Winner-Takes-All-Funktion.
Erhöht man $|\mathbf{b}|$ schrittweise während des Boosting-Prozesses, so kann man von einem Annealing-Prozeß (Abkühlungsprozeß) [2, 34] sprechen. Annealing wird häufig bei nichtlinearen Optimierungen benutzt, um das Steckenbleiben in lokalen Minima zu verhindern. Analog zu einem physikalischem Prozeß [34] definiert man eine Temperatur ϑ, die die Sensitivität der Softmax-Funktion regelt:

$$\eta_i = \frac{\exp(-\zeta_i)^{1/\vartheta}}{\sum_{j=1}^{l}\exp(-\zeta_j)^{1/\vartheta}} \ .$$

Man kann also $|\mathbf{b}| =: 1/\vartheta$ als einen Annealing-Parameter auffassen.

Die Trainingsbeispiele mit den Stabilitäten $\mathrm{stab}_{t-1}(\mathbf{z}^{(i)})$ stehen im Wettbewerb untereinander. Große Stabilitäten erzeugen kleine Gewichte in $\mathbf{p}$ – und umgekehrt.
Der Wert von $|\mathbf{b}|$ steigt ungefähr linear mit der Anzahl der Iterationen. Das ist in Abbildung 3.6 exemplarisch dargestellt, und es ist auch wegen (2.15) und (3.1) (vgl. Lemma 3.3) plausibel, da das Minimale Gewicht, daß eine Hypothese bekommen kann, $\log\frac{1-\phi+\Delta}{\phi-\Delta} + \log\phi 1 - \phi$ (wegen $\epsilon \leq \phi - \Delta$) ist. Somit muß der Anstieg mindestens linear in der Anzahl der Iterationen sein. In Experimenten hat sich oft ein Mittelwert für ϵ_t eingestellt. Damit wird der Wettbewerb im Verlaufe des Boosting-Prozesses verschärft.

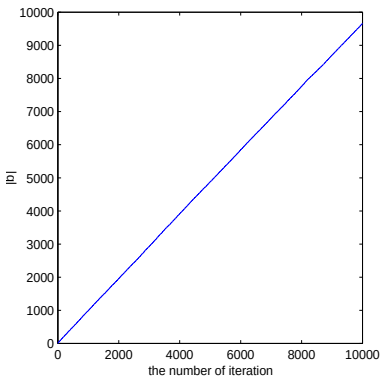


Abbildung 3.6: $|\mathbf{b}|$ steigt fast linear mit der Anzahl der Iterationen (30 RBF-Zentren auf Datensatz #1)

Asymptotisch werden nur solche Beispiele Gewichte verschieden von 0 bekommen, die die kleinste Stabilität haben. Neue Hypothesen h_t werden also insbesondere schwierige Muster richtig klassifizieren. Das führt zu einer Erhöhung der Stabilität dieser Beispiele und zur Verringerung der Stabilitäten der restlichen Beispiele [53].
Intuitiv wird damit auch klar, daß der Trainingsfehler zu Null werden muß: Trainingsbeispiele mit negativen Stabilitäten würden äußerst große Gewichte bekommen. Das ändert sich erst dann, wenn die restlichen Beispiele etwa gleiche Stabilitäten haben[13].

[13] Daß nicht alle Beispiele negative Stabilität haben können, ist klar.

Beispiel 3.7 *Angenommen zwei Beispiele hätten die Stabilitäten* $\text{stab}(\mathbf{z}^{(1)}) = 0.2$ *und* $\text{stab}(\mathbf{z}^{(2)}) = 0.3$ *und* $|\mathbf{b}| = 1000$. *Die beiden Beispiele haben dann sehr unterschiedliche Anteile am Fehler:* $\exp\left(-\frac{0.3\times 1000}{2}\right) = \exp(-150)$ *und* $\exp\left(-\frac{0.2\times 1000}{2}\right) = \exp(-100)$. *Das Gewicht des ersten Beispiels wird also etwa* 10^{20} *mal größer sein als des zweiten.*

Wenn die Stabilitäten der Trainingsbeispiele unabhängig voneinander optimierbar wären, dann müßte man einen Punkt erreichen, an dem alle Beispiele die gleiche Stabilität haben. Da es aber sehr häufig passiert, daß Beispiele schon richtig klassifiziert werden, wenn eine Menge von anderen Beispielen[14] richtig klassifiziert wird, trifft das nicht ohne Einschränkung zu.
Damit erhalten wir

Beobachtung 3.8 *Bei AdaBoost-ähnlichen Algorithmen findet man eine Teilmenge* **S** *der Trainingsbeispiele* **Z**, *die asymptotisch die gleiche Stabilität haben. Das sind die Beispiele mit der kleinsten Stabilität.*

Das wird durch die obige Diskussion gestützt und stimmt mit den experimentell gefundenen Ergebnissen überein. Diese Beispiele mit gleicher Stabilität liegen in der Regel nahe den Entscheidungslinien der Klassen. Betrachtet man die in Abschnitt 2.5 eingeführten Margin-Verteilungs-Graphen, so findet man im asymptotischen Fall einen sehr steilen Anstieg bei den kleineren Stabilitäten (vgl. Abbildung 3.7). Dieser Anstieg zeigt, daß relativ viele Beispiele (in dieser Abbildung etwa 20%) die gleiche Stabilität haben. Man könnte vermuten, daß man (asymptotisch) beim Training mit AdaBoost nur auf den $S-$*Beispielen* die gleichen Entscheidungslinien erhält, wie als wenn man auf dem gesamten Trainingsdatensatz trainiert (Analogie zu den Support-Vektoren).

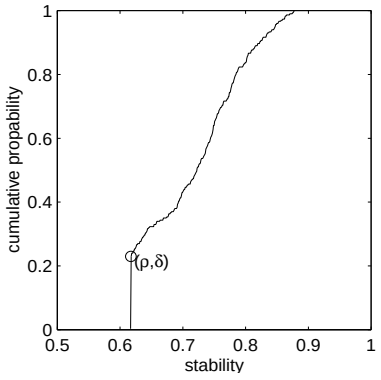


Abbildung 3.7: Ein Margin-Verteilungs-Graph in der Asymptotik (wie in 3.6)

Im Folgenden werden die Beispiele gleicher Stabilität nun **S**-Beispiele genannt und die Stabilität dieser Beispiele wird mit ρ bezeichnet, d.h.

$$\rho := \min_{i=1,\ldots,l} \lim_{t\to\infty} \text{stab}_t(\mathbf{z}^{(i)}) \ . \tag{3.14}$$

Der Anteil der **S**-Beispiele im Verhältnis zur Gesamtzahl sei $\delta := \frac{\#\mathbf{S}}{\#\mathbf{Z}}$.

In [23] und [8] wurde bereits ein Zusammenhang zwischen AdaBoost und *linearer Optimierung* festgestellt. Folgende Proposition drückt diesen Zusammenhang aus:

Anmerkung 3.9 *AdaBoost benutzt einen Annealing-Prozeß, um das Ungleichungssystem*

$$\text{stab}(\mathbf{z}^{(i)}) \geq \rho, \qquad \textit{für alle } i = 1,\ldots,l \tag{3.15}$$

zu erfüllen. Dabei wird versucht ρ *zu maximieren.*

[14] Es könnte eine Klasse aus einer Wolke von Beispielen bestehen: Punkte im Inneren der Wolke werden meist bereits richtig klassifiziert, wenn die Randpunkte der Wolke richtig klassifizert werden.

NB: Für die **S**-Beispiele werden die Ungleichungen asymptotisch zu Gleichungen.

Begründung: Mit Lemma 3.3 gilt asymptotisch

$$-\frac{\mathrm{stab}(\mathbf{z}^{(i)})}{2}+\frac{1}{2}-\phi \leq -\Theta, \qquad \Theta \geq 0, \quad \forall i=1,\ldots,l\ . \tag{3.16}$$

Vergleicht man (3.16) mit (3.3), so folgt: je größer Θ, desto kleiner $\widetilde{G}$ und umgekehrt (für eine feste Anzahl von Iterationen). Da $\widetilde{G}$ minimiert wird, kommt es approximativ[15] auch zur Maximierung von Θ. Mit (3.14) erhält man asymptotisch

$$\rho = 1 + 2\Theta - 2\phi\ .$$

Dewegen wird auch versucht ρ zu maximieren, wobei die Ungleichungen (3.15) erfüllt werden. ■

Vergleicht man die Margin-Verteilungen von AdaBoost und von einer *Support-Vektor-Maschine* (SVM), so stellt man starke Ähnlichkeiten fest: Auch bei der SVM gibt es Beispiele, die eine gleiche Stabilität haben (separabler Fall), die *Support-Vektoren* (SV) [3, 14, 64]. Experimentell kann man die Übereinstimmung von **S**-Beispielen mit den Support-

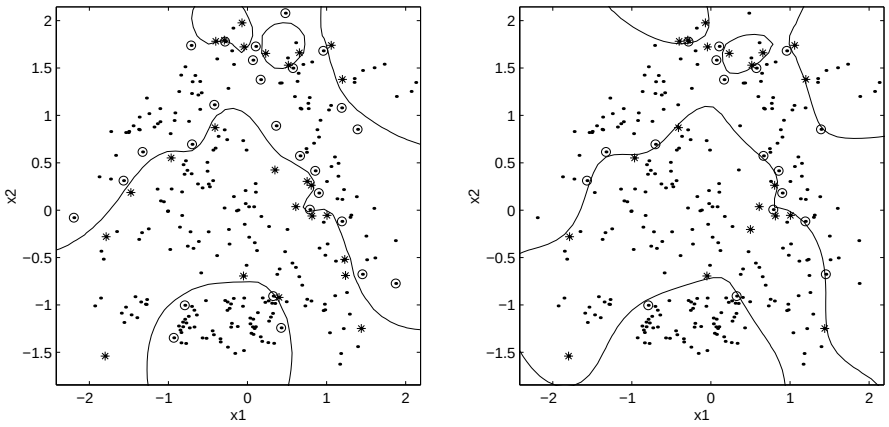


Abbildung 3.8: Entscheidungslinien für AdaBoost mit RBF-Netzen (links) und SVM mit RBF-Kernel (rechts) für den Datensatz #1 (wenig Rauschen). Es wurden RBF-Netze mit 13 Zentren und ein RBF-Kernel mit $\sigma^2 = 0.72$ (Durchschnitt der Radien der Zentren der RBF-Netze) benutzt. Die **S**-Beispiele und die Support-Vektoren sind mit $*$ (positive Beispiele) und $\circ$ (negative Beispiele) gekennzeichnet.

Vektoren bestimmen. In Abbildung 3.8 sind exemplarisch die **S**-Beispiele und die Support-Vektoren für den Datensatz #1 dargestellt. Fast alle Support-Vektoren sind auch **S**-Beispiele. In Tabelle 3.4 werden die Support-Vektoren mit den **S**-Beispielen für Datensatz

[15] Es wird nicht behauptet (und es wahrscheinlich ist auch nicht so), daß das Maximum für Θ gefunden wird. Das Maximum wird erreicht, wenn man lineare Optimierung benutzt und wahrscheinlich auch, wenn man unendlich langsam abkühlt.

RBF-Zentren	RBF(σ^2) 0.1	0.72	3.0	POLY(p) 3	5	7
7	44	59	33	25	59	75
13	46	61	37	26	58	71
20	48	62	37	27	58	71

RBF σ^2	POLY(p) 3	5	7
0.1	69	45	39
0.72	85	74	60
3.0	90	51	33

Tabelle 3.4: Experimentelle Bestimmung der Übereinstimmung der Support-Vektoren mit den **S**-Beispielen (links). Dazu im Vergleich die Übereinstimmung der Support-Vektoren für RBF- und POLY-Kernel (rechts). Ein Tabelleneintrag ist das Verhältnis (in %) wieviele Support-Vektoren auch **S**-Beispiele sind zur Gesamtzahl der Support-Vektoren. Insbesondere zwischen SVen aus einer POLY-SVM mit höheren Graden und den **S**-Beispielen findet man eine hohe Übereinstimmung. Außerdem bei SVM mit RBF-Kernel für σ =Mittelwert der Weiten der RBF-Zentren.

#1 verglichen. Dabei werden verschiedene Anzahlen der Zentren für die RBF-Netze benutzt und unterschiedliche Parameter für die SVM mit RBF und polynomiellen Kernel verwendet. Man findet findet i.d.R. eine große Übereinstimmung[16], insbesondere im Vergleich zu den Übereinstimmungen bei SVMs mit RBF-/POLY-Kernel.

NB: Die unterschiedlichen Tendenzen in Tabelle 3.8 sollten nicht überschätzt werden (insbesondere bei POLY-Kernel), da sie größtenteils aus der unterschiedlichen Anzahl der Support-Vektoren resultieren. Beispielsweise hat die SVM mit Polynomgrad drei etwa 180 SV und jene mit Grad sieben nur 40 SV. Beim AdaBoost-Algorithmus findet man etwa 20%, d.h. etwa 60 **S**-Beispiele. Damit ist klar, daß man hier bei höheren Graden höhere Übereinstimmungen findet. Außerdem hängt die Anzahl der Support-Vektoren von der Regularisierung ab, die hier willkürlich gewählt wurde.

Welche Beispiele Support-Vektoren werden, hängt von der Abbildung in den Feature-Raum und damit vom Kernel ab. Je nachdem wie die Datenwolken nichtlinear verformt werden, liegen andere Beispiele nahe der Entscheidungslinie und werden Support-Vektoren.
Bei AdaBoost hängt das von der Basishypothese ab: Je nachdem, welche Beispiele von der Basishypothese leicht richtig klassifiziert werden können, gibt es einmal diese und einmal jene **S**-Beispiele. Ist die Entscheidunglinie fast linear, so wird man wahrscheinlich mit Decision Trees weniger **S**-Beispiele erhalten, als z.B. mit RBF-Netze. Hat man sehr ,,runde" Daten, so wird dies möglicherweise umgekehrt sein.
Eine gute Übereinstimmung findet man zwischen den **S**-Beispielen von AdaBoost und den Support-Vektoren einer SVM mit RBF-Kernel, bei der die Weite des Kernels dem Durchschnitt der Weiten der RBF-Netze entspricht. Dies zeigt, daß die Daten auf eine ähnliche Art und Weise nichtlinear abgebildet werden, und daß sie mit etwa den selben ,,Maßen" interpretiert werden (siehe auch Schölkopf et al. [54] für eine Vergleich der von unterschiedlichen SVM erzeugten SV). Das führt zu gleichen Beispielen im Margin-Bereich.

[16] Was nicht unbedingt überraschend ist, da es sich immer um die Beispiele in der Nähe der Entscheidungslinien handelt.

3.4 Wie groß ist der Margin?

Die Stabilität der **S**-Beispiele wird durch den i.A. sehr komplexen und durch das Annealing gesteuerten Optimierungsprozeß bestimmt. Allgemein gilt für Annealing-Prozesse: Wenn die Abkühlungsgeschwindigkeit klein ist, so wird ein besseres Minimum gefunden [34], d.h. die Stabilitäten werden größer (vgl. (3.3)). Folgender Satz ist eine Verallgemeinerung von Satz 5 in [53] (vgl. Satz 2.2) für den Fall $\phi \neq \frac{1}{2}$:

Satz 3.10 *Angenommen, $\epsilon_1, \ldots, \epsilon_T$ seien die gewichteten Klassifikationsfehler der Hypothesen $h_1, \ldots, h_T$, die beim Ablauf eines AdaBoost-ähnlichen Algorithmus erzeugt werden und es sei $1 > \phi > \max_{t=1,\ldots,T} \epsilon_t$. Dann gilt für jedes $\theta \in [-1, 1]$:*

$$\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] \leq \left(\varphi^{\frac{1+\theta}{2}} + \varphi^{-\frac{1-\theta}{2}}\right)^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \ , \tag{3.17}$$

wobei $\varphi = \frac{\phi}{1-\phi}$ ist.

Der Beweis befindet sich im Anhang A.2.2.
NB: Für $\phi = \frac{1}{2}$ gilt $\left(\varphi^{\frac{1+\theta}{2}} + \varphi^{-\frac{1-\theta}{2}}\right) = 2$ und wir erhalten Satz 2.2.
Der Einfluß der Wahl von ϕ und der Trainingsfehler ϵ_t auf ρ wird asymptotisch durch den folgenden Satz ausgedrückt [47]:

Korollar 3.11 *AdaBoost-ähnliche Algorithmen erzeugen asymptotisch ($t \to \infty$) Margin-Verteilungen mit einer kleinsten Stabilität ρ, für die gilt*

$$\rho \geq \frac{\ln\left(\phi\epsilon^{-1}\right) + \ln\left((1-\phi)(1-\epsilon)^{-1}\right)}{\ln\left(\phi\epsilon^{-1}\right) - \ln\left((1-\phi)(1-\epsilon)^{-1}\right)} \ , \tag{3.18}$$

wobei $\epsilon = \max_t \epsilon_t$.

Beweis: Das Maximum von $\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}$ bzgl. ϵ_t ist für $\frac{1}{2}(1-\theta)$ erreicht. Wegen (3.5) und (3.1) gilt $\epsilon \leq \frac{1}{2}(1-\rho)$ und man kann für $\theta \leq \rho$ in Gleichung (3.17) ϵ_t durch ϵ ersetzen:

$$\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] \leq \left(\left(\varphi^{\frac{1+\theta}{2}} + \varphi^{-\frac{1-\theta}{2}}\right)\epsilon^{\frac{1-\theta}{2}}(1-\epsilon)^{\frac{1+\theta}{2}}\right)^T \ .$$

Wenn die Basis auf der rechten Seite kleiner 1 ist, dann gilt asymptotisch $\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] = 0$. Asymptotisch wird es also kein Beispiel geben, das eine kleinere Stabilität als θ hat. Für die größtmögliche Stabilität $\theta_{\max}$ gilt dann

$$\left(\varphi^{\frac{1+\theta_{\max}}{2}} + \varphi^{-\frac{1-\theta_{\max}}{2}}\right)\epsilon^{\frac{1-\theta_{\max}}{2}}(1-\epsilon)^{\frac{1+\theta_{\max}}{2}} = 1 \ .$$

Löst man diese Gleichung nach θ_{max} auf, erhält man

$$\theta_{\max} = \frac{\ln\left(\phi\epsilon^{-1}\right) + \ln\left((1-\phi)(1-\epsilon)^{-1}\right)}{\ln\left(\phi\epsilon^{-1}\right) - \ln\left((1-\phi)(1-\epsilon)^{-1}\right)} \ .$$

Da ρ größergleich $\theta_{\max}$ sein muß, erhalten wir die Behauptung. ■

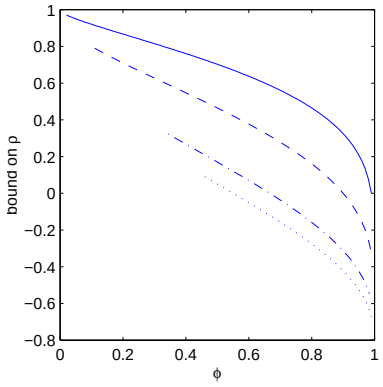


Abbildung 3.9: Abhängigkeit zwischen ϕ und der unteren Schranke für ρ für verschiedene $\epsilon = \frac{1}{100}$ (durchgezogen), $\frac{1}{10}$ (gestrichelt), $\frac{1}{3}$ (gestrichpunktet), $\frac{9}{20}$ (gepunktet).

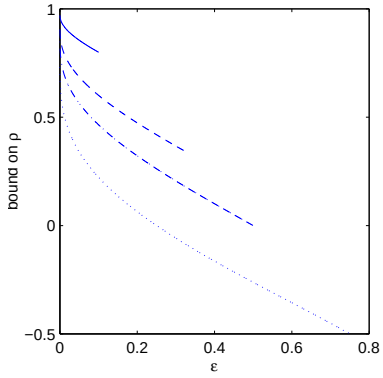


Abbildung 3.10: Abhängigkeit zwischen ϵ und der unteren Schranke für ρ für verschiedene $\phi = \frac{1}{10}$ (durchgezogen), $\frac{1}{3}$ (gestrichelt), $\frac{1}{2}$ (gestrichpunktet), $\frac{3}{4}$ (gepunktet).

Ein Vergleich von experimentell bestimmten ρ mit der unteren Schranke für ρ in Korollar 3.11 zeigt, daß die Abschätzung (3.18) noch nicht sehr gut ist. Deutlich bessere Abschätzungen liefert die direkte Anwendung von Satz 3.10. Die Tabellen 3.5 und 3.6 zeigen diesen Vergleich für verschiedene ϕ auf dem Datensatz #1. Es werden verschieden komplexe RBF-Netze (mit verschieden vielen Zentren) benutzt und ρ experimentell nach 10000 Iterationen bestimmt.

ϕ \ #Zentren	Experiment				Satz 3.10			
	7	13	20	30	7	13	20	30
0.3	-	-	0.444	0.663	-	-	0.591	0.701
0.5	0.122	0.248	0.418	0.617	0.119	0.247	0.400	0.616
0.7	0.086	0.211	0.373	0.557	0.241	0.304	0.410	0.5650

Tabelle 3.5: Vergleich von experimentell bestimmtem und dem sich aus Satz 3.10 ergebendem ρ für Datensatz #1 mit $\sigma^2 = 0$ für verschiedene ϕ und Anzahl der Zentren. Für $\phi = 0.3$ und 7 Zentren war der Lerner zu schwach. Bei 13 Zentren und $\phi = 0.3$ war nach 10.000 Iterationen noch nicht die Asymptotik erreicht und ρ ließ sich nicht bestimmen.

Das sich mit Satz 3.10 theoretisch ergebende ρ stimmt für $\phi = \frac{1}{2}$ auffallend genau mit den experimentellen Ergebnissen überein. Die Benutzung von Korollar 3.11 (vgl. Tabelle 3.6 (a)) erweist sich oft als schwierig. Es werden häufig Hypothesen erzeugt, die einen gewichteten Trainingsfehler $\epsilon(h)$ haben, der nur unbedeutend kleiner als ϕ ist (vgl. (3.1)). Das bedeutet, daß die Approximation für ρ in diesen Fällen fast Null ist. Statt der Maximumsbildung wurde in (b) der Mittelwert gebildet (Schranke ist für diesen Fall nicht bewiesen!), d.h. $\epsilon = \sum_{t=1}^{T} \epsilon_t / T = \langle \epsilon_t \rangle$. Dieses ϵ ist der erwartete Trainingsfehler und stimmt (etwa)

ϕ \ #Zentren	Korollar 3.11 (a)				Korollar 3.11 (b)			
	7	13	20	30	7	13	20	30
0.3	-	-	0.534	0.535	-	-	0.584	0.684
0.5	0.000	0.007	0.102	0.349	0.101	0.229	0.398	0.589
0.7	0.141	0.160	0.1217	0.122	0.238	0.298	0.3969	0.490

Tabelle 3.6: Vergleich von experimentell bestimmtem und dem sich aus Korollar 3.11 ergebendem ρ für Datensatz #1 mit $\sigma^2 = 0$ für verschiedene ϕ und Anzahl der Zentren (experimentelle Ergebnisse siehe Tabelle 3.5).

mit dem in Kapitel 2 eingeführtem γ aus der PAC-Theorie überein. Damit findet man bessere Approximationen für ρ.
Man findet in den beiden Tabellen aber auch Inkonsistenzen: Für $\phi \neq \frac{1}{2}$ sind die theoretisch berechneten Werte größer als die experimentell bestimmten Werte. Man kann vermuten, daß das an numerischen Ungenauigkeiten liegt, die nach 10000 Iterationen sehr groß werden.

Zusammenfassung

Ensemble-Methoden können durch Kombination vieler Klassifikatoren runde Entscheidungslinien (d.h. ohne Ecken und Kanten) erzeugen und verringern die Varianz der Klassifikation.
AdaBoost optimiert eine Margin-Fehlerfunktion und erzeugt so große Stabilitäten der Trainingsbeispiele. Das Finden eines guten Minimums der Fehlerfunktion bzw. eines guten Maximums der kleinsten Stabilität wird durch einen intrinsischen Annealing-Prozeß unterstützt.
Asymptotisch ist AdaBoost ein Maximum-Margin-Klassifikator (mit Hard-Margin), was insbesondere bei Problemen der Handschrifterkennung zu guten Ergebnissen führt.

Kapitel 4

Warum AdaBoost verbessert werden sollte

4.1 AdaBoost ist nicht noise-robust

4.1.1 Noise verschlechtert die Ergebnisse

In Abbildung 3.1 findet man weitere Ergebnisse (vgl. auch Unterabschnitt 12) von Quinlan [51] für den Vergleich AdaBoost – Einzelklassifikator. Bei diesem Vergleich wurden die Klassenzugehörigkeiten in den Datensätzen bei 5% und 15% aller Trainingsbeispiele verändert (Klassifikationsrauschen). Bei 5% Rauschen hat AdaBoost bereits fast alle Vorteile in der Generalisierungsfähigkeit verloren. Bei 15% Klassifikationsrauschen ist AdaBoost in 19 von 27 Fällen schlechter als der Einzelklassifikator [51]. Es wurden keine Angaben zur Varianz dieser Werte gemacht.

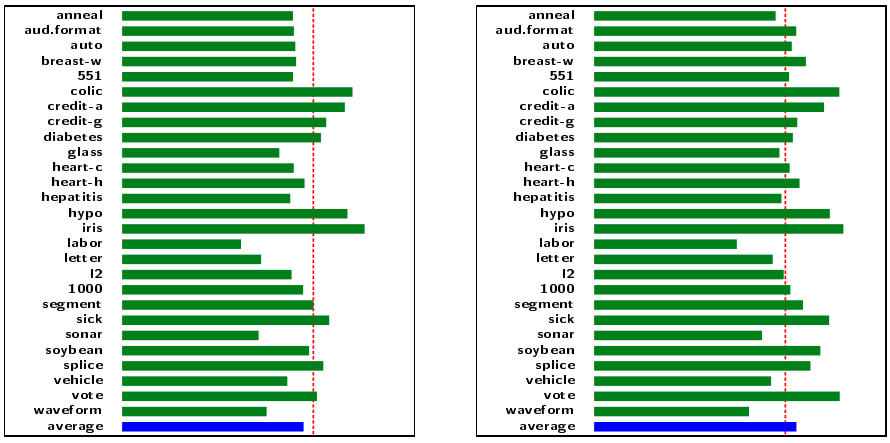


Abbildung 4.1: Vergleich Einzelklassifikator – AdaBoost auf UCI-Datensätzen mit 5% (links) und 15% (rechts) Klassifikationsrauschen (vgl. auch Abb. 3.1) [51]. AdaBoost scheint nicht stabil zu sein, wenn die Daten viel Rauschen enthalten.

Fügt man dem Datensatz #1 Rauschen hinzu, so findet man deutlich schlechtere Ergebnisse, insbesondere im Vergleich zu Bagging und dem Einzelklassifikator. Tabelle 4.1 zeigt die Ergebnisse dieses Vergleichs. Vergleicht man außerdem die Ergebnisse mit denen in

	Datensatz #1			
#Zentren	7	13	20	30
RBF-Netz	10.0	5.8	5.9	5.9
Bagging	6.9	5.5	5.7	6.1
AdaBoost	8.2	7.8	8.3	9.0

Tabelle 4.1: Testset-Fehler (in %) der Einzelklassifikatoren, für Bagging und für AdaBoost auf Datensatz #1 mit relativ viel Rauschen ($\sigma = 0.16$). Bei Bagging und AdaBoost wurden 10^4 Hypothesen kombiniert. Die Fehlerraten wurden über mehrere Realisierungen des Datensatzes gemittelt (Standardabweichung $0.2 - 1.0\%$)

Tabelle 3.2, so erkennt man das Problem: RBF-Netze und Bagging sind sehr robust gegen die Verrauschung der Daten: Der Generalisierungsfehler ist kaum schlechter als im Fall ohne Rauschen, und er vergrößert sich erst dann, wenn die Komplexität der RBF-Netze deutlich steigt. Das ist für AdaBoost jedoch nicht so: die Ergebnisse verschlechtern sich deutlich.

4.1.2 Overfitting bei hoher Zahl der Iterationen

In Freund et al. [22] und Schapire et al. [53] wurden Experimente mit AdaBoost und Decision Trees durchgeführt. Es wurde festgestellt, daß sich der Generalisierungsfehler mit steigender Anzahl der Iterationen stetig verkleinert. Das konnte für Datensätze mit Rauschen nicht bestätigt werden.
Betrachtet man die Generalisierungsfehler der Endhypothesen, die in jeder Iteration entstehen, so findet man Kurven, die schon nach einigen hundert (oder weniger) Iterationen ihr Minimum haben. Es ist also durchaus sehr entscheidend, wieviele Hypothesen kombiniert werden. Diese Erfahrung wurde in vielen Experimenten bei Vorhandensein von Rauschen gemacht. Abbildung 4.2 zeigt die Generalisierungsfehler der Endhypothesen auf Datensatz #1 mit starker Verrauschung. In Experimenten hat sich sehr oft gezeigt, daß sich das Minimum des Generalisierungsfehlers (über der Anzahl der Iterationen) stark vom Fehler der Endhypothese unterscheiden kann. Diesen Fall findet man insbesondere dann, wenn viel Rauschen in den Trainingsdaten ist. Das bedeutet, daß AdaBoost Hypothesen hoher Kapazität erzeugt, die zu Overfitting führen können. Oft wurde AdaBoost auf Probleme der Handschifterkennung angewandt [39, 60]. Da man davon ausgehen kann, daß diese Probleme größtenteils rauschfrei sind, konnte man diesen Effekt auch nicht so einfach bemerken. Die Komplexität konnte mit AdaBoost (fast) beliebig erhöht werden (was man durch AdaBoost gut erreichen kann) und es kam trotzdem nicht zu Overfitting [60]. Fast irreführend ist die Interpretation von Satz 2.9, die sagt, daß der Generalisierungsfehler nicht von der Anzahl der Iterationen abhängt. Sicher ist nur: Die linke Seite von (2.23) hängt nicht von T ab. Das Minimum des Generalisierungsfehlers muß aber nicht

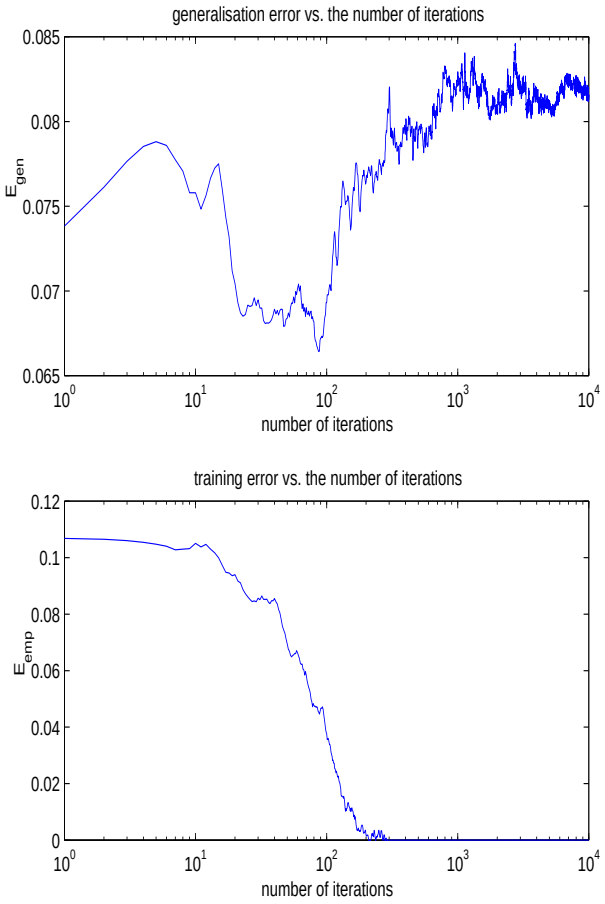


Abbildung 4.2: Typisches Beispiel für die Entwicklung des Generalisierungsfehlers (geglättet) mit der Anzahl der Iterationen für Datensatz #1 mit relativ viel Rauschen ($\sigma^2 = 0.16$): Der Testfehler (oben) hat bereits nach einigen Iterationen sein Minimum erreicht. Der Trainingsfehler ist zu diesem Zeitpunkt noch nicht Null. In diesem Fall ist das Erzeugen von Hypothesen, die Trainingsfehler Null haben, nicht sinnvoll. Oft zeigt sich sogar, daß die erste Hypothese (das RBF-Netz) besser generalisiert als die kombinierten Hypothesen.

unbedingt bei $T \to \infty$ liegen. Damit wäre es verwunderlich, wenn diese Schranke für den Generalisierungsfehler dicht wäre, denn diese Schranke gilt auch für $T \to \infty$.

4.2 Marginmaximierung vs. Overfitting

In Unterabschnitt 3.1.3 wurde erarbeitet, daß Bagging im Lernprozeß von AdaBoost eine Rolle spielt. Es erscheint plausibel, daß man durch Bagging die Komplexität des Ensembles nicht deutlich erhöhen kann. AdaBoost kann durchaus die Komplexität des Ensembles (beliebig) erhöhen (Trainingsfehler wird in jedem Fall zu Null!). Es spielt also eine wichtige Rolle, wie die Gewichte für die Beispiele und Hypothesen gewählt werden. Wählt man sie

uniform (Bagging), bleibt die Komplexität klein. Läßt man unterschiedliche Gewichte zu wie bei AdaBoost, so kann die Komplexität erhöht werden. Wählt man die Gewichte sehr unterschiedlich wie bei LP-AdaBoost (Grove et al. [27]), so kann die Komplexität deutlich gesteigert werden. Damit hängt die *effektive Komplexität* vom Wichtungsschema ab und je höher die effektive Komplexität ist, desto größer kann auch der Margin gemacht werden. Benutzt man nun die obere Schranke für den Generalisierungsfehler (2.23) und abstrahiert sie zu der Aussage (vgl. auch [64])

$$E_{gen} \leq \mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[\text{stab}(\mathbf{z}) \leq \theta] + \mathcal{O}(\text{effektive Komplexität des Ensembles}) \;, \qquad (4.1)$$

dann folgt, daß

1. der Generalisierungsfehler ein Minimum haben kann und daß
2. das Minimum nicht notwendigerweise durch Maximierung der kleinsten Stabilität erreicht wird.

In Breiman [8] und Grove et al. [27] versuchte man, die Generalisierungsfähigkeit durch die Größe der kleinsten Stabilität auf den Trainingsbeispielen zu erklären. Insbesondere wurde entgegen den Erwartungen festgestellt: Verwendet man ,,intelligentere" Algorithmen (z.B. *Lineare Programmierung*), um den Margin im Nachhinein zu vergrößern, so kommt es in vielen Fällen zu einer Verschlechterung der Generalisierungsfähigkeit [27]. Da durch dieses Vorgehen die (effektive) Komplexität des Ensembles erhöht wird, kann es wegen (4.1) zu einer Verschlechterung der Generalisierungsfähigkeit kommen.

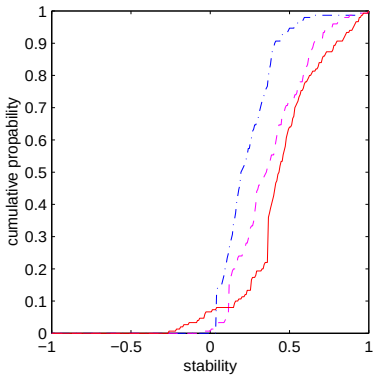


Abbildung 4.3: Typische Margin-Verteilungen (normalisiert) für eine SVM ohne Regularisierung $C \to 0$ (blau) und für zwei verschiedene Stärken der Regularisierung $C = 0.01$ (magenta) und $C = 0.4$ (rot) für Datensatz #1 ($\sigma = 0$)

Daß beim SV-Algorithmus (mit Slack-Variablen) nicht die kleinsten Stabilitäten maximiert werden, stützt diese Betrachtungen. Es besteht ein enger Zusammenhang zwischen AdaBoost und einer SVM ohne Slack-Variablen, der nicht zuletzt an der starken Ähnlichkeit der Margin-Verteilungsgraphen zu erkennen ist (vgl. Abbildung 4.3). Die gleiche Argumentation, die der Einführung von Slack-Variablen galt (siehe Vapnik [64]), kann benutzt werden, um zu erklären, daß AdaBoost nicht optimal ist und uneingeschränkte Maximierung der kleinsten Stabilität zu Overfitting führt. In Abbildung 4.3 sind dazu die Margin-Verteilungen dreier Support-Vektor-Maschinen mit unterschiedlich starker Regularisierung dargestellt. Ohne Regularisierung (d.h. $C \to 0$; ohne Slack-Variablen [64]) ist die Stabilität sehr vieler Beispiele klein. Erhöht man C, so läst man mehr und mehr zu, daß die Stabilitäten einiger Beispiele nicht mehr größer als die der Support-Vektoren sind. Dadurch erhält man eine allgmeine Erhöhung der restlichen Stabilitäten. Der Generalisierungsfehler ist für die rote Verteilung am kleinsten.

Beispiel 4.1 *Abbildung 4.4 zeigt beispielhaft, daß die Maximierung der kleinsten Stabilität bei vorhandenem Rauschen oder Klassifikationsrauschen nicht der beste Weg ist. Denn so kann jedes Beispiel eine beliebig große Bedeutung (,,Kraft") für die Entscheidungslinie haben.*

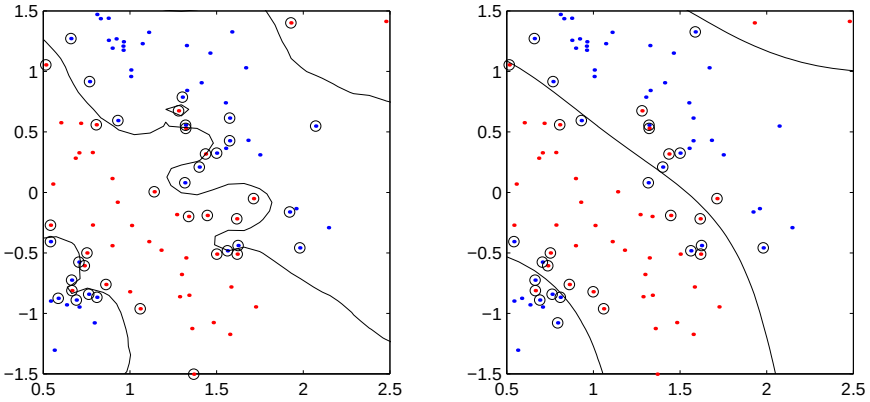


Abbildung 4.4: Entscheidungslinien (Ausschnitt) für AdaBoost (links) und eine SVM mit gut gewählter Regularisierung (rechts). Da die Trainingsfehler in jedem Fall zu Null werden, kann jedes Beispiel die Entscheidungslinie beliebig stark beeinflussen. Es kommt zu nichtglatten Entscheidungslinien. Unter der Annahme, daß die wahren Linien glatt sind, führt das zu Overfitting. Das Rauschen wird auswendig gelernt.

In Tabelle 4.2 findet man eine Zusammenstellung der Ergebnisse von Grove et al. [27] für LP-AdaBoost. Die Experimente demonstrieren, daß Maximierung der kleinsten Stabilität allein nicht der Schlüssel zu guter Generalisierungsfähigkeit ist.
Betrachtet man (4.1) und (2.23) genauer, erkennt man einen möglichen Grund für den Mißerfolg des Ansatzes von Grove et al.: Optimiert man nur die kleinste Stabilität, so wird es viele Beispiele geben, die deswegen eine kleinere Stabilität bekommen. Nun findet man in den oben angegebenen Schranken für den Generalisierungfehler aber nicht nur die Beispiele mit der kleinsten Stabilität, sondern auch die restlichen, da $\mathbf{P}_{\mathbf{z}\sim\mathbf{Z}}[\mathrm{stab}(\mathbf{z}) \leq \theta]$ etwas über die Stabilitätsverteilung aussagt. Es müssen Wege gefunden werden, wie man auch auf Beispielen größerer Stabilität, diese maximieren kann. Eine interessante Frage ist, wie optimale Margin-Verteilungen aussehen müssen, damit man zu kleinen Generalisierungsfehlern kommt. Es ist scheint klar, daß man versuchen sollte die Stabilitäten aller Beispiele zu erhöhen. Aber was hat man zu machen, wenn die Stabilität eines Beispiels nicht unabhängig von den anderen Stabilitäten ist? Oft verringert man die Stabilitäten einige Beispiele, wenn man die anderer erhöht. Es ist also eine Frage des Maßes für Stabilitätsverteilungen: Wieviel ist es Wert die Stabilität eines Musters, daß beispielsweise eine kleine Stabilität hat zu vergrößern, wenn dafür viele andere Beispiele an Stabilität verlieren – oder umgekehrt. AdaBoost strebt ein Extrem asymptotisch an: nur die kleinste Stabilität ist wichtig.
Es bleibt fraglich, ob die Betrachtung der Margin-Verteilung allein ausreicht, um gute Ergebnisse zu erhalten.

	AdaBoost		LP-AdaBoost	
Datensatz	E_{gen}	ρ	E_{gen}	ρ
Adiology	52.3	-1.00	52.3	-1.00
Banding	27.0	-0.08	22.4	0.02
Chess	5.2	-0.10	6.5	0.00
Colic	18.0	-0.18	23.1	-0.01
Glass	47.8	-1.00	47.8	-1.00
Hepatitis	18.2	-0.03	21.4	-0.04
Labor	6.5	0.26	7.0	0.30
Promoter	8.8	0.17	9.5	0.21
Sonar	16.8	0.05	17.8	0.11
Soybean	69.5	-1.00	69.5	-1.00
Splice	10.6	-0.70	17.1	-0.41
Vote	3.4	-0.06	4.0	-0.00
Wine	4.0	0.01	3.1	0.07

Tabelle 4.2: Vergleich der Generalisierungsfähigkeiten (in %) von Klassifikatoren, die mit AdaBoost und mit LP-AdaBoost trainiert wurden [27] (über 100 Datensätze gemittelt). LP-AdaBoost maximiert nach Abschluß des AdaBoost-Prozesses die kleinste Stabilität auf den Trainingsbeispielen durch lineare Programmierung. Bei allen Datensätzen (UCI-Repository) wird durch die LP der Margin vergrößert. Entgegen der Erwartung verschlechtert sich dadurch die Generalisierungsfähigkeit in fast allen Fällen.

Kapitel 5

Ansätze zu Verbesserungen des AdaBoost-Algorithmus

AdaBoost kann möglicherweise durch eine geeignete Wahl von ϕ verbessert werden. Außerdem kann man Veränderungen an der Fehlerfunktion vornehmen, um den Einfluß einzelner Beispiele zu begrenzen, um mehr Robustheit gegen Rauschen zu erreichen. In den Abschnitten 5.3 und 5.4 wird gezeigt, wie man die Kombination der Hypothesen zur Endhypothese verbessern kann. Die verschiedenen Modifikationen von AdaBoost werden im nächsten Kapitel experimentell auf ihre Brauchbarkeit in realen Anwendungen getestet.

5.1 Optimierung von ϕ

Da AdaBoost das Problem der Linearen Programmierung zu asymptotisch zu approximieren versucht[1] (vgl. Anmerkung 3.9 und [6, 27]) und die Qualität der Lösung dabei möglicherweise von der Geschwindigkeit des Annealing-Prozesses abhängt, kann folgen, daß AdaBoost mit kleinerem ϕ bessere Approximationen zur Lösung eines LP-Solvers liefert [47]. Das bedeutet, daß der Margin (=minimale Stabilität) größer wird (vgl. auch Korollar 3.11).
Man betrachte Gleichung (3.11) zur Bestimmung von b_t. Für $\phi = \frac{1}{2}$ ist der Term $\log \frac{\phi}{1-\phi} = 0$. Wird ϕ sehr Nahe an 1 gewählt, so hat man $\log \frac{\phi}{1-\phi} \to \infty$. Der linke Term der rechten Seite in (3.11) hat dann keinen Einfluß mehr und alle Hypothesen werden gleichgewichtet. Dem entgegen steht der Effekt, daß mit ϕ auch die Geschwindigkeit der Abkühlung vergrößert wird, also im Annealing-Prozess schneller die Asymptotik erreicht wird. Wenn man eine feste Anzahl von Iterationen voraussetzt, werden die Beispiele in jeder Iteration extremer gewichtet, da $|\mathbf{b}|$ größer wird. Der umgekehrte Effekt tritt ein, wenn ϕ sehr klein, d.h. nahe $\max \epsilon_t$ gewählt wird.
So ist es mit ϕ möglich, die ,,Quelle der Komplexität" zu bestimmen: Wählt man ϕ groß, sind die Gewichte der Hypothesen fast gleich (keine Komplexitätserhöhung) und die Gewichte der Beispiele sind sehr unterschiedlich (Komplexitätserhöhung). Wird ϕ klein gewählt, so ist die Approximation zur Lösung durch linearen Optimierung besser, d.h. die Gewichte der Hypothesen sind sehr unterschiedlich (Komplexitätserhöhung). Da die

[1] Das ist noch nicht bewiesen es wird vermutet, daß AdaBoost asymptotisch gegen die Lösung eines Linearen Programmes konvergiert [27].

	Ziffer										Gesamt
Architektur	0	1	2	3	4	5	6	7	8	9	
SVM	16	8	25	19	29	23	14	12	25	16	4.2%
AdaBoost $\phi = 0.7$	17	12	30	**20**	**30**	25	14	11	31	16	4.9 %
AdaBoost $\phi = 0.5$	**14**	12	29	21	**30**	**21**	**11**	10	30	16	4.5%
AdaBoost $\phi = 0.1$	16	**11**	**22**	23	31	26	12	**9**	29	**11**	4.5%
AdaBoost $\phi = 0.02$	17	**11**	27	22	37	23	13	10	**27**	12	4.5%

Tabelle 5.1: Vergleich der Anzahlen der Beispiele aus dem Testdatensatz, die von AdaBoost mit verschiedenen ϕ (20 Iterationen mit RBF-Netz mit 20 Zentren) und von einer SVM mit (entsprechender Regularisierung) falsch klassifiziert wurden. [59]

Abkühlung deutlich langsamer erfolgt, unterscheiden sich die Gewichte der Beispiele nicht deutlich (keine Komplexitätserhöhung).
Zusammenfassend kann man sagen, daß ϕ einen komplexen Einfluß auf den Lernprozeß hat. Für jedes Extrem (ϕ groß bzw. klein) kann es wünschenswerte Folgen geben. Jedoch lassen sich die positiven Eigenschaften nicht vereinigen. Es scheint, daß $\phi = \frac{1}{2}$ eine gute Wahl ist, die in vielen Fällen gute Ergebnisse liefert. In Experimenten hat sich auch gezeigt, daß der Einfluß von ϕ schwer berechenbar ist. In manchen Fällen erhält man bessere und in machen Fällen schlechtere Ergebnisse für große bzw. kleine ϕ. In Experimenten mit unterschiedlichen ϕ hat sich für den USPS-Datensatz (vgl. Anhang A.3.2) gezeigt, daß kleine ϕ machmal bessere Minima erzeugen als große. Tabelle 2 zeigt die Ergebnisse von AdaBoost für den USPS-Datensatz. Interessanterweise sind die Generalisierungsfehler für alle Fälle mit $\phi \leq \frac{1}{2}$ gleich groß; nur für $\phi = 0.7$ erhält man schlechtere Ergebnisse, was auch mit den vorherigen Betrachtungen plausibel ist. Für einige Ziffern ist sind die Ergebnisse für $\phi = 0.7$ am besten. Möglicherweise sind diese Ziffern sehr gut von den anderen Ziffern trennbar. Für ein großes ϕ werden die Werte von $|\mathbf{b}|$ größer und der Wettbewerb zwischen den Beispielen schneller stärker und man erreicht im leicht trennbaren Fall mit nur 20 Iterationen bessere Ergebnisse.
Man könnte nun argumentieren, daß aus jeder Ziffer ein anderes Lernproblem resultiert, daß möglicherweise jeweils ein anderes ϕ erfordert. Es sieht zunächst so aus, daß das stimmt, da die Testfehler auf den einzelnen Ziffern für verschiedene ϕ sich z.T. deutlich unterscheiden. Benutzt man jeweils den besten Klassifikator (d.h. das beste ϕ) auf der jeweiligen Klasse (auf dem Testset gewählt!), so stellt man leider nur eine sehr kleine Veränderung fest und erält eine Fehlerrate von 4.4%. Die Veränderung ist kleiner als erwartet. Insbesondere weil die daraus resultierenden Einzelfehlerraten in sechs Fällen kleiner als die der SVM ist, war zu erwarten, daß der Testfehler entsprechend sinkt. Möglicherweise ist eine einzelne Realisierung nicht des Trainings nicht aussagekräftig genug[2].

[2]Es wurden aber keine weiteren Experimente unternommen, da der Rechenaufwand sehr hoch ist. Für einen vollständigen Klassifikator (10 Klassen) benötigt man etwa 200-300 Stunden Rechenzeit auf einer SPARC ULTRA 2, 200Mhz (bei RBF-Netzen mit 20 Zentren, 50 RBF-Iterationen und 20 AdaBoost-Iterationen).

5.2 Regularisierung durch veränderte Kostenfunktion

Mit Satz 3.10 ist klar, daß alle Beispiele nach endlich vielen Iterationen positive Stabilitäten haben werden (für entsprechende ϕ und ϵ_t). Das wird durch wiederholte Benutzung hoher Gewichte für die schwierigen Beispiele (z.B. **S**-Beispiele) erreicht.
Theoretisch muß es zu Overfitting kommen, da jedes Beispiel mit beliebig großer Kraft auf die Entscheidungsgrenze einwirken darf (siehe auch Betrachtungen in Abschnitt 4.2). Die Entscheidungslinie kann dadurch weniger glatt als wünschenswert werden, insbesondere, wenn nicht jedem Beispiel absolut vertraut werden darf (Ausreißer, Rauschen).
Dieses Problem von AdaBoost kann man beheben, wenn man das lineare Optimierungsproblem, das AdaBoost zugrunde liegt verändert (Einführung eines *Soft-Margin*). Dazu betrachte man (3.3) und setze Korollar 3.11 voraus: In der Asymptotik gilt für alle Beispiele (Proposition 3.9)

$$-\frac{\text{stab}(\mathbf{z}^{(i)})}{2}+\frac{1}{2}-\phi<\Theta$$

und auch

$$\text{stab}(\mathbf{z}^{(i)})\geq\rho\geq\Theta+1-2\phi\ , \tag{5.1}$$

wobei die Ungleichung für die **S**-Beispiele zur Gleichung wird. Man betrachte die Ungleichung (5.1) genauer: Wenn man erlaubt, daß diese Ungleichung nicht notwendigerweise für alle Beispiele erfüllt sein muß, d.h. wenn man sie verändert zu

$$\text{stab}_t(\mathbf{z}^{(i)})\geq\rho-\xi_i^t\ , \tag{5.2}$$

kann man das Overfitting verhindern (Analogie zu SV-Slack-Variablen). Die Slack-Variablen haben die intuitive Bedeutung, daß sie das Klassifkationsrauschen der Daten des Trainingssets approximieren. Hat man Rauschen auf den Eingabedaten, so approximieren sie das *effektive*[3] *Rauschen* der Daten.
Unter der Annahme, daß insbesondere die Beispiele von der Basishypothese schwer klassifiziert werden können, die stark verrauscht sind, sollten die ξ_i dann groß sein, wenn ein Beispiel bereits besonders hohes Gewicht in den vorherigen Iterationen hatte. Das ist so, wenn man

$$\text{(a)}\qquad \xi_i^t=Cv_i^t \qquad\qquad \text{oder} \qquad\qquad \text{(b)}\qquad \xi_i^t=C(v_i^t)^2 \tag{5.3}$$

wählt, wobei

$$v_i^t=\sum_{r=1}^{t}c_t p_i^r \tag{5.4}$$

NB: Wird $C=0$ gewählt, so erhält man den originalen AdaBoost-ähnlichen Algorithmus. Je größer C, desto mehr wird eine Gewichtung zugunsten eines Beispiels bestraft. Wird C sehr groß gewählt, erhält man eine spezielle Form des Baggings; alle Trainingsbeispiele haben die gleichen Gewichte.
Für die Fehlerfunktion $\widetilde{G}$ ergibt sich damit die Form

$$\widetilde{G}_{reg}(\mathbf{b}^t,\mathbf{h}^t)=\sum_{i=1}^{l}\exp\left(|\mathbf{b}^t|(\text{err}_t(\mathbf{z}^{(i)},\mathbf{c}^t)-\phi-\xi_i^t)\right)\ . \tag{5.5}$$

[3]Effektiv deshalb, weil die Verrauschung nur dann wichtig ist, wenn das Beispiel in der Nähe der Entscheidungslinie liegt.

Algorithm AdaBoost$_{reg1}(\phi, C)$

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$

Basislernalgorithmus $\mathcal{L}$

Anzahl der zu kombinierenden Hypothesen T

Initialize:

Gewichte für Trainingsbeispiele: $w_i^1 = 1/l, \qquad W_i^0 = 0$

Do for $t = 1 : T$,

1a. Setze $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{l} w_i^t}$

1b. Erzeuge Trainingsbeispiele $\mathbf{Z}_t$ aus $\mathbf{Z}$ entsprechend der Verteilung $\mathbf{p}^t$

2a. Starte $\mathcal{L}$ mit $\mathbf{Z}_t$ und erhalte $h_t : \mathbf{x} \mapsto \{-1, 1\}$

3a. Bestimme Minimum b_t von

$$G_{reg1}(b_t) = \sum_{i=1}^{l} \exp\left(\mathrm{err}_{t-1}(\mathbf{z}^{(i)}, \mathbf{b}^{t-1}) - |\mathbf{b}^{t-1}|\phi - CW_i^{t-1} + b_t(l_t(\mathbf{z}^{(i)}) - \phi - Cp_i^t)\right)$$

3b. Berechne die neuen Gewichte für die Trainingsbeispiele

$$\begin{aligned} w_i^{t+1} &= p_i^t \beta_t^{I(h_t(\mathbf{x}^{(i)}) = y^{(i)}) + Cp_i^t}, \\ W_i^t &= W_i^{t-1} + b_t p_i^t \end{aligned}$$

wobei $\beta_t = \exp(-b_t)$ ist.

Output:

Endhypothese

$$f(\mathbf{x}) = \frac{1}{|\mathbf{b}|} \sum_{t=1}^{T} b_t h_t(\mathbf{x})$$

Abbildung 5.1: Der AdaBoost-ähnliche Algorithmus mit (linearer) Regularisierung. Problematisch kann die Minimierung von G_{reg1} in Zeile 3a. sein, da für große Werte von $|\mathbf{b}|$ die entstehenden Zwischenergebnisse sehr kleine Zahlen sind und es zu numerischen Ungenauigkeiten kommt.

Durch Ableitung von $\widetilde{G}_{reg}$ nach $l_t(\mathbf{z}^{(i)})$ erhält man hieraus auch die Gewichte p_i^t für die Beispiele (vgl. Lemma 3.4 und (3.9))

$$\begin{aligned} p_i^t &= \frac{\frac{\partial \widetilde{G}(\mathbf{b}^{t-1},\mathbf{h}^{t-1})}{\partial l_t(\mathbf{z}^{(i)})}}{\sum_{j=1}^{l} \frac{\partial \widetilde{G}(\mathbf{b}^{t-1},\mathbf{h}^{t-1})}{\partial l_t(\mathbf{z}^{(j)})}} \\ &= \frac{\exp\left\{|\mathbf{b}|\left(\mathrm{err}_{t-1}(\mathbf{z}^{(i)},\mathbf{c}^{t-1}) - \phi - \xi_i^{t-1}\right)\right\}}{\sum_{j=1}^{l} \exp\left\{|\mathbf{b}^{t-1}|\left(\mathrm{err}_{t-1}(\mathbf{z}^{(j)},\mathbf{c}^{t-1}) - \phi - \xi_j^{t-1}\right)\right\}} \; . \end{aligned} \tag{5.6}$$

Damit kann man eine Update-Regel für die Beispielsgewichtungen entwickeln:

$$\begin{aligned} w_i^t &= \exp\left\{|\mathbf{b}^{t-1}|\left(\mathrm{err}_{t-1}(\mathbf{z}^{(i)},\mathbf{c}^{t-1}) - \phi - \xi_i^{t-1}\right)\right\} \\ &= \exp\left\{\sum_{r=1}^{t-1} b_r l_r(\mathbf{z}^{(i)}) - \phi \sum_{r=1}^{t-1} b_r - \xi_i^{t-1} \sum_{r=1}^{t-1} b_r\right\} \\ &= \exp\left\{\sum_{r=1}^{t-2} b_r l_r(\mathbf{z}^{(i)}) - \phi \sum_{r=1}^{t-2} b_r\right\} \exp\left\{b_{t-1} l_{t-1}(\mathbf{z}^{(i)}) - \phi b_{t-1}\right\} \cdot \\ &\qquad \cdot \exp\left\{-\xi_i^{t-1} \sum_{r=1}^{t-1} b_r\right\} \\ &= w_i^{t-1} \exp\left\{b_{t-1} I(y^{(i)} \neq h_{t-1}(\mathbf{x}^{(i)})) + \xi_i^{t-2}|\mathbf{b}^{t-2}| - \xi_i^{t-1}|\mathbf{b}^{t-1}|\right\} \; . \end{aligned} \tag{5.7}$$

Setzt man $b_{t-1} = \log \frac{1}{\beta_{t-1}}$, erhält man aus (5.7)

$$w_i^t \sim p_i^{t-1} \beta_{t-1}^{I(y^{(i)} = h_{t-1}(\mathbf{x}^{(i)}))} \exp\left\{\xi_i^{t-2}|\mathbf{b}^{t-2}| - \xi_i^{t-1}|\mathbf{b}^{t-1}|\right\} \; . \tag{5.8}$$

Für $C = 0$ ist $\xi_i \equiv 0$ und man erhält die bei AdaBoost benutzte Formel zur Bestimmung von $\mathbf{w}^t$ (vgl. Zeile 3c. in Abbildung 2.3). Für den Fall (5.3.a) vereinfacht sich (5.8) zu

$$w_i^t \sim p_i^{t-1} \beta_{t-1}^{I(y^{(i)} = h_{t-1}(\mathbf{x}^{(i)})) + C p_i^{t-1}} \; . \tag{5.9}$$

Die Bestimmung des Gewichtes der Hypothese b_t ist schwieriger. Eine analytische Lösung wie im Beweis von Proposition 2.5 (siehe Anhang A.2.1) ist leider nicht allgemein möglich. Man hat b_t durch eine Line-Search-Prozedur über

$$G_{reg}(b_t) = \sum_{i=1}^{l} \exp\left(\mathrm{err}_{t-1}(\mathbf{z}^{(i)},\mathbf{b}^{t-1}) - |\mathbf{b}^{t-1}|\phi + b_t(l_t(\mathbf{z}^{(i)}) - \phi) - |\mathbf{b}^t|\xi_i^t\right) \tag{5.10}$$

zu optimieren. Für $C = 0$ erhält man die aus Gleichung (2.12) bekannte Form. Da $\frac{\partial^2}{\partial b_t^2} G_{reg} > 0$ gilt und damit die Funktion konvex ist, existiert eine eindeutige Lösung für b_t.

In Abbildung 5.1 findet man die Beschreibung des daraus entstehenden Algorithmus für den einfacheren Fall der Regularisierung (linear: (5.3.a)). Die Veränderungen für den zweiten Fall ergeben sich analog zu den obigen Betrachtungen. In Tabelle 5.2 findet man die

	Datensatz #1			
#Zentren	7	13	20	30
C=1	7.4	7.2	8.4	8.9
C=10	7.0	7.7	7.2	9.1
C=100	6.5	7.3	7.3	7.6
C=1e3	5.9	5.9	5.9	6.9
C=5e3	5.4	5.6	6.1	6.0

Tabelle 5.2: Klassifikationsfehler auf dem Testdatensatz (in %) von $\text{AdaBoost}_{\text{reg2}}$ (quadratisch: (5.3.a)) für Datensatz #1 mit relativ viel Rauschen ($\sigma = 0.16$). Es wurden 10^4 Hypothesen kombiniert. (Standardabweichung etwa $0.4\% - 0.8\%$)

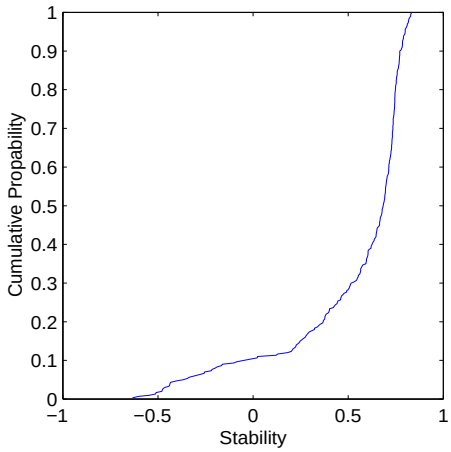


Abbildung 5.2: Margin-Verteilungsgraph von $\text{AdaBoost}_{\text{reg2}}$ für Datensatz #1 mit relativ viel Rauschen ($\sigma = 0.16$).

Ergebnisse einiger Experimente zu dieser Modifikation des Algorithmus. Es zeigt sich, daß durch die Einführung von Slack-Variablen deutlich bessere Ergebnisse erreichbar sind, sofern der Parameter C günstig eingestellt ist (vgl. auch mit Ergebnissen in Tabelle 4.1). Der resultieren Margin-Verteilungsgraph für den besten Eintrag in Tabelle 5.2 ist in Abbildung 5.2 dargestellt (vgl. auch Abbildungen 2.4 und 4.3). Man findet eine gute Übereinstimmung zwischen der Form dieser Margin-Verteilung mit der der regularisierten Support-Vektor-Maschnine.
Dieser Algorithmus wurde auch auf dem USPS-Datensatz (vgl. Anhang A.3.2) getestet. Die Abbildung 5.3 zeigt die Testfehler für verschiedene Regularisierungskonstanten C über der Anzahl der Iterationen. Für zu kleine C erhält man keine besonders guten Ergebnisse und für zu große auch nicht. Wählt man ein geeignetes C, so kommt man mit den Fehlerraten sehr nahe an die zur Zeit besten Klassifikationsalgorithmen heran (vgl. Tabelle A.3.2 im Anhang). Man sieht auch, daß man hier durch Early-Stopping bei zu schwacher Regularisierung nicht (deutlich) besser werden kann.

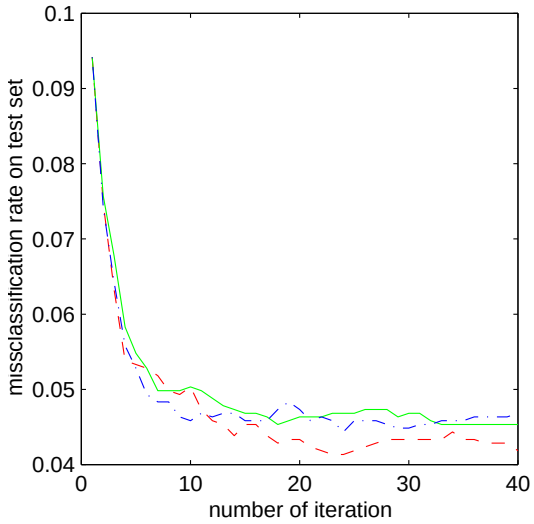


Abbildung 5.3: Ergebnisse von $\text{AdaBoost}_{\text{reg2}}$ auf dem USPS-Datensatz für verschiedene Regularisierungskonstanten: $C = 500$ (blau), 2000 (rot) und 8000 (grün). Bei $C = 2000$ findet man bei Iteration 23-34 einen Fehler von 0.4.1% und nach 40 Iterationen von 4.3% (vgl. Anhang A.3.2).

5.3 Lineare Programmierung zur Maximierung der Stabilitäten

Proposition 3.9 zeigt, daß AdaBoost das Problem der Linearen Programmierung approximativ löst. Um diese Lösung im Nachhinein zu verbessern, wurde in Grove et al. [27] ein linearer Optimierer benutzt, um die Gewichte der Hypothesen zu bestimmen. Ziel dieser Optimierung ist es, die kleinsten Stabilität der Beispiele zu maximieren (was nicht notwendigerweise die besten Ergebnisse liefern muß [27]). Dazu definiert man das Optimierungsproblem folgender Art: Sei L die Fehlermatrix, die nach Ablauf des AdaBoost-Algorithmus aus den Fehlern der Einzelhypothesen wie folgt bestimmt wird:

$$L_{i,t} = \begin{cases} 1 & \text{wenn } h_t(\mathbf{x}^{(i)}) = y^{(i)} \\ -1 & \text{wenn } h_t(\mathbf{x}^{(i)}) \neq y^{(i)} \end{cases} . \tag{5.11}$$

Es wird nun der Gewichtsvektor $\mathbf{c}$ für die Hypothesen gesucht, der ρ (kleinste Stabilität) maximiert. Der Vektor der Stabilitäten der Beispiele ergibt sich aus $L\mathbf{c}$. Dabei hat man die Nebenbedingungen $\sum_{r=1}^{T} c_r = 1$, $c_t > 0$ und $\sum_{r=1}^{T} L_{i,r} c_r \geq \rho$ zu erfüllen. Hieraus ergibt sich der in Abbildung 5.4 dargestellte Algorithmus *LP-AdaBoost* [27].
Tabelle 4.2 zeigt die Ergebnisse eines experimentellen Vergleichs [27] von AdaBoost mit LP-AdaBoost. Die kleinste Stabilität ρ ist in jedem Fall größer als bei AdaBoost (sowohl experimentell als auch theoretisch). Trotzdem sind die Generalisierungsfehler von LP-AdaBoost in fast allen Fällen schlechter als von AdaBoost. Das hat (mindestens) zwei Ursachen: einerseits die in Abschnitt 4.2 genannten Gründe bzgl. Overfitting und andererseits, daß bei LP-AdaBoost die Stabilitäten der leichteren Beispiele nicht erhöht werden.

Algorithm LP-AdaBoost

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$

T Hypothesen $\{h_t\}_{t=1}^{T}$, die durch AdaBoost erzeugt wurden

Do:

Erzeuge Fehlermatrix gemäß (5.11)

Löse Problem der linearen Optimierung:

Minimiere ρ unter den Nebenbedingungen

$$\begin{aligned} \sum_{t=1}^{T} L_{i,t} c_t &\geq \rho \qquad i = 1, \ldots, l \\ c_t &\geq 0 \qquad t = 1, \ldots T \\ \sum_{t=1}^{T} c_t &= 1 \end{aligned}$$

Abbildung 5.4: Der LP-AdaBoost Algorithmus [27]

Außerdem erzeugt ein LP-Solver i.A. sparse Lösungen[4], d.h. es wird nur eine Teilmenge der im ursprünglichen Ensemble vorhandenen Hypothesen benutzt. Mit den Ergebnissen in 3.1.2 und 3.1.3 kann man erklären, daß sparse Hypothesenkoeffizienten zu höheren Generalisierungsfehlern führen können.

5.3.1 Beschränkung der Gewichte

Wichtigkeit der Hypothesen

Durch eine Erweiterung des lineraren Programmes kann man die Wichtigkeit der Hypothesen nach oben und nach unten beschränken. Da jede Einzelhypothese aufgrund der höheren Varianz (vgl. Abschnitt 3.1.3) möglicherweise unzuverlässig ist, kann das zu einer Verbesserung führen. Man erreicht außerdem, daß genügend Hypothesen kombiniert werden, so daß die Entscheidungslinien glatt bzw. rund genug werden (vgl. Abschnitt 3.1.2). Technisch ist die Begrenzung der Wichtigkeiten der Hypothesen unproblematisch. Man führt $C_h^{\downarrow}$ als die untere Grenze und $C_h^{\uparrow}$ als die obere Grenze für die Wichtigkeit der Hypothesen ein. Das Ungleichungssystem in Abbildung 5.4 wird dann um folgende Ungleichungen erweitert:

$$C_h^{\downarrow} \leq c_t \leq C_h^{\uparrow} \qquad t = 1, \ldots, T \ . \tag{5.12}$$

Läßt man $C_h^{\downarrow}$ von unten oder $C_h^{\uparrow}$ von oben gegen $1/T$ konvergieren, so entspricht das dem der Fall der Gleichgewichtung. Die Komplexität wird dadurch vermindert (Analogie zu

[4]Unter einer *sparsen Lösung* wird hier verstanden, daß der Gewichtsvektor für die Hypothesen relativ wenig von Null verschiedene Werte enthält.

C_δ	1	10^{-1}	10^{-2}	10^{-3}	10^{-5}	LP-AdaBoost	AdaBoost
E_{gen}	8.4	8.4	7.1	6.9	6.6	8.4	6.2

Tabelle 5.3: Vergleich der Generalisierungfehler von LP-AdaBoost, LP-AdaBoost mit Gewichtsbeschränkung für die Beispiele und AdaBoost für ein Ensemble von 200 Hypothesen auf Datensatz #1 mit relativ viel Rauschen ($\sigma = 0.16$). C_p wurde $1/300 + C_\delta$ gewählt (300 Trainingsbeispiele). Die besten Generalisierungsfehler werden bemerkenswerterweise für die Gleichgewichtung der Beispiele erreicht. Das Ergebnis von LP-AdaBoost wird deutlich verbessert, aber der originale AdaBoost-Algorithmus erreicht kleinere Fehlerraten.

$\phi \to 1$).
Leider erbrachte diese Modifikation keine wesentliche Veränderung des Ergebnisses[5]. Für Hypothesenklassen, die eine größere Varianz (als RBF-Netze) haben (wie z.B. Decision Trees) müßte dieser Ansatz mehr Erfolge bringen.

Wichtigkeit der Beispiele

In Abschnitt 4.2 wurde herausgestellt, daß AdaBoost unter anderem deswegen asymptotisch schlecht generalisiert, weil der Einfluß eines Beispiels sehr groß werden kann (vgl. auch Ansatz in 5.2). Ähnlich wie in Unterabschnitt 5.3.1 kann man – die Gewichtungen der Beispiele im AdaBoost-Prozess berücksichtigend – das Gewicht eines Beispiels begrenzen.
Sei p_i^t das Gewicht des i-ten Beispiels in der t-ten Iteration von AdaBoost und $C_p > 1/l$ die obere Schranke für das Gewicht

$$v_i = \sum_{t=1}^{T} c_t p_i^t \tag{5.13}$$

eines Beispiels. Dann kann man folgende Ungleichungen zum System in 5.4 hinzufügen:

$$\sum_{t=1}^{T} c_t p_i^t \leq C_p \qquad\qquad i = 1, \ldots, l \ .$$

Diese Erweiterung konnte die Lösung von LP-AdaBoost deutlich verbessern (abhängig vom Parameter C_p). In den durchgeführten Experimenten wurde jedoch keine Verbesserung gegenüber dem Ergebnis von AdaBoost festgestellt. In Tabelle 5.3 findet man einige experimentelle Ergebnisse. Es wird damit gezeigt, daß die Idee prinzipiell richtig ist, aber noch nicht alle Probleme löst.

5.3.2 Einführung von Slack-Variablen

Wie bereits in Abschnitt 4.2 angedeutet, ist Maximierung der minimalen Stabilität problematisch, da die Stabilitäten der leichteren Beispiele nicht Gegenstand des Optimierungsprozesses sind. Betrachtet man noch einmal Gleichung (2.23) oder (4.1), so folgt,

[5] Sie kann eine relative Verbesserung von weniger als 3% erreichen und war damit immer noch deutlich schlechter als das originale AdaBoost

C_s	1	10^{-1}	50^{-1}	10^{-2}	200^{-1}	500^{-1}	10^{-3}
E_{gen}	8.4	8.4	7.7	5.6	6.3	7.2	8.7

Tabelle 5.4: Klassifikationsfehler (in %) auf dem Testdatensatz von LP$_{\text{reg}}$-AdaBoost (mit Slack-Variablen) für eine Realisierung von Datensatz #1 ($\sigma = 0.16$) für verschiedene C_s. Es wurden 200 Hypothesen kombiniert. Die für $C_s = 10^{-2}$ erreichten Ergebnisse sind die besten bis hierhin erreichten.

daß auch die übrigen Beispiele eine möglichst große Stabilität haben müssen; insbesondere diejenigen Beispiele, die Stabilitäten nur wenig größer als ρ haben. Beim Support-Vektor-Algorithmus bedient man sich dazu der Slack-Variablen (vgl. Abschnitt 5.2).
Dazu verändert man das Optimierungsproblem in 5.4 wie folgt:

Minimiere

$$J = -\rho + C_s \sum_{t=1}^{T} \xi_t$$

mit den Nebenbedingungen

$$\begin{aligned} \sum_{t=1}^{T} L_{i,t} c_t &\geq \rho - \xi_t && i = 1, \ldots, l \\ c_t &\geq 0 && t = 1, \ldots, T \\ \sum_{t=1}^{T} c_t &= 1 \ , \end{aligned} \tag{5.14}$$

wobei C_s eine fest gewählte Konstante ist.

Damit werden kleine Verletzungen der Ungleichungen zugelassen, wenn dadurch die Stabilität der restlichen Beispiele höher wird. Wie tolerant die Optimierung gegenüber solchen Verletzungen ist, wird durch die Regularisierungskonstante C_s bestimmt.
Tabelle 5.4 zeigt die Ergebnisse dieser Modifikation für Datensatz #1. Der Generalisierungsfehler zeigt ein Minimum für 10^{-2}. Das entspricht den Erwartungen, da wir für $C \to \infty$ den Fall LP-AdaBoost erreichen und bei $C = 0$ die Ungleichungen (5.14) keine Rolle mehr spielen.

5.4 Quadratische Programmierung zur Maximierung der Stabilitäten

Beim Support-Vektor-Algorithmus wird eine andere Art der Optimierung benutzt. Dort minimiert man die Länge des Gewichtsvektors (L_2-Norm) im Feature-Raum, was der Komplexität der Hypothese entspricht [64]. Außerdem werden Slack-Variablen für den nichtseparablen Fall [3, 14, 64] benutzt und man löst das folgende Problem der Quadratischen Optimierung:

Minimiere

$$J = ||\mathbf{w}||^2 + C\sum_{i=1}^{l} \xi_i \tag{5.15}$$

mit den Nebenbedingungen

$$y^{(i)} h^{\mathbf{w}}(\mathbf{x}^{(i)}) \geq 1 - \xi_i \qquad i = 1, \ldots, l \ . \tag{5.16}$$

Man kann diese Art der Optimierung sinngemäß auch auf die Bestimmung der Gewichte der Hypothesen für AdaBoost anwenden. In den Abschnitten 4.2 und 5.1 wurde festgestellt, daß sich die Komplexität der Endhypothese erhöht, wenn man unterschiedliche Gewichtungen der Basishypothesen zuläßt. Ein Maß (wenn auch sicher nicht das einzige) für die Komplexität der Endhypothese ist also die L_2-Norm des Hypothesengewichtsvektors[6]. Mit dem gleichen Ansatz wie beim Support-Vektor-Algorithmus erhält man das folgende Optimierungsproblem:

Minimiere

$$J = ||\mathbf{b}||^2 + C\sum_{i=1}^{l} \xi_i \tag{5.17}$$

mit den Nebenbedingungen

$$\begin{aligned} y^{(i)} h^{\mathbf{b}}(\mathbf{x}^{(i)}) &\geq 1 - \xi_i \qquad i = 1, \ldots, l \ . \\ b_t &\geq 0 \end{aligned} \tag{5.18}$$

Gleichung (5.18) kann man umschreiben und erhält

$$\sum_{t=1}^{T} b_t L_{i,t} \geq 1 - \xi_i \qquad i = 1, \ldots, l \ . \tag{5.19}$$

NB: ρ ist nun auf 1 festgesetzt, dafür fällt aber die Nebenbedingung $|\mathbf{b}| = 1$ weg und der Gewichtsvektor für die Hypothesen summiert sich nun nicht mehr notwendigerweise zu 1. Bei der Berechnung der Ausgabe wird aber wieder der normalisierte Vektor $\mathbf{c} = \mathbf{b}/|\mathbf{b}|$ benutzt.

Möglicherweise problematisch ist hier die Tatsache, daß zwar die Komplexität der Endhypothese durch das Gleichgewichten der Hypothesen klein gehalten werden kann, aber jede Hypothese durch unterschiedliche Gewichtung der Trainingsbeispiele entstanden ist, die bisher nicht einfließt. Ein Lösungsansatz ist es, eine Gewichtung der Hypothesen zu finden, so daß der Einfluß v_i (vgl. Gleichung 5.13) aller Beispiele gleich ist, d.h. $v_i = 1/l, i = 1, \ldots, l$. Diese Gewichtung könnte man dann in den Optimierungsprozess einfließen lassen. Für eine kleine Anzahl von Hypothesen stellt sich das in der Praxis als sehr schwierig heraus.

Berechnet man aber die L_2-Norm des Vektors der Gewichte für die Beispiele der t-ten Hypothese, so ist diese klein, wenn die Beispiele annähernd gleichgewichtet sind und groß, wenn einige Beispiele große Gewichte haben. Somit kann man $||\mathbf{p}^t||_2$ als Maß dafür nehmen, wie unterschiedlich die Gewichtungen der Beispiele für die t-te Hypothese ist.

[6] Ist $|\mathbf{c}| = 1$, so wird das Minimum von $||\mathbf{c}||_2$ für die Gleichverteilung erreicht.

Das Optimierungsproblem kann somit wie folgt geändert werden:

Minimiere

$$J = \mathbf{b}^\top \begin{pmatrix} ||\mathbf{p}^1||_2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & ||\mathbf{p}^T||_2 \end{pmatrix} \mathbf{b} + C \sum_{i=1}^{l} \xi_i \tag{5.20}$$

mit den Nebenbedinungen

$$\begin{aligned} y^{(i)} h^{\mathbf{b}}(\mathbf{x}^{(i)}) &\geq 1 - \xi_i \qquad\qquad i = 1, \dots, l \ . \\ b_t &\geq 0 \end{aligned}$$

Hypothesen, die auf nur einige Beispiele fixiert sind, werden so mehr bestraft als Hypothesen, die alle Beispiele gelernt haben.

QP-AdaBoost vs. Support-Vektor-Algorithmus

Angenommen man würde QP-AdaBoost mit einer sehr einfachen Hypothesenklasse benutzen (z.B. jede Hypothese besteht nur aus einem RBF-Zentrum um ein Trainingsbeispiel) und außerdem seien die Hypothesen reellwertig (kleine Änderung, die auch schon in Freund et al. [52] vorgeschlagen wurde). Dann wird der AdaBoost-Algorithmus Endhypothesen erzeugen, die aus der Linear-Kombination z.B. der RBF-Zentren der wichtigen Beispiele bestehen. Zentren, die mehrfach auftreten, können zusammengefaßt werden. Wahrscheinlich werden alle Beispiele eine Hypothese erzeugen, die in der Nähe der Entscheidunglinie liegen, und somit Kandidaten für Support-Vektoren sind. Benutzt man danach die Quadratische Programmierung zur Maximierung des Margins, so wird auf eine sehr ähnliche Art und Weise wie beim Support-Vector-Algorithmus eine Gewichtung der Hypothesen (und damit der Beispiele) gefunden (vgl. Kernel-Dictionary in Weston et al. [66]). Damit existiert ein enger Zusammenhang zwischen QP-AdaBoost und einer Support-Vector-Maschine. Ähnliche Zusammenhänge findet man zwischen LP-AdaBoost und einer linearen Support-Vektor-Maschine [66, 25]. Die Regularisierung bei der SVM beruht natürlich auf anderen Prinzipien und ist theoretisch fundiert, aber auch $\mathrm{LP}_{\mathrm{reg}}$- und $\mathrm{QP}_{\mathrm{reg}}$-AdaBoost (d.h. mit Slack-Variablen) liefern gute Ergebnisse (siehe experimentellen Vergleich im nächstes Kapitel).

Kapitel 6

Experimenteller Vergleich der Methoden

Um zu zeigen, welche der vorherigen Überlegungen wichtiger und welche weniger wichtig sind, wird im Folgenden ein experimenteller Vergleich der Methoden durchgeführt. Zu den zwei bisher benutzten Datensätzen werden nun noch acht weitere aus dem UCI-Repository bzw. der DELVE-Benchmark-Collection verwendet. Diese Datensätze stellen ein breites Spektrum von Lernproblemen dar: Es gibt Datensätze, für die niedrige und hohe Komplexität notwendig ist, solche auf denen man kleine (Ringnorm, Image-Segment), und solche auf denen man relative hohe Generalisierungsfehler (Titanic, Breast-Cancer[1]) erreicht. Außerdem sind einige höherdimensionale Datensätze dabei. Der Vergleich auf diesen Daten sollte damit relativ aussagekräftig sein.

6.1 Datensätze und Modellselektion

Für den im Abschnitt 6.2 durchgeführten Vergleich der Modifikationen von AdaBoost mit Bagging, dem Einzelklassifikator und einer SVM werden einige Parameter für die Modelle benötigt. Es wird k-Fold-Cross-Validation [2] benutzt, um sämtliche Parameter einzustellen. Um dabei möglichst gut vergleichbare Ergebnisse zu erhalten und um die Varianz der Bestimmung der Generalisierungsfehler so klein wie möglich zu halten, wird k so groß wie möglich (unter Berücksichtigung der verfügbaren Rechenzeit) gewählt. Sind die Parameter einmal eingestellt, so werden die Testfehler auf 5, 20 oder 50 Realisierungen (je nach Größe des Datensatzes) bestimmt.
Um auch eine möglichst große Unabhängigkeit von einer einzelnen Realisierung eines Datensatzes zu erreichen (die Datensätze sind z.T. sehr klein), wird die Modellselektion auf jeweils einem Viertel der Datensätze(nur die Trainingsdaten) durchgeführt, um so die besten Parameter zu erhalten. Dazu wird auf 1, 5 oder 10 Realisierungen der Datensätze (je nach Größe des Datensatzes) eine Modellselektion durchgeführt und dann der Median der bestimmten Parameter gebildet, um so eine robuste Schätzung für die besten Parameter zu erhalten. Das ist sicher kein Weg, den man normalerweise gehen kann (da man nur eine Realisierung zur Verfügung hat), dient aber der statistischen Sicherheit der Ergebnisse.

[1] Der Datensatz Breast-Cancer stammt aus dem *University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia.* Hiermit sei M. Zwitter und M. Soklic für die zur Verfügungstellung der Daten gedankt.

Die folgenden Tabellen 6.1 und 6.2 zeigen die relevanten Informationen zur Modellselektion für die zehn im Vergleich in Abschnitt 6.2 benutzten Datensätze . Die Tabellen unterteilen sich in jeweils acht Teile:

- **Beschreibung des Datensatzes**
  Enthält die relevanten Informationen zum Datensatz selbst, wie z.B. die Herkunft (UCI-Repository oder DELVE-Benchmark-Collection), Dimensionalität der Trainingsdaten, Anzahl der Trainings- und Testbeispiele und das Verhältnis zwischen positiven und negativen Beispielen. Außerdem enthält dieser Abschnitt allgemeine Informationen zur Modellselektion und zur Bestimmung des Testfehlers wie z.B. die Anzahl der Datensätze, über die der Testfehler (in Tabelle 6.4) gemittelt und Anzahl der Trainingsdatensätze, mit denen die Modellselektion durchgeführt wird.

- **Modellselektion für RBF-Netze**
  Für ein RBF-Netz sind zwei Parameter einzustellen, die Anzahl der Zentren und die der Iterationen für die Nachadaption (vgl. A.1). Die ersten beiden Einträge in den Tabellen geben das Ergebnis der Modellselektion aus der im dritten Eintrag angegebenen Anzahl von möglichen Modellen an. Dabei wird ein Vortest gemacht, um die Anzahl der zu betrachtenden Modelle zu reduzieren. Bei der Anzahl der Zentren werden i.A. Modelle mit 2-20 Zentren und 0-10 Iterationen betrachtet.

- **Bagging und AdaBoost**
  Für Bagging und AdaBoost ist die Anzahl der Iterationen festzulegen. Hier wird eine Anzahl für alle Datensätze festgelegt, mit dem i.A. gute Erfahrungen gemacht wurde: 200. Davon wurde nur in einigen wenigen Fällen abgewichen, um den Rechenaufwand in erträglichen Grenzen zu halten. Als Basishypothese wird in allen Fällen das beste RBF-Modell benutzt.

- **Modellselektion für RBF-SVM**
  Für eine RBF-SVM sind ebenfalls zwei Parameter einzustellen, die Weite des Kernels (σ) und die Regularisierungskonstante (C) (vgl. [64]). Die ersten beiden Einträge in den Tabellen geben das Ergebnis der Modellselektion aus der im dritten Eintrag angegebenen Anzahl von möglichen Modellen an. Zuvor wird ein Vortest gemacht, um die Anzahl der zu betrachtenden Modelle zu reduzieren. Für σ wird meist im Bereich $0.1 - 200$ und für C im Bereich $10^{-6} - 10^{6}$ nach optimalen Werten gesucht.

- **Bestimmung der besten Anzahl der Iterationen für $\mathbf{AdaBoost_{es}}$**
  Für $\text{AdaBoost}_{\text{es}}$ ist die beste Anzahl der Iterationen durch Cross-Validierung zu bestimmen. Dabei wird eine maximale Anzahl von Iterationen (meist die unter ,,Bagging und AdaBoost" angegebene) festgesetzt und dann auf den Validierungs-Daten die beste Anzahl bestimmt. Der erste Eintrag in diesem Teil gibt das Ergebnis der Modellselektion an.

- **Modellselektion für $\mathbf{AdaBoost_{reg}}$, $\mathbf{QP_{reg}}$-AdaBoost, $\mathbf{LP_{reg}}$-AdaBoost**
  Für diese drei Algorithmen ist eine Regularisierungskonstante zu bestimmen. Jeweils im ersten Eintrag in den Tabellen findet man das Ergebnis der Modellselektion. Der zweite Eintrag gibt die Anzahl der betrachteten Regularisierungskonstanten an. Die Anzahl der möglichen Konstanten wird durch Vortests reduziert. Für QP_{reg}- und LP_{reg}-AdaBoost wird im Bereich von $10^{-6} - 10^{3}$ nach guten Werten gesucht; für $\text{AdaBoost}_{\text{reg}}$ im Bereich von $10^{-2} - 10^{6}$.

		Banana 16%	Breast-Cancer	Image-Segment	Ring-norm	FSonar
Beschreibung						
Herkunft		A.3.1	UCI	DELVE	UCI	UCI
Dimensionalität	$a =$	2	9	18	20	9
Trainingsbeispiele	$l =$	400	200	1300	400	666
Testbeispiele		4900	70	1010	7000	400
Anzahl pos. Beispiele	in %	44.8	29.2	57.1	49.5	55.3
Anzahl Datensätze		20	50	5	20	50
k-fold-CV	$k =$	10	5	3	10	10
Anzahl CV-Datensätze		5	10	1	5	10
RBF-Netz						
Anzahl Zentren	$K =$	18	5	95	12	4
Anzahl Iterationen		3	4	10	3	3
mögl. Kombinationen		30	30	33	30	35
RBF-SVM						
Kernel-Weite	$\sigma =$	1	50	3	10	0.1
Regularisierung	$C =$	$3.2 \cdot 10^{-3}$	0.66	0.1	10^{-5}	40
mögl. Kombinationen		65	80	63	48	63
Bagging & AdaBoost						
Anzahl Iterationen	$T =$	200	200	50	200	200
AdaBoost$_{\mathbf{es}}$						
Anzahl Iterationen	$T_{\mathrm{es}} =$	12	4	28	7	45
von maximal	$T =$	200	200	50	200	200
AdaBoost$_{\mathbf{reg1}}$						
Regularisierung	$C =$	100	31.6	10	3.2	100
mögl. Kombinationen		12	12	12	12	15
AdaBoost$_{\mathbf{reg2}}$						
Regularisierung	$C =$	$3.2 \cdot 10^{4}$	10^{5}	10	$3.2 \cdot 10^{4}$	10^{5}
mögl. Kombinationen		12	12	12	12	12
LP$_{\mathbf{reg}}$-AdaBoost						
Regularisierung	$C =$	$3.2 \cdot 10^{-4}$	10^{-5}	10	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
mögl. Kombinationen		12	12	12	12	12
QP$_{\mathbf{reg}}$-AdaBoost						
Regularisierung	$C =$	10^{-4}	10^{-4}	10^{-3}	$3.2 \cdot 10^{-5}$	10^{-2}
mögl. Kombinationen		12	15	15	15	15

Tabelle 6.1: Informationen, Modellselektion und Festlegung einiger Paramater für die Datensätze Banana 16%, Breast Cancer, Image-Segment, Ringnorm und FSonar. Siehe Text.

		Splice	Thyroid	Titanic	Two-norm	Wave-form
Beschreibung						
Herkunft		DELVE	UCI	DELVE	UCI	UCI
Dimensionalität	$a =$	60	5	3	20	21
Trainingsbeispiel	$l =$	1000	140	150	400	400
Testbeispiele		2174	70	2051	7000	4600
Anzahl pos. Beispiele	in %	48.1	30.2	32.3	50.0	32.9
Anzahl Datensätze		20	50	50	20	20
k-fold-CV	$k =$	5	5	10	10	10
Anzahl CV-Datensätze		2	10	10	5	5
RBF-Netz						
Anzahl Zentren	$K =$	6	8	4	12	10
Anzahl Iterationen		10	5	1	3	0
mögl. Kombinationen		20	32	16	30	20
RBF-SVM						
Kernel-Weite	$\sigma =$	70	2	2	40	1
Regularisierung	$C =$	10^{-4}	1	10^{-5}	0.32	20
mögl. Kombinationen		63	63	36	45	63
Bagging & AdaBoost						
Anzahl Iterationen	$T =$	200	200	200	200	200
AdaBoost$_{\mathbf{es}}$						
Anzahl Iterationen	$T_{\text{es}} =$	69	6	13	7	8
von maximal	$T =$	200	200	200	200	200
AdaBoost$_{\mathbf{reg1}}$						
Regularisierung	$C =$	10	1	3.2	10^3	32
mögl. Kombinationen		9	15	15	9	15
AdaBoost$_{\mathbf{reg2}}$						
Regularisierung	$C =$	10^4	6.5	10^{-3}	$3.2 \cdot 10^3$	10^6
mögl. Kombinationen		15	15	15	9	15
LP$_{\mathbf{reg}}$-AdaBoost						
Regularisierung	$C =$	10^{-3}	0.002	0.1	0.05	10^{-5}
mögl. Kombinationen		12	12		12	12
QP$_{\mathbf{reg}}$-AdaBoost						
Regularisierung	$C =$	10^{-3}	0.01	10^{-3}	10^{-6}	$3.2 \cdot 10^{-4}$
mögl. Kombinationen		12	15	23	15	12

Tabelle 6.2: Informationen, Modellselektion und Festlegung einiger Paramater für die Datensätze Splice, Thyroid, Titanic, Two-Norm, Waveform. Siehe Text.

6.2 Simulationen

6.2.1 Experimentelle Ergebnisse

In Tabelle 6.3 findet man eine Auflistung und die Spezifikation der Methoden, die auf den zehn Datensätzen verglichen werden. Zu den hier betrachteten Methoden werden auch die Ergebnisse des Support-Vektor-Algorithmus angegeben, um eine gute Referenz zu haben. Die z.T. für diese Algorithmen benötigten Parameter werden durch k-fold-Cross-Validation für jeden Datensatz bestimmt bzw. *a priori* festgelegt (vgl. 6.1). Um Rechenzeit zu sparen gibt man sich dabei auch mit mehr oder weniger guten Approximationen zu den optimalen Parametern zufriedengegeben. Dadurch kann es zu einer leichten Verfälschung der Ergebnisse kommen, da mögliche Schwierigkeiten bei der Bestimmung oder hohe Sensibilitäten in diesen Parametern die Ergebnisse beeinflußen. Die Testfehler in den beiden Tabellen sind über 5, 20 oder 50 Durchgänge (je nach Größe des Datensatzes) gemittelt und die Standardabweichungen angegeben. Nicht in allen Fällen findet man statistitisch signifikan-

No.	Name form	Kurz-	Beschreibung
1.	RBF-Netz	RBF	einzelnes RBF-Netz mit K Zentren und Nachadaption der Zentren (vgl. Anhang A.1)
2.	Bagging	BAG	Bagging (vgl. Abschnitt 2.1) mit T Iterationen und RBF-Netzen als Basishypothesen (vgl. Abschnitt 2.1)
3.	AdaBoost	AB	AdaBoost mit T Iterationen und RBF-Netzen als Basishypothesen (vgl. Abschnitt 2.2)
4.	$\text{AdaBoost}_{\text{es}}$	ABES	AdaBoost mit Early Stopping mit RBF-Netzen
5.	SVM mit RBF-Kern	RSVM	Support-Vektor-Maschine mit RBF-Kern mit Weite σ und Regularisierung C (vgl. [14, 3, 64])
6.	LP-AdaBoost	LP-AB	AdaBoost mit T Iterationen und Optimierung der Hypothesenkoeffizienten durch lineares Programm (vgl. [27])
7.	QP-AdaBoost	QP-AB	AdaBoost mit T Iterationen und Optimierung der Hypothesenkoeffizienten durch quadratisches Programm (vgl. Abschnitt 5.4 mit $C = \infty$, d.h. ohne Slack-Variablen)
8.	LP_{reg}-AdaBoost	LPR-AB	AdaBoost mit T Iterationen und Optimierung der Hypothesenkoeffizienten durch lineares Programm mit Slack-Variablen (vgl. Abschnitt 5.3.2)
9.	QP_{reg}-AdaBoost	QPR-AB	AdaBoost mit T Iterationen und Optimierung der Hypothesenkoeffizienten durch quadratisches Programm mit Slack-Variablen (vgl. Abschnitt 5.4)
10.	$\text{AdaBoost}_{\text{reg1}}$	ABR1	AdaBoost mit (linear) veränderter Kostenfunktion (vgl. Abschnitt 5.2 und (5.3.a))
11.	$\text{AdaBoost}_{\text{reg2}}$	ABR2	AdaBoost mit (quadratisch) veränderter Kostenfunktion (vgl. Abschnitt 5.2 und (5.3.b))

Tabelle 6.3: Vereinbarung von Namen und Kurzbeschreibung der im Vergleich benutzten Algorithmen

Datensatz	RBF	BAG	AB	ABES	RSVM
Banana 16%	10.9±0.5	10.8±0.4	12.3±0.7	11.7±0.6	11.5±0.7
Breast Cancer	28.7±5.3	26.8±4.0	30.5±4.5	27.0±5.3	26.1±4.8
Image Segment	2.8±0.7	2.6±0.5	2.5±0.7	2.5±0.6	2.9±0.7
Ringnorm	1.7±0.3	1.5±0.1	2.0±0.2	2.1±0.4	1.7±0.1
FSonar	34.6±2.1	34.2±1.8	35.6±1.9	35.1±1.7	32.5±1.7
Splice	10.0±0.8	9.3±0.6	10.1±0.5	10.1±0.6	10.9±0.7
Thyroid	4.8±2.4	4.2±2.3	4.4±1.9	4.5±2.3	4.8±2.2
Titanic	23.4±1.7	23.3±1.6	22.7±1.2	23.1±1.0	22.4±1.0
Twonorm	2.8±0.2	2.7±0.1	3.1±0.3	3.1±0.3	3.0±0.2
Waveform	10.7±1.0	9.8±0.8	10.8±0.4	11.0±0.7	9.8±0.3

Tabelle 6.4: Experimenteller Vergleich der Methoden: Angegeben sind die Klassifikationsfehler (in %) auf dem Testdatensatz für den Einzelklassifikator, Bagging, AdaBoost, $\text{AdaBoost}_{\text{es}}$ und eine SVM mit RBF-Kernel jeweils mit nahezu optimal eingestellten Modellparametern.

Datensatz	LP-AB	QP-AB	LPR-AB	QPR-AB	ABR1	ABR2
Banana 16%	13.2±0.9	13.0±0.8	10.8±0.4	10.9±0.5	11.0±0.5	10.7±0.5
Breast Cancer	30.9±5.2	32.7±5.0	31.0±4.2	26.2±4.7	25.3±4.3	26.3±4.3
Image Segment	2.6±0.6	2.5±0.7	2.6±0.6	2.4±0.5	2.5±0.6	2.5±0.7
Ringnorm	2.2±0.3	2.0±0.2	2.2±0.4	1.9±0.2	1.9±0.3	1.7±0.2
FSonar	39.6±5.5	36.7±2.3	35.7±4.5	36.2±1.7	34.2±1.8	33.6±1.7
Splice	10.2±0.5	10.1±0.6	10.2±1.6	10.1±0.5	9.5±0.6	9.5±0.7
Thyroid	4.4±1.9	4.5±2.3	4.4±2.0	4.4±2.2	4.5±2.3	4.4±2.1
Titanic	30.9±8.6	23.5±4.5	22.9±1.9	22.7±1.0	22.7±1.6	22.5±1.1
Twonorm	3.6±0.3	3.5±0.3	3.4±0.6	3.0±0.3	2.7±0.1	2.7±0.2
Waveform	12.2±0.7	11.9±0.6	10.6±1.0	10.1±0.5	10.6±0.6	9.9±0.9

Tabelle 6.5: Experimenteller Vergleich der Methoden (Fortsetzung von Tabelle 6.5): Klassifikationsfehler (in %) auf dem Testdatensatz für LP-, QP-, LP_{reg}- und QP_{reg}-AdaBoost, sowie $\text{AdaBoost}_{\text{reg1}}$ und $\text{AdaBoost}_{\text{reg2}}$ mit jeweils nahezu optimal eingestellten Modellparametern.

te Unterschiede zwischen den Methoden. Als Auswertung der Tabellen zeigt Abbildung 6.8 wie oft die einzelnen Methoden auf einem Datensatz das *beste* Ergebnis von allen Methoden hat. Diese Aussage ist natürlich schwächer, faßt jedoch die Ergebnisse übersichtlich zusammen. Für diese wird eine Menge von Klassifikatoren auf jeweils dem gleichen Datensatz erzeugt und dann bestimmt, welcher am besten ist, d.h. den kleinsten Testfehler hat. Die Wahrscheinlichkeit dafür wird auf der Y-Achse aufgetragen. In Abbildung 6.8 sind die Säulen zu zehn Gruppen zusammengefaßt; für jeden Datensatz eine Gruppe. Innerhalb der Gruppe findet man dann die Gewinnwahrscheinlichkeiten der Algorithmen für diesen Datensatz in der Reihenfolge: RBF-Netz, Bagging, AdaBoost, $\text{AdaBoost}_{\text{es}}$, RBF-SVM, L/QP-AdaBoost L/QP_{reg}-AdaBoost, $\text{AdaBoost}_{\text{reg1,2}}$. Darunter ist der gleiche Sachverhalt auf eine andere Art und Weise dargestellt, die besser die Verhältnisse zwischen den Wahrscheinlichkeiten darstellt.

6.2.2 Diskussion der Ergebnisse

Einzelklassifikator vs. Bagging

Vergleicht man die Generalisierungsfehler der Einzelhypothesen mit denen von Bagging, so stellt man fest, daß diese für Bagging in (fast) jedem Fall geringer sind. Das ist auf die in Abschnitt 3.1.3 herausgestellte Varianzverminderung durch Bagging zurückzuführen. Außerdem liefert Bagging im Vergleich zu den anderen Methoden in vielen Fällen die kleinsten Generalisierungsfehler. Das ist sicher auch auf den Umstand zurückzuführen, daß Basishypothesen gewählt wurden, die für dieses Problem passende Komplexitäten haben. Hätte man zu schwache Basishypothesen gewählt, so wären die Ergebnisse von Bagging sicher schlechter. Würde man stärkere Basishypothesen benutzten, so bräuchte man evtl. mehr Iterationen im Bagging-Prozeß, um genügend zu mitteln, so daß sich die Varianz verringert (vgl. dazu z.B. Tabellen 3.2 und 4.1). Ein weiterer Grund dafür, daß Bagging in diesen Experimenten gute Ergebnisse liefert, ist, daß die Basishypothese in vielen Fällen schon sehr gut generalisiert (vgl. Ergebnisse für die Einzelhypothesen in Tabelle 6.4). Benutzt man Basishypothesen, die höhere Generalisierungsfehler haben und die auch nicht durch Modellselektion verbesserbar sind, so wird man Schwierigkeiten haben, solch gute Ergebnisse zu erzielen.
Wertend kann man damit sagen, daß Bagging sehr gute Ergebnisse liefert kann, wenn die Basishypothesen geeignet sind.

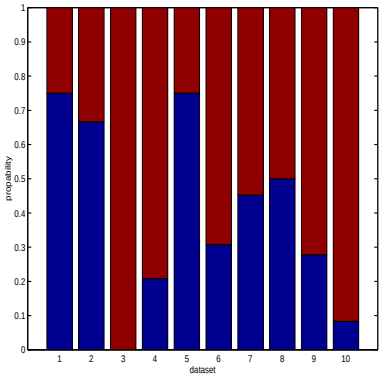


Abbildung 6.1: Vergleich der Gewinnraten RBF-Netz – Bagging

Early Stopping

In Abschnitt 4.1.2 wurde bereits experimentell gezeigt, daß man bei AdaBoost Overfitting in der Anzahl der Iterationen beobachten kann. Das wird durch die hier präsentierten Ergebnisse bestätigt. In sehr vielen Fällen brachte das Stoppen nach einigen wenigen Iterationen (meist schon nach 10; vgl. Tabellen 6.1 und 6.2) eine Verbesserung des Ergebnisses. In den restlichen Fällen (außer beim Datensatz Titanic) sind die Unterschiede nicht signifikant. Hier zeigt sich auch, daß das Problem der Modellselektion, d.h. des Findens einer guten Anzahl von Iterationen schwierig ist. Würde man bessere Methoden benutzen, als die hier verwendete, so müßte man besssere Ergebnisse erhalten. In Abbildung 6.3 findet man die Gewinnwahrscheinlichkeiten für AdaBoost und $\text{AdaBoost}_{\text{es}}$. Das vorzeitige Stoppen liefert aber selten signifikant bessere Ergebnisse als die regularisierten Versionen LP_{reg}-, QP_{reg}-AdaBoost und $\text{AdaBoost}_{\text{reg}1,2}$. Man könnte daraus schließen, daß AdaBoost nicht die richtige Fehlerfunktion minimiert und dies als Bestätigung dafür nehmen, daß die Minimierung der kleinsten Stabilität (Schaffung eines harten Margins) bei Vorhandensein von Rauschen in den Daten kein gutes Konzept ist.

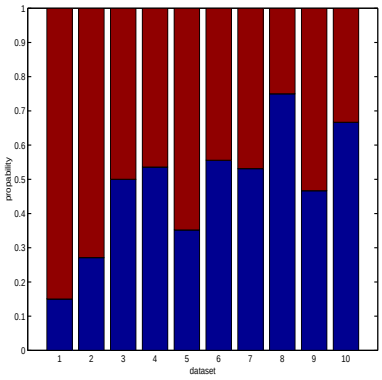


Abbildung 6.2: Vergleich der Gewinnwahrscheinlichkeiten für AdaBoost, $\text{AdaBoost}_{\text{es}}$, LP-AdaBoost, QP-AdaBoost

LP- vs. QP-AdaBoost

Der bestimmte durchschnittliche Generalisierungsfehler von LP-AdaBoost ist in fast allen Fällen größer als der von QP-AdaBoost. Aber nur in drei Fällen statistisch signifikant (Dazu wurde ein t-Test für die Generalisierungsfehler unter Berücksichtigung der Varianzen und der Anzahl der Messungen durchgeführt. Statistisch signifikant heißt hier, daß die Null-Hypothese weniger als 15% wahrscheinlich ist.)
Somit ist der Unterschied zwischen diesen Algorithmen klein, aber vorhanden. Das bestätigt die in Abschnitt 5.3.1 gemachten Beobachtungen, daß eine untere oder obere Schranke für die Gewichte der Hypothesen keine signifikanten Verbesserungen bringt.
Die dennoch vorhandenen Unterschiede lassen sich möglicherweise mit den Varianzbetrachtungen in Abschnitt 3.1.3 erklären. Mehr Hypothesen, die ein von Null verschiedenes Gewicht haben, führen zu mehr Mittelung und man ist nicht so stark abhängig von der Varianz der Basishypothesen. Bei QP-AdaBoost werden i.d.R. dichtbesetzte Hypothesenkoeffizienten erzeugt, da die L_2-Norm des Gewichtsvektors dadurch kleiner wird. Der Effekt ist dann deswegen nicht so groß, weil die Varianz der RBF-Netze nicht sehr hoch ist.

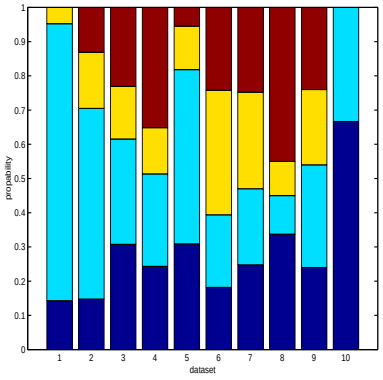


Abbildung 6.3: Vergleich der Gewinnwahrscheinlichkeiten für AdaBoost, AdaBoost$_{\text{es}}$, LP-AdaBoost, QP-AdaBoost

Offensichtlich aus diesen Experimenten ist auch, daß die nachträgliche Maximierung der kleinsten Stabilität keine guten Ergebnisse liefert. LP- und QP-AdaBoost verschlechtern die Ergebnisse von AdaBoost in vielen Fällen signifikant. AdaBoost bzw. AdaBoost mit Early Stopping liefert häufig bessere Ergebnisse (vgl. Abbildung 6.3) .

AdaBoost$_{\text{reg1,2}}$ vs. L/QP$_{\text{reg}}$-AdaBoost

Abbildung 6.4 zeigt die Wahrscheinlichkeiten für die regularisierten Versionen von AdaBoost. Die Modifikationen der Fehlerfunktion (AdaBoost$_{\text{reg1,2}}$) arbeiten in vielen Fällen besser als die nachträgliche Änderung der Hypothesenkoeffizienten der durch AdaBoost erzeugten Hypothesen.
Das zeigt zum einen, daß diese Optimierungsverfahren noch nicht optimal gestellt sind und noch verfeinert werden müssen (z.B. bessere Berücksichtigung der Beispielsgewichtungen), und zum anderen daß die von AdaBoost erzeugt Hypothesen nicht optimal sind. AdaBoost erzeugt Hypothesen, die zum Erreichen eines ,,harten" Margins sinnvoll sind. Diese sind nicht notwendigerweise brauchbar für die Klassifikation mit einem ,,weichem" Margin. Ein direktes Einbringen der linearen bzw. quadratischen Programmierung, wie z.B. für LP-AdaBoost in Grove et al. [27] gezeigt, könnte zu deutlichen Verbesserungen führen.
Die Ergebnisse von AdaBoost$_{\text{reg1}}$ sind oft ein wenig besser als AdaBoost$_{\text{reg2}}$.

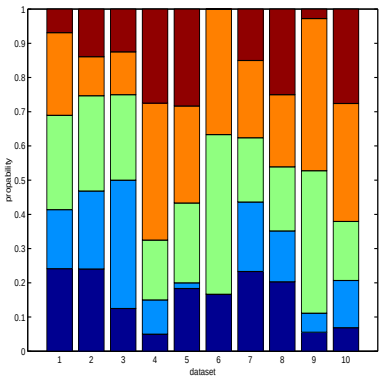


Abbildung 6.4: Vergleich der Gewinnwahrscheinlichkeiten für LPreg-, QPreg-AdaBoost, AdaBoost$_{\text{reg1,2}}$ und eine SVM.

Man kann vermuten, daß durch den quadratische Term (5.3.b) die Unterschiede zwischen den leichten und schwierigen Beispielen mehr kontrastiert wird als bei (5.3.a). Bei $\text{AdaBoost}_{\text{reg1}}$ werden die Gewichte von mehr Beispielen klein, als notwendig. Jede dieser Modifikationen der Zielfunktion (linear, quadratisch o.ä) implizieren spezielle Eigenschaften der resultierenden Margin-Verteilung, die im noch einzelnen untersucht und gedeutet werden mußsen. Der optimale Term (linear, quadratisch o.ä.) hängt wahrscheinlich von der Beschaffenheit der Daten und vom Problem ab. Im Vergleich zu den Ergebnissen des Support-Vektor-Algorithmus zeigen die Algorithmen $\text{AdaBoost}_{\text{reg1}}$ und $\text{AdaBoost}_{\text{reg2}}$ eine bessere durchschnittliche Leistung (vgl. Abbildung 6.5 und 6.7). Die Leistungsfähigkeit der Support-Vektor-Maschinen ist möglicherweise auf höherdimensionalen Daten besser, aber auf den hier benutzten Datensätzen (bis 60 dimensional) sind die Resultate eher durchnittlich (vgl. 6.5 und 6.7).

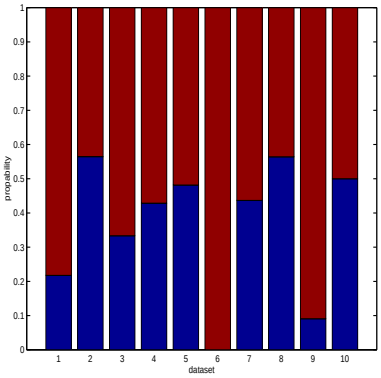


Abbildung 6.5: Direkter Vergleich der Gewinnwahrscheinlichkeiten für $\text{AdaBoost}_{\text{reg2}}$ mit einer SVM.

AdaBoost vs. $\text{AdaBoost}_{\text{reg1,2}}$

Die regularisierten Versionen von AdaBoost zeigen deutlich bessere Ergebnisse als der originale Algorithmus. Das wird durch die Abbildung 6.6 illustriert. Für viele Fälle sind die Ergebnisse von AdaBoost schlechter, als jene der beiden Modifikationen.
Die benutzten Datensätze haben zum Teil relativ hohe erreichbare Generalisierungsfehler. Wenn AdaBoost versucht, für diese Datensätze den Trainingsfehler zu Null zu machen, dann können dabei leicht Entscheidunglinien entstehen, die nicht gut generalisieren (vgl. Abschnitt 4.2).
Mit der Veränderung der Zielfunktion wurde ein Eingriff in den AdaBoost-Lernprozeß vorgenommen, der eine gewisse Unabhängigkeit von dem in den Trainingsdaten vorhandenen Rauschen impliziert (vgl. z.B. Abbildung 4.2). In den Experimenten hat sich auch gezeigt, daß man durch eine geeignete Wahl der Regularisierungskonstanten die Notwendigkeit des *Early Stopping* umgehen kann (das entspricht der bekannten Tatsache, daß man durch Early-Stopping regularisieren kann.). AdaBoost konvergiert dann auch asymptotisch gegen eine Lösung, die niedrige Generalisierungsfehler liefert.

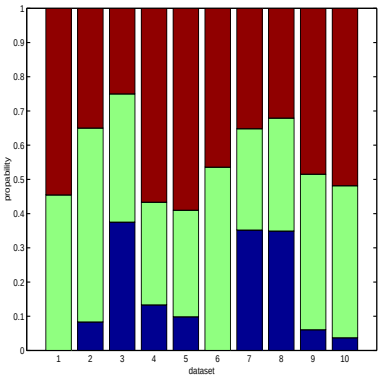


Abbildung 6.6: Vergleich der Gewinnwahrscheinlichkeiten für AdaBoost, $\text{AdaBoost}_{\text{reg1}}$ und $\text{AdaBoost}_{\text{reg2}}$.

Zusammenfassung

Mit dem durchgeführten experimentellem Vergleich der Methoden wurden einige Vermutungen und Beobachtungen der letzten Abschnitte bestätigt. Ein wenig überraschend sind die meist sehr guten Ergebnisse von Bagging mit RBF-Netzen. Das ist unter anderem auf die speziellen experimentellen Bedingungen zurückzuführen.

Gute Ergebnisse liefert die beiden Algorithmen zur nachträglichen Optimierung der Hypothesenkoeffizienten. Mit der Erweiterung des linearen bzw. quadratischen Programms konnten sich die in Grove et al. [27] herausgestellten Probleme teilweise behoben werden. Die Algorithmen zeigen nun ein deutlich besseres durchschnittliches Verhalten als AdaBoost. Sehr gute Ergebnisse liefern die beiden Modifikationen der Zielfunktion von AdaBoost. Die durchschnittlichen Gewinnwahrscheinlichkeiten auf allen Datensätzen sind für diese beiden Algorithmen am höchsten (vgl. Tabelle 6.7 links oben). Analysiert man die drei anderen Diagramme in Abbildung 6.7, so kann man eine Aussage zur Stabiltät der Algorithmen erhalten. Die Algorithmen AdaBoost$_{\text{reg}1,2}$ zeigen am häufigsten die besten Ergebnisse. Außerdem sind die Ergebnisse häufig im 50er und 75er Percentil gut. Diese Algorithmen scheinen stabil arbeiten, da auch das 25er Percentil (vgl. Bildunterschrift) für diese Algorithmen am kleinsten ist. Anders ist das bei L/QP$_{\text{reg}}$-AdaBoost: Diese Algorithmen zeigen relativ häufig sehr gute Ergebnisse, aber auch nicht sehr selten sehr schlechte Ergebnisse (25er Percentil ist hoch). Das ist insbesondere bei LP$_{\text{reg}}$-AdaBoost so.
Nach L/QP-AdaBoost schneidet in diesem Vergleich der originale Algorithmus (mit und ohne Early-Stopping) am schlechtesten ab. Das zeigt, daß die eingeführten Modifikationen von AdaBoost sinnvoll sind. *AdaBoost konnte deutlich verbessert werden und zeigt nun häufig bessere Generalisierungsfähigkeit als der Support-Vektor-Algorithmus.*

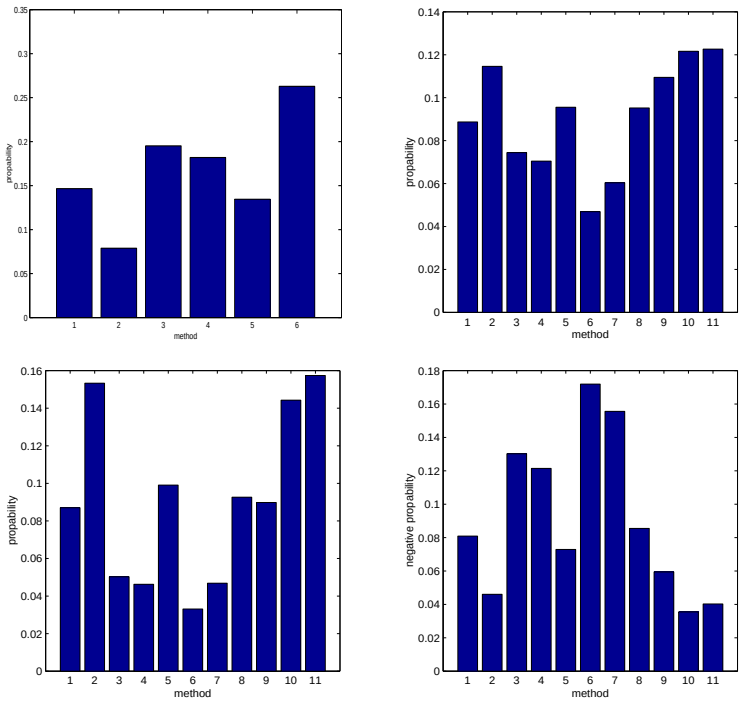


Abbildung 6.7: Links oben ist Gewinnwahrscheinlichkeiten der elf Methoden (wie in Tabelle 6.3) für alle Datensätze und rechts oben die Wahrscheinlichkeit, daß die jeweilige Methode besser als die Hälfte der anderen Methoden ist (Median). Links unten sind die Wahrscheinlichkeiten angegeben, daß die Methode *besser* als 75% und rechts unten daß sie *schlechter* als 75% aller Methoden ist.

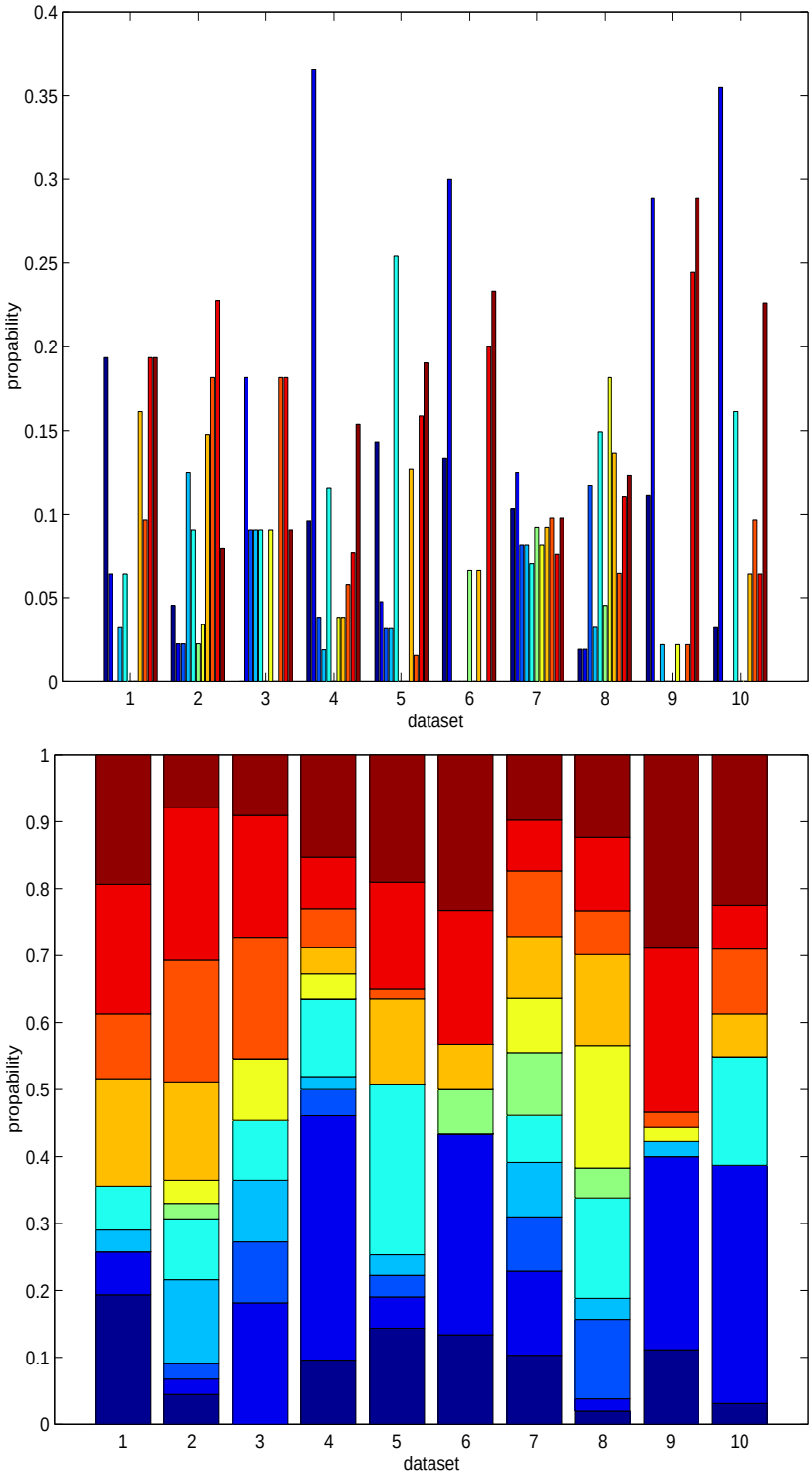


Abbildung 6.8: Dargestellt sind die Gewinnraten der einzelnen Algorithmen in der Reihenfolge RBF (dunkelblau), BAG (blau), AB(himmelblau), ABES (seeblau), RSVM (türkis), LP-AB (grün), QP-AB (sonnengelb), LPR-AB (gelb), QPR-AB (orange), ABR1 (rot), ABR2 (braun) für die zehn Datensätze (gleiche Reihenfolgen wie in Tabellen 6.4 und 6.5). Der gleiche Sachverhalt wird auf zwei verschiedene Arten dargestell. Erklärung siehe Text.

Kapitel 7

Zusammenfassung

Ensemble-Lernmethoden liefern i.A. gute Ergebnisse im Vergleich zu den Einzelklassifikatoren und zu anderen Lernalgorithmen. Die Varianz der Ausgabe des Lernalgorithmus $\mathcal{L}$ wird durch die Kombination vieler Hypothesen verringert und man erhält bessere Generalisierungsfehler. Das wurde von Breiman [7] theoretisch und experimentell in den Abschnitten 3.1.3 und 6.2 (vgl. Tabelle 6.4 für Einzelklassifikator – Bagging) gezeigt. Außerdem wurde gezeigt, daß durch ein Ensemble eine gewisse Unabhängigkeit von den Eigenschaften des Basislernalgorithmus geschaffen wird (Abschnitt 3.1.2).

AdaBoost ist eine spezielle Ensemble-Lernmethode, die zusätzlich zu den Vorteilen, die ein Ensemble von Klassifikatoren mit sich bringt, noch die (meist) positive Eigenschaft hat, daß sie eine Fehlerfunktion minimiert, die den Margin maximiert. Diese Eigenschaft ist sehr wichtig und führt (unter bestimmten Umständen bzw. mit gewissen Veränderungen) zu guten Ergebnissen (vgl. Support-Vektor-Algorithmus und Maximum-Margin-Klassifikation). Es wurde ein Zusammenhang zwischen der Optimierung bei AdaBoost und einem Annealingprozeß gefunden, bei dem die L_1-Norm des Hypothesengewichtsvektors $\mathbf{b}$ als die Temperatur des Annealingprozesses aufgefaßt werden kann. Daraus wurde abgeleitet, daß AdaBoost asymptotisch der approximativen Lösung eines Problems der Linearen Programmierung entspricht. Zusammenhänge zwischen AdaBoost und der Maximum-Margin-Klassifikation wurden erläutert und festgestellt, daß man ähnlich wie beim Support-Vektor-Algorithmus Beispiele gleicher Stabilität findet (**S**-Beispiele).

Außerdem wurde gezeigt, daß man AdaBoost als ein Gradientenverfahren auffassen kann, das asymptotisch das Funktional $\widetilde{G}$ minimiert. Die Minimierung dieses Funktionals impliziert einen möglichst großen Margin, d.h. eine möglichst große kleinste Stabilität ρ der Trainingsbeispiele. Die Konvergenz zu einem möglichst großen ρ wird durch einen Annealingprozeß unterstützt. Die Güte der Lösung hängt von der Geschwindigkeit des Annealings ab. Um das zu zeigen, wurde der Einfluß des Parameters ϕ bei den AdaBoost-ähnlichen Algorithmen untersucht und festgestellt, daß dieser Parameter u.a. die Geschwindigkeit des Annealings steuert. Als Ergebnis konnte eine untere Schranke für ρ bewiesen werden.

Es hat sich auch gezeigt, daß die direkte Lösung des linearen Programmes für den Fall mit Rauschen keine guten Ergebnisse liefert. Das wurde anhand der Stabilitätsverteilungen der Trainingsbeispiele erklärt und in Experimenten bestätigt. Damit ist auch gezeigt, daß AdaBoost asymptotisch, d.h. bei einer hohen Anzahl von Iterationen, schlechte Ergebnisse liefern kann und *nicht robust* gegen die Verrauschung der Trainingsdaten ist.

Um AdaBoost robuster gegen Rauschen in den Trainingsdaten zu machen, wurde das dem Algorithmus zugrundeliegende lineare Optimierungsproblem verändert. Dadurch wurde der Einfluß eines Beispiels auf die Entscheidungslinie begrenzt und ein Gleichgewicht zwischen der Stabilität und der Wichtigkeit eines Beispiels im AdaBoost-Lernprozeß geschaffen. Es wurde damit der Schritt vom *harten Margin* zum *weichen Margin* gegangen. Diese Veränderung des Algorithmus liefert oft signifikant verringerte Testfehler als der originale AdaBoost-Algorithmus und bessere durchschnittliche Ergebnisse als der Support-Vektor-Algorithmus. Das wurde durch aussagekräftige Simulationen gezeigt.

Im letzten Teil der Arbeit wurde auf die nachträgliche Verbesserung eines vorhandenen Ensembles durch Optimierung der Hypothesenkoeffizienten eingegangen. Die beschriebenen Probleme der Margin-Maximierung konnten durch geeignete Nebenbedingungen und Umformulierungen der Optimierungsaufgaben gemildert werden; z.B. Ergebnisse von Grove et al. [27] mit LP-AdaBoost wurden dadurch deutlich verbessert. Die experimentellen Ergebnisse in Abschnitt 6.2 zu den verschiedenen Veränderungen von LP-AdaBoost (Beschränkung der Gewichte) bestätigten die angestellten theoretischen Überlegungen und gefundenen Ergebnisse.

Diese Arbeit ist der Ausgangspunkt für weitere Forschung zum besseren Verständis von Ensemble-Lernmethoden. Es ist u.a. nach weiteren Zusammenhängen zwischen dem Support-Vektor-Lernen und den Ensemblen-Lernmethoden zu suchen. Außerdem ist es sehr wichtig, die hier vorgestellten Heuristiken weiter theoretisch zu analysieren. Es muß noch besser verstanden werden, wie optimale Margin-Verteilungen auszusehen haben und welche Margin-Verteilungen die verschiedenen Algorithmen implizieren. Eine interessante Frage ist welche Eigeschaften die Algorithmen haben, bei denen man die lineare oder quadratische Programmierung direkt bei der Erzeugung der Hypothesen, d.h. im AdaBoost-Prozeß benutzt (vgl. Grove et al. [27]).

Anhang

A.1 Klassifikation mit RBF-Netzen

In den folgenden Abschnitten wird kurz in die Thematik der *Radiale Basisfunktions-Netze* (*RBF-Netze*) eingeführt. Im Abschnitt A.1.1 werden zunächst traditionelle, d. h. *lineare* RBF-Netze beschrieben. Im Abschnitt A.1.2 werden diese dann um die *Nachadaption der Zentren* erweitert.
In vielen Beispielen hat sich gezeigt, daß RBF-Netze mit Nachadaptierung deutlich bessere Ergebnisse als lineare Netze liefern. Mit dieser Erweiterung können sich RBF-Netze mit den besten zur Zeit bekannten Lernalgorithmen messen (Müller et al. [46]).
Für die Benutzung der RBF-Netze beim Boosting kann eine *gewichtete Fehlerfunktion* (Trainingsbeispiele haben unterschiedliche Gewichtungen) benutzt werden[1]. Um nicht alle Ableitungen und Betrachtungen für beide Fälle (ungewichtete und gewichtete Fehlerfunktion) wiederholen zu müssen, wird immer die gewichtete Fehlerfunktion angenommen. Für den ungewichteten Fall vereinfachen sich dann alle Formeln.
Dieses Kapitel soll nur einen kleinen Ausschnitt aus der Thematik der RBF-Netze darstellen, gerade so viel, daß man die Funktionsweise der Netze versteht und sie ggf. implementieren könnte. Weiterführende Betrachtungen findet man in Bishop [2], Broomhead et al. [11], Duda et al. [19], Haykin [29], Hertz et al. [30] und Orr [48].

A.1.1 Lineare RBF-Netze

Architektur

Ein RBF-Netz hat eine sehr einfache Struktur. Es besteht aus drei Schichten: der *Eingabe-, Hidden- und Ausgabe-Schicht*. Abbildung A.1 zeigt die Architektur solcher Netze.
Die Eingabe-Schicht hat a Units, wobei a die Dimensionalität der Eingabedaten ist; die Hidden-Schicht hat K Hidden-Units. Die Ausgabe-Schicht hat in den hier betrachteten Fällen immer genau eine Unit ($b = 1$). Alle Betrachtungen für die RBF-Netze lassen sich ohne weiteres für $b > 1$ verallgemeinern. Mehr als eine Ausgabe-Unit werden bei der Klassifikation in mehr als zwei Klassen benötigt, wenn z. B. die 1-aus-c-Kodierung (siehe Bishop [2], S. 225) benutzt wird.
Soll das Netz auf ein Klassifikationsproblem angewandt werden, so sind a und b bereits durch die Daten gegeben. Die *Komplexität* der durch ein Netz dargestellten Funktionen wird u. a. durch die Anzahl der Hidden-Units bestimmt. Wieviel Komplexität benötigt

[1] Anstelle einer gewichteten Fehlerfunktion könnte man auch Bootstrap-Replikate erzeugen und die Netze damit trainieren. Allerdings hat sich in Experimenten gezeigt, daß die direkte Verwendung der Gewichte, die der Boosting-Algorithmus erzeugt, bessere Ergebnisse liefert (Scapire et al. [53]).

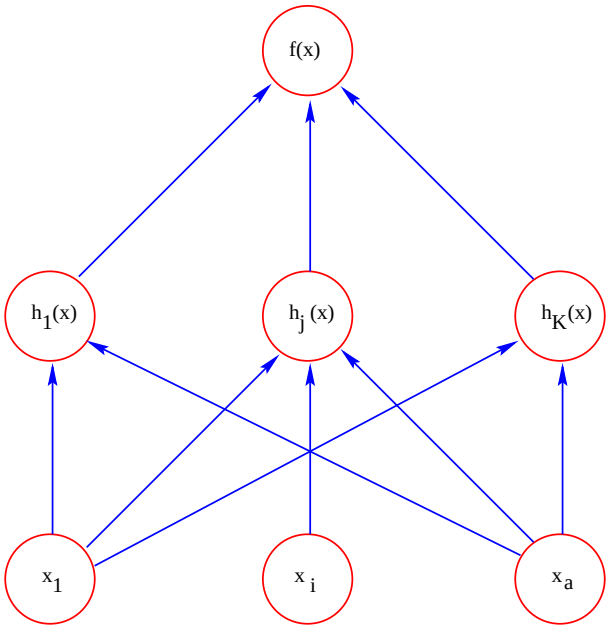


Abbildung A.1: Das traditionelle RBF-Netz. Zuerst wird die Aktivierung jeder Hidden-Unit für einen Eingabevektor $\mathbf{x}$ berechnet. Dann werden die Aktivierungen mit $\{w^{[k]}\}_{k=1}^{K}$ gewichtet und zur Ausgabe $f(\mathbf{x})$ aufsummiert.

wird, werden hängt vom Problem ab[2]. Auf das Problem der Bestimmung der optimalen Anzahl der Hidden-Units (Problem der *Modell-Selektion*) soll im folgenden nicht näher eingegangen werden (siehe z.B. Bishop [2], S. 371–377 oder Orr [48], S. 19–22).
Ein *lineares* Netz hat die Form

$$f(\mathbf{x}) = \sum_{k=1}^{K} w^{[k]} g^{[k]}(\mathbf{x}) \ . \tag{A.1}$$

Die Funktion f ist also eine *Linearkombination* von K fest gewählten *Basisfunktionen* $\{g^{[k]}\}_{k=1}^{K}$. Prinzipiell könnten beliebige Basisfunktionen benutzt werden, besonders einfach ist jedoch der Fall, bei dem alle Basisfunktionen einer bestimmten Klasse von Funktionen angehören.
Radiale Basisfunktionen sind eine spezielle Klasse von Funktionen. Sie haben die charakteristische Eigenschaft, daß ihre Aktivierung monoton mit steigendem Abstand von einem *zentralen Punkt* μ steigt bzw. sinkt. Die allgemeine Form einer radialen Basisfunktion ist

$$g(\mathbf{x}) = \phi\left((\mathbf{x}-\mu)^{\top}\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \ .$$

Die Funktion $\phi(x)$ und die Wahl von $\mathbf{\Sigma}$ bestimmen den konkreten Charakter der Basisfunktion.

[2]Ist die Komplexität zu klein gewählt, so erhält man eine schlechte Näherung der zugrundeliegenden Funktion (*underfitting*). Wird die Komplexität zu groß gewählt, so kann es zum Auswendiglernen (*overfitting*) kommen.

Hier soll es speziell um *Gaußsche Basisfunktionen* der Form

$$g^{[k]}(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x} - \mu^{[k]}||^2}{2(\sigma^{[k]})^2}\right) \tag{A.2}$$

gehen[3]. Man kann jedoch auch andere Klassen von Basisfunktionen benutzen (multiquadratische, inverse multiquadratische, Wavelet- und Cauchy-Funktionen für ϕ; vgl. Orr [48], S. 40). Die Gaußschen Basisfunktionen, wie auch z. B. die Wavelet-Basis, haben den Vorteil, daß die Aktivierung mit steigendem Abstand vom Zentrum stark abfällt. Diese Eigenschaft impliziert die erwünschte Lokalität: Jede Hidden-Unit ist nur für einen Teil des Eingaberaums zuständig - das RBF-Netz hat K *rezeptive Felder* bzw. *Zentren*.
Das RBF-Netz unterscheidet sich von einem traditionellen Multi-Layer-Perceptron in zwei wichtigen Punkten [36]:

1. Der Verwendung *lokaler* Aktivierungsfunktionen in der Hiddenschicht: Jede Unit der Hidden-Schicht liefert nur in einem eng begrenztem Bereich im Raum der Eingabevektoren eine nennenswerte Aktivierung. Diese Eigenschaft macht die Optimierung und Analyse der Netze einfacher und die Ergebnisse interpretierbarer[4].

2. Der Benutzung eines *hybriden* Lernverfahrens: einer Kombination aus *unsupervised* und *supervised learning*.

   - *unsupervised learning* („Lernen ohne Lehrer"). Wird die Information über die korrekte Ausgabe nicht benutzt, spricht man von unsupervised learning. Das Netz sucht selbständig eine Repräsentation der Eingabedaten. Gesteuert wird es dabei nur durch die Eingabedaten selbst. Die Hidden-Schicht wird auf diese Weise trainiert.
   - *supervised learning* („Lernen mit Lehrer"). Hierbei wird vorausgesetzt, daß die korrekte Ausgabe zu einem Eingabevektor $\mathbf{x}$ bekannt ist. Die Ausgabe $f(\mathbf{x})$ des Netzes wird mit der korrekten Ausgabe $\hat{y}(\mathbf{x})$ verglichen und die Netzparameter daraufhin entsprechend verändert. Auf diese Weise wird die Ausgabe-Schicht trainiert.

Bestimmung der Parameter der Basisfunktionen

Bestimmung der Zentren Für die Bestimmung der Zentren der Basisfunktionen im unsupervised Lernschritt gibt es einige Heuristiken (siehe z. B. Duda et al. [19], S. 217ff.). Der bekannteste dieser Algorithmen ist der K-Means-Cluster-Algorithmus. Der Algorithmus wird hier nur kurz vorgestellt, dabei wird den Ausführungen in Bishop [2] (S. 187ff.) gefolgt und auf den Fall gewichteter Trainingsbeispiele erweitert.
Die Zentren der Basisfunktionen werden so gewählt, daß sie die Daten möglichst gut repräsentieren, d. h. bestmöglich die Datendichte im Eingaberaum wiedergeben.
Bei ungewichteten Trainingsbeispielen $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(l)}\}$ kann man das durch Minimierung

[3] $\phi(x) = \exp(x)$ und $\mathbf{\Sigma} = \mathbf{I}\sigma^2$.

[4] Beispielsweise kann man bei RBF-Netzen, die auf OCR-Daten trainiert wurden, Zentren finden, die einem Buchstaben oder einer Ziffer ähneln bzw. entsprechen.

des Funktionals

$$J = \sum_{k=1}^{K} \sum_{i \in \mathcal{S}_k} ||\mathbf{x}^{(i)} - \mu^{[k]}||^2 \tag{A.3}$$

erreichen. Die Zentren sollen möglichst kleine (euklidische) Abstände[5] zu allen assoziierten Trainingsbeispielen $\mathcal{S}_k$ haben. Dabei ist das k-te Zentrum durch

$$\mu^{[k]} = \frac{1}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \mathbf{x}^{(i)} \tag{A.4}$$

definiert (Schwerpunkt) und die Menge der Trainingsbeispiele, die zum k-ten Zentrum gehören, durch

$$\mathcal{S}_k = \left\{ i \mid k = \underset{q=1,\ldots,K}{\operatorname{argmin}} ||\mathbf{x}^{(i)} - \mu^{[q]}|| \right\}. \tag{A.5}$$

Der K-Means-Cluster-Algorithmus ist eine Heuristik, die versucht, J zu minimieren. Die *Batch-Version*[6] des Algorithmus (Lloyd [41]) beginnt mit der zufälligen Aufteilung der Trainingsbeispiele in K Teilmengen. Von diesen Teilmengen wird dann der Schwerpunkt entsprechend ($A.4$) berechnet. Danach werden die Mengen nach Gleichung ($A.5$) neu definiert. Solange sich die Einteilung der Trainingsbeispiele in die Cluster ändert, wird abwechselnd der Schwerpunkt und die Einteilung neu berechnet. Es kann gezeigt werden (Linde et al. [40]), daß jede dieser Iterationen den Wert von J nicht vergrößert. Damit konvergiert dieses Verfahren.
Die Erweiterung für gewichtete Trainingsbeispiele $\{(\mathbf{x}^{(1)}, \gamma_1), \ldots, (\mathbf{x}^{(l)}, \gamma_l)\}$ ist einfach: Gleichung ($A.3$) verändert sich zu

$$J = \sum_{k=1}^{K} \sum_{i \in \mathcal{S}_k} \gamma_i ||\mathbf{x}^{(i)} - \mu^{[k]}||^2 \ , \tag{A.6}$$

und ($A.4$) zu

$$\mu^{[k]} = \frac{1}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \gamma_i \mathbf{x}^{(i)} \ . \tag{A.7}$$

Bestimmung der Radien Die Radien $\{\sigma^{[k]}\}_{k=1}^{K}$ bestimmen die Größe der *rezeptiven Felder*, und damit den Grad ihrer gegenseitigen Überlappung. Es gibt einige Ad-hoc-Berechnungen für die Radien (Kohlmorgen, [36]; Moody et al., [45]).
Die Radien kann man auf eine sehr einfache Weise berechnen: man bestimmt den Abstand zum nächsten Zentrum und benutzt diesen als Radius

$$\sigma^{[k]} = \min_{k \neq j = 1,\ldots,K} ||\mu^{[k]} - \mu^{[j]}|| \ .$$

[5] andere Normen können natürlich genauso benutzt werden (vgl. Duda et al. [19], S. 217ff.)
[6] für Online-Version siehe z. B. Moody et al. [45]; Bishop [2], 187ff.

Bestimmung der Gewichte

Die Gewichte der Ausgabeschicht $\mathbf{w}$ werden supervised bezüglich dem *mittlerem quadratische Fehler* (mean squared error) bestimmt:

$$MSE = \sum_{i=1}^{l} \gamma^{(i)} \left(\hat{y}^{(i)} - f(\mathbf{x}^{(i)}) \right)^2 .$$

Um Overfitting und numerische Effekte zu verhindern, benutzt man oft *weight decay* als eine einfache Form der *Regularisierung* [2]. Das äußert sich in der Erweiterung der Fehlerfunktion:

$$C = \sum_{i=1}^{l} \gamma^{(i)} \left(\hat{y}^{(i)} - f(\mathbf{x}^{(i)}) \right)^2 + \frac{\lambda}{l} \sum_{k=1}^{K} (w^{[k]})^2 . \tag{A.8}$$

Diese Regularisierung bewirkt, daß Modelle mit kleinen Gewichten bevorzugt werden. Dies drückt unser Wissen (*prior knowledge*) über die zu approximierende Funktion aus, von der wir annehmen, daß sie flach und glatt ist. Je größer die *Regularisierungskonstante* λ gewählt wird, desto flacher wird die durch das Netz dargestellte Funktion werden[7].
Um die optimalen Gewichte zu finden, berechnet man zunächst die Ableitungen

$$\begin{aligned} \frac{\partial C}{\partial w^{[q]}} &= -2 \sum_{i=1}^{l} \gamma^{(i)} \left(\hat{y}^{(i)} - f(\mathbf{x}^{(i)}) \right) \frac{\partial}{\partial w^{[q]}} f(\mathbf{x}^{(i)}) + 2 \frac{\lambda}{l} w^{[q]} \\ &= -2 \sum_{i=1}^{l} \gamma^{(i)} \left(\hat{y}^{(i)} - f(\mathbf{x}^{(i)}) \right) g^{[q]}(\mathbf{x}^{(i)}) + 2 \frac{\lambda}{l} w^{[q]} \end{aligned} \tag{A.9}$$

wobei

$$\frac{\partial}{\partial w^{[q]}} f(\mathbf{x}^{(i)}) = \frac{\partial}{\partial w^{[q]}} \sum_{k=1}^{K} w^{[k]} g^{[k]}(\mathbf{x}^{(i)}) = g^{[q]}(\mathbf{x}^{(i)}) .$$

Für die optimalen Gewichte gilt $\frac{\partial C}{\partial w^{[q]}} = 0$, für alle $q = 1, \ldots, K$, d. h.

$$\begin{aligned} \frac{\lambda}{l} w^{[q]} &= \sum_{i=1}^{l} \gamma^{(i)} \left(\hat{y}^{(i)} - f(\mathbf{x}^{(i)}) \right) g^{[q]}(\mathbf{x}^{(i)}) \\ \sum_{i=1}^{l} \gamma^{(i)} \hat{y}^{(i)} g^{[q]}(\mathbf{x}^{(i)}) &= \sum_{i=1}^{l} \gamma^{(i)} f(\mathbf{x}^{(i)}) g^{[q]}(\mathbf{x}^{(i)}) + \frac{\lambda}{l} w^{[q]} . \end{aligned}$$

Mit $\mathbf{g}^{(i)} = \left[g^{[1]}(\mathbf{x}^{(i)}), \ldots, g^{[K]}(\mathbf{x}^{(i)}) \right]^\top$ erhält man

$$\begin{aligned} \sum_{i=1}^{l} \gamma^{(i)} \hat{y}^{(i)} \mathbf{g}^{(i)} &= \sum_{i=1}^{l} \gamma^{(i)} f(\mathbf{x}^{(i)}) \mathbf{g}^{(i)} + \frac{\lambda}{l} \mathbf{w}^\top \\ \sum_{i=1}^{l} \gamma^{(i)} \hat{y}^{(i)} \mathbf{g}^{(i)} &= \sum_{i=1}^{l} \gamma^{(i)} (\mathbf{w}^\top \cdot \mathbf{g}^{(i)}) \mathbf{g}^{(i)} + \frac{\lambda}{l} \mathbf{w}^\top . \end{aligned}$$

[7] Auf die Modell-Selektion bzgl. λ wird nicht näher eingegangen, siehe dazu [48].

Faßt man dann die Vektoren $\mathbf{g}^{(i)}$ zu einer Matrix G zusammen, kann man schreiben:

$$\begin{aligned} G \cdot (\gamma^\top \odot \hat{\mathbf{y}}^\top) &= G \cdot \left(\gamma^\top \odot (G^\top \cdot \mathbf{w}^\top)\right) + \frac{\lambda}{l}\mathbf{w}^\top \\ &= \left(G \cdot (\Gamma \odot G^\top)\right) \cdot \mathbf{w}^\top + \frac{\lambda}{l}\mathbf{w}^\top \\ &= \left(G \cdot (\Gamma \odot G^\top) + \frac{\lambda}{l} I\right) \cdot \mathbf{w}^\top \ , \end{aligned}$$

wobei $\odot$ elementweises Multiplizieren bedeutet und $\Gamma = \underbrace{[\gamma, \ldots, \gamma]}_{K mal}$.

Damit ist der optimale Gewichtsvektor durch

$$\mathbf{w}^\top = \left(G \cdot (\Gamma^\top \odot G^\top) + \frac{\lambda}{l} I\right)^{-1} \cdot G \cdot (\gamma^\top \odot \hat{\mathbf{y}}^\top) \tag{A.10}$$

bestimmt.

Die Berechnung der Gewichte reduziert sich damit auf einige nicht sehr aufwendige Matrix-Operationen.

A.1.2 Nichtlineare Erweiterung der RBF-Netze

In diesem Abschnitt werden die linearen Netze um die Nachadaption der Zentren erweitert. Zunächst werden die Gradienten für die Radien und die Zentren benötigt:

- **Gradienten für die RBF-Radien**

$$\frac{\partial C}{\partial \sigma^{[q]}} = 2 \sum_{i=1}^{l} \gamma^{(i)} \left(f(\mathbf{x}^{(i)}) - \hat{y}^{(i)}\right) \frac{\partial}{\partial \sigma^{[q]}} f(\mathbf{x}^{(i)}) \tag{A.11}$$

  wobei

$$\frac{\partial}{\partial \sigma^{[q]}} f(\mathbf{x}^{(i)}) = 2 w^{[q]} \frac{||\mu^{[q]} - \mathbf{x}^{(i)}||^2}{(\sigma^{[q]})^3} g^{[q]}(\mathbf{x}^{(i)}) \ .$$

- **Gradienten für die RBF-Zentren**

$$\frac{\partial C}{\partial \mu^{[q]}} = 2 \sum_{i=1}^{l} \gamma^{(i)} (f(\mathbf{x}^{(i)}) - \hat{y}^{(i)}) \frac{\partial}{\partial \mu^{[q]}} f(\mathbf{x}^{(i)}) \tag{A.12}$$

  wobei

$$\frac{\partial}{\partial \mu^{[q]}} f(\mathbf{x}^{(i)}) = 2 w^{[q]} \frac{x^{(i)} - \mu^{[q]}}{(\sigma^{[q]})^2} g^{[q]}(\mathbf{x}^{(i)}) \ .$$

Leider kann man die Parameter der Basisfunktionen σ und μ nicht per linearer Algebra optimieren. Es ist ein nichtlineares Verfahren, wie z.B. ein *Konjugiertes Gradienten Verfahren* (CG-Verfahren, siehe z. B. [42], [50]) oder eine (Quasi-)-*Newton-Methode* (z. B. BFGS-Verfahren) zu benutzen. Man könnte die Gradienten in $(A.9), (A.11), (A.12)$ direkt zur *globalen Optimierung* benutzen. Allerdings würde man damit die exakte Lösung für

die Gewichte ($A.10$) nicht benutzen. Andererseits ist der Aufwand für die Berechnung der Gradienten vergleichbar mit dem Aufwand für die Bestimmung des optimalen Gewichtsvektors.
Es hat sich folgende Lösung bewährt: Man benutzt das z. B. CG-Verfahren (es wurde die Implementation aus [50] benutzt) mit ($A.11$) und ($A.12$), um die Radien und Zentren bezüglich C zu optimieren, wobei bei jeder Auswertung der Fehlerfunktion der Gewichtsvektor $\mathbf{w}$ optimal mit ($A.10$) eingestellt ist. Dies sichert eine schnelle Konvergenz. Nachteilig ist, daß die den Optimierungsverfahren oft inhärente *Line-Search-Prozedur* die Fehlerfunktion möglicherweise häufig auswertet. Die Verwendung der optimalen Gewichte in jedem Schritt zeigte deutliche Konvergenzverbesserungen.

Algorithm RBF-Net

Input:

Sequenz gelabelter Trainingsbeispiele $\mathbf{Z} = \langle(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)})\rangle$

Anzahl der RBF-Zentren K

Regularisatierungskonstante λ

Anzahl der Iterationen T

Initialize:

Benutze K-Means-Clustering als Startwerte für μ_k und σ_k (k=1,...,K)

Do for $t = 1 : T$,

1. Berechne die optimalen Endgewichte $\mathbf{w} = \left(GG^\top + \frac{\lambda}{l} I\right)^{-1} G\mathbf{y}^\top$
2a. Berechne die Gradienten $\frac{\partial}{\partial \mu_k} C$ und $\frac{\partial}{\partial \sigma_k} C$ mit optimal eingestelltem $\mathbf{w}$ und erzeuge einen Gradientenvektor $\mathbf{v}$
2b. Berechne die konjugierte Richtung $\overline{\mathbf{v}}$
3a. Finde mit Hilfe einer Line-Search-Prozedur die minimierende Schrittweite δ in Richtung $\overline{\mathbf{v}}$; in jeder Auswertung von C werden die Endgewichte optimal bestimmt (wie in Zeile 1)
3b. Aktualisiere μ_k und σ_k mit $\overline{\mathbf{v}}$ und δ

Output: Optimiertes RBF-Netz

Abbildung A.2: Der Algorithmus zum Trainieren der RBF-Netze

A.2 Boosting

A.2.1 Analytische Bestimmung von b_t für AdaBoost(ϕ)

Im Beweis von Proposition 2.5 wird die analytische Minimierung des Funktionals G bzgl. b_t benötigt (vgl. Zeile 3a. beim nichtnormalisierten Arcing). Gleichung (2.12) läst sich für den speziellen Fall $s \equiv \exp$ umschreiben zu

$$G(b_t) = \sum_{i=1}^{l} \exp\Big(\mathrm{err}_{t-1}(\mathbf{z}^{(i)}, \mathbf{b}^{t-1}) - |\mathbf{b}^{t-1}|\phi + b_t(l_t(\mathbf{z}^{(i)}) - \phi)\Big) \ . \tag{A.13}$$

Diese Funktion hat positive 2. Ableitungen und es existiert somit ein globales bei $\frac{\partial}{\partial b_t} g(b_t) = 0$. Es ist also

$$\begin{aligned}
0 = \frac{\partial}{\partial b_t} g(b_t) &= \sum_{i=1}^{l} (l_t(\mathbf{z}^{(i)}) - \phi) \exp\Big(\mathrm{err}_{t-1}(\mathbf{z}^{(i)}, \mathbf{b}^{t-1}) - |\mathbf{b}^{t-1}|\phi + b_t(l_t(\mathbf{z}^{(i)}) - \phi)\Big) \\
0 &= \sum_{i=1}^{l} (l_t(\mathbf{z}^{(i)}) - \phi) p_i^{t-1} \exp\Big(b_t(l_t(\mathbf{z}^{(i)}) - \phi)\Big) \\
&= -\phi \sum_{i:h_t(\mathbf{x}^{(i)})=y^{(i)}} p_i^{t-1} e^{-b_t\phi}(1-\phi) \sum_{i:h_t(\mathbf{x}^{(i)})\neq y^{(i)}} p_i^{t-1} e^{b_t(1-\phi)} \\
&= -\phi(1-\epsilon_t) e^{-b_t\phi}(1-\phi)\epsilon_t e^{b_t(1-\phi)}
\end{aligned}$$

zu lösen. Damit erhält man

$$\phi(1-\epsilon_t) = (1-\phi)\epsilon_t e^{b_t}$$

und somit

$$b_t = \log \frac{\phi(1-\epsilon_t)}{(1-\phi)\epsilon} \ . \tag{A.14}$$

A.2.2 Beweis für Satz 3.10

Der Beweis erfolgt nach dem gleichen Schema wie der von Satz 5 in [53]. Satz 3.10 stellt eine Generalisierung für $\phi \neq \frac{1}{2}$ dar.

Beweis: Wenn $yf(x) \leq \theta$, dann gilt

$$y \sum_{t=1}^{T} b_t h_t(x) \leq \theta \sum_{t=1}^{T} b_t \ ,$$

und damit auch

$$\exp\left\{ -\frac{y}{2} \sum_{t=1}^{T} b_t h_t(x) + \frac{\theta}{2} \sum_{t=1}^{T} b_t \right\} \geq 1 \ .$$

Deshalb gilt

$$\begin{aligned}
\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] &\leq \frac{1}{l} \sum_{i=1}^{l} \exp\left\{ -\frac{y^{(i)}}{2} \sum_{t=1}^{T} b_t h_t(\mathbf{x}^{(i)}) + \frac{\theta}{2} \sum_{t=1}^{T} b_t \right\} \\
&= \frac{\exp\left(\frac{\theta}{2} \sum_{t=1}^{T} b_t\right)}{l} \sum_{i=1}^{l} \exp\left\{ -\frac{y^{(i)}}{2} \sum_{t=1}^{T} b_t h_t(\mathbf{x}^{(i)}) \right\} \ ,
\end{aligned}$$

wobei

$$\begin{aligned}
& \sum_{i=1}^{l} \exp\left\{-\frac{y^{(i)}}{2}\sum_{t=1}^{T} b_t h_t(\mathbf{x}^{(i)})\right\} \\
= & \sum_{i=1}^{l} \exp\left\{-\frac{y^{(i)}}{2}\sum_{t=1}^{T-1} b_t h_t(\mathbf{x}^{(i)})\right\} \exp\left\{-\frac{y^{(i)}}{2} b_T h_T(\mathbf{x}^{(i)})\right\} \\
= & \sum_{i:h_T(\mathbf{x}^{(i)})=y^{(i)}} \exp\left\{-\frac{y^{(i)}}{2}\sum_{t=1}^{T-1} b_t h_t(\mathbf{x}^{(i)})\right\} e^{-b_T/2} + \\
& + \sum_{i:h_T(\mathbf{x}^{(i)})\neq y^{(i)}} \exp\left\{-\frac{y^{(i)}}{2}\sum_{t=1}^{T-1} b_t h_t(\mathbf{x}^{(i)})\right\} e^{b_T/2} \\
= & \left(\sum_{i=1}^{l} \exp\left\{-\frac{y^{(i)}}{2}\sum_{t=1}^{T-1} b_t h_t(\mathbf{x}^{(i)})\right\}\right)\left((1-\epsilon_T)e^{-b_T/2} + \epsilon_T e^{b_T/2}\right) \ ,
\end{aligned}$$

da

$$\epsilon_T = \frac{1}{\sum_{j=1}^{l} w_j^T} \sum_{i:h_T(\mathbf{x}^{(i)})\neq y^{(i)}} w_i^T \ .$$

Mit $\sum_{i=1}^{l} \exp(0) = l$ (für $t = 1$) erhält man damit rekursiv

$$\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] \ \leq \ \exp\left(\frac{\theta}{2}\sum_{t=1}^{T} b_t\right)\prod_{t=1}^{T}\left((1-\epsilon_t)e^{-b_t/2} + \epsilon_t e^{b_t/2}\right) \ .$$

Setzt man die Definition für b_t ein erhält man

$$\begin{aligned}
\mathbf{P}_{(\mathbf{x},y)\sim\mathbf{Z}}[yf(\mathbf{x}) \leq \theta] \ \leq & \left(\prod_{t=1}^{T}\frac{1-\epsilon_t}{\epsilon_t}\ \prod_{t=1}^{T}\frac{\phi}{1-\phi}\right)^{\theta/2} \cdot \\
& \cdot\left(\sqrt{\frac{\phi}{1-\phi}} + \sqrt{\frac{1-\phi}{\phi}}\right)^{T}\prod_{t=1}^{T}\sqrt{(1-\epsilon_t)\epsilon_t} \\
= & \left(\left(\frac{\phi}{1-\phi}\right)^{\frac{1+\phi}{2}} + \left(\frac{1-\phi}{\phi}\right)^{\frac{1-\phi}{2}}\right)^{T}\prod_{t=1}^{T}\sqrt{(1-\epsilon_t)^{1+\theta}\epsilon_t^{1-\theta}} \\
= & \left(\varphi^{\frac{1+\theta}{2}} + \varphi^{-\frac{1-\theta}{2}}\right)^{T}\prod_{t=1}^{T}\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \ .
\end{aligned}$$

■

A.3 Datensätze

A.3.1 Datensätze zur Veranschaulichung

In vielen Fällen ist es sinnvoll, sich die Arbeitsweise eines Algorithmus an einfachen Beispielen zu veranschaulichen. Das schafft ein gutes Verständnis und spart Rechenzeit. Die folgenden beiden Datensätze wurden für diesen Zweck entworfen und benutzt.

Datensatz #1: „Banana shape"

Um sich die entstehenden Entscheidungslinien in Grafiken ansehen zu können, braucht man einen zweidimensionalen Datensatz. Der Datensatz darf auch nicht zu trivial sein, da man sonst die daraus gewonnenen Ergebnisse nicht auf andere (komplexere) Datensätze übertragen kann. Datensatz #1 erfüllt diese Bedingungen relativ gut: Die Modellselektion dafür ergibt ein RBF-Netz, das nicht allzu einfach ist (10-20 Zentren) und auch der erreichbare Testfehler ist groß genug, um Vergleiche anstellen zu können (etwa 4.5%).
Der Datensatz wurde durch eine Überlappung mehrerer (verformter) Gauß-Glocken und gleichverteilter Rechtecke erzeugt (vgl. Teilabbildung 1 in Abbildung 3.2). In dieser Abbildung ist auch eine Approximation der Bayes-Entscheidunglinie abgebildet, die essentiell die Form einer *Banane* beschreiben.
Um auch die Rausch-Robustheit der Algorithmen untersuchen zu können, wurden verschiedene Stärken gleichverteilten Rauschens (Standardabweichungen $\sigma = 0.0\%$, 1.4%, 4.3%, 8.7%, 15.9%, wobei Daten eine Standardabweichung von 1 haben) zu den Eingabedaten der Trainingsmenge addiert. Für das Training der Klassifikatoren werden 300 Trainingsbeispiele benutzt (im Benchmark 100 weitere Beispiele zur Validierung). Zum Testen der Generalisierungsfähigkeit werden 5000 Testbeispiele verwendet.

Datensatz #2: „Waveform"

Datensatz #1 hat den Vorteil und auch das Problem, daß er niederdimensional ist. Manche Probleme entstehen erst, wenn man die Anzahl der Dimensionen erhöht [2]. Deshalb wird noch ein zweiter Datensatz benutzt, der höherdimensional ist.
Datensatz #2 stammt aus dem UCI-Repository und wurde schon oft für die Analyse von Decision Trees benutzt [9]. Die Klassen wurden künstlich aus Stützstellen (21) von 3 „Wellenfunktionen", die unterschiedliche Formen haben, erzeugt. Ziel ist es, die Funktion anhand der Werte der Stützstellen zu erkennen. Für die Benutzung in dieser Arbeit wurden zwei Klassen zu einer zusammengefaßt und die Klassifikation erfolgt zwischen den beiden Klassen und der dritten Klasse. Für das 3 Klassen-Problem ist der bestmögliche Testfehler 14%.
Für das Training der Klassifikatoren werden 300 Trainingsbeispiele benutzt (im Benchmark 100 weitere Beispiele zur Validierung). Zum Testen der Generalisierungsfähigkeit werden 4600 Testbeispiele verwendet.

A.3.2 USPS-Datensatz

Der *US Postal Service (USPS)* Datensatz enthält 9298 handgeschriebene Ziffern (7291 für das Training, 2007 zum Testen), die von Briefumschlägen aus Buffalo stammen [38]. Jede

Ziffer ist ein 16×16 Bild, daß durch einen 256-dimensionalen Vektor repräsentiert ist, der Einträgen zwischen -1 und 1 hat. Die Bilder wurden mit mit einem Gaußschen Kernel ($\sigma = 0.75$) geglättet [55].

Es ist bekannt, daß der USPS-Testdatensatz sehr schwierig zu klassifizieren ist – der Fehler bei einer manuellen Klassifikation liegt bei 2.5% [10]. Für einige in der Literatur angegeben Ergebnissen wurde ein erweitertes Trainingsset benutzt. Beispielsweise in [18] wurden einige gedruckte Buchstaben hinzugefügt (Trainingsmenge umfaßte dann 9709 Beispiele), was zu einer Verbesserung der Ergebnisse führte. In den hier beschriebenen Experimenten wurden nur die originalen 7291 Beispiele benutzt. In Tabelle A.3.2 findet man eine Zusammenstellung der bisher auf diesem Datensatz erreichen Fehlerraten [55].

Tabelle A.1: Überblick über die bisher erreichten Fehlerraten auf dem USPS-Datensatz. Die menschliche Fehlerrate ist relativ hoch – 2.5% – und liegt relativ nahe an der von den Support-Vektor-Maschinen erreichten Fehlerrate.

Architektur	Fehlerrate	Referenz
Human	2.5%	Bromley et al., 1991 [10]
Tangent Distance	2.6%	Simard et al., 1993 [61]
2-layer perceptron	5.9%	Vapnik, 1995 [64]; Schölkopf, 1997 [55]
LeNet1	5.0%	LeCun, 1989, [38]
Polynomial SVM	4.0%	Schölkopf et al., 1995 [54]
tanh SVM	4.1%	Schölkopf et al., 1995 [54]
RBF SVM	4.2%	Schölkopf et al., 1995 [54]
Virtual RBF SVM	3.2%	Schölkopf et al., 1997 [58]
Virtual Poly. SVM	3.2%	Schölkopf et al., 1996 [56]
..with local kernel	3.0%	Schölkopf et al., 1998 [57]
k-means RBF*	6.7%	Schölkopf et al., 1997 [58]
RBF with SV centers	4.9%	Schölkopf et al., 1997 [58]
NN	5.9%	Simard et al., 1993 [61]

* Mit den in Anhang A.1 beschriebenen RBF-Netzen kann man Fehlerraten von unter 4.5% erreichen.

A.4 Implementation

Für die Experimente mit AdaBoost wurde das weitverbreitete Programmsystem MATLAB[8] eingesetzt. MATLAB ist eine matrixorientierte Programmiersprache, die für numerische Simulationen und Visualisierung hervorragend geeignet ist.

> "The central data construct in MATLAB is the *numeric array*, an ordered collection of real or complex numeric data with two or more dimensions. The basic data objects of signal processing (one-dimensional signalsor sequences, multi-channel signals, and two-dimensional signals) are all naturally suited to array representation." [44]

Die Hauptdatenstruktur von MATLAB, das *numerische Feld*, eignet sich besonders gut zur Repräsentation der für die Klassifikation verarbeiteten mehrdimensionalen Daten. Die umfangreichen Funktionsbibliotheken können in eigenen Routinen benutzt (oder erweitert) werden und ermöglichen so eine schnelle Umsetzung der Verfahren. Die Lösung vieler numerischer Aufgaben ist damit in einem Bruchteil der Zeit möglich, die bei Verwendung von Fortran oder C aufgewendet werden müßte, da die interne Codierung sorgfältig in C optimiert wurde.
Die Ensemble-Lernalgorithmen wurden deshalb in MATLAB implementiert. Sämtliche Diagramme und Funktionszeichungen entstanden mit den in MATLAB vorhandenen Graphikfunktionen.
Ein weiterer Vorteil von MATLAB ist die Plattformunabhängigkeit, d.h. die problemlose Portierbarkeit der MATLAB–Programme auf verschiedene Betriebssysteme und Rechner-Architekturen. So konnten problemlos auf vier verschiedenen Architekuren gerechnet werden. Es wurde u.a. auch auf einigen Rechnern mit mehreren Prozessoren gerechnet, für die ein Teil der Funktionen parallelisiert wurde.
MATLAB stellt auch objektorientierte Elemente in der Sprache zur Vefügung. Die verschiedenen Modifikationen von AdaBoost wurden durch Ableiten von einer gemeinsamen Basisklasse erhalten. Das senkte den Implementationsaufwand deutlich und verbesserte die Wartbarkeit des Quelltextes.
Für die Lösung der Probleme der linearen und quadratischen Programmierung wurde das LOQO-System von R.J. Vanderbei (siehe im Internet unter http://www.princeton.edu/ rvdb) benutzt, dem hiermit für die Zuverfügungstellung gedankt sei. LOQO hat sich für die Experimente für die nachträgliche Optimierung der Hypothesenkoeffizienten als ein sehr schnelles und zuverlässiges System zur Lösung quardratischer Optimierungsprobleme herausgestellt. Bei der linearen Optimierung gab es machmal (selten) Probleme mit der Konvergenz des Optimierers. Möglicherweise sind einige Ergebnisse leicht verfälscht.

[8] MATLAB ist ein eingetragenes Warenzeichen der *The MathWorks, Inc.*

Literaturverzeichnis

[1] A. Agresti. *Categorical Data Analysis.* John Wiley and Sons., Inc., 1990.

[2] C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[3] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

[4] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th International Conference Pattern Recognition and Neural Networks*, pages 77 – 87. IEEE Computer Society Press, 1994.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[6] L. Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California, June 1997.

[7] L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, July 1997.

[8] L. Breiman. Prediction games and arcing algorithms. Technical Report 504, Statistics Department, University of California, December 1997.

[9] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks, 1984.

[10] J. Bromley and E. Säckinger. Neural-network and k-nearest-neighbor classifiers. Technical Report 11359–910819–16TM, AT&T, 1991.

[11] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[12] C.J.C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In *Advances in Neural Information Processing Systems 9*, pages 275–381. MIT Press, 1997. http://svm.first.gmd.de/papers/nips96.ps.gz.

[13] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Mens.*, 20:37 – 46, 1960.

[14] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[15] T.G. Dietterich and E.B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, 1995.

[16] H. Drucker and C. Cortes. Boosting decision trees. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 479–485. The MIT Press, 1996.

[17] H. Drucker, C. Cortes, L.D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[18] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705 – 719, 1993.

[19] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1974.

[20] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, September 1995.

[21] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EUROCOLT: EUROCOLT, European Conference on Computational Learning Theory, EuroCOLT,*. LNCS, 1994.

[22] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[23] Y. Freund and R.E. Schapire. Game theory, on-line prediction and boosting. In *Proc. 9th Annu. Conf. on Comput. Learning Theory*, pages 325–332. ACM Press, New York, NY, 1996.

[24] J.H. Friedman. On bias, variance, 0/1–loss, and the corse of dimensionality. In Usama Fayyad, editor, *Data Mining and Knowledge Discovery*, volume I, pages 55–77. Kluwer Academic Publishers, 1997.

[25] T.T. Friess. Linear programing support vector machines and related algorithms. March 1998. Talk at GMD FIRST.

[26] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[27] A.J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artifical Intelligence*, 1998. To appear.

[28] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. Technical Report UCSC-CRL-91-02, University of California, Santa Cruz, 1989.

[29] S. Haykin. *Neural Networks; A comprehensive Foundation*. Macmillan, New York, 1 edition, 1994.

[30] J. Hertz, R. Palmer, and A. Krogh. *Neural Computation*, volume 1. Addison-Wesley, Reading, MA, USA, 1991. Prepared with TEX.

[31] T. Heskes. Balancing between bagging and bumping. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference*, pages 466–472, Cambridge, 1997. MIT Press.

[32] T. Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 1998. ftp://ftp.mbfys.kun.nl/snn/pub/reports/Heskes.biasvar.ps.Z, Accepted.

[33] C. Ji and S. Ma. Combinations of weak classifiers. *IEEE Transactions on Neural Networks*, 8(1):32–42, January 1997.

[34] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *J. Statistical Physics*, 34:975–986, 1984.

[35] R. Kohavi and D.H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proc. 13th International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann, 1996.

[36] J. Kohlmorgen. *Analyse schaltender und driftender Dynamik mit neuronalen Netzen.* PhD thesis, Technische Universität Berlin, Fachbereich 13 - Informatik; Prof. Dr. Jähnichen, Thesis Supervisor, März 1998.

[37] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551, 1989.

[39] Y. LeCun, L. Bottou L.D. Jackel, H. Drucker C. Cortes, J.S. Denker, I.Guyon, U.A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Learning algorithms for classification: A comparism on handwritten digit recognistion. *Neural Networks*, pages 261–276, 1995.

[40] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Trans. on Communications*, COM-28(1):84–95, January 1980.

[41] S.P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:128–137, 1982.

[42] D.G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Co., Reading, second edition, May 1984. Reprinted with corrections in May, 1989.

[43] D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *Proc. 14th International Conference on Machine Learning*, pages 211–218. Morgan Kaufmann, 1997.

[44] The MathWorks, editor. *MATLAB Signal Processing Toolbox User's Guide.* The MathWorks, Inc., 1996.

[45] J. Moody and C.J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[46] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Using support vector machines for time series prediction. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings ICANN'97, Int. Conf. on Artificial Neural Networks*, volume 1327 of *LNCS*, pages 999–1004, Berlin, 1997. Springer.

[47] T. Onoda, G. Rätsch, and K.-R. Müller. An asymptotic analysis of adaboost in the binary classification case. In *Proceedings ICANN'98, Int. Conf. on Artificial Neural Networks*, volume VI of *LNCS*, Berlin, 1998. Springer. Submitted.

[48] M.J.L. Orr. Introduction to radial basis function networks. Technical report, Centre for Neural Systems, Edinburgh University, 1996. http://www.cns.ed.ac.uk/people/mark/papers/intro.ps.

[49] M.P. Perrone. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.* PhD thesis, Brown University, Institute for Brain and Neural Systems; Dr. Leon N Cooper, Thesis Supervisor, May 1993.

[50] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, second edition, 1992.

[51] J.R. Quinlan. Boosting first-order learning (invited lecture). *Lecture Notes in Computer Science*, 1160:143, 1996.

[52] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions ... mar 1998. Unpublished.

[53] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[54] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining.* AAAI Press, Menlo Park, CA, 1995.

[55] B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, Munich, 1997.

[56] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[57] B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, Cambridge, MA, 1998. MIT Press. In press.

[58] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. Sign. Processing*, 45:2758 – 2765, 1997.

[59] B. Schölkopf, K.-K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, November 1997.

[60] H. Schwenk and Y.Bengio. Adaboosting neural networks. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings ICANN'97, Int. Conf. on Artificial Neural Networks*, volume 1327 of *LNCS*, pages 967–972, Berlin, 1997. Springer.

[61] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann.

[62] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. The MIT Press, 1996.

[63] R. Tibshirani. Bias, variance and prediction error for classifikation rules. Technical report, Department of Statistics, University of Toronto, 1996.

[64] V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[65] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2):264–280, 1971.

[66] J. Weston, A. Gammerman, M. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Density estimation using support vector machines. Technical Report CSD-TR-97-23, Computer Science Department, Royal Holloway, University of London, February 1998.