

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung

Hauptseminar Neuroinformatik

AdaBoost

„Fast and Robust Classification using Asymmetric AdaBoost
and a Detector Cascade“ von Paul Viola und Michael Jones

Sommersemester 2002

Betreuung durch Prof. H.-M. Groß

Manuela Bischoff

Matrikel-Nr. 27475

Studiengang Informatik

Inhaltsverzeichnis

1 Aufgabe und Zielstellung	2
2 Boosting	2
2.1 Zur Historie (aus [FS99])	2
2.2 Was ist Boosting?	2
2.3 Der Boosting-Algorithmus	3
3 AdaBoost	4
3.1 Adaptives Boosting	4
3.2 Der Algorithmus	4
3.3 Weitere Aspekte	5
3.4 Vor- und Nachteile	6
4 Anwendung von AdaBoost im Kontext der Gesichtsdetektion	7
4.1 Training der Klassifikator-Kaskade mit AdaBoost	7
4.2 Asymmetric AdaBoost	9
5 Vorhandene Implementierungen und Demos	10
6 Fazit	11
Literatur	12

1 Aufgabe und Zielstellung

Die vorliegende Arbeit entstand im Rahmen der Bearbeitung des Papers von Viola und Jones [VJ02]; die Aufbereitung dieses Papers und weiterführende Recherche erfolgte gemeinsam mit Stefan Wender, dessen Arbeit sich im wesentlichen auf das zur Gesichtsdetektion verwendete Verfahren bezog. Nach Viola und Jones liefert dieses Verfahren, welches AdaBoost zur Extraktion von Merkmalen und zum Training einer Klassifikator-Kaskade verwendet, vielversprechende Resultate. Des Weiteren kommt eine angepasste Variante von AdaBoost, das sog. Asymmetric AdaBoost, zum Einsatz, welches zur Verbesserung der Ergebnisse beiträgt. Eine detailliertere Beschreibung des gesamten Ablaufs des Verfahrens liefern Viola und Jones in [VJ01]; hier vergleichen sie auch andere bekannte Verfahren mit diesem Ansatz.

Ziel der vorliegenden Arbeit ist die nähere Betrachtung von Boosting im Allgemeinen, dem speziellen Verfahren *AdaBoost* und die Einordnung in die Anwendung bei Viola und Jones. Auch sollen Vor- und Nachteile von AdaBoost genannt und vorhandene Implementierungen kurz vorgestellt werden.

Sofern nicht anders angegeben beziehen sich alle Betrachtungen auf Zweiklassenprobleme; lediglich in Abschnitt 3.3 soll kurz auf die Anwendung im Zusammenhang mit Mehrklassenproblemen eingegangen werden.

2 Boosting

2.1 Zur Historie (aus [FS99])

Michael Kearns und Leslie G. Valiant waren die ersten, die die Frage aufwarfen, ob es möglich ist, einen „schwachen“ Lernalgorithmus zu einem beliebig genauen Lernalgorithmus zu verstärken, ihn zu „boosten“¹. Robert E. Schapire lieferte 1989 den ersten Polynomzeit-Algorithmus für das Boosting. Ein Jahr später entwickelte Yoav Freund einen effizienteren Algorithmus, der jedoch immer noch einige praktische Nachteile hatte. Erst 1995 präsentierten dann Y. Freund und Robert E. Schapire einen Boosting Algorithmus, der sich auch in der Praxis bewährte: AdaBoost. Bis heute wurden verschiedene Implementierungen und Weiterentwicklungen von AdaBoost und anderer Boosting-Algorithmen vorgestellt, darunter „Gentle AdaBoost“, „Brown-Boost“, „AsymBoost“ (Abschnitt 4.2) und „LocBoost“ (Java-Applet aus Abschnitt 5).

Boosting is a general method for improving the accuracy of any given learning algorithm. [FS99]

2.2 Was ist Boosting?

Boosting zählt zu den Meta-Lernverfahren und macht sich die Kombination von Entscheidungen mehrerer einzelner Klassifikatoren zu Nutze (Abb. 1); ein ihm verwandtes Verfahren ist *Bagging*. Beide Methoden konstruieren die Einzelklassifikatoren während der Trainingsphase selbst; Ausgangspunkt ist hierbei die Manipulation der Trainingsdaten, in deren Art und Weise sich Bagging und Boosting jedoch unterscheiden. Beiden gemeinsam ist allerdings die Voraussetzung eines instabilen Lernsystems, d. h. geringe Änderungen der Trainingsmenge führen zu unterschiedlichen Klassifikatoren [Sor99]. Die Manipulation der Trainingsdaten erfolgt bei Bagging durch Ziehen von Stichproben mit Zurücklegen. So entstehen unterschiedliche Mengen von Trainingsdaten, die zum Training der Einzelklassifikatoren genutzt werden. Sollen T unterschiedliche Einzelklas-

¹engl. to boost: (ver)stärken, ankurbeln, erhöhen

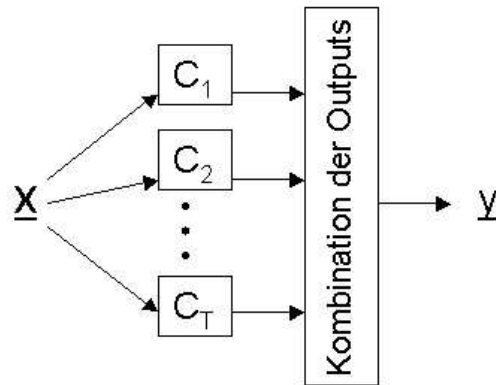


Abbildung 1: Kombination von Klassifikationen $C_1 \dots C_T$ zu einer Gesamtentscheidung

sifikatoren konstruiert werden, sind T solcher Mengen von Trainingsdaten anzulegen und jeder Einzelklassifikator auf einer dieser Mengen zu trainieren (detaillierter in [Sor99]).

2.3 Der Boosting-Algorithmus

Während Bagging die Trainingsmenge mit Hilfe des Zufalls manipuliert, versucht Boosting durch Gewichtung der einzelnen Trainingsdatensätze unterschiedliche Trainingsmengen für die einzelnen Klassifikatoren zu generieren, d. h. jedem Trainingsdatensatz wird ein veränderbares Gewicht zugeordnet.

Während des Trainings eines Einzelklassifikators wird das Gewicht einer jeden fehlklassifizierten Instanz erhöht, sodass im nächsten Durchlauf diese Daten mehr Einfluss im Training des folgenden Einzelklassifikators haben. Soll nun ein neues Muster x klassifiziert werden, so bilden die unterschiedlichen Klassifikatoren durch ein Voting-Verfahren eine gemeinsame finale Entscheidung. Die Stimmen der Einzelklassifikatoren besitzen allerdings eine Gewichtung in Abhängigkeit ihrer Leistungsfähigkeit, also ihrer Fehler-rate auf den Trainingsdaten. [Sor99]

Die Einzelklassifikatoren sollen sich ergänzen, nicht überlagern. [KM01]

Bei einem Training über T Runden lernt also in jeder Runde der neue Einzelklassifikator aus den Fehlern der vorhergehenden, d. h. er spezialisiert sich auf deren Fehlklassifikationen; der Gesamtklassifikator bildet dann eine gewichtete Mehrheitsentscheidung. Folgende Zusammenstellung zeigt kurz die Trainingsphase:

Boosting: Learning on the errors of others. [Kow01]

Konstruiere Einzelklassifikatoren C_1, C_2, \dots, C_T wie folgt:

- Trainiere C_1 auf den Originaldaten
- Trainiere C_2 „mit mehr Beachtung“ der von C_1 fehlklassifizierten Datensätze
- Trainiere C_3 „mit mehr Beachtung“ der von C_1 und C_2 fehlklassifizierten Datensätze
- usw.

Freund, Schapire in [FS99]: „Boosting can be viewed as repeated play of a certain game.“

Als Basisklassifikator können beliebige, einfache Klassifikatoren (auch schwache Klassifikatoren genannt), wie z. B. Single-Layer-Perceptron oder Naives Bayes, verwendet werden. Voraussetzung ist jedoch, dass der Klassifikationsfehler auf den Trainingsdaten

kleiner als 0.5 ist, was im Wesentlichen bedeutet, dass der gewählte Basisklassifikator sich zumindest besser als der Zufall verhalten muss.

Das in Abschnitt 5 näher betrachtete Java-Applet gibt einen Einblick, welche einfachen Klassifikatoren als Basis gewählt werden können.

3 AdaBoost

3.1 Adaptives Boosting

Die grundlegendste Eigenschaft von AdaBoost ist seine Fähigkeit den Trainingsfehler zu minimieren. Sei $\epsilon_t = \frac{1}{2} - \gamma_t$ der Fehler des Klassifikators C_t auf den Trainingsdaten. Da die Fehlerrate des Zufalls auf Zweiklassenproblemen $\frac{1}{2}$ ist, gibt γ_t also an, um wie viel die Hypothese des Klassifikators C_t besser ist als eine zufällige Entscheidung. In [FS95] leiten Schapire und Freund damit eine obere Schranke für den Trainingsfehler des finalen Gesamtklassifikators her. Daraus ergibt sich laut [FS99] folgendes: Wenn jede schwache Hypothese etwas besser ist als der Zufall, so dass $\gamma_t \geq \gamma$ für ein $\gamma > 0$, dann sinkt der Trainingsfehler exponentiell (Abb. 2). Aus diesem Zusammenhang ergibt sich die Entstehung von AdaBoost aus den frühen Boosting-Algorithmen (↗ 2.1):

A similar property is enjoyed by previous boosting algorithms. However, previous algorithms required that such a lower bound γ be known a priori before boosting begins. In practice, knowledge of such a bound is very difficult to obtain. AdaBoost, on the other hand, is adaptive in that it adapts to the error rates of the individual weak hypotheses. This is the basis of its name – "Ada" is short for "adaptive". [FS99]

Wie diese Anpassung an den aktuellen Fehler umgesetzt wird, soll im folgenden Abschnitt, der sich mit dem Algorithmus beschäftigt, näher erläutert werden.

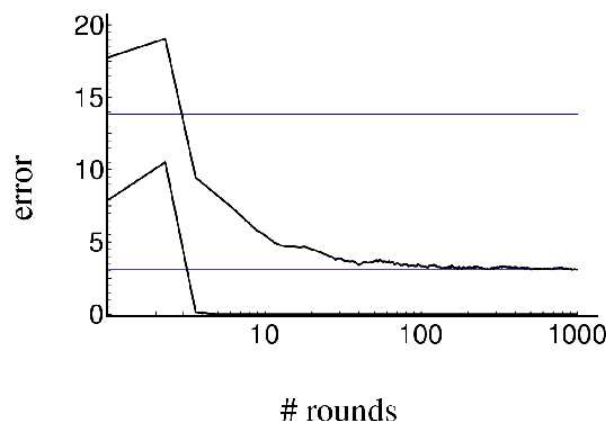


Abbildung 2: Fehler des resultierenden Klassifikators auf den Trainings- und den Validierungsdaten (aus [FS99]).

3.2 Der Algorithmus

Im folgenden geht es um den prinzipiellen Ablauf von Training und Klassifikation mit AdaBoost; Abb. 3 veranschaulicht diesen.

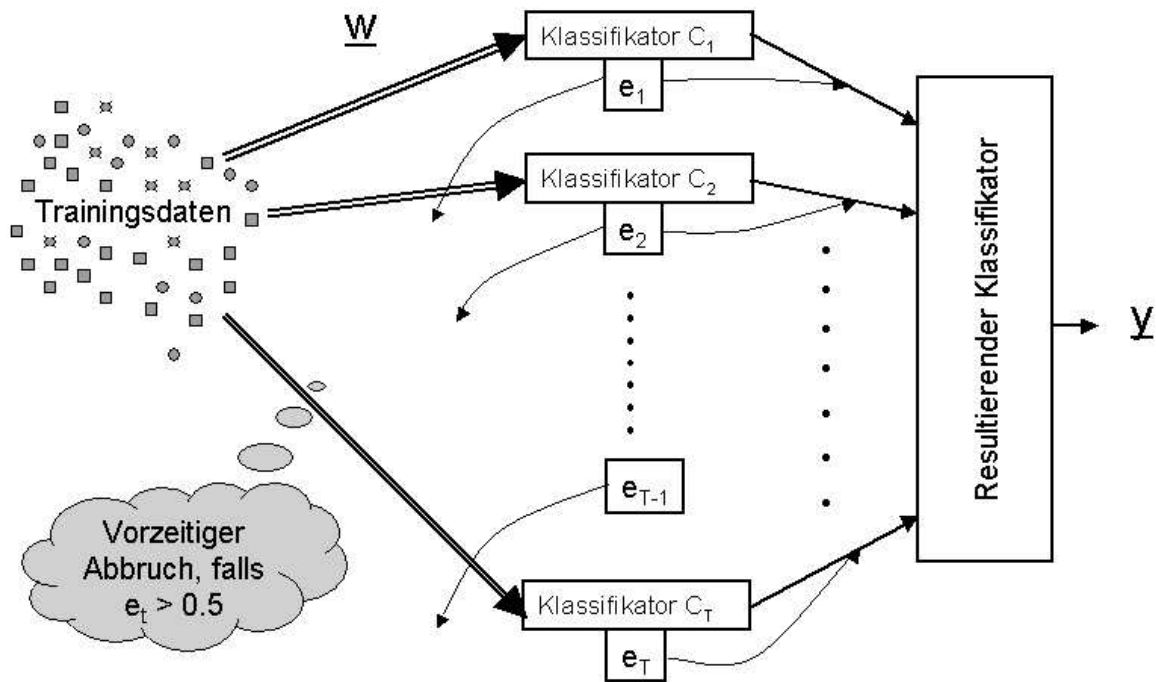


Abbildung 3: Der prinzipielle Ablauf von AdaBoost

Ausgehend von einem Zweiklassenproblem werden auf den Trainingsdaten T Einzelklassifikatoren $C_1 \dots C_T$ trainiert. In das Training ist die Gewichtung \underline{w} der Trainingsdatensätze geeignet einzubeziehen (Bsp. in Abschnitt 4). Die Anpassung dieser Gewichte erfolgt jeweils in Abhängigkeit vom vorhergehenden Fehler e_t , wobei das Gewicht von fehlklassifizierten Datensätzen erhöht wird. Höhere Gewichte konzentrieren sich demnach auf schwerer zu klassifizierende Datensätze. Nach T Runden ist das Training beendet und es wurden T Einzelklassifikatoren erzeugt; lediglich wenn der Fehler e_t größer also 0.5 ist, wird das Training früher abgebrochen (ein Fehler $e_t < 0.5$ ist Grundvoraussetzung für sinnvolle Ergebnisse beim Training mit AdaBoost) und es entstehen weniger Einzelklassifikatoren.

Die Kombination der Einzelhypothesen erfolgt durch eine gewichtete Mehrheitsentscheidung anhand des Trainingsfehlers e_t ; jedem Klassifikator C_t wird hierzu ein Gewicht w_t mit

$$w_t = -\log\left(\frac{e_t}{1 - e_t}\right)$$

zugewiesen. Das Gewicht eines Klassifikators ist um so größer je geringer sein Trainingsfehler; d. h. ein besserer Klassifikator hat mehr Einfluss auf das Gesamtergebnis und w_t ist somit ein Maß für die Wichtigkeit der Hypothese des Klassifikators C_t .

3.3 Weitere Aspekte

Generalisierungsfehler

Freund und Schapire zeigten in [FS95] (u. a. mit Hilfe des Trainingsfehlers, der Größe der Datensamples und der Anzahl der Trainingsrunden T) wie der Generalisierungsfehler eingeschränkt ist. Laut [FS99] lieferten sie eine Schranke, die zeigt, dass Boosting

bei zu hoher Anzahl von Trainingsrunden zu Overfitting neigt. Jedoch findet dieses Overfitting in vielen Anwendungen nicht statt, sondern AdaBoost senkte den Generalisierungsfehler noch weiter obwohl der Trainingsfehler bereits Null erreicht hatte. Ein neuer Ansatz in [SFBL97] bzgl. des Trainingsfehlers lieferte 1997 eine vom Trainingsfehler unabhängige obere Schranke für den Generalisierungsfehler und behob diesen Widerspruch zwischen Theorie und Praxis.

Mehrklassenprobleme, Reelle Ausgaben

Wurde bisher von einem Zweiklassenproblem mit binärer Ausgabe ausgegangen, so soll hier die Problematik der Klassifikation in mehrere Klassen und die Nutzung von Basisklassifikatoren mit reeller Ausgabe kurz durch die Angabe weiterführender Quellen beleuchtet werden.

In [SS98] zeigten Schapire und Singer, wie AdaBoost auch bei reell-wertiger Ausgabe (oder auch bei sog. „confidence-rated predictions“) des Basisklassifikators angewendet werden kann. Dieses Paper enthält des Weiteren mehrere Methoden zum Umgang mit Mehrklassenproblemen, verschiedene Boosting-Algorithmen dazu und die Ergebnisse durchgeführter Experimente.

Auch Freund und Schapire gehen in ihrer Einführung zu Boosting [FS99] kurz auf die Behandlung von Mehrklassenproblemen ein, beziehen sich zum Teil auf das Beispiel der Buchstabenklassifikation und nennen weitere Publikationen zu diesem Thema.

Parallelen zu Support Vector Machines

Ausgehend von der Annahme, dass die N Einzelklassifikatoren schon gefunden wurden und nur noch deren Gewichtungen für den Gesamtklassifikator zu wählen ist, erklären Freund Schapire (ebenfalls in [FS99]) kurz warum Support Vector Machines und AdaBoost sich ähnlich sind und worin die grundlegenden Unterschiede bestehen.

3.4 Vor- und Nachteile

Ein wesentlicher Vorteil von AdaBoost ist seine schnelle, einfache und leichte Implementierung und die Tatsache, dass keine Parameter zu adaptieren sind (außer der Anzahl der Trainingsrunden T); des Weiteren kann jeder beliebige einfache Klassifikator als Basis genutzt werden. Eine Zusammenstellung theoretischer Grundlagen garantiert überdies das in der Praxis beobachtete Verhalten; so z. B. der exponentiell fallende Trainingsfehler. Für den Designer eines Lernsystems bietet AdaBoost einen neuen Ansatz:

...instead of trying to design a learning algorithm that is accurate over the entire space, we can instead focus on finding weak learning algorithms that only need to be better than random. [FS99]

AdaBoost can be interpreted as a coordinate-wise gradient descent in the space of linear classifiers (over weak hypotheses). [FS99]

Die tatsächliche Performanz von AdaBoost bei konkreten Problemen hängt jedoch stark von der Menge der Trainingsdaten und der Art des einfachen (oder auch „schwachen“) Basisklassifikators ab; vor allem darf dieser nicht zu schwach, jedoch auch nicht zu komplex sein. Eine u. U. negative Eigenschaft von AdaBoost ist seine Anfälligkeit für Rauschen; die starke Gewichtung von „Ausreißern“ kann jedoch auch zur Identifikation dieser genutzt werden. Varianten von AdaBoost, wie „Gentle AdaBoost“ und „Brown-Boost“, wurden entwickelt um das Problem der störenden „Ausreißer“ zu beheben.

4 Anwendung von AdaBoost im Kontext der Gesichtsdetektion

Im Zusammenhang mit der Gesichtsdetektion nach Viola und Jones aus [VJ01, VJ02] erledigt eine Variante von AdaBoost gleich zwei Aufgaben mit einmal:

- Selektion einer angemessen kleinen Menge von Merkmalen aus den 180 000 möglichen Merkmalen
- Training von Klassifikatoren mit den gewählten Merkmalen

Der Vorteil von AdaBoost ist hierbei, angesichts der Menge der Trainingsdaten und zu selektierenden Merkmale, die Geschwindigkeit des Lernens, wobei die Selektion des 100. Merkmals nicht mehr Aufwand benötigt als die des ersten. Welche Merkmale wie und warum berechnet wurden, soll hier nicht erläutert werden und ist den beiden genannten Publikationen von Viola und Jones oder der Ausarbeitung von Stefan Wender in Zusammenhang mit diesem Hauptseminar zu entnehmen. Hier wird lediglich das Training der von Viola und Jones verwendeten Klassifikator-Kaskade betrachtet.

4.1 Training der Klassifikator-Kaskade mit AdaBoost

Die Idee einer Klassifikator-Kaskade zur Gesichtsdetektion, wie Abb. 4 sie zeigt, besteht darin, dass Klassifikatoren zu Beginn der Kaskade sehr einfach und damit entsprechend schnell sind, Klassifikatoren am Ende der Kaskade dagegen komplexer und damit langsamer. Ziel ist es, in den ersten Stufen große Mengen an Nicht-Gesichtern zu verwerfen, um die komplexeren Klassifikatoren am Ende nur auf wenige Daten, also potentiellen Gesichtern, anwenden zu müssen; Einsparungen bei der Rechenzeit für die Gesichtsdetektion sollen das Ergebnis sein (dazu mehr in den Publikationen von Viola und Jones oder in der Ausarbeitung von Stefan Wender).

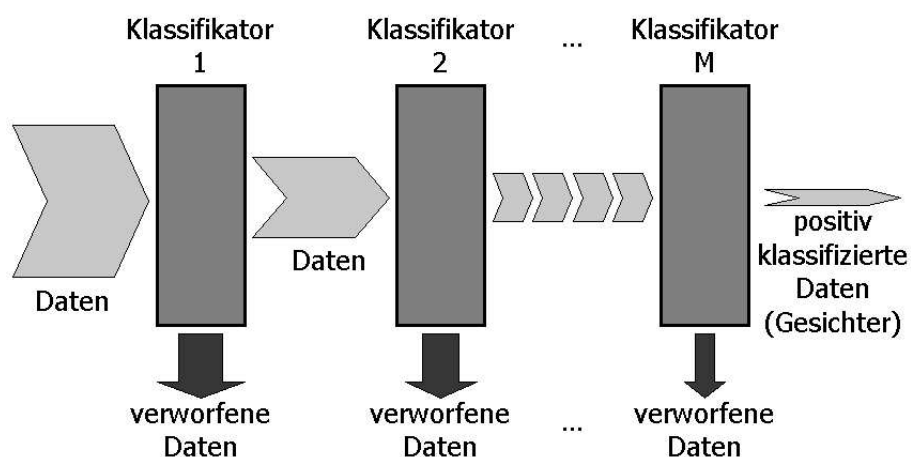


Abbildung 4: Schema einer Klassifikator-Kaskade wie sie zur Gesichtsdetektion genutzt wird (von Stefan Wender)

Jede der Stufen der Kaskade soll einzeln mit AdaBoost trainiert werden, wobei die Möglichkeit besteht, für jede Stufe die gewünschte Güte festzulegen. Es sind also zunächst einfache Klassifikatoren zu liefern, dann für die höheren Stufen komplexere.

Das Training eines Klassifikators der unteren Stufen ist weniger aufwändig, da für dessen Konstruktion nur wenige Boosting-Runden nötig sind. Wie das Training einer Stufe der Kaskade abläuft soll folgender vereinfachter Ablauf aus [VJ01] zeigen:

- Geg.:
 - n Datensätze (x_i, y_i) mit $y_i = 0$ bzw. 1 für Gesichter bzw. Nicht-Gesichter
 - $m =$ Anzahl der Gesichter, $l =$ Anzahl der Nicht-Gesichter
 - Merkmale j
- Datensätze $1 \dots n$ werden initial mit $w_{1,i} = \frac{1}{2m}$ (Datensatz i ist Gesicht) bzw. $\frac{1}{2l}$ (Datensatz i ist Nicht-Gesicht) gewichtet
- Für $t = 1, \dots, T$

- Normiere die Gewichte

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

- Trainiere einen Klassifikator h_j für jedes Merkmal j und ermittle alle Fehler ϵ_j unter Berücksichtigung der Gewichte der Datensätze

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$$

- Wähle aus den h_j den Klassifikator mit dem geringsten Fehler $\rightarrow h_t$
- Passe die Gewichte für alle Datensätze x_i ($i = 1 \dots n$) an

$$w_{t+1,i} = \begin{cases} w_{t,i} \beta_t & \text{falls } x_i \text{ korrekt klassifiziert wurde} \\ w_{t,i} & \text{falls } x_i \text{ falsch klassifiziert wurde} \end{cases}$$

mit $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$

- Der finale Klassifikator $h(x)$ ergibt sich aus einer gewichteten Mehrheitsentscheidung der Einzelklassifikatoren:

$$h(x) = \begin{cases} 1 & \text{falls } \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{sonst} \end{cases}$$

mit $\alpha_t = \log \frac{1}{\beta_t}$

Im beschriebenen Ablauf wird klar, dass jeder Einzelklassifikator genau ein Merkmal repräsentiert, welches anhand seines Fehlers aus den 180 000 Merkmalen gewählt wird; der finale Klassifikator berechnet also T Merkmale. Für die Kaskade ist es nötig für jede Stufe anzugeben, wie gut der Klassifikator sein soll. Dies bedeutet, dass die Rundenanzahl T nicht von vorn herein fest ist, sondern nach jeder Runde getestet werden muss, ob der finale Klassifikator bereits die vorgegebene Güte erreicht hat. So besteht z. B. der erste Klassifikator bei den Versuchen von Viola und Jones aus lediglich 2 Merkmalen, erkannte jedoch nur 40% der Nicht-Gesichter fälschlich als Gesicht. Dies jedoch bewirkte, dass in der ersten Stufe bereits 60% der Nicht-Gesichter verworfen wurden und den folgenden Stufen weniger Daten präsentiert werden mussten. Für jede Stufe werden also einem Klassifikator mit AdaBoost so viele Merkmale hin-

zugefügt, bis die für diese Stufe gewünschte Güte erreicht ist, wobei Detektions- und Falsch-Positiv-Rate nach dem Training mittels eines Schwellwertes noch verändert werden können (↗ [VJ01]). Der gesamten Kaskade werden so lange Stufen hinzugefügt, bis das Gesamtergebnis den Anforderungen entspricht. Viola und Jones konstruierten mit AdaBoost laut [VJ01] eine 38-stufige Kaskade, die im ganzen 6061 Merkmale umfasste.

4.2 Asymmetric AdaBoost

In [VJ02] beschreiben Viola und Jones ein Problem, welches AdaBoost speziell im Zusammenhang mit dem Kaskaden-Ansatz hat: AdaBoost minimiert zwar den Klassifikationsfehler, jedoch nicht die Anzahl der Falsch-Negativen. Dies ist jedoch gerade bei der Nutzung einer Kaskade ein wesentliches Ziel, da es darum geht auf jeder Stufe so viele Nicht-Gesichter wie möglich zu verwerfen. Die mit Hilfe des in Abschnitt 4.1 angegebenen Algorithmus ausgewählten Merkmale zeigen kein optimales Verhalten beim Zurückweisen von negativen Beispielen. Deshalb entwickelten Viola und Jones ein Verfahren, welches AdaBoost für ihre Zwecke anpasst.

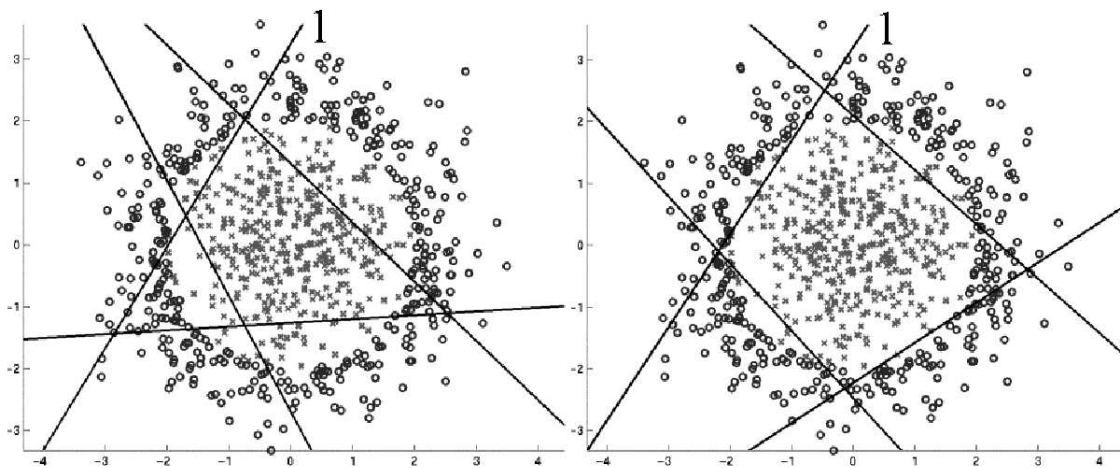


Abbildung 5: [VJ02]: Two simple examples: positive examples are 'x', negative 'o' and weak classifiers are linear separators. On the left is the naive asymmetric result. The first feature selected is labelled '1'. Subsequent features attempt to balance positive and negative errors. Notice that no linear combination of the 4 weak classifiers can achieve a low false positive and low false negative rate. On the right is the asymmetric boosting result. After learning 4 weak classifier the positives are well modelled and most of the negative are rejected.

In einem ersten, trivialen Ansatz modifizierten sie die initiale Gewichtung der Datensätze, indem die Gewichte der positiven Beispiele erhöht wurden. Doch diese asymmetrische Gewichtung bleibt nur bis zur Neugewichtung nach der ersten Rund erhalten (Abb. 5 rechts) und bringt demzufolge nicht die gewünschte Verbesserung. Dies führte Viola und Jones zu einer Variante von AdaBoost, die sie als „AsymBoost“ bezeichnen. Die Neugewichtung der Datensätze erfolgt hier nicht mehr nur in Abhängigkeit vom aktuellen Klassifikationsfehler, sondern auch mit Augenmerk darauf, welcher Klasse der entsprechende Datensatz eigentlich angehört, also ob ein positives oder ein negatives Beispiel falsch klassifiziert wurde. Dazu führten den Voila und Jones den asymmetrischen Fehler $ALoss(i)$ für jeden Datensatz ein, der für jeden Datensatz angibt, ob ein

Fehler und wenn ja, welche Art von Fehler, gemacht wurde. Ausgehend von diesem Ansatz leiten sie einen asymmetrischen Faktor her, der in jeder Runde in die Neugewichtung der Datensätze eingeht. Damit wird erreicht, dass die konstruierten Klassifikatoren möglichst viele Nicht-Gesichter verwerfen; das einfache Beispiel in Abb. 5 verdeutlicht dies. Eine detailliertere Beschreibung vom AsymBoost liefert der vierte Abschnitt der Publikation von Viola und Jones [VJ02].

5 Vorhandene Implementierungen und Demos

Die Recherche zu AdaBoost lieferte drei Implementierungen, die sich entweder direkt mit AdaBoost beschäftigen oder dies als ein Teil des Gesamtsystems umsetzen:

- Java-Applet zur Simulation verschiedener Klassifikatoren
<http://www.cs.technion.ac.il/~rani/LocBoost/index.html>
- Matlab-Implementierung verschiedener Lernverfahren und AdaBoost-Varianten von Gunnar Rätsch <http://mlg.anu.edu.au/~raetsch/Software.html>
- Classification Toolbox for Matlab
<http://tiger.technion.ac.il/~eladyt/classification/index.htm>

Lediglich das Java-Applet wurde in Zusammenhang mit dieser Arbeit näher betrachtet und zu Demonstrationszwecken verwendet; die beiden Matlab-Implementierungen liegen der Arbeit jedoch auf CD-ROM bei.

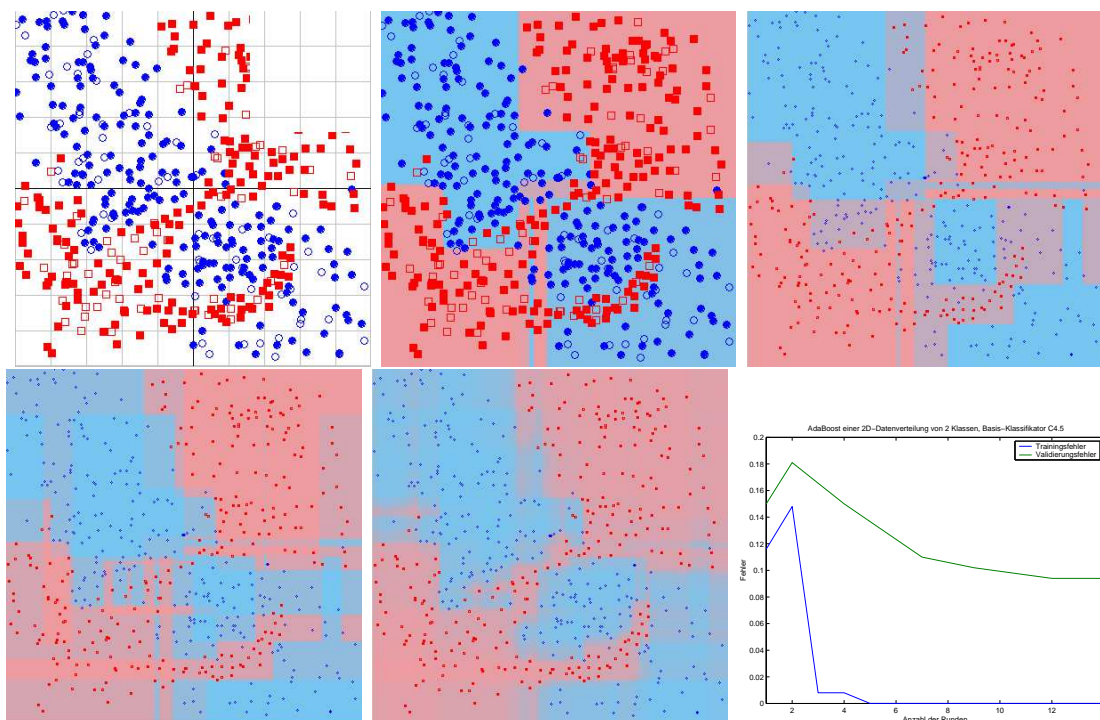


Abbildung 6: Klassifikation der Input-Daten (1. o.) nach 1, 2, 3 und 60 Runden AdaBoosting mit C4.5 als Basisklassifikator. Die Grafik u. r. veranschaulicht den Trainings- und den Validierungsfehler auf diesen Daten.

Bemerkung: Die Größe der Datenpunkte in der grafischen Darstellung (groß: oben rechts/mitte, klein: sonst) ist nicht von Bedeutung und kann im Applet vom Nutzer nach Belieben festgelegt werden.

Das Java-Applet stellt die Simulation verschiedener Klassifikatoren zur Verfügung und veranschaulicht dies graphisch. Eine Zweiklassen-2D-Datenverteilung wird vom Nutzer selbst festgelegt. Zur Klassifikation stehen verschiedene Verfahren zur Verfügung, darunter Meta-Lernverfahren wie AdaBoost und LocBoost. Bei diesen ist es wiederum möglich aus verschiedenen Basisklassifikatoren zu wählen. Abb. 6 zeigt das Boosting von C4.5² mit AdaBoost. Hier wird deutlich, wie AdaBoost mit steigender Rundenzahl das Ergebnis verbessert und den Trainingsfehler exponentiell senkt.

6 Fazit

Die vorliegende Arbeit beschäftigt sich mit dem Lernverfahren AdaBoost und beleuchtet seine Entwicklung und Eigenschaften. Am Beispiel der Gesichtsdetektion wird deutlich, dass ein gewisses Potential auch in der Merkmalsextraktion liegt.

Die Einfachheit der Idee und die seine Variabilität Neuerungen betreffend machen AdaBoost zu einem Lernverfahren, welches in vielen Gebieten Einsatz findet und ständigen Neuentwicklungen unterliegt. Die auf CD-ROM beiliegenden Implementierungen verdeutlichen die Funktionsweise von AdaBoost und geben einen kleinen Einblick in die Möglichkeiten; angegebene Quellen sowie die verschiedenen Angebote des Internets (↗ Webseite auf CD-ROM) geben die Möglichkeit für tiefere Recherche und weiterführende Informationen zum Thema.

²C4.5 ist Lernverfahren, das auf Entscheidungsbäumen basiert

Literatur

- [FS95] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.
<http://citeseer.nj.nec.com/freund95decisiontheoretic.html>.
- [FS99] Yoav Freund and Robert E. Schapire. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 4(5):771780, 1999.
- [KM01] Ralph Kölle and Thomas Mandl. Data Mining Vorlesung. Abschnitt Klassifikation: SVM und Meta-Lernverfahren, 2001.
http://www.uni-hildesheim.de/~mandl/DataMining_SS01/.
- [Kow01] Wojtek Kowalczyk, 2001. Data Mining Course (Vrije Universiteit Amsterdam), Slides Lecture 8.
- [SFBL97] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun 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.
<http://citeseer.nj.nec.com/article/schapire97boosting.html>.
- [Sor99] Udo Sorges. Kooperative Klassifikation, 1999. RWTH-Aachen, Lehrstuhl für Informatik VI.
- [SS98] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational Learning Theory*, pages 80–91, 1998.
<http://citeseer.nj.nec.com/article/singer99improved.html>.
- [VJ01] P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.
<http://citeseer.nj.nec.com/480926.html>.
- [VJ02] P. Viola and M. Jones. Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.