Verbesserungen des Perzeptron-Algorithmus

Improving the Perceptron-Algorithm

Bachelor-Thesis von Tobias Krönke aus Frankfurt am Main

Juni 2011

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Verbesserungen des Perzeptron-Algorithmus
Improving the Perceptron-Algorithm

Vorgelegte Bachelor-Thesis von Tobias Krönke aus Frankfurt am Main

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 30. Juni 2011

(T. Krönke)

Inhaltsverzeichnis

Teil I.
Einführung

Der Perzeptron-Algorithmus als binärer linearer Klassifizierer erfreut sich wachsender Beliebtheit. In letzter Zeit wurden daher einige Arbeiten veröffentlicht, die sich damit befassen, den Perzeptron-Algorithmus mit unterschiedlichen Ansätzen zu erweitern und zu verbessern: [CDK+06], [KW07], [WCV10], [TST07], [WV10] und [LZH+02]. Hierzu gehören verbesserte Modell-Updates, Margins und Batch-Lernen erweiterter Modelle, um die Performanz bei verrauschten Daten zu erhöhen, sowie Reduzierung der Epochen und Support-Vektoren, um den Lern- und Klassifizierungsprozess zu beschleunigen.

1 Aufgabenstellung

Aufgabe dieser Arbeit ist es, zu untersuchen, inwieweit sich diese Erweiterungen sinnvoll und effektiv kombinieren lassen und welche Vorteile sie bei der Klassifizierung realer Daten bringen. Die Implementierung wird in Weka [HFH+09] integriert, was eine einfache Kombination der einzelnen Verbesserungen mittels graphischer Oberfläche ermöglicht. Dabei unterstützt Weka den einheitlichen Vergleich im Experimentalteil.

1.1 Überblick

Kapitel 2 bildet eine Grundlage für die in Teil II vorgestellten Verbesserungen des Perzeptrons, die in Teil III bezüglich ihrer Kombinationsfähigkeit empirisch evaluiert werden. Viele Erweiterungen sind durch die Supportvektormaschine motiviert, weshalb diese als Referenzmodell dient. Daraus wird abschließend in Kapitel 7 ein Fazit mit möglichem Ausblick gezogen.

2 Grundlagen des Perzeptrons

Dieses Kapitel stellt einige Grundlagen vor, um ein einheitliches Verständnis vom Perzeptron in dieser Arbeit sicherzustellen. Dazu gehören die mathematische Notation und ausgewählte Grundlagen aus dem maschinellen Lernen. Es wird versucht, Grundlagen, die über die des Perzeptrons dieser Arbeit hinausgehen, wegzulassen. Wissen über diesen Tellerrand hinaus kann jedoch nicht schaden, um das Anwendungsgebiet des Perzeptrons in die vielfältigen Konzepte maschinellen Lernens einzuordnen.

2.1 Notation

Wenn im Einzelfall nicht anders erklärt, dient die Notation aus Tabelle 2.1 als Konvention für alle mathematischen Konstrukte in Formeln und Algorithmen. Sie ähnelt der Notation aus [Bis07]. Falls nicht anders angegeben, ist der Wertebereich kleingeschriebener Variablen die Menge der natürlichen Zahlen von 1 bis zum Wert der entsprechenden großgeschriebenen Konstante.

Beispiel	Erklärung
N	Großbuchstaben für Konstanten
$\mathbf{x}$	Spaltenvektor: fett gedruckte Kleinbuchstaben
$\mathbf{M}$	Matrix: fett gedruckte Großbuchstaben
$\mathbf{x}^{\mathrm{T}}, \mathbf{M}^{\mathrm{T}}$	Hochgestelltes T transponiert
$\mathbf{w}^{\mathrm{T}} = (w_1, \ldots, w_D)$	Elemente eines Zeilenvektors mit D Elementen
$\mathbf{w}^{\mathrm{T}} = (w_d)$	Abkürzung für Elemente eines Zeilenvektors mit D Elementen
$\mathbf{X} = \left[\mathbf{x}_n\right]$	Matrix aus N Spaltenvektoren
$\langle \mathbf{w}, \mathbf{x} \rangle$	Skalarprodukt $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ der Vektoren $\mathbf{w}$ und $\mathbf{x}$
$\mathbf{0}$	Null-Vektor mit Dimension entsprechend dem Kontext
$\mathbf{I}$	Identitätsmatrix mit Dimension entsprechend dem Kontext
$\mathcal{M}$	Mengen und andere abstrakte Konstrukte

Tabelle 2.1.: Mathematische Konventionen

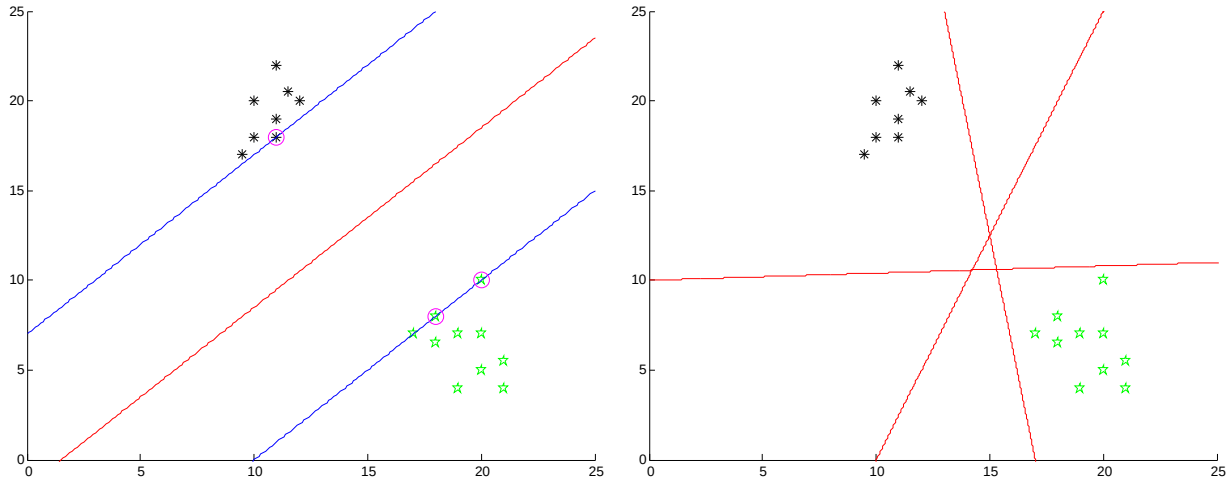
2.2 Motivation im Kontext maschinellen Lernens

Ein lernendes System ist nach [Mit97] definiert durch die „Erfahrung“, aus der gelernt wird, der gewünschten „Zielfunktion“, ein mathematisches Modell $f_{\mathcal{W}}$ mit Modellparametern $\mathcal{W}$, diese zu „repräsentieren“ und einen Algorithmus, $f_{\mathcal{W}}$ durch Lernen von $\mathcal{W}$ an die „Zielfunktion anzunähern“. Ziel ist es, eine „Aufgabe“ $\mathcal{A}$ mit „Erfahrung“ $\mathcal{Z}$ gemäß „Performanzmaß“ $\mathcal{P}$ möglichst gut zu lösen. Abbildung 2.1 soll als konstruiertes Beispiel dienen, um dies exemplarisch weiter zu erörtern. Die zwei reellen Merkmale seien charakteristische Eigenschaften von Pilzen, mit denen es Experten möglich ist, zu entscheiden, ob sie giftig oder ungiftig sind. Mit einer zweidimensionalen Trennlinie scheint dies problemlos möglich. Welche Trennlinie die beste ist, um z. B. unerfahrene Sammler mit einer mobil einsetzbaren Klassifizierungsmaschine zu unterstützen hängt stark vom Performanzmaß ab. Es sollte z. B. großen Wert darauf gelegt werden, keine giftigen Pilze als ungiftig zu deklarieren, um die Gesundheit des Pilzesammlers nicht zu gefährden.

Für diese Arbeit wird angenommen, dass in der Zielfunktion jedem Objekt $\mathbf{x} \in \mathcal{X}$ genau ein $y \in \mathcal{Y}$ zugeordnet ist. Somit ist $f_{\mathcal{W}}$ eine Funktion $f_{\mathcal{W}} : \mathcal{X} \mapsto \mathcal{Y}$. Erfordert $\mathcal{A}$ die Zuordnung von Eingabedaten aus $\mathcal{X} \subseteq \mathbb{R}^S$ zu Zielwerten (Klassen) aus $\mathcal{Y} = \{-1, +1\}$ (binäre Klassifikation) und sind die Eingabedaten gemäß ihrer Klassifikation möglichst fehlerfrei linear separierbar, kann das mathematische Modell linearer Trennebenen eine gute Wahl sein. So ist der Bayes-optimale Klassifizierer zweier korrelierter Normalverteilungen mit identischer, invertierbarer Kovarianz nach [Hay08] eine lineare Hyperebene. Ein lineares Modell kann also die bestmögliche Genauigkeit erreichen, obwohl die Daten – wie bei Normalverteilungen üblich – im Merkmalsraum überlappen.

Das Perzeptron, wie auch die Supportvektormaschine (SVM), sind solche linearen Klassifizierer. Sie unterscheiden sich hauptsächlich in der Art und Weise, wie sie gelernt werden: Während bei der SVM die möglichst fehlerfrei

trennende Hyperebene mit dem größten *margin* ‚Abstand' zu allen Datenpunkten gesucht wird, woraus ein aufwendig zu lösendes quadratisches Optimierungsproblem resultiert [Bis07], wird beim Perzeptron über alle Daten iteriert und bei jeder Fehlklassifizierung das Modell ein bisschen angepasst. Eine Iteration über alle Daten wird *Epoche* genannt. Außer einer reinen Trennung der Daten werden keine weiteren Anforderungen an das Ergebnis gestellt. Dies wird in Abbildung 2.1 illustriert.



(a) Eindeutig definierte SVM

(b) Mögliche Modelle des Perzeptrons

Abbildung 2.1.: Die Trennebene (rot) der SVM hat bei korrekter Klassifizierung den maximalen Abstand (blau) zu allen Datenpunkten. Für das Perzeptron kommen hingegen mehrere Lösungen in Frage.

Aufgaben mit $|\mathcal{Y}| > 2$ werden nicht gesondert betrachtet und können mittels *one-against-all* bzw. *one-against-one* auf binäre Klassifikation reduziert werden. [Bis07]

2.2.1 Lernen durch Erfahrung

Der Wertebereich von Erfahrung $\mathcal{Z}$ ist bereits in Abschnitt 2.2 festgelegt. Doch in welcher Form $\mathcal{Z}$ beim Lernprozess verfügbar ist, ist noch unklar. Die behandelten Algorithmen befassen sich mit den folgenden Lernszenarien und -strategien.

Überwacht

Beim überwachten Lernen ist die Klassifikation sämtlicher Trainingsdaten beim Lernprozess bekannt. Sie bestehen daher aus Paaren $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. Halb- und unüberwachte Lernverfahren werden nicht behandelt. Die Verfügbarkeit der einzelnen Trainingsbeispiele wird wie folgt unterteilt.

Online

Beim Online-Lernen steht ein konstanter Strom von Trainingsdaten zur Verfügung. Das zu lernende Modell wird dabei aufgrund der neu eintreffenden Daten stets weiter angepasst. Bereits erhaltene ältere Daten werden oft nicht erneut verwendet und müssen daher nicht gespeichert werden.

Batch

Batch-Lernen setzt voraus, dass zu Trainingsbeginn alle Daten, auf denen gelernt werden soll, bekannt sind. Weitere Beispiele sind nicht zu erwarten und es kann ein finales Modell angegeben werden, sofern der Lernalgorithmus stets terminiert.

2.3 Das Perzeptron

2.3.1 Generalisiertes Modell

Wie bereits erwähnt, stellt das Perzeptron eine lineare Trennebene

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{2.1}$$

dar. Gemäß [Bis07] lässt sich dieses Modell mit Hilfe nicht-linearer Transformationsfunktionen $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^D$ generalisieren: statt $\mathbf{x}$ wird nun stets $\boldsymbol{\phi}(\mathbf{x})$ als Featurevektor verwendet. Ein höherdimensionaler Merkmalsraum kann die Separierbarkeit verbessern, das Skalarprodukt benötigt jedoch $\mathcal{O}(D)$ Multiplikationen und Additionen. Damit wird die Laufzeit- und/oder Speicherkomplexität linear mit D erhöht. In Untersektion 2.3.3 wird gezeigt, wie man mit Kernelfunktionen auf die explizite Auswertung von $\boldsymbol{\phi}(\mathbf{x})$ verzichten kann. Zur einfacheren Notation wird der Bias b der Trennebene in $\mathbf{w}$ als w_D augmentiert, indem $\phi_D(\mathbf{x}) = R$ für $R \in \mathbb{R}^+$ fix, aber beliebig gesetzt wird. Dies ergibt das zu trainierende Modell

$$f_{\mathbf{w}}(\mathbf{x}) = sign\left(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle\right) \tag{2.2}$$

mit der Vorzeichenfunktion

$$sign(a) = \begin{cases} -1 & \text{falls } a \leq 0 \\ +1 & \text{sonst} \end{cases}, \tag{2.3}$$

wobei standardmäßig $R = 1$ gilt. Neben der hier verwendeten Vorzeichenfunktion werden keine weiteren Aktivierungsfunktionen betrachtet. Offensichtlich ist die Wahl der Klasse für $\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle = 0$ willkürlich und wird mit der Einführung der Erweiterungen aus [KW07] und [LZH+02] während des Lernprozesses stets als Fehler betrachtet werden. Tatsächlich hat dies jedoch keine Auswirkungen auf die Konvergenz des nun folgenden Lernalgorithmus.

2.3.2 Lernalgorithmus

Nach [Bis07] ist ein $\mathbf{w}$ gesucht, sodass für alle Trainingsbeispiele $(\mathbf{x}_n, y_n)$ gilt, dass $y_n \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle > 0$. Sei $\mathcal{M}$ die Indexmenge aller von $f_{\mathbf{w}}$ falsch klassifizierten Beispiele. Korrekt klassifizierte werden ignoriert (Fehler von 0). Für die resultierende Fehlerfunktion

$$e_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} y_n \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle, \tag{2.4}$$

„Perzeptron-Kriterium“ genannt [Bis07], ergibt sich der ursprüngliche Perzeptron-Algorithmus (Algorithmus 2.1) von Rosenblatt [Ros62] mittels Minimierung von e_P durch stochastische[1] (Zeile 4) Gradientensuche (Zeilen 5 & 6). [Bis07] Der Algorithmus iteriert also über alle Trainingsdaten und addiert die fehlklassifizierten Beispiele entsprechend ihrer korrekten Klassifikation zum Modell hinzu. Die Lernrate η steuert die Stärke einer solchen Anpassung. Algorithmus 2.1 terminiert, sobald er alle Beispiele innerhalb einer Iteration (Epoche) korrekt klassifiziert.

Dass das Update in Zeile 6 mit $\eta = 1$ den Teilfehler $-y_n \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle$ auf dem aktuellen Featurevektor $\boldsymbol{\phi}(\mathbf{x}_n)$ tatsächlich verringert, wird in [Bis07] mittels

$$-y_n \langle \mathbf{w}^{(t+1)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle = -y_n \langle \mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle - \langle y_n \boldsymbol{\phi}(\mathbf{x}_n), y_n \boldsymbol{\phi}(\mathbf{x}_n) \rangle < -y_n \langle \mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle \tag{2.5}$$

begründet. Dies liefert jedoch nur eine Intuition für den Algorithmus. Das stochastische Update kann den Fehler für andere – sogar bisher korrekt klassifizierte – Beispiele wieder erhöhen.

[1] Updates anhand einzelner falsch klassifizierter Trainingsbeispiele

Algorithmus 2.1 Ursprünglicher Perzeptron-Algorithmus

Benötige: Unter $\boldsymbol{\phi}(\cdot)$ linear separierbare Trainingsdaten $\mathcal{Z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subseteq \mathcal{X} \times \{-1, +1\}$
Benötige: Augmentierung $R \in \mathbb{R}^+$
Benötige: Transformation $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^D$ mit $\phi_D(\mathbf{x}) = R$
Benötige: Lernrate $\eta \in \mathbb{R}^+$
1: $t \leftarrow 0$
2: $\mathbf{w}^{(t)} \leftarrow \mathbf{0}$
3: **wiederhole**
4: **for** $n \leftarrow 1$ **to** N **do**
5: **if** $f_{\mathbf{w}^{(t)}}(\mathbf{x}_n) \neq y_n$ **then**
6: $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta y_n \boldsymbol{\phi}(\mathbf{x}_n)$
7: $t \leftarrow t + 1$
8: **end if**
9: **end for**
10: **bis** sich t nicht mehr ändert
11: $T \leftarrow t$
12: **return** $\mathbf{w}^{(T)}$

Konvergenz

Warum der Algorithmus auf linear separierbaren Daten dennoch stets nach endlicher Anzahl Updates terminiert, ist mit Novikoffs Theorem [Nov62] begründet:

Theorem 1. *Sei die Trainingsmenge $\mathcal{Z}$ definiert wie in Algorithmus 2.1 mit mindestens einem positiven und einem negativen Beispiel. Setze $K \equiv \max_n \|\mathbf{x}_n\|$. Unter der Annahme, dass $\mathcal{Z}$ linear separierbar ist, existiert ein mit 1 augmentiertes Gewicht $\mathbf{w}^\star$ mit $\|\mathbf{w}^\star\| = 1$, sodass $\gamma_n = y_n \langle \mathbf{w}^\star, \mathbf{x}_n \rangle \geq \gamma$ für alle Beispiele n. Dann benötigt Algorithmus 2.1 nicht mehr als $(K/\gamma)^2$ Anpassungen, also gilt $T \leq (K/\gamma)^2$.*

Wird das Modell mit $\mathbf{0}$ initialisiert, wird mit der Lernrate das Modell lediglich skaliert. Für den Beweis wird deshalb vereinfachend $\eta = 1$ gesetzt. Es lässt sich zeigen, dass $\|\mathbf{w}^{(T)}\|^2$ einerseits nach oben durch TK^2, andererseits nach unten durch $T^2\gamma^2$ beschränkt ist.

2.3.3 Dualform und Kerneltrick

Obwohl meist in Primalform angegeben, werden alle Algorithmen auch in Dualform implementiert, um vom sogenannten Kerneltrick Gebrauch machen zu können. Hierbei wird das Modell $\mathbf{w}$ nicht als Vektor, sondern als gewichtete Summe

$$\mathbf{w}_\mathbf{a}^{(t)} = \sum_{n=1}^{N} a_n^{(t)} \boldsymbol{\phi}(\mathbf{x}_n) \tag{2.6}$$

aller Featurevektoren gespeichert, wobei die Koeffizienten $a_n^{(t)}$ die Updates auf Beispiel $\mathbf{x}_n$ (Zeile 6) bis zum Zeitpunkt t aufsummieren. Somit gilt nach [LZH+02] intuitiv $a_n^{(t)} \propto \eta y_n$. Der Algorithmus beruht nunmehr lediglich auf dem Skalarprodukt zweier Featurevektoren, welches nach [GHW00] durch „Mercer-Kernel"-Funktionen [CST00] $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ ersetzt werden kann:

$$\langle \mathbf{w}_\mathbf{a}^{(t)}, \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{n=1}^{N} a_n^{(t)} \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{n=1}^{N} a_n^{(t)} k(\mathbf{x}_n, \mathbf{x}). \tag{2.7}$$

Die nicht-lineare Transformation kann nun implizit in k vorgenommen werden. Ein beliebtes Beispiel hierfür ist der Polynomial-Kernel

$$k_{poly}(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + const)^p, \tag{2.8}$$

welcher für $const > 0$ implizit alle Terme über x_c und x_c' vom Grad $\leq P$ enthält. [Bis07] Es entfällt der Aufwand, diese manuell mit $\boldsymbol{\phi}$ zu erzeugen. Für den ebenfalls sehr beliebten RBF[2]-Kernel

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \tag{2.9}$$

divergiert D implizit sogar gegen ∞ [Bis07], was sich mittels expliziter Transformation natürlich nur annähern ließe.

[2] Radial-Basis-Funktion

Teil II.

Erweiterungen zum Perzeptron-Algorithmus

Der größte Nachteil von Algorithmus 2.1 liegt in der Natur der stochastischen Updates begraben. Zum einen entspricht jede Epoche für sich einem live lernenden Algorithmus (Abschnitt 2.2.1), ohne einen Vorteil daraus zu ziehen, dass alle Daten stets vorhanden sind. Zum anderen trifft der Algorithmus keine Annahmen über die stochastische Verteilung der Daten. Insbesondere verrauschte und/oder überlappende Klassen bereiten ihm Probleme, da er niemals terminiert. Die Qualität des aktuellen Modells bei künstlichem Abbruch nach I Epochen ist ungewiss. Aber auch bei separierbaren Daten wird lediglich „irgendeine“ Lösung gefunden. Die Verbesserungen der beiden folgenden Kapitel werden anschließend in einer Übersicht in Kapitel 5 zusammengeführt.

3 Batch-Learning ausnutzen

Dieses Kapitel stellt Möglichkeiten vor, wie das Perzeptron die Verfügbarkeit aller Trainingsdaten beim Lernen zu seinem Vorteil nutzen kann. Die Ansätze aus Abschnitt 3.1–3.3 werden zu einem Algorithmus in Abschnitt 3.4 vereint und verallgemeinert, welcher die Grundlage für die Erweiterungen aus Kapitel 4 bildet.

3.1 Am längsten überlebendes Perzeptron

Standardmäßig wird das letzte Modell $\mathbf{w}^{(T)}$ retourniert. Ohne natürliche Terminierung des Algorithmus ist dies jedoch nicht einmal auf den Trainingsdaten zwangsweise das beste Modell. Um ein Mindestmaß an Qualität zu garantieren, wird nun das Modell ausgegeben, welches beim Lernprozess die meisten Beispiele korrekt klassifiziert. Es ist das *longest surviving* Modell nach [KW07], welches die meisten Beispiele überlebt, bevor es angepasst wird.

3.2 Abstimmende Perzeptronen

Ähnlich wie beim *longest survivor* gibt sich die Variante der *voting perceptrons* [FS99], ebenfalls von [KW07] vorgeschlagen, nicht mit dem zuletzt gefundenen Modell zufrieden. Sie weist jedem der für $1 \leq t \leq T$ erzeugten $\mathbf{w}^{(t)}$ ein Stimmrecht in Höhe der Anzahl von $\mathbf{w}^{(t)}$ beim Lernen korrekt klassifizierter Beispiele zu. Das finale Modell ist hierbei der nach Stimmrecht gewichtete Durchschnitt der Vorhersagen aller $\mathbf{w}^{(t)}$. In der Dualform lässt sich dieses Modell durch Zwischenspeicherung und Wiederverwendung der Kernelprodukte ohne große Steigerung der Laufzeitkomplexität implementieren. [KW07]

3.3 Reduzierte Epochen

Bei den *reduced epochs* aus [TST07] handelt es sich um eine Heuristik zur Reihenfolge, in der die Trainingsdaten dem Lernalgorithmus präsentiert werden. Jeder vollen Epoche folgen bis zu H reduzierte Epochen auf jenen Featurevektoren, die in der jeweils vorangegangenen (reduzierten) Epoche zu einer Änderung des Modellvektors geführt haben. Bei linear separierbaren Daten hat dies offensichtlich keine negativen Auswirkungen auf die Konvergenzeigenschaften des Algorithmus. Die Hoffnung, durch die intensive Behandlung der Problemfälle, schneller gegen eine gute Lösung zu konvergieren, könnte jedoch durch verrauschte *outliers* ‚Außenseiter in den Daten' zerschlagen werden: statt wiederholter Updates, wäre es u. U. besser, sie zu ignorieren. Siehe hierzu beispielhaft Abbildung 3.1.

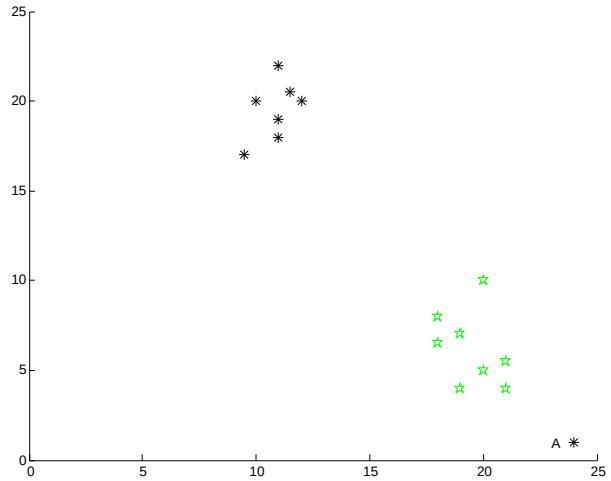


Abbildung 3.1.: Ein lineares Modell im dargestellten Raum, welches Beispiel A korrekt klassifiziert, macht entweder einen Fehler auf allen grünen oder auf den verbleibenden schwarzen Daten. A könnte jedoch aus einer verrauschten Klassifikation [CDK+06] stammen.

3.4 Zusammenfassung

Die bislang vorgestellten Erweiterungen sind in Algorithmus 3.1 zusammengefasst. Die volle Epoche bei $h = 0$ entspricht einer reduzierten Epoche auf allen Beispielen. Im Hinblick auf die nachfolgenden Verbesserungen in Kapitel 4, die sich hauptsächlich damit befassen, unter welchen Bedingungen und wie stark das Modell bezüglich des aktuellen Trainingsvektors angepasst wird, wird dieser Lernschritt in eine Update-Prozedur ausgelagert. Sie benötigt das letzte Modell, das aktuelle Trainingsbeispiel, die entsprechende Ausgabe und die Anzahl der bisherigen Anpassungen. Für den Standard-Algorithmus mit erzwungener Anpassung bei $o = 0$ ist sie durch Prozedur 3.2 gegeben. Ein neues Gewicht $\mathbf{w}^{(t+1)}$ aufgrund einer Anpassung in Zeile 10 erhält zu Beginn eine Stimme $v^{(t+1)} = 0$. Für jedes weitere Beispiel, für das $\mathbf{w}^{(t+1)}$ nicht angepasst werden muss (Zeile 15), wird seine Stimme $v^{(t+1)}$ um 1 erhöht. Als Anpassung zählt dabei nicht eine bloße Skalierung von $\mathbf{w}^{(t)}$ um einen Faktor $r \in \mathbb{R}^+$, da dies die Entscheidungsebene nicht verändert.

Wie die Stimmen je nach Variante verwendet werden, wird in Tabelle 3.1 formal erläutert. Das am längsten überlebende Perzeptron ist das Gewicht mit den meisten Stimmen. Die Vorhersage des abstimmenden Modells ist die Klassifizierung mit den meisten Stimmen aller zurückgegebenen Gewichte.

Algorithmus 3.1 Perzeptron-Algorithmus mit Batch-Erweiterungen

Benötige: Trainingsdaten $\mathcal{Z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subseteq \mathcal{X} \times \{-1, +1\}$
Benötige: Maximal H reduzierte Epochen pro einer von höchstens I Epochen
Benötige: Augmentierung $R \in \mathbb{R}^+$
Benötige: Transformation $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^D$ mit $\phi_D(\mathbf{x}) = R$
Benötige: Update-Prozedur update

1: $\mathbf{u} \leftarrow \mathbf{0}$
2: $t \leftarrow 0$
3: $\mathbf{w}^{(t)} \leftarrow \mathbf{0}$
4: $v^{(t)} \leftarrow 0$
5: **wiederhole**
6: $\quad\mathcal{N} \leftarrow \{1, \ldots, N\}$ // Indexmenge der Trainingsdaten
7: $\quad$**for** $h \leftarrow 0$ **to** H **do**
8: $\quad\quad$**for all** $n \in \mathcal{N}$ **do**
9: $\quad\quad\quad o \leftarrow \langle \mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle$ // Ausgabe ohne Aktivierung
10: $\quad\quad\quad \mathbf{w}^{(t+1)} \leftarrow \mathsf{update}(\mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n), y_n, o, u_n, t)$ // Abstrahierter Lernschritt
11: $\quad\quad\quad$**if** $\exists r \in \mathbb{R}^+ : \mathbf{w}^{(t+1)} \neq r\mathbf{w}^{(t)}$ **then**
12: $\quad\quad\quad\quad u_n \leftarrow u_n + 1$ // Zähle Updates auf Beispiel n
13: $\quad\quad\quad\quad v^{(t+1)} \leftarrow 0$ // Initialstimme für neues Gewicht
14: $\quad\quad\quad\quad t \leftarrow t + 1$
15: $\quad\quad\quad$**else**
16: $\quad\quad\quad\quad \mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t+1)}$ // Behalte u. U. skaliertes Gewicht
17: $\quad\quad\quad\quad v^{(t)} \leftarrow v^{(t)} + 1$ // Erhöhe Stimmrecht
18: $\quad\quad\quad\quad \mathcal{N} \leftarrow \mathcal{N} \setminus \{n\}$ // Reduzierte Epochen ohne Beispiel n
19: $\quad\quad\quad$**end if**
20: $\quad\quad$**end for**
21: $\quad$**end for**
22: **bis** sich t nicht mehr ändert, höchstens I Durchläufe
23: $T \leftarrow t$
24: **return** $\{(\mathbf{w}^{(1)}, v^{(1)}), \ldots, (\mathbf{w}^{(T)}, v^{(T)})\}$

Standard	Längster Überlebender	Gewichtete Abstimmung
$f_{\mathbf{w}^{(T)}}(\mathbf{x})$	$f_{\mathbf{w}^{(t^\star)}}(\mathbf{x})$ für $t^\star = \underset{t \in \{1,\ldots,T\}}{\arg\max}\, v^{(t)}$	$sign\left(\sum_{t=1}^{T} v^{(t)} f_{\mathbf{w}^{(t)}}(\mathbf{x})\right)$

Tabelle 3.1.: Vorhersage-Modell für neues Beispiel $\mathbf{x}$ je nach Batch-Variante

Prozedur 3.2 Standard Update-Prozedur

Benötige: $\mathbf{w}, \mathbf{x}, y, o, u, t$
Benötige: Lernrate $\eta \in \mathbb{R}^+$
1: **if** $yo \leq 0$ **then**
2: **return** $\mathbf{w} + \eta y \mathbf{x}$
3: **else**
4: **return** $\mathbf{w}$
5: **end if**

3.4.1 Ausblick

Algorithmus 3.3 stellt auf sehr reduzierte, abstrakte Weise in Aussicht, wie die nun folgenden Erweiterungen eingebunden werden.

Algorithmus 3.3 Perzeptron-Algorithmus stark abstrahiert

Benötige: Trainingsdaten
1: Initialisierung
2: **for all** Epochen **do**
3: **for all** Trainingsdaten **do**
4: Update als Aufruf der Kette (4.23) von Erweiterungen
5: λ-Trick 4.2: garantiert Separierbarkeit
6: α-Bound 4.3: limitiert die Updates pro Beispiel
7: Budget auf Supportvektoren (SVs) 4.5: forciert maximale Anzahl an SVs
8: Margin 4.4, 4.6.3: für Lernschritt
9: Lernschritt 4.6: Anpassung des Modells
10: Verwalte Stimmrecht
11: **end for**
12: **end for**
13: **return** generierte Modelle

4 SVM-inspirierte Verbesserungen

Alle Erweiterungen dieses Kapitels lassen sich in eine update-Prozedur, wie sie in Algorithmus 3.1 verwendet wird, integrieren. Um ihren Bezug zur Supportvektormaschine (SVM) besser nachvollziehen zu können, ist es sinnvoll, zunächst SVMs und deren Herleitung unter 4.1 kurz zu motivieren. Die Erweiterungen in 4.2–4.6 werden anschließend unter 4.7 in eine sinnvolle Reihenfolge gebracht.

4.1 Die Supportvektormaschine (SVM)

Dieser Überblick ist größtenteils aus [Bis07] übernommen. Wie das Perzeptron auch, ist die SVM ein reiner Klassifizierer ohne Wahrscheinlichkeitsangabe in Form eines linearen Modells[1]

$$s_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle + b. \tag{4.1}$$

Für die resultierende Hyperebene $s_{\mathbf{w},b}(\mathbf{x}) = 0$ ergibt sich für den Vektor $\mathbf{x}$ mit Klassifikation y die Entfernung $|s_{\mathbf{w},b}(\mathbf{x})|/\|\mathbf{w}\| = y s_{\mathbf{w},b}(\mathbf{x})/\|\mathbf{w}\|$ zur Ebene bei korrekter Klassifizierung. [Bis07] Für separierbare Trainingsdaten $\mathcal{Z}$ wie in Algorithmus 2.1 wird nun die Trennebene gesucht, die alle Beispiele korrekt klassifiziert und dabei den kleinsten senkrechten Abstand zum nächsten Vektor maximiert. Auf ambitionierte Begründungen aus der „*statistical learning theory*“ [Bis07], warum dieser Ansatz die Daten gut generalisiert, wird verzichtet. Aus der gewünschten Maximierung des minimalen Abstands ergibt sich das Minimierungsproblem

$$\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \tag{4.2}$$

unter der Bedingung, dass

$$y_n s_{\mathbf{w},b}(\mathbf{x}_n) \geq 1 \quad \forall n. \tag{4.3}$$

Intuitiv verlangt (4.3), dass alle Trainingsbeispiele korrekt klassifiziert werden und einen Mindestabstand von 1 zur Hyperebene einhalten. Die Wahl von 1 für diesen *margin* ist beliebig, da sich die Parameter $\mathbf{w}, b$ beliebig um einen Faktor skalieren lassen. Bei eingehaltenem *margin* sorgt (4.2) nun für die Maximierung desselben. Diese harte Anforderung an den *margin* lässt sich mit Hilfe sogenannter *slack variables* ‚Schmutzvariablen‘ $\xi_n \geq 0$ für jeden Eingabevektor aufweichen. Hält $\mathbf{x}_n$ den *margin* nicht ein, oder wird er gar falsch klassifiziert, so gilt $\xi_n = |y_n - s_{\mathbf{w},b}(\mathbf{x}_n)|$, sonst $\xi_n = 0$. (4.2) und (4.3) werden nun durch

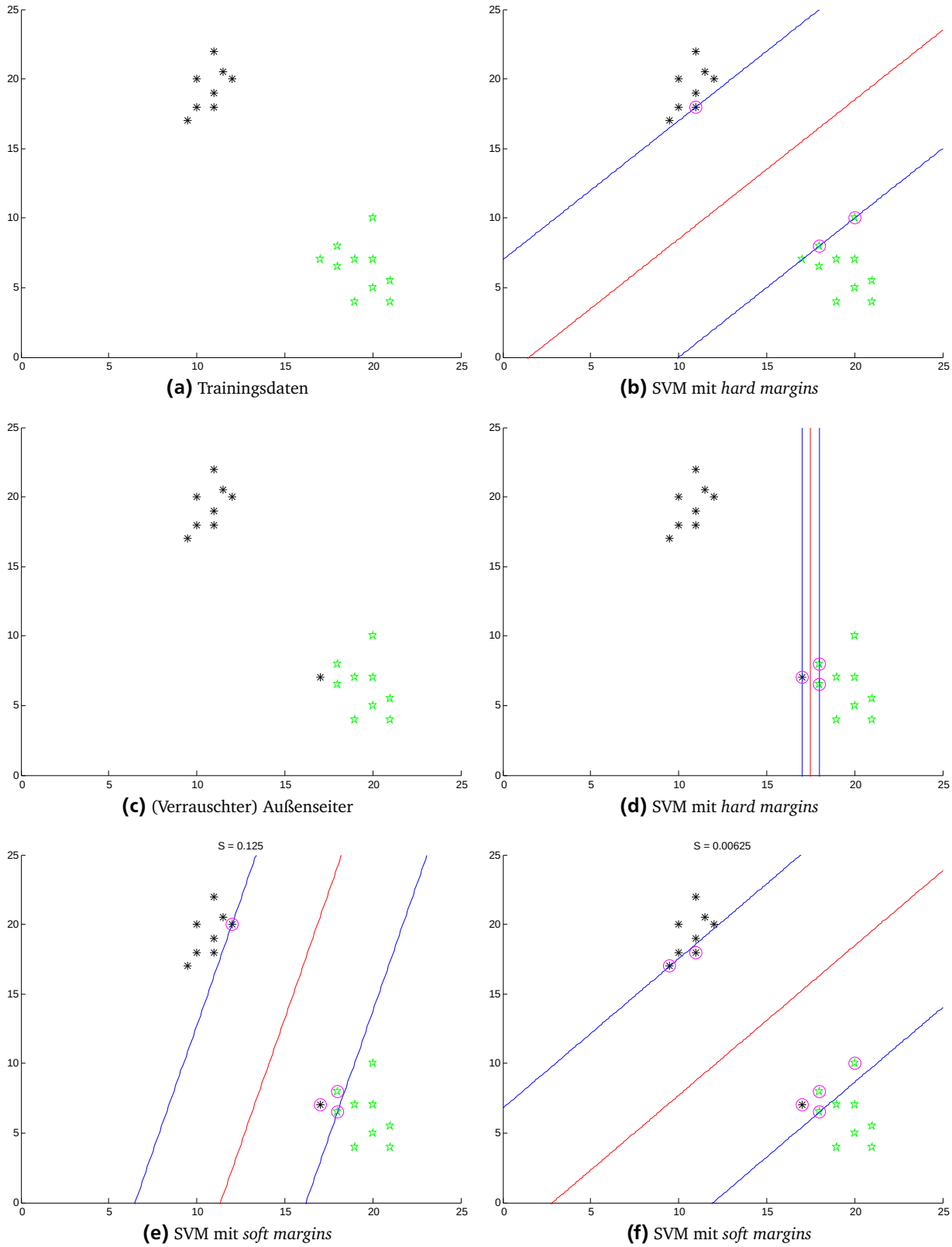
$$\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n \tag{4.4}$$

und

$$y_n s_{\mathbf{w},b}(\mathbf{x}_n) \geq 1 - \xi_n \quad \forall n \tag{4.5}$$

ersetzt, was erlaubt, mit Parameter $C > 0$ die Stärke der Missachtung des *margins* zu regulieren. Abbildung 4.1 zeigt, dass dies auch für separierbare Daten sinnvoll sein kann. Ohne die Lösung dieses Problems anzugeben, sei angemerkt, dass das resultierende Modell in seiner Dualform von jenen Featurevektoren $\boldsymbol{\phi}(\mathbf{x}_n)$ aufgespannt wird, welche den *margin* berühren ($y_n s_{\mathbf{w},b}(\mathbf{x}_n) = 1$) oder gar verletzen ($\xi_n > 0$). Sie werden *Supportvektoren* (SVs) genannt, da sie den *margin* ‚stützen‘.

[1] b wird für SVMs explizit angegeben und berechnet, wird aber bei den Erweiterungen mittels Augmentierung in (2.2) gemäß [SSS07] ignoriert werden.



(a) Trainingsdaten **(b)** SVM mit *hard margins*

(c) (Verrauschter) Außenseiter **(d)** SVM mit *hard margins*

(e) SVM mit *soft margins* **(f)** SVM mit *soft margins*

Abbildung 4.1.: Auf den originalen Trainingsdaten liefert die SVM mit *hard margins* ein scheinbar gutes Ergebnis. Sie passt sich anschließend jedoch einem einzigen Außenseiter sehr stark an. Mit geschickter Wahl des *soft margin* Parameters C (in der Graphik mit S bezeichnet) lässt sich dieser ignorieren und das Ergebnis ähnelt immer mehr dem auf den ursprünglichen Daten.

4.2 Der λ-Trick

Ein Problem von Algorithmus 2.1 wird schon durch das Konvergenztheorem 1 deutlich. Die transformierten Trainingsdaten müssen linear separierbar sein. Dies lässt sich mit dem λ-Trick nach [LZH+02] garantieren, indem für $|\mathcal{Z}| = N$ die Transformation $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^D$ durch $\boldsymbol{\phi}^\lambda : \mathcal{X} \mapsto \mathbb{R}^{D+N}$ ersetzt wird. Sie wird also für jedes Trainingsbeispiel um eine Dimension erweitert. In den ersten D Dimensionen sind $\boldsymbol{\phi}$ und $\boldsymbol{\phi}^\lambda$ identisch. Um den gewünschten Effekt zu erzielen, setzt man

$$\begin{aligned} \phi^\lambda_{D+n}(\mathbf{x}_n) &= \sqrt{\lambda} && \forall n, \\ \phi^\lambda_{D+i}(\mathbf{x}_n) &= 0 && \forall 1 \leq i \leq N, i \neq n. \end{aligned} \tag{4.6}$$

Offensichtlich lässt sich $\mathcal{Z}$ nun fehlerfrei durch ein Gewicht $\mathbf{w}^\dagger$ separieren, für das gilt

$$\begin{aligned} w^\dagger_d &= 0 && \forall d, \\ w^\dagger_{D+n} &= y_n && \forall n. \end{aligned} \tag{4.7}$$

Natürlich ist $\mathbf{w}^\dagger$ für neue Testdaten nutzlos, da das Skalarprodukt für diese stets 0 ergibt. Für die Dualform lässt sich der λ-Trick besonders leicht implementieren. Wegen

$$\langle \boldsymbol{\phi}^\lambda(\mathbf{x}), \boldsymbol{\phi}^\lambda(\mathbf{x}') \rangle = \begin{cases} \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle & \text{falls } \mathbf{x} \neq \mathbf{x}' \\ \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle + \lambda & \text{falls } \mathbf{x} = \mathbf{x}' \end{cases} \tag{4.8}$$

muss lediglich die bislang verwendete Kernelfunktion k durch

$$k_\lambda(\mathbf{x}, \mathbf{x}') = \begin{cases} k(\mathbf{x}, \mathbf{x}') & \text{falls } \mathbf{x} \neq \mathbf{x}' \\ k(\mathbf{x}, \mathbf{x}') + \lambda & \text{falls } \mathbf{x} = \mathbf{x}' \end{cases} \tag{4.9}$$

ersetzt werden. Ist ein Featurevektor $\boldsymbol{\phi}(\mathbf{x}_n)$ beim Lernen bereits Teil des Modells (2.6), wird er in späteren Epochen leichter ignoriert, da nun bei der Klassifizierung mittels (2.7) noch das Produkt $\alpha_n^{(t)}\lambda$ addiert wird, welches für alle (noch nicht) betrachteten Lernschritt-Prozeduren das korrekte Vorzeichen von y_n trägt. Damit entspricht der λ-Trick der SVM mit quadrierten *soft margins*

$$\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n^2, \tag{4.10}$$

da diese nach [CST00] äquivalent zu (4.1) mit Kernel (4.9) und $\lambda = 1/C$ ist. *Overfitting* ‚übermäßige Anpassung' an einzelne Featurevektoren kann somit vermieden werden. Um jedoch nicht noch einen Parameter zu erhalten, der geschickt an die Trainingsdaten angepasst werden muss, schlägt [KW07] vor, λ mit dem Kernelquadrat (ohne Augmentierung) des aktuellen Beispiels zu skalieren, um zu verhindern, dass Vektoren mit besonders großer Euklidischer Norm das Modell dominieren.

4.3 Die α-Bound

Noch simpler, aber mit ähnlichen Zielen wie der λ-Trick, agiert die α-Bound. Sie ist durch die sogenannten *box constraints* motiviert, die sich bei Nullsetzen der Differenzierung der Lagrange-Formulierung von (4.4) ergeben: $0 \leq a_n \leq C$ für die Koeffizienten a_n der SVs wie in (2.6). [Bis07] Daher fordert sie für das Perzeptron, dass jedes Trainingsbeispiel höchstens α-mal eine tatsächliche Anpassung des Modells in Zeile 10 in Algorithmus 3.1 verursachen darf, weil $\mathbf{w}^{(t)}$ die Summe der $a_n^{(t)}\boldsymbol{\phi}(\mathbf{x}_n)$ ist. Wird für den Lernschritt bislang die Prozedur update verwendet, wird nun Prozedur 4.1 dieser vorgeschaltet.

Prozedur 4.1 α-Bound auf Updates

Benötige: $\mathbf{w}, \mathbf{x}, y, o, u, t$
Benötige: $\alpha \in \mathbb{N}$
Benötige: Update-Prozedur update
1: **if** $u < \alpha$ **then**
2: **return** update$(\mathbf{w}, \mathbf{x}, y, o, u, t)$
3: **else**
4: **return** $\mathbf{w}$
5: **end if**

4.4 (Uneven) Margins

Modellanpassungen erfolgen bislang lediglich, wenn $\mathbf{x}_n$ falsch klassifiziert wird. Inspiriert von den *hard margins* (4.3) forciert diese Erweiterung ein Update, falls $y_n \langle \mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle \leq \tau$ für *margin* $\tau \in \mathbb{R}^+$. [KW07] Für Datensätze, bei denen eine Klasse viel häufiger auftritt, schlägt [LZH+02] nun das Konzept der *uneven margins* vor. Die Updatebedingung wird hier auf $y_n \langle \mathbf{w}^{(t)}, \boldsymbol{\phi}(\mathbf{x}_n) \rangle \leq \tau_{y_n}$ für $\tau_{-1}, \tau_{+1} \in \mathbb{R}^+$ erweitert. Jede Klasse muss also ihren eigenen *margin* einhalten. Für beide Varianten gelten weiterhin zu Theorem 1 ähnliche, verallgemeinerte Konvergenzschranken. Auch gelten gewisse untere Schranken für den erreichten Abstand der Ebene zu den Datenpunkten, aber da die ursprüngliche Update-Prozedur das Gewicht nach einer Anpassung nicht normiert und seine Norm dabei meist steigt, verlieren die konstanten *margins* mit der Zeit an Bedeutung. [KW07] Ähnlich wie beim λ-Trick auch, wird in [KW07] versucht, τ unabhängig von den Daten zu wählen, indem der Parameter mit dem durchschnittlichen Kernelquadrat aller Trainingsdaten skaliert wird. Dieses Vorgehen wird auf die *uneven margins* übertragen, indem man sowohl den negativen als auch den positiven *margin* skaliert.

Diese Erweiterung der *uneven margins* – in Prozedur 4.2 beschrieben – wird einer Update-Prozedur vorgeschaltet, welche von einem *margin* abhängt. Siehe dazu Abschnitt 4.6.

Prozedur 4.2 (Uneven) Margins für Vorhersagen

Benötige: $\mathbf{w}, \mathbf{x}, y, o, u, t$
Benötige: $\tau_{-1}, \tau_{+1} \in \mathbb{R}^+$
Benötige: Update-Prozedur update aus 4.6
1: **return** update$(\mathbf{w}, \mathbf{x}, y, o, \tau_y, u, t)$

4.5 Budget auf Supportvektoren

Wie bereits in Unterabschnitt 2.3.3 erwähnt, werden das Perzeptron und alle hier vorgestellten Erweiterungen in der Dualform implementiert. Die Auswertung von $\langle \mathbf{w}_\mathbf{a}, \boldsymbol{\phi}(\mathbf{x}) \rangle$ erfordert gemäß (2.7) $\mathcal{O}(N)$ Auswertungen des Kernels. Schon allein aufgrund des generellen Bedarfs nach möglichst kurzen Laufzeiten ist eine Reduzierung der Anzahl der SVs wünschenswert. Aber auch die guten Generalisierungseigenschaften von SVMs deuten bei geschickter Reduzierung auf eine verbesserte Qualität des gelernten Modells hin. So hängt die Lösung von (4.4) nur von den Vektoren ab, die den *margin* berühren oder verletzen (Sektion 4.1). [Bis07]

Die grundlegende Herangehensweise aus [WCV10] besteht darin, nach jedem Update die Anzahl der SVs des zurückgegebenen Modells zu überprüfen. Übersteigt diese einen gewissen Wert $B \in \mathbb{N}$, wird ein[2] SV wieder entfernt. Sei

$$\mathcal{SV}_\mathbf{a}^{(t)} := \left\{ n \mid a_n^{(t)} \neq 0 \right\} \tag{4.11}$$

die Indexmenge der SVs von $\mathbf{w}_\mathbf{a}^{(t)}$. Mit (4.11) implementiert Prozedur 4.3 den Ansatz dieser Erweiterung, wieder basierend auf einer beliebigen Update-Prozedur update. Einmalig liefert sie hier das Ergebnis in Zeile 1 explizit in Dualform. Um den in Zeile 3 gemachten Fehler möglichst klein zu halten, wird $\|\mathbf{w}_{\Delta\mathbf{a}}\|$ über $\Delta\mathbf{a}$ mittels

$$\arg\min_{\Delta\mathbf{a}} \left\| \mathbf{w}_{\Delta\mathbf{a}} \right\|^2 \tag{4.12}$$

[2] B wird höchstens um 1 überschritten, da beim Perzeptron pro Update höchstens ein SV hinzukommt.

nach [WCV10] quadriert minimiert, um weiterhin Kernels verwenden zu können. Dabei bezeichnet $\mathbf{w}_{\mathbf{\Delta a}}$ das Gewicht in Dualform mit den Koeffizienten Δa_n, welches vom aktuellen Modell subtrahiert wird, um einen SV zu entfernen. Von nun an seien $p, s \in \mathcal{SV}_{\mathbf{a}}^{(t+1)}$. Konventionell bezeichnet s den Index eines beliebigen, p den Index des zu entfernenden SVs.

Prozedur 4.3 Gerüst für Budget auf SVs

Benötige: $\mathbf{w}, \mathbf{x}, y, o, u, t$
Benötige: $B \in \mathbb{N}$
Benötige: Update-Prozedur update
Garantiere: $|\mathcal{SV}_{\mathbf{a}}^{(t+1)}| \leq B$
1: $\mathbf{w}_{\mathbf{a}}^{(t+1)} \leftarrow \mathsf{update}(\mathbf{w}, \mathbf{x}, y, o, u, t)$ // (2.6) liefert alle SVs und ihre Koeffizienten
2: **if** $|\mathcal{SV}_{\mathbf{a}}^{(t+1)}| > B$ **then**
3: $\quad \mathbf{w}_{\mathbf{a}}^{(t+1)} \leftarrow \mathbf{w}_{\mathbf{a}}^{(t+1)} - \mathbf{w}_{\mathbf{\Delta a}}$ // Garantie: $\Delta a_p = a_p$ entfernt SV p
4: **end if**
5: **return** $\mathbf{w}_{\mathbf{a}}^{(t+1)}$

4.5.1 Budget einhalten durch simples Entfernen

Mit dieser Methode wird ein SV einfach ersatzlos entfernt. Um in Zeile 3 den SV p zu entfernen, setzt man $\mathbf{w}_{\mathbf{\Delta a}} = a_p^{(t+1)} \boldsymbol{\phi}(\mathbf{x}_p)$. [WCV10] Dies ergibt mit (4.12) und Kernel k den zu entfernenden SV

$$p = \arg\min_s \left(a_s{}^{(t+1)}\right)^2 k(\mathbf{x}_s, \mathbf{x}_s). \tag{4.13}$$

4.5.2 Budget einhalten durch Projektion

Weiterhin wird in [WCV10] vorgeschlagen, den Verlust von SV p durch Anpassung der Koeffizienten Δa_s der verbleibenden SVs s zu kompensieren:

$$\begin{aligned} \mathbf{\Delta a}^\star &= \arg\min_{\Delta a_s, s \neq p} \frac{1}{2} \left\| \sum_{s, s \neq p} \Delta a_s \boldsymbol{\phi}(\mathbf{x}_s) - a_p \boldsymbol{\phi}(\mathbf{x}_p) \right\|^2 \\ &= \arg\min_{\mathbf{\Delta a}'} \frac{1}{2} \left\| \mathbf{\Phi \Delta a}' - \mathbf{t} \right\|^2 \end{aligned} \tag{4.14}$$

für

$$\begin{aligned} \mathbf{\Phi} &:= \left[\boldsymbol{\phi}(\mathbf{x}_s)\right] \quad s \neq p, \\ \mathbf{\Delta a}' &:= (\Delta a_s)^{\mathrm{T}} \quad s \neq p, \\ \mathbf{t} &:= a_p \boldsymbol{\phi}(\mathbf{x}_p). \end{aligned} \tag{4.15}$$

Dies entspricht tatsächlich der *least squares* ‚geringster quadratischer [Fehler]' Regression aus [Bis07] mit der Lösung

$$\mathbf{\Delta a}^\star = \left(\mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^{\mathrm{T}} \mathbf{t}, \tag{4.16}$$

die sich in Vorabtests jedoch als numerisch viel zu instabil herausstellt. Sie bestätigen die Beobachtung stark wachsender Koeffizienten aus [Bis07]. Es wird daher auf *regularized least squares* Regression [Bis07] zurückgegriffen:

$$\mathbf{\Delta a}^\star = \left(\lambda \mathbf{I} + \mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^{\mathrm{T}} \mathbf{t}. \tag{4.17}$$

Da $\mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}$ der Kernelproduktmatrix aller SVs außer p entspricht, wird die Regularisierung bereits durch den λ-Trick implementiert. Dennoch wird ein neuer Parameter eingeführt, um auch bei deaktiviertem λ-Trick die Regularisierung einstellen zu können. Die Lösung $\mathbf{\Delta a}^\star$ lässt sich in $\mathcal{O}(B^2)$ bestimmen, allerdings müsste man sie für jedes

mögliche zu entfernende p ausrechnen, um das wirkliche Minimum zu erhalten, was zu einer Gesamtlaufzeitskomplexität $\mathcal{O}(B^3)$ führt, weshalb p heuristisch mittels (4.13) gewählt wird. [WCV10]

Ich füge die Option hinzu, bei maximal B SVs, diese bei der Projektion durch lediglich $B' \leq B$ zufällig ausgewählte SVs zu ersetzen. Testen möchte ich zusätzlich noch die Heuristik, welche stets den zuletzt hinzugekommenen SV entfernt (auch ohne anschließende Projektion). Dies steht zwar nach [WCV10] im Widerspruch zum *Forgetron*, bringt aber dahingehend Stabilität beim Lernen, da das Modell stets aus den ersten B SVs besteht und so wenigstens auf diesen intensiv gelernt werden kann.

4.6 Lernschritte

Es folgen nun die Rekursionsanker der Update-Prozedur 4.2, die von *(uneven) margins* Gebrauch machen. Wie Prozedur 3.2 bestimmen sie lediglich, ob und wie das Modell angepasst werden soll.

4.6.1 Konstante Lernrate

Prozedur 4.4 implementiert die Erweiterung unter 4.4 für Anpassungen mit konstanter Lernrate.

Prozedur 4.4 Standard Update-Prozedur mit *margin*

Benötige: $\mathbf{w}, \mathbf{x}, y, o, \tau, u, t$
Benötige: Lernrate $\eta \in \mathbb{R}^+$
1: **if** $yo \leq \tau$ **then**
2: **return** $\mathbf{w} + \eta y \mathbf{x}$
3: **else**
4: **return** $\mathbf{w}$
5: **end if**

4.6.2 Passive-Aggressive (PA)

Die *Passive-Aggressive* (PA)-Methode [CDK+06] existiert in drei Varianten, die in Prozedur 4.5 zusammengefasst sind. Die erste resultiert aus dem Bestreben, ein Gewicht zu finden, das auf dem aktuellen Beispiel den *margin* τ einhält, dabei aber minimal vom ursprünglichen Gewicht abweicht:

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}\in\mathbb{R}} \frac{1}{2} \left\| \mathbf{w} - \mathbf{w}^{(t)} \right\| \quad \text{mit} \quad loss = 0 \tag{4.18}$$

für die „Hinge-Loss"-Funktion $loss$ wie in Zeile 1. Die anderen beiden wollen zusätzlich von *soft margins* profitieren, was – sehr ähnlich zur SVM – mit einer *slack variable* $\xi \in \mathbb{R}$ gelöst wird, deren „Schmutz" mit Faktor C justiert wird:

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}\in\mathbb{R}} \frac{1}{2} \left\| \mathbf{w} - \mathbf{w}^{(t)} \right\| + C\xi \quad \text{mit} \quad loss \leq \xi, \xi \geq 0 \tag{4.19}$$

für $V =$ PA-I und

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}\in\mathbb{R}} \frac{1}{2} \left\| \mathbf{w} - \mathbf{w}^{(t)} \right\| + C\xi^2 \quad \text{mit} \quad loss \leq \xi \tag{4.20}$$

für $V =$ PA-II. Die Lösungen werden wieder mittels Differenzierung der jeweiligen Lagrange-Formulierung ermittelt, wobei die Autoren hervorheben, dass PA-II dem einfachen PA mit λ-Trick und $\lambda = \frac{1}{2C}$ entspricht. Ursprünglich für das Online-Lernen konzipiert, beruht PA mit seinem $loss$-gesteuerten Update auf der Zielsetzung, das aktuelle Beispiel nachhaltig in das Modell einzufügen, das Modell aber so wenig wie möglich zu ändern, um vergangene Updates nicht zu dominieren.

Prozedur 4.5 PA Update

Benötige: $\mathbf{w}, \mathbf{x}, y, o, \tau, u, t$
Benötige: $\tau > 0$, da sonst wegen $\mathbf{w}^{(0)} = \mathbf{0}$ für alle t $loss = 0$ gilt
Benötige: Aggressivität $C \in \mathbb{R}^+$
Benötige: Variante $V \in \{\text{PA}, \text{PA-I}, \text{PA-II}\}$
1: $loss = \max\{0, \tau - yo\}$
2: **if** $V = \text{PA}$ **then**
3: $\quad \eta \leftarrow \frac{loss}{\|\mathbf{x}\|^2}$
4: **else if** $V = \text{PA-I}$ **then**
5: $\quad \eta \leftarrow \min\{C, \frac{loss}{\|\mathbf{x}\|^2}\}$
6: **else if** $V = \text{PA-II}$ **then**
7: $\quad \eta \leftarrow \frac{loss}{\|\mathbf{x}\|^2 + (2C)^{-1}}$
8: **end if**
9: **return** $\mathbf{w} + \eta y \mathbf{x}$

4.6.3 Mistake-Controlled Rule Algorithm (MICRA$^{\epsilon,\zeta}$)

Der *Mistake-Controlled Rule Algorithm* (MICRA$^{\epsilon,\zeta}$) [TST07] strebt zur *maximum margin* Lösung, indem er berücksichtigt, dass spätere Updates mit einer konstanten Lernrate kaum einen Effekt haben. Deshalb wird die Norm und die bisherige Anzahl der Anpassungen für den *margin* 4.6 und den Lernschritt 4.7 berücksichtigt. Diese Idee wird umgesetzt, indem die „effektive Lernrate“ zum Zeitpunkt t

$$\eta_{eff}^{(t)} = \eta t^{-\zeta} \tag{4.21}$$

gesetzt wird. Die „effektive Lernrate“ ist ein Maß für die tatsächliche Stärke einer Anpassung des Modells $\mathbf{w}$:

$$\eta_{eff}^{(t)} = \frac{\eta^{(t)} K}{\|\mathbf{w}\|}. \tag{4.22}$$

Gleichsetzen von (4.21) und (4.22) ergibt $\eta^{(t)}$ in Zeile 3 der Update-Prozedur 4.7. Analog beträgt der *margin* $\|\mathbf{w}\| \beta t^{-\epsilon}$ zum Zeitpunkt t. Siehe hierzu (5), (6) und (7) aus [TST07]. Die wichtigsten Parameter ϵ und ζ steuern den Verfall des *margins* und der Lernrate, da $\|\mathbf{w}\|$ erwartungsgemäß steigt. Mit δ kann η abhängig von β – dem Start-*margin* – gewählt werden. Bei geschickter Wahl dieser Parameter „tendiert“ [TST07] MICRA$^{\epsilon,\zeta}$ für $\beta/K \to \infty$ zur Lösung mit maximalem Abstand zu den Daten, Separierbarkeit vorausgesetzt.

Wird Prozedur 4.6 und/oder 4.7 für Algorithmus 3.1 verwendet, so muss unmittelbar vor der ersten Schleife $\mathbf{w}^{(1)}$ mit $y_1 \boldsymbol{\phi}(\mathbf{x}_1)$ initialisiert werden, was einem Standard-Update mit $\eta = 1$ entspricht, weshalb auch u_1 und t jeweils inkrementiert werden. Erst jetzt können der *margin* 4.6 und der Lernschritt 4.7 von $\|\mathbf{w}\|$ und t^{-x} abhängen.

Prozedur 4.6 MICRA$^{\epsilon,\zeta}$ Margin

Benötige: $\mathbf{w}, \mathbf{x}, y, o, \tau, u, t$
Benötige: $\epsilon \in \mathbb{R}^+$
Benötige: $\beta \in \mathbb{R}^+$ wie in Prozedur 4.7
Benötige: Update-Prozedur update aus 4.6
1: $\beta^{(t)} \leftarrow \|\mathbf{w}\| \beta t^{-\epsilon}$
2: **return** update$(\mathbf{w}, \mathbf{x}, y, o, \beta^{(t)}, u, t)$

4.6.4 Primal Estimated sub-GrAdient SOlver for SVM (Pegasos)

Primal Estimated sub-GrAdient SOlver for SVM (Pegasos) ergibt sich nach [SSS07] aus der Reformulierung der SVM mit Hilfe der „Hinge-Loss“-Funktion wie in Prozedur 4.5. Der stochastische Gradient dieser Reformulierung und eine anschließende Projektion zurück in den Raum (4) aus [SSS07] der optimalen Lösung der SVM ergibt Prozedur 4.8.

Prozedur 4.7 MICRA$^{\epsilon,\zeta}$ Update

Benötige: $\mathbf{w}, \mathbf{x}, y, o, \tau, u, t$
Benötige: $\zeta, \eta^{(0)}, \delta \in \mathbb{R}^+$
Benötige: $\beta \in \mathbb{R}^+$ wie in Prozedur 4.6
Benötige: $K = \max_n \|\boldsymbol{\phi}(\mathbf{x}_n)\|$ aus den transformierten Trainingsdaten
$\eta \leftarrow \eta^{(0)}(\beta/K)^{-\delta}$
$\overline{\eta} \leftarrow \eta/K$
$\eta^{(t)} \leftarrow \|\mathbf{w}\|\overline{\eta}t^{-\zeta}$
if $yo \leq \tau$ **then**
  return $\mathbf{w} + \eta^{(t)} y\mathbf{x}$
else
  return $\mathbf{w}$
end if

Prozedur 4.8 Pegasos Update

Benötige: $\mathbf{w}, \mathbf{x}, y, o, \tau, u, t$
Benötige: $\lambda \in \mathbb{R}^+$ // Regularisiert beim (stochastischen) Gradienten, wie stark der *loss* minimiert wird
Benötige: *Static* ‚statische' Variable $t_{\text{Pegasos}} \leftarrow 1$ // zählt die Aufrufe von 4.8
$loss = \max\{0, \tau - yo\}$
if $loss > 0$ **then**
  $\eta \leftarrow (\lambda t_{\text{Pegasos}})^{-1}$
else
  $\eta \leftarrow 0$
end if
$\mathbf{w}^{(t+\frac{1}{2})} \leftarrow (1 - t_{\text{Pegasos}}^{-1})\mathbf{w} + \eta y\mathbf{x}$ // Stochastischer Gradient von (1) aus [SSS07]
$\mathbf{w}^{(t+1)} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}^{(t+\frac{1}{2})}\|}\right\}\mathbf{w}^{(t+\frac{1}{2})}$ // Projektion in den Raum der optimalen Lösung (4) aus [SSS07]
$t_{\text{Pegasos}} \leftarrow t_{\text{Pegasos}} + 1$
return $\mathbf{w}^{(t+1)}$

4.7 Zusammenfassung

Die Verbesserungen dieses Kapitels werden sinnvollerweise in dieser Reihenfolge zusammengefügt

$$\lambda\text{-Trick 4.2} \rightarrow \alpha\text{-Bound 4.3} \rightarrow \text{Budget auf SVs 4.5} \rightarrow \text{Margin 4.4, 4.6.3} \rightarrow \text{Lernschritt 4.6}, \tag{4.23}$$

um unnötige Berechnungen zu vermeiden und die benötigten Parameter zu bedienen.

5 Zusammenführung

In diesem Kapitel werden die Verbesserungen aus Kapitel 3 und 4 zusammengeführt. Dazu liefert Sektion 5.1 eine zusammengefasste Übersicht aller Module und ihrer Parameter. In Sektion 5.2 wird dann die Brücke zur GUI der Implementierung in Weka geschlagen.

5.1 Überblick

Algorithmus 3.1 mit den Parametern aus Tabelle 5.1 startet pro Beispiel eine Verkettung von Meta-Update-Prozeduren aus Tabelle 5.2, die in einer Lernschritt-Prozedur aus Tabelle 5.3 mündet.

Parameter	Bedeutung
I	Maximale Anzahl Epochen
H	Maximale Anzahl reduzierter Epochen pro voller Epoche
update	Auf Beispiel reagieren und Modell u. U. anpassen
Modell	Standard, längster Überlebender oder gewichtete Abstimmung

Tabelle 5.1.: Batch-Erweiterungen aus Kapitel 3

Meta-update	Parameter	Bedeutung
λ-Trick	λ	Zusätzliche Dimension pro Beispiel mit Wert λ garantiert Separierbarkeit
α-Bound	α	Maximal α Anpassungen pro Beispiel
(Uneven) Margins	τ_y	Ruft Lernschritt auf, der Abstände berücksichtigt
Budget auf SVs	B	Maximale Anzahl von SVs
	Auswahlheuristik	Bestimmt, welcher SV p entfernt wird
	Projektion, λ, B'	Projiziere p auf auf B' SVs mit Regularisierung λ
MICRA$^{\epsilon,\zeta}$	β	Start-*margin*
	ϵ	Stärke der Degradierung des geforderten Abstands über die Zeit

Tabelle 5.2.: Meta-Update-Prozeduren aus Kapitel 4, die das Modell nicht selbst anpassen, sondern dafür eine weitere Update-Prozedur aufrufen.

Lernschritt update	Parameter	Bedeutung
Konstante Lernrate	η	Update mit Stärke η bei Betreten des *margins*
PA	Variante	Ob und wie *soft margins* verwendet werden
	C	Abwägung zwischen *soft* und *hard margins*
MICRA$^{\epsilon,\zeta}$	$\eta^{(0)}$	Start-Lernrate
	ζ	Stärke der Degradierung der Lernrate über die Zeit
	δ, β	Lernrate in Abhängigkeit von β, jedoch standardmäßig $\delta = 0$
Pegasos	λ	*slack trade-off* [WCV10]

Tabelle 5.3.: Lernschritte mit *margins* aus 4.6

5.2 Graphical User Interface (GUI)

Das Hauptfenster aus Abbildung 5.1, implementiert durch die Klasse **weka.classifiers.pla.Perceptrovement**, entspricht Algorithmus 3.1, der die verwendeten Erweiterungen und deren Parameter einstellt. Die Parameter aus Tabelle 5.1 sind direkt verfügbar, die den Tabellen 5.2 und 5.3 entsprechenden Einstellungen sind weitestgehend in Untermenüs verfügbar. Hinter der Auswahlmöglichkeit **weight** verbergen sich jene Erweiterungen, die bei der Dualform von der verwendeten Kernelfunktion abhängen. Das entsprechende Dialogfenster ist in Abbildung 5.2 dargestellt. Es existiert auch eine Implementierung in linearer Primalform, welche in vielen Fällen schneller gelernt werden kann.

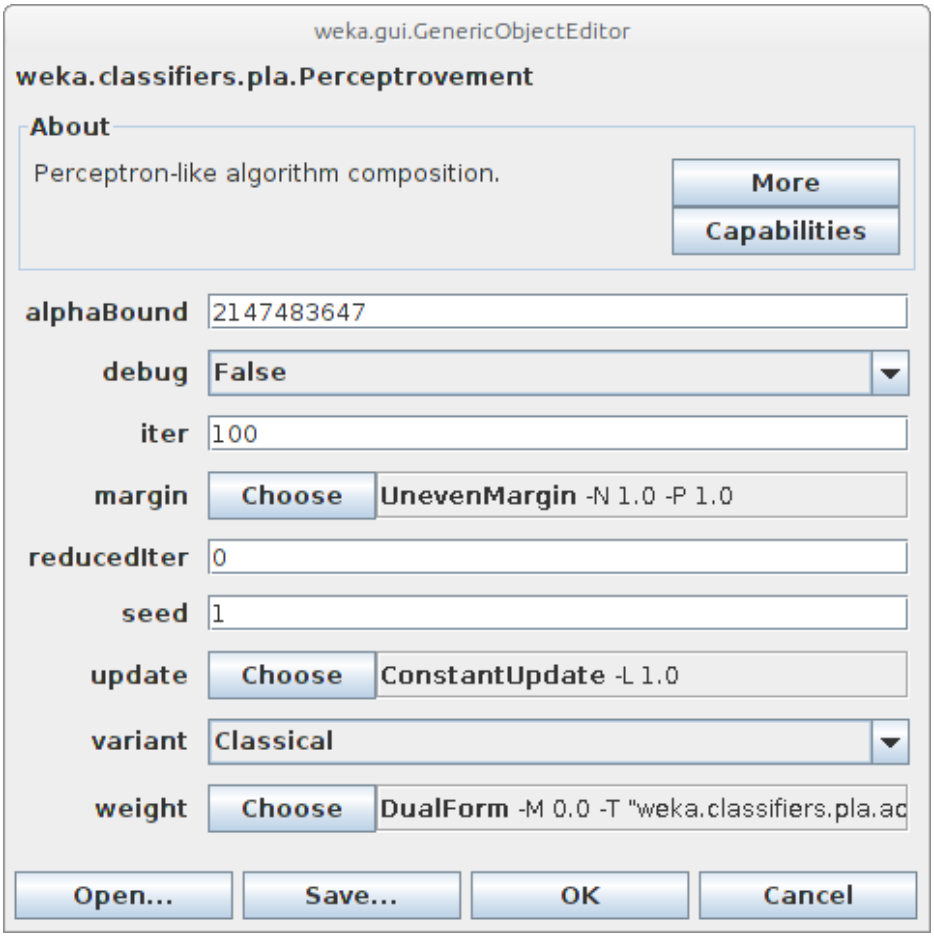


Abbildung 5.1.: Hauptfenster

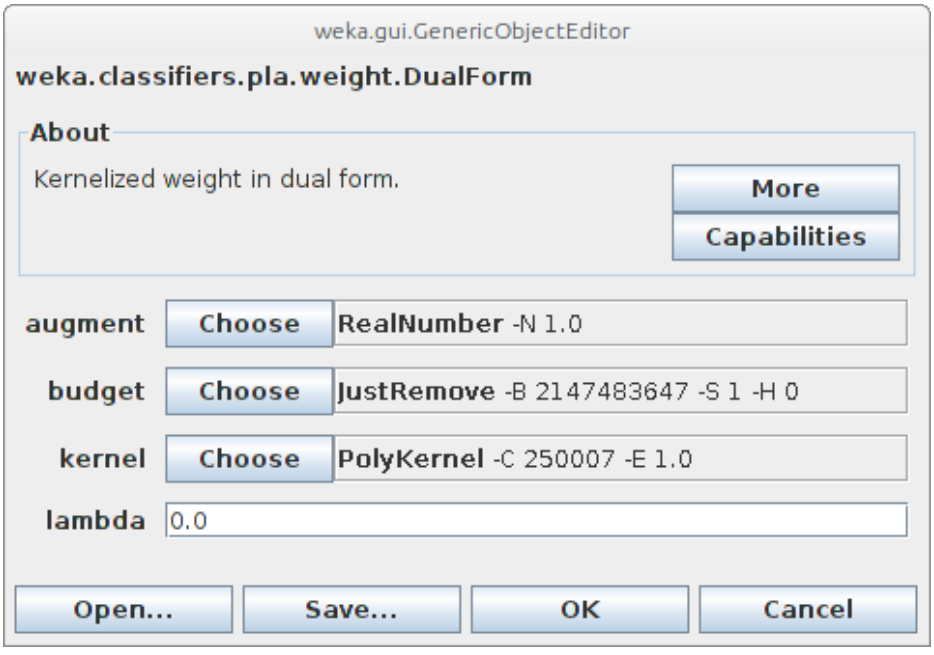


Abbildung 5.2.: Optionen bei einer Implementierung in Dualform

Teil III.
Evaluation

Die in Teil II eingeführten Verbesserungen lassen sich nahezu beliebig kombinieren und bringen viele neue Parameter mit. Genauso unterschiedlich sind die Experimente der jeweiligen Autoren aufgebaut, je nachdem, was sie mit ihrer Verbesserung im Detail erreichen wollen. Nicht immer steht die erreichte Genauigkeit im Vordergrund. So wird in [TST07] gemessen, wie schnell welcher *margin* erreicht wird. Und natürlich konzentriert sich [WCV10] auf den Unterschied der Genauigkeit und der Geschwindigkeit mit und ohne Budget auf SVs.

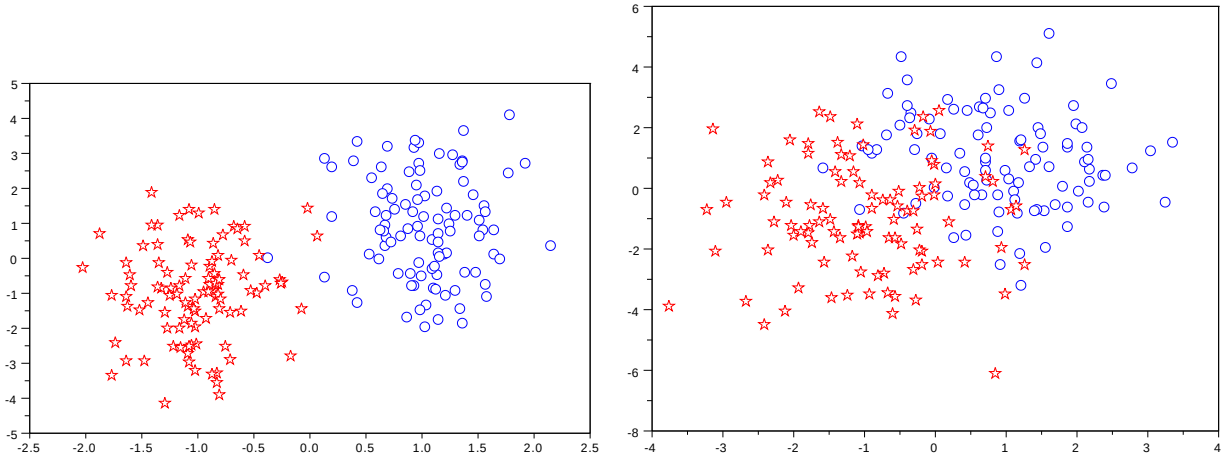
6 Experimente

Sämtliche solche Verfahren und Kriterien aller Autoren unterzubringen, würde den Rahmen dieser Arbeit sprengen. Stattdessen orientiert sich dieser Experimentalteil an der *parameter search* ‚Parametersuche' und der *greedy parameter optimization* „‚gierige" Parameteroptimierung' aus [KW07]. Bei der *parameter search* werden möglichst viele Kombinationen mit beliebten Werten der Autoren für die jeweiligen Parameter auf künstlichen und realen Datensätzen evaluiert. Das Performanzmaß hierbei ist die durchschnittliche Genauigkeit von 10 Durchläufen einer zehnfachen *cross-validation* ‚Kreuzvalidierung' [Bis07]. Bei der *greedy parameter optimization* wird eine zusätzliche fünffache Kreuzvalidierungsebene eingeführt, um den vermutlich besten Parameter automatisch zu bestimmen. Damit wird verhindert, dass die Parameter an die Testdaten angepasst werden. Da dies sehr rechenaufwendig ist, wird dabei *greedy* vorgegangen: jeder Parameter wird einzeln nacheinander optimiert und danach nicht mehr geändert. Stets wird mit den Ergebnissen der linearen SVM aus [CL11] verglichen. Sie wird mittels [EMH05] in Weka eingebunden und als Referenzklassifizierer verwendet. Darauf aufbauend bestimmt Weka, ob ein getesteter Klassifizierer signifikant davon abweicht. Pro folgender Sektion wird festgelegt, zu welchem Zweck welche Kombinationen auf welchen Datensätzen evaluiert werden.

6.1 Überblick gewinnen

Der in Abbildung 6.1 dargestellte, nach [CDK+06] künstlich generierte Datensatz soll in diesem Experiment dazu verwendet werden, einen ersten Überblick über einige Module und Erweiterungen aus Teil II zu gewinnen: „Worauf lohnt es sich zu konzentrieren, welche Kombinationen sind unbrauchbar?" Getestet wird mit den Standardparametern aus Tabelle 6.1, ohne Budget und α-Bound. Ihr Vorteil kommt bei größeren Datensätzen zum Tragen. Außerdem wird kein augmentierter Bias benötigt, da die Bayes-optimale Trennebene durch den Ursprung verläuft. Pro Messreihe werden die Parameter aus Tabelle 5.1 variiert: $(I, H, \text{Modell}) \in \{10, 100\} \times \{0, 10\} \times \{\text{Standard, Längster Überlebender, Gewichtete Abstimmung}\}$. Diese Messreihen stehen in den Tabellen 6.2–6.10.

Besonders auffällig ist, dass sehr viele Kombinationen eine sehr hohe Genauigkeit auf den unverfälschten Daten erreichen. Dies gilt selbst für eine geringe Zahl an Epochen. Eine weitere Auffälligkeit sind die schlechten Werte für (31) und (33) bei steigender Epochenzahl. Numerische Instabilität ist der Grund. PA ohne *slack variables* erleidet einen zu hohen *loss*, kontert mit einem starken Lernschritt, dies erhöht $\|\mathbf{w}\|$, wodurch der *margin* von MICRA$^{\epsilon,\zeta}$ wieder steigt und somit der *loss* beim nächsten Fehler noch größer wird, usw. Zusätzlich sind (35) und (37) bei geringer Epochenzahl nahezu immer signifikant schlechter als (1). Scheinbar lässt sich der PA-Lernschritt generell schlecht mit dem variablen *margin* von MICRA$^{\epsilon,\zeta}$ kombinieren. Interessant ist die Frage, ob 10 reduzierte bei 10 vollen Epochen mit 100 vollen Epochen mithalten können. Für das Standardmodell scheint dies, mit Ausnahme von (3), (5), (19), (21), (23) und (39), zuzutreffen. (41) entspricht (39) mit λ-Trick und kann in diesem Fall *overfitting* in den reduzierten Epochen verhindern. Im Fall von (3), (19), (23) und (39) sind die Ergebnisse mit den zusätzlichen reduzierten Epochen sogar deutlich schlechter als ohne. Viele Kombinationen, z. B. (3), (5), (11), (19) und (23) – als Standardmodell signifikant schlechter –, lassen sich mit Hilfe der komplexeren Modelle zuverlässig verbessern. Das Gegenteil ist bei (15) und (17) für wenige Epochen der Fall. Bemerkenswerterweise vermag keine Kombination die SVM bei gleichzeitigem Instanz- und Klassifikationsrauschen zu schlagen. Besonders robust scheinen sich (25) und (41) über alle Messreihen hinweg zu verhalten. Während (25) klassischem, konstantem Update mit MICRA$^{\epsilon,\zeta}$-*margin* und λ-Trick entspricht, bildet (41) eine ebenfalls unerwartete Kombination aus MICRA$^{\epsilon,\zeta}$-Margin, λ-Trick und Pegasos. Ebenso ergibt sich für (3)–(13) beim abstimmenden Modell selten eine signifikante Abweichung von (1). Besonders bei wenigen Epochen kann hier ein Geschwindigkeitsvorteil gewahrt werden.



(a) Normalverteilt mit $\mu_\circ = -\mu_\star = (1,1)^{\mathrm{T}}, \Sigma = \left(\begin{smallmatrix} 0.2 & 0 \\ 0 & 2 \end{smallmatrix}\right)$

(b) Gaußsches Rauschen mit $\mu = \mathbf{0}, \Sigma = \mathbf{I}$

(c) Klassifikation mit Wahrscheinlichkeit 0.15 falsch

(d) Instanzdaten und Klassifikation verrauscht

Abbildung 6.1.: Synthetischer Datensatz wie in [CDK+06] mit Instanz- und/oder Klassifikationsrauschen für $N = 200$ generiert.

Tabelle 6.1.: Standardparameter, L für η, M für λ, N für τ_{-1}, P für τ_{+1}, Rest wie im Pseudo-Code

(1)	functions.LibSVM '-K 0 -C 1.0'
(3)	pla.Perceptrovement '-U \"pla.addon.update.ConstantUpdate -L 1.0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(5)	pla.Perceptrovement '-U \"pla.addon.update.ConstantUpdate -L 1.0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(7)	pla.Perceptrovement '-U \"pla.addon.update.MICRAUpdate -L 2.0 -Z 0.875 -B 1.0 -D 0.0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(9)	pla.Perceptrovement '-U \"pla.addon.update.MICRAUpdate -L 2.0 -Z 0.875 -B 1.0 -D 0.0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(11)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C Infinity -V 0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(13)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C Infinity -V 0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(15)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C 0.001 -V 0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(17)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C 0.001 -V 0\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(19)	pla.Perceptrovement '-U \"pla.addon.update.Pegasos -M 1.0E-4\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(21)	pla.Perceptrovement '-U \"pla.addon.update.Pegasos -M 1.0E-4\" -G \"pla.addon.margin.UnevenMargin -N 1.0 -P 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(23)	pla.Perceptrovement '-U \"pla.addon.update.ConstantUpdate -L 1.0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(25)	pla.Perceptrovement '-U \"pla.addon.update.ConstantUpdate -L 1.0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(27)	pla.Perceptrovement '-U \"pla.addon.update.MICRAUpdate -L 2.0 -Z 0.875 -B 1.0 -D 0.0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(29)	pla.Perceptrovement '-U \"pla.addon.update.MICRAUpdate -L 2.0 -Z 0.875 -B 1.0 -D 0.0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(31)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C Infinity -V 0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(33)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C Infinity -V 0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(35)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C 0.001 -V 0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(37)	pla.Perceptrovement '-U \"pla.addon.update.PAUpdate -C 0.001 -V 0\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 1.0\“
(39)	pla.Perceptrovement '-U \"pla.addon.update.Pegasos -M 1.0E-4\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 0.0\“
(41)	pla.Perceptrovement '-U \"pla.addon.update.Pegasos -M 1.0E-4\" -G \"pla.addon.margin.MICRAMargin -E 0.0625 -B 1.0\" -W \"pla.weight.DualForm -M 1.0\“

Tabelle 6.2.: $I = 10, H = 0$, Modell = Standard

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.1	**98.5**∘	**98.5**∘	**98.5**∘	95.1	**98.5**∘	97.9	97.9	98.0	98.4∘	**98.5**∘	**98.5**∘	**98.5**∘	**98.5**∘	95.4	97.6	93.8•	93.8•	98.3	98.3
6.1b	86.2	82.9•	85.2	85.8	85.7•	74.3•	85.2	**86.0**	**86.0**	80.3•	83.4•	85.4	**86.0**	85.8	85.9	73.5•	77.6•	81.5•	81.9•	85.0	85.6
6.1c	81.1	70.2•	76.4•	81.0	**81.5**∘	68.0•	79.3•	80.6•	80.6•	68.7•	68.1•	74.1•	**81.5**∘	81.2	**81.5**∘	57.4•	71.8•	71.6•	72.5•	73.6•	80.6
6.1d	74.3	66.0•	65.4•	72.3•	73.5	59.6•	69.9•	73.7	73.7	64.4•	62.5•	65.6•	**74.0**	71.7•	73.6	54.2•	62.6•	64.4•	65.3•	67.5•	71.7

∘, • statistically significant improvement or degradation

Tabelle 6.3.: $I = 10, H = 0$, Modell = Längster Überlebender

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	**98.5**∘	**98.5**∘	**98.5**∘	**98.5**∘	98.3	**98.5**∘	97.5•	97.5•	98.4	98.4∘	**98.5**∘	**98.5**∘	**98.5**∘	**98.5**∘	98.3	98.3	92.0•	92.2•	98.2	98.3
6.1b	86.2	85.6	**86.2**	85.6	85.9	83.8•	85.7	85.3	85.2•	84.0•	85.2	85.5	85.6	85.7	85.9	84.6•	84.1•	78.4•	78.7•	85.5	84.4
6.1c	81.1	80.6	80.3•	79.1	80.4	80.0	80.4	80.1•	80.0•	79.6	79.9•	80.9	**81.1**	80.2	81.0	68.8	79.4•	69.2•	68.1•	80.4	80.6
6.1d	74.3	71.2•	72.7	70.0•	72.7•	70.5•	71.4•	**73.3**•	**73.3**•	70.5•	72.1	71.7•	73.0	70.0•	73.0	61.5•	71.9•	62.6•	59.9•	71.7•	71.7•

∘, • statistically significant improvement or degradation

Tabelle 6.4.: $I = 10, H = 0$, Modell = Gewichtete Abstimmung

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.5∘	98.5∘	98.5∘	98.5∘	**98.6**∘	98.5∘	97.1•	97.2•	98.4∘	98.4∘	98.5∘	98.5∘	98.5∘	98.5∘	98.5∘	98.5∘	91.8•	92.1•	98.5∘	98.5∘
6.1b	86.2	85.9	**86.4**	86.0	85.8	85.4	86.0	84.9•	84.9•	85.5	85.9	86.1	86.0	85.8	85.9	86.3	86.0	78.6•	78.8•	86.0	86.2
6.1c	81.1	81.2	81.1	80.6	**81.5**∘	81.0	81.3	80.7•	80.7•	81.1	80.5	81.2	**81.5**∘	80.6	**81.5**∘	77.9	81.0	67.7•	67.3•	81.0	81.0
6.1d	74.3	73.2•	73.2	72.0•	73.6	73.6	73.2•	73.4•	73.4•	73.0•	73.1•	**73.7**	**73.7**	70.9•	73.6	70.8	73.3	60.9•	60.7•	73.4	73.5

∘, • statistically significant improvement or degradation

Tabelle 6.5.: $I = 10, H = 10$, Modell = Standard

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	96.2	98.4∘	98.5∘	98.5∘	95.0	98.5∘	98.5∘	98.5∘	93.1•	**98.7**∘	98.5∘	98.5∘	98.5∘	98.5∘	59.4•	97.7	98.5∘	98.5∘	98.5∘	98.5∘
6.1b	86.2	73.2•	**86.2**	85.7•	85.6•	74.3•	85.7•	85.8•	85.8	73.6•	84.9	79.8•	85.8	85.7•	85.7•	43.4•	0.0•	82.6	85.7•	77.0•	86.1
6.1c	81.1	66.9•	79.8	81.4	**81.5**∘	68.0•	**81.5**∘	**81.5**∘	**81.5**∘	65.0•	76.5	63.8•	**81.5**∘	80.7	**81.5**∘	55.3•	0.0•	66.1•	**81.5**∘	62.8•	**81.5**∘
6.1d	74.3	60.4•	70.9•	73.7	73.5	59.6•	73.9	73.4	73.5	58.7•	62.3•	58.0•	73.8	73.1•	73.5•	56.2•	0.0•	59.2•	73.6	57.9•	**74.1**

∘, • statistically significant improvement or degradation

Tabelle 6.6.: $I = 10, H = 10$, Modell = Längster Überlebender

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.5∘	98.4∘	98.5∘	98.5∘	98.3	98.5∘	98.5∘	98.5∘	98.4	**98.7**∘	98.5∘	98.5∘	98.5∘	98.5∘	61.6•	98.4∘	98.4∘	98.5∘	98.5∘	98.5∘
6.1b	86.2	85.2	**85.9**	85.4•	85.8	83.6•	85.8	85.8	85.8	85.2	85.5	85.1•	85.3	85.8	85.6	50.7•	0.0•	83.3	84.4•	85.4•	84.7•
6.1c	81.1	80.6	80.1	79.6	80.4	80.0	**81.4**	81.3	81.2	79.4	77.6	80.7	81.0	81.3	80.9	65.4•	0.0•	79.7	80.4	80.3	80.3
6.1d	74.3	70.8•	62.3•	70.9•	72.1•	70.3•	**73.7**	73.6	73.5	71.4•	60.4•	70.7•	73.3	72.1•	72.5•	65.0•	0.0•	68.1•	69.9•	71.0•	72.5•

∘, • statistically significant improvement or degradation

Tabelle 6.7.: $I = 10, H = 10$, Modell = Gewichtete Abstimmung

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.5◦	98.4◦	98.5◦	98.5◦	98.6◦	98.5◦	98.5◦	98.5◦	98.4◦	**98.7**◦	98.5◦	98.5◦	98.5◦	98.5◦	98.5◦	98.5◦	98.1	98.2	98.5◦	98.5◦
6.1b	86.2	**86.3**	**86.3**	85.8	85.7•	85.4•	85.7•	86.1	86.1	86.0	85.8	86.2	85.8	85.7•	85.8	85.5	85.5	85.6	85.4	85.9	86.1
6.1c	81.1	81.3	81.4◦	81.4	81.5◦	81.0	81.5◦	81.5◦	81.5◦	81.2	81.3	81.3	81.5◦	80.9	81.5◦	76.1•	80.3	80.7	**81.6**◦	81.3	81.5◦
6.1d	74.3	73.2	73.8	73.7	73.8	73.6	**74.0**	73.7	73.7	73.1•	73.7	73.2•	73.4	72.5•	73.5•	71.9	71.3•	70.9•	73.5	73.2	73.8

◦, • statistically significant improvement or degradation

Tabelle 6.8.: $I = 100, H = 0$, Modell = Standard

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.2	**98.5**◦	**98.5**◦	**98.5**◦	95.0	**98.5**◦	**98.5**◦	**98.5**◦	98.4◦	98.4◦	**98.5**◦	**98.5**◦	**98.5**◦	**98.5**◦	77.6•	97.7	**98.5**◦	**98.5**◦	**98.5**◦	**98.5**◦
6.1b	86.2	82.6•	86.0	85.8	85.7•	74.3•	85.7•	85.8•	85.8	83.3•	85.6	84.8	85.8	85.7•	85.7•	48.0•	7.3•	84.2•	85.7•	84.5	**86.1**
6.1c	81.1	70.2•	81.2	81.4	**81.5**◦	68.0•	**81.5**◦	**81.5**◦	**81.5**◦	72.9•	80.9	72.2•	**81.5**◦	80.6	**81.5**◦	57.5•	1.5•	63.0•	**81.5**◦	71.8•	**81.5**◦
6.1d	74.3	65.0•	73.2	73.5	73.5	59.6•	73.8	73.4	73.4	67.5•	73.5	66.9•	73.6	72.6•	73.4	54.2•	0.0•	60.6•	73.5	66.2•	**74.1**

◦, • statistically significant improvement or degradation

Tabelle 6.9.: $I = 100, H = 0$, Modell = Längster Überlebender

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	**98.5**◦	**98.5**◦	**98.5**◦	**98.5**◦	98.3	**98.5**◦	**98.5**◦	**98.5**◦	98.3	98.4◦	**98.5**◦	**98.5**◦	**98.5**◦	**98.5**◦	79.5•	98.4◦	98.3	98.4	**98.5**◦	98.4◦
6.1b	86.2	85.0	**86.0**	85.6•	85.8	83.8•	85.7•	85.9	85.9	84.0•	85.6	85.3	85.6	85.8	85.7	55.9•	8.8•	84.0•	83.8•	85.4	85.1
6.1c	81.1	80.4	81.2	79.7	80.6	80.0	**81.5**◦	81.2	81.2	79.6	80.9	80.5	81.1	80.7	81.0	68.9	1.6•	75.2	79.7	80.4	80.6
6.1d	74.3	71.4•	73.2	71.0•	72.6•	70.5•	73.3	**73.6**	73.5	70.1•	73.5	71.8	72.9	71.2•	73.1	61.4•	0.0•	69.8•	67.4•	71.9•	71.7•

◦, • statistically significant improvement or degradation

Tabelle 6.10.: $I = 100, H = 0$, Modell = Gewichtete Abstimmung

Data	(1)	(3)	(5)	(7)	(9)	(11)	(13)	(15)	(17)	(19)	(21)	(23)	(25)	(27)	(29)	(31)	(33)	(35)	(37)	(39)	(41)
6.1a	98.1	98.5◦	98.5◦	98.5◦	98.5◦	**98.6**◦	98.5◦	98.5◦	98.5◦	98.5◦	98.4◦	98.5◦	98.5◦	98.5◦	98.5◦	98.5◦	98.5◦	98.2	98.2	98.5◦	98.5◦
6.1b	86.2	85.9	86.0	85.9	85.7•	85.4	85.7•	85.8	85.8	86.1	85.6	**86.2**	85.8	85.6•	85.7•	86.1	86.1	85.7	85.7	**86.2**	86.0
6.1c	81.1	81.1	81.2	81.1	81.5◦	81.1	81.5◦	81.5◦	81.5◦	81.0	80.9	81.1	81.5◦	81.0	81.5◦	77.2	80.9	80.3	**81.6**◦	81.1	81.5◦
6.1d	74.3	73.1•	73.2	73.4	73.5	73.6	**74.0**	73.3	73.4	73.3•	73.5	73.5	73.4	72.3•	73.4	70.7	71.6•	73.3•	73.3•	73.5	73.8

◦, • statistically significant improvement or degradation

6.2 Budget

Für die Evaluierung der Budget-Erweiterung werden die künstlichen Daten des vorangegangenen Experiments in fünffacher Größe generiert. Für die Genauigkeit in Tabelle 6.12 dient nun die SVM mit optimiertem C als Referenz. Das Perzeptron wird von Konfiguration (41) aus Tabelle 6.1 vertreten, die im vorangegangenen Experiment überraschend gut abschneidet. (2) schneidet im Vergleich zur SVM sehr gut ab. Das Budget mittels Projektion einzuhalten, funktioniert offensichtlich auch sehr gut. Insbesondere die Konfigurationen mit wenigen projizierenden SVs sind nie signifikant schlechter. Simples Entfernen schneidet deutlich schlechter ab; das Budget müsste noch weiter erhöht werden. Die Auswahlheuristik, immer den neuesten SV zu entfernen, bringt nur auf Datensatz 6.1c die erhoffte Stabilisierung. Als Referenz zum Vergleich der Trainingszeiten in Tabelle 6.13 wird (2) ohne Budget in Primalform ausgeführt. Dabei zeigt sich fast immer ein signifikanter Geschwindigkeitsvorteil der Varianten mit Budget. Größte Ausnahme hiervon bildet die Projektion des entfernten SVs auf die 50 verbleibenden SVs. Hier ist der konstante Mehraufwand ($B^2 = 2500$) größer als die gesamte Datenmenge ($N = 1000$).

Tabelle 6.11.: Budget-Testprofile

(1) (1) aus T. 6.1 mit $C \in \{10^{-4}, \ldots, 10^{2}\}$ optimiert
(2) (41) aus T. 6.1 (MICRA$^{\epsilon,\zeta}$-Margin, λ-Trick, Pegasos), $I = 100, H = 0$, Modell = Standard, in Primalform implementiert, ohne Budget
(3) (2) in Dualform, $B = 50$, Simples Entfernen, Auswahlheuristik: (4.13)
(4) (2) in Dualform, $B = 10$, Simples Entfernen, Auswahlheuristik: (4.13)
(5) (2) in Dualform, $B = 50$, Simples Entfernen, Auswahlheuristik: neuester SV
(6) (2) in Dualform, $B = 10$, Simples Entfernen, Auswahlheuristik: neuester SV
(7) (2) in Dualform, $B = 50$, Projektion auf alle SVs, Auswahlheuristik: (4.13)
(8) (2) in Dualform, $B = 50$, Projektion auf 10 SVs, Auswahlheuristik: (4.13)
(9) (2) in Dualform, $B = 10$, Projektion auf alle SVs, Auswahlheuristik: (4.13)
(10) (2) in Dualform, $B = 50$, Projektion auf alle SVs, Auswahlheuristik: neuester SV
(11) (2) in Dualform, $B = 50$, Projektion auf 10 SVs, Auswahlheuristik: neuester SV
(12) (2) in Dualform, $B = 10$, Projektion auf alle SVs, Auswahlheuristik: neuester SV

Tabelle 6.12.: Genauigkeit mit Budget

Data	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
6.1a	98.59	**98.88**∘	98.87∘	98.58	98.57	98.00•	98.87∘	98.86∘	98.85∘	98.87∘	98.86∘	98.83∘
6.1b	86.09	86.14	85.92	85.04•	85.94	83.67•	**86.18**	86.10	86.12	86.13	86.11	86.09
6.1c	83.72	83.87∘	76.69•	70.38•	83.34•	80.86•	**83.89**∘	83.88∘	83.86	83.86∘	83.86∘	83.82
6.1d	74.35	**74.42**	72.44•	70.36•	73.28•	67.90•	74.25	74.40	74.21	**74.42**	74.41	74.27

∘, • statistically significant improvement or degradation

Tabelle 6.13.: Trainingszeit mit Budget vs. Primalform ohne Budget

Data	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
6.1a	1.87	0.33•	0.15•	0.29•	0.13•	5.05∘	0.73•	0.35•	4.85∘	0.66•	0.33•
6.1b	2.64	0.53•	0.28•	0.78•	0.34•	11.05∘	1.35•	0.86•	10.94∘	1.86	0.70•
6.1c	3.11	1.03•	0.26•	0.40•	0.20•	15.41∘	2.29	0.86•	14.86∘	1.75•	0.85•
6.1d	4.43	1.06•	0.26•	0.47•	0.31•	26.43∘	3.95	1.31•	21.54∘	2.50	1.15•

∘, • statistically significant degradation or improvement

6.3 Realdaten

Auf einigen realen UCI-Datensätzen [FA10] sollen nun interessante „Grundkonfigurationen“ (Tabelle 6.14) getestet werden. Damit ist eine Kombination aus Margin- und Update-Prozedur gemeint. Dabei wird jede Grundkonfiguration in jeder Batch-Variante, aber ohne reduzierte Epochen ausgeführt. Alle Modelle werden mit 1 augmentiert und erhalten maximal 100 Epochen. Aufgrund des enormen Rechenaufwands werden alle Parameter *greedy* in einer zweifachen Kreuzvalidierung optimiert. Da die Primalform verwendet wird, wird auf die Budget-Erweiterung verzichtet. Für die relativ große Anzahl an Datensätzen und Klassifizierern sollte eine bequeme Interpretation der Ergebnisse in Tabelle 6.15 anhand des zweiseitigen Nemenyi-Tests wie in [Dem06] möglich sein. Dafür muss man zunächst mit dem Friedmann-Test wie in [Dem06] die Hypothese wiederlegen, dass alle Konfigurationen gleich gut sind, sie also den selben durchschnittlichen Rang haben müssten. Für $N = 9$ Datensätze und $k = 13$ Klassifizierer ergibt sich $\chi_F^2 = 46.97$ und $F_F = 6.158$. Für das Signifikanzniveau $\alpha = 0.05$ muss die Hypothese bei einem kritischen Wert von $F(12, 96) = 1.854$ abgelehnt werden. Damit lautet für das selbe Signifikanzniveau die *critical distance* ‚kritische Distanz‘ für die durchschnittlichen Ränge 6.082 ($q_{0.05} = 3.313$). Ist der durchschnittliche Rang einer Konfiguration also um $CD = 6.082$ höher als der einer anderen, so ist diese signifikant ($\alpha = 0.05$) schlechter (über alle Datensätze zusammen gesehen). Bezüglich der SVM trifft das auf (8)–(11) zu. Da sich also reines MICRA$^{\epsilon,\zeta}$ nicht einmal mit den Batch-Varianten verbessert, liegt die Vermutung nahe, dass die *greedy* Parameter-Optimierung für MICRA$^{\epsilon,\zeta}$ nicht geeignet ist. Folglich bleibt auch Pegasos mit MICRA$^{\epsilon,\zeta}$-Margin hinter seinen Erwartungen zurück. Immerhin verbessern die erweiterten Batch-Varianten die entsprechenden Ränge deutlich (wenn auch nicht signifikant). Dies ist auch bei den anderen Konfigurationen der Fall. Zu erwähnen bleibt, dass für die letzten drei Datensätze der Parameter C der SVM stark verringert wird, da sonst keine schnelle Konvergenz möglich ist. Außerdem wird hier die SMO-Variante aus Weka verwendet, da mit der SVM aus [CL11] unerwartete Probleme bei der Konvertierung von nominalen zu binären Attributen auftreten.

Tabelle 6.14.: Testprofile für Realdaten

(1) (1) aus T. 6.1 mit $C \in \{10^{-6}, \ldots, 10^{0}\}$ optimiert, für die letzten drei $C \in \{10^{-4}, \ldots, 10^{-3}\}$
(2) Uneven *margin*, konstante Lernrate, Standard-Modell
(3) Uneven *margin*, konstante Lernrate, längster Überlebender
(4) Uneven *margin*, konstante Lernrate, gewichtete Abstimmung
(5) Uneven *margin*, PA, Standard-Modell
(6) Uneven *margin*, PA, längster Überlebender
(7) Uneven *margin*, PA, gewichtete Abstimmung
(8) MICRA$^{\epsilon,\zeta}$-Margin, MICRA$^{\epsilon,\zeta}$-Update, Standard-Modell
(9) MICRA$^{\epsilon,\zeta}$-Margin, MICRA$^{\epsilon,\zeta}$-Update, längster Überlebender
(10) MICRA$^{\epsilon,\zeta}$-Margin, MICRA$^{\epsilon,\zeta}$-Update, gewichtete Abstimmung
(11) MICRA$^{\epsilon,\zeta}$-Margin, Pegasos, Standard-Modell
(12) MICRA$^{\epsilon,\zeta}$-Margin, Pegasos, längster Überlebender
(13) MICRA$^{\epsilon,\zeta}$-Margin, Pegasos, gewichtete Abstimmung

Tabelle 6.15.: Genauigkeit auf Realdaten

Data	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
wbc	96.60	96.11	96.35	96.61	96.14	96.37	96.41	95.42•	95.24•	95.21•	95.62•	96.15	**96.84**
ionos.	87.10	86.98	86.89	87.15	86.95	86.81	86.81	86.50	85.59•	86.18	86.25	86.93	**87.86**
liver	69.27	69.00	69.11	69.25	68.74	68.54	69.32	67.00•	68.16	67.96	67.09	69.24	**69.48**
sonar	**77.50**	74.63	75.19•	75.82	75.96	74.46•	76.35•	74.92	75.34	75.44	72.66•	75.57•	76.16
tictactoe	**98.33**	98.32	98.31	98.31	**98.33**	**98.33**	**98.33**	98.15	98.16	98.28	98.08	98.25	98.27
vote	95.72	95.68	95.63	95.79	95.83	95.76	**95.90**	95.47	95.42	95.33	95.56	95.77	95.45
credit	76.19	71.78•	73.93•	72.90•	74.67	76.39	**78.49**◦	67.20•	67.57•	67.46•	65.87•	66.68•	67.23•
wdbc	**93.83**	91.00•	91.39•	91.69•	90.63•	91.18•	91.25•	90.67•	91.11•	90.98•	90.21•	90.49•	90.48•
wpbc	**78.11**	73.64•	77.09	77.51	73.29•	75.79•	75.39•	74.03•	76.13	75.04	76.29•	76.29•	76.29•
ØRang	2.44	7.67	5.89	3.67	6.11	6.11	3.56	10.78	9.67	10.11	10.89	7.22	5.78

◦, • statistically significant improvement or degradation

7 Fazit

Diese Arbeit bietet Ausblick auf einige weitere Themen und Aufgaben. Welchen Vorteil (normierende) Kernelfunktionen bieten ist noch ungeklärt. Wegen der großen Vielfalt ist nicht nur hier noch Raum für weitere Experimente. In [SSS07] wird vorgeschlagen, b aus (2.1) explizit zu behandeln und so den stochastischen Gradienten gemäß der SVM zu korrigieren. Es können noch weitere Verbesserungen hinzugefügt werden. Das erstellte Framework ist z. B. erweiterbar um die zahlreichen weiteren Lernschritte aus [WV10], welche ein Budget auf SVs und den sogenannten *ramp-loss* gegen *outliers* direkt in (4.19) und dessen analytische Lösung integrieren. Die Tests auf den künstlichen und realen Datensätzen zeigen, dass das erweiterte Perzeptron oft mit der SVM mithalten kann. Hauptproblem bleibt jedoch die Vielfalt an Kombinationsmöglichkeiten und Parametern, die es zu selektieren gilt.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmen- und Prozedurenverzeichnis

Abkürzungsverzeichnis

SVM Supportvektormaschine
SV Supportvektor
PA Passive-Aggressive
MICRA$^{\epsilon,\zeta}$ Mistake-Controlled Rule Algorithm
Pegasos Primal Estimated sub-GrAdient SOlver for SVM
GUI Graphical User Interface
RBF Radial-Basis-Funktion

Literaturverzeichnis

[Bis07] Christopher M. Bishop: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. Corr. 2nd printing Auflage, October 2007.

[CDK+06] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz und Yoram Singer: *Online Passive-Aggressive Algorithms*. Journal of Machine Learning Research, 7:551–585, March 2006.
Erreichbar unter: `http://www.jmlr.org/papers/volume7/crammer06a/crammer06a.pdf`.

[CL11] Chih-Chung Chang und Chih-Jen Lin: *LIBSVM: A library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CST00] Nello Cristianini und John Shawe-Taylor: *An introduction to support vector machines : and other kernel-based learning methods*. Cambridge University Press, 1 Auflage, March 2000.

[Dem06] Janez Demsar: *Statistical Comparisons of Classifiers over Multiple Data Sets*, 2006.

[EMH05] Yasser EL-Manzalawy und Vasant Honavar: *WLSVM: Integrating LibSVM into Weka Environment*, 2005. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[FA10] A. Frank und A. Asuncion: *UCI Machine Learning Repository*, 2010.
Erreichbar unter: `http://archive.ics.uci.edu/ml`.

[FS99] Yoav Freund und Robert E. Schapire: *Large Margin Classification Using the Perceptron Algorithm*. Machine Learning, 37(3):277–296, 1999.
Erreichbar unter: `www.cse.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf`.

[GHW00] Thore Graepel, Ralf Herbrich und Robert C. Williamson: *From Margin to Sparsity*. In: *Advances in Neural Information Processing Systems (NIPS) 13*, Seiten 210–216, 2000.
Erreichbar unter: `http://citeseer.ist.psu.edu/401898.html`.

[Hay08] Simon Haykin: *Neural Networks and Learning Machines*. Prentice Hall, 3 Auflage, November 2008.
Erreichbar unter: `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0131471392`.

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann und I. Witten: *The WEKA data mining software: an update*. Special Interest Group on Knowledge Discovery and Data Mining Explorer Newsletter, 11(1):10–18, November 2009.
Erreichbar unter: `http://dx.doi.org/10.1145/1656274.1656278`, `doi:10.1145/1656274.1656278`.

[KW07] Roni Khardon und Gabriel Wachman: *Noise Tolerant Variants of the Perceptron Algorithm*. J. Mach. Learn. Res., 8:227–248, 2007.
Erreichbar unter: `http://portal.acm.org/citation.cfm?id=1314506`.

[LZH+02] Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor und Jaz S. Kandola: *The Perceptron Algorithm with Uneven Margins*. In: *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, Seiten 379–386, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
Erreichbar unter: `http://portal.acm.org/citation.cfm?id=645531.655993`.

[Mit97] Tom M. Mitchell: *Machine Learning*. McGraw-Hill, New York, 1997.

[Nov62] Albert B. Novikoff: *On convergence proofs for perceptrons*. In: *Proceedings of the Symposium on the Mathematical Theory of Automata*, Band 12, Seiten 615–622, 1962.
Erreichbar unter: `http://citeseer.comp.nus.edu.sg/context/494822/0`.

[Ros62] Frank Rosenblatt: *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, 1962.

[SSS07] Shai S. Shwartz, Yoram Singer und Nathan Srebro: *Pegasos: Primal Estimated sub-GrAdient SOlver for SVM*. In: *ICML '07: Proceedings of the 24th international conference on Machine learning*, Seiten 807–814, New York, NY, USA, 2007. ACM.
Erreichbar unter: http://ttic.uchicago.edu/~{}shai/papers/ShalevSiSr07.pdf, doi:10.1145/1273496.1273598.

[TST07] Petroula Tsampouka und John Shawe-Taylor: *Approximate maximum margin algorithms with rules controlled by the number of mistakes*. In: *Proceedings of the 24th international conference on Machine learning*, ICML '07, Seiten 903–910, New York, NY, USA, 2007. ACM.
Erreichbar unter: http://doi.acm.org/10.1145/1273496.1273610, doi:http://doi.acm.org/10.1145/1273496.1273610.

[WCV10] Zhuang Wang, Koby Crammer und Slobodan Vucetic: *Multi-Class Pegasos on a Budget*. In: *ICML*, 2010.

[WV10] Zhuang Wang und Slobodan Vucetic: *Online Passive-Aggressive Algorithms on a Budget*. In: *JMLR W&C Proc. Int. Conf. on Artificial Intelligence and Statistics*, AISTATS '10, Seiten 908–915, 2010.