Skript Maschinelles Lernen

Jan Poland

Mai 2011

Inhaltsverzeichnis

0 Vorbemerkung

Dieses Skript ergänzt die Unterrichtsmaterialien des Teils *Maschinelles Lernen* der Vorlesung *Computational Intelligence*. Es wurde geschrieben, um den Studenten das Nachvollziehen und Rekapitulieren des Stoffes zu Hause zu erleichtern. Es sollte zusammen mit den Folien und den Mitschrieben aus der Vorlesung studiert werden. Insbesondere alle Abbildungen befinden sich in den Folien, aber auch manche zusätzliche Beispiele.

Das Skript enthält den vollständigen "roten Faden" der Vorlesung, es ersetzt die Vorlesung aber nicht. In der Vorlesung wurden einige Themen etwas ausführlicher behandelt als hier. Für die Prüfungen gilt: wer das Skript vollständig verstanden hat, sollte alle Prüfungen bestehen können. Um die Bestnote zu erreichen, ist darüberhinausgehendes Wissen nötig, wie es z.B. in der Vorlesung vermittelt wurde.

0.1 Notation

Mit $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ bezeichnen wir die natürlichen, ganzen und reellen Zahlen. $\mathbb{R}^d$ ist der d-dimensionale Eulklidische Raum. $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{F}$, $\mathcal{J}$ etc. sind *Mengen*, wobei $\mathcal{X}$ meistens die Eingabemenge und $\mathcal{Y}$ die Ausgabemenge ist. Oft ist $\mathcal{X} \subset \mathbb{R}^d$ Untermenge des Euklidischen Raums und $\mathcal{Y} \subset \mathbb{R}$ Untermenge der reellen Zahlen. Kleinbuchstaben $x, y, \ldots$ bezeichnen Zahlen oder Vektoren. Üblicherweise ist $x \in \mathcal{X}$ und $y \in \mathcal{Y}$. Ferner sind üblicherweise $i, j, \ldots$ natürliche oder ganzzahlige Indexvariablen, d ist die Dimension des (Eingabe-)Raums, und n ist die Anzahl der Trainingsdaten. Funktionen werden oft mit $f, g, \phi, \ldots$ bezeichnet. Grosse Buchstaben beziehen sich auf Matrizen $(A, B, \ldots)$ oder machmal auf Zufallsvariablen. Für eine Zufallsvariable X bezeichnet $\mathbb{E}X = \int x d\mu_X(x)$ ihren Erwartungswert.

1 Einführung Maschinelles Lernen

1.1 Grundbegriffe

Maschinelles Lernen umfasst Methoden, die es erlauben, automatisch aus Daten Informationen zu gewinnen. Oft sind Daten mit Messfehlern behaftet, daher basiert Masichnelles Lernen oft auf *statistischen* Verfahren. Wir unterscheiden grundsätzlich zwei verschiedene Aufgabenstellungen:

- *Überwachtes Lernen*: Hier sind Datenpunkte zusammen mit *Labels* gegeben, und die Aufgabe besteht darin, Gesetzmässigkeiten zu erkennen, so dass die Labels für neue Datenpunkte vorhergesagt werden können.

- *Unüberwachtes Lernen*: Hier sind Datenpunkte ohne Labels gegeben, und die Aufgabe besteht darin, die Datenpunkte zu nach Ähnlichkeiten gruppieren. Das wird auch als *Clustering* bezeichnet.

Beim überwachten Lernen unterscheiden wir weiterhin zwischen *Regression* und *Klassifikation*: Regression bezeichnet Aufgabenstellungen und Methoden, bei denen die Labels reel- oder vektorwertig sind. Hier besteht die Aufgabe darin, eine "Ausgeleichsfunktion" durch die Datenpunkte zu finden, die einerseits die Daten gut approximiert und andererseits die Messfehler nicht abbildet. Bei Klassifikationsproblemen sind hingegen die gegebenen Datenpunkte in Klassen unterteilt, und die Aufgabe besteht darin, eine Verallgemeinerung zu lernen, die es erlaubt, neue Datenpunkte zu klassifizieren.

Wir legen uns hier auf folgenden mathematischen Rahmen fest: Es sei $\mathcal{X}$ eine beliebige Menge. Oft ist $\mathcal{X} \subset \mathbb{R}^d$, eine Untermenge des d-dimensionalen Euklidischen Raums, und wird *Featureraum*[1] genannt. Für ein Regressionsproblem sei $\mathcal{Y} = \mathbb{R}$, gegeben sind Datenpunkte $x_1, x_2, \ldots, x_n \in \mathcal{X}$ mit zugehörigen Labels $y_1, y_2, \ldots, y_n \in \mathcal{Y}$, und gesucht ist eine Regressionsabbildung $f : \mathcal{X} \to \mathcal{Y}$. Üblicherweise ist dasjenige f aus der betrachteten Klasse der Regressionsfunktionen gesucht, das ein *Fehlerkriterium* minimiert, oft den quadratischen Fehler. Eine Begründung hierzu werden wir im Verlauf der Vorlesung kennenlernen.

Für ein Klassifikationsproblem ist $\mathcal{Y} \subset \mathbb{Z}$, gegeben sind wieder Datenpunkte $x_1, x_2, \ldots, x_n \in \mathcal{X}$ mit zugehörigen Labels $y_1, y_2, \ldots, y_n \in \mathcal{Y}$, und gesucht ist ein Klassifikator $f : \mathcal{X} \to \mathcal{Y}$. Ein häufiger Spezialfall ist die *binäre Klassifikation*, bei der $\mathcal{Y} = \{0, 1\}$ ist.

1.1.1 Beispiel. Es sei $\mathcal{X} = \{\text{Arbeit}, \text{Sportverein}, \text{Supermarkt}\}$, $\mathcal{Y} = \mathbb{R}$, mit der folgenden Interpretation: die Labels bezeichnen die Fahrzeit in Minuten, die Person A zum angegebenen Ziel in $\mathcal{X}$ benötigt. Gegeben sind folgende Datenpunkte: (Arbeit, Supermarkt, Arbeit, Supermarkt, Sportverein) mit Labels (15, 5, 20, 20, 30). Die Regressionsaufgabe besteht hier darin, für jedes Ziel eine repräsentative Fahrzeit zu berechnen.

1.1.2 Beispiel. Wieder sei $\mathcal{X} = \{\text{Arbeit}, \text{Sportverein}, \text{Supermarkt}\}$, $\mathcal{Y} = \{\text{kein Stau}, \text{Stau}\}$, mit der folgenden Interpretation: die Labels beschreiben, ob auf der Fahrt ein Stau aufgetreten ist.Gegeben sind folgende Datenpunkte: (Arbeit, Supermarkt, Arbeit, Supermarkt, Sportverein) mit Labels (kein Stau, kein Stau, Stau, Stau, kein Stau). Die Klassifikationsaufgabe besteht hier darin, für jedes Ziel eine Vorhersage Stau oder nicht zu erstellen.

[1] Oft werden Lernalgorithmen nicht auf Rohdaten angewendet, sondern auf bereits vorverarbeiteten Daten, nach einer sogenannten *Featureextraktion*.

1.2 Mittelwert und Normalverteilung

1.2.1 Beispiel. Es sei $\mathcal{X} = \{\text{Arbeit}\}$, $\mathcal{Y} = \mathbb{R}$, mit der folgenden Interpretation: die Labels bezeichnen die Fahrzeit in Minuten, die Person A für den Weg zur Arbeit benötigt. Gegeben sind folgende Datenpunkte: (Arbeit, Arbeit, Arbeit) mit Labels (15, 20, 17). Die Regressionsaufgabe besteht hier darin, eine repräsentative Zahl (Schätzung $\hat{y}$) für die Fahrzeit zur Arbeit zu bestimmen.

In diesem einfachsten Regressionsfall ist die Featuremenge einelementig und nicht weiter interessant. Wir haben also Messdaten $y_1, y_2, \ldots, y_n$ und suchen einen Schätzung $\hat{y}$. Im obigen und in vielen anderen Beispielen kann man die Daten mathematisch als *Zufallsvariablen* modellieren. Dann suchen wir nicht nur nach einer einzigen Zahl $\hat{y}$, die unsere Daten beschreibt, sondern nach einer *Verteilung*. Im Falle von Regressionsproblemen nehmen wir in den allermeisten Fällen an, dass die Daten *normalverteilt* sind: $y \sim \mathrm{N}(\mu, \sigma^2)$ steht für die Verteilungsdichte

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}. \tag{1}$$

Hier heisst μ der *Mittelwert* oder *Erwartungswert* der Normalverteilung, und σ^2 die *Varianz*.

Warum legt man gerne die Normalverteilung zugrunde? Die Antwort liefert der folgende *Zentrale Grenzwertsatz*:

1.2.2 Satz. *Gegeben seien unabhängige und identisch verteilte Zufallsvariablen $X_1, X_2, \ldots$. Wenn der Erwartungswert μ und die Varianz σ^2 der Verteilung existieren, dann konvergiert*

$$\frac{X_1 + X_2 + \ldots X_n - n\mu}{\sqrt{n\sigma^2}} \tag{2}$$

für $n \to \infty$ gegen die Standard-Normalverteilung N(0,1).

Dieser Satz sagt also aus, dass eine Verteilung, die aus *irgendwelchen* unabhängigen kleinen Einflüssen "zusammengesetzt" ist, immer annähernd eine Gaussverteilung ist, wenn ihre Varianz existiert. Deswegen trifft man bei physikalischen Messprozessen und den dazugehörigen Messfehlern üblicherweise auf Normalverteilungen.

Wir schauen uns jetzt an, was die Normalverteilungsannahme im Fall $\mathcal{Y} = \mathbb{R}$ und gegebenen Daten $y_1, y_2, \ldots, y_n \in \mathcal{Y}$ für die Schätzung der Verteilung bedeutet. Um eine Normalverteilung zu schätzen, müssen wir zwei Parameter schätzen, nämlich den Mittelwert μ und die Varianz σ^2. Für gegebende

Daten $y_1, y_2, \ldots, y_n \in \mathcal{Y}$ und gegebene Parameter μ und σ^2 ist die Wahrscheinlichkeitsdichte[2] proportional zu

$$p(y_1, y_2, \ldots, y_n) \sim \prod_{i=1}^{n} e^{-\frac{(y_i-\mu)^2}{2\sigma^2}} = e^{-\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2}}. \tag{3}$$

Die Frage ist jetzt: für die vorliegenden Daten $y_1, y_2, \ldots, y_n \in \mathcal{Y}$, welches Paar (μ, σ^2) ist am besten?

Wir präzisieren die Frage folgendermassen: unter welchem Paar (μ, σ^2) sind die beobachteten Daten am *wahrscheinlichsten*? Die Beantwortung dieser Frage führt zum *Maximum Likelihood Schätzer*.

1.2.3 Definition. Seien $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ Beobachtungen und $p(y|x, \theta)$ eine durch $\theta \in \Theta$ parametrierte Menge von Wahrscheinlichkeitsverteilungen oder -dichten. Ein Schätzer $\hat{\theta} \in \Theta$ heisst *Maximum Likelihood* (ML), falls

$$p(y_1, y_2, \ldots, y_n | x_1, x_2, \ldots, x_n, \hat{\theta}) = \max_{\theta \in \Theta} p(y_1, y_2, \ldots, y_n | x_1, x_2, \ldots, x_n, \theta) \tag{4}$$

gilt.

1.2.4 Satz. *Unter der Normalverteilungsannahme ist der ML-Schätzer für μ eindimensionale Daten $y_1, y_2, \ldots, y_n \in \mathbb{R}$ gleich dem Mittelwert $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. Dies gilt unabhängig von σ^2.*

Beweis. Wir suchen dasjenige $\mu \in \mathbb{R}$ so dass

$$e^{-\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2}} \tag{5}$$

maximal wird. Durch logarithmieren und negieren ist dies äquivalent zur Minimierung von

$$\sum_{i=1}^{n}(y_i - \mu)^2, \tag{6}$$

unabhängig von σ^2. Diese Funktion von μ kann ihr Minimum nur dort annehmen, wo ihre Ableitung verschwindet. Leiten wir die Funktion nach μ ab und setzen sie gleich 0, so erhalten wir

$$2\sum_{i=1}^{n}(y_i - \mu) = 0, \tag{7}$$

woraus die Behauptung folgt. □

[2]Kontinuierliche Verteilungen werden durch Dichten beschrieben. Während die Wahrscheinlichkeit eines Ereignisses, also einer Zahl, immer null ist, ist die Wahrscheinlichkeitsdichte wohldefiniert und im Allgemeinen (bei einer Normalverteilung: immer) von null verschieden.

1.2.5 Bemerkung. *Wie wir gerade gesehen haben, führt die Normalverteilungsannahme zum* Schätzer der kleinsten Quadrate*, der in sehr vielen Anwendungen für statistische Schätzung Verwendung findet.*

1.3 Eigenschaften von Schätzern

Sind ML-Schätzer immer "gut"? Der (negativen) Antwort darauf soll uns das nächste Beispiel näherbringen.

1.3.1 Beispiel (Taxiproblem). In einer Stadt gibt es N Taxis, die mit fortlaufenden Nummern $1 \dots N$ gekennzeichnet sind. Wir wollen N schätzen. Dafür setzen wir uns an eine Strassenkreuzung und notieren die Nummern der vorbeifahrenden Taxis.

Am Abend haben wir folgende Nummer notiert: 5, 7, 7, 7, 14, 84, 99, 125, 125, 126, 230, 281, 399, 412.

Folgendes einfache Argument führt uns direkt zum ML Schätzer: Würden wir $\hat{N} < 412$ wählen, so wäre die Wahrscheinlichkeit für die beobachteten Nummern null (die Beobachtung wäre so unmöglich). Ab $\hat{N} \geq 412$ sind die Schätzer theoretisch möglich. Je grösser wir $\hat{N}$ wählen, desto kleiner wird jedoch die Wahrscheinlichkeit für die Beobachtung, da jede Einzelbeobachtung unwahrscheinlicher wird. Folglich ist der ML-Schätzer genau $\hat{N} = 412$.

Andererseits ist es offensichtlich, dass wir in vielen Fällen nicht das Taxi mit der grössten Nummer beobachten werden. Wenn wir also immer $\hat{N}$ als das Maximum der beobachteten Nummern wählen, werden wir die wahre Grösse üblicherweise *unterschätzen*. Wir sagen dann, die Schätzung hat einen *Bias*. Wenn die Parametermenge Θ die Bildung eines Erwartungswertes zulässt, dann können wir einen Schätzer ohne Bias definieren:

1.3.2 Definition. Ein Schätzer $\hat{\theta} = \hat{\theta}\big((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\big)$, der als Funktion der Daten gegeben ist, heisst *erwartungstreu*, falls für jeden wahren Parameter $\theta \in \Theta$ gilt dass

$$\mathbb{E}_\theta \hat{\theta}\big((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\big) = \theta. \tag{8}$$

Hierbei bezeichnet $\mathbb{E}$ den Erwartungswert, $\mathbb{E}_\theta$ ist der Erwartungswert unter dem Parameter θ. Also ist

$$\mathbb{E}_\theta y = \int_{\mathcal{Y}} y \cdot dp(y|\theta). \tag{9}$$

(Hier ist $p(\cdot|\theta)$ die Wahrscheinlichkeitsdichte unter dem Parameter θ, bezüglich derer integriert wird, daher die Schreibweise $dp(\cdot|\theta)$.)

1.3.3 Beispiel (Taxiproblem). Für das Beispiel aus 1.3.1 können wir leicht einen erwartungstreuen Schätzer konstruieren: $\hat{N} = \max_{1\leq i\leq n} y_i + \min_{1\leq i\leq n} y_i - 1$ hat diese Eigenschaft. Zum Beweis sehen wir, dass

$$\begin{aligned} \mathbb{E}\hat{N} &= \mathbb{E}\big(\max_{1\leq i\leq n} y_i + \min_{1\leq i\leq n} y_i - 1\big) && (10)\\ &= \mathbb{E}\big(N - (N - \max_{1\leq i\leq n} y_i) + \min_{1\leq i\leq n} y_i - 1\big) && (11)\\ &= N - \mathbb{E}(N - \max_{1\leq i\leq n} y_i) + \mathbb{E}(\min_{1\leq i\leq n} y_i - 1) && (12)\\ &= N. && (13) \end{aligned}$$

Hieraus folgt auch, dass der vorige ML-Schätzer $\hat{N} = \max_{1\leq i\leq n} y_i$ nicht erwartungstreu ist.

1.3.4 Satz. *Der Mittelwertschätzer ist ein erwartungstreuer Schätzer für den Paramter μ der Normalverteilung.*

Der Beweis ist eine einfache Übung (und findet sich auch in den Folien).

Analog zum Erwartungswert kann man für die Normalverteilung auch Schätzer für die Varianz σ^2 angeben. Material dazu wurde in der Vorlesung präsentiert und findet sich in den Folien. Wozu würde man die Varianz schätzen wollen? Die Varianz ist ein Mass für den *Messfehler*, mit dem die Daten behaftet sind. Auch kann man daraus Konfidenzintervalle für die Schätzung gewinnen. Diese fortgeschrittenen Themen werden in der Vorlesung nicht weiter vertieft, wir werden uns im folgenden darauf konzentrieren, *eine* Repräsentation (ein Modell) für die Daten zu lernen.

1.4 Lineare Regression

Nach dieser etwas umfangreicheren Vorarbeit für den einfachen Fall von eindimenionalen, reellen Labels wenden wir uns nun der Linearen Regression zu, die sich als eine einfache Erweiterung des bisher betrachteten Falles erweist.

1.4.1 Definition (Lineare Regression). Gegeben sind Daten als Paare von *Features* und Werten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, wobei $\mathcal{X} \subset \mathbb{R}^d$ und $\mathcal{Y} = \mathbb{R}$ ist. Das zugrunde liegende Modell sei

$$y = b^T x \tag{14}$$

wobei $b \in \mathbb{R}^d$ der Parametervektor (Spaltenvektor) ist (b übernimmt also die Funktion von θ zuvor). Wenn wir $\hat{b}$ nach der Methode der kleinsten Quadrate schätzen, erhalten wir einen Schätzer, der sowohl ML als auch erwartungstreu ist: die *lineare Regression*.

Der ML-Schätzer für b wird berechnet, indem wir die normalverteilte Dichtefunktion logarithmieren und negieren:

$$\sum_{i=1}^{n}(y - b^T x)^2 = (Y - b^T X)^T (Y - b^T X) = \min! \tag{15}$$

(wobei X und Y die Matrix bzw. Zeilenvektor sind, die sich aus spaltenweise Konkatenation von x_i und y_i ergeben). Mit den entsprechenden Werkzeugen aus der Vektoranalysis können wir den Gradienten aus dem Ausdruck berechnen, der verschwinden muss, damit ein Minimum angenommen werden kann:

$$0 = 2X^T X b^T - 2X^T Y \tag{16}$$

Dies lösen wir nach b auf und erhalten die *Formel für die lineare Regression*

$$\hat{b} = \left(\left(X^T X\right)^{-1} X^T Y\right)^T. \tag{17}$$

Ferner sehen wir, dass $\hat{b}$ linear in den Beobachtungen Y ist. Hieraus folgt leicht, dass $\hat{b}$ ein erwartungstreuer Schätzer ist (Übungsaufgabe).

Ein einfaches Beispiel mit Zahlen für eine lineare Regression findet sich in den Folien. Wir schauen uns kurz Beispiele für (auf den ersten Blick) "nichtlineare" Regression an, die sich als Spezialfälle der linearen Regression herausstellen.

1.4.2 Beispiel (Affine Regression). Wir betrachten das Modell $y = b^T x + c$. Durch *Augmentieren* des Featurevektors x durch Anfügen einer Zeile mit dem Wert 1 erhalten wir einen neuen Featurevektor $\tilde{x} = [x^T 1]^T$. Jetzt ist unser Modell linear: $y = \tilde{b}^T \tilde{x}$, und wir können es mit der oben beschriebenen Formel lösen.

1.4.3 Beispiel (Quadratische Regression). Dasselbe passiert, wenn wir ein quadratisches Modell $y = x^T Ax + b^T x + c$ schätzen wollen. Der neue Featurevektor $\tilde{x}$ muss nun alle rein und gemischt quadratischen Terme $x^{(i)}x^{(j)}$, die linearen Terme $x^{(i)}$ sowie den konstanten Term 1 enthalten (wir schreiben hier $x^{(i)}$ für die i-te Komponente des Vektors x, um Verwechslung mit dem i-ten Datenpunkt x_i zu vermeiden). Damit wird das Problem wieder ein lineares Regressionsproblem.

1.5 Bayes Regel

Bisher haben wir nach einem Schätzer $\hat{\theta} \in \Theta$ gesucht, ohne bestimmte θ anderen vorzuziehen. In vielen Anwendungen ist es jedoch günstig, eine

Struktur auf der Schätzermenge Θ vorzugeben, aufgrund derer bestimmte $\theta \in \Theta$ über andere bevorzugt werden sollten. Ein grundlegendes Prinzip, das mit Gewinn in vielen Verallgemeinerungsprozessen verwendet wird, ist als *Ockhams Sparsamkeitsprinzip (Occam's razor)* bekannt:

1.5.1 Ockhams Sparsamkeitsprinzip. Entitäten dürfen nicht über das Notwendige hinaus vermehrt werden.

1.5.2 Beispiel. Eine bekannte historische Anwendung von Ockhams Sparsamkeitsprinzip ist die Entscheidung, ob der Mensch an das heliozentrische Weltbild glauben soll oder an das geozentrische. Beobachtet man die Planetenbahnen und legt das geozentrische Weltbild zugrunde, so sind die Gesetzmässigkeiten, mit denen sich die Bahnen beschreiben lassen, sehr kompliziert. Unter dem heliozentrischen Weltbild genügen drei einfache mathematische Zusammenhänge, nämlich Keplers Gesetze, um die Plantenbahnen zu erklären. Lange bevor es Technologien wie Raumfahrt gab, war dies für die Wissenschaftler die überzeugende Begründung dafür, dass das heliozentrische Weltbild richtig und das geozentrische falsch ist.

Eine besonders einfache Formalisierung von Ockhams Sparsamkeitsprinzip in probabilistischen Termen ist *Bayes Regel*:

1.5.3 Bayes Regel. Sei Θ eine Menge Hypothesen und $\mathcal{Y}$ eine Menge möglicher Beobachtungen. Falls eine Hypothese $\theta \in \Theta$ zutrifft, so ist $p(y|\theta)$ die Wahrscheinlichkeitsverteilung oder -dichte, dass $y \in \mathcal{Y}$ beobachtet wird. Für eine Menge von Beobachtungen $Y = y_1, y_2, \ldots, y_n$ ist $p(Y|\theta) = \prod_i p(y_i|\theta)$ die zugehörige Wahrscheinlichkeit(sdichte). Ferner haben wir eine *Wahrscheinlichkeitsverteilung (oder -dichte) auf Theta*: für jedes $\theta \in \Theta$ ist $P(\theta)$ der sogenannte *Prior*. Nach Beobachtung von Y möchten wir jedem θ eine *a posteriori Wahrscheinlichkeit*, den *Posterior* $P(\theta|Y)$ zuordnen. Den Posterior berechnen wir mit Bayes Regel:

$$P(\theta|Y) = \frac{p(Y|\theta) \cdot P(\theta)}{\int_\Theta p(Y|\theta) dP(\theta)} = \frac{p(Y|\theta) \cdot P(\theta)}{p(Y)}. \tag{18}$$

Der Term $p(Y|\theta)$ wird auch als *Likelihood* bezeichnet. Dasjenige $\hat{\theta} \in \Theta$, das die Posterior-Wahrscheinlichkeit maximiert, nennen wir den *Maximum A-Posteriori (MAP) Schätzer*.

Falls der Prior tatsächlich eine Wahrscheinlichkeitsverteilung auf Θ beschreibt, so kann man Bayes Regel leicht mit der Formel der bedingten Wahrscheinlichkeiten beweisen. In den meisten Fällen jedoch entspricht der Prior nicht einer Wahrscheinlichkeitsverteilung, sondern lediglich einer *Modellannahme*

("Belief Prior"). Dann ist Bayes Regel zwar kein Satz mehr, aber immer noch ein in der Praxis oft sehr erfolgreiches Werkzeug.

Ein Beispiel für einen MAP-Schätzer werden wir weiter unten kennenlernen, wenn wir Neuronale Netze mit Weight Decay trainieren.

2 Neuronale Netze

Seit den 1960er Jahren beschäftigt sich die Forschung damit, Strukturen, die natürlichen neuronalen Netzen ähneln, künstlich nachzubilden und zu simulieren. Die anfängliche wichtigste Motivation dafür war es, künstliche Intelligenz zu schaffen. Dieses Ziel wurde über Jahrzehnte verfehlt. Nichtsdestotrotz sind aus diesen Forschungen nützliche Werkzeuge herausgekommen, die seitdem mit Erfolg in praktischen Anwendungen eingesetzt werden. Für uns in diesem Kurs soll eine neuronales Netz nichts weiter als eine *nichtlineare Regression* sein, also eine Funktion

$$y = f(x, \theta) \tag{19}$$

die von einem Eingangsdatum $x \in \mathcal{X}$ und einem Parametervektor $\theta \in \Theta$ abhängt. Wir verwenden den Begriff[3] des neuronalen Netzes hauptsächlich, weil er sich in der Fachwelt für solche nichtlinearen Regressionsmodelle vielfach etabliert hat.

2.1 Netzarchitektur und Aktivierungsfunktion

Prinzipiell könnte eine solche nichtlineare Regressionsfunktion f *irgendetwas* sein. Wir beschränken uns hier für die weitere Diskussion auf zwei Spezialfälle:

$$f(x, \theta) = b_0 + \sum_{i=1}^{h} b_i \tanh\left(w_{i,0} + \sum_{j=1}^{d} w_{i,j} x_j\right) \quad \text{und} \tag{20}$$

$$f(x, \theta) = \tanh\left(b_0 + \sum_{i=1}^{h} b_i \tanh\left(w_{i,0} + \sum_{j=1}^{d} w_{i,j} x_j\right)\right). \tag{21}$$

Dies entspricht einem *zweischichtigen neuronalen Netz* (vgl. die Abbildungen in den Folien) mit tanh-Aktivierungsfunktion in einer verdeckten Schicht mit h verdeckten Neuronen und linearer bzw. tanh-Aktivierungsfunktion in der Ausgabeschicht. Der Parametervektor θ besteht hier aus den b_i und den $w_{i,j}$. Diese Parameter werden auch *Gewichte* genannt, da sie bestimmen, wie

[3] Etwas präziser schränken wir uns auf *Feedforward-Netze* ein, also Architekturen ohne Rückführungen und Schleifen.

stark der Einfluss einer Eingangsgrösse auf die Aktivierung der verdeckten Neuronen sowie der Einfluss eines verdeckten Neurons auf die Ausgangsgrösse ist.

Warum beschränken wir uns auf diese beiden Spezialfälle? Es sind die beiden gebräuchlichsten Setups, einerseits für Regression (20) und andererseits für (binäre) Klassifikation (21). Im letzteren Fall werden die positiven Labels +1 und die negativen Labels -1 (für das Training ist es oft günstiger, die positiven Labels als +0.9 und die negativen als -0.9 zu setzen, ansonsten benötigt das Training länger zum konvergieren, und es besteht die Gefahr von Overfitting, s.u.). Es gibt mögliche Alternativen, die aber aus folgenden Gründen weit weniger bedeutend sind:

- Eine mehr als zweischichtige Topologie oder eine ganz andere Topologie: Satz 2.1.2 besagt, dass mit der zweischichtigen Topologie und mit tanh-Aktivierungsfunktion *jede beliebige* Funktion *beliebig genau* approximiert werden kann.
- Eine andere Aktivierungsfunktion: Verwenden wir lineare Aktivierungsfunktionen, so wird die Regression auf den Fall der linearen Regression 1.4.1 reduziert. Sigmoide Aktivierungsfunktionen werden durch die Analogie zu natürlichen Neuronalen Netzen motiviert, sind in der Praxis erfolgreich, und haben die in Satz 2.1.2 beschriebenen Approximationseigenschaften. Die *logistische Funktion*

$$f(x) = \frac{1}{1 + e^{-x}} \tag{22}$$

  ist ebenfalls populär, ist aber bis auf Skalierung und Verschiebung gleich zum Tangens Hyperbolicus. Der tanh ist dagegen symmetrisch um die 0 und daher vorteilhaft für das Training.
- Das *Perzeptron*, eines der frühesten betrachteten Modelle für ein Neuronales Netz, verwendet eine Stufenfunktion als Aktivierungsfunktion. Das hat den Nachteil, dass keine Standard-Optimierungsverfahren für das Training verwendet werden können.

2.1.1 Definition. Eine Klasse von Funktionen $f : \mathbb{R}^d \to \mathbb{R}$, $\mathcal{F}$, approximiert die stetigen Funktionen auf $\mathbb{R}^d$ beliebig genau, falls für jede stetige Funktion $g : \mathbb{R}^d \to \mathbb{R}$, für jede kompakte (= beschränkte und abgeschlossene) Teilmenge des $\mathbb{R}^d$ und für jedes $\epsilon > 0$ eine Funktion $f \in \mathcal{F}$ existiert, das

$$\max x \in C|f(x) - g(x)| < \epsilon \tag{23}$$

erfüllt.

Wir haben eine "stetige Funktion" nicht definiert. Intuitiv ist das eine Funktion, die man zeichnen kann, ohne den Stift abzusetzen. Eine exakte Definition, die ϵ-δ Definition, findet sich leicht in Lehrbüchern oder auf Wikipedia.

2.1.2 Satz. *Sei $\phi : \mathbb{R} \to \mathbb{R}$ eine stetige Funktion (messbar würde ausreichen, aber dies wollen wir hier nicht definieren). Dann gilt: Ein Neuronales Netz mit Architektur (20) und Aktivierungsfunktion ϕ anstelle des in (20) verwendeten tanh ist genau dann in der Lage, die stetigen Funktionen auf $\mathbb{R}^d$ beliebig genau zu approximieren, wenn ϕ kein Polynom ist. Insbesondere gilt diese Eigenschaft für die Architektur in (20).*

Für das *Training* von Neuronalen Netzen, das wir als Nächstes betrachten wollen, benötigen wir einige Grundlagen zur Optimierung.

2.2 Einschub: Optimierung

Optimierungsprobleme und ihre Lösung werden in dieser Vorlesung an vielen Stellen eine wichtige Rolle spielen, sie gehören zu den wichtigsten Methoden der Computational Intelligence. Ein *einkriterielles Optimierungsproblem* ist gegeben durch eine Funktion $f : \mathcal{X} \to \mathbb{R}$, die auf einer Menge $\mathcal{X}$ definiert ist. Das Optimierungsproblem (Minimierungsproblem) besteht dann darin, das Minimum von f auf $\mathcal{X}$ zu finden

$$x^* = \arg\min_{x \in \mathcal{X}} f(x). \tag{24}$$

In den meisten Fällen ist $\mathcal{X} \subset \mathbb{R}^d$ eine Untermenge des d-dimensionalen Euklidischen Raums.

Ein Maximierungsproblem kann selbstverständlich durch Minimierung der negativen Funktion $-f$ gelöst werden.

Je nachdem, welche Eigenschaften f und $\mathcal{X}$ haben, kann das Optimierungsproblem sehr einfach bis sehr schwer zu lösen sein. Im schwierigsten Fall gleicht die Optimierung der sprichwörtlichen Suche nach der Nadel im Heuhaufen, und es gibt keine wesentlich effizientere Methode, als die gesamte Menge $\mathcal{X}$ zu enumerieren. Im einfachsten Fall ist die Suche nach dem Optimum nicht anspruchsvoller als die Lösung eines linearen Gleichungssystems. Im Folgenden führen wir wichtige Klassen von Optimierungsproblemen ein, für die (teilweise nur einigermassen) effiziente Algorithmen zur Lösung bekannt sind. Mehr noch, es gibt frei verfügbare und kommerzielle Software zur Lösung dieser Problemklassen, sogenannte *Löser*.

Der Suchraum $\mathcal{X}$ wird in den folgenden Problemklassen immer durch sogenannte *Nebenbedingungen* (*Constraints*) spezifiziert. Weiterhin sei die Funktion f in den kontinuierlichen Variablen mindestens zweimal stetig differenzierbar, so dass alle Ableitungen bis zur zweiten berechnet werden können.

2.2.1 Nichtlineares Programm (NLP). Durch

$$\min f(x) \tag{25}$$
$$\text{wobei } g(x) \leq 0 \tag{26}$$
$$\text{und } h(x) = 0 \tag{27}$$

mit drei zweimal stetig differenzierbaren Funktionen $f : \mathbb{R}^d \to \mathbb{R}$, $g : \mathbb{R}^d \to \mathbb{R}^m$, und $h : \mathbb{R}^d \to \mathbb{R}^n$ (g und h dürfen also vektorwertig sein), wird ein nichtlineares Programm spezifiziert. Ein sehr leistungsfähiger freier Löser für nichtlineare Programme ist zur Zeit IpOpt.

Will man ein nichtlineares Programm mit einer entsprechenden Software, beispielsweise IpOpt, lösen, so muss man die Funktionen f, g, und h in geeigneter Weise spezifizieren. Das geschieht (wenn keine weiteren Werkzeuge wie Modellierungssprachen verwendet werden) als *Callback-Funktionen*: Die Funktionen und ggf. ihre Ableitungen werden vom Anwender implementiert, und Referenzen auf diese Funktionen werden dem Löser übergeben. Die folgenden zwei Spezialfälle kommen ohne dies aus, da hier die Funktionen komplett via Matrizen spezifiziert werden können.

2.2.2 Lineares Programm (LP). Sei $c \in \mathbb{R}^d$, $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $A^{eq} \in \mathbb{R}^{n \times d}$, und $b^{eq} \in \mathbb{R}^n$. Dann definiert

$$\min c^T x \tag{28}$$
$$\text{wobei } Ax \leq b \tag{29}$$
$$\text{und } A^{eq} x = b^{eq} \tag{30}$$

ein lineares Programm.

2.2.3 Quadratisches Programm (QP). Sei $c \in \mathbb{R}^d$, $Q \in \mathbb{R}^{d \times d}$, $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $A^{eq} \in \mathbb{R}^{n \times d}$, und $b^{eq} \in \mathbb{R}^n$. Dann definiert

$$\min c^T x + x^T Q x \tag{31}$$
$$\text{wobei } Ax \leq b \tag{32}$$
$$\text{und } A^{eq} x = b^{eq} \tag{33}$$

ein quadratisches Programm.

Ein freier Löser für LP und QP ist beispielsweise CLP. Quadratische Programme werden wir für das Training von Support Vektor Maschinen benötigen.

In den obigen Optimierungsproblemklassen war x immer ein Element aus $\mathbb{R}^d$, das lediglich durch die Nebenbedingungen weiter eingeschränkt wird. Insbesondere ist üblicherweise der Suchraum $\mathcal{X}$ topologisch zusammenhängend. Das ist nicht mehr der Fall bei *Mixed Integer Problemen* (MIP), in denen einige oder alle der Variablen ganzzahlig sein müssen.

2.2.4 Mixed Integer Lineares Programm (MILP). Sei $c \in \mathbb{R}^d$, $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $A^{eq} \in \mathbb{R}^{n \times d}$, $b^{eq} \in \mathbb{R}^n$, und $\mathcal{J} \subset \{1, \ldots, d\}$ eine Indexmenge. Dann definiert

$$\min c^T x \tag{34}$$
$$\text{wobei } Ax \leq b \tag{35}$$
$$\text{und } A^{eq}x = b^{eq} \tag{36}$$
$$\text{und } x_i \in \mathbb{Z} \text{ für } \; i \in J \tag{37}$$

ein gemischtes integer lineares Programm.

2.2.5 Mixed Integer Quadratisches Programm (MIQP). Sei $c \in \mathbb{R}^d$, $Q \in \mathbb{R}^{d \times d}$, $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $A^{eq} \in \mathbb{R}^{n \times d}$, $b^{eq} \in \mathbb{R}^n$, und $\mathcal{J} \subset \{1, \ldots, d\}$ eine Indexmenge. Dann definiert

$$\min c^T x + x^T Q x \tag{38}$$
$$\text{wobei } Ax \leq b \tag{39}$$
$$\text{und } A^{eq}x = b^{eq} \tag{40}$$
$$\text{und } x_i \in \mathbb{Z} \text{ für } \; i \in J \tag{41}$$

ein gemisches integer quadratisches Programm.

Ein bekannter kommerzieller Löser für MILP und MIQP ist Cplex. Freie Löser für MILP (mit teilweise Unterstützung für MIQP) sind CBC und SCIP (letzterer ist nur für akademische Benutzer frei).

Schliesslich gibt es noch, analog zum NLP, das MINLP. Für diese Klasse gibt es jedoch noch keine effizienten allgemeinen Löser.

Besonders einfach lassen sich Optimierungsprobleme lösen, wenn sie *konvex* sind. Ein Optimierungsproblem ist konvex, wenn sowohl der Suchraum $\mathcal{X}$ als auch die Funktion f konvex sind.

2.2.6 Definition. Eine Menge $\mathcal{X} \subset \mathbb{R}^d$ heisst *konvex*, falls für alle $x_1, x_2 \in \mathcal{X}$ und $0 \leq \alpha \leq 1$ gilt dass $(1 - \alpha)x_1 + \alpha x_2 \in \mathcal{X}$ ist.

2.2.7 Definition. Eine Funktion $f : \mathcal{X} \to \mathbb{R}$ ist *konvex*, wenn sowohl die Menge $\mathcal{X}$ also auch der *Epigraph* der Funktion, die Menge $\{\tilde{x} \in \mathbb{R}^{d+1} : \tilde{x}_{1\ldots d} \in \mathcal{X} \text{ und } \; \tilde{x}_{d+1} \geq f(\tilde{x}_{1\ldots d})\}$, konvex sind.

Lineare Programme sind immer konvex. Quadratische Programme sind üblicherweise konvex, nämlich dann, wenn die Matrix Q symmetrisch und positiv semidefinit ist, d.h. dass alle ihre Eigenwerte ≥ 0 sind. Nichtlineare Programme sind nicht immer konvex.

Konvexe Optimierungsprobleme haben die Eigenschaft, dass es nur ein *globales Optimum* gibt, und dass jedes *lokale Optimum* auch global ist. Ein lokales Optimum ist ein Punkt $x \in \mathcal{X}$, der optimal in einer bestimmten Umgebung ist (aber nicht notwendigerweise auf ganz $\mathcal{X}$). Eine Funktion mit mehreren lokalen Optima heisst *multimodal* und ist evtl. schwierig (global) zu optimieren. Konvexe Funktionen sind dagegen immer *unimodal*.

Während die praktische Lösung von LP, QP, MILP und MIQP bei gegebenen Matrizen und gegebenem Löser (unter der Annahme, dass der Löser in vernünftiger Rechenzeit ein richtiges Ergebnis liefert) leicht zu implementieren ist, da lediglich die Matrizen dem Löser übergeben werden müssen, benötigen NLP evtl. noch die Implementierung der Ableitungen. Löser, die die Ableitungen verwenden, sind Lösern, die dies nicht tun, leistungsmässig deutlich überlegen. Glücklicherweise gibt es heutzutage auch recht leistungsfähige *Derivative Free Optimization* (DFO) Software, also Löser, die ohne die Ableitungen auskommen. Um nicht unnötig viel Zeit zum Lösen der Programmierübungen zu verwenden, empfehlen wir für die Lösung von NLP (z.B. für das folgende Training der neuronalen Netze) DFO Löser. Ein gutes und freies Paket ist NLopt, das 4 DFO Löser beinhaltet. Von diesen ist PRAXIS eine CG (Conjugate Gradients) Variante und gut für das Training von Neuronalen Netzen geeignet, dasselbe gilt für BOBYQA als quadratisch approximierenden Löser. COBYLA approximiert linear und ist daher weniger gut geeignet, und NELDERMEAD ist ein älterer und deutlich weniger leistungsfähiger Algorithmus.

2.3 Training Neuronaler Netze

Wir erinnern uns: ein neuronales Netz ist eine nichtlineare Funktion $f : \mathcal{X} \times \Theta \to \mathcal{Y}$, die in Abhängigkeit von einem Eingabedatum x und einem Parametervektor θ einen Ausgabewert y berechnet. Für gegebene Trainingsdaten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, unser Ziel wird es sein, den Parameter θ so zu bestimmen, dass das Netz möglichst gut für die Trainingsdaten passt.

2.3.1 Regression mit neuronalen Netzen. Im Falle einer Regression legen wir wie zuvor die Normalverteilungsannahme zugrunde. Wie vorher ist der ML-Schätzer $\hat{\theta}$ gerade dasjenige $\theta \in \Theta$, dass den quadratischen Fehler minimiert. Wir erhalten also die *Fehlerfunktion*

$$f^{\text{sse}}(\theta) = \sum_{i=1}^{n} \big(y_i - f(x_i, \theta)\big)^2, \tag{42}$$

wobei f für unsere Beispiele in der Form (20) vorliegt. Diese Fehlerfunktion ist zweimal stetig differenzierbar (sogar beliebig oft stetig differenzierbar)

und erfüllt damit die Voraussetzungen in 2.2.1 an ein NLP. Wir können sie also mit einem gebräuchlichen Löser optimieren, etwa einem ableitungsbasierten Löser wie IpOpt oder einem DFO-Löser wie PRAXIS. Der resultierende optimierte Parametervektor $\hat{\theta}$ ist ein *lokaler ML-Schätzer*. Die Fehlerfunktion kann multimodal sein, eine Garantie, dass wir das globale Optimum gefunden haben, gibt es nicht.

2.3.2 Bemerkung. *Es hat sich in der Praxis bewährt, vor dem Training eines neuronalen Netzes alle Komponenten Trainingsdaten auf Mittelwert null und Varianz 1 zu verschieben / reskalieren. In vielen Fällen ist das Training dann numerisch robuster und konvergiert schneller.*

In der Literatur der neuronalen Netze wird das Training herkömmlicherweise als integraler Bestandteil des Netzes gesehen. Der Standpunkt, dass irgendein Optimierungsalgorithmus diese Aufgabe erledigen kann, ist nicht üblich. Insbesondere wurden in der Geschichte der neuronalen Netze viele spezialisierte Trainingsalgorithmen entwickelt, die jedoch nicht dieselbe Leistungsfähigkeit aufweisen wie die aus dem Forschungsgebiet der Optimierung resultierenden Algorithmen. Der Author dieses Skriptes ist der Meinung, dass es wesentlich einfacher und effizienter ist, einen modernen Optimierungsalgorithmus auf die Fehlerfunktion anzuwenden als einen historischen und auf neuronale Netze spezialisierten Algorithmus wie z.B. "Backpropagation".

Wie schon erwähnt, empfehlen wir für die Übungen insbesondere die Algorithmen PRAXIS und BOBYQA aus der NLopt Toolbox, die als DFO-Löser einfach zu verwenden und gleichzeitig leistungsfähig sind.

2.3.3 Klassifikation mit neuronalen Netzen. Auch im Fall einer Klassifikation können wir die Fehlerfunktion

$$f^{\text{sse}}(\theta) = \sum_{i=1}^{n} \big(y_i - f(x_i, \theta)\big)^2, \tag{43}$$

minimieren. Hier ist f für in der Form (21) gegeben. Dieser Ansatz führt zu praktisch guten Resultaten und wird daher für die weiteren Übungen empfohlen. Wir weisen kurz darauf hin, dass die probabilistische Analyse für die Klassifikation eigentlich eine andere ist: Für ein Trainingspaar (x_i, y_i) ist $y_i \in \{-1, 1\}$ (oder, wie bereits erwähnt, oft günstiger $y_i \in \{-0.9, 0.9\}$), während die Netzausgabe $f(x_i)$ mit f in der Form (21) die *Wahrscheinlichkeit* angibt, dass das Label an dieser Stelle x_i positiv ist. Um komplizierte Reskalierung in der folgenden Notation zu vermeiden, nehmen wir kurz an, dass $y_i \in \{0, 1\}$ und die Aktivierungsfunktion logistisch ist, d.h. gegen 0

konvergiert wenn das Argument gegen $-\infty$ geht. Dann ist die Wahrscheinlichkeit für (x_i, y_i) unter dem Parameter θ gleich der Bernoulli-Verteilung

$$\big(1 - f(x_i, \theta)\big)^{1-y_i} f(x_i, \theta)^{y_i}. \tag{44}$$

Wollen wir unter dieser Annahme an die Verteilung einen ML-Schätzer berechnen, so führt dies zu der Fehlerfunktion der *Kreuzentropie*. Interessierte Leser finden Details im Buch [Bishop: Neural Networks for Pattern Recognition].

2.4 Overfitting und Regularisierung

Im Gegensatz zu linearer Regression sind nichtlineare Regressionsmodelle sehr anfällig gegen *Overfitting*. Als Overfitting wird die Situation bezeichnet, in der das Training einen Schätzer liefert, der sehr gut an die Trainingsdaten passt, aber sehr schlecht generalisiert. In der Vorlesung finden wir einige Beispiele hierfür.

Es gibt verschiedene Strategien gegen Overfitting:

- Unproblematisch ist die Situation, wenn die Anzahl der Freiheitsgrade im Regressionsmodell (d.h. die Anzahl der Elemente in θ) deutlich kleiner ist als die Anzahl der Messdaten. Dann hat das Modell nicht hinreichend Freiheitsgrade, um sich dem Messfehler anzupassen.

- Oft teilt man die verfügbaren Daten in eine *Trainingsmenge* und eine *Testmenge* auf. Während lediglich die Trainingsmenge für das Training verwendet wird, wird das Modell hinterher mit der Testmenge verifiziert. Stellt sich hierbei ein Overfitting heraus (der Testfehler ist signifikant grösser als der Trainingsfehler), so wird das trainierte Modell verworfen. Der Nachteil ist, dass die evtl. bereits knappen Trainingsdaten durch Abspalten der Testmenge weiter reduziert werden.

- Ebenfalls gebräuchlich für das Training neuronaler Netze ist es, eine weitere Menge aus den Trainingsdaten abzuspalten: die *Validierungsmenge*. Auf dieser Menge wird der Fehler (Validierungsfehler) *während* des Trainings ausgewertet. Das Training wird abgebrochen (*"Early Stopping"*), wenn der Validierungsfehler signifikant ansteigt. Auch diese Methode hat denselben Nachteil der Verkleinerung der Trainingsmenge.

- Hier wollen wir etwas genauer die *Regularisierung* betrachten. Das ist eine Anwendung von Ockhams Sparsamkeitsprinzip 1.5.1 und Bayes Regel 1.5.3 auf das Training neuronaler Netze.

Theoretische Überlegungen sowie praktische Studien zeigen, dass die Ausgabe neuronaler Netze relativ glatt ist, solange die Trainingsgewichte klein sind. Im Fall von Overfitting dagegen neigen die Netze dazu, relativ "eckige" Ausgabegraphen zu erzeugen, und die Gewichte (Einträge von θ), die aus dem Training resultieren, sind teilweise gross oder sehr gross. Aus dieser Beobachtung heraus *regularisieren* wir das Training, indem wir grosse Gewichte in der Fehlerfunktion bestrafen. Dies passt in das Bayessche Framework 1.5.3: Wir führen einen *Gauss'schen Prior* auf den Gewichten ein. Der Posterior ist dann das Produkt aus Prior und Likelihood, und nach Logarithmieren und Negieren führt das zu einer Summe aus dem quadratischen Fehlerterm und einen quadratischen Strafterm für die Gewichte:

2.4.1 Training mit Weight Decay. Minimierung des Kriteriums

$$f^{\text{sse+weight decay}}(\theta) = \sum_{i=1}^{n} \big(y_i - f(x_i, \theta)\big)^2 + \alpha \sum_j \theta_j^2 \tag{45}$$

führt zu einem Schätzer $\hat{\theta}$, der einem MAP-Schätzer unter einem normalverteilten Prior entspricht. Der Wert von $\alpha > 0$, dem *Weight-Decay Parameter*, bestimmt die Stärke der Regularisierung. In vielen praktischen Anwendungen werden Werte von α im Bereich 10^{-2} oder 10^{-3} erfolgreich eingesetzt.

3 Support Vektor Maschinen

3.1 Optimale Lineare Klassifizierung

Betrachten wir die Situation, dass wir ein binäres Klassifizierungsproblem mit Trainingsdaten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, $\mathcal{X} \subset \mathbb{R}^d$ und $\mathcal{Y} = \{-1, 1\}$ lösen wollen, indem wir die positiven und die positiven und die negativen Instanzen *linear* in $\mathcal{X}$ separieren. Im Kapitel über neuronale Netze haben wir eine Methode kennengelernt, die hierfür geeignet ist. Beispielsweise können wir die Netzarchitektur (21) verwenden und ein entsprechendes Netz trainieren. Das ergibt allerdings eine nichtlineare Klassifikation. Wir könnten (21) dahingehend abändern, dass wir die inneren tanh-Aktivierungsfunktionen durch lineare Funktionen ersetzen. Stellen wir uns allerdings die Frage, ob dies der *beste* Weg ist, das Problem zu lösen, so lautet die Antwort aus mehreren Gründen nein. Beispielsweise hat die Fehlerfunktion im Allgemeinen mehrere lokale Optima, und wir werden feststellen, dass das Training oft ein unbefriedigendes Ergebnis liefert. Hier wollen wir einen geometrische Ansatz etablieren, der zu einer *optimalen* linearen Klassifikation führt.

3.1.1 Definition (Hyperebene). Eine *Hpyerebene* H in $\mathbb{R}^d$ ist eine $(d-1)$-dimensionale affine Mannigfaltigkeit. Sie kann beschrieben werden

durch einen *Normalenvektor* $v \in \mathbb{R}^d \setminus \{0\}$ und einen *Offset* $b \in \mathbb{R}$:

$$H = \{x \in \mathbb{R}^d : v^T x + b = 0\} \tag{46}$$

Wir schreiben hierfür auch kurz $H = H(v, b)$.

3.1.2 Maximum Margin Classifier. Gegeben seinen Trainingsdaten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$. Falls die Daten *linear separierbar* sind, was bedeutet, dass es eine Hyperebene $H = H(v, b)$ gibt, so dass

$$v^T x_i + b \geq 0 \iff y_i = 1 \tag{47}$$

gilt, so ist der *Maximum Margin Classifier* diejenige Hyperebene, für die (47) gilt und

$$\min_{1 \leq i \leq n} \|v^T x_i + b\| \tag{48}$$

maximal wird. Dies ist, unter allen Hyperebenen, die

$$y_i(v^T x_i + b) \geq 1 \text{ für alle } \; 1 \leq i \leq n \tag{49}$$

erfüllen, diejenige mit minimalen v

$$\min \|v\|. \tag{50}$$

Die verwendete Norm $\|\cdot\|$ ist üblicherweise die Euklidische Norm, $\|x\| = \sqrt{x^T x}$.

(49) und (50) erlauben es sofort, die Suche nach dem Maximum Margin Classifier als *quadratisches Programm* (s. 2.2.3) zu schreiben:

$$\min \sum_{i=1}^{d} v_i^2 = \|v\|^2 \tag{51}$$

$$\text{wobei } y_i(v^T x_i + b) \geq 1 \text{ für alle } \; 1 \leq i \leq n. \tag{52}$$

(52) definiert hier genauso viele Nebenbedingungen, wie Trainingsdatenpunkte vorhanden sind.

3.2 Der Nicht Linear Separierbare Fall

Das Optimierungsproblem, das den Maximum Margin Classifier 3.1.2 berechnet, kann nur erfolgreich gelöst werden, wenn die Daten in der Tat linear separierbar sind. Falls dies nicht der Fall ist, so liefert die Optimierung das Ergebnis “infeasible”.

Um auch in diesen Fällen einen linearen Separator zu erhalten, modifizieren wir den Ansatz leicht: Wir erlauben jeden Datenpunkt, auf der falschen Seite

der klassifizierenden Hyperebene zu liegen, *bestrafen* dies jedoch mit einem zusätzlichen Kostenterm. Wir ersetzen also die Nebenbediungen (49) bzw. (52) durch

$$y_i(v^T x_i + b) \geq 1 - \xi_i \text{ für alle } \ 1 \leq i \leq n \tag{53}$$

und die Kostenfunktion (51) durch

$$\|v\|^2 + C \sum_{i=1}^{n} \xi_i. \tag{54}$$

Die Variablen $\xi_i \geq 0$ heissen *Schlupfvariablen* und dienen dazu, die Nebenbedingungen bei Bedarf zu relaxieren. Auf diese Weise erreichen wir, dass der Löser für jeden Datensatz ein Ergebnis findet. Der Preis, den wir dafür bezahlen müssen, ist die Tatsache, dass wir einen zusätzlichen Parameter $C > 0$ einstellen müssen, der angibt, wie stark Fehlklassifizierungen in den Trainingsdaten bestraft werden sollen.

3.3 Duales Problem und Kerne

(53) und (54) definieren ein Optimierungsproblem, das sogenannte *primale Problem*. Es handelt sich hierbei um ein konvexes quadratisches Optimierungsproblem. Insbesondere kann das Problem effizient gelöst werden, und jedes lokale Optimum ist auch ein globales Optimum. Die Theorie der konvexen Optimierung stellt ein weiteres, sehr nützliches Werkzeug zur Verfügung: Das primale Problem kann umformuliert werden in ein sogenanntes *duales Problem*. Ein duales Problem zu unserem primalen Problem ist das folgende:

$$\max \sum_{i=1}^{n} \left(\alpha_i - \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \right) \tag{55}$$

$$\text{wobei } 0 \leq \alpha_i \leq C \text{ für alle } 1 \leq i \leq n \tag{56}$$

$$\text{und } \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{57}$$

Die Lösung dieses Problems, welches wieder ein QP ist, ist äquivalent zur Lösung des primalen Problems. Wir bemerken einige Unterschiede zum primalen QP: Erstens hat (55)-(57) eine Gleichheitsnebenbediungung. Zweitens treten die Eingangsvektoren x_i *nur innerhalb von Skalarprodukten* auf. Die letztere Eigenschaft wird sich, in Zusammenhang mit dem folgenden Satz, als sehr nützlich erweisen.

3.3.1 Definition (positiv definiter Kern). Eine Abbildung $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ heisst *positiv definiter Kern*, falls für beliebige Eingangsdatenpunkte

$x_1, x_2, \ldots, x_n \in \mathcal{X}$, die *Gram-Matrix*

$$\begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \ldots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \ldots & k(x_2,x_n) \\ \vdots & & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \ldots & k(x_n,x_n) \end{pmatrix} \tag{58}$$

positiv definit ist. (Der Begriff "positiv definit" wurde in dem Abschnitt nach Definition 2.2.7 eingeführt, er besagt, dass alle Eigenwerte der Matrix nichtnegativ sind.)

3.3.2 Satz (Mercers Satz). *Sei $\mathcal{X}$ der Raum der Eingangsdaten, und sei $\mathcal{F}$ ein beliebiger Vektorraum (also z.B. ein $\mathbb{R}^n$, oder ein unendlich dimensionaler Raum), der* Featureraum. *Sei $\phi : \mathcal{X} \to \mathcal{F}$ eine Abbildung. Dann definiert ϕ einen positiv definiten Kern auf $\mathcal{X}$ via*

$$k(x_1, x_2) = \phi(x_1)^T \phi(x_2). \tag{59}$$

Die Umkehrung gilt auch: Für jeden positiv definiten Kern $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ lässt sich ein Featureraum $\mathcal{F}$ und eine Abbildung $\phi : \mathcal{X} \to \mathcal{F}$ finden, so dass (59) erfüllt ist.

Es ist der zweite Teil von Mercers Satz, der nichttrivial ist. Und mit ihm können wir die *Kern-Variante* einer SVM formulieren, die die gebräuchlichste ist:

3.3.3 Support Vektor Maschine mit Kern. Gegeben seinen Trainingsdaten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$. Wir wählen einen positiv definiten Kern $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Dann liefert die Lösung des folgenden QP

$$\max \sum_{i=1}^{n} \left(\alpha_i - \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \right) \tag{60}$$

$$\text{wobei } 0 \leq \alpha_i \leq C \text{ für alle } 1 \leq i \leq n \tag{61}$$

$$\text{und } \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{62}$$

eine trainierte SVM mit dem gewählten Kern k. Von Mercers Satz wissen wir, dass diese nichtlineare Klassifikation einer linearen Klassifikation in einem Featureraum entspricht. Wir müssen aber diesen Featureraum niemals explizit kennen, und wir müssen dort keine Berechnungen vornehmen. Alle Berechnungen passieren via Auswertung der Kernfunktion.

Jetzt können wir auch den Begriff *Support Vektor Maschine* erklären. Nach dem Training der SVM werden typischerweise viele der α_i gleich null sein. Nur diejenigen α_i sind positiv, für die die zugehörigen x_i entweder den minimalen Abstand (im Sinne des primalen Problems) von der separierenden Hyperebene haben oder sogar einen kleineren Abstand (letztere sind insbesondere die Trainingsdaten, die von der SVM falsch klassifiziert werden). Diese x_i, für die die α_i positiv sind, heissen *Support Vektoren*.

3.3.4 Beispiele von Kernen. Einige gebräuchliche Kerne werden im Folgenden aufgelistet:
(a) Linear: $k(x_1, x_2) = x_1^T x_2$. Hier ist der Featureraum der Ursprungsraum.
(b) Polynomiell: $k(x_1, x_2) = (x_1^T x_2)^n$. Hier ist der Featureraum höher dimensional als der Ursprungsraum für $n > 1$. Beispielsweise ist für $d = 2$ und $n = 2$ der Featureraum dreidimensional.
(c) Gausskern: $k(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$. Hier ist der Featureraum unendlich dimensional.
(d) Sigmoid: $k(x_1, x_2) = \tanh(\gamma x_1^T x_2 + \beta)$. Hier ist der Featureraum unendlich dimensional.

3.4 Auswertung einer SVM

Wir wird eine trainierte SVM ausgewertet? Die Antwort ist prinzipiell einfach:

$$y = \text{sign}(w^T x + b) \tag{63}$$

liefert die gewünschte Vorhersage für $x \in \mathcal{X}$, vorausgesetzt dass w (im Featureraum) und b bekannt sind. Diese Formel kann verwendet werden, wenn die SVM als primales Problem trainiert wurde. (Hier ist sign die Vorzeichenfunktion, d.h. sign einer negativen Zahl ist -1 und sign einer positiven Zahl ist +1.)

Wenn die SVM als duales Problem trainiert wurde, so kann sie mittels

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b\right) \tag{64}$$

ausgewertet werden. Die Summe darf dabei über diejenigen i laufen, für die $\alpha_i > 0$ ist, also über die *Supportvektoren*. Um die Auswertung vornehmen zu können, müssen wir nach dem Training der SVM b berechnen. Nach dem Training haben wir, dass

$$\max_{1 \le j \le n:\ \alpha_j > 0 \text{ und } y_j = 1} \left(\sum_{i=1}^{n} \alpha_i y_i k(x_i, x_j) + b\right) = 1, \tag{65}$$

denn dieses Maximum wird genau auf den Supportvektoren angenommen, die korrekt positiv klassifiziert wurden. Daraus folgt

$$b = 1 - \left(\max_{1 \le j \le n:\ \alpha_j > 0 \text{ und } y_j = 1} \sum_{i=1}^{n} \alpha_i y_i k(x_i, x_j) \right). \tag{66}$$

4 Clustering

Während wir bislang ausschliesslich *überwachtes Lernen* behandelt haben, wenden wir uns im letzten Teil des Skriptes zu den Grundlagen des *unüberwachten Lernens* oder Clusterings zu. Das grundlegende Lernproblem ist hier das folgende: Gegeben sind Datenpunkte $x_1, x_2, \ldots, x_n \in \mathcal{X}$, diesmal ohne Labels. Gesucht ist nach einer "natürlichen" Gruppierung / Einteilung dieser Datenpunkte in Klassen $C_1, C_2, \ldots, C_k$, $\dot{\bigcup}_{j=1}^{k} C_j = \{x_i : 1 \le i \le n\}$ (wobei $\dot{\bigcup}$ die *diskjunkte Vereinigung*[4] bezeichnet, d.h. die Vereinigung der Mengen $C_1, C_2, \ldots, C_k$ wobei gleichzeitig der paarweise Durchschnitt leer sein muss).

Wie soll eine automatische Einteilung in Cluster stattfinden? Dies ist im Allgemeinen *problemspezifisch*: Es hängt von der jeweiligen Aufgabenstellung ab, welche Datenpunkte wir als zusammengehörig betrachten sollten und welche als verschieden. Wir nehmen im Folgenden wiederum an, dass der Eingaberaum eine Untermenge des Euklidischen Raums ist, $\mathcal{X} \subset \mathbb{R}^d$. Ferner nehmen wir an, dass Punkte, die im Euklidischen Raum dicht beieinander liegen, (wahrscheinlich) zusammengehören.

4.1 Clustering mit Graph-Cuts

Ein allgemeiner Ansatz für das Clustering ist es, die gegebenen Datenpunkte in einen Graph einzubetten und das Clustering per Optimierung eines Schnitt-Kriteriums zu erhalten.

4.1.1 Definition (Graph). Ein *ungerichteter, gewichteter Graph* $G = G(V, E)$ ist definiert durch eine Menge von Knoten V ("vertices") und eine Menge von Kanten $E \subset \big\{\{v_1, v_2\} : v_1, v_2 \in V, v_1 \neq v_2\big\}$ ("edges"). Für jede Kante $e \in E$ existiert ein Gewicht $w_e = w(e) \in \mathbb{R}$. Im Folgenden ist immer $w_e \ge 0$.

Einem Datensatz $x_1, x_2, \ldots, x_n \in \mathcal{X}$ können wir sofort zwei Typen von Graph zuordnen: Eines ist der *Distanzgraph*, bei dem die Knoten gleich den Datenpunkten sind und zwischen je zwei Datenpunkten x_i, x_j eine Kante

[4] Es gibt auch Ansätze, bei denen die Cluster sich überlappen dürfen.

des Wertes $w(e_ij) = \|x_i - x_j\|$ definiert wird. Auf der anderen Seite wird ein *Ähnlichkeitsgraph* bestimmt durch wiederum die Datenpunkte als Knotenmenge und Kanten zwischen je zwei Datenpunkten, die nun als Wert ein *Ähnlichkeitsmass* haben. Mögliche Ähnlichkeitsmasse sind beispielsweise das Skalarprodukt $w(e_ij) = x_i^T x_j$, oder der Gausskern $w(e_ij) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$, mit einer festzulegenden Breite σ.

4.1.2 Definition (Schnitt). Ein (binärer) Schnitt in einem Graphen $G(V, E)$ ist eine Teilmenge der Knoten $V_1 \subset V$. Der *Wert* des Schnittes ist gegeben durch die Kanten, die zwischen V_1 und ihrem Komplement verlaufen:

$$w(V_1) = \sum \big\{ w_e : e \in E, \exists v_1 \in V_1 \cap e \text{ und } \exists v_2 \in (V \setminus V_1) \cap e \big\}. \tag{67}$$

Ein *Max-Cut* ist ein Schnitt in einem Graphen der einen maximalen Wert unter allen Schnitten hat. Ein *Min-Cut* ist ein Schnitt mit einem minimalen Wert.

Es ist nun intuitiv klar, dass ein Max-Cut auf einem Distanzgraph und ein Min-Cut auf einem Ähnlichkeitsgraph jeweils ein binäres Clustering (als ein Clustering mit $k = 2$ Klassen) produziert. (Clusterings mit $k > 2$ Klassen werden wir im nächsten Abschnitt betrachten.)

In der Praxis stellt sich allerdings oft heraus, dass ein Min-Cut Clustering nicht robust gegen Ausreisser und daher ungeeignet ist. Während Min-Cut effizient (in polynomieller Zeit) berechnet werden kann, ist die Suche nach einem Max-Cut NP-schwer. Wir stellen hier eine Formulierung der Max-Cut Aufgabe als MILP (s. 2.2.4) vor.

4.1.3 Max-Cut als MILP. Ein MILP (2.2.4) wird definiert durch eine Menge Variablen, der linearen Kostenfunktion c, den Nebenbedingungen, und der Spezifikation der Variablentypen. Wir benötigen hier lediglich Ungleichheitsnebenbedingungen, müssen also die Matrizen A und b definieren. Um Verwechslungen in der Notation zu vermeiden, bezeichnen wir hier die Variablen mit $u_1, \ldots, u_\lambda$. Hierbei ist die Variable u_i für $1 \leq i \leq n$ ein Indikator, ob der Punkt x_i im Schnitt V_1 vorkommen soll. Für $n+1 \leq i \leq n+|E|$ ist u_i ein Indikator dafür, ob die Kante e_{i-n} im Schnitt verwendet werden soll. Die Variablen sind also alle binär, und es gibt $\lambda = n + |E| = n + \frac{n(n-1)}{2}$ viele. Die Kosten sind $c_i = 0$ für $1 \leq i \leq n$ und $c_i = -w(e_{i-n})$ (wir minimieren den negativen Wert $\Rightarrow$ maximieren den Wert des Schnittes) für $n+1 \leq i \leq n+|E|$. Schliesslich müssen wir noch dafür sorgen, dass eine Kante nur angewählt werden kann, wenn von den angrenzenden Knoten einer in V_1 ist und einer nicht. Das geht per Constraints: Wir fordern, für alle $1 \leq i \leq |E|$

$$u_{n+i} \leq u_{v_1(i)} + u_{v_1(i)}, \tag{68}$$

wobei v_1 und v_2 Funktionen sind, die jedem Index einer Kante den Index des jeweils ersten und zweiten angrenzenden Knotens (Reihenfolge egal) zuordnen. Auf diese Weise verhindern wir, dass eine Kante angewählt wird, obwohl beide angrenzenden Knoten nicht in V_1 enthalten sind. Genauso fordern wir

$$u_{n+i} \leq 2 - u_{v_1(i)} - u_{v_1(i)} \tag{69}$$

für alle $1 \leq i \leq |E|$, was sicherstellt, dass die Kante nicht angewählt werden kann, falls beide angrenzenden Knoten in V_1 enthalten sind. Die beiden Ungleichungen (68) und (69) definieren, über alle $1 \leq i \leq |E|$, die Matrix A und den Vektor b.

Versucht man allerdings, dieses MILP für das Clustering eines Datensatzes zu verwenden, so stellt man fest, dass für bereits weniger als 50 Datenpunkte die benötigte Rechenzeit stark ansteigt. Da die Zeit, die eine MILP für die Lösung benötigt, üblicherweise exponentiell wächst, besteht wenig Hoffnung, praktische Clusteringprobleme mit tausenden oder hunderttausenden Datenpunkten auf diese Weise zu lösen.

Für Max-Cut sind Relaxationen bekannt, z.B. eine, die auf SDP (Semidefiniter Programmierung) beruht. Wir gehen hier nicht weiter auf diese und andere Graph-basierte Ansätze ein. Auch die Verallgemeinerung auf $k > 2$ Cluster, die mittels Max-k-Cut möglich ist, soll hier nicht behandelt werden.

4.2 k-Means

Stattdessen wenden wir uns einem anderen, viel effizienteren Ansatz zum Clustering zu, der für eine beliebige Anzahl von Clustern k verwendet werden kann: Der *k-Means* Algorithmus.

4.2.1 k-Means Algorithmus. Gegeben Datenpunkte $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, Anzahl Cluster $k \geq 2$. Ausgabe des Algorithmus: Clusterzuordnungen $c_i \in \{1, \ldots, k\}$, $1 \leq i \leq n$.

1 Initialisiere Zentren $x_1, \ldots z_k \in \mathbb{R}^d$ zufällig
2 Wiederhole bis Konvergenz
3 Für jedes $i \in \{1, \ldots, n\}$, setze $c_i = \arg\min_{1 \leq j \leq k} \|x_i - z_j\|$
4 Für jedes $j \in \{1, \ldots, k\}$, setze $z_j = \frac{1}{|\{1 \leq i \leq n : c_i = j\}|} \sum_{1 \leq i \leq n : c_i = j} x_i$

Man kann zeigen, dass die Schleife in diesem Algorithmus immer nach endlich vielen Schritten terminiert, d.h. der Algorithmus konvergiert immer.

Ein mathematisches Modell, das mit dem k-Means Algorithmus im Beziehung steht, ist das *Mixture of Gaussians*:

4.2.2 Definition (Mixture of Gaussians). Es seien Zentren $z_1, \ldots, z_k \in \mathbb{R}^d$, ein Vektor von Mischungskoeffizienten $p \in [0,1]^k$ mit $\sum_j p_j = 1$ und eine Weite $\sigma > 0$ gegeben. Dann ist

$$p(x) = \sum_{j=1}^{k} p_j p(x|z_j) = \frac{1}{\left(2\pi\sigma^2\right)^{\frac{d}{2}}} \sum_{j=1}^{k} p_j e^{-\frac{\|x-z_j\|^2}{2\sigma^2}} \tag{70}$$

ein Mixture of Gaussians.

Man kann leicht die folgenden beiden Aussagen zeigen:

4.2.3 Satz. *Unter der Annahme, dass die Datenpunkte gemäss einem Mixture of Gaussians mit fester Weite σ, uniformen Mischungskoeffizienten und Zentren bei den momentanen Vektoren z_j verteilt sind, entspricht die Zuordnung der Punkte zu den Clustern im Schritt `3` des Algorithmus 4.2.1 einer Maximum Likelihood Zuordnung.*

Beweis. Die Wahrscheinlichkeitsdichte, die zu dem Vektor z_j gehört, welcher x_i am nächsten ist, ist grösser als die Wahrscheinlichkeitsdichte, die zu irgendeinem anderen $z_{j'}$ gehört. Dies gilt für alle Datenpunkte x_i. □

4.2.4 Satz. *Unter der Normalverteilungsannahme sind die Neuzuordnungen der Zentren z_j in Schritt `4` des Algorithmus 4.2.1 ML-Schätzungen für die Zentren der jeweiligen Normalverteilungen.*

Beweis. Dies folgt sofort aus einer d-dimensionalen Variante von Satz 1.2.4. □

Die beiden vorigen Aussagen zeigen, dass der k-Means Algorithmus ein Spezialfall des *EM-Algorithmus* ist. Der EM-Algorithmus führt, auf einer zugrunde liegenden Mischung von Verteilungen, abwechselnd einen *Estimate*-Schritt (E-Schritt) und einen *Maximize*-Schritt (M-Schritt) aus. Der E-Schritt besteht darin, dass die Schätzung des Modells aufdatiert wird, also die Parameter für die Mischung der Verteilungen neu bestimmt werden. Der M-Schritt besteht darin, dass auf Grund des aktuellen Modells die Datenpunkte anhand eines ML-Kriteriums den einzelnen Komponenten des Modells zugeordnet werden.

4.3 Bestimmung der Anzahl der Cluster

Am besten funktionieren Clusteringalgorithmen, wenn die Anzahl der Cluster k a-priori bekannt ist. Ist das nicht der Fall, so kann man das Clustering

für mehrere k ausprobieren. Dann benötigt man ein Kriterium, zu entscheiden, welches k zu den besten Ergebnissen geführt hat. Am besten geeignet ist immer ein problemspezifisches Kriterium für die jeweilige Anwendung. Steht ein solches nicht zur Verfügung, gibt es geometrische Kriterien, die die Intra-Cluster-Varianz und die Varianz zwischen den Clustern in Beziehung setzen. Eine andere Möglichkeit ist ein wahrscheinlichkeitstheoretisches Kriterium, das auf dem Mixture of Gaussians Modell beruht.

Eine weitere Möglichkeit ist die Verwendung eines Spectral Clustering und die Analyse der Gaps in den Eigenwerten. Auf alle diese Verfahren soll hier nicht weiter eingegangen werden.