Einführung in die Mustererkennung

Zusammenfassung des Stoffes
Michael Ari und Moritz Resl
März 2012

1. Einführung und Bildverarbeitung

Was ist Mustererkennung?

Aufgabe: Klassifikation von Mustern (patterns) anhand von Merkmalen (features)
Muster werden durch Merkmale beschrieben:
konkreter Wert eines Merkmals: Merkmalsausprägung
Merkmale werden häufig in Merkmalsvektoren (feature vectors) zusammengefasst

Trainingsset: zum Trainieren der Generalisierungsfähigkeit des Systems
Güte eines Mekrmals hängt davon ab
- wie schnell/einfach es berechnet werden kann
- wie diskriminativ es ist (von anderen unterscheidbar ist)

Merkmale können zB. mit Histogramm visualisiert werden.
Bessere Klassifizierung bei Kombination von mehreren Merkmalen.
Klassifikator: trennt unterschiedliche Einheiten
zu einfacher Klassifikator: schlechte Ergebnisse (underfitting)
zu komplexer K.: zu sensitiv (overfitting)

Entwicklungszyklus: Bayes-Theorem
start ▸ collect data ▸ "chose features ▸ choose model" (prior knowledge) ▸ train classifier ▸ evaluate classifier ▸ end

Anwendungen der Mustererkennung

Personenerkennung (Biometrie), Inhaltsbasierte Bildsuche, Buchstabenerkennung, Robotik/ Bewegung, industrielle Inspektion, Fernerkennung (Satellit- & Luftbilder), Biomedizinische Anwendungen

Elementare Grundlagen der Bildverarbeitung

Exkurs: Digitale Bilder
2-dimensionales Bild ist eine Funktion, die eine Helligkeit mit jeder Position (x,y) verbindet.
Jedes Element der x,y-Matrix heißt Pixel.
analog ▸ digital: sampling (Abtasten), quantisation (Wertzuweisung)
Operatoren für Bilder: Histogramm, Faltung (Mittelwert Filer, Kantenerkennung), Schwellwert-Operator,...

2. Merkmalsextraktion 1

Statistische und Syntaktische Mustererkennung

Überwachtes Verfahren (zB. k-NN, CART: Classification and Regression Trees): Beim Trainieren ist jede Klasse eines Datensatzes bekannt
Unüberwachtes Verfahren: Beim Trainieren ist nicht bekannt, um welche Klasse es sich bei einem Datensatz handelt

Statistische Mustererkennung (am häufigsten verwendet)
- Objekte werden durch Merkmalsvektoren beschrieben
- statistische Methoden werden verwendet, die Vektoren zu klassifizieren

Syntaktische Mustererkennung
Objekte werden durch Primitive und den Verhältnissen zwischen diesen beschreiben
Die Gruppe von Regeln, wie man Objekte aus Primitiven erstellen kann, nennt man Gramatik (für jede Klasse unterschiedlich).

Merkmalsextraktion

Oft zu viele Merkmale, daher Reduktion der Datenmenge auf Merkmale, die gut unterscheidbar sind. Konzentration auf die für Klassifikation wichtige Information.

Beispiele aus der Biometrie

Beispiel Objekterkennung: Schwellwert, Labelling, Komponente (4-fach, 8-fach Konnektivität), feature extraction, Zusammenfassung der Merkmale in Vektoren

Bei 8-fach Konnektivität für Vordergrund, muss 4-fach Konnektivität für Hintergrund verwendet werden!

Arten von Merkmalen: (Merkmale sollen sein:)
positioninvariante
orientierungsinvariante
skalierungsinvariante

Regionsmerkmale: charakterisieren räumliche Verteilung der Pixel in einem Blob
Bilder vorsichtig drehen: Detailverlust möglich!
Schwerpunkt, Umfang, Euler Zahl, Ellipse Fitting
Konvexe Hülle: Ein Blob X ist konvex, wenn jede Gerade zwischen 2 beliebigen Punkten in X vollständig in X ist.

Die Anzahl der Merkmale soll reduziert werden, ohne wichtige Informationen zu verwerfen. (feature selection)

Alle Merkmale eines Objekts werden in einem Merkmalsvektor zusammengefasst. Für bestimmte Anwendungen sind gute Merkmale schwierig zu finden.

Minutiae matching (Fingerabdruckmessung) am häufigsten verwendet. Minutiae sind die kleinen Details eines Fingerabdrucks. (Die häufigsten: *ridge ending* und *bifurcation* (Gabelung)).

3. k-NN und Statistische Grundlagen

K-NN Algorithmus

Bei einer Testmenge mit gegebenem Punkt: die k umliegenden Punkte werden betrachtet (k üblicherweise ungerade) und die Testmenge (▸ Farbe), die am häufigsten Vorkommt bestimmt den Zugehörigkeit von x'. Trainigsset ist bereits vollständig klassifiziert.

Voronoi

Jeder Punkt in dem Polygon ist genau dem Punkt am nächsten, der vom Polygon umgeben wird. (siehe asugedrucktes Skriptum für Skizze!)

Eigenschaften von k-NN:
Benötigt kein eigentliches Training, sondern speichert gesamtes Trainingsset ab.
Sowohl Speicher als auch Laufzeit wachsen linear mit der Größe des Trainingssets. [O(n)]
Euklidische Distanz wird am häufigsten verwendet, aber andere Metrik auch anwendbar.
Ist ein nicht parametrisches Verfahren!

Statistische Grundlagen (Diskrete Zufallsvariablen)

P(A): Wahrscheinlichkeit für Eintreten von Ereignis A
Zufallsvariable heißt diskret, wenn endlich viele Möglichkeiten möglich sind.

Erwartung / Mittelwert

$$\varepsilon[x] = \mu = \sum_{x \in X} xP(x) = \sum_{i=1}^{m} v_i p_i$$

Mehr allgemein, sei f(x) eine beliebige Funktion von x. Die Erwartung von f(x) ist

$$\varepsilon[f(x)] = \sum_{x \in X} f(x)P(x)$$

Varianz

$$\mathrm{var}[x] = \sigma^2 = \varepsilon[(x-\mu)^2] = \sum_{x \in X} (x-\mu)^2 P(x)$$

Standardabweichung σ

von x und bezeichnet wie weit die x-Werte wahrscheinlich von Mittelwert liegen. Nützlich ist die folgende Gleichung:

$$\mathrm{var}[x] = \sigma^2 = \varepsilon[x^2] - (\varepsilon[x])^2$$

Paare von Diskreten Zufallsvariablen

Seien x und y Zufallsvariablen. Die Wahrscheinlichkeit, dass jedes Wertepaar (für alle Möglichkeiten von x und von y) eintritt, ist durch die joint probability (Verbundwahrscheinlichkeit) gegeben:

$$p_{ij} = P(x = v_i, y = w_j)$$

So wie bei Einzelwahrscheinlichkeiten gelten wieder zwei Voraussetzungen:

$$P(x,y) \geq 0$$

und

$$\sum_{x \in X} \sum_{y \in Y} P(x,y) = 1$$

Randverteilung (marginal distribution)
Jeweils Zeilen oder Spalten summieren.

$$p_i = p_{i,\cdot} = \sum_{j=1}^{n_y} p_{ij}$$

Unabhängigkeit
(independence) von x und y gilt

Zwei Variablen x und y sind statistisch genau dann unabhängig, wen

$$p(x,y) = p(x)p(y)$$

$$p_{ij} = p_{i,\cdot} p_{\cdot,j}$$

Bedingte Wahrscheinlichkeit
Bezeichnet A das Ereignis x = i und B das Ereignis y = j. Die bedingte Wahrscheinlichkeit (conditional probability) von A unter B, P(A | B) ist die Wahrscheinlichkeit, dass A eintritt, nachdem B bereits eingetreten ist. Sie ist gegeben durch

$$P(A \mid B) = \frac{P(A,B)}{P(B)} = \frac{P(x=i, y=j)}{P(y=j)} = \frac{p_{ij}}{p_{\cdot,j}}$$

Sind die bedingten Wahrscheinlichkeiten und die Randverteilungen bekannt, so kann die joint probability wie folgt berechnet werden:

$$P(A,B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

Sind x und y unabhängig, so gilt:

$$P(A \mid B) = P(A)$$

Bayes-Theorem
erlaubt, die bedingte Wahrscheinlichkeit P(A | B) als Funktion der Randverteilungen P(A), P(B) und der bedingten Wahrscheinlichkeit P(A | B) auszudrücken.

$$P(A \mid B) = \frac{P(A \mid B)P(B)}{P(A)}$$

in Worten:

$$\text{A-posteriori-Wahrscheinlichkeit} = \frac{\text{Likelihood} \times \text{A-priori-Wahrscheinlichkeit}}{\text{Evidenz}}$$

Bayes-Theorem in der Mustererkennung

x repräsentiert ein Merkmal und w die Klassenzugehörigkeit von Mustern, so gibt im Falle der beobachteten Merkmalsausprägung x = i

$$P(\omega = j \mid x = i) = \frac{P(x = i \mid w = j)P(w = j)}{P(x = i)}$$

die Wahrscheinlichkeit an, dass das Muster zur Klasse j gehört. In der Literatur wird of statt w = j auch wj geschrieben.

Der Nenner P(x=i) kann so berechnet werden. c ist die Anzahl der Klassen.

$$P(x = i) = \sum_{j=1}^{c} P(x = i \mid \omega_j)P(\omega_j)$$

Bayes Decision Rule

Bemerkung: Um eine Entscheidung über Klassenangehörigkeit zu treffen, braucht man den Nenner nicht.

$$k = \arg\max_j P(\omega_j \mid x = i)$$

Die Fehlerrate (error rate) berechnet sich dann als

$$P(error) = \sum_{i=1}^{n} P(error \mid x = i)P(x = i)$$

4. Stetige Merkmale

Statistische Grundlagen (Stetige Zufallsvariablen)
Bisherige Zufallsvariablen waren diskret. (zB. es gibt 4 Fischlängen)
Neu: stetige Zufallsvariablen:

Es macht keinen Sinn, über die Wahrscheinlichkeit, dass x einen bestimmten Wert annimmt (zB. x = 1,3422), da diese Wahrscheinlichkeit von einem exakten Wert fast immer 0 ist. Stattdessen verwenden wir die Wahrscheinlichkeit, dass x in irgendeinem Intervall (a, b) fällt.

Dichtefunktion
Statt die Wahrscheinlichkeit P(x), wird eine Dichtefunktion p(x) verwendet (pdf, probability density function).

$$P\left[x \in (a,b)\right] = \int_a^b p(x)\,dx$$

Vorausgesetzt:

$$p(x) \geq 0 \quad \text{und} \quad \int_{-\infty}^{\infty} p(x) = 1$$

Erwartung

$$\varepsilon\left[f(x)\right] = \int_{-\infty}^{\infty} f(x)p(x)\,dx$$

Mittelwert

$$\mu = \varepsilon\left[x\right] = \int_{-\infty}^{\infty} xp(x)\,dx$$

Varianz

$$\mathrm{var}\left[x\right] = \sigma^2 = \varepsilon\left[(x-\mu^2)\right] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \varepsilon\left[x^2\right] - (\varepsilon\left[x\right])^2$$

Bayes-Theorem für stetige Merkmale

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j)P(\omega_j)}{p(x)}$$

$$p(x) = \sum_{j=1}^{c} p(x \mid \omega_j)P(\omega_j)$$

Likelihood Ratio

Entscheidung über Klassenzugehörigkeit

Übersteigt die likelihood ratio den threshold, entscheidet man sich für *w1*, sonst für *w2*

$$\frac{p(x \mid \omega_1)}{p(x \mid \omega_2)} > \frac{P(\omega_2)}{P(\omega_1)}$$

Links: Likelihood Ratio, Rechts: Threshold

Fehlerwahrscheinlichkeit

Mittlerer Fehler P(error) oder auch error rate berechnet sich als

für c = 2:

$$P(error) = \int_{-\infty}^{\infty} P(error \mid x)p(x)dx$$

für c >= 2:

$$P(error \mid x) = \sum_{j \neq i} P(\omega_j \mid x) = 1 - P(\omega_i \mid x)$$

Optimalität der Bayes Decision Rules

$$k = \arg\max_j P(\omega_j \mid x)$$

$$P(error \mid x) = \min\left[P(\omega_1 \mid x), P(\omega_2 \mid x)\right]$$

$$P(error) = \int_{-\infty}^{\infty} P(error \mid x)p(x)dx = \int_{-\infty}^{\infty} \min\left[P(\omega_1 \mid x), P(\omega_2 \mid x)\right]p(x)dx$$

Bayes-Theorem in der Mustererkennung (2)

Schätzung der Prior $P(\omega_j)$:

Oft recht einfach. Basiert auf Klassenverteilung des Trainingssets und weiterem Wissen über diese Verteilung. Oft gleiche Priors für jede Klasse.

Evidence:

$$p(x) = \sum_{j=1}^{c} p(x \mid \omega_j) P(\omega_j)$$

c ... Anzahl von Klassen

Schätzung der pdf $p(x \mid \omega_j)$:

Schwieriger. Jetzt nur ein stetiges Merkmal x pro Stichprobe. Problem: Betrachtung der Stichproben einer Klasse ▸ Suchen Dichtefunktion

Parametrische & Nichtparametrische Verfahren

2 Möglichkeiten
- parametrisches Verfahren
    1. zuerst eine Annahme über Form der pdf Funktion machen (zB. Gauss Kurve)
    2. parameter der Kurve schätzen

    Dichte wird für so viele Punkte wie möglich berechnet, weitere Punkte werden interpoliert.
    Vorteil: Wenn Form bereits bekannt, gibt es weniger Parameter zu schätzen
        Man kommt mit kleinerem Trainingsset aus.
    Nachteil: nicht so flexibel wie nicht parametrisches Verfahren.

- nicht parametrisches Verfahren
    1. keine Annahme über die pdf machen

Nicht Parametrische Verfahren

<u>k-Nearest Neighbour Verfahren</u>

- Punkt x_0 wählen
- Linie von x_0 bis k Stichproben
- V_x ... Länge der Linie am Punkt x_0
- $1/V_x$ ... Schätznung der Dichte

Parametrisches Verfahren

Die Normalverteilung (Grundlageneinwurf)

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left[(x-\mu)^2/2\sigma^2\right]}$$

$$p(x) \sim N(\mu, \sigma^2)$$

Eigenschaften der Normalverteilung

$$\varepsilon[1] = \int_{-\infty}^{\infty} p(x)dx = 1$$

Mittelwert:

$$\varepsilon[x] = \int_{-\infty}^{\infty} p(x)dx = \mu$$

Varianz:

$$\varepsilon\left[(x-\mu)^2\right] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2$$

Schätzung der Parameter der Normalverteilung

Mittelwert:

$$\hat{m}_j = \frac{1}{N_j} \sum_{x_i \in \omega_j} x_i$$

Schätzung der Populationsvarianz

$$\hat{s}_j^2 = \frac{1}{N_j - 1} \sum_{x_i \in \omega_j} (x_i - \hat{m}_j)^2 = \frac{1}{N_j - 1} \left(\sum_{x_i \in \omega_j} x_i^2 - N_j \hat{m}_j^2 \right)$$

5. Multivariate Merkmale

Von einem Merkmal zu zwei Merkmalen.
Univariate pdfs ▸ bivariate pdfs (Funktionen)

Statistik von bivariaten Funktionen

Die Kovarianz

$$\sigma_{XY} = \varepsilon\Big[(x-\mu_X)(y-\mu_y)\Big] = \sum_{x\in X}\sum_{x\in X}(x-\mu_X)(y-\mu_Y)P(x,y)$$

Die Kovarianz ist ein Maß für die statistische Abhängigkeit zwischen den Variablen x und y.

Schätzung der Kovarianz

$$\widehat{s}_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\hat{m}_x)(y_i-\widehat{m}_y)$$

Multivariate Merkmale

Mittelwert-Vektor

$$\mu = \varepsilon[x] = \begin{bmatrix}\mu_1\\ \mu_2\\ \vdots\\ \mu_p\end{bmatrix} = \begin{bmatrix}\varepsilon[x_1]\\ \varepsilon[x_2]\\ \vdots\\ \varepsilon[x_p]\end{bmatrix}$$

Kovarianz-Matrix

$$Cov(x) = \Sigma = \begin{bmatrix}\sigma_{11} & \cdots & \sigma_{1p}\\ \vdots & \ddots & \vdots\\ \sigma_{p1} & \cdots & \sigma_{pp}\end{bmatrix} = \varepsilon\Big[(x-\mu)(x-\mu)^T\Big]$$

Schätzung des Mittelwert-Vektors

$$\hat{m} = \frac{1}{N}\sum_{i=1}^{N}x_i$$

Schätzung der Kovarianz-Matrix

$$\hat{C} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\hat{m})(x_i-\hat{m})^T$$

Eigenschaften der Kovarianz-Matrix

$$\sum = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_3^2 \end{bmatrix}$$

Hauptdiagonale = Varianz, Rest = Kovarianz

Die Multivariate Normalverteilung (Grundlageneinwurf)

Mahalanobis-Distanz

$$d^2(x) = (x-\mu)^T \sum^{-1} (x-\mu)$$

Die Menge aller Punkte für die die Mahalanobis Distanz gleich einer Konstanten c ist, ist durch ein Hyperelipsoid mit Mittelpunkt p gegeben.

6. Merkmalsextraktion 2

Principal Component Analysis (PCA)

Die Principal Component Analysis (PCA, Hauptkomponentenanalyse) ist ein Verfahren der multivariaten Statistik. Sie dient dazu, umfangreiche Datensätze zu strukturieren, zu vereinfachen und zu veranschaulichen, indem eine Vielzahl statistischer Variablen durch eine geringere Zahl möglichst aussagekräftiger Linearkombinationen (die „Hauptkomponenten“) genähert wird.

PCA steht für *Principal Component Analysis* und bedeutet, dass ein p-dimensionaler Vektor ohne Fehler durch die Summe von p linear unabhängigen Vektoren dargestellt werden kann.

Gesichtserkennung mit Eigenfaces

PCA in der Gesichtserkennung: Eigenfaces
Gesichtserkennung mit Eigenfaces
Trainingsdaten sind standardisierte Bilder von Gesichtern.
Repräsentation der Gesichter mittes reduzierter Basis von Eigenvektoren: die Eigenfaces!

Verfahren:
Bild wird standardisiert & als eine Summe von bestehenden Eigenfaces repräsentiert
Eigenface-Koeffizienten werden mit Koeffizienten einer Datenbank von Gesihctern verglichen
Ähnlichster Vektor wird identifiziert.

Noch ein Gesichtserkennungsverfahren: Elastic graph matching
Ein Model eines Gesichts als Graph mit Merkmalen in den Knoten gespeichert
Graph wird an Bilder angepasst um Bildgraph zu erzeugen
Matching zw. Bildgraph und Graphen aus Datenbank wird ausgeführt

Mood Recognition: nicht gut für Gesichtserkennung, aber Gefühlserkennung
Leistungsevaluierung von biometrischen Systemen:

Leistungsevaluierung von Biometrischen Systemen
90% richtig, wenn ersten zwei Treffer beachtet werden

Vier Leistungsmaße:
- True acceptence
- True rejection
- False acceptence rate (FAR)*
- False rejection rate (FRR)*

*Fehlermaße

Link zwischen FAR und FRR
je niedriger FAR, umso höher FRR und umgekehrt.
Ein Sicherheitsgrad basierend auf der Anwendung soll ausgewählt werden.

Evaluierung von Gesichtserkennungs-Systemen
Performance over time

Curse of Dimensionality
Allgemein nicht so viele Merkmale wie möglich in Merkmalsvektor packen. In der Praxis können zusätzliche Merkmale of die Leistung des Klassifikators vermindern, wenn Anzahl der Trainingsbeispiele gering ist.

Für kleines Trainingsset soll kleine Anzahl hervortretender Merkmale ausgewählt werden. Fasutregel: Sei n die Anzahl der Trainingsstichproben pro Klasse, und d die Anzahl an Merkmalen. Das Verhältnis n/d soll > 10 sein. (Für komplexere Klassifikatoren soll es noch größer sein.)

Iriserkennung
Iris ist sehr präzises biometrisches Merkmal:
- sehr datenreiche physikalische Struktur
- Stabilität mit der Zeit
- physikalischer Schutz (durch Hornhaut)

Foto von Auge machen
Iris wird erkannt
Iris-Merkmale werden durch Faltung berechnet (komplexe 2D Gabor Wavelets, unterschiedliche Positionen)
Merkmale werden in Binärcode konvertiert.

Iris-Code Vergleich
Der Unterschied zw. zwei Iris Binärcodes ist die Anzahl ungleicher Bits (**Hamming Distanz**)
100101
000110
…. ergibt spaltenmäßig 2 Unterschiede: in der ersten und 5. Spalte. HD = 2/6 = 1/3
HD = 0 … identisch
HD = 1… total unterschiedliche
HD <= 0.32 … ausreichend ähnlich

Iris Aufnahmegeräte
active: Kamera ist beweglich, Benutzer muss Kamera in richtige Position bringen, System gibt Feedback
passive: Benutzer steht vor dem Gerät, Gerät macht alles notwendige

Passive System Beispiel
Iriserkennung Evaluierung
ICE 2006 Evaluierung
Vergleich mit Face Recognition

7. Bayes-Klassifikation für normalverteilte Merkmale

Diskriminanten-Funktion

Ein Klassifikator kann auch als eine Gruppe von Diskriminanten-Funktionen dargestellt werden. Mit jeder Klasse wi wird eine Funktion gi(x) verbunden.

$$g_i(x) > g_j(x) \text{ für alle } j \neq i$$

Gegeben c Klassen. Die Diskriminanten-Funktionen teilen den Merkmalsraum in c Entscheidungsregionen (decision regions)

$$\text{Wenn } g_i(x) > g_j(x) \text{ für alle } j \neq i, \text{ dann } x \text{ in } \Re_i \text{ und gehört zur Klasse } \omega_i.$$

Die Regionen sind durch Entscheidungsgrenzen (decision boundaries) getrennt. Die Entscheidungsgrenze zwischen den Klassen wj und wk ist durch die Gleichung

$$g_j(x) = g_k(x)$$

gegeben.

Bayes-Klassifikation für normalverteilung Merkmale

Sind im Speziellen die Merkmale für alle Klassen normalverteilt mit pdf, so erhält man durch Logarithmieren der posteriors die folgenden (optimalen) Diskriminanten-Funktionen:

$$g_j(x) = \ln \frac{P(\omega_j)p(x \mid \omega_j)}{p(x)}$$

$$= -\frac{1}{2}(x-\mu_j)^{\mathrm{T}} \sum_j^{-1} (x-\mu_j) - \frac{1}{2}\ln \mid \sum_j \mid + \ln P(\omega_j) - \frac{p}{2}\ln 2\pi - \ln p(x)$$

Man bemerkt, dass die beiden letzten Komponenten

$$-\frac{p}{2}\ln 2\pi - \ln p(x)$$

nicht von wj abhängen und daher beim Vergleich der gj nicht berücksichtigt werden müssen. Die gj sind im Falle normalverteilter Merkmale somit quadratische Funktionen in x

$$g_j(x) = -\frac{1}{2} d_j^2(x) + \left[-\frac{1}{2}\ln \mid \sum_j \mid + \ln P(\omega_j) \right]$$

$$\text{wobei } d_j^2(x) \text{ die Mahalanobis-Distanz der Klasse } \omega_j \text{ bezeichnet:}$$

$$d_j^2(x) = (x-\mu_j)^T \sum_j^{-1} (x-\mu_j)$$

Siehe Beispiel in den Folien: Entscheidungsgrenze für 2 Klassen mit 2 Merkmalen wir berechnet. Vorgangsweise: 1.) Mittelwerte und Kovarianz-Matrizzen für jede Klasse berechnen 2.) Berechnung von g1(x) und g2(x) 3.) g1(x) mit g2(x) gleichsetzen.

Mahalanobis-Distanz:

$$d_j^2(x) = (x - \mu_j)^T \sum_j^{-1} (x - \mu_j)$$

Spezialfall 1:

$$\sum_j = I\sigma^2$$

Merkmale sind innerhalb jeder Klasse dekorelliert. Weiters weisen alle Komponenten die selbe Varianz auf.
Wenn Terme für alle Klassen gleich sind, kann man sie weglassen.
Spezialfall 2: Summe j = Summe
Allgemeinfall: Summe j Beliebig
Mehr als zwei Klassen

8. Lineare Diskriminanten- Funktionen

Einleitung

Warum lineare Diskriminanten-Funktion?
- Sie haben viele angenehme Eigenschaften.
- Sie können optimal sein, wenn die Datenverteilungen geeignet sind (während der Definition von Merkmalen zu beachten).
- Lineare Modelle sind schnell und einfach zu trainieren und auszuwerten.
- Sie sind oft geeignet für einen ersten „Proben-Klassifikator".

Das Perceptron

Ein einfacher binärer Klassifikator: das Perceptron
Das Perceptron stellt einen Spezialfall eines binären, linearen Klassifikators dar. Es wurde ca 1960 von Rosenblatt als Modell eines künstlichen neuronalen Netzwerks entwickelt. Die Architektur des Perceptrons entspricht einer linearen Diskriminanten-Funktion mit nachgeschaleter Signum-Funktion.

Wir gehen im folgenden von d-dimensionalen Merkmalsvektoren $x \in \mathbb{R}^d$ und zwei Klassen ω_1, ω_2 aus.

Ziel ist es, eine Diskriminanten-Funktion $g : \mathbb{R}^d \rightarrow \mathbb{R}$ zu finden, welche die Klassenzugehörigkeit wie folgt kodiert:

$$g(x) > 0 \text{ falls } x \in \omega_1$$
$$g(x) < 0 \text{ falls } x \in \omega_2$$

wobei der Absolutbetrag von g das „Vertrauen" in die vorgesagte Klassenzugehörigkeit von x widerspiegelt.

Im speziellen Fall einer linearen Diskriminanten-Funktion hat g die folgende Form:

$$g(x) = \sum_{i=1}^{d} w_i x_i - \theta = w^T x - \theta$$

wobei:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$w \in \mathbb{R}^d$ wird oft als Gewichtsvektor und $\theta \in \mathbb{R}$ als bias oder threshold bezeichnet.

Die Aufgabe besteht nun darin, geeignete Werte für w und θ zu finden.

Wenn wir mit o(x) die Ausgabe des Perceptrons bezeichnen, so haben wir

$$o = o(x) = \text{sgn}(w^T x - \theta) = \begin{cases} 1 & \text{falls } w^T x \geq \theta \\ -1 & \text{falls } w^T x < \theta \end{cases}$$

Das Ziel ist nun, den Gewichtsvektor w und bias θ zu bestimmen, sodass:

$$o(x) = 1 \qquad (\Leftrightarrow \mathrm{w}^T x \geq \theta) \;\; \text{falls } x \in \omega_1$$
$$o(x) = -1 \qquad (\Leftrightarrow \mathrm{w}^T x < \theta) \;\; \text{falls } x \in \omega_2$$

Geometrische Interpretation

Für $w,\ x \in \mathbb{R}^d$, legt

$$\sum_{i=1}^{d} w_i x_i = w^T x = \theta$$

eine in den $\mathbb{R}^d$ eingebettete (d – 1)-dimensionale Hyper-Ebene mit Normalvektor w fest (im Fall d = 2 eine Gerade).

Im Fall θ = 0 geht die Hyper-Ebene durch den Ursprung, anderenfalls ist sie entlang w um den Betrag $\theta / ||w||$ vom Ursprung verschoben.

Lineare Spearierbarkeit (linear separability)

Sei $X = (x_1, x_2, ..., x_N) \in \mathbb{R}^{d \times N}$ eine Menge von N Merkmalsvektoren mit zugeordneten Klassen-Labels $y = (y_1, \mathrm{y}_2, ..., y_N),\ \mathrm{y}_i \in \{1, -1\}$.

Wir sagen dass X linear separierbar (bzg. y) ist, falls es einen Gewichtsvektor w und bias θ gibt, sodass

$$o(x_i) = \text{sgn}(w^T x_i - \theta) = y_i, \quad 1 \leq i \leq N$$

Homogene Koordinaten (Homogenous Coordinates)

Der bias kann durch einen kleinen Kunstgriff in den Gewichtsvektor „hineingezogen“ werden.

Wir führen zu diesem Zweck zusätzliche Koordinaten $x_0 \equiv 1$ und $w_0 = -\theta$ ein.

$$ {}^{a}x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \quad {}^{a}x = \begin{bmatrix} -\theta \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} $$

Wir haben somit

$$ g(x) = {}^{a}w^{T}\,{}^{a}x = \sum_{i=0}^{d} w_i x_i = -\theta + \sum_{i=1}^{d} w_i x_i = w^T x - \theta $$

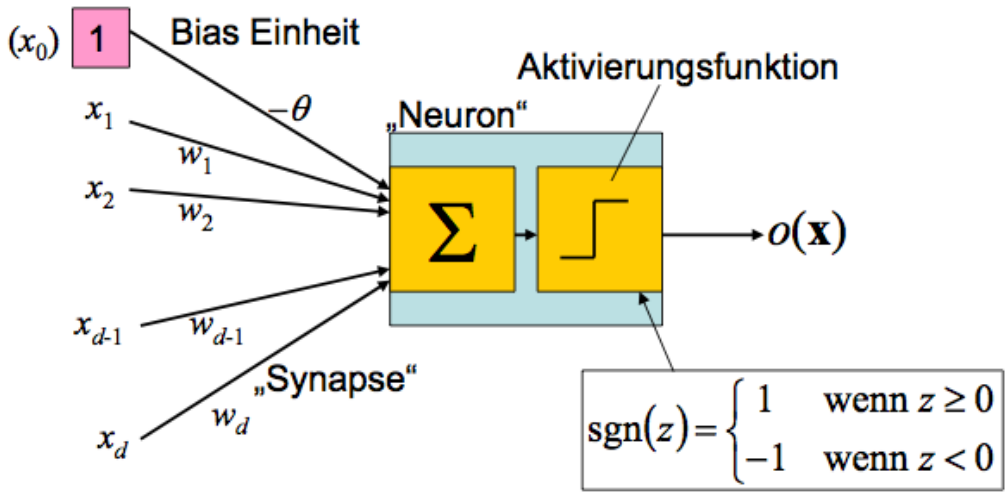
Die Transformation in homogene Koordinaten vereinfacht unser ursprüngliches Problem, indem es dessen Dimensionalität um 1 (von d auf d + 1) erhöht – die Gleichung oben definiert nun eine d-dimensionale Hyper-Ebene im $\mathbb{R}^{(d+1)}$, welche durch den Ursprung geht.

Wir werden im folgenden – falls nicht anders erwähnt – stets homogene Koordinaten annehmen und daher das Superscript a weglassen.

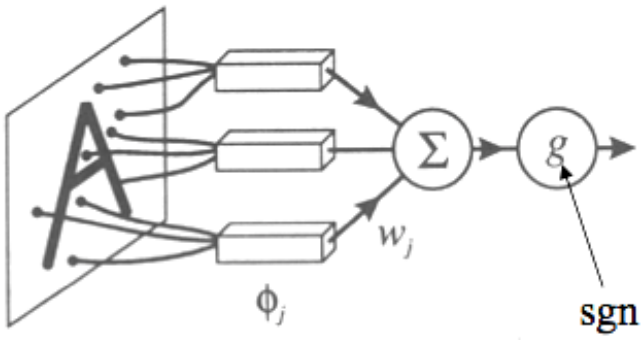
Das Perceptron als „Neutron“
Wdh.: Die Ausgabe eines Perceptrons ist

$$ o(x) = \operatorname{sgn}\left({}^{a}w^{T}\,{}^{a}x\right) = \operatorname{sgn}\left(\sum_{i=0}^{d} w_i x_i\right) = \operatorname{sgn}\left(-\theta + \sum_{i=1}^{d} w_i x_i\right) $$

Das Perceptron kann wie folgt dargestellt werden.



Rosenblatt hat 1958 ein mechanisches Perceptron entwickelt, das mit Buchstabenerkennung erfolgreich war.



Er hat eine zusätzliche Schicht eingebaut, aus Einheiten ϕ_i
– die üblicherweise wieder eine Schwellwert-Funktion implementiert haben.
– mit unveränderbaren Gewichten.
– die an eine zufällige Teilmenge der Eingabepixel angeschlossen waren.

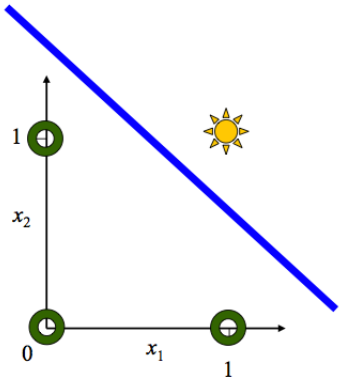
Die Ausgabe dieses Perceptrons ist

Der Perceptron Training Algorithmus wurde verwendet, um die Gewichte w_j zu verändern.

Dieser Algorithmus kann nicht in der ersten Schicht (die θ_j Einheiten) verändern.

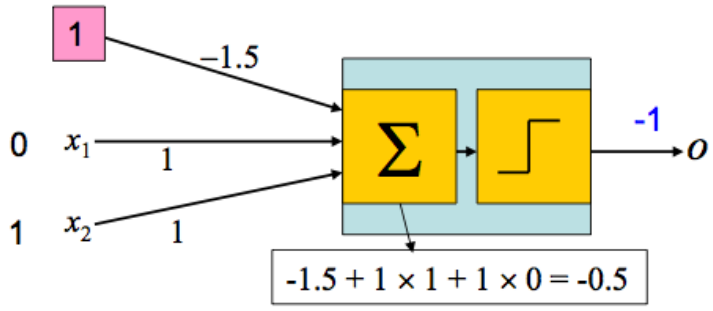
AND-Problem

x_1	x_2	o
0	0	0
0	1	0
1	0	0
1	1	1



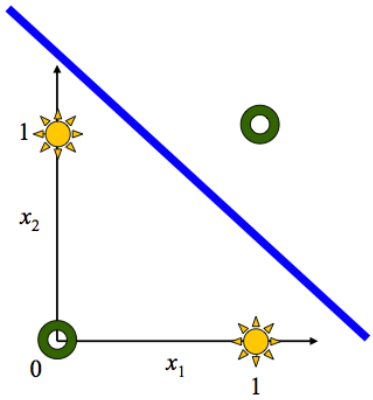
Linear Separierbar

Lösung:



XOR-Problem

x_1	x_2	o
0	0	0
0	1	1
1	0	1
1	1	0



Ein Perceptron kann die XOR Funktion nicht modellieren, weil sie nicht linear-separierbar ist. Es kann durch ein multi-layer Perceptron repräsentiert werden welches im nächsten Kapitel gezeigt wird.

Wichtiger Nachteil
Ein einstufiges Perceptron kann nur linear separierbare Mengen, das heißt Mengen, die durch eine Hyperebene trennbar sind, klassifizieren.

Training eines Perceptrons
Sei $S_{Tr} = \{X, y\}$ das Trainingsset,

eine Menge von N homogenen Merkmalsvektoren $X = (x_1, x_2, \ldots, x_N) \in \mathbb{R}^{(d+1)\times N}$

und korrespondierenden Klassen-Labels $y = (y_1, y_2, \ldots, y_N),\ y_i \in \{1, -1\}$.

Wollten wir z.B. das binäre AND-Problem mittels eines Perceptrons lösen, so hätte unser Trainingsset folgende Form:

$$X = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \qquad y = (-1,-1,-1,\ 1)$$

Ziel: finde einen homogenen Gewichtsvektor w, sodass

$$o(x_i) = \operatorname{sgn}(w^T x_i) = y_i, \quad 1 \le i \le N$$

Idee: falls ein „positiver" Trainingsvektor x_j mit $y_j = 1$ falsch klassifiziert wurde $(\Rightarrow w^T x_j < 0)$, so addiere ein Vielfaches von x_j zu w : dadurch wird die Hyper-Ebene auf den falsch klassifizierten Vektor hinbewegt.

Man sieht dass

$$(w + \gamma x_j)^T x_j = w^T x_j + \gamma \left\| x_j \right\|^2 > w^T x_j, \quad \gamma > 0$$

Der positive Faktor γ wird auch Lernrate genannt.

Analog zum obigen Fall, sollte im Fall eines missklassifizierten „negativen" Trainingsvektors x_j die Hyper-Ebene von diesem wegbewegt werden (indem wir Vielfaches von x_j von w subtrahieren).

In beiden Fällen ist es möglich, dass (abhängig vom Wert von γ), zuvor korrekt klassifizierte Vektoren durch die neue Hyper-Ebene nun falsch klassifiziert werden.

Wir können beide Fälle (positiv und negativ) abdecken, indem wir beachten dass

$$\operatorname{sgn}(w^T x_i) = y_i \Leftrightarrow \operatorname{sgn}(w^T x_i) y_i = 1 \quad \Rightarrow \quad (w^T x_i) y_i > 0 \Leftrightarrow w^T (x_i y_i) > 0$$

Ausgehend von der untersten Gleichung, suchen wir nun nach einem Gewichtsvektor, welcher das modifizierte Trainingsset $x_i, y_i, \ 1 \le i \le N$ (mit ausschließlich positiven Klassen-Labels) in die positive Halb-Ebene abbildet.

Dies führt uns zum...

Online Perceptron Training Algorithmus

1. Initialise **w**, γ
2. **do**
3. **for** $i = 1$ **to** N
4. **if** $\mathbf{w}^T(\mathbf{x}_i y_i) \leq 0$ (misclassified ith pattern)
5. $\mathbf{w} \leftarrow \mathbf{w} + \gamma \mathbf{x}_i y_i$
6. **end if**
7. **end for**
8. **until** all patterns correctly classified

Die Schritte 3.–7. (Präsentation aller N Trainingsbeispiele) werden häufig als Epoche bezeichnet, der Schritt 5. als Gewichts-Update.

Zwei wichtige Fragen:
- Wie sollen **w**, γ initialisiert werden?
- Terminiert der Algorithmus immer in einer endlichen Anzahl von Schritten?

Initialisierung

Sei $w = 0$. In diesem Fall ist der mit dem obigen Algorithmus erhaltene Gewichtsvektor w_p eine Linearkombination der während des Trainings falsch klassifizierten Merkmalsvektoren:

$$w_p = \sum_{i=1}^{N} x_i (y_i \gamma k_i) = \gamma \sum_{i=1}^{N} x_i (y_i k_i), \quad k_i \in \mathbb{N}_0$$

wobei k_i angibt, wie oft der i-te Merkmalsvektor falsch klassifiziert wurde.

Wir sehen, dass wenn wir **w** und θ mit demselben positiven Faktor $\alpha \in \mathbb{R}^+$ multiplizieren, die Entscheidungsregionen unverändert bleiben:

$$w^T x = \theta \Leftrightarrow (\alpha w)^T x = \alpha\theta \quad \left(\forall x \in \mathbb{R}^d\right)$$

$$w^T x \geq \theta \Leftrightarrow (\alpha w)^T x \geq \alpha\theta \quad \left(\forall x \in \mathbb{R}^d\right)$$

Folglich ist γ lediglich ein Skalierungsfaktor und hat keinen Einfluss auf die Entscheidungsgrenze.

Daher können wir bequemerweise einfach $\gamma = 1$ setzen (dies gilt nicht für alle Lernverfahren!).

Perceptron Konvergenz-Theorem

Es ist möglich zu beweisen, dass der online Perceptron Training Algorithmus mit fixer Lernrate γ für jedes linear separierbare Trainingsset mit Lösung w_p terminiert.

Der Algorithmus terminiert nicht im Falle eines nicht linear separierbaren Trainingssets (z.B. XOR- Problem).

Margin

Der geometrische Margin $gm(x_i)$ von einem Vektor x_i ist der Abstand zwischen x_i und der trennenden Hyper-Ebene.

Man beachte, dass $gm(x_i) > 0$ g.d.w. x_i korrekt klassifiziert wird.

Der Vektor *j* mit minimaler geometrischer Margin

$$j = \arg\min_i gm(x_i), \quad 1 \le i \le N$$

legt die geometrische Margin $gm(w)$ der Hyper-Ebene bzg. des Trainingssets {X, y} fest:

$$gm(w) = gm(x_j)$$

Die Generalisierungsfähigkeit des Perceptrons wird umso besser sein, je größer $gm(w)$ ist. Diese Idee – den minimalen Abstand der Trainingspunkte von der Hyper-Ebene respektive die Margin $gm(w)$ zu maximieren – liegt der support vector machine (SVM) zugrunde.
Man spricht in diesem Zusammenhang auch von maximum margin classifiers.

Verwandte Verfahren

Der Perceptron-Algorithmus in der hier präsentierten Form hat zwei wesentliche Nachteile, welche die Entwicklung leistungsfähiger Verfahren motiviert haben:

- Der Perceptron-Algorithmus terminiert nicht im Fall nicht linear separierbare Daten. Der mit dem Perceptron verwandte Ho-Kashyap Algorithmus erkennt diesen Fall und terminiert auch auf nicht linear separierbaren Daten.

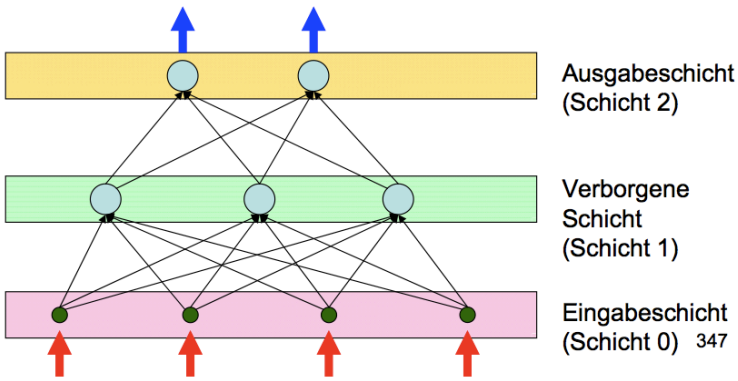
- Das Perceptron findet nicht unbedingt die optimale Lösung mit maximaler Margin. Die moderneren SVMs hingegen finden die optimale Lösung. Es gibt auch verschiedene Erweiterungen der SVMs für nicht linear separierbare Daten.

9. Neuronale Netze

Multilayer Perceptron (MLP)

Das Backpropagation Netz
Backpropagation ist ein Algorithmus für das Training multischichtiger neuronaler Netze. (In den 1970ern entwickelt, für viele Jahre vergessen und dann wieder entdeckt.) Der Begriff Multilayer Perceptron (MLP) wird auch für das Backpropagation-Netz verwendet.

Ein Backpropagation-Netz besteht aus:
– einer Eingabeschicht
– einer Ausgabeschicht
– mindestens einer verborgenen Schicht (hidden layer).
– Verbindungen zwischen den Schichten.



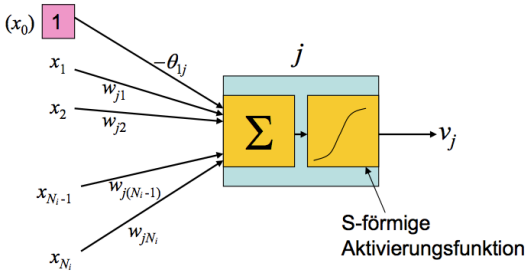
347

Ein **Neuronales Netz** besteht typischerweise aus 10 bis 1000 künstlichen Neuronen (Einheiten). Das menschliche Gehirn besteht aus circa 10 000 000 000 Neuronen.

– ist nicht intelligent.
– kann nicht selbst denken oder überlegen.
– kann begrenzte Mustererkennungsprobleme lösen.

Es ist aber nicht als gegeben anzunehmen, dass ein Neuronales Netz besser als ein klassischer Mustererkennungs-Algorithmus funktioniert.

Die Einheiten (Units)
Sie earbeiten die Daten nicht. Sie schicken die Daten unverändert an jede Einheit der zweiten Schicht. Die verborgenen und Ausgabe-Einheiten schauen wie ein *Perceptron* aus.
Eine verborgene Einheit j ist hier dargestellt:



Aktivierungsfunktion

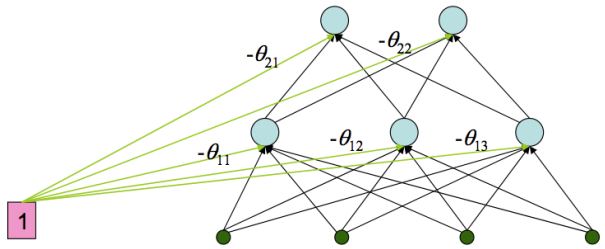
Üblicherweise eine differenzierbare S-förmige Aktivierungsfunktion der Einheiten ist üblicherweise:

– die sigmoid Funktion: $g(x) = \frac{1}{1+\exp(-x)}$

– oder die tanh Funktion: $g(x) = \tanh(x)$

Die zusätzliche Einheit

Wie bei dem Perceptron gibt es eine zusätzliche Einheit, die immer eine Aktivierung von 1 hat. Sie ist mit allen Einheiten verbunden. Der Bias jeder Einheit kann dann im Rahmen des Trainingsalgorithmus verändert werden.



Oft wird bei Darstellungen von eines Netzes allerdings auf die Darstellung dieser Einheit verzichtet.

Anzahl der Gewichte

$$N_i \times N_j + N_j \times N_k + N_j \times N_k$$

Gewichtsvektor

Der Zustand des Netzes kann durch einen Gewichtsvektor **w** beschrieben werden. Dieser enthält alle Gewichte (wji und wkj) und alle Biaswerte.

Forward-Pass

Berechnung der Ausgabe für einen Eingabevektor x:
Zuerst werden alle Aktivierungswerte der verborgenen Schicht berechnet.

$$v_j = g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right) = g\left(\sum_{i=1}^{N_i} w_{ji} x_i - \theta_{1j}\right)$$

Anschließend werden die Aktivierungswerte der Ausgabeschicht (Ausgabewerte) berechnet. (Ähnlich wie für die verborgenen Einheiten.)

$$o_k = g\left(\sum_{j=0}^{N_j} w_{kj} v_j\right) = g\left(\sum_{j=1}^{N_j} w_{kj} v_j - \theta_{2k}\right)$$

Diese zwei Gleichungen können zusammengefasst werden, sodass die Ausgabe direkt berechnet werden kann.

$$o_k = g\left(\sum_{j=0}^{N_j} w_{kj} v_j\right)$$

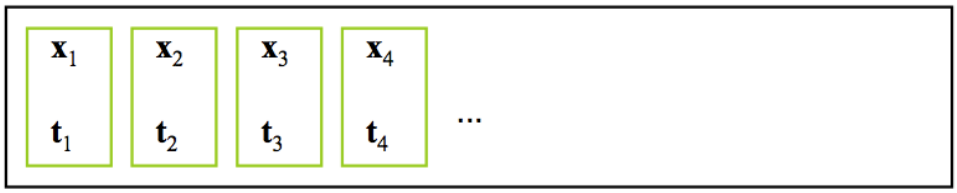
$$v_j = g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right)$$

$$o_k = g\left[\sum_{j=0}^{N_j} w_{kj}\, g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right)\right]$$

Lernen

Am Anfang hat ein Netz Zufallswerte als Gewichte.
Der Backpropagation Algorithmus modifiziert die Gewichte anhand der Beispiele in einem Trainingsset Dieser Prozess heißt **überwachtes Lernen**.
Das Trainingsset besteht aus Eingabevektoren xi und zugehörigen Ausgabevektoren (target vectors) ti (so viel wie möglich). Bei jedem Schritt soll der Fehler $J(\mathbf{w}) = \frac{1}{2}\|\mathbf{t}_i - \mathbf{o}_i\|^2$ kleiner werden.

Trainingsset:



Unter einem Trainingszyklus (Epoche) versteht man die vollständige Präsentation aller Trainingsbeispiele (jedes genau einmal).
Üblicherweise werden Netze in mehreren Zyklen trainiert. Da die Gefahr besteht, dass das Backpropagation Netz die „Reihenfolge“ der Trainingsdaten lernt, sollte die Reihenfolge der präsentierten Beispiele in jedem Zyklus unterschiedlich sein. Wir möchten, dass das Netz nach dem Training

- bei Präsentation des Eingabevektors xi den korrespondierenden Ausgabevektor ti berechnet.
- eine gute Generalisierungsfähigkeit hat. Das heißt, dass das Netz auch richtige Ausgabevektoren für Eingabevektoren, die nicht während des Trainings verwendet wurden, berechnet.

Das Netz muss an die Daten angepasst werden:

- Anzahl Eingabe-Einheiten = Anzahl Werte in Eingabevektoren.
- Anzahl Ausgabe-Einheiten = Anzahl Werte in Ausgabevektoren.

Die Anzahl verborgener Einheiten kann nicht so einfach festgestellt werden.

- Erfahrung
- Experiment

Eingabevektoren des Trainingsset

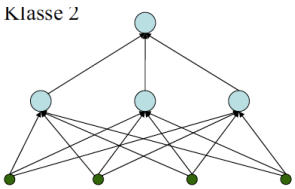
Die Eingabevektoren x sind Merkmalsvektoren, so wie bei anderen Algorithmen. „Gute“ diskriminierende Merkmale sollen verwendet werden.

Ausgabevektoren des Trainingsset

Die Ausgabevektoren t zeigen die Klassen Zugehörigkeit an. Mit zwei Klassen braucht man nur einen Wert im Vektor.

– t = {0} => Klasse 1

– t = {1} => Klasse 2



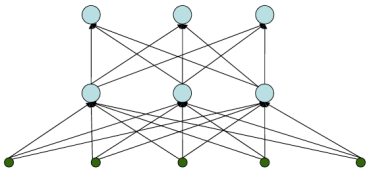
Mit $N > 2$ Klassen wird eine 1-of-N Kodierung verwendet.

– Der Vektor **t** enthält N Werte.

– Die Klasse j wird durch den Wert 1 an der Position j und sonst überall 0 dargestellt.

– z.B. **t** = {0, 0, 1, 0, 0} => Klasse 3 von 5

– **t** = {1, 0, 0} => Klasse 1 von 3



Ausgabe des Netzes

Obwohl das Netz mit exakten Werten in den Ausgabevektoren trainiert wird, ist der Ausgabevektor **o** des Netzes üblicherweise nicht so genau. Das heißt, mit 2 Klassen, gibt ein Netz Ausgabewerte im Bereich [0:1], aber nicht genau 0 oder 1. Wir verwenden dafür die folgende Regel:

Ein Merkmalsvektor **x** gehört zur
- Klasse 1 wenn $o_1 < 0.5$
- Klasse 2 wenn $o_1 \geq 0.5$

Für N Klassen, gehört **x** zur Klasse mit dem maximalen Aktivierungswert in der Ausgabeschicht.

Z.B. **o** = {0.15, 0.28, 0.90, 0.18}

x gehört zur Klasse 3

o = {0.10, 0.52, 0.48, 0.30}

x gehört zur Klasse 2, aber mit wenig Bestimmtheit.

Backpropagation Algorithmus (Überblick)

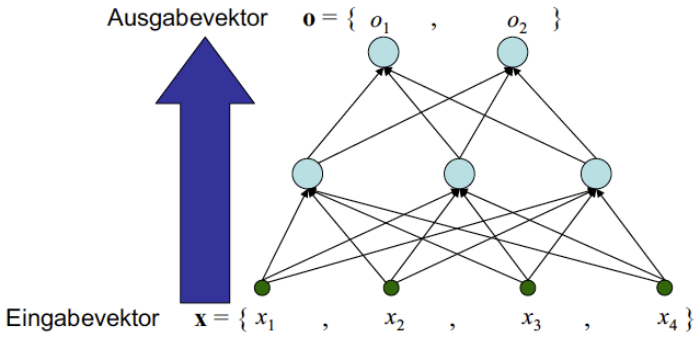
Die Beispiele vom Trainingsset werden verwendet. Die folgenden Schritte laufen alternierend ab:
- Forward-Pass
- Bestimmung des Fehlers
- Backward-Pass

Bis das Netz gut genug trainiert ist (der Fehler klein genug ist).

Die folgenden Schritte laufen alternierend ab:

– Forward-Pass

Ein beliebiger Eingabevektor **x** aus dem Trainingsset wird eingegeben (präsentiert). Der Ausgabevektor **o** wird berechnet.



– Bestimmung des Fehlers

- Die vom Netz gelieferte Ausgabe **o** wird mit der korrekten Ausgabe **t** verglichen.
- Üblicherweise ist das Ziel des Lernens diesen Fehler zu minimieren.

– Backward-Pass

- Der Backward-Pass erfolgt in umgekehrter Richtung wie der Forward-Pass.
- In ihm werden sukzessive die Gewichte, beginnend mit den Gewichten der Ausgabeschicht, anhand des Fehlers verändert.

Fehlerbestimmung

Der Fehler J(w) eines Netzes ist durch $J(\mathbf{w}) = \frac{1}{2}\|\mathbf{t}-\mathbf{o}\|^2 = \frac{1}{2}\sum_{k=1}^{N_k}(t_k - o_k)^2$ gegeben (**Sum of Squares Error**, SSE).

Der Fehler kann für ein Trainingsbeispiel oder für das ganze Trainingsset berechnet werden.

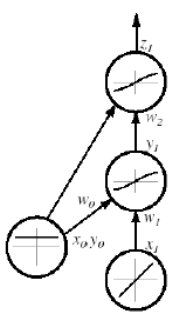
– Es gibt *N* Beispiele im Trainingsset.

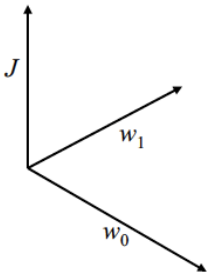
– *Jp* ist der für Beispiel *p* berechnete Fehler.

– Der Gesamt-Fehler ist $J = \sum_{p=1}^{N} J_p$

Fehleroberflächen

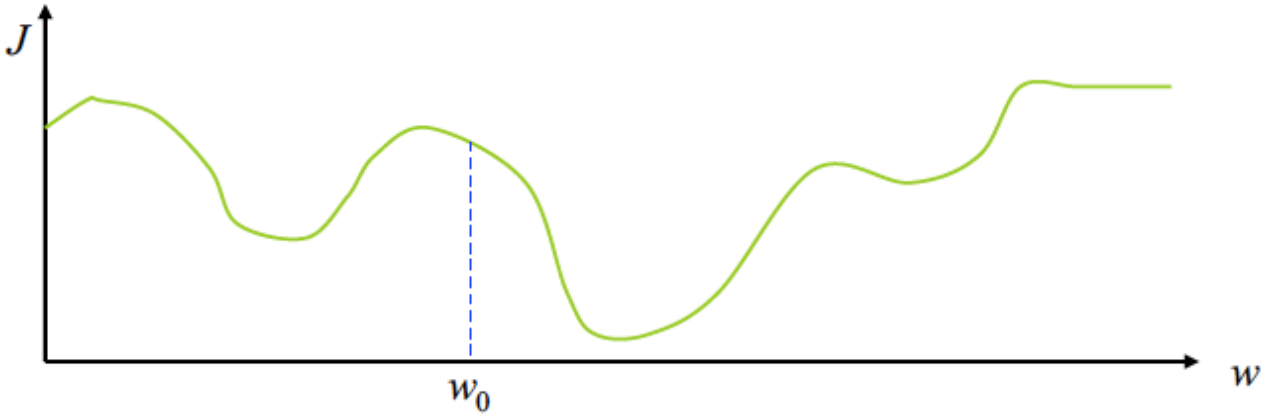
Mit einem einfachen Netz ist es leicht, eine Fehleroberfläche zu visualisieren. Für eine Teilmenge der Gewichte, können die Fehler *J*(wo, *wI)* berechnet werden, und ein Diagramm gezeichnet werden.





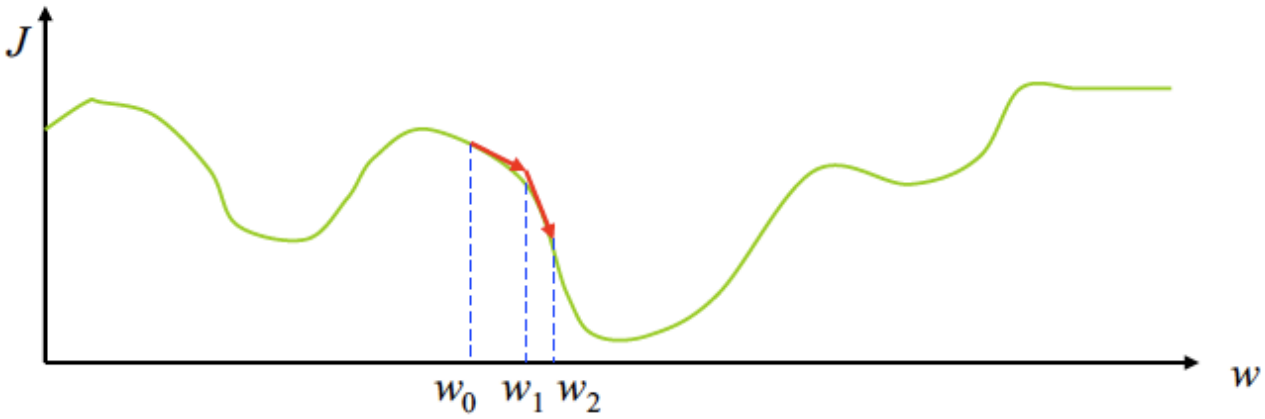
Die Backpropagation Lernregel

Zu einem gegebenen Gewichtsvektor wo eines Netzes gehört ein bestimmter Punkt auf der Fehleroberfläche (üblicherweise eine multi-dimensionale Oberfläche).
Die Lernregel soll die Werte von wo so verändern, dass wo in ein (möglichst globales) Minimum bewegt wird.



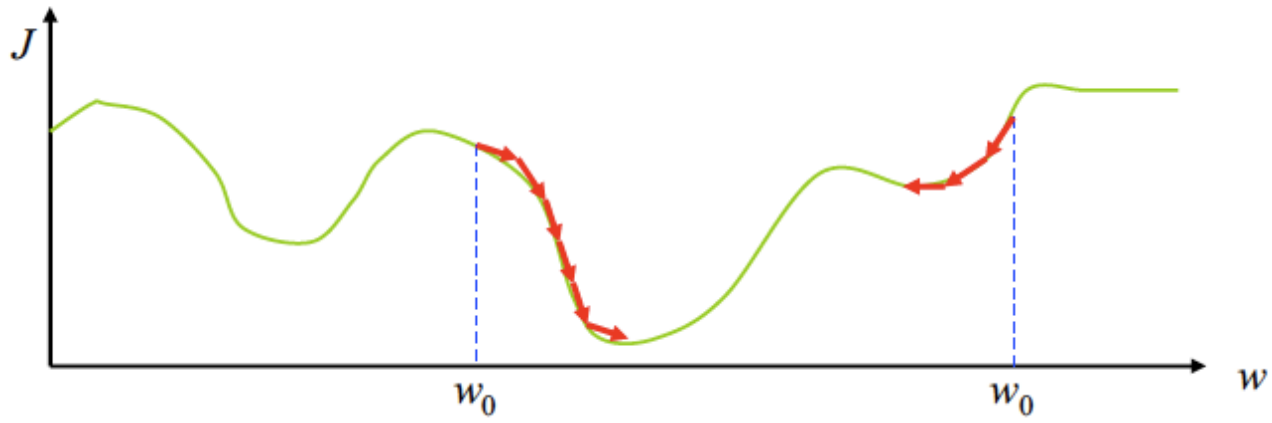
Der Backpropagation Algorithmus verwendet ein Gradientenabstiegsverfahren.

- Punkt **wO** wird in die Richtung des Steilsten Abstiegs um eine gewisse vorgegebene Länge bewegt.
- Man erhält so den Gewichtsvektor **wI**, und wiederholt das Verfahren.
- Der Algorithmus wird in der Vorlesung „Neural Computation 1“ besprochen.



Nachteile des Backpropagation Algorithmus

Lokale Minima: Es ist nicht sicher, dass der Backpropagation Algorithmus das globale Minimum findet. Dies hängt von den Anfangsgewichten ab.



Weitere Nachteile:

Konvergenzzeit: langsam (Anzahl benötigter Iterationen)

- Parameter Auswahl:
  - Anzahl verborgener Einheiten N_j
  - Lern-Parameter η
- Wann soll der Trainingsalgorithmus gestoppt werden? (Wie weiß man, dass das globale Minimum gefunden ist?)

Weiter Trainingsmethoden (Überblick)

Es gibt auch andere Trainingsalgorithmen, z.B.:

– Backpropagation mit Momentum.

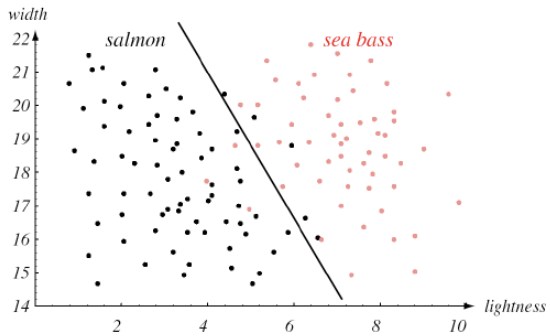
– Quickprop (Idee: die Fehleroberfläche wird durch eine quadratische Fläche angenähert).

– *Conjugate gradient descent* (Idee: line search method, eine Richtung wird gewählt und das Minimum entlang einer Linie in dieser Richtung wird gefunden).

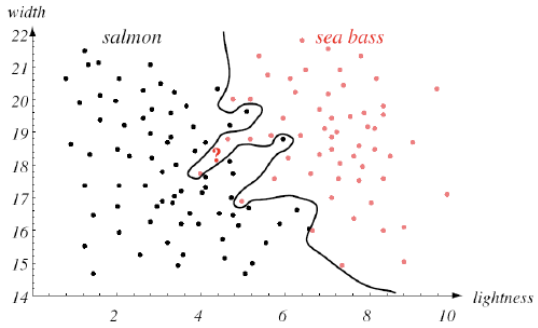
Mächtigkeit des Backpropagation-Netzes

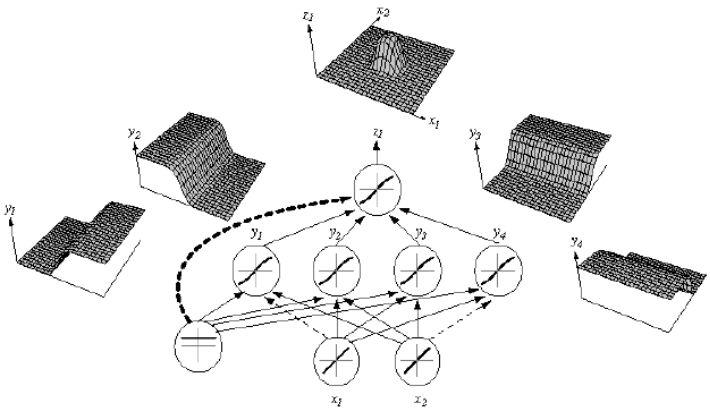
Ein zwei-, drei-, vier-, ..., schichtiges Netz mit genug verborgenen Einheiten, kann beliebige Entscheidungsgrenzen implementieren. Es gibt aber keine genauen Regeln wie viele verborgene Einheiten für ein bestimmtes Problem benötigt werden.

Ein Netz mit zu wenig verborgenen Einheiten kann das Trainingsset nicht lernen (**underfitting**).



Ein Netz mit zu viel verborgenen Einheiten, das zu lang trainiert wird, hat eine schlechte Generalisierungsfähigkeit (**overfitting**).





1.) Durch geeignete Wahl der Gewichte kann das gezeigte MLP eine Glockenfunktion erzeugen
2.) Durch eine „Superposition" einer genügend großen Anzahl verschiedener solcher „Glocken" an verschiedenen Positionen lässt sich jede Funktion erzeugen.

Testen eines Netzes

- Für das Testen der Klassifikationsfähigkeit eines Netzes (wie für allen Mustererkennungsalgorithmen), soll das Trainingsset nicht verwendet werden.
  – Ein Netz das alle Beispiele des Trainingsset richtig klassifiziert, hat nicht notwendigerweise eine gute Generalisierungsfähigkeit.
- Ein Testset wird dafür verwendet.
  – Die Beispiele des Testsets dürfen nicht für das Training verwendet werden.

Regularisierung (Regularization)

Mit normalem Lernen versuchen wir den Ausgabefehler des Netzes zu minimieren.
Mit regularisiertem Lernen versuchen wir, gleichzeitig die Komplexität des Netzes zu vereinfachen. Z.B. Die *Optimal Brain Damage* und *Optimal Brain Surgeon* Algorithmen:
– Entfernen Gewichte die den kleinsten Einfluss auf den Ausgabefehler haben.

Vorteile der Neuronalen Netze

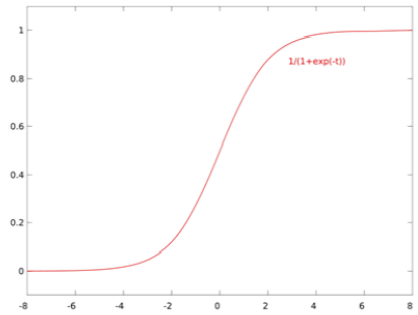
- Parallel
- Der Trainingsalgorithmus ist einfach
- Kann Entscheidungsgrenzen mit komplizierten Formen implementieren.
- Eine sehr schnelle Hardware-Implementierung ist möglich.

Nachteile der Neuronalen Netze

- Oft schwierig oder langwierig zu trainieren.
- Für das Training sind viele Parameter zu wählen (Anzahl verborgene Einheiten, Lernrate, etc.)
- Ein Netz mit vielen verborgenen Einheiten overfittet oft das Trainingsset.
  => Das Trainingsset soll sehr groß sein.

- „Black Box“ System – Es ist nicht leicht herauszufinden wie ein Netz ein Problem repräsentiert hat.

Praktische Techniken für Backpropagation
- Aktivierungsfunktionen
    Eigenschaften der Aktivierungsfunktion
        - Nonlinear
        - Continuirlich
        - Differenzierbar
        - Monoton
        - Saturated
        - → Sigmoid funktion



Skalierung der Eingaben
• Merkmale in verschiedene Einheiten, z.B. Mass in Meter (*m*) und Länge in Millimeter (mm)
• NN werden die Merkmale mit größeren Einheiten bevorzugen
• Wir müssen die Merkmale standardisieren
    - Jeder merkmal soll die selbe Varianz haben z.B. 1.0

Anzahl der verborgene Einheiten
n ist die Anzahl der Trainingmuster. Die Anzahl der vorborgenen Einheiten soll ungefähr *n/10* betragen. Mittels Test und Trainingset vermittelt man die beste Anzahl der verborgenen Einheiten:
    1. Mit eine grosse Anzahl anfangen
    2. Verborgene Einheiten entfernen – pruning
    3. Test und Training error messen

Initializierung des Gewichtsvektors
- Gewichtsvektor soll nicht mit 0 initializiert
    – BP kann keine update führen
- Zufällige Werte von eine uniformen Distribution

$$-\frac{1}{\sqrt{n_h}} < w_{ij} < \frac{1}{\sqrt{n_h}}$$

Lernrate
Kleine Lernraten führen zur Konvergenz. Im Allgemeinen wird der NN aber nicht bis zur Konvergenz trainiert. Der Criterion Funktion (Error) wird durch eine quadratische Funktion modelliert. Für viele Probleme $\eta = o.1$.

Momentum
Error manifold haben viele plateaus
- weil es zu viele Gewichten gibt

Der Gewichtsvektor wird upgedated mit momentum
- Ein momentum Parameter muss definiert sein
- $\alpha = 0.9$

Online, Batch oder Stochastic training?
- Online training
  - Data set zu gross, memory zu teuer
- Stochastic learning
  - Trainig set mit sehr viel redundanz
- Batch training
  - Training set mir sehr wenig redundanz

Die Anzahl der verborgene Layers
Drei Schichten sind in meisten Fällen genug. Vier oder mehr nur in bestimmten Fällen
- Z.B. Translations-, Rotations- und Skalierungsinvarianz in Bildverarbeitung

Praxis zeigt dass NN mit mehr als einer verborgenen Schicht sehr oft in lokalen Minimas stecken.

Zusammenfassung
Das Perceptron kann nur linear separierbare Mengen klassifizieren. Ein Backpropagation Netz besteht aus mindestens drei Schichten. Das Netz Wird mittels des Backpropagation Algorithmus trainiert und kann Entscheidungsgrenzen mit komplizierten Formen repräsentieren.

Was ist eine Lineare Maschine
Eine Lineare Maschinen ist beispielsweise eine lineare Diskriminantenfunktion. Sie bildet einen linenaren Klassifikator. Sie wird z.B. in mehrlagigen künstlichen Neuronalen Netzen in den verborgenen Einheiten verwendet.

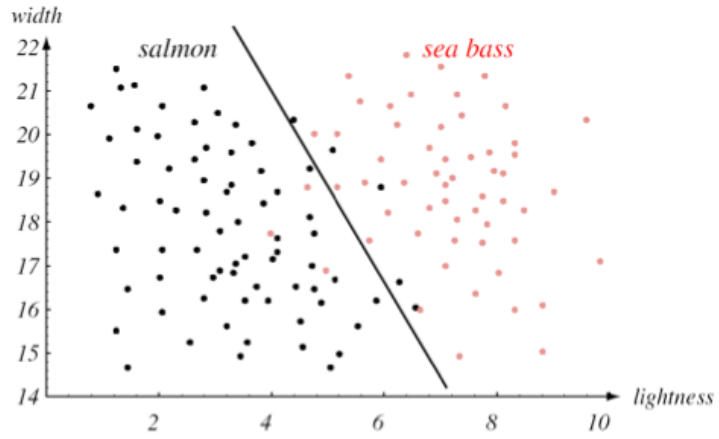
10. Bias, Varianz und Generalisierungsfähigkeit

Bias und Varianz
Bias und Varianz geben die Güte eines Klassifikators an. Ist der Bias hoch wurde der Klassifikator zu einfach gewählt. Ist die Varianz hoch, wurde der Klassifikator zu komplex gewählt.
Es gibt keinen universalen besten Klassifikator.
Zwei Kriterien für die Güte eines Klassifikators werden besprochen:

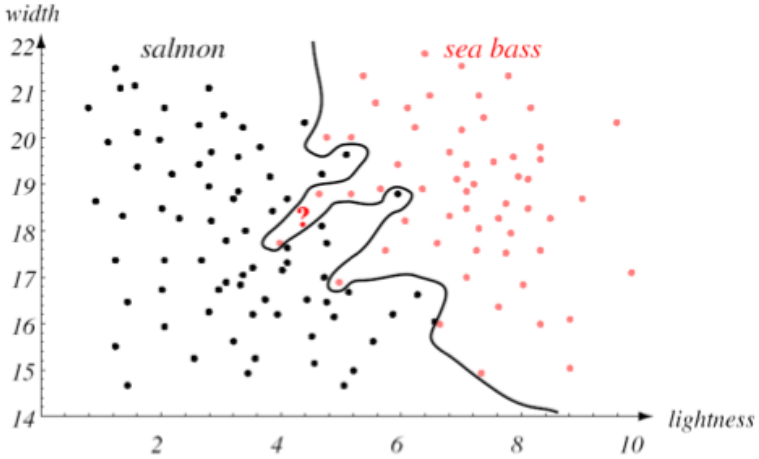
- Bias
Hoch für einen Klassifikator, der zu einfach oder unflexibel ist.
-Varianz
Hoch für einen Klassifikator, der zu viel Flexibilität in Bezug auf das jeweilige Datenset hat.

Die zwei Begriffe sind nicht unabhängig voneinander.

Bias hoch (underfitting):

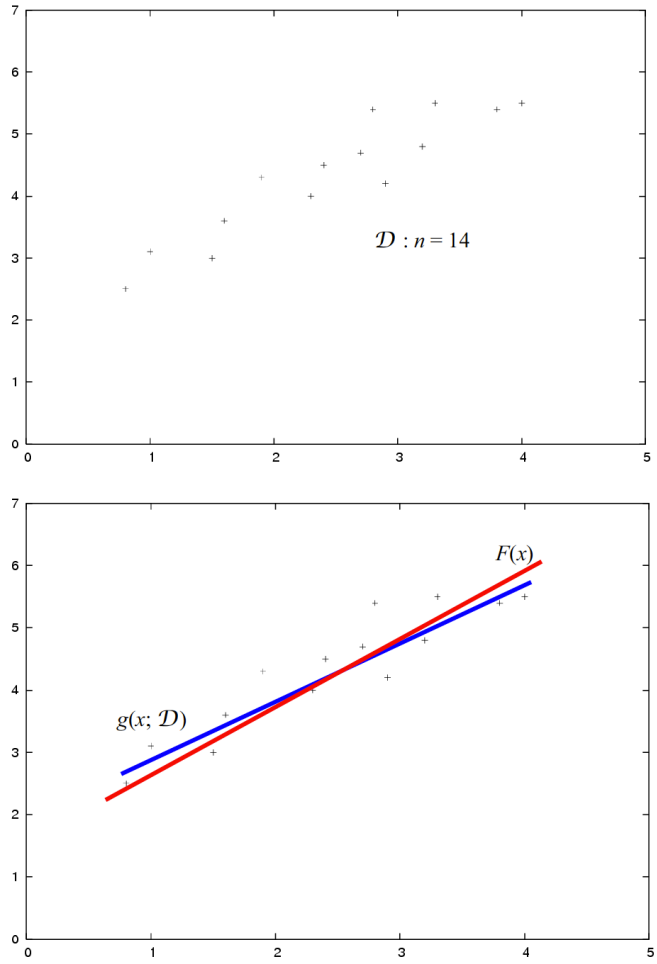


Varianz hoch - schlechte Generalisierungsfähig (overfitting):



Bias und Varianz für Regression
Es gibt eine richtige (aber unbekannte) Funktion F(x).
Wir möchten F(x) anhand einer Menge D von n samples schätzen.
Die geschätzte Regressionsfunktion wird durch g(x; ‚D') angegeben.
Wir interessieren uns dafür, wie g(x; D) von dem Trainingsset D abhängig ist.

- Es kann sein, dass für manche Trainingssets die Schätzung sehr gut ist, aber für andere, mit der gleichen Anzahl an Trainingsvektoren, schlecht ist.

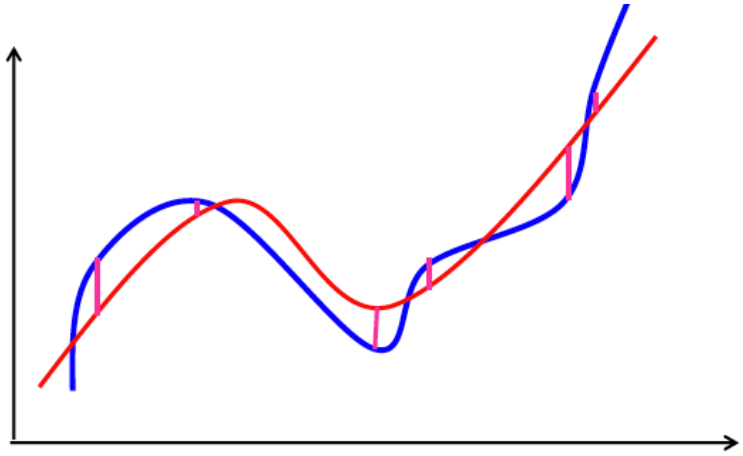


Um die Effektivität einer Schätzfunktion zu messen, wird die mean squared Abweichung von der Optimalen Funktion verwendet:

$$\varepsilon_D \left\{ \left[g(x;D) - F(x) \right]^2 \right\}$$

ε ... Erwartung
D ... Über alle Mengen D mit fixierten Größen N

$\left[g(x;D) - F(x) \right]^2$ ... Die mean squared Abweichung



Diese Abweichung kann in zwei Teile aufgespaltet werden:

$$\mathcal{E}_{\mathcal{D}}\{\,[\,g(\mathbf{x};\mathcal{D}) - F(\mathbf{x})\,]^2\,\}$$

$$= \underbrace{\{\mathcal{E}_{\mathcal{D}}[\,g(\mathbf{x};\mathcal{D})\,] - F(\mathbf{x})\,\}^2}_{\text{Bias}^2} + \underbrace{\mathcal{E}_{\mathcal{D}}\{\,(\,g(\mathbf{x};\mathcal{D}) - \mathcal{E}_{\mathcal{D}}\,[\,g(\mathbf{x};\mathcal{D})\,]\,)^2\,\}}_{\text{Varianz}}$$

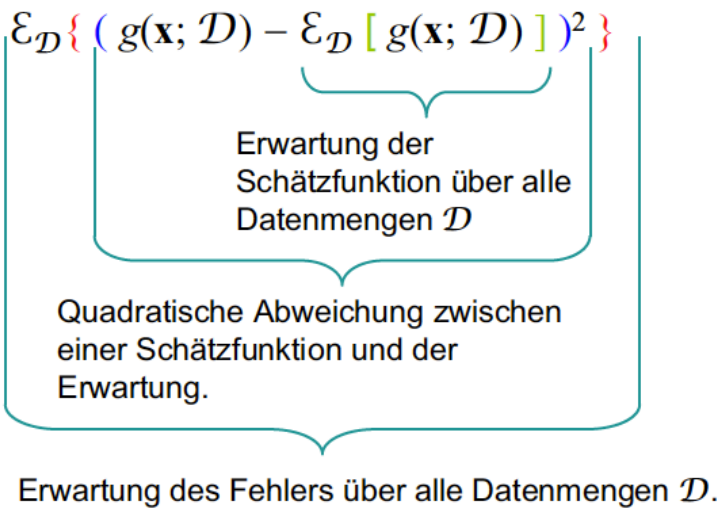
Analyse des Bias Ausdrucks

$$\{\underbrace{\mathcal{E}_{\mathcal{D}}[\,g(\mathbf{x};\mathcal{D})\,]}_{\substack{\text{Erwartung der Schätzfunktion über alle Datenmengen } \mathcal{D}}} - F(\mathbf{x})\,\}^2$$

Unterschied zwischen diesen Erwartung und der richtigen Funktion F.

Kernfrage: Wie gut passt die Schätzfunktion zu den Daten?

Analyse des Varianz Ausdrucks



Kernfrage: Wie viel verändert sich die Schätzfunktion mit verschiednen Trainingssets?

Bias und Varianz für Klassifikation

Im Fall der Klassifikation, wird der Boundary Bias verwendet.
– Zeigt wie gut die Entscheidungsgrenze die Klassen trennt.

Die Varianz zeigt wie stark sich die Entscheidungsgrenze in Abhängigkeit von der Wahl des Trainingssets verändert.

Eine kleine Varianz ist üblicherweise mit einer guten Generalisierungsfähigkeit verbunden.
Ein größeres Trainingsset führt zu
– niedriger Varianz.
– die Möglichkeit einen komplexer Klassifikator (mehr Parameter) zu wählen niedriger Bias.

Vorherige Kenntnisse: Mit Kenntnissen der Verteilung der Daten, kann ein passendes Model gewählt werden niedriger Bias.

11. Unüberwachtes Lernen und Clusteranalyse

Überwachtes Lernen

Bisher haben wir immer labelisierte Trainingsdaten verwendet. D.h., wir wissen die Klassenzugehörigkeit für jeden Merkmalsvektor im Trainingsset.

Trainingsset:

$[\mathbf{x}_1, c_1]$

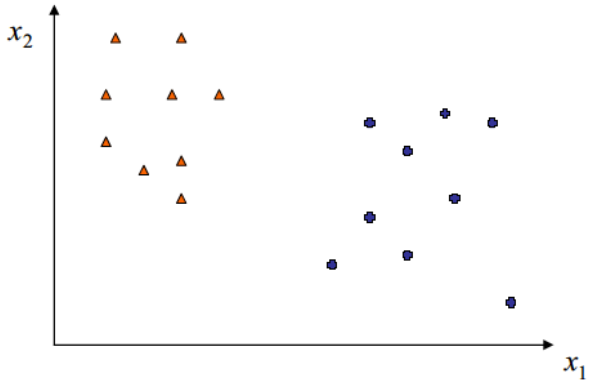
$[\mathbf{x}_2, c_2]$

$[\mathbf{x}_3, c_3]$

.

.

.



Unüberwachtes Lernen

Heute sehen wir, was man mit unlabelisierten Daten machen kann. D.h., wir haben nur die Merkmalsvektoren und wissen die Klassenzugehörigkeiten nicht.

Trainingsset:

$[\mathbf{x}_1]$

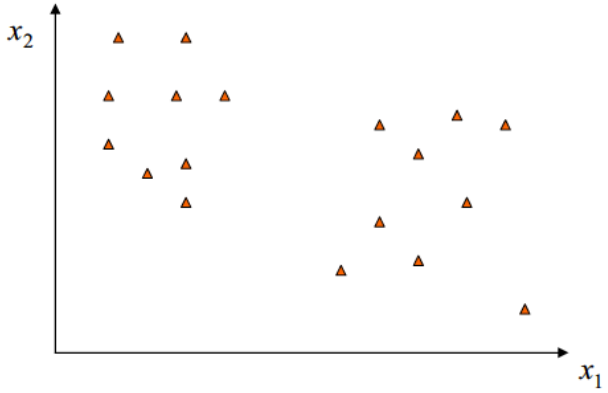
$[\mathbf{x}_2]$

$[\mathbf{x}_3]$

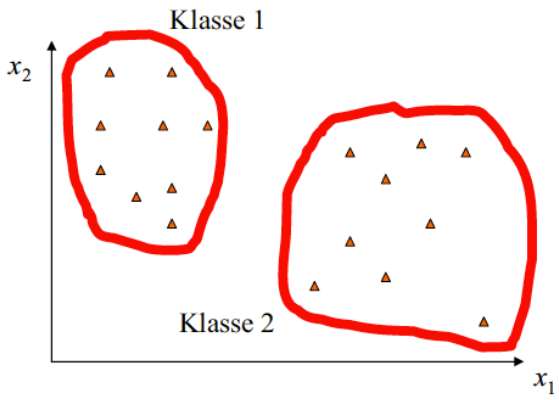
.

.

.



Mit unüberwachten Lernen, kann man z.B. versuchen, die Klassenzugehörigkeit von der Datenstruktur zu lernen. Beispiele werden aufgrund von Ähnlichkeit/Distanz zusammengefasst.



Warum ist unüberwachtes Lernen interessant?
- Die Labelisierung von Daten kann sehr teuer sein.

- In mehreren Anwendungen ändern sich die Merkmale mit der Zeit. Es ist effizienter, wenn der Klassifikator sich selbst an die Veränderungen anpassen kann.

- Man kann „Data-mining“ (heißt auch knowledge discovery) damit machen. Es wird versucht, Gruppen oder Trends in den Daten zu finden, bedeutsame Merkmale zu finden, etc.

Anwendungsbeispiele
- Datenanalyse in der Wissenschaft
    - Gruppierung von Genen und Proteinen
- Marketing
    - Produktanalyse (Gruppierung von Produkten mit ähnlichen Eigenschaften)
    - Warenkorb-Analyse (Gruppierung von Käufern mit ähnlichem Kaufverhalten)
- Datenbanken
    - Dokumentenklassifikation (thematische Gruppen)
- Astronomie
    - Sterne wurden anhand ihrer Helligkeit gegen Temperatur analysier ▸ Klassifikation von Sternen

Clustering Algorithmen

2 Arten:

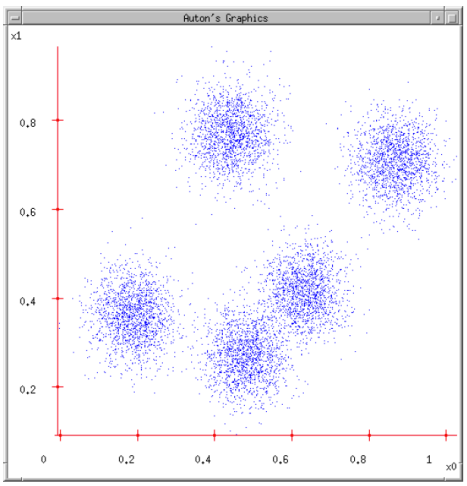
Partitionierungsverfahren (K-means)

Hierarchische Clusterverfahren (Agglomerative Clustering)

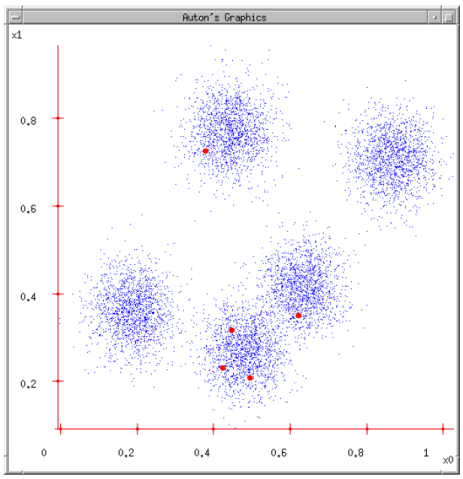
Partitionierungsverfahren
Ausgehend von k a-priori festgelegten Clustern werden diese so lange optimiert, bis die Cluster untereinander so heterogen und innerhalb so homogen wie möglich sind.

K-means ist der häufigsten verwendete Algorithmus in der Clusteranalyse.

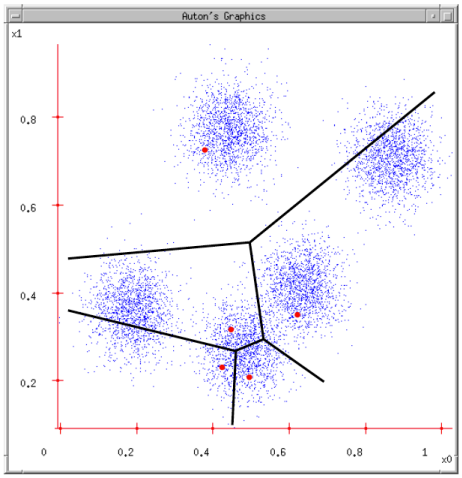
K-Means
1. Der User wählt die Anzahl von Clustern (z.B. k=5)



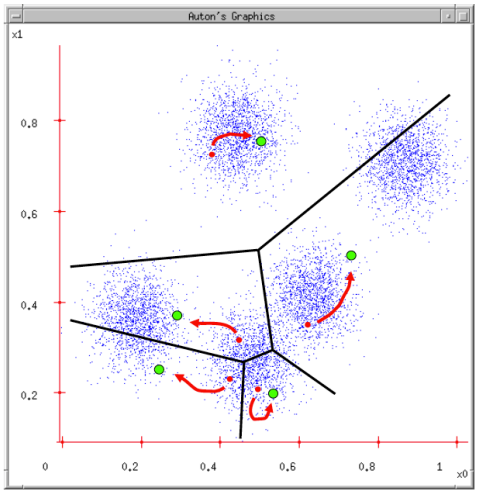
2. Positionen von k Cluster Mittelpunkten werden zufällig gewählt



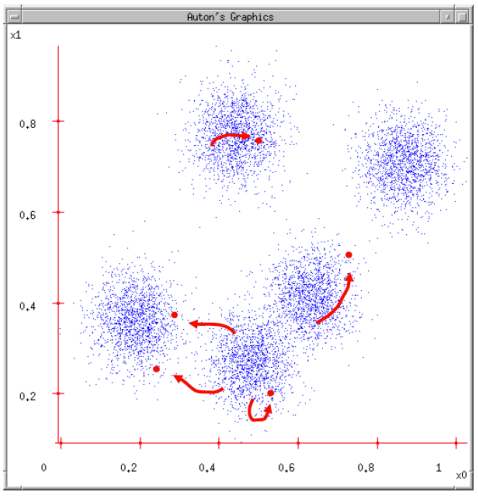
3. Jede Datenpunkt findet den nächsten Mittelpunkt



4. Jeder Mittelpunkt findet den Schwerpunkt von seinen Datenpunkten



5. ...und springt dorthin



6. ...Wiederholung bis zum Ende!

K-Means Algorithmus

1. **begin initialise** n, k, μ_1, μ_2, …, μ_k
2. **do** classify n samples according to nearest μ_i
3. recompute μ_i
4. **until** no change in μ_i
5. **return** μ_1, μ_2, …, μ_k
6. **end**

K-Means Andwendungsbeispiel: Video Google
Ein System, um Objekten oder Szenen in einem Film zu finden.

Funktion:
1. Interest points und ihre umliegenden Regionen werden detektiert.
2. Die SIFT Merkmale für jede Region werden berechnet (128-dimensinale Vector)
3. Die Merkmalsvektoren werden geclustered um „Wörter“ zu bilden (mittels K-means Clustering)
4. Diese “Wörter” können verwendet werden um die Bilder in denen sie vorkommen zu indizieren.

Unwichtige Wörter werden gefiltert (Stop-Wörter, z.B. und, der, die, das,...)

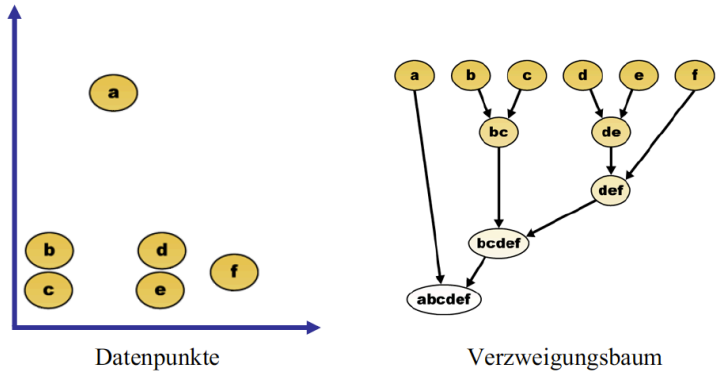
Hierarchical Clustering

Hierarchische Clusterverfahren
Hierarchische Clusterverfahren geben nicht nur Teilmengen aus, sie strukturieren die Gruppen auch hierarchisch in einem Dendrogram (Verzweigungsbaum).

Agglomerative Hierarchische Verfahren beginnen bei n Clustern (gleich die Zahl aller Einzeldaten) und fassen diese Cluster immer weiter zusammen.

Divisive Hierarchische Verfahren beginnen bei einem Cluster (der alle Einzeldaten umfasst) und teilen die Daten immer weiter zu noch heterogenen Clustern auf.

Agglomerative Hierarchical Clustering



Funktion:
1. Am Anfang ist jeder Punkt ein eigener Cluster.
2. Die ähnlichsten zwei Cluster werden gefunden.
3. Diese zwei Cluster werden in einem „Muttercluster“ fusioniert.
4. Schritte 2 and 3 werden wiederholt… bis alle Datenpunkte in einem Cluster sind.

Ähnlichkeit:
- Kleinster Abstand zwischen Punkten in Clustern (single linkage clustering).
- Größter Abstand zwischen Punkten in Clustern (complete linkage clustering).
- Durchschnittsabstand zwischen Punkten in Clustern (average linkage clustering).

Ergebnis ist ein Dendrogram (Verzweigungsbaum). Ein Vorteil ist, dass man eine Hierarchie bekommt. Um k Gruppen zu bekommen, schneidet man einfach die (k – 1) längste Verbindungen. Es gibt keine echte statistische Begründung für dieses Verfahren.

SLHC Anwendungsbeispiel: Bildsegmentierung
Verwendung von Farbenclustering um Bilder zu segmentieren.
Funktion:
1. Reduktion auf 500 Farben und von RGB in den CIELAB Farbraum wechseln.
2. SLHC wird angewendet, um diese 500 Farben zu clustern.
3. Das Clustering wird gestoppt, wenn die Euklidische Distanz zwischen den zwei am nächsten gelegenen Clustern größer als z.B. 10 ist.
4. Das Ergebnis ist eine Anzahl von Cluster.

Die Länge der Euklidischen Distanz bestimmt wie viele Cluster gebildet werden bzw. wie detailliert das Bild aufgelöst wird. Größere Distanz ist gröber ▸ weniger Cluster, weniger Detailiert.

Beschreiben Sie das Single Linkage Hierarchical Clustering Verfahren. Wie funktioniert der Algorithmus? Erklären Sie mittels einem 2-dimensionalen Beispiel.

SLHC ist ein Agglomeratives Clusteringverfahren bei dem ein Dendogramm erstellt wird.
1) Am Anfang ist jeder Punkt ein Cluster
2) Die ähnlichsten 2 Cluster werden gefunden. (kleinster Abstand - SLC, größter Abstand - Complete LC,
mittlerer Abstand - ALC)
3) Diese 2 Cluster werden in einem MutterCluster fusioniert
4) Schritte 2 und 3 werden wiederholt bis alle Datenpunkt in einem Cluster sind oder abgebrochen wird.
Das Ergebnis ist ein Dendogram (Verzweigungsbaum)

12. Entscheidungsbäume

Einleitung

Entscheidungsbäume (*Decision trees*) sind im Fall nicht-numerischer Daten anwendbar. Ein Merkmalsvektor für ein Stück Obst könnte so aussehen: (rot, glänzend, süss, klein)

Es gibt einen Stamm (root), Verzweigungen / Kanten (links / branches) und Blätter (leafs).

Vorteile von Entscheidungsbäumen
Interpretierbarkeit (einfacher zu interpretieren als z.B. Neuronale Netze):
- Weg durch Baum einfacher zu bestimmen
- Oft können Regeln für Kategorien ausgelesen (zB. Apfel: wenn *grün* und *mittelgroß*) und vereinfacht werden (*mittelgroß* und nicht *gelb*)

Schnelle Klassifizierung
A priori Wissen von Experten kann relativ einfach eingebaut werden

Einen Entscheidungsbaum züchten
Ziel: automatisch einen Entscheidungsbaum für gelabelte Trainingsdaten wachsen lassen. Der Baum teilt die Trainingsdaten in progressiv kleiner werdende Untermengen. Im idealen Fall wäre jede Untermenge rein (beinhaltet nur Beispiele von einer Klasse). Üblicherweise gibt es jedoch eine Mischung von Klassen in jeder Untermenge. Eine Entscheidung muss dann getroffen werden: Ist der Fehler klein genug, sodass die Teilung gestoppt werden kann? Oder soll eine weitere Unterteilung stattfinden?

Perfekter Entscheidungsbaum: Wenig Knoten und nicht tief.

Beispiel: Restaurant mit diversen Aufzeichnungen in Kategorien.

CART Algorithmus (Classification and Regression Tree)

Dieser Algorithmus bezeichnet eine Methode zur Erzeugung eines Entscheidungsbaums. Der Vorgang erfolgt iterativ. Daten eines Knotens sind gegeben: dieser wird dann entweder als Blatt deklariert und eine Kategorie für den Knoten ausgewählt, oder ein Merkmal wird gefunden und die Daten in weitere Untergruppen aufgeteilt.

5 Fragen zu beantworten
1. Anzahl von Verzweigungen (Wieviele *splits*?): binär oder mehrere
Wird oft als *branching factor* oder *branching ratio* bezeichnet und ist vom Designer abhängig. Jede Entscheidung kann binär dargestellt werden (einfacher zu trainieren, oft bevorzugt).

2. Welche Eigenschaft soll im Knoten geprüft werden? Query Selection
Baum soll so einfach wie möglich sein. Kriterium besteht aus einem oder mehreren Merkmalen. zB: Grün? Ja: rechts, nein: links. Patrons? None: links, some: mitte, full: rechts. $x < 10$? Ja: links, nein: rechts.

Perfektes Kriterium führt zu *reinem* Knoten (beinhalten nur Mitglieder einer Klasse), schlechtes ist *unrein* und beinhalten viele verschiedene Klassen.

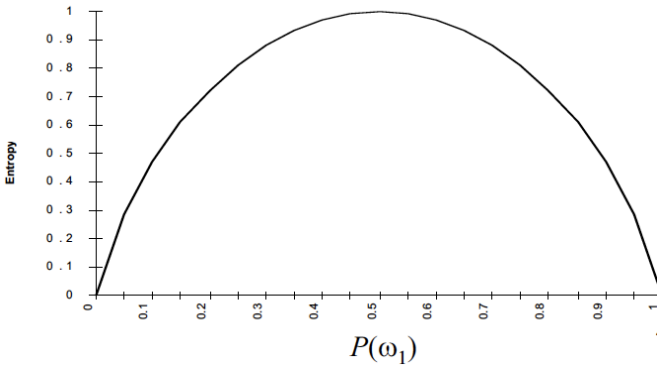
Die Unreinheit (impurity) $i(N)$ eines Knotens soll 0 sein, wenn alle Muster in N zur selben Klasse gehören. Es soll groß sein, wenn alle Klassen gleich oft vorkommen.

Populäres Maß ist die entropy impurity, wobei P(j) den Anteil der Muster in Knoten N, die zur Klasse j gehören, angibt.

$$i(N) = -\sum_j P(\omega_j)\log_2 P(\omega_j)$$

$P(\omega_j)$... Anteil der Muster in Knoten N, die zur Klasse ω_j gehören

Für 2 Klassen schaut sie so aus:



Ein Kriterium soll gefunden werden, dass die Muster eines Knotens N so teilt, dass die impurity so klein wie möglich ist. Das heißt die Änderung der impurity $\Delta i(N)$ soll maximiert werden. Berechnung für eine binäre Trennung:

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$$

N_L... resultierende Knoten links

N_R... resultierende Knoten rechts

P_L... Anteil von Mustern, die zum linken Knoten gehen

3. Stoppen des Trainings

Wann soll ein Knoten in ein Blatt umgewandelt werden?
D.h. wann soll die Trennung von Knoten gestoppt werden?
Wenn das Training erst gestoppt wird, wenn jeder Knoten rein ist, dann hat der Baum üblicherweise die Trainingsdaten overfittet.
Aber Training soll auch nicht zu früh gestoppt werden.
Entscheidungsbäume sind oft sensibel auf kleine Unterschiede in den Trainingsdaten. (Bsp.: Kanu fahren, Numerischer Baum)

Es gibt mehrere mögliche Stopping-Kriterien:

– Training mit unabhängigen Validation-Daten:

Knoten werden gesplittet bis der Fehler auf den Validation-Daten minimal ist.

– Schwellwert auf Δi :

Ein Knoten wird nicht mehr geteilt, wenn $\max_s \Delta i(s) \leq \beta$, wo *s* ein Kriterium ist. Der Schwellwert *ß* ist oft schwierig einzustellen.

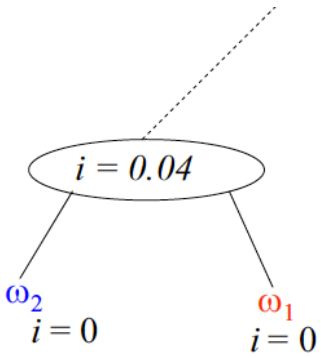
– Schwellwert auf der Anzahl von Punkten in einem Knoten:

Ein Knoten wird nicht mehr geteilt, wenn die Anzahl von Punkten weniger als ein Schwellwert ist (z.B. 10 Punkte oder 5% der Trainingdaten)

4. Pruning

Wenn Baum zu groß: wie kann er kleiner / einfacher gemacht werden? Die Entscheidung, das Training abzubrechen, wird sehr lokal getroffen (horizon effect).
Wenn das Stopping Kriterium „zu früh" erfüllt wird, wird eine nützliche Teilung weiter unten im Baum vielleicht verhindert.

Eine andere Möglichkeit ist:
– das Training durchlaufen zu lassen bis jedes Blatt rein ist (impurity = 0)
– Blätter zu fusionieren wenn die impurity nur wenig steigt (pruning).

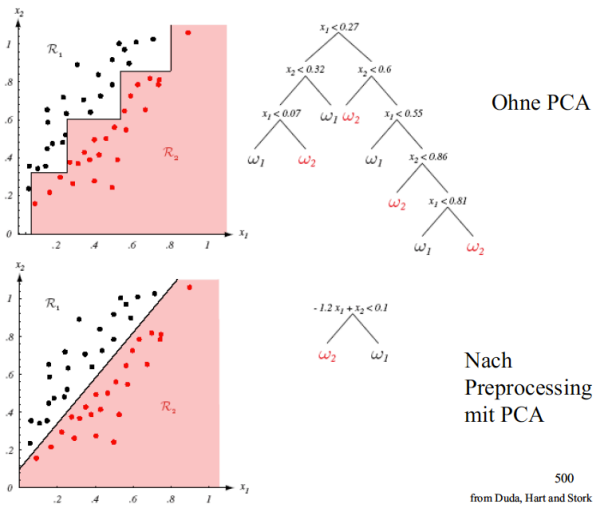


5. Blattklassen

Wenn ein Blatt nicht rein ist, wie soll eine Entscheidung über die Klasse getroffen werden?
Einfach: dem Knoten wird jene Klasse zugewiesen, die am häufigstem im Knoten vorkommt.

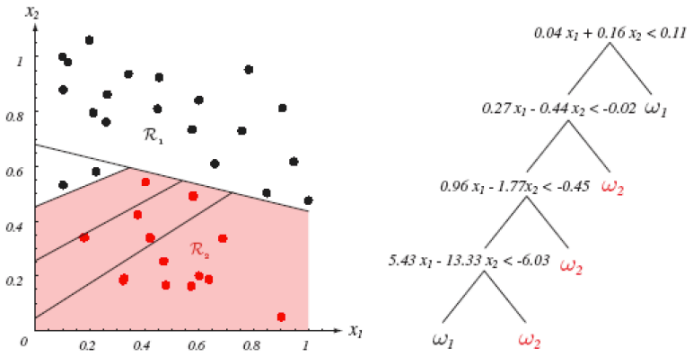
Auswahl von Merkmalen

So wie mit allen Mustererkennungs-Algorithmen, funktioniert CART besser wenn die Daten „gut" sind. PCA ist oft effektiv, weil es „wichtige" Achsen findet.



Multivariate Entscheidungsbäume

Manchmal ist es nicht ausreichend Trennungslinien zu verwenden die parallel zu den Achsen liegen.
Trennungslinien die nicht parallel sind: Z.B. ein Art Perzeptron in jedem Knoten implementieren.



Bemerkungen

Andere Baumerzeugungs-Methoden: ID3, C4.5

Entropy Impurity funktioniert akzeptabel in den meisten Fällen. Andere möglichen impurity Maße:
– Gini impurity
– Misclassification impurity

Pruning nutzt mehr von den Informationen, die im Trainingset vorhanden sind. Pruning für große Datensätze kann aber sehr rechenaufwendig sein.

13. Anwendungen

3 Anwendungen:
Biometrie in der Praxis, Automatische Chromosom-Klassifizierung, Inhaltsbasierte Bildsuche

Biometrie in der Praxis

Evaluierung der Biometrie in der Praxis
nichtssagendes Beispiel eines Labors aus England

FRR - FAR mit einem Versuch
Falsch- Rückweisungsrate (False Reject Rate, FRR):
Ein "False Reject" entsteht, wenn das biometrische System einer autorisierten Person den Zutritt / Zugriff nicht erlaubt.
Falsch Akzeptanz Rate (False Accept Rate, FAR): Ein "False Accept" entsteht, wenn das biometrische System einer nicht autorisierten Person den Zutritt / Zugriff erlaubt.

Welche 2 Arten von Identifizierung in biometrischen Systemen gibt es?
Verifizierung und Erkennung
Verifizierung
> Authentifizierung von einer Identität. “Ich bin Helmut Huber” (Richtig oder Falsch?)
> Solche Systeme bestehen aus zwei Etappen: Registrierung (Enrollment) und Verifizierung

Erkennung
> Das System stellt die Identität von einer Person fest, mit Hilfe einer Datenbank von Personen. “Wer bin ich?”

Beispiele von FAR Fehlerraten
Tabelle

Biometrie Zusammenfassung
Keine Fehlerfreie Erkennung bei Menschen. Die Anzahl der Menschen, denen der Zugang falsch verweigert wird (FRR) hängt von dem Sicherheitsgrad ab. **Spoofing** von biometrischen Systemen ist ein wichtiges Problem.

Automatische Chromosom-Klassifizierung

- Forschungsbereich seit +- 40 Jahre

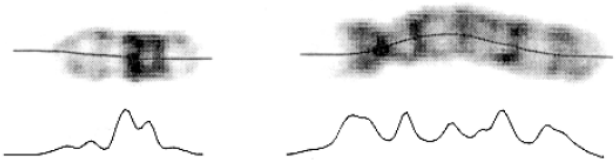
Biologische Grundlagen
menschliche Zelle: 46 Chromosomen, können eingefärbt und unter Mikroskop sichtbar gemacht werden => strukturelle und numerische Abnormalitäten können ausgemacht werden

Denver Klassen: teilen Chromosomen nach Länge und anderen sichtbaren Unterschieden ein.

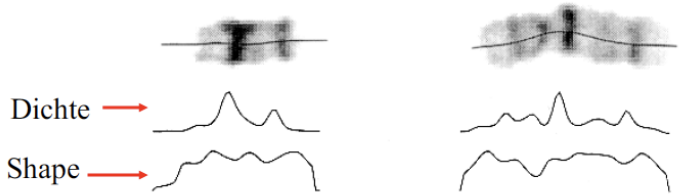
Das Problem
Durch Mustererkennung soll automatische Chromosomen-Klassifizierung möglich werden.

Merkmalsextraktion aus Chromosomen

Dichteprofil (*integrated density profile*) kann extrahiert werden.



Diverse Schritte führen zur Extraktion der Chromosomen-Achse.
Centromere wird mittels Analyse des Randes des Chromosoms sowie des Dichteprofils gefunden. (Suche mittels Dichteprofil (=> Shape profil) effektiver.)



Klassifikation der Chromosome aufgrund diverser Merkmale (Fläche, Umfang konvexe Hülle, Länge, Position des Centromere, Anzahl Bande, ...). Klassifikation entweder *conext-free* oder *context-sensitive*.

Zusammenfassung
Klassifikation schnell für trainierte Personen. Viele Merkmals- & Mustererkennungsalgorithmen werden angewandt. Automatische Klassifikation, die menschliche Arbeit ersetzt ist sehr schwer umzusetzen. Viele mögliche Lösungen.

Inhaltsbasierte Bildsuche

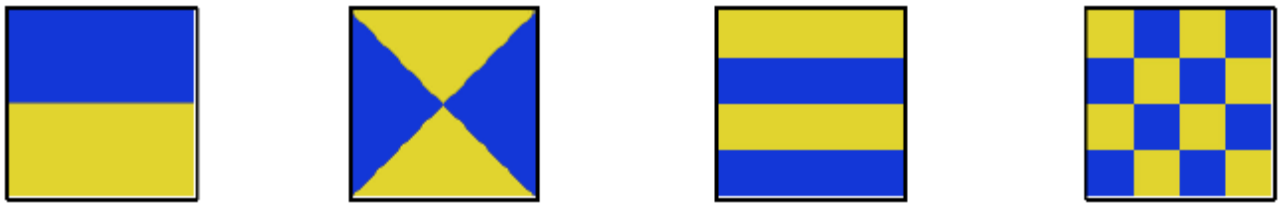
Wie gut funktioniert Google?
(Im "Skriptum" irgendwelche längst nicht mehr aktuellen Ausführungen, wie Bildsuchalgorithmen vor einiger Zeit funktioniert haben.)

Farbhistogramm
Histogramm zeigt die Farbaufteilung im Bild. Dadurch werden prominente Farbanteile leicht sichtbar. Histogramm kann bei der Bildsuche eingesetzt werden und kann zu besseren Ergebnissen führen. Aber: durchaus unterschiedliche Bilder können gleiches Histogramm haben.

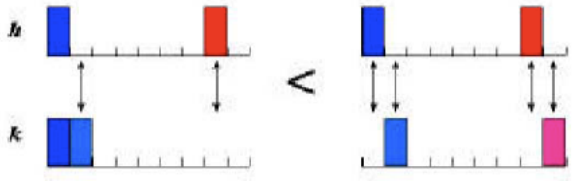
Farbquantisierung
Distanz zwischen zwei Histogrammen mit gleicher Anzahl an Bins:

$$D(H,H') = \frac{\sum_i \min(H_i, H'_i)}{\sum_i H'_i}$$

Nachteile dieses Verfahren
Durchaus unterschiedliche Bilder können gleiches Histogramm haben.



Ein kleiner Farbunterschied kann einen großen Einfluss auf die Farbdistanz haben.



Segmentierung und semantische Klassifizierung von Objekten

Aufsätze:

parametrische / nicht-parametrische Verfahren k-means
CART, 5 Fragen und kurze Antworten dazu Neural Networks Backward Propagation Perceptron
PCA in der Gesichtserkennung

Der Aufbau von Entscheidungsbäumen

Entscheidungsbäume kommen dann zu Einsatz, wenn nicht-numerische Daten vorliegen. Ein Beispiel hierfür sind Merkmale für Obst (Geschmack, Farbe, Form, Größe, usw.). Für jedes Merkmal gibt es eine Liste von Ausprägungen. Ein wesentlicher Vorteil von Entscheidungsbäumen ist die einfache Interpretierbarkeit. Man kann dem Weg durch den Baum für einen bestimmten Merkmalsvektor folgen und oft Regeln für Kategorien auslesen. Diese Regeln können auch oft vereinfacht werden. Des Weiteren ist eine schnelle Klassifizierung möglich und a priori-Wissen von Experten kann relativ einfach eingebaut werden. Beim Züchten von Entscheidungsbäumen werden die Trainingsdaten in progressiv kleiner werdende Untermengen geteilt. Im Idealfall wäre jede Untermenge rein, d.h. beinhaltet nur Beispiele einer Klasse. Üblicherweise gibt es jedoch eine Mischung von Klassen in jeder Untermenge. Dann muss entschieden werden, ob der Fehler klein genug ist (=Teilungsstopp), oder eine weiter Unterteilung erfolgen muss. Eine Methode zur Erzeugung ist *CART* (Classification *And Regression Trees).* Hier wird der Entscheidungsbaum iterativ aufgebaut. Gegeben sind die Daten von einem Knoten. Der Knoten wird entweder als Blatt deklariert und einer Kategorie zugeordnet, oder ein Merkmal wird gefunden, um die Daten weiter in Untermengen zu unterteilen.

PCA in der Gesichtserkennung

PCA steht für *Principal Component Analysis* und bedeutet, dass ein p-dimensionaler Vektor ohne Fehler durch die Summe von p linear unabhängigen Vektoren dargestellt werden kann. Eine Anwendung findet sich in der Gesichtserkennung. Diese wird als das benutzerfreundlichste Biometrieverfahren angesehen, wobei selbst Menschen nicht 100% zuverlässig sind. Oft angewendete Verfahren sind *Elastic Graph Matching* und *Eigenfaces.*
Bei den Eigenfaces werden als Trainingsdaten standardisierte Bilder von Gesichtern verwendet. Die Gesichter werden durch eine reduzierte Basis von Eigenvektoren repräsentiert (=Eigenfaces). Die Rekonstruktion braucht wesentlich weniger Speicherplatz als z.B. ein JPEG-komprimiertes Bild. Bei der Gesichtserkennung wird das Eingabebild standardisiert und durch eine Summe von bestehenden Eigenfaces repräsentiert. Die Eigenface-Koeffizienten werden mit den Koeffizienten einer Datenbank von bekannten Gesichtern verglichen, der Vektor mit den Ähnlichsten wird gefunden. Bei den *Erkennungssystemen* erfolgt die Leistungsevaluierung durch den Prozent- Anteil der Fälle, in denen die richtige Antwort in den ersten Treffern erscheint. Hier ist bei der Interpretation jedoch Vorsicht geboten, da je nach Trefferanzahl sehr unterschiedliche Prozentwerte herauskommen. Neben der Erkennung gibt es noch die *Verifikation,* bei der Identitäten bestätigt werden. Hier sind vier Leistungsmaße vorhanden: *True Acceptance, True Rejection,* sowie die Fehlermaße *False Acceptance Rate* (FAR) und *False Rejection Rate* (FRR).

Bias, Varianz und Generalisierungsfähigkeit

Es gibt keinen universalen besten Klassifikator. Es gibt jedoch zwei Kriterien für die Güte eines Klassifikators. Einerseits gibt es den *Bias,* der bei einem zu einfachen und unflexiblen Klassifikator sehr hoch ist. Andererseits gibt es die *Varianz,* die für einen Klassifikator, der zu viel Flexibilität in Bezug auf das Datenset hat, hoch ist.
[Grafik Varianz & Bias, Folie 391]

Es gibt eine richtige Funktion F(x), die aber unbekannt ist. Diese soll durch eine Menge D an n Daten geschätzt und in form der geschätzten Regressionsfunktion g(x;D) dargestellt werden. Interessant ist hierbei, wie g(x;D) vom Trainingsset abhängig ist. Je nach Set kann die Schätzung sehr gut oder sehr schlecht sein (bei gleicher Set-Größe).
[Grafik F(x) und Trainingsset, Folie 395]
Um die Effektivität einer Schätzung zu messen, wid die *Mean Squared Abweichung* von der optimalen Funktion verwendet. Im Fall der Klassifikation wird der *Boundary Bias* verwendet. Er zeigt, wie gut die Entscheidungsgrenze die Klassen trennt. Die Varianz hingegen zeigt, wie stark sich die Entscheidungsgrenze in Abhängigkeit von der Wahl des Trainingssets verändert.
Eine kleine Varianz bedeutet eine gute Generalisierungsfähigkeit. Ein größeres Trainingsset führt zu niedriger Varianz und niedrigerem Bias (komplexerer Klassifikator möglich). Auch Vorkenntnisse über die Verteilung der Daten sind gut für einen niedrigeren Bias (Wahl des passenden Modells).
(197 Wörter)

Parametrische und nicht–parametrische Lernverfahren: Algorithmen, Vorteile und Nachteile

Es gibt zwei Möglichkeiten für die Schätzung der *pdf* (Probability *Density Function – Wahrscheinlichkeitsdichtefunktion):*
Erstens gibt es die nicht-parametrischen Verfahren. Hier wird keine Annahme über die pdf gemacht. Um p(x|ωj) von nj Stichproben der Klasse ωj zu schätzen, wird zunächst eine kürzere Linie mit Zentrum am Punkt x0 erstellt. Diese wird expandiert, bis sie k Stichproben enthält, wobei k vorher festgelegt wird. Vx ist die Länge der Linie, daraus ergibt sich 1/Vx als Schätzung der Dichte. Die Dichte wird für so viele Punkte wie möglich geschätzt, weitere Punkte werden interpoliert. In höheren Dimensionen ist das Verfahren ebenso anwendbar, im zweidimensionalen Fall wird ein Kreis oder Viereck anstatt der Linie verwendet, Vx ist dann die Oberfläche. Die Form der geschätzten Dichtefunktion kann sich sehr mit dem Wert von k ändern. Die geschätzte Dichtefunktion ist normalerweise nicht glatt und nähert sich der echten Funktion, sofern n→∞ geht. Das k-Nearest Neighbour Verfahren basiert darauf, hier wird statt der Likelihood direkt die a posteriori-Wahrscheinlichkeit geschätzt.
Die zweite Möglichkeit sind die parametrischen Verfahren, bei denen zu Beginn eine Annahme über die Form der pdf gemacht wird. Hier werden die Parameter der Kurve geschätzt. Vorteile sind hier die geringe Anzahl der zu schätzenden Parameter sowie das Auskommen mit kleineren Trainingssets. Dem gegenüber steht der Nachteil, dass parametrische Verfahren nicht so flexibel sind wie nicht parametrische. Die am häufigsten verwendete Kurve ist die Normalverteilung. Gründe hierfür sind die guten analytischen Eigenschaften.

Mustererkennung in der automatischen Chromosom–Klassifizierung

Jede menschliche Zelle hat 46 Chromosomen. Während der chromosomalen Zellteilung können sie nach einer Färbung mittels eines Mikroskops gesehen werden. Mit bestimmten Färbungen sind Bande auch auf den Chromosomen sichtbar. Die Chromosomen werden anhand der Bande, Länge usw. unterschieden und in einen Karyotyp eingerichtet. Es ist dann möglich, chromosomale Abnormalitäten (strukturelle oder numerische) zu untersuchen. Das Problem bei der Mustererkennung ist, das der Aufbau eines Karyotyps so weit wie möglich automatisiert sein soll. Aus den Chromosomen werden 30 Merkmale extrahiert, dazu gehören u.a. Fläche, Umfang der konvexen Hülle, Länge, Position des Centrometers, Anzahl der Bande und Banding Merkmale.
Die Klassifikation erfolgt auf zwei Arten: unabhängig vom Kontext und kontextabhängig. Menschen verwenden kontextabhängige Informationen für die Klassifikation. Für einen Teil der 30 Merkmale ist eine parametrische Klassifikation möglich (multivariate *Gauß'sche Verteilung).*

Die Anzahl der Chromosomen pro Klasse wird (kontextabhängig) beobachtet durch einfache Austausch-Algorithmen, lineare Programmierung und genetische Algorithmen. Zusammenfassend lässt sich sagen, dass die Klassifikation von Chromosomen leicht von trainierten Menschen durchgeführt werden kann. Es wird eine Vielzahl von Merkmalsextraktions- und Mustererkennungsalgorithmen angewendet. Der Aufbau eines dem Menschen ebenbürtigen Mustererkennungssystems ist nicht einfach, wobei es viele Methoden für die Lösung des Mustererkennungsproblems gibt.