

Lösung 3: Learning Residuals

Einführung von transparenten Bypass-Connections

→ BP ist besser dann Residuals ein trainieren, also Abrechnung von Bypass.

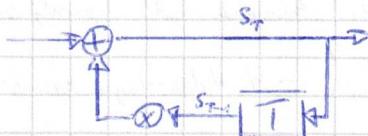
→ ohne Residuals performen sehr tiefe selbst auf Trainingsdaten schlechter als flachere NN. Mit Residuals sind auch sehr tiefe Netze trainierbar (652 im Papier)

Lösung 4: Long-Short-Term-Memory (LSTM) Recurrent NNs

Einführung des Constant Error Cancell (CEC)

hinzufüllt zu CEC:

- Multiplikatives Input-Gate
- Multiplikatives Forget-Gate
- Multiplikatives Output-Gate



↳wendlicher Inputort → LSTM

Aufbau LSTM-Blocks S. 252

- LSTM lernen schnell, aber leider auch Fehler im Trainingsset ⇒ schlecht im Generalisieren

Unsupervised Learning and Clustering

Unlabeled Data oft in großen Mengen vorhanden, Labeled Data ist teuer

- RBM pre-training
- autoencoder
- Clustering

Abbildungsmenge

- Euklidische Distanz

- Kostefunktion MSE: $J = \sum_{i=1}^n \sum_{x \in C_i} \|x - \mu_i\|^2$

k-Means

- Platziere μ der Cluster zufällig

- Setze μ neu

→ ordne Datenpunkte den μ zu

→ wiederhole, bis μ nicht mehr ändert

Hierarchisches Clustering

Agglomerative Approach: start: - jeder Datenpunkt ist eigenes Cluster

- kombiniere immer Cluster mit niedrigster Distanz, bis Anzahl an Clustern erreicht

Metriken: Minimum of sample distances

$$D(C_i, C_j) = \min_{x \in C_i, x' \in C_j} \|x - x'\|$$

Minimum of cluster means,

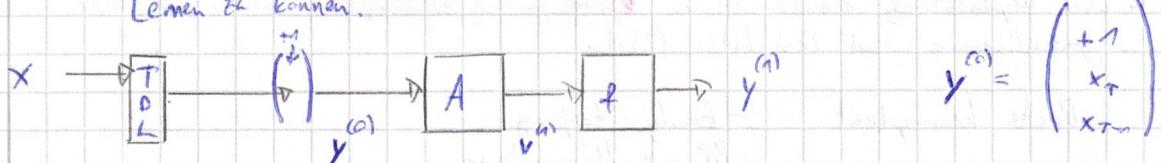
$$D(C_i, C_j) = \min_{\mu_i, \mu_j} \|\mu_i - \mu_j\|$$

Minimum of cluster sample dists

$$D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{x' \in C_j} \|x - x'\|$$

Neuronale Netze mit Memorg

Idee: Fügt einen tapped delay line (TDL) um zeitliche Zusammenhänge lernen zu können.



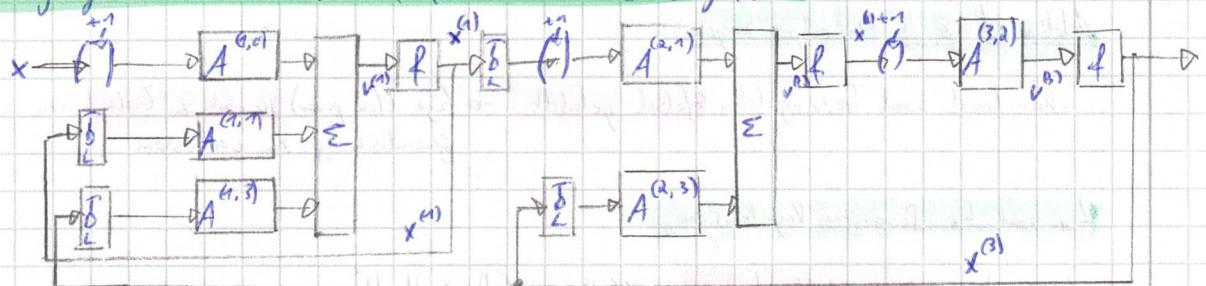
Kenn zusammen mit Feedforward Netz zeitintensive short-term Strukturen lernen.

Spezialfälle: $N=1$, linear Aktivierungsfnktin. \Rightarrow Feedforward: FIR-Filte

Recurrent : IIR-filter

Recurrent Neural Network (RNN)

Ausgang von Schichten wird auf früheren Schichten zurückgeführt.

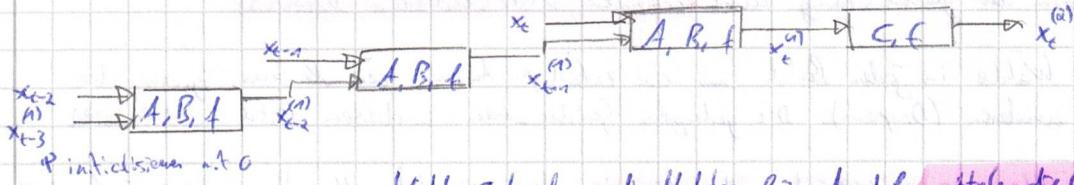


Minimales Delay für rückgeführte Inputs ist 1.

Training von RNNs:

Backpropagation through time (BPTT)

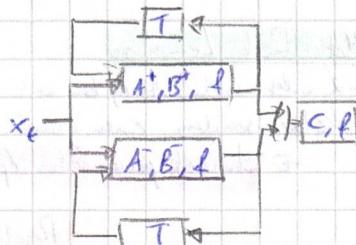
möglich mit Online-Backpropagation durch Abrufen der TDL bis zu gewisser Tiefe.



Wichtig: berechne die Updates für A und B , entziele diese Updates, und wende sie dann erst auf A und B an - ansonsten kein stochastic BP möglich/sinnvoll.

Bidirectional RNN (BRNN)

Trainieren Netze in positiver und negativer Richtung



Posteriors

Möglich, Posteriors zu erhalten, wenn: - MSE (oder NC ϵ) Kostenfunktion

- gängig Neuronen in Hidden-Layer
- unbegrenztes Training material
- Softmax im Output-Layer
- One-Hot-Training

→ Wenn Trainingsprior und echter Prior nicht übereinstimmen, dann ist es besser nach dem Training aus den Posteriors Likelihoods zu berechnen.

→ Posteriors werden in der Regel überschätzt → Overconfidence

Training:

Zum Überprüfen der Qualität während des Trainings wird ein Validierungsdaten-Set verwendet
- Stop-Kriterium darin: Kosten auf Validierungsset → early stopping

Dannen-Regel für optimale Parameteranzahl:

$$|A| \approx T/10$$

$T =$ Anzahl Trainingsdaten

$$\begin{aligned} 2-N &\rightarrow NN: \\ |A| &\approx 5 \cdot N \end{aligned}$$

Normalisierung der Input-Vektoren

Merkmal, die größere Werte annimmt als andere, würde im Training schneller in der Kostenfunktion dominieren. → Merkmalsvektoren vorher normalisieren.

→ so stellen, dass jedes Merkmal im Merkmalsvektor mittelwertsfrei ist und Varianz 1 hat.

→ Skalieren Trainings-, Validierungs- und Test-Daten mit den gleichen Faktoren

Batch Normalization

intend covariance shift verlangsamt Lernen. → Normale Merkmale nicht nur im Input-Layer, sondern auch in Hidden-Layer. Nutzen für Mini-Batches und normalisiere pro MB.

→ ermöglicht höhere Lernraten und ist zugleich Form der Regularisierung

Initialisierung der Gewichte

Mit 0 initialisieren würde das Training der Gewichte verhindern.

↪ initialisieren mit gleichverteilten Zufalls-werten → Dieser auf richtige Range achten

Momentum

wenn Kostenfunktion plateausiert ist, kann stochast. BP sehr langsam werden

→ Momentum: betrachte frühere Gradienten in Update mit ein

Regularisierung / Weight Decay

Große Gewichte sind schlecht für Generalisierung. → führt Regularisierung ein, der die Geweite nach Update wieder verringert

$$\Delta a_{jS} = a_{jS} - \mu \Delta a_{jS}$$

$$J(A) = J(A) + \frac{\epsilon}{2} a^2$$

Hervorheben von Klassen / Unbalance DataSets

In manchen Anwendungen sind manche Klassen wichtiger als andere → führt importance weight in Kosten ein

$$J(A) = \sum_{i=1}^T \sum_{j=1}^N \beta_i \cdot J_{ij}$$

Kann auch gewichtet werden um ungewolltes Trainingssets aufzublocken, wenn Prior eigentlich gleich ist

Activation functions / transfer functions (f)



Linear: $y_i := f(v_i) = v_i \in \mathbb{R}$ → Für Output-Layer von Funktions-Approximation



Sigmoid (tanh) $y_i := \tanh(v_i) = 1 - \frac{e^{-v_i}}{1+e^{-v_i}} \in [-1; 1]$ klassische Wahl für hidden-Layer



Sigmoid (logistic) $y_i := \frac{1}{1+e^{-v_i}} = \frac{1}{2}(1 + \tanh(\frac{v_i}{2})) \in [0; 1]$



ReLU $y_i := \max(v_i, 0)$ erhält schnelle Konvergenz und schnelle Berechnung → DeepLearning
(rectified linear unit)



Softmax: $y_i := \frac{e^{v_i}}{\sum_{j=1}^n e^{v_j}} \in [0; 1]$ → Posterior estimation

Kolmogorov's Theorem: Jede kontinuierliche Funktion $g(x)$ mit $x \in [0; 1]^n$ kann mit 2-Layer-NN realisiert werden

Probleme: nicht bekannt, welche Aktivierungsfunktionen und wieviele Nodes in Layer notwendig

Backpropagation

Idee: Gebe Trainingsdaten $y^{(0)}$ in W und erhalte Outputs $y^{(1)}$. Vergleiche diese mit gewünschten Outputs z . Verändere Gewichte A um Fehler zu minimieren.

2-lag.-NN:

$$(x) \quad y_j^{(1)} = f\left(\underbrace{\sum_{d=0}^D a_{j,d}^{(1)} g_d}_{v_j^{(1)}}\right)$$

$$(xx) \quad y_i^{(2)} = f\left(\underbrace{\sum_{j=0}^n a_{i,j}^{(2)} \cdot y_j^{(1)}}_{v_i^{(2)}}\right) = g_i(x)$$

Kosten-Funktion: $J(A) = \frac{1}{2} \sum_{i=1}^n (z_i - y_i^{(2)})^2 = \frac{1}{2} \|z - y^{(2)}\|^2$

„Mean square error“ (MSE)

$$A = \{A^{(1)}, A^{(2)}\} \quad \text{Input-Layer: } S \quad \text{Hidden-Layer: } j \quad \text{Output-Layer: } i.$$

Initialisiere mit zufälligen Gewichten $a(k=0)$. Adaptiere diese Gewichte dann wie bei gradient descent learning in Richtung des negativen Gradienten.

! $\Delta a = -\mu \frac{\partial J(A)}{\partial a} \rightarrow \text{Dann } a(k+1) = a(k) + \Delta a(k)$

Output-Schicht: $\frac{\partial J}{\partial a_{i,j}} = \frac{\partial J}{\partial v_i} \frac{\partial v_i}{\partial a_{i,j}} = -\Delta_i \cdot \frac{\partial v_i}{\partial a_{i,j}}$

Sensitivität: $\Delta_i = -\frac{\partial J}{\partial v_i} = -\frac{\partial J}{\partial y_i} \cdot \frac{\partial y_i}{\partial v_i} = (z_i - y_i) \cdot f'(v_i)$

Lernrate • Fehler • Ableitung d. Aktivierung • Input

$$\frac{\partial v_i}{\partial a_{i,j}} = g_j$$

→ $\Delta a_{i,j} = \mu \cdot \Delta_i \cdot g_j = \mu \cdot (z_i - y_i) \cdot f'(v_i) \cdot g_j$

Wenn Lagrange-Multiplikatoren dann bekannt sind, gilt:

SVM-Klassifikation:

$$g(x) = w^T \phi(x) + b = \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x, x_j) + b$$

$$\xi = \frac{1}{|\mathcal{I}_{\text{sup}}|} \sum_{t \in \mathcal{I}_{\text{sup}}} \left(z_t - \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x_t, x_j) \right)$$

Vorteil der SVMs: Zur Klassifikation werden nur die Support-Vektoren benötigt

Merkmalstransformation $\phi(x)$ mappt Datenvektoren in einen hochdimensionalen Merkmalsraum \mathcal{H} , Hilbert-Space genannt.

$\phi(x)$ interessiert meist nicht, sondern kernel.

Beispiele: Linear: $k(x, x') = x^T x'$ Polynomial: $k(x, x') = (x^T x' + r)^p$

Radial Basis Function: $k(x, x') = e^{-\gamma \|x-x'\|^2}$ \rightarrow Gauß-Kernell: $\gamma = \frac{1}{2\sigma^2} > 0$
(RBF)

Sigmoidal: $k(x, x') = \tanh(\gamma x^T x' + r)$

Eigenschaften der SVMs:
- Optimierungsproblem ist konkav \Rightarrow jedes lokale Optimum ist auch global
- durch nicht-linearen Kernel sind auch linear nicht-trennbare Daten trennbar

Relaxed Case

Auch Fehlklassifizierungen erlaubt \rightarrow Slack-Variablen $\xi_t \geq 0 \quad t \in \mathcal{T}$

$\xi_t = 0 \rightarrow$ auf richtiger Seite der Margin befindet

$0 < \xi_t < 1 \rightarrow$ zwischen margin befindet und decision boundary

$\xi_t \geq 1 \rightarrow$ hinter der Decision boundary (Fehlklassifiziert)

SVMs sind alle Vektoren mit $\xi_t \geq 0$ und $\gamma > 0$

\rightarrow neue Lagrange-Multiplikatoren: $\mu_t \cdot \xi_t = 0$ und $0 \leq \mu_t \leq C$

SVM-Klassifikation (soft margin)

$C \hat{=} \text{Kostentoller}$

! $g(x) = \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x, x_j) + b$

neue Constraint: $0 \leq \lambda_t \leq C$

Mehrklassen-Probleme:

SVM eigentlich nur für 2-Klassen-Probleme \rightarrow Training N-Klassif. "One-versus-the-

Normalisierung
- Trainings- (und Test-) Daten sollten normalisiert werden, damit einzelne große Vektoren keinen zu großen Einfluss haben

- wenn eine Klasse deutlich mehr Trainingsdaten hat \rightarrow Klassa-verschiedene Kostenfunktionen

MMSE by LMS

$$\nabla J_s(a) = \sum_{t=1}^T (a^T y_t - b_t) y_t \quad \rightarrow \text{(ohne Summe) in Batch Perceptron einsetzen}$$

→ abfallendes μ ist ein Muss

- glatte Kettfunktion

Geh so über alle Trainingsvektoren

Mehrklassen: - Batch-Perceptron anwendbar

MMSE müssen angepasst werden.

$$g_i(x) = a_i^T y = 1 \quad \text{für alle } y \in \mathcal{Y}; \quad i \in S = \{1, 2, \dots, N\}$$

$$g_i(x) = a_i^T y = 0 \quad \text{für alle } y \notin \mathcal{Y}_i$$

$$s^* = \arg \max_{i \in S} g_i(1)$$

$$(y_1, y_2, \dots, y_N) = (1, 0, \dots, 0)$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

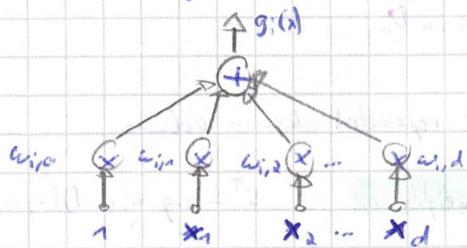
$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

$$y_1 = 1, y_2 = 0, \dots, y_N = 0$$

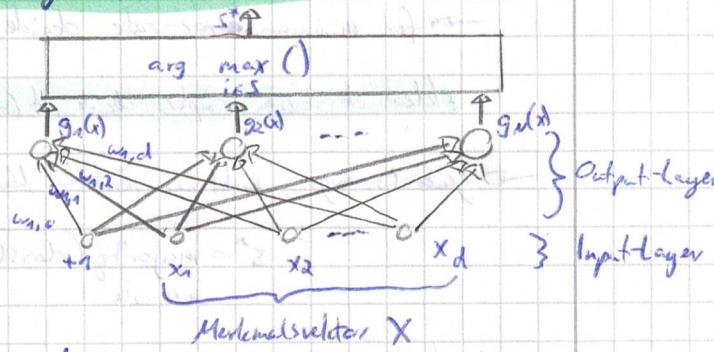
5. da: schätzen der Likelihood-Koeffizienten von Trennfunktionen zu berechnen.

Ziel: direkt Parameter einer linearen Trennfunktion finden

$$g_i(x) = w_i^T x + w_{i,0} = \sum_{j=1}^d w_{i,j} x_j + w_{i,0} \quad w_{i,0} = b_i \triangleq \text{Bias}$$



Single-layer neural network



2-Klassen-Fall:

Legt Hypothese $g(x)=0$ in Raum. Entscheidende für Klasse, je nach Seite der Ebene, also $g(x)>0$ oder $g(x)<0$

Mehrklassen-Fall:

wenn man es in mehrere 2-Klassprobleme zerlegt \rightarrow mehrdeutige Regeln

Es nicht machen!

Statt Klassen: $s^* = \arg \max_{i \in S} g_i(x)$

\rightarrow Simply connected convex decision regions

Bias $w_{i,0}$ oft mit in den Gewichtsvektor:

$$g_i(x) = w_i^T x + w_{i,0} = a_i^T y$$

$$y = \begin{pmatrix} x \\ 1 \end{pmatrix}$$

Lineare Trennbarkeit besteht, wenn es einen Gewichtsvektor a gibt, der alle Samples korrekt klassifiziert

Einfacher Trick: Mappe alle Vektoren $y_2 \rightarrow -y_2 = y'_2$ und $y_1 \rightarrow y'_1$. Suche nun Vektor a , $M=2$

der für alle y' $g(x) = a^T y' > 0$ erfüllt. Bei $g(x) = a^T y' > b > 0$

\Rightarrow Scharfe Trennlinie

Basic Gradient Descent

Idee: Definiere statische Kostenfunktion $J(a)$, die alle Trainingsdaten erfasst und minimal wird, wenn a ist ein Lösungsvektor ist.

INITIALIZE $a(0) = 0$, $\theta = \epsilon$ (small number), $\mu = 1$, $k = -1$

DO: $k = k + 1$

$$a(k+1) = a(k) - \mu \cdot \nabla J(a(k))$$

UNTIL $\|a(k+1) - a(k)\| < \theta$

RETURN $a(k+1)$

Nonparametric: wir nehmen keine Struktur für die Verteilung an

Histogramm-Methode: einfachster Weg Verteilung zu schätzen

Probleme: wieviele Bins?

Schätzen der PDF über Volumina: $p_n(x) = \frac{k_n/n}{V_n}$

k_n : number of samples in V_n
 n : total number of samples
 V_n : some interval/area/volume centered around x

V_n small: High resolution, aber auch Rauschen

V_n high: geglättete PDF

\rightarrow entweder V_n oder k_n festlegen, abhängig von n , der anderen ist dann von den Daten abhängig

1) Volumen fixed $V_n = V_1 / \sqrt{n} = h_n$ 2) # samples fixed to $k_n = \sqrt{n}$

Kernel-Based Methods (z.B. $V_n = V_1 / \sqrt{n}$)

Definieren Kernel zum Zählen der Samples im Volumen, z.B. $\varphi(u) = \sum_{i=1}^n \begin{cases} 1 & \text{if } f_{d=1..d}(x_i) \leq 0.5 \\ 0 & \text{else} \end{cases}$

$$k_n = \sum_{i=1}^n \varphi\left(\frac{x - x_i}{h_n}\right)$$

Verschiedene Kernel möglich, aber sie müssen alle selbstne PDF sein.

Kernel density estimation

$$p_n(x) = \frac{1}{n} \cdot \frac{k_n}{V_n} = \frac{1}{n} \sum_{i=1}^n \varphi_n(x - x_i)$$

$$\varphi(u) \geq 0$$

$$\int \varphi(u) du = 1$$

auf jeden muss Kernen mit n abnehmen, ob nicht

zu schnell, dann: $\lim_{n \rightarrow \infty} V_n = 0$ und $\lim_{n \rightarrow \infty} n \cdot V_n = \infty$

Mögliche Kernel: - Uniform Kernel

$$\Rightarrow V_n = V_1 / \sqrt{n} \text{ oder } V_n = V_1 / n^{1/d}$$

- Multivariate Gaußkernel

$$\varphi(u) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|u\|^2}{2}}$$

- Generalized Multivariate kernel

$$\text{imang case} \Rightarrow p_n(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \varphi_n\left(\frac{x - x_i}{h_n}\right)$$

Für $n \rightarrow \infty$: $p_n(x) \approx p(x)$

k_n -nearest-Neighbour PDF-Estimation (z.B. $k_n = \sqrt{n}$)

z.B. $k_n = k_1 \cdot \sqrt{n} \rightarrow$ Volumen wird angepasst

- kleine k_n : - PDF wird „spitzig“ - Risiko von Überfitting

- große k_n : - glatte PDF - Risiko detaillierte Entscheidungsgerüste zu verlieren

$n \rightarrow \infty$: konstante PDF $p_n(x) \approx p(x)$

Konvergenz: $\lim_{n \rightarrow \infty} k_n = \infty \quad \lim_{n \rightarrow \infty} k_n/n = 0$

$$\rightarrow \text{z.B. } k_n = k_1 \cdot \sqrt{n}$$

Training eines Classifiers

Training eines Bayes-Classifiers: Schätzen der Priors und Likelihoods

Supervised: Trainingsmaterial liegt mit Labels vor \rightarrow folgende Sache erfordert alle supervised

Unsupervised: Trainingsdaten liegen ohne Label vor

Prior-Schätzung: entweder einfach oder weggelassen

Likelihood-Schätzung: - Parametrisch (wissen Form der Likelihood, z.B. Gauß)

- Nicht-parametrisch \rightarrow wir machen keine Annahmen über pdf-Form

Parametric Estimation

Parameter-Mödell Θ_i : enthält alle R zum schätzenden Parameter z.B. $p(x|\theta_i) \sim N(x; \mu_i, \Sigma_i)$

$$p(x = o_i | \Theta_i) = \prod_{t=1}^T p(x = o_{i,t} | \Theta_i) \quad \Rightarrow \Theta = \{\mu_i, \Sigma_i\}$$

Wie gut modelliert PDF mit Θ_i die Trainingsdaten O_i .

Maximum Likelihood: $\hat{\Theta}^{(ML)} = \arg \max_{\Theta} [p(\Theta | \Theta)]$

$$\hat{\Theta}^{(ML)} = \arg \max_{\Theta} [LL(\Theta)] \quad \text{mit Log-Likelihood } LL(\Theta) = \sum_{t=1}^T \log(p(x=o_{i,t} | \Theta))$$

Maximum Likelihood Parameter Estimation

Vorgehensweise: 1.) Entscheide für ein Modell, z.B. Multivariate Gauß

2.) Besorge Trainingsmaterial

3.) Suche über Parameterraum, durch Sache des maximalen LL.

\hookrightarrow z.B. durch partielle Ableitung und Nullsetzen

Beispiel Multivariate Gaußfunktion als PDF:

$$\hat{\mu} = \underbrace{\frac{1}{T} \sum_{t=1}^T o_t}_{\text{Sample mean}} \quad \hat{\Sigma} = \frac{1}{T-1} \sum_{t=1}^T \underbrace{(o_t - \hat{\mu})(o_t - \hat{\mu})^T}_{\text{Sample covariance}}$$

GMM: Parameter-Suche via Ableitung nicht lösbar \rightarrow EM-Algorithmus

Kullback Leibler (KL) divergence:

$$D(p_{\text{true}} \| p) = \int_{\mathbb{R}^d} p_{\text{true}}(x) \ln \frac{p_{\text{true}}(x)}{p(x)} dx$$

Berechnet Ähnlichkeit zwischen zwei pdfs.

2-Klassen-Probleme

Likelihood-Ratio: $LR(x) = \frac{p(x|s=1)}{p(x|s=0)} \rightarrow \Theta \Rightarrow \text{Decide for } s=1, \text{ else } s=0$

Möglichkeiten für Θ : $\Theta = 1 \rightarrow \text{Maximum Likelihood Entscheidung (ML)}$

$$\Theta = \frac{p(s=1)}{p(s=0)} \rightarrow \text{MAP}$$

$$\Theta = \frac{\lambda_{11} - \lambda_{21}}{\lambda_{21} - \lambda_{11}} \cdot \frac{P(s=1)}{P(s=0)} \rightarrow \text{Bayes risk decision}$$

Discriminant Functions / Trennfunktionen

Bayes Klassifikator mit Trennfunktionen:

$$s^* = \arg \max_{i \in S} [g_i(x)] \quad a^* = \arg \max_{i \in A} [g_i(x)]$$

Allgemein: $g_i(x) = -R(a=i|x)$

Minimierer Fehler Fall: $g_i(x) = P(s=i|x)$

Beispiele für $g_i(x)$:

$$g_i(x) = p(x|s=i) \cdot P(s=i) \quad g_i(x) = \log p(x|s=i) + \log P(s=i)$$

Mögliche Operationen:
ohne Klassifikation
zu verändern

$$g_i(x) \rightarrow \alpha g_i(x), \alpha > 0$$

$$g_i(x) \rightarrow \alpha + g_i(x), \alpha \in \mathbb{R}$$

$$g_i(x) \rightarrow f(g_i(x)) \quad \text{f(t) monoton steigende Funktion}$$

2-Klassen-Fall:

typischerweise eine einzelne Trennfunktion: $g(x) = g_1(x) - g_2(x)$

$$\rightarrow s^* = 1 \text{ if } g(x) > 0 \quad s^* = 2 \text{ if } g(x) < 0$$

$$g(x) = P(s=1|x) - P(s=0|x) \approx \log \frac{P(s=1|x)}{P(s=0|x)} + \log \frac{P(s=1)}{P(s=0)} = LLR(x) + \log \frac{P(s=1)}{P(s=0)}$$

(log-Likelihood-Ratio)

Decision boundary / Entscheidungsgrenze:

$$g(x) = 0$$

$$-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2$$

$$-\frac{1}{2} \frac{(x-\mu)^T (x-\mu)}{\Sigma}$$

Univariater Gauß:

$$p(x) = N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Multivariater Gauß:

$$p(x) = N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \cdot e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)}$$

Gaussisches Mixture Modell (GMM)

$$p(x) = \sum_{k=1}^{\infty} c_k \cdot N(x; \mu_k, \Sigma_k) \approx \sum_{k=1}^K c_k \cdot N(x; \mu_k, \Sigma_k) \quad \sum_{k=1}^K c_k = 1$$

Missing Features

Hinzunehmen über gute Features

Noisy Features

Definition: „channel model“ $p(x_{62}|x_{92})$

(Bayesian) Belief Networks

Möglichkeit, expert (prior) knowledge in Entscheidungsprozess einzubringen

→ Statt Graphstrukturen, deren μ und Σ wir oft nicht kennen, Graph, dessen Verbindungen wir kennen