1 Begriffliche Grundlagen

Deduktion (logisches Schließen): Vom Allgemeinen aufs Spezifische schließen *(von generellen Axiomen auf spezielle Theoreme schließen)*.

Induktion (Lernen durch Generalisierung von Beispielen): Vom Spezifischen aufs Allgemeine schließen *(von speziellen Beispielen auf generelle Regeln, Theorien schließen)*.

↪ Schlussfolgerung über die Validität einer Theorie basiert auf Erfahrung

↪ Theorien sind nie "wahr"oder "existent"

↪ Theorien können falsifiziert werden, aber nicht verifiziert

↪ Theorien können nur durch Beobachtungen gestützt werden

Abduktion: Schließen ausgehend von Beispielen anhand von Regeln. Von einem speziellen Beispiel wird durch Rückwärtsanwendung von bekannten Regeln versucht daraus zu schließen (kann logisch richtig zu falschen Schlüssen führen).

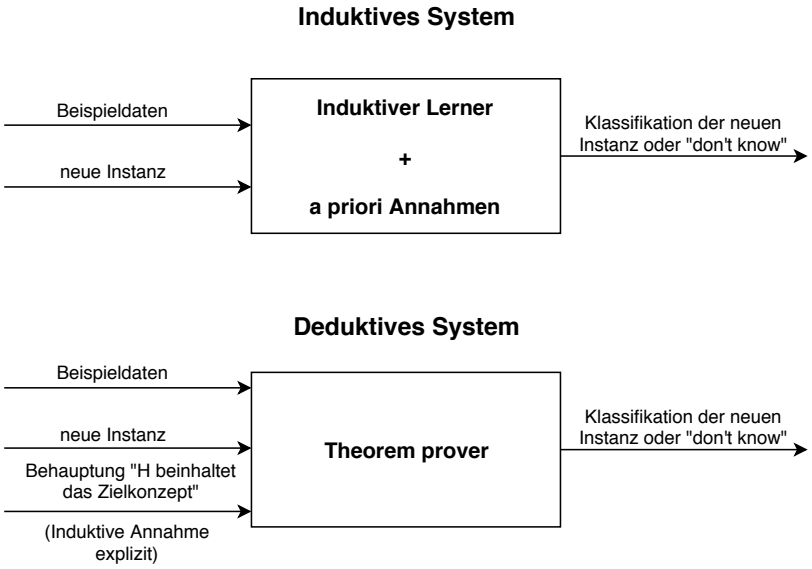

Abbildung 1: Induktives Lernen und äquivalentes deduktives System

Generalisierung (generalization): Gelerntes Modell auf neue Eingaben (die nicht zum Lernen benutzt wurden) anwenden.

Feature extraction: Eingabedaten werden Vorverarbeitet (pre-processing) damit das Problem einfacher wird (auch um Zeit-/Platzkomplexität zu reduzieren).

↪ Dimensionsreduktion

↪ Informationsverlust

Klassifikation (classification): Jede Eingabe wird einer endlichen Anzahl von diskreten Kategorien zugewiesen.

Regression (regression): Jede Eingabe wird einer oder mehreren (Vektor) stetigen Variablen zugewiesen.

Eingaben (inputs): Menge von Eingaben $X = \{x_1, ..., x_n\}$ zu einer Funktion.

Ausgaben (targets): Menge von Sollausgaben $T = \{t_1, ..., t_n\}$. zu einer Funktion.

Trainingsdaten (training set): Große Menge von Eingabedaten $X = \{x_1, ..., x_n\}$, manchmal mit den zugehörigen Sollausgaben $T = \{t_1, ..., t_n\}$.

Lineare Modelle (linear models): Funktionen, die linear in den unbekannten Parametern sind.

Inductive learning hypothesis: Jede Hypothese, die die Zielfunktion gut über eine ausreichend große Anzahl an Trainingsdaten approximiert, approximiert die Zielfunktion auch gut über neue Daten.

↪ Daten sind charakteristisch für generelle Regel/Funktion

↪ ist immer eine implizite Annahme

Learning bias: Bezeichnet die impliziten Annahmen über die Art der Generalisierung.

↪ ist immer notwendig!

Modellselektion (model selection): Auswahl einer parametrisierte Form des (Daten-) Modells.

Parameteroptimierung (parameter optimisation, learning): Optimiere (lerne) die Parameter eines ausgewählten Modells anhand der Trainingsdaten.

2 Überwachtes Lernen (supervised learning)

- gegeben eine Anzahl an Trainingsdaten (Eingaben, Sollausgaben)
- **Ziel:** finde Modell für die Daten
- **Ziel:** Generalisierung (Exploitation)
- verschieden Möglichkeiten für **Exploitation**
  - **deterministisch**: berechne neue Ausgabe für neue Eingabe
  - **probabilistisch**: erzeuge zufällige Ausgabe für neue Eingabe mit gelernter Wahrscheinlichkeitsverteilung (predictive distribution)

2.1 Regression (Deterministisch)

Datenmodell

- Polynom vom Grad M

- **Annahmen:**
  - ↪ Form des Modells: Polynom
  - ↪ Grad des Polynoms
    - ↪ bestimmt Komplexität des Modells
    - ↪ bestimmt die Anzahl der Parameter

Parameteroptimierung

- Summe der quadratischen Fehler:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2$$

- **Annahmen:**
  - ↪ quadratische Fehlerfunktion
  - ↪ "gute"Parameter für das Modell finden
- Betrachte $E(\boldsymbol{w})$ als Funktion der Parameter $\boldsymbol{w}$
- **Minimiere** quadratischen Fehler bezüglich $\boldsymbol{w}$
- wir erhalten durch Einsetzen der Trainingsdaten den **Trainingsfehler**
- Danach: teste mit Daten, die nicht zum Lernen verwendet wurden, wir erhalten den **Testfehler**
  - ↪ **Ziel:** Minimiere den Testfehler

Problem: Finden der richtigen Modellkomplexität.

- Fehlerminimierung allein funktioniert nicht
  - ↪ $E(\boldsymbol{w}) = 0$ ist bei verrauschten Daten unerwünscht
  - ↪ ist das Modell zu komplex (zu viele freie Parameter), dann kann es das Rauschen lernen $E(\boldsymbol{w}) << 1$
  - ↪ wenn Rauschen gelernt ist, dann ist die Generalisierung schlecht ⇒ **Overfitting**
- E_{RMS} zeigt Overfitting an

$$E_{RMS} = \sqrt{2E(\boldsymbol{w}^*)/N}$$

Maßnahmen gegen Overfitting:

↪ Modellselektion

↪ verändere #Daten relativ zu #Parameter

↪ stelle zusätzliche Bedingungen an die Parameter ⇒ **Regularisierung**

Regularisierung (hier):

- verbiete große Koeffizientenwerte
  ↪ beugt Oszillation vor (Bias!)
  ↪ macht die gelernte Funktion glatter (Bias!)

Regularisierung durch Bestrafung großer Koeffizienten:

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2 + \underbrace{\frac{\lambda}{2}\|\boldsymbol{w}\|^2}_{\substack{\text{Regularisierungsterm} \\ \lambda = \text{wie wichtig ist Regularisierung}}}$$

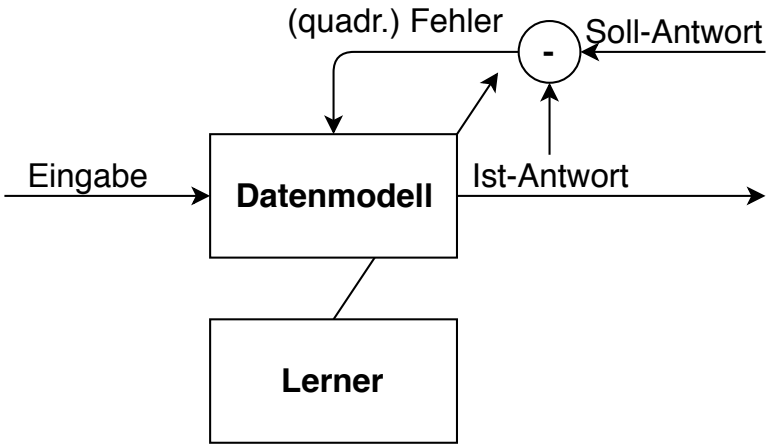

Abbildung 2: Vorgehensweise beim überwachten Lernen

2.2 Regression (Probabilistisch)

Modellierung durch Likelihood

- für gegebenes parametrisiertes Datenmodell $y(x, \boldsymbol{w})$
- **Likelihood** = bedingte Wahrscheinlichkeit eine Ausgabe t bei gegebenen Parametern $\boldsymbol{w}, \beta$ und Eingabe x_0 zu erhalten: $p(t|x_0, \boldsymbol{w}, \beta)$.
  ↪ ist nach Annahme gaußverteilt mit Mittelwert $y(x_0, \boldsymbol{w})$ und Varianz $\beta^{-1} = \sigma^2$ (1-dim. Fall):

$$p(t|x_0, \boldsymbol{w}, \beta) = N(t|y(x_0, \boldsymbol{w}), \sigma^2)$$

↪ β heißt Präzision

↪ multidimensional:

$$p(\boldsymbol{t}|\boldsymbol{x_0}, \boldsymbol{w}, \Sigma^{-1}) = N(\boldsymbol{t}|\boldsymbol{y}(\boldsymbol{x_0}, \boldsymbol{w}), \Sigma^{-1})$$

Zugrundeliegende Annahmen

- es gibt nur einen Datensatz
- es gibt einen "wahren, idealen"Satz Modellparameter
- $\boldsymbol{w_{ML}}$, β_{ML} sind gute Approximationen
- Modellierung der Unsicherheit durch Störung resultiert in Verteilung
- Daten sind endlich und zufällig
- damit sind $\boldsymbol{w_{ML}}$, β_{ML} ebenfalls datenabhängig zufällig
- Unsicherheit in der Wahl des Datensatzes ist (noch) nicht modelliert

Von der Messung zur Wahrscheinlichkeit

- für gegebenes parametrisiertes Datenmodell $y(x, \boldsymbol{w})$ und gegebene Sollausgabe t

$$t = y(x, \boldsymbol{w}) + \underbrace{N(0, \beta^{-1})}_{\text{Rauschen}} \Leftrightarrow t - y(x, \boldsymbol{w}) \sim N(0, \beta^{-1}) \Leftrightarrow t \sim N(y(x, \boldsymbol{w}), \beta^{-1})$$

- **Datenmodell:** Normalverteilung um das Modell $y(x, \boldsymbol{w})$ mit Eingabe x und Parametern $\boldsymbol{w}$

Likelihood für alle Daten (Data-Likelihood)

Annahme: Daten unabhängig von einander erzeugt.

↪ gemeinsame Verteilung = Produkt der einzelnen Likelihoods:

$$\begin{aligned} L(\boldsymbol{w}) &= P(T|X, \boldsymbol{w}) \\ &= \prod_{n=1}^{N} N(t_n|y(x_n, \boldsymbol{w}), \beta^{-1}) \\ &= \prod \frac{1}{\mathcal{N}} e^{-\frac{(t_n - y(x_n, \boldsymbol{w}))^2}{2\sigma^2}} \end{aligned}$$

Parameteroptimierung

- Data-Likelihood ist ein stochastisches Datenmodell
- $L(\boldsymbol{w})$ ist Funktion aller Parameter des Datenmodells und von β
- Parameteroptimierung = Maximierung des Likelihood
  ↪ maximiert Wahrscheinlichkeit, die gemessenen Ausgaben zu beobachten, gegeben die Modellparameter und Eingaben

Vorgehen:

- bilde $-log\ L(\boldsymbol{w})$
  ↪ Produkt wird zur Summe
- dann finde $argmin\ -log\ L(\boldsymbol{w})$
  ↪ führt wieder zur Minimierung des quadratischen Fehlers!
  ↪ wir erhalten die optimalen Parameter $\boldsymbol{w_{ML}}$, β_{ML}

$$ln\ p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \underbrace{-\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2}_{\beta E(\boldsymbol{w})} + \frac{N}{2}ln(\beta) - \frac{N}{2}ln(2\pi)$$

  ↪ Berechne $\boldsymbol{w_{ML}}$ indem $E(\boldsymbol{w})$ minimiert wird

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w_{ML}}) - t_n)^2$$

Generalisierung

- verwende die $\boldsymbol{w_{ML}}$, β_{ML} Parameter
- dann ist die optimale Output Verteilung

$$p(t|x, \boldsymbol{w_{ML}}, \beta_{ML}) = N(t|y(x, \boldsymbol{w_{ML}}), \beta_{ML}^{-1})$$

- die wahrscheinlichste Ausgabe für ein neues x ist der Mittelwert $y(x, \boldsymbol{w_{ML}})$
- aber: auch zufälliges Generieren von Ausgaben mit maximum Likelihood möglich (sampling, generatives Modell)
- β_{ML} gibt Konfidenz an

Bayes'scher Ansatz

Interpretiere Wahrscheinlichkeiten als Wissen/Unsicherheit über Parameter.

Annahme: Es gibt nur einen Datensatz, der unvollständig bekannt ist.

- verwende wieder stochastisches Datenmodell
- wenn neue/weitere Daten gemessen werden, dann ändert sich das Wissen/die Unsicherheit über die Parameter
- modelliere initiale Unsicherheit über die Parameter als $P(w)$
- $P(w)$ ist die a priori Wahrscheinlichkeit bevor die Daten beobachtet werden
- modelliere den Likelihood wie vorher
- dann berechne die a posteriori Wahrscheinlichkeit mit der Bayes Formel

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)},$$

$$\underbrace{p(\boldsymbol{w}|\boldsymbol{x},\boldsymbol{t},\alpha,\beta)}_{\text{posterior}} \propto \underbrace{p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w},\beta)}_{\text{likelihood}} \underbrace{p(\boldsymbol{w}|\alpha)}_{\text{prior}},$$

$$p(\boldsymbol{w}|\alpha) = N(\boldsymbol{w}|0,\alpha^{-1}I) = \frac{\alpha}{2\pi}^{(M+1)/2} e^{\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}}.$$

  ↪ initiale Unsicherheit abhängig vom Hyperparameter α

Der **posterior** drückt das Wissen/die Unsicherheit über die Modellparameter aus nachdem die Daten beobachtet wurden.

↪ Daten werden selbst nur unter Unsicherheit beobachtet und damit durch stochastisches Datenmodell approximiert

Parameteroptimierung

- bilde $\boldsymbol{w_{MAP}} = argmax\ P(w|D) \Leftrightarrow argmin - log\ P(w|D)$

  ↪ $\beta\tilde{E}(\boldsymbol{w}) = \underbrace{\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n,\boldsymbol{w}) - t_n)^2}_{\text{likelihood}} + \underbrace{\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}}_{\text{prior}}$

  ↪ Maximum a posteriori für gaußverteiltes stochastisches Datenmodell ist äquivalent zur Fehlerminimierung + Regularisierung

  ↪ **Overfitting** ist "automatisch"verhindert

Generalisierung

- Generalisiere durch $t_{new} = \underbrace{y(x_{new}, \boldsymbol{w_{MAP}})}_{\text{Mittelwert der posterior-Verteilung}}$
- möglich wäre auch "ziehen"eines Parameters $\boldsymbol{w'}$ aus der posterior-Verteilung: $\boldsymbol{w'} \sim p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)$ und Generalisierung durch $y(x_{new}, \boldsymbol{w'})$

Bayesian Predictive Distribution

Vollständige Berücksichtigung von Unsicherheit.

- bekannt: Data-Likelihood und Parameter posterior
- integriere über alle möglichen Parameter gewichtet mit ihrer Wahrscheinlichkeit:

$$p(t|x, \boldsymbol{x}, \boldsymbol{t}) = \int p(t|x, \boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}) dw = N(t|m(x), s^2(x))$$

- diese Wahrscheinlichkeit kann für lineare Modelle ebenfalls explizit berechnet werden und ist gaußverteilt
- dann generalisiere durch ziehen aus $\underbrace{p(t|x, \boldsymbol{x}, \boldsymbol{t})}_{\text{predictive distribution}}$

Bayesian Predictive Distribution vs. Maximum Likelihood

Der volle Bayes'sche Ansatz zeigt mehr Unsicherheit, da die Unsicherheit in den Parametern auch modelliert ist!

Inkrementelles Bayes'sches Lernen

Wenn Daten sequentiell beobachtet werden, wiederhole den Inferenzschritt durch Anwendung der Bayesregel für jeden Datenpunkt oder Datensatz $D_1, D_2, D_3, ...$:

$$\begin{aligned} P(w|D_1) &= \frac{P(D_1|w)P(w)}{P(D_1)} \\ P(w|D_2) &= \frac{P(D_2|w)P(w|D_1)}{P(D_2)} \\ P(w|D_{k+1}) &= \frac{P(D_{k+1}|w)P(w|D_k)}{P(D_{k+1})} \end{aligned}$$

Hier wird der letzte Posterior zum neuen Prior!

2.3 Lineare vs. nichtlineare Modelle

Parameteroptimierung: Lineare Datenmodelle

Lineare Modelle haben die generelle Form

$$y(x_n, \boldsymbol{w}) = \sum_{m=0}^{M} w_m \Phi_m(x_n) = \boldsymbol{w}^T \Phi(x_n)$$

wobei $\phi(x_n) = (\Phi_1(x_n), ..., \Phi_m(x_n))^T$

$\hookrightarrow$ $\Phi_m(x)$ kann eine beliebige (nichtlineare) Funktion der Eingabe sein

Beispiele:
Polynom: $\Phi_j(x) = x^j$
j-th component: $\Phi(\boldsymbol{x}) = x_j$
RBF-net: $\Phi_j(x) = e^{-\frac{\|\mu_j - x\|^2}{2\sigma^2}}$
neural net: $\Phi_j(x) = \sigma(\frac{x-\mu_j}{s})$, $\sigma(a) = \frac{1}{1+e^a}$
wenn μ_j, s_j zufällig gewählt werden: Extreme Learning Machine (ELM).

Fehlerminimierung

Maximum Likelihood

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T \Phi(x))^2$$

Minimierung:

- bilde den Gradienten
- setze = 0
- löse auf

Wir erhalten das Resultat:

$$\boldsymbol{w_{ML}} = (\Phi^T \Phi)^{-1} \Phi^T t = \Phi^{\#} \boldsymbol{t}$$

wobei $\Phi^{\#} = (\Phi^T \Phi)^{-1} \Phi^T$ die Moore-Penrose Pseudoinverse ist und
$\Phi(X) = \begin{pmatrix} \Phi_1(x_1) & ... & \Phi_M(x_1) \\ \vdots & \ddots & \vdots \\ \Phi_1(x_n) & ... & \Phi_M(x_n) \end{pmatrix} \in \mathbb{R}^{N \times M}$ ist die sogenannte Designmatrix.

Maximum A-Posteriori

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\Phi(x))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

- minimiere Fehler + Regulasrisierung
- berechne Gradienten, setze gleich null, löse

Resultat:

$$\boldsymbol{w_{MAP}} = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$$

(Erwünschter) Seiteneffekt: die Matrix wird numerisch stabiler.

Full Bayesian linear regression

Wir erhalten folgendes analytisches Resultat für die predictive distribution:

$$\begin{aligned} p(t|x,\boldsymbol{x},\boldsymbol{t}) &= \int p(t|x,\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{x},\boldsymbol{t})dw = N(t|m(x),s^2(x)) \\ m(x) &= \beta(\Phi(x)^T S\sum_{n=1}^{N}\Phi(x_n)t_n \\ s^2(x) &= \beta^{-1} + \Phi(x)^T S\Phi(x) \\ S^{-1} &= \alpha I + \beta\sum_{n=1}^{N}\Phi(x_n)\Phi(x_n)^T \end{aligned}$$

($\Phi(x_n) = (x_n^0, ..., x_n^M)^T$ für Polynommodell).

Parameteroptimierung: nichtlineare Modelle

Problem: Minimiere quadratischen Fehler.

↪ keine analytische Lösung

↪ suche Minimum iterativ

↪ meist durch Gradientenabstieg

↪ notwendig für

- direkte Minimierung des Fehlers als Funktion der Parameter

$$\boldsymbol{w^*} = argmin\ E(\boldsymbol{w})$$

- Maximum Likelihood

$$\begin{aligned} \boldsymbol{w_{ML}} &= argmax\ logP(D|\boldsymbol{w}) \\ &\sim argmin\ E(\boldsymbol{w}) \end{aligned}$$

- direkte Minimierung des Fehlers als Funktion der Parameter + Regularisierung

$$\boldsymbol{w} = argmin\ [E(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|^2]$$

- Maximum A-Posteriori

$$\begin{aligned} \boldsymbol{w_{MAP}} &= argmax\ logP(\boldsymbol{w}|D) \\ &\sim argmin\ [E(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|^2] \end{aligned}$$

Gradientenabstieg = iterative Optimierung, die lokales Minimum findet:

- definiere Startpunkt $\boldsymbol{w_0}$ (guter Tipp oder zufällig)
- iteriere: $\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - \mu\nabla E(\boldsymbol{w_k})$
- bis $\nabla E(\boldsymbol{w_k}) \approx 0$

Gradientenabstieg führt zu einer allgemeinen Lernregel mit Lernrate μ.
Wobei $E(\boldsymbol{w}) = \frac{1}{2N}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2$. Das Modell $y(x, \boldsymbol{w})$ kann eine beliebige (nichtlineare) Funktion der Parameter und des Inputs sein. **Problem:**

- Minimum hängt von Startwert ab
- zu große Schrittweite kann Minimum verfehlen/oszilieren
- zu kleine Schrittweite konvergiert langsam

⇒ adaptive Schrittweitenbestimmung.

Problem: Berechnung der Gradienten mittels Kettenregel.
↪ wenn Modell komplex, muss Kettenregel rekursiv angewandt werden

Multilayer Perceptron (MLP)

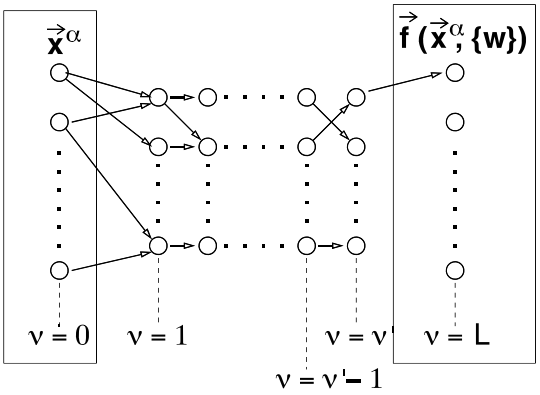

Abbildung 3: Multilayer Perceptron (MLP)

L = Anzahl der Schichten, L-1 innere Schichten ("hidden layers")

Für jeden Knoten:

$$\begin{aligned} s_i^{v+1} &= \sigma(\sum w_{ij} s_j^v) \\ \sigma(a) &= tanh(a) \text{ oder } \sigma(a) = \frac{1}{1+e^{-a}} \end{aligned}$$

Modellselektion

Kapazität

- Jede kontinuierliche Funktion kann theoretisch mit jeglicher Genauigkeit angenähert werden
- Die Anzahl notwendiger Neuronen (Knoten) ist aber unbekannt

Konkrete Modellparameter

- Anzahl der Schichten
  - ↪ eine innere Schicht ist ausreichend
  - ↪ häufig funktionieren 2 Schichten aber besser
  - ↪ modern: deep learning (viele, spezielle Schichten)
- Anzahl der Neuronen bestimmt Modellkomplexität
- Overfitting ist häufig ein Problem (Regularisierung notwendig)
- Feature extraction wichtig

Berechnung des Gradienten mittels Backpropagation

- Rekursive Formel zur Berechnung der Gradienten bzgl. der inneren Parameter w_{ij}
- Analytischer Ausdruck vorhanden (direkte Lösung möglich)
- Geringer Rechenaufwand
- **Backpropagation** = Methode zur Berechnung von Gradienten ⇒ funktioniert auch für andere Modelle

2.4 Lokale Modelle

Radial Basis Funktionen-Netz (RBF)

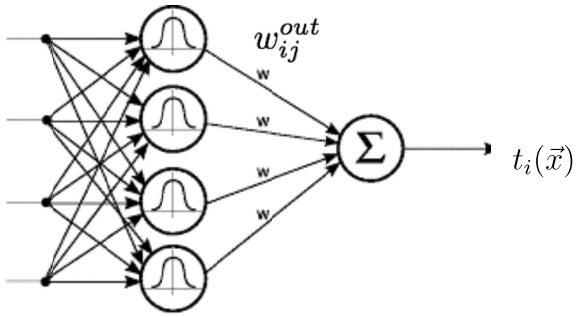

Abbildung 4: Radiales Basis Funktionen-Netz

- innere Schicht aus $m = 1...M$ radialen Basisfunktionen (RBF, Gaussfunktionen)
- lineare Kombination an der Ausgabe mit Parametern (Gewichten w_m^{out})

$$t_i(\boldsymbol{x}) = \sum_{j=1}^{J} w_{ij}^{out} \cdot \Phi_j(\boldsymbol{x}, \boldsymbol{w_j^{in}}), \text{ mit } i \in 1, ..., L, \ j \in 1, ..., J$$

- RBF hängen nur vom Abstand des inputs zum Zentrum ab:

$$\Phi_j(\boldsymbol{x}, \boldsymbol{w_j^{in}}) = R(\|\boldsymbol{x} - \boldsymbol{w_j^{in}}\|), \ j \in 1, ..., J$$

  ↪ $R()$ ist eine Gaussverteilung $R(s) = e^{-\frac{1}{2\sigma_j^2} s^2}$

  ↪ RBF: $\Phi_m(\boldsymbol{x}, \boldsymbol{w_j^{in}}) = e^{-\frac{\|\boldsymbol{x} - \boldsymbol{w_j^{in}}\|}{2\sigma_j^2}}$

RBF sind:

- universelle Approximatoren
- lineare Modelle in der Ausgabeschicht
- gut verstanden, da viele Methoden um die Parameter der Gaussfunktion zu wählen
- die RBF-Mittelwerte können auf (einige) Daten gesetzt werden
- Wahl von Sigma ("Breite") wichtig um gute Abdeckung der Daten zu erreichen

Berechnung:

- für L-dimensionale Ausgabe

$$t_i(\boldsymbol{x}) = \sum_{j=1}^{M} w_{ij} \cdot \Phi_j(\boldsymbol{x}, \boldsymbol{w_j^{in}}),\ i \in 1, ..., L,\ j \in 1, ..., M$$

- Zusammenfassung der Parameter in Vektor Θ
- und für 1-D Ausgabe: Linearkombination nichtlinearer RBF Φ

$$t(x) = \sum_{j=1}^{M} w_j \cdot \Phi_j(x, \Theta)$$

Lösung:

- für RBF wird der quadratische Fehler des linearen Modells zu

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T \Phi(x_n))^2$$

- und die Minimierung hat, wie bekannt eine analytische Lösung
  $\hookrightarrow$ Ableitung bilden, gleich Null setzen, umstellen fertig!
- wie vorher

$$\begin{aligned} \hat{\boldsymbol{w}} &= argmin\ E(\boldsymbol{w}) = \|T - \underline{\Phi} \cdot \boldsymbol{w}\| \\ &\Leftrightarrow -\underline{\Phi}^T (T - \underline{\Phi} \cdot \boldsymbol{w}) = 0 \\ &\Rightarrow \hat{\boldsymbol{w}} = (\underline{\Phi}^T \cdot \underline{\Phi})^{-1} \underline{\Phi}^T \cdot T \end{aligned}$$

- wobei T alle Daten enthält (multidimensional, dann ist T Matrix)
- $\underline{\Phi}$ ist die Design Matrix $\Phi_{ij} = \Phi_j(\boldsymbol{x_j})$

Lineare gewichtete Summe von (nichtlinearen) Gaussfunktionen ist äquivalent zu Gauss-gewichteter Summe von (konstanten) linearen Funktionen:

$$t(x) = \sum_{j=1}^{M} w_j \cdot \Phi_j(x, \Theta) \Leftrightarrow t(x) = \sum_{j=1}^{M} \Phi_j(x, \Theta)(w_j)$$

Gewichtete lineare Regression

Wir betrachten Regression nun mit zusätzlichen Gewichtungen für die einzelnen Datenpunkte.

Methode gewichteter kleinster Quadrate

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} d_n(t_n - \boldsymbol{a}^T x_n)^2$$

mit $\boldsymbol{a}$ als Parametervektor für eine Gerade

Hinweis: Im einfachen linearen Modell werden hier die Datenpunkte direkt verwendet und nicht vorverarbeitet $\Phi_m(\boldsymbol{x}) = x_m$.
Die Lösung der Normalengleichung wird dann zu

$$\boldsymbol{a^*} = (\Phi^T D\Phi)^{-1}\Phi^T D\boldsymbol{t} = (X^T DX)^{-1}X^T D\boldsymbol{t}$$

mit $D_{nn} = d_n$, wobei D eine diagonale Skalierungs-Matrix darstellt $D = \begin{pmatrix} d_1 & ... & \\ \vdots & \ddots & \vdots \\ & ... & d_n) \end{pmatrix}$.
Als Gewichtsfunktion wird die Dichtefunktion der Normalverteilung mit Mittelwert μ_1 und Kovarianz Σ genutzt:

$$d_n = \Phi(x_n, \Theta) = N(x_n|\mu_1, \Sigma) = e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}$$

damit tragen nur Datenpunkte in der Nähe des Erwartungswertes effektiv zum Fehler bei.

Lokale gewichtete Regression (LWR)

- bestimme mehrere Geraden-Segmente für verschiedene Gauß-Kurven, die jeweils mit μ_e, Σ_e parametrisiert sind

  $$\boldsymbol{a_e} = (X^T D_e X)^{-1}X^T D_e\boldsymbol{t}, \text{ für } e = 1...E$$

  mit Gaussverteilter Gewichtsfunktion mit $\Theta = (\mu_e; \Sigma_e)$

- finale Modell wird dann als Summe überlagerter, linear skalierter Gauß-Funktionen bzw. aus überlagerten Geradensegmenten gebildet:

  $$t(x) = \sum_{e=1}^{E} \Phi(x, \Theta_e)(\boldsymbol{a_e}^T x)$$

- es ist auch möglich die Geraden zu verschieben

  $$t(x) = \sum_{e=1}^{E} \Phi(x, \Theta_e)(\boldsymbol{a_e}^T x + b_e)$$

indem eine erweiterte Regressions-Matrix verwendet wird

$$\mathrm{X}=\begin{pmatrix} x_{11} & \dots & x_{1p} & 1 \\ \vdots & \ddots & \vdots & 1 \\ x_{n1} & \dots & x_{np} & 1 \end{pmatrix}, x_n \in R^p$$

Hinweis: hier ist X die (erweiterte) Designmatrix für die "innere Regression", da ja $\Phi_m(\boldsymbol{x}) = x_m$.

RBF ist Spezialfall von LWR ($E = M, b_e = w_e, a \equiv 0$).

↪ Das RBF kann als Summe sehr einfacher Modelle (einer Konstanten w_j) verstanden werden, die mit Gauss-Funktionen gewichtet sind.

Damit:

LWR: Überlagerung von Geraden, die lokal gewichtet sind.

↪ benötigt E Regressionsschritte und ist nichtlinear in den Parametern

↪ ist ein allgemeineres Modell

RBF: Überlagerung von konstanten Funktionen mit Gaußgewichten.

↪ benötigt nur einen Schritt, da linear in den Parametern

Vereinheitlichtes Modell (unified modell)

Viele Algorithmen verwenden die gleiche Modellform. Daher kann die gegebene Form

$$t(x) = \sum_{e=1}^{E} \Phi(x, \Theta_e)(\boldsymbol{a_e}^T x + b_e)$$

als vereinheitlichtes (unified) Modell verstanden werden.

Spezialfälle sind zum Beispiel:

- einfache lineare Regression
- RBF
- Gauß-gewichtete lineare Regression
- Gauß-gewichtete lokale lineare Regression
- auch probabilistische Modelle
- noch weitere Modelle

2.5 Modellierung von Unsicherheit (statistischer Ansatz)

Annahmen:

- $\boldsymbol{w}^*$ ist echter Parameter eines parametrisierten Modells
- Daten sind verrauscht $t_n = f(x_n, \boldsymbol{w}) + \nu$
- durch einen der besprochenen Algorithmen (z.B. Maximum Likelihood) erhalten wir eine Schätzung für den echten Parameter $\boldsymbol{w}^*$:

$$\hat{\boldsymbol{w}}(Z),\ Z = (X, T)$$

  - ↪ $\hat{\boldsymbol{w}}(Z)$ wird **Schätzer** genannt und ist selbst eine Zufallsvariable, weil die Daten Z verrauscht sind.
  - ↪ Versuch mit neuen Daten wiederholen um einen neuen Schätzer zu berechnen $\hat{\boldsymbol{w}}'(Z)$

Gewünschte Eigenschaften:

- ein Schätzer sollte **erwartungstreu** (unbiased) sein:
  - liefert im Mittel den "wahren"Parameter
  - d.h. der Schätzer macht keinen systematischen Fehler
  - Erinnerung: braucht die Annahme, dass es einen "wahren"Parameter $\boldsymbol{w}^*$ gibt
  - falls es einen Bias (= systematische Abweichung) gibt, kann man diesen manchmal ausrechnen und diesen dann wiederum in den Schätzer integrieren
- ein Schätzer sollte minimale Varianz haben
  - geringe Varianz erhöht die Wahrscheinlichkeit bei einer/wenigen Durchführungen des Experimentes dem wahren Wert nahe zu kommen

Aber: Es gibt nicht immer einen erwartungstreuen Schätzer.

⇒ Schätzer soll erwartungstreu und eine minimale Varianz besitzen
$< (\hat{\boldsymbol{w}} - < \hat{\boldsymbol{w}} >)^2 > \rightarrow\ min.$

↪ diese Schätzer werden als **UMV** (unbiased minimal variance) bezeichnet

Cramer-Rao Grenze

Es gibt eine untere Grenze für die Varianz von erwartungstreuen Schätzern, die mit der log-likelihood $l(\boldsymbol{w})$ verbunden ist:

$$Var(\hat{\boldsymbol{w}}) \geq \frac{1}{-<\frac{\delta^2 l(\boldsymbol{w})}{\delta \boldsymbol{w}^2}>} = \frac{1}{<(\frac{\delta l}{\delta w})^2>}$$

Interpretation: Je flacher (gemessen durch die mittlere Krümmung, bzw. Ableitung) die log-likelihood Funktion, desto größer die minimal erreichbare Varianz.
⇒ Schätzer, die diese Grenze erreichen werden **effizient** genannt.

$$- < \frac{\delta^2 l(\boldsymbol{w})}{\delta \boldsymbol{w}^2} >_{\boldsymbol{z}} \begin{cases} \text{large} \\ \text{small} \end{cases} \Leftrightarrow Var(\hat{\boldsymbol{w}}) \begin{cases} \text{small} \\ \text{large} \end{cases}$$

Bestimmung von effizienten Schätzern:

- Die Cramer-Rao Grenze ist erreicht, wenn folgendes gilt

$$Var(\hat{\boldsymbol{w}}) < (\frac{\delta l}{\delta w})^2 >= 1$$

- Es gibt eine Zerlegung der folgenden Form

$$\frac{\delta ln P(\boldsymbol{z}|\boldsymbol{w})}{\delta \boldsymbol{w}} = \lambda(\boldsymbol{w}) \cdot (\hat{\boldsymbol{w}}(\boldsymbol{z}) - \boldsymbol{w})$$

  Mit den Funktionen $\lambda(\boldsymbol{w}), \hat{\boldsymbol{w}}(\boldsymbol{z})$ und dem wahren Parameter $\boldsymbol{w}$

- $\lambda(\boldsymbol{w}) = Var(\hat{\boldsymbol{w}}(\boldsymbol{z}))^{-1}$ und $\hat{\boldsymbol{w}}(\boldsymbol{z})$ sind effiziente Schätzer

Lineare Regression zur Schätzung

Betrachte das generelle, lineare, stochastische Modell mit nichtlinearer Eingangsfunktion $\Phi(\boldsymbol{x})$

$$y = w_0 + \sum_{j=1}^{M} w_j \Phi_j(\boldsymbol{x}) + \nu$$

Wir definieren $\boldsymbol{x}$ als Datenvektor nach der Vorverarbeitung neu $\boldsymbol{x}_k = (\Phi_1(x_k), ..., \Phi_M(x_k))^T \in \mathbb{R}^M$. Mit der folgenden Notation lässt sich das Modell sehr kompakt schreiben:

$$\begin{aligned} X &= \begin{pmatrix} 1 & 1 & ... & 1 \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & ... & \boldsymbol{x}_k \end{pmatrix}^T \in \mathbb{R}^k \times \mathbb{R}^{M+1} \\ \boldsymbol{w} &= (w_0, ..., w_k)^T \in \mathbb{R}^k \\ \boldsymbol{\nu} &= (\nu_1, ..., \nu_k)^T \in \mathbb{R}^k \\ \boldsymbol{y} &= (y_1, ..., y_k)^T \in \mathbb{R}^k \\ \boldsymbol{y} &= X\boldsymbol{w} + \boldsymbol{\nu} \end{aligned}$$

Dafür erhalten wir mit der Zerlegung die folgenden Schätzer

$$\begin{aligned} \Lambda(\boldsymbol{w}) &= \frac{1}{\sigma^2} X^T X \\ \boldsymbol{w}(z) &= (X^T X)^{-1} X^T \boldsymbol{y} \\ \Rightarrow \hat{\boldsymbol{w}} &= (X^T X)^{-1} X^T \boldsymbol{y} \end{aligned}$$

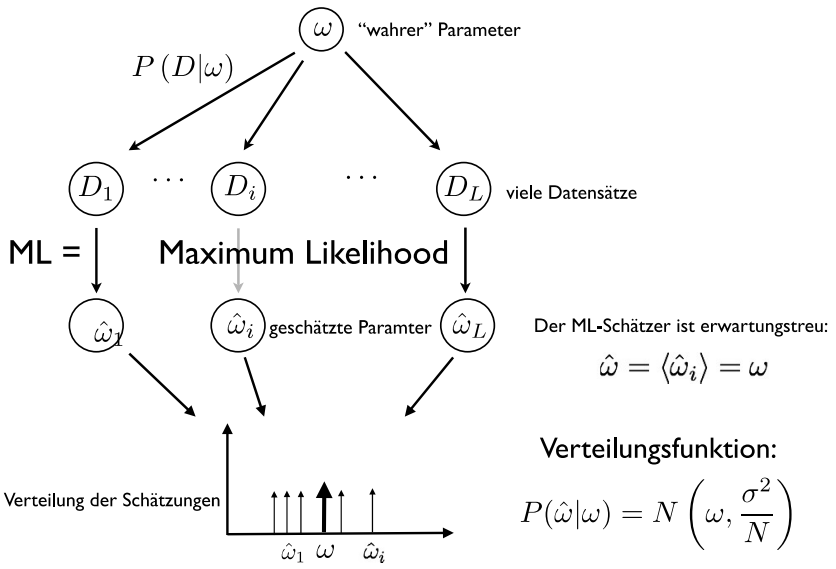

Abbildung 5: Die statistische Interpretation des Likelihood

Maximum Likelihood als Schätzer

In der Theorie liefert die Maximum Likelihood Methode UMV Schätzer, wenn ein Parameter gefunden werden kann, der die Likelihood maximiert

$$\begin{aligned} \hat{\boldsymbol{w}}(D) &= argmax\ P(D|\boldsymbol{w}) \\ P(\hat{\boldsymbol{w}}|\boldsymbol{w}) &\sim N(\boldsymbol{w}|\frac{\sigma^2}{N}) \end{aligned}$$

- der Schätzer ist dann selbst normalverteilt um den Mittelwert
- die Varianz nimmt mit zunehmender Datenmenge ab

In der Praxis wissen wir, dass wir den Fehler minimieren müssen aber häufig nur lokale Minima finden können.

2.6 Klassifikation

Aufgabe: Ordne Datum x einer von K diskreten, disjunkten Klassen zu.

↪ sofern die Klassen linear trennbar sind, können die Grenzen durch Hyperebenen dargestellt werden

Linear separierbarer Fall

Definition einer Hyperebene:

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0 \text{ (Gerade für } \boldsymbol{x} \in \mathbb{R}^2)$$

Zuordnung (hier für zwei Klassen):

Klasse 0 : $\boldsymbol{w}^T\boldsymbol{x} + w_0 < 0$,
Klasse 1: $\boldsymbol{w}^T\boldsymbol{x} + w_0 > 0$,
Grenze : $\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0$.

$K = 2$ Klassen:

Hier gilt für die Label/Sollausgaben t:

$$t \in \{-1, 1\} \text{ oder } t \in \{0, 1\}$$

$\hookrightarrow$ Die Label können im letzteren Fall als Wahrscheinlichkeit interpretiert werden

$K > 2$ Klassen:

Hier wird gewöhnlich die 1-of-K Kodierung verwendet

$$\begin{aligned} \boldsymbol{t} &= (0, ..., 0, 1, 0, ..., 0) \in \mathbb{R}^K \\ t_k &= 1 \Leftrightarrow x \in C_k \end{aligned}$$

wobei C_k die k-te Klasse bezeichnet.

Drei Ansätze:

1. Finde eine direkte Abbildung f von den Eingängen auf die Klassen.
   - f heißt Diskriminatenfunktion (**Diskriminate**)
   - Wahrscheinlichkeiten werden nicht betrachtet

   $$f(x) : x \to 0, 1$$

2. Bayes'scher Ansatz:
   a) Modellieren der Likelihood
   b) Modellierung der a-priori Wahrscheinlichkeiten für jede Klasse
   c) Berechne im nächsten Schritt den Posterior $p(C_k|x)$
3. Direkte Modellierung des Posteriors

Fisher-Diskriminante

Ansatz zur linearen Trennung (d.h. Diskriminante f ist Hyperebene)

- berechne Projektion der Daten auf eine Linie
- optimiere die Wahl der Linie um beste Trennung zu erreichen
- wähle Wert auf der Linie, zur Diskriminierung

Berechnung (zwei Klassen C_1, C_2 mit N_1, N_2 Punkten):

- Mittelwert $m_{1/2}$
- Projektion von 2D auf 1D mit linearem Modell

$$y(\boldsymbol{x} = \boldsymbol{w}^T \boldsymbol{x})$$

  $\hookrightarrow$ Projizierte Punkte liegen auf Geraden in Richtung $\boldsymbol{w}$

  $\hookrightarrow$ Projektion verläuft senkrecht

- einfacher Ansatz: Abstand der projizierten Mittelwerte maximieren (maximize inter-class variance)

$$max\ |m_1' - m_2'| \Leftrightarrow\ max\ |\boldsymbol{w}^T \cdot m_1 - \boldsymbol{w}^T \cdot m_2|$$

  $\Rightarrow \boldsymbol{w} \propto (m_2 - m_1)$ ($\boldsymbol{w}$ zeigt in Richtung $m_2 - m_1$) Beachte:
  - Richtung, Länge von $\boldsymbol{w}$ unwichtig
  - Projektionen können stark überlappen

- Besser: Zusätzlich die Varianz innerhalb der Klasse minimieren (minimize intra-class varianz)

$$s_k^2 = \sum_{x_n \in C_k} (w_n x_n - m_k')^2$$

  Dann maximiere die folgende Kostenfunktion:

$$J(\boldsymbol{w}) = \frac{(m_1' - m_2')^2}{s_1^2 + s_2^2}$$

- $J(\boldsymbol{w})$ wird maximal, wenn gilt:

$$\begin{aligned} \boldsymbol{w} &\propto S_{\boldsymbol{w}}^{-1}(m_2 - m_1) \\ S_{\boldsymbol{w}} &= \sum_{x_n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{x_n \in C_2} (x_n - m_2)(x_n - m_2^T) \end{aligned}$$

Bemerkung: Abstand zwischen Mittelwerten nicht mehr maximal. Es lässt sich zeigen, dass das selbe Ergebnis durch Minimierung des quadratischen Fehlers erzielt werden kann

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^T x_n + w_0 - t_n)^2 \text{ mit } t_n = \frac{N}{N_n}$$

Achtung: die traditionelle Bezeichnung "Fisher Diskriminante"ist etwas irreführend!

- Vektor $\boldsymbol{w}$ gibt nur Projektionsgerade an, nicht die Klassengrenze
- Diskriminierungsfunktion notwendig

Optimale Klassifikation für neuen Datenpunkt $x \in C_1$ wenn $\boldsymbol{w}^T(x - m) > 0$ und sonst in der anderen Klasse.

Bayes'scher Ansatz

- modelliere die Likelihood $p(x|C_k)$ und Prior $p(C_k)$ (hier Likelihood = class conditional)
  $\hookrightarrow$ Indirekter Ansatz um die Wahrscheinlichkeit für eine Klasse (posterior) $p(C_k|x)$ zu finden
- Bayes Theorem für 2 Klassen:

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \frac{1}{1+e^{-a}},\ a = ln\frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$$

- Generalisierung für $k > 2$ Klassen:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)} \text{ (softmax Funktion), } a_j = ln\, p(x|C_j)p(C_j)$$

- die Softmax Funktion kann als kontinuierliche differenzierbare Version des Maximums verstanden werden. Wenn $a_k >> a_j$ dann gilt: $p(C_K|x) \approx 1, p(C_j|x) \approx 0, j \neq k$

Der Bayes'sche Ansatz mit sigmoiden Funktionen führt zu einer einfachen Lösung wenn die Klassen als Normalverteilungen modelliert werden.

- Annahme: gleiche Kovarianz

$$p(x|C_k) = \frac{1}{2\pi^{\frac{D}{2}}}\frac{1}{|\Sigma|^{\frac{1}{2}}}exp(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)$$

- für den Posterior erhält man dann lineare Klassengrenzen:

$$p(C_1|x) = \sigma(ln\frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)})$$

- dies wird auch **generalized linear model** genannt, wobei die Ausgabe durch die sigmoid Funktion nichtlinear wird und als Wahrscheinlichkeit interpretierbar ist

$$\begin{aligned} p(C_1|x) &= \sigma(\boldsymbol{w}^T x + w_0) \\ \boldsymbol{w} &= \Sigma^{-1}(\mu_1 - \mu_2) \\ w_0 &= -\frac{1}{2}\mu_1\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2\Sigma^{-1}\mu_2 + ln\frac{p(C_1)}{p(C_2)} \end{aligned}$$

  Anmerkung: Wegen der Annahme gleiche Kovarianz kürzen sich bei der Berechnung die Normalisierungsterme und quadratischen x-Terme heraus

Direkter Maximum Likelihood für Labels:

- direkter Ansatz um Daten auf Labels abzubilden
- Voraussetzung: gelabelte Daten (x_n, t_n)
- Annahmen:
  - wie vorher: $p(x|C_k)$ normalverteilt
  - Prior gegeben als $\pi = p(C_1) = 1 - p(C_2)$
- Damit wird Label-Data-Likelihood zu

$$p(\boldsymbol{t}|\pi, \mu_1, \mu_2, \Sigma) = L(\pi, \mu_1, \mu_2, \Sigma) = \prod_n [\pi N(\boldsymbol{x}|\mu_1, \Sigma)]^{t_n} [(1-\pi)N(\boldsymbol{x}|\mu_2, \Sigma)]^{1-t_n}$$

- argmax log L() liefert optimale Parameter
- Likelihood zur Zuordnung von Labels auf neue Daten

Berechnung:

- zuerst nur für Terme die von π abhängen

$$argmax_\pi \sum_n (t_n ln(\pi) + (1-t_n) ln(1-\pi))$$

$$\Rightarrow \pi = \frac{1}{N}\sum_n t_n = \frac{N_1}{N} \Leftrightarrow 1 - \pi = \frac{N_2}{N}$$

- intuitiv richtig: A-priori $\sim$ Anzahl der Punkte in der jeweiligen Klasse

– Mittelwert und Kovarianz

$$\begin{aligned}\mu_1 &= \frac{1}{N_1}\sum_{n=1}^{N} t_n x_n \\ \mu_2 &= \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)x_n \\ \Sigma &= \frac{N_1}{N}s_1 + \frac{N_2}{N}s_2 \\ s_i &= \frac{1}{N_i}\sum_{x_n \in C_i}(x-\mu_i)(x-\mu_i)^T\end{aligned}$$

– intuitiv richtig: Summe der Intra-Klassen-Varianzen

Probabilistische Diskriminierung

– Posterior als Funktion von x modellieren

$$p(C_k|x) = y(x_n) = \sigma(\boldsymbol{w}^T x_n), x_n \in C_k$$

– diese Modelltypen nennen sich **logistische Regression**

– andere Herangehensweise: Vorgabe des Modells in Sigmoid Funktion

– optimiere Parameter der log-Likelihood

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_n y_n^{t_n}(1-y_n)^{(1-t_n)}$$

$$argamx_{\boldsymbol{w}} \; -log\ L(\boldsymbol{w})$$

$$= -\sum_{n=1}^{N} ln(y_n^{t_n}) + ln((1-y_n)^{(1-t_n)})$$

– Dies entspricht der Minimierung der sogenannten **cross-entropy**

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \underbrace{t_n ln(y_n)}_{0 \text{ für } t_n = 0} + \underbrace{(1-t_n)ln(1-y_n)}_{0 \text{ für } t_n = 1}$$

– $y(x)$ ist dann direkt als Wahrscheinlichkeit $p(C_1|x)$ (posterior) zu interpretieren

– Optimierung mittels **gradient descent**
  ↪ Analytische Lösung
  ↪ Erweiterung auf mehrere Klassen

2.7 Konzeptlernen

- behandelt den allgemeinen Fall nicht numerischer Daten
- Abstraktes Framework zur Betrachtung von Mengen/Teilmengen
- $X :=$ Menge von Instanzen x
- $C :=$ **Konzept** = Untermenge von X = Funktion $c : X \to \{0, 1\}$ (**Indikator**) beschreibt Untermenge
- Anzahl der möglichen Konzepte $2^{|X|}$
- Beschreibung des Konzepts = Klassifikation aller Elemente von X

Konzeptlernen: Gewinnung einer boolschen Funktion aus den Instanzen und deren Label in den Trainingsdaten.

Äquivalent: Identifizierung einer bestimmten Untermenge aus den Trainingsdaten.

Hypothesenraum

- suche im gesamten Konzeptraum zu komplex
- schränke die Suche auf weniger Funktionen/Teilmengen ein
- $H :=$ Hypothesenraum von Funktionen h

$$h : X \to \{0, 1\}$$

- äquivalent: reduziere Anzahl möglicher Untermengen
- Hypothesenraum drückt Annahme über das Problem aus (Bias)
  - ermöglicht Generalisierung
  - induktives Hypothesenlernen
- Grundannahme (**inductive learning bias**): Jede Hypothese, die die Zielfunktion auf einem genügend großen Datensatz annähert ist auch eine gute Hypothese für den gesamten Raum X

Spezialfall: Beschreibung durch Attribute

- Datenbeschreibung als Tupel diskret-wertiger Attribute
  ("Wert(Attribut 1)", "Wert(Attribut 2)", ...)
  N = #Kombinationen = #Wert(Attribut 1) $\cdot$ #Wert(Attribut 2)$\cdot$... $\to 2^N$ Konzepte

Hypothesen

- Konjunktionen von Bedingungen an die Attribute
- "constraint(Attribut 1)"∧ "constraint(Attribut 2)"∧ ...
- mögliche constraints:
  - ? = alle Werte sind erlaubt
  - $\emptyset$ = kein Wert ist erlaubt
  - v = nur der Wert v ist erlaubt

Bemerkung: Es gibt Hypothesen die semantisch äquivalent aber syntaktisch verschieden sind. Es gibt mehr Konzepte als semantisch verschiedene Hypothesen.

Allgemeinere oder genauso allgemeine (**generelle**) Hypothesen:

$$h_i \geq_g h_j$$

h_i ist genereller als h_j genau dann wenn $\forall x \in X : (h_j(x) = 1 \Rightarrow h_i(x) = 1$.

Strikt mehr generelle Hypothesen:

"ist genereller als"= "schließt die gleiche oder Obermenge ein"
$h_i \geq_g h_j$
$(h_i \geq_g h_j) \wedge (h_i \neq h_j)$
$\{x \in D | h_i(x) = 1\} \subseteq \{x \in D | h_j(x) = 1\}$

Einfachster Algorithmus für attributbezogene Daten:

Algorithm 1 Generalisierung (nur positive Daten)

```
1: INIT h with < ∅, ∅, ... >                    // Initialisierung mit spezifischer Hypothese
2: for all positive examples x do                // Iteriere über alle Trainingsdaten
3:     for all constraints a_i do                // Iteriere über alle attribute Bedingungen
4:         if a_i is fullfilled by x then        // Wenn Bedingung für Datenpunkt erfüllt ist
5:             OK
6:         else
7:             generalize a_i                    // Ansonsten wähle generellere Bedingung. Ersetze die
   Bedingung a_i mit der nächst generelleren Bedingung a_j, für die x die Bedingung
   erfüllt
```

Probleme:

- nur positive Beispiele werden verwendet
- Konvergenz kann nicht geprüft werden
- nicht robust gegen fehlerhafte Daten

Versionsraum

Eine Hypothese ist **konsistent** auf Beispielmenge $D \Leftrightarrow h(x) = c(x), \forall x \in D$. Das heißt die Hypothese klassifiziert korrekt für alle Datenpunkte in D. Die Menge aller konsistenten Hypothesen heißt **Versionsraum** $V_{H,D}$.

Der Versionsraum lässt sich durch zwei Hypothesensätze vollständig beschreiben:

$$\begin{aligned} G &:= \{g \in H | g \text{ ist konsistent und } \neg\exists g' : g' >_g g \wedge g' \text{ konsistent}\} \\ G &= \text{Satz generellster konsistenter Hypothesen} \\ S &:= \{s \in H | s \text{ ist konsistent und } \neg\exists s' : s >_g s' \wedge s' \text{ konsistent}\} \\ S &= \text{Satz spezifischster konsistenter Hypothesen} \end{aligned}$$

Damit wird der Versionsraum durch G und S begrenzt definiert:

$$V_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G) : g \geq_g h \geq_g s\}$$

Kandidateneliminierung

Eliminiere Hypothesen um Versionsraum zu erhalten. Kombiniere dazu die Ansätze Generalisierung und Spezialisierung.

Algorithm 2 Kandidateneliminierung

OUTPUT: Versionsraum

```
1: for all examples x do
2:     if example positive then
3:         generalize S
4:     else if example negative then
5:         specialize G
```

Konvergiert zu eine richtigen Konzept C, wenn $h = c \in H$ und alle Beispiele richtig klassifiziert werden.

Problem: Wenn ein Datenpunkt falsch gelabelt (inkonsistent) ist, wird die korrekte Hypothese fälschlicherweise gelöscht.

- Falls Versionsraum nur eine Hypothese enthält, verwende diese
- Falls Versionsraum noch viele Hypothesen zulässt
  - Generell mögliche Klassifikationsregel: Mehrheit + Konfidenz

Induktiver Bias

- L sei ein Konzeptlerner (Algorithmus)

- Für X gelte:

$$\begin{aligned} c &= \text{Konzept} : c : X \to \{0,1\} \\ D &= \{x, c(x)\} = \text{Trainingsdatensatz (korrekt gelabelt)} \end{aligned}$$

- Bezeichne mit $L(x), c$ die Klassifizierung von $x \in X$ nach Training D
- Dann heißt ein willkürlicher Satz von Annahmen B induktiver Bias von L wenn für c und D gilt: $\forall x' \in X$ mit $(B \wedge D \wedge x')$ kann $L(x', D)$ durch Deduktion bestimmt werden

Entscheidungsbäume

- Instanzen durch diskrete Werte und Attribute veschrieben
- Klassifikation in mehreren Schritten
  - betrachte jeweils ein Attribut einzeln
- Handhabung korrupter Daten
- Gesamter Datensatz im Wurzelknoten (root)
- Knoten zerteilen Datensatz auf Basis der Attribute
- Blatt entspricht Klasse
  - Annahme: Instanzen gehören zu Klasse (bzw. Großteil der Instanzen)
- Änderung in der Auswertungsreihenfolge resultiert in unterschiedlichen Bäumen

Fragen:

- Was ist die optimale Reihenfolge?
- Welches Attribut sollte als nächstes Ausgewertet werden?
- Was heißt optimal?

ID3

1. Berechne den Informationsgehalt für jedes übrige Attribut des Unterbaums
2. Wähle Attribut, das größten Informationsgewinn (information gain) liefert

$$S = -\sum_{i=0,1} p_i \cdot log_2\ p_i$$

Zu 1.

Shannon Information Interpretation: Wie viel Information ist noch in den Instanzen enthalten?
Frage: Welches Attribut trennt die Instanzen bestmöglich indem es die meisten richtigen ja und nein Antworten erzeugt?

Zu 2.

Informationsgewinn entspricht der Abnahme an Entropy. Hohe Entropy bedeutet, dass die Instanzen nicht gut getrennt sind. Maximal für $p = 0,5$

Shannon Entropy

Frage: Wie bestimmt man Entropy in Entscheidungsbäumen?

Für gegebenen Wert des Attributs x ist die **Shannon Entropy**

$$S_{x=k} = -\sum_{i=0,1} P_{i|x=k} \cdot log_2\ P_{i|x=k}$$

Sie beschreibt die restliche Unsicherheit für gegebenen Wert k des Attributs in dem Unterbaum.

Merke: S ist klein für gute Trennung!

Summierung über alle möglichen Attribut-Werte beschreibt die gesamte Unsicherheit wenn Attribut ausgewertet wird

$$S_x = \sum_k P_{x=k} S_{x=k} = \sum_k \frac{|S_k|}{|S|} S_{x=k}$$

Wähle das Attribut, das den Informationsgewinn maximiert

$$\begin{aligned} \Delta_x S &= S - S_x \\ \hat{x} &= argmax_x = \Delta_x S \end{aligned}$$

Anmerkungen:

- Ansatz auf > 2 Klassen übertragbar
- Erweiterte Ansätze erlauben kontinuierliche Attribute
- verbesserte Suche
- C 4.5, C 5.0
- decision forest

- Hypothesenraum des ID3 ist Menge möglicher Entscheidungsbäume
- Hypothesenraum ist vollständig:
  - enthält alle endlichen diskreten Klassifikationen
  - entspricht dem Konzeptraum
  - kein Bias durch a-priori Beschränkung der Hypothesen

Anmerkungen:

- Beginn mit "leerer"Hypothese
- nur eine einzelne Hypothese wird jeweils verfolgt
- Anzahl konsistenter Hypothesen kann nicht bestimmt werden
- robust gegen korrupte Daten

Induktiver Bias

ID3:

- unvollständige Suche
- Algorithmus schaut nicht voraus, ist "greedy"
- Bias durch Suchstrategie: **preference bias**

Candidate Elimination:

- unvollständiger Hypothesenraum
- vollständige Suche
- Bias durch begrenzten Hypothesenraum: **restriction bias**

Regression:

- Architekturdesign (**restriction bias**)
- Lern-Algorithmus (**preference bias**) (z.B. Gradientenverfahren findet nur lokale Minima)

Bayes'scher Ansatz

Verbinde Konzeptlernen und Bayes'sche Ansätze

- Nutzen von Prior, Likelihood und Posterior für Hypothesen

– Spezialfall: Diskrete Werte

$$\begin{aligned} P(D) &= \sum_h P(D|h)P(h) \\ P(h|D) &= \frac{P(D|h)P(h)}{\sum_h P(D|h)P(h)} \end{aligned}$$

– **Annahme:** Hypothesen schließen sich gegenseitig aus (sind disjunkt)

Wie vorher: Betrachtung von Maximum Likelihood oder a-posteriori

$$\begin{aligned} h_{MAP} &= argmax_h\ P(h|D) \\ &= argmax_h\ \frac{P(D|h)P(h)}{P(D)} \\ &= argmax_h\ P(D|h)P(h) \end{aligned}$$

Einfachster und naivster Ansatz: Berechne alles mit der Bayes-Regel

– Für alle Hypothesen $h \in H$, die Posteriors berechnen

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$$\overline{h} = argmax_h\ P(h|D)$$

Beispiel:

– D sei rauschfrei (korrekte Label)

– c stamme aus Hypothesenraum H

– kein a-priori Vorwissen

$$P(h) = \frac{1}{|H|} \Rightarrow \sum_{h \in H} P(h) = 1$$

– Likelihood

$$P(D|h) = \begin{cases} 1 \text{ wenn } y_i = h(x_i)\ \forall (x_i, y_i) \in D; \\ 0 \text{ sonst.} \end{cases}$$

– Zwei Fälle:

1. ist h nicht konsistent, dann ist $P(D|h) = 0$ und der Posterior ist $P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$
2. ist h konsistent, dann ist der Posterior $P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{\frac{|V_{H,D}|}{|H|}} = \frac{1}{|V_{H,D}|}$

Gegeben ein Hypothesenraum H, was ist die optimale Hypothese?

Problem: Maximum Likelihood gibt nicht das beste Ergebnis ⇒ Predictive Distribution nutzen

$$P(\overline{y}|D) = \sum_{h_i \in H} P(\overline{y}|h_i)P(h_i|D)$$

Optimale Klassifikation ist dann

$$argmax_{\overline{y}} \sum_{h_i \in H} P(\overline{y}|h_i)P(h_i|D)$$

Keine andere Klassifikation kann im Mittel besser sein unter Annahme von gleichem H und a-priori Wissen.

Aber:

- unrealistisch alle Posteriors zu berechnen
- optimaler Klassifikator nicht in H enthalten

Naive Bayes

Verwende nun anstelle der Hypothesen direkt die Attribute. Naive Bayes bestimmt den Klassifikator direkt

- Annahme von N diskreten Attributen a_i
- Annahme von k Labeln y_i

$$\begin{aligned} y_{MAP} &= argmax_{y_j}\ P(y_j|a_1, ..., a_N) \\ &= argmax_{y_j}\ \frac{P(a_1, ..., a_N|y_j)P(y_j)}{P(a_1, ..., a_N)} \\ &= argmax_{y_j}\ P(a_1, ..., a_N|y_j)P(y_j) \\ &= argmax_{y_j}(\text{likelihood} \cdot \text{prior}) \end{aligned}$$

Bestimme nun den Likelihood und Prior aus den Daten

- Prior $P(y_j) \approx$ Frequenz der Klasse y_j in den Daten
- Likelihhod
  - zählen nicht anwendbar, da nicht alle a_i für jedes Label vorhanden
  - nutze starke Annahme statistische Unabhängigkeit zwischen Attributen (in Klasse)
    $$P(a_1, ..., a_N|y_j) = \prod_{a_i} P(a_i|y_j)$$
  - bestimme $P(a_i|y_j)$ dann durch zählen

→ Naive Bayes Klassifikator

$$\tilde{y} = argmax_{y_j}\ \prod_{a_i} P(a_i|y_j)P(y_j)$$

2.8 Lazy learning

Bisher:

- Großer Aufwand für das Training
- Auswertung des gelernten Modells einfach und effizient

Alternativ:

- Speichere die Trainingsdaten oder Prototypen
- Großer Aufwand für die Auswertung
  $\hookrightarrow$ rekombiniere gespeicherte Daten für die Ausgabe

K-nächste Nachbarn (k-nearest neighbour)

Naiv:

input: x_q
output: $f(\hat{x})$ mit $\hat{x} = argmin_x \|x - x_q\|$ (Die Ausgabe von dem Beispiel, das der neuen Eingabe am nächsten ist)

Problem: Finde (berechne) den nächsten Nachbarn.

Besser:

- Bilde Mittelwert über k-nächsten Nachbarn

$$\hat{f}(x_q) = \frac{1}{k} \sum_{i=1}^{k} f(x_i)$$

- k- nächsten Nachbarn müssen erst gefunden werden, die ist Aufwendig

Problem: Finde k-nächsten Nachbarn.

Weighted k-nearest neighbours

- Bilde Mittelwert über k-nächsten Nachbarn und gewichte sie zusätzlich gemäß ihrer Entfernung zum Input

$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}, \text{ mit } w_i = d(x_i, x_q)^{-1} = \frac{1}{\text{euklidische Distanz}}$$

Problem:

- Finde k-nächste Nachbarn
- Definiere eine gute Metrik (kann schwierig sein für hochdimensionale Daten)

Bemerkung:

- die vorgestellten Methoden sind speicherintensiv
- der **induktive Bias** ist: die Funktion ist glatt, variiert lokal wenig
- die Methoden sind lokal, falls $k < N$
- kritisch ist die Wahl der Metrik $\Rightarrow$ Datamining, selbst ein Lernproblem
- es gibt Algorithmen um die nächsten Nachbarn effizient zu finden
- Gewichtungen werden erst zur Laufzeit bekannt

Anmerkung: Es scheint, dass viele "deep learning"Methoden effektiv die nächste-Nachbarn Lösung berechnen, d.h. gut als "starke Kompression mit lokaler Generalisierung"verstanden werden können.

Der Fluch der Dimensionalität

Problem: Der hochdimensionale Raum ist quasi "leer".

Beispiel (Volumen einer D-dim Kugel):

Sei $r = 1$. Wie Groß ist der Anteil des Volumens der Kugel, der zwischen $r = 1$ und $r = 1 - \epsilon$ liegt?

$$V_D(r) = K_D r^D$$

$$\frac{V_D(1) - V_D(1-\epsilon)}{V_D(1)} = 1 - \underbrace{(1-\epsilon)^D}_{<1 \underset{D\to\infty}{\to} 0} \underset{D\to\infty}{\to} 1$$

$\Rightarrow$ das gesamte Volumen der Einheitskugel konzentriert sich für $D \to \infty$ in der Kugelschale

Kernel regression

- nutze Kernel Funktionen, die von der Distanz abhängen $K_i(d(x_i, x_q))$
- verwendet Gaussfunktionen $K_\sigma(x_q - x_i)$

$$K_\sigma(x_q - x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{\|x_i - x_q\|^2}{2\sigma^2}}$$

- durch Substitution erhalten wir den **Nadaraya-Watson kernel regression estimator**

$$f(x_q) = \sum_i y_i \frac{K_\sigma(x_q - x_i)}{\sum_j K_\sigma(x_q - x_j)}$$

- verwendet formal alle Trainingsdaten, praktisch sind aber nur wenige relevant (da die Gaussfunktion lokal ist)

3 Unüberwachtes Lernen (unsupervised learning)

3.1 Prototypen und Clustering

- Clustering zielt darauf, Gruppen von "ähnlichen"Datenpunkten zu finden (und gegebenenfalls zu repräsentieren)
- in einfacher Form ordnet ein Clusteringalgorithmus einen Punkt jeweils einem Prototypen zu (auch genannt: representative, code word, center)
- "Ähnlichkeit, Nähe"wird durch eine Metrik gemessen
- Metrik ist definiert durch die folgenden drei Eigenschaften:
  - Symmetrie: $d(a,b) = d(b,a)$
  - Positive Definitheit: $d(a,b) \geq 0$
  - Dreiecksungleichung: $d(a,b) + d(b,c) \geq d(a,c)$

Metriken

- Euklidische Metrik:

$$d(a,b) = \sqrt{\sum_{i=1}^{D}(a_i - b_i)^2}$$

- Mahalanobis Distanz:

$$d(a,b) = \sqrt{(a-b)^T \Sigma^{-1}(a-b)}, \text{ mit } \Sigma^{-1} =< (x- < x >)(x- < x >)^T >$$

Vektorquantifizierung

- ein sehr einfacher Clusteralgorithmus
- gegben die Daten x_n
- initialisiere und optimiere K Prototypen $w_k,\ k = 1,..,K$
  - Zuordnung von Daten zum nächstliegenden Prototypen bzgl. Metrik $d()$

$$x_n \mapsto w_c, \text{ mit } c = argmin_k\ d(x_n, w_k)$$

- Lernziel: wähle die Position der Prototypen um den Quantisierungsfehler zu minimieren:

$$E_{VQ} = \frac{1}{N}\sum_{n=1}^{N} d(x_n, w_c(x_n))$$

Einfacher VQ-Algorithmus

1. Wähle # der Prototypen K (Modellselektion)
2. initialisiere die Position der K Prototypen
3. iteriere über die Daten:
   a) wähle zufällig x_k
   b) finde den nächstliegenden Prototypen w_c
   c) bewege den Prototypen in Richtung des Datenpunktes

$$w_c^{new} = \epsilon(w_c^{old} - x_k), \text{ mit Lernrate } \epsilon \text{ (die über die Zeit geringer wird)}$$

   d) Abbruch, wenn der Fehler konstant bleibt

k-means

- ein einfacher und effektiver Clusteralgorithmus
- K verschiedene Cluster (Modellselektion)
- K Prototypen $w_k,\ k = 1, ..., K$
- N Datenpunkte $x_n,\ n = 1, ..., N$
- ordne jedem Punkt genau einem Cluster zu durch eine Variable

$$r_{nk} = \begin{cases} 1 \text{ wenn } \|x_n - w_k\|^2 < \|x_n - w_l\|^2\ \forall l \neq k,\ k = 1, .., K \\ 0 \text{ sonst} \end{cases}$$

- minimiere den Quantisierungsfehler J (für euklidische Metrik)

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - x_k\|^2$$

- direkte simultane Optimierung von J bzgl. r_{nk} und w_k unmöglich
- Ansatz: Wechsel zwischen zwei Mengen von Variablen

Algorithm 3 k-means

Require: Number of prototypes K
Require: Initialize prototyopes $\{w_k\},\ k = 1, ..., K$

1: **repeat**
2: **E step**. Minimize J with respect to r_{nk} while keeping the w_k fixed
3: **M step**. Minimize J with respekt to w_k while keeping r_{nk} fixed
4: **until** convergence criterion is fullfilled

- Wechsel zwischen sogenannten E-step und M-step (vereinfachte Version des viel allgemeineren E(xpectation)-M(axmimization Algorithmus)
- E-step: Zuordnung von Punkten zu Clustern für feste Prototypen

$$r_{nk} = \begin{cases} 1 \text{ wenn } \|x_n - w_k\|^2 < \|x_n - w_l\|^2 \ \forall l \neq k,\ k = 1, .., K \\ 0 \text{ sonst} \end{cases}$$

- M-step: Update der Clusterzentren w_k für feste Zuordnung

$$w_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

  (damit liegt der neue Prototyp im Zentrum der zugeordneten Punkte)
- k-means konvergiert!

LGB

- wie findet (oder automatisch verbessert) man die richtige Anzahl an Clustern
- Vorgehen: Iterativ erhöhe Anzahl von Clustern bis
  - Quatisierungsfehler unter einem Schwellwert ist
  - oder: Maximum an Clustern erreicht ist

Algorithm 4 LGB

Require: Maximal number of prototypes L_{max}
Require: Threshold e_{min} for minimal quantization error E_{VQ}
Require: Initial codebook comprising a single prototype $w_1 = \frac{1}{N}\sum_{n=1}^{N} x_n$

1: **repeat**
2: **for all** prototypes w_l, $l = 1, ..., L$ (or for the M prototypes with largest quantization errors) **do**
3: Create two new prototypes

$$\begin{aligned} c_1 &= w_l + \epsilon \\ c_2 &= w_l - \epsilon \end{aligned}$$

where $\epsilon \in \mathbb{R}^D$ is an random vector of small length
4: $L = 2L$ (or $L = L + M$)
5: Apply an arbitrary clustering algorithm with L prototypes starting from the current codebook until convergence, e.g. k-means with $k = L$
6: **until** quantization error $E_{VQ} < e_{min}$ or size of codebook $|\{w_k\}| = L_{max}$

Gaussian Mixture Modell (GMM)

– k-means gruppiert "hart"jeden Punkt zu einer Gruppe, GMM kann "soft"clustering

Soft Clustering

– Wahrscheinlichkeit für Cluster

– probabilistischer Ansatz

Modellselektion: Wie viele Gauß Funktionen?

Parameteroptimierung: Mittelwerte, Varianz(matrizen), a-Priori Wahrscheinlichkeiten

Datenmodell:

$$\begin{aligned} P(x) &= \sum_{k=1}^{K} \underbrace{\pi_k}_{\text{prior}} \underbrace{N(x|w_k, \Sigma_k)}_{\text{Likelihood}}, \ \sum_k \pi_k = 1 \\ \text{Posterior: } P_k(x_n) &= \frac{\pi_k N(x_n|w_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i N(x_n|w_i, \Sigma_i)} \end{aligned}$$

Posterior: Wahrscheinlichkeit, dass Punkt x_n zu Klasse k gehört, entspricht einer weichen Zuordnung.

Problem: Likelihood hängt nun von allen Parametern ab.

- Parameter können nicht direkt berechnet werden

Lösung: E(xpectation)-M(aximization)

- Init: Wähle Startparameter
- Wiederhole:
  - E: Berechne Posterior aller Punkte
  - M: Berechne (optimiere) Parameter

EM für GMM

Gegeben ein GMM, Ziel ist es den Likelihood zu maximieren.

1. Initialisiere die Mittelwert μ_k, die Kovarianzen Σ_k und die Priors π_k und berechne den Anfangswert des log Likelihoods
2. **E step**. Berechne die Posteriors (responsibilities) mit den momentanen Parametern
$$\gamma_k(x_n) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i N(x_n|\mu_i, \Sigma_i)}$$
3. **M step**. Optimiere die Parameter
$$\begin{aligned} N_k &= \sum_{n=1}^{N} \gamma_k(x_n) \\ \mu_k &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k(x_n) x_n \\ \Sigma_k &= \frac{1}{N_k} \sum_{n=1}^{K} \gamma_k(x_n)(x_n - \mu_k)(x_n - \mu_k)^T \\ \pi_k &= \frac{N_k}{N} \end{aligned}$$
4. Berechne den log Likelihood
$$lnp(X|\mu, \Sigma) = \sum_{n=1}^{N} ln \sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k)$$
Konvergieren die Parameter oder der log Likelihood? Wenn nicht gehe zu 2..

Anmerkungen

- Likelihood
  - Erhöht sich mit jeder Iteration
  - Algorithmus konvergiert
- Initialisierung
  - Kann zufällig erfolgen
  - k-means oder LGB zur Initialisierung der Mittelwerte
  - Methoden zur automatischen Wahr der Anzahl existiert
- Update Schritte
  - Geschlossene Form für Gauß-Verteilungen
  - Für generelle Mixtures muss das nicht sein