Formeln

Lineare Regression

Linearer Zusammenhang zwischen den Eingabevariablen x und der Ausgabevariable y wird modelliert.

Hypothesenfunktion:
$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$

Kostenfunktion (MSE):
$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
mit n... Anzahl Trainingsdaten
Je kleiner $J(\theta)$, desto besser die Hypothese.

Ziel:
Finde Parameter θ um J zu minimieren
$\min J(\theta)$

Multivariat:
Mehrere Features $x_1, x_2, ..., x_n$

Polynom-Regression:
$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ...$
Wird eingesetzt, wenn wenn Features nicht-linearen Zusammenhang haben.

Logistische Regression

Hypothese:
$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

$g(z) = \frac{1}{1+e^{-z}}$..... Sigmoidfunktion
mit $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$

Klassifikation:
$h_\theta(x)$ ist quasi Wahrscheinlichkeit (0...1)
$h_\theta(x) \geq 0.5 \rightarrow$ Klasse 1
$h_\theta(x) < 0.5 \rightarrow$ Klasse 0

Entscheidungsgrenze:
$\theta^T x = 0$ weil da wird $h_\theta(x) = 0,5$

Nicht-linearität:
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + ...)$

Polynom-Regression:
$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_{2_+} \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + ...$
Fähigkeit, nicht-lineare Entscheidungsgrenzen zu erzeugen

Neuronale Netzwerke

$z_j^{(l)}$ ist der gewichtete Input für Neuron j in Layer l.
$a_j^{(l)}$ ist die Aktivierung des Neurons j in Layer l.

Parametermatrizen:
$\dim(\theta^{(l)}) = (s_{l+1} \times (s_l + 1))$
θ^l.....Gewichtsmatrizen, l..... layer,
s_l..... Anzahl Units in Layer (l), in $(s_l + 1)$, das +1 steht für Bias-Unit (zusätzliche Spalte)

Fehlerterm δ:
$\delta_{ij}^{(l)} = h_\theta\left(x^{(i)}\right) - y^{(i)}$
i... Trainingsbeispiel, j... Neuron, l... layer,
$h_\theta\left(x^{(i)}\right)$... Netzwerkvorhersage,
$y^{(i)}$... tatsächlicher Zielwert,
n... Anzahl Trainingsbeispiele,

Gradientenberechnung (Backpropagation):
$\frac{\partial}{\partial \Theta_{ij}^{(l-1)}} J(\Theta) = \frac{1}{n} \sum_{k=1}^{n} \delta_{ki}^{(l)} a_j^{(l-1)}$

Feedforward:
$z^{l+1} = \theta^l a^l$
$a^{l+1} = g\left(z^{l+1}\right)$

Backpropagation:
$\delta_j^{(l)} = \left(\sum_{k=1}^{s_{l+1}} \left(\Theta_{kj}^{(l)} \cdot \delta_k^{(l+1)}\right)\right) \cdot g'\left(z_j^{(l)}\right)$
$\Theta_{kj}^{(l)}$... Gewicht zwischen Neuron j in Layer l und Neuron k im Layer $l+1$
$\delta_k^{(l+1)}$)... Fehlerterm aus der folgenden Schicht (bereits berechnet).
$g'\left(z_j^{(l)}\right)$ Ableitung der Aktivierungsfunktion am Wert $z_j^{(l)}$

$\delta^L = a^L - y$
$\delta^l = \left(\theta^l\right)^T \delta^{l+1} \cdot g'\left(z^l\right)$

Gradientenabstieg:
$\theta^l := \theta^l - \alpha \delta^l a^{l-1}$

Aktivierungsfunktionen:
Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax

Entscheidungsbäume

Grundidee:
Baumstruktur zur schrittweisen Entscheidung basierend auf Features.

Split-Kriterien:
- Entropie: $H(S) = -\sum p_i \log_2(p_i)$
- Informationsgewinn
- Gini-Index: $\text{Gini}(S) = 1 - \sum p_i^2$

Vorteile:
- Interpretierbar
- Keine Skalierung notwendig

Nachteile:
- Overfitting bei tiefen Bäumen
- Instabil bei kleinen Datenänderungen

Pruning (Beschneiden):
Reduziert Komplexität und Overfitting

Ensemble-Methoden:
- Random Forests: viele Bäume, Voting
- Boosting (z.B. AdaBoost): sequentielle Optimierung

Reinforcement Learning

Grundidee:
Ein Agent lernt durch Interaktion mit einer Umgebung, um Belohnungen zu maximieren.

Zentrale Begriffe:
- Agent, Environment
- Zustand s, Aktion a, Belohnung r, Politik $\pi(a|s)$
- Ziel: Maximiere erwarteten kumulierten Reward

Belohnungsformel:
$R_t = \sum_{\{k=0\}}^{\{\infty\}} \gamma^k r_{\{t+k+1\}}$ mit Diskontfaktor $\gamma \in [0, 1]$

Q-Learning:
$Q(s,a) := Q(s,a) + \alpha\left[r + \gamma \max_{\{a'\}} Q(s',a') - Q(s,a)\right]$

Strategien:
- Exploration vs. Exploitation
- var ε-greedy Policy

Anwendungen:
- Spiele (z.B. AlphaGo)
- Robotik
- Empfehlungssysteme

Gradient Descent

Update-Regel:
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
α..... Lernrate
$\frac{\partial}{\partial \theta_j} J(\theta)$..... Ableitungsterm, gibt Richtung des steilsten Anstiegs an

Update-Regel für lineare Regression:
$\theta_j := \theta_j + \alpha \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) \cdot x_j^{(i)}$
mit θ_j als Parameter und $x_0^{(i)} = 1$ für den Bias-Term.

Lernrate α:
Zu groß ⟶ Divergenz,
zu klein ⟶ (zu) langsame Konvergenz

Eigenschaften:
- Bei geeigneter Lernrate konvergiert Gradient Descent zu einem lokalen Minimum der Kostenfunktion. Bei konvexen Funktionen sogar zum globalen Minimum.
- Eine konstante Lernrate kann ausreichen, wenn sie angemessen gewählt ist. Adaptive Lernraten können die Konvergenz verbessern.
- Steilere Kostenfunktionen erfordern kleinere Lernraten

Support Vector Machines

Ziel:
$\min_{w,b} \left\{\frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} \xi_i\right\}$

Nebenbedingungen:
$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i$, mit $\xi_i \geq 0 \quad (i = 1, ..., n)$

C kontrolliert Trade-off:
großes C ⟶ weniger Fehler,
kleines C ⟶ größerer Margin

Kernel:
$K(x, l) = \exp\left(-\frac{||x-l||^2}{2\sigma^2}\right)$

Kernel-Funktionen:
- **Linear**: $K(x_i, x_j) = x_i^T x_j$
  Für lineare Trennungen
- **Polynomial**: $K(x_i, x_j) = \left(x_i^T x_j + r\right)^d$
  r..... konstanter Offset
  d..... Grad des Polynoms
  Erlaubt gekrümmte trennungen (Parabeln, ...)
- **RBF (Gaussian)**: $K(x_i, x_j) = e^{-\gamma ||x_i - x_j||^2}$
  Komplexe, nicht-lineare Trennungen
- **Sigmoid**: $K(x_i, x_j) = \tanh\left(\kappa x_i^T x_j + c\right)$
  Inspiriert von neuronalen Netzen, selten verwendet

Pricipal Component Analysis(PCA)

Ziel:
Reduktion der Dimensionalität bei maximalem Erhalt der Varianz.

Schritte:
1. Zentrieren der Daten
2. Kovarianzmatrix berechnen
3. Eigenvektoren & -werte berechnen
4. Hauptkomponenten auswählen (größte Eigenwerte)
5. Projektion der Daten auf neue Achsen

Mathematisch:
Gegeben Datenmatrix X, berechne $C = 1\}\{n\} X^T X$
Finde Eigenvektoren v mit $Cv = \lambda v$

Eigenschaften:
- Unüberwachtes Verfahren
- Hauptachsen sind orthogonal
- Keine Label nötig

Anwendungen:
- Visualisierung
- Vorverarbeitung für ML
- Rauschreduktion

Modell Evaluation

Konfusionsmatrix:
TP = True Positive - Patienten, die krank sind und als krank klassifiziert wurden
FP = False Positive - Patienten, die gesund sind, aber als krank klassifiziert wurden
TN = True Negative - Patienten, die gesund sind und als gesund klassifiziert wurden
FN = False Negative - Patienten, die krank sind, aber als gesund klassifiziert wurden

Metriken:
Accuracy = $\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
Precision (Relevanz) = $\frac{\text{TP}}{\text{TP} + \text{FP}}$
Recall (Sensitivity) = $\frac{\text{TP}}{\text{TP} + \text{FN}}$
F1-Score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Beispiel:
Ein Modell mit hohem Recall identifiziert möglichst viele positive Fälle, auch auf Kosten von mehr False Positives.

Wann ist Recall wichtiger?
- Krankheitserkennung
- Sicherheitschecks
- Betrugserkennung

von Eva Hölzl

Aufgaben

Regularisierung

Kostenfunktion mit L2-Regularisierung:
$J(\theta) = \frac{1}{2n}\left(\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{d}\theta_j^2\right)$

Effekt von λ (Regularisierungsparameter):
- $\lambda = 0 \longrightarrow$ kein Penalty
- kleines $\lambda \longrightarrow$ schwache Bestrafung von großen Parameterwerten, Gefahr von Overfitting
- großes $\lambda \longrightarrow$ starke Bestrafung von großen Parameterwerten, Gefahr von Underfitting

Bias-Term θ_0 wird oft nicht regularisiert

Convolutional Neuronal Networks

Output Convolutional Filter:
Output = $\sum_{i=0}^{2}\sum_{j=0}^{2}\text{Input}\ [i,j]\cdot\text{Filter}[i,j]$
$= 0\cdot 1 + 4\cdot 2 + 2\cdot 1 + 13\cdot 2 + 25\cdot 4 + 5\cdot 2 + 8\cdot 1 + 6\cdot 2 + 3\cdot 1 = 169$

Hauptidee:
Extraktion lokaler Merkmale durch Faltungsoperationen.

Faltungsschicht (Convolution Layer):
wendet Filter (Kerne) an:
$z = W * x + b$

Pooling-Schicht:
Reduktion der Dimensionalität (z.B. Max-Pooling oder Average-Pooling)

Aktivierung:
Typisch: ReLU $f(x) = \max(0, x)$

Architektur-Beispiel:
Input $\longrightarrow$ Conv $\longrightarrow$ ReLU $\longrightarrow$ Pool $\longrightarrow$ Dense $\longrightarrow$ Output

Parameteranzahl:
Abhängig von Filtergröße und Anzahl

Vorteile:
- Translation-Invarianz
- Weniger Parameter als vollständig verbundene Netze

Backpropagation Aufgabe (21)

Gegeben:
- $x^{(1)} = \left(x_1^{(1)}, x_2^{(1)}\right) = \begin{pmatrix}0.5 & 0.7\end{pmatrix}$,
- $y^{(1)} = (1)$, ("Klasse 1"),
- $h_\theta\left(x^{(1)}\right) = 0.5$,
- Aktivierungsfunktion: Sigmoid $a_j^{(i)} = g\left(z_j^{(i)}\right) = \frac{1}{1+e^{-z_j^{(i)}}}$
- Gewichtsmatrizen:
  $\Theta^{(1)} = \begin{pmatrix}1 & 1 & 2\\ 2 & 1.5 & 0.7\\ -0.5 & -1 & 2\end{pmatrix}$,
  $\Theta^{(2)} = \begin{pmatrix}1 & -0.9 & -0.7 & 0.9\end{pmatrix}$

Forward-Propagation:
- Schritt 1 - Input Layer (inklusive Bias):
  $a^{(1)} = \begin{pmatrix}1 & x_1^{(1)} & x_2^{(1)}\end{pmatrix} = \begin{pmatrix}1 & 0.5 & 0.7\end{pmatrix}$
- Schritt 2 - Hidden Layer Input berechnen:
  $z^{(2)} = \Theta^{(1)}\cdot a^{(1)}\begin{pmatrix}1 & 1 & 2\\ 2 & 1.5 & 0.7\\ -0.5 & -1 & 2\end{pmatrix}\cdot\begin{pmatrix}1 & 0.5 & 0.7\end{pmatrix}$
  $= \begin{pmatrix}1\cdot1+1\cdot0.5+2\cdot0.7 & 2\cdot1+1.5\cdot0.5+0.7\cdot0.7 & -0.5\cdot1-1\cdot0.5+2\cdot0.7\end{pmatrix}$
  $= \begin{pmatrix}2.9 & 3.24 & 0.4\end{pmatrix}$
- Schritt 3 - Aktivierungsfunktion (Sigmoid) anwenden:
  $a^{(2)} = g\left(z^{(2)}\right) = \left(1, \frac{1}{1+e^{-z_1^{(2)}}}, \frac{1}{1+e^{-z_2^{(2)}}}, \frac{1}{1+e^{-z_3^{(2)}}}\right) \approx$
  $\begin{pmatrix}1 & 0.948 & 0.962 & 0.599\end{pmatrix}$, die 1 ist der Bias-Term.
- Schritt 4 - **Gradientenberechnung (Backpropagation)**
  $\frac{\partial}{\partial\Theta_{ij}^{(l-1)}}J(\Theta) = \frac{1}{n}\sum_{k=1}^{n}\delta_{ki}^{(l)}a_j^{(l-1)}$
  $\delta_{11}^{(3)}\cdot a^{(2)} = -0.5\cdot 0.948 = -0.474$
- Schritt 5 **Prüfen mittels Gradientenquotienten**:
  $J(\Theta) = \frac{1}{2n}(\sum_{i=1}^{n}\left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right)^2$
  $= \frac{1}{2}(0.5-1)^2 = 0.125$

von Eva Hölzl