Künstliche Intelligenz

Starke KI: Hat um Ziel eine künstliche Intelligenz zu erschaffen, die wie der Mensch kreativ nachdenken und Probleme lösen kann und sich durch ein Bewusstsein bzw. Emotionen auszeichnet.

Turing-Test: Ein Mensch stellt einer KI Fragen und kann nicht unterscheiden ob es sich um einen Menschen oder eine Roboter handelt. Soll ein Kriterium sein, wann eine Maschine eine dem Menschen gleichwertige Intelligenz simuliert. Aussagekraft?

Schwache KI: Es geht darum konkrete Probleme zu meistern. Dazu zählt auch die Simulation von intelligentem Verhalten.

Machine Learning - Probleme:

- Regression: Vorhersagen über zukünftige Events zu machen.
- Klassifikation: Objekterkennung
- Clustering: Ähnlich zur Klassifikation aber mit (unsupervised Learning), Sportler in Cluster einteilen, Netflix Film Vorschläge, Spam-filter, ...
- Strategie-Entscheidung

Maschine Learning - Trainingsmethode:

- unsupervised: anhand von vorgegebenen Regeln - Clustering
- supervised: mit gelabelten Daten - Regression, Klassifikation,
- reinforcement: über Belohnungen

K-Means

Dieser Algorithmus ist ein unsupervised clustering Algorithmus, der Daten in genau k Cluster basierend auf gewissen Eigenschaften einteilt. Die Anzahl der Eigenschaften gibt also die Dimension der Einzelnen Datenpunkte an.

Definitionen: Sei $K \in \mathbb{N}$ die Anzahl der Cluster und X eine Menge mit N, d-dimensionalen Vektoren, die die einzelnen Datenpunkte darstellen. Sein außerdem μ_k mit $k \in \{1, \dots, K\}$ die Cluster-Mittelpunkte - also d-dimensionale Vektoren - und $r \in \{0,1\}^{N \times K}$ eine binäre Matrix, wobei $r_{i,j} = 1$, genau dann wenn der i-te Datenpunkt dem j-ten Cluster zugeordnet ist.

Cluster-Varianz/Zielfunktion: Als Cluster-Varianz (J) bezeichnet man die Summe aller Abstände der Datenpunkte zu dem Mittelpunkt des ihnen zugeordneten Clusters. Die Abstände zweier Vektoren wird durch die quadratische Euklidische Distanz berechnet. Es ergibt sich also

$$J = \sum_{k=1}^{K} \sum_{n=1}^{N} r_{n,k} \cdot ||x_n - \mu_k||^2$$

Algorithmus: Sei $k \in \mathbb{N}$ die Anzahl der Cluster und X eine Menge mit N, d-dimensionalen Vektoren, die die einzelnen Datenpunkte darstellen. Sein außerdem μ_i mit $i \in \{1, \dots, K\}$ die Cluster-Mittelpunkte - also d-dimensionale Vektoren. So besteht der Algorithmus aus 3 Schritten:

1. *Initialisierung der Mittelpunkte*: Heißt wir weißen den Werten μ_k für $k \in \{1, \dots, K\}$ je zufällige Vektoren aus $\mathbb{R}^d$ zu.
2. *Zuordnung/Erwartungsschritt*: Jeder Datenpunkt wird nun demjenigen Cluster zugeordnet, bei dem die Cluster-Varianz am wenigsten erhöht wird. Also: "Jeder Datenpunkt wir dem Cluster zugeordnet dessen Mittelpunkt am nähsten an dem Datenpunkt ist.".
3. *Aktualisierung/Maximierungsschritt*: Berechne die Mittelpunkte der Cluster neu. Such also für jedes μ_k einen Wert, sodass J minimal wird. Also leiten wir J für jedes μ_i ab, setzen 0 und formen um. So bekommen wir raus, dass die neuen Cluster-Mittelpunkte einfach der Schwerpunkt der zu dem Cluster gehörenden Datenpunkte ist.

$$\begin{aligned} J_k &= \sum_{n=1}^{N} r_{n,k} \cdot ||x_n - \mu_k||^2 \\ \frac{dJ_k}{d\mu_k} &= 2 \cdot \sum_{n=1}^{N} r_{n,k} \cdot (x_n - \mu_k) = 0 \\ \mu_k &= \frac{\Sigma_n \cdot r_{n,k} \cdot x_n}{\Sigma_n \cdot r_{n,k}} \end{aligned}$$

4. Wiederhole `2`, `3` beliebig oft.

Konvergenz und Minimum: Da die Schritte `2` und `3` die Zielfunktion immer verkleinern konvergiert der Algorithmus sicher, allerdings erreicht er evl. nur lokale Minima.

Lineare Regression

Dieser Algorithmus ist ein Supervised regressions Algorithmus. Dabei wir eine abhängige Variable y durch eine oder mehrere unabhängige Variablen(=Features) X dargestellt.

Definitionen: Sei $m \in \mathbb{N}$ die Anzahl der Trainings-Daten und $n \in \mathbb{N}$ die Anzahl der Features. Sei $X \in \mathbb{R}^{m \times n}$ eine Matrix die alle Trainings-Daten enthält, wobei je ein n-dimensionaler Daten-Punkt eine Zeile ist. Sei außerdem $\hat{y}$ ein Vektor, der die Vorhersage des Models für jede der Trainings-Daten enthält. Sein $w \in \mathbb{R}^n$ dann noch die Gewichte, also ein Wert für jedes Feature und $b \in \mathbb{R}$ der Bias.

Zielfunktion: Bei lineare Regression wollen wir die Werte für w und b so anpassen, dass die Gerade, die sie beschreiben, einen möglichst geringen Abstand zu den Trainings-Daten-Punkten hat. Heißt die Zielfunktion ist die Summe der quadratischen Abstände.

$$J = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

wobei y_i hier der wahre Wert der abhängigen Variable für einen Trainings-Datenpunkt ist.

Algorithmus: Unser Model berechnet eine Vorhersage durch die normale Geraden-Formel. Also für einen Trainings-Datenpunkt mit Index i gilt

$$\hat{y}_i = \sum_{j=1}^{n} x_{i,j} w_j + b$$

bzw.

$$\hat{y} = Xw + \{b, b, \ldots\}$$

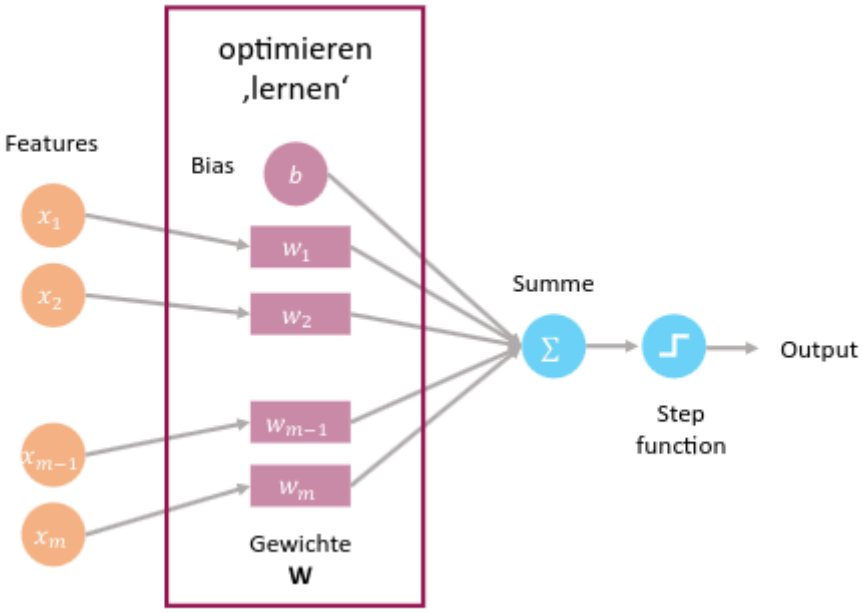
Konkret funktioniert der Algorithmus also nun nach den folgenden Schritten:

1. Zufällige Zuweisung von Werten für w und b.
2. Berechnen aller Vorhersagen des Modells.
3. Optimierungsschritt: Minimierung der Zielfunktion durch Gradienten-abstieg. Also updaten der Gewichte und des Bias mit

$$w_j \to w_j - \alpha \frac{\delta J}{\delta w_j}; \quad \frac{\delta J}{\delta w_j} = 2 \frac{1}{m} \sum_{i=0}^{m} x_{i,j} \cdot (\hat{y}_i - y_i)$$

$$b \to b - \alpha \frac{\delta J}{\delta b}, \quad \frac{\delta J}{\delta b} = 2 \frac{1}{m} \sum_{i=0}^{m} (\hat{y}_i - y_i)$$

Lineare Regression als Neuronales Netz: Hinzufügen einer Stepfunktion wodurch ein Perzeptron entsteht. Dann durch kombination von mehreren Perzeptronen - Neuronales Netz.


optimieren
‚lernen‘
Features
Bias
b
x_1
x_2
x_{m-1}
x_m
w_1
w_2
w_{m-1}
w_m
Gewichte
W
Summe
$\sum$
Step
function
Output