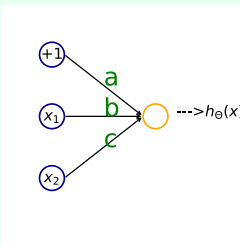


Formeln

Lineare Regression	Support Vector Machines	Neuronale Netzwerke	Entscheidungsbaeume	Modell Evaluation																
<p>Linearer Zusammenhang zwischen den Eingabevariablen x und der Ausgabevariable y wird modelliert.</p> <p><b>Hypothesenfunktion:</b></p> $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ <p><b>Kostenfunktion (MSE):</b></p> $J(\theta) = \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$ <p>mit n... Anzahl Trainingsdaten</p> <p>Je kleiner <math>J(\theta)</math>, desto besser die Hypothese.</p> <p><b>Ziel:</b> Finde Parameter <math>\theta</math> um J zu minimieren <b>min <math>J(\theta)</math></b></p> <p><b>Multivariat:</b></p> <p>Mehrere Features <math>x_1, x_2, \dots, x_n</math></p> <p><b>Polynom-Regression:</b></p> $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots$ <p>Wird eingesetzt, wenn wenn Features nicht-linearen Zusammenhang haben.</p>	<p><b>Ziel:</b></p> $\min_{w,b} \left\{ \frac{1}{2} \ w\ ^2 + C \sum_{i=1}^n \xi_i \right\}$ <p><b>Nebenbedingungen:</b></p> $y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \text{ mit } \xi_i \geq 0 \quad (i = 1, \dots, n)$ <p><b>C kontrolliert Trade-off:</b></p> <p>Um <b>Outlier</b> weniger zu gewichten und eine robustere Entscheidungsgrenze zu erhalten, muss der Parameter C verkleinert werden. Dadurch ist weniger Gewicht auf den „Strafterm“ für das Nicht-Einhalten der Nebenbedingung großes <math>C \rightarrow</math> weniger Fehler, kleines <math>C \rightarrow</math> größerer Margin</p> <p><b>Kernel:</b></p> $K(x, l) = \exp\left(-\frac{\ x-l\ ^2}{2\sigma^2}\right)$ <p><b>Kernel-Funktionen:</b></p> <ul style="list-style-type: none"><li>• <b>Linear:</b> <math>K(x_i, x_j) = x_i^T x_j</math> Für lineare Trennungen</li><li>• <b>Polynomial:</b> <math>K(x_i, x_j) = (x_i^T x_j + r)^d</math> <math>r</math>..... konstanter Offset <math>d</math>..... Grad des Polynoms Erlaubt gekrümmte Trennungen (Parabeln, ...)</li><li>• <b>RBF (Gaussian):</b> <math>K(x_i, x_j) = e^{-\gamma \ x_i - x_j\ ^2}</math> Komplexe, nicht-lineare Trennungen</li><li>• <b>Sigmoid:</b> <math>K(x_i, x_j) = \tanh(\kappa x_i^T x_j + c)</math> Inspiriert von neuronalen Netzen, selten verwendet</li></ul> <p><b>Wählen von Stützpunkten (“Landmarks”):</b></p> <ul style="list-style-type: none"><li>• Oft einfach: Alle Trainingspunkte <math>\rightarrow</math> jeder Punkt wird zum Landmark (klassisch bei RBF-Kernel)</li><li>• Alternativ: nur ausgewählte Punkte, z.B.<ul style="list-style-type: none"><li>▸ Support-Vektoren (nur „wichtige“ Punkte)</li><li>▸ Clusterzentren (z.B. mit K-Means)</li><li>▸ Gleichmäßig verteilt im Raum (für generalisierte Modelle)</li></ul></li></ul> <p>Ziel: Landmark-Punkte definieren die Feature-Transformation durch den Kernel.</p> <ul style="list-style-type: none"><li>• Vorhersage: <math>y = 1</math> <math>y = 1</math>, wenn <math>b + w_1 f_1 + w_2 f_2 + w_3 f_3 \geq 0</math>, sonst <math>y = -1</math></li><li>• Neue Landmark berücksichtigen <math>\rightarrow</math> passenden <math>w_i</math> erhöhen</li><li>• Bereich zwischen zwei Landmarks ausschließen <math>\rightarrow w</math> verkleinern</li><li>• Bereich mit <math>y = 1</math> vergrößern <math>\rightarrow w</math> oder <math>b</math> erhöhen</li><li>• Bereich mit <math>y = 1</math> verkleinern <math>\rightarrow w</math> oder <math>b</math> verringern</li></ul>	<p><math>z_j^{(l)}</math> ist der gewichtete Input für Neuron <math>j</math> in Layer <math>l</math>. <math>a_j^{(l)}</math> ist die Aktivierung des Neurons <math>j</math> in Layer <math>l</math>.</p> <p><b>Parametermatrizen:</b></p> <p><math>\dim(\theta^{(l)}) = (s_{l+1} \times (s_l + 1))</math> <math>\theta^l</math>.....Gewichtsmatrizen, <math>l</math>..... layer, <math>s_l</math>..... Anzahl Units in Layer (<math>l</math>), in <math>(s_l + 1)</math>, das +1 steht für Bias-Unit (zusätzliche Spalte)</p> <p><b>Fehlerterm <math>\delta</math>:</b></p> $\delta_{ij}^{(l)} = h_{\theta}(x^{(i)}) - y^{(i)}$ <p><math>i</math>... Trainingsbeispiel, <math>j</math>... Neuron, <math>l</math>... layer, <math>h_{\theta}(x^{(i)})</math>... Netzwerkvorhersage, <math>y^{(i)}</math>... tatsächlicher Zielwert, <math>n</math>... Anzahl Trainingsbeispiele,</p> <p><b>Gradientenberechnung (Backpropagation):</b></p> $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{n} \sum_{k=1}^n \delta_{ki}^{(l)} a_j^{(l-1)}$ <p><b>Feedforward:</b></p> $z^{l+1} = \theta^l a^l$ $a^{l+1} = g(z^{l+1})$ <p><b>Backpropagation:</b></p> $\delta_j^{(l)} = \left( \sum_{k=l+1}^{s_{l+1}} \Theta_{kj}^{(l)} \cdot \delta_k^{(l+1)} \right) \cdot g' \left( z_j^{(l)} \right)$ <p><math>\Theta_{kj}^{(l)}</math> ... Gewicht zwischen Neuron <math>j</math> in Layer <math>l</math> und Neuron <math>k</math> im Layer <math>l + 1</math> <math>\delta_k^{(l+1)}</math>.... Fehlerterm aus der folgenden Schicht (bereits berechnet). <math>g' \left( z_j^{(l)} \right)</math> Ableitung der Aktivierungsfunktion am Wert <math>z_j^{(l)}</math></p> $\delta^L = a^L - y$ $\delta^l = \left( \theta^l \right)^T \delta^{l+1} \cdot g' \left( z^l \right)$ <p><b>Gradientenabstieg:</b></p> $\theta^l := \theta^l - \alpha \delta^l a^{l-1}$ <p><b>Aktivierungsfunktionen:</b></p> <p>Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax</p> <p><b>Symmetrie:</b></p> <p>Austausch der Neuronen in einem Hidden Layer (inkl. zugehöriger Gewichte im nächsten Layer) <math>\rightarrow</math> hat keinen Einfluss auf den Output <math>h_{\Theta}(x)</math>, da die Gesamtfunktion identisch bleibt.</p>	<p><b>Grundidee:</b></p> <p>Baumstruktur zur schrittweisen Entscheidung basierend auf Features.</p> <p><b>Split-Kriterien:</b></p> <ul style="list-style-type: none"><li>• Entropie: <math>H = -p_i \log_2(p_i) - p_j \log_2(p_j) - \dots</math> <math>p_i</math>...Wahrscheinlichkeit (relative Häufigkeit) der Klasse <math>i</math>, <math>p_j</math>...Wahrscheinlichkeit der Klasse <math>j</math></li></ul> $p_i = \frac{n_i}{n}, p_j = \frac{n_j}{n}$ <p><math>n</math>... Gesamtanzahl Datenpunkte, <math>n_i</math>... Anzahl Datenpunkte der Klasse <math>i</math>, <math>n_j</math>... Anzahl Datenpunkte der Klasse <math>j</math></p> $H = 0 \rightarrow \text{maximale Ordnung (reine klasse)}$ $H = 1 \rightarrow \text{maximale Unordnung}$ <p><b>Informationsgewinn Gain(A):</b></p> $\text{Gain}(A) = H(\text{Ursprungspartition}) - \left( \frac{ P1 }{ P1 + P2 } \cdot H(P1) + \frac{ P2 }{ P1 + P2 } \cdot H(P2) \right)$ <p><math>A</math>... Partitionierung des Datensatzes <math> P1 </math> ...Anzahl der Datenpunkte in Partition 1 <math> P2 </math> ...Anzahl der Datenpunkte in Partition 2 <math>( P1  +  P2 )</math>...Anzahl ALLER Datenpunkte</p> <ul style="list-style-type: none"><li>• Gini-Index: <math>\text{Gini}(S) = 1 - \sum p_i^2</math></li></ul> <p><b>Vorteile:</b></p> <ul style="list-style-type: none"><li>• Interpretierbar</li><li>• Keine Skalierung notwendig</li></ul> <p><b>Nachteile:</b></p> <ul style="list-style-type: none"><li>• Overfitting bei tiefen Bäumen</li><li>• Instabil bei kleinen Datenänderungen</li></ul> <p><b>Pruning (Beschneiden):</b></p> <p>Reduziert Komplexität und Overfitting</p>	<p><b>Konfusionsmatrix:</b></p> <p><b>TP = True Positive</b> - Patienten, die krank sind und als krank klassifiziert wurden</p> <p><b>FP = False Positive</b> - Patienten, die gesund sind, aber als krank klassifiziert wurden</p> <p><b>TN = True Negative</b> - Patienten, die gesund sind und als gesund klassifiziert wurden</p> <p><b>FN = False Negative</b> - Patienten, die krank sind, aber als gesund klassifiziert wurden</p> <p><b>Metriken:</b></p> $\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$ $\text{Precision (Relevanz)} = \frac{\text{TP}}{\text{TP} + \text{FP}}$ $\text{Recall (Sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$ $\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$ <p><b>Beispiel:</b></p> <p>Ein Modell mit hohem Recall identifiziert möglichst viele positive Fälle, auch auf Kosten von mehr False Positives.</p> <p><b>Wann ist Recall wichtiger?</b></p> <ul style="list-style-type: none"><li>• Krankheitserkennung</li><li>• Sicherheitschecks</li><li>• Betrugserkennung</li></ul> <p><b>Hohe Fehler auf Training &amp; CV</b></p> <p><math>\rightarrow</math> (Underfitting) Braucht mehr hidden Layers, Neuronen, weniger Regularisierung</p> <p><b>Gute Trainingsperformance, schlechte CV</b></p> <p><math>\rightarrow</math> (Overfitting) Braucht mehr Trainingsdaten, weniger Features, mehr Regularisierung</p>																
Gradient Descent																				
<p><b>Update-Regel:</b></p> $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ <p><math>\alpha</math>..... Lernrate</p> <p><math>\frac{\partial}{\partial \theta_j} J(\theta)</math>..... Ableitungsterm, gibt Richtung des steilsten Anstiegs an</p> <p><b>Update-Regel für lineare Regression:</b></p> $\theta_j := \theta_j + \alpha \frac{1}{n} \sum_{i=1}^n \left[ \left( y^{(i)} - h_{\theta}(x^{(i)}) \right) \cdot x_j^{(i)} \right]$ <p>mit <math>\theta_j</math> als Parameter und <math>x_0^{(i)} = 1</math> <b>für den Bias-Term, beim ersten <math>\theta_0</math> Update.</b></p> <p><b>Lernrate <math>\alpha</math>:</b></p> <p>Zu groß <math>\rightarrow</math> Divergenz, zu klein <math>\rightarrow</math> (zu) langsame Konvergenz</p> <ul style="list-style-type: none"><li>• <b>Beispiel:</b> Kostenfunktion mit den neuen Parametern (nach einem Gradient Descent Update) ist größer als mit den alten Parametern (vor dem Update). Vermutlich ist die Lernrate <math>\alpha</math> zu groß gewählt</li></ul> <p><b>Eigenschaften:</b></p> <ul style="list-style-type: none"><li>• Bei geeigneter Lernrate konvergiert Gradient Descent zu einem <b>lokalen Minimum</b> der Kostenfunktion. Bei konvexen Funktionen sogar zum <b>globalen Minimum</b>. Bei Update nach dem Erreichen des Minimums bleibt <math>\theta</math> konstant.</li><li>• Eine konstante Lernrate kann ausreichen, wenn sie angemessen gewählt ist. Adaptive Lernraten können die Konvergenz verbessern.</li><li>• Steilere Kostenfunktionen erfordern kleinere Lernraten</li></ul>																				
Boolesche Neuronale Netzwerke																				
																				
<table><tr><th>Funktion</th><th>a</th><th>b</th><th>c</th></tr><tr><td><math>x_1</math> and <math>x_2</math></td><td>-30</td><td>20</td><td>20</td></tr><tr><td>(not <math>x_1</math>) and (not <math>x_2</math>)</td><td>10</td><td>-20</td><td>-20</td></tr><tr><td><math>x_1</math> or <math>x_2</math></td><td>-10</td><td>20</td><td>20</td></tr></table>					Funktion	a	b	c	$x_1$ and $x_2$	-30	20	20	(not $x_1$ ) and (not $x_2$ )	10	-20	-20	$x_1$ or $x_2$	-10	20	20
Funktion	a	b	c																	
$x_1$ and $x_2$	-30	20	20																	
(not $x_1$ ) and (not $x_2$ )	10	-20	-20																	
$x_1$ or $x_2$	-10	20	20																	

Formeln 2

Convolutional Neuronal Networks	Backpropagation Aufgabe (21)	Pricipal Component Analysis(PCA)	Reinforcement Learning	Logistische Regression																																																																		
<p><b>Stride:</b> Bestimmt, um wie viele Pixel der Filter bei jedem Schritt verschoben wird. Ein größerer Stride führt zu einer stärkeren Dimensionsreduktion.</p> <p><b>Padding:</b>Das Hinzufügen zusätzlicher Pixel am Rand des Eingabebildes.</p> <ul style="list-style-type: none"><li>• <b>“Valid”</b> Padding: Kein zusätzliches Padding, der Filter wird nur angewendet, wenn er vollständig innerhalb des Bildes liegt.</li><li>• <b>“Same”</b> Padding: Fügt Padding hinzu, sodass die Ausgabedimension gleich der Eingabedimension bleibt (bei Stride 1).</li></ul> <p>Ausgabedimension = <math>\frac{\text{Eingabedimension} - \text{Filtergröße} + 2 \cdot \text{Padding}}{\text{Stride}} + 1</math></p> <p><b>Output Convolutional Filter:</b></p> <p>Output = <math>\sum_{i=0}^2 \sum_{j=0}^2 \text{Input} [i, j] \cdot \text{Filter}[i, j]</math></p> <p>Beispiel:</p> <div><div>Filter:</div><table><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>2</td><td>4</td><td>2</td></tr><tr><td>1</td><td>2</td><td>1</td></tr></table><div>Input:</div><table><tr><td>2</td><td>5</td><td>4</td></tr><tr><td>13</td><td>25</td><td>15</td></tr><tr><td>8</td><td>6</td><td>6</td></tr></table></div> <p><math>= 2 * 1 + 5 * 2 + 4 * 1 + 13 * 2 + 25 * 4 + 15 * 2 + 8 * 1 + 6 * 2 + 6 * 1 = 2 + 10 + 4 + 26 + 100 + 30 + 8 + 12 + 6 = 198</math></p> <p><b>Hauptidee:</b></p> <p>Extraktion lokaler Merkmale durch Faltungsoperationen.</p> <p><b>Faltungsschicht (Convolution Layer):</b></p> <p>wendet Filter (Kerne) an:</p> $z = W * x + b$ <p><b>Aktivierung:</b></p> <p>Typisch: ReLU <math>f(x) = \max(0, x)</math></p> <p><b>Architektur-Beispiel:</b></p> <p>Input → Conv → ReLU → Pool → Dense → Output</p> <p><b>CNN Outputdimension:</b></p> <p>Input: 100x100 Pixel, 3 Kanäle</p> <p>Kernel: 3x3, Stride = 1, „same padding“</p> <ol style="list-style-type: none"><li>1. 1 Filter → Output-Dim: 100x100x1</li><li>2. 60 Filter → Output-Dim: 100x100x60</li></ol> <p>Faustregel: Anzahl der Filter = Anzahl Output-Kanäle</p> <p><b>Vorteile:</b></p> <ul style="list-style-type: none"><li>• Translation-Invarianz</li><li>• Weniger Parameter als vollständig verbundene Netze</li></ul>	1	2	1	2	4	2	1	2	1	2	5	4	13	25	15	8	6	6	<p><b>Gegeben:</b></p> <ul style="list-style-type: none"><li>• <math>x^{(1)} = \begin{pmatrix} x_1^{(1)} , x_2^{(1)} \end{pmatrix} = (0.5 \ 0.7)</math>,</li><li>• <math>y^{(1)} = (1)</math>, (“Klasse 1”),</li><li>• <math>h_{\theta}(x^{(1)}) = 0.5</math>,</li><li>• Aktivierungsfunktion: Sigmoid <math>a_j^{(i)} = g\left(z_j^{(i)}\right) = \frac{1}{1+e^{-z_j^{(i)}}}</math></li><li>• Gewichtsmatrizen: <math>\Theta^{(1)} = \begin{pmatrix} 1 &amp; 1 &amp; 2 \\ 2 &amp; 1.5 &amp; 0.7 \\ -0.5 &amp; -1 &amp; 2 \end{pmatrix}</math>, <math>\Theta^{(2)} = (1 \ -0.9 \ -0.7 \ 0.9)</math></li></ul> <p><b>Forward-Propagation:</b></p> <ul style="list-style-type: none"><li>• Schritt 1 - Input Layer (inklusive Bias): <math>a^{(1)} = (1 \ x_1^{(1)} \ x_2^{(1)}) = (1 \ 0.5 \ 0.7)</math></li><li>• Schritt 2 - Hidden Layer Input berechnen: <math>z^{(2)} = \Theta^{(1)} \cdot a^{(1)} \begin{pmatrix} 1 &amp; 1 &amp; 2 \\ 2 &amp; 1.5 &amp; 0.7 \\ -0.5 &amp; -1 &amp; 2 \end{pmatrix} \cdot (1 \ 0.5 \ 0.7)</math> <math>= (1 \cdot 1 + 1 \cdot 0.5 + 2 \cdot 0.7 \ 2 \cdot 1 + 1.5 \cdot 0.5 + 0.7 \cdot 0.7 \ -0.5 \cdot 1 - 1 \cdot 0.5 + 2 \cdot 0.7)</math> <math>= (2.9 \ 3.24 \ 0.4)</math></li><li>• Schritt 3 - Aktivierungsfunktion (Sigmoid) anwenden: <math>a^{(2)} = g(z^{(2)}) = \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 (1 \ 0.948 \ 0.962 \ 0.599)</math>, die 1 ist der Bias-Term.</li><li>• Schritt 4 - <b>Gradientenberechnung (Backpropagation)</b> <math>\frac{\partial}{\partial \Theta_{ij}^{(l-1)}} J(\Theta) = \frac{1}{n} \sum_{k=1}^n \delta_{ki}^{(l)} a_j^{(l-1)}</math> <math>\delta_{11}^{(3)} \cdot a^{(2)} = -0.5 \cdot 0.948 = -0.474</math></li><li>• Schritt 5 <b>Prüfen mittels Gradientenquotienten:</b> <math>J(\Theta) = \frac{1}{2n} (\sum_{i=1}^n (h_{\Theta}(x^{(i)}) - y^{(i)})^2</math> <math>= \frac{1}{2} (0.5 - 1)^2 = 0.125</math></li></ul>	<p><b>Ziel:</b></p> <p>Reduktion der Dimensionalität bei maximalem Erhalt der Varianz.</p> <p><b>Schritte:</b></p> <ol style="list-style-type: none"><li>1. Zentrieren der Daten</li><li>2. Kovarianzmatrix berechnen</li><li>3. Eigenvektoren &amp; -werte berechnen</li><li>4. Hauptkomponenten auswählen (größte Eigenwerte)</li><li>5. Projektion der Daten auf neue Achsen</li></ol> <p><b>Eigenschaften:</b></p> <ul style="list-style-type: none"><li>• Unüberwachtes Verfahren</li><li>• Hauptachsen sind orthogonal</li><li>• Keine Label nötig</li></ul> <p><b>Anwendungen:</b></p> <ul style="list-style-type: none"><li>• Visualisierung</li><li>• Vorverarbeitung für ML</li><li>• Rauschreduktion</li></ul>	<p><b>Value Function:</b></p> <p>Erwarteter kumulativer Reward für eine gegebene Policy <math>\pi</math>. Maß für die “Güte” eines Zustands:</p> <p><math>v_{\pi}(S_1)</math> = unmittelbarer Reward + Wert des Folgezustands</p> <p><math>v_{\pi}(S_1) = R_2 + v_{\pi}(S_2)</math></p> <p><math>s...</math> Zustand, <math>\pi...</math> Policy, <math>R...</math> unmittelbarer Reward</p> <p><math>v_{\pi}(s)...</math> Wertfunktion für Zustand <math>s</math></p> <p>“<b>Wert von Zustand „S“</b>“:</p> <p>Einfach alle Werte der kommenden Felder, entlang der Policy <math>\pi</math> addieren.</p> <p><math>v_{\pi}(S) = (-1) * 16</math></p> <table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>-1</td><td>-1</td><td>-1</td><td>-1</td><td>-1</td><td>-1</td><td>G</td></tr><tr><td></td><td>-1</td><td></td><td></td><td>-1</td><td></td><td></td><td></td></tr><tr><td></td><td>-1</td><td></td><td>-1</td><td>-1</td><td>-1</td><td>-1</td><td></td></tr><tr><td></td><td>-1</td><td>-1</td><td>-1</td><td></td><td></td><td></td><td>-1</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>S</td></tr></table> <p><b>Grundidee:</b></p> <p>Ein Agent lernt durch Interaktion mit einer Umgebung, um Belohnungen zu maximieren.</p> <p><b>Zentrale Begriffe:</b></p> <ul style="list-style-type: none"><li>• Agent, Environment</li><li>• Zustand <math>s</math>, Aktion <math>a</math>, Belohnung <math>r</math>, Politik <math>\pi(a s)</math></li><li>• Ziel: Maximiere erwarteten kumulierten Reward</li></ul> <p><b>Belohnungsformel:</b></p> $R_t = \sum_{k=0}^{\{\infty\}} \gamma^k r_{\{t+k+1\}}$ <p>mit Diskontfaktor <math>\gamma \in [0, 1]</math></p> <p><b>Q-Learning:</b></p> $Q(s, a) := Q(s, a) + \alpha \left[ r + \gamma \max_{\{a'\}} Q(s', a') - Q(s, a) \right]$ <p><b>Strategien:</b></p> <ul style="list-style-type: none"><li>• Exploration vs. Exploitation</li><li>• var <math>\varepsilon</math>-greedy Policy</li></ul> <p><b>Anwendungen:</b></p> <ul style="list-style-type: none"><li>• Spiele (z.B. AlphaGo)</li><li>• Robotik</li><li>• Empfehlungssysteme</li></ul>										-1	-1	-1	-1	-1	-1	G		-1			-1					-1		-1	-1	-1	-1			-1	-1	-1				-1								S	<p><b>Hypothese:</b></p> $h_{\theta}(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$ <p><math>g(z) = \frac{1}{1+e^{-z}}.....</math> Sigmoidfunktion</p> <p>mit <math>\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n</math></p> <p><b>Klassifikation:</b></p> <p><math>h_{\theta}(x)</math> ist quasi Wahrscheinlichkeit (0..1)</p> <p><math>h_{\theta}(x) \geq 0.5 \rightarrow</math> Klasse 1</p> <p><math>h_{\theta}(x) &lt; 0.5 \rightarrow</math> Klasse 0</p> <p><b>Entscheidungsgrenze:</b></p> <ol style="list-style-type: none"><li>1. <math>\theta^T x = 0</math> bzw. <math>z = 0</math> weil da wird <math>h_{\theta}(\theta^T x) = 0.5</math></li><li>2. Umformen zu <math>x_2 = x_1 + d</math></li><li>3. <b>Beispiel</b> mit <math>x_2 = 2 - 1.5x_1</math>:</li></ol> <div><p>Entscheidungsgrenze und Klassenbereiche</p></div>
1	2	1																																																																				
2	4	2																																																																				
1	2	1																																																																				
2	5	4																																																																				
13	25	15																																																																				
8	6	6																																																																				
	-1	-1	-1	-1	-1	-1	G																																																															
	-1			-1																																																																		
	-1		-1	-1	-1	-1																																																																
	-1	-1	-1				-1																																																															
							S																																																															