# Contents

# Machine Learning

## SVMs und Neuronale Netzwerke

## 1 Machine Learning

Machine Learning ist ein Konzept, bei dem der Computer durch Algorithmen aus Daten lernt, gewisse Probleme selbstständig zu lösen. Dabei erkennt der Computer nach einer gewissen Lernphase Gesetzmäßigkeiten, durch die er dann in der Lage ist, mit neuen, ihm unbekannte Datensätze umzugehen. Dabei werden verschiedene Modelle genutzt, doch zunächst einmal sollten wir uns mit den grundsätzlichen Begriffen Klassifikation und Regression vertraut machen.

### 1.1 Klassifikation und Regression

Bei Klassifikation und Regression handelt es sich um zwei verscheidene Möglichkeiten zur Problemlösung im Bereich Machine Learning.

**Klassifikation.** Bei der Klassifikation sorgt der Algorithmus am Ende dafür, dass der Datensatz vom Computer in verschiedene Klassen eingeteilt wird. Die Klassen müssen vorher vom Mensch festgelegt werden. Der Computer klassifiziert die Datensätze dann anhand der Attribute (sogenannten *features*) des entsprechenden Datensatzes. Die klassifizierten Datensätze enthalten dann meistens eine weitere Dimension, in der die Klasse anhand einer Zahl gespeichert ist.

**Regression.** Bei der Regression geht es darum, den entsprechenden Datensätzen am Ende einen festen Wert zuzuordnen. Der Unterschied zur Klassifikation besteht darin, dass der zugeordnete Wert nicht für eine Klasse steht, sondern eine konkrete reelle Zahl beinhaltet. Ein Beispiel wäre der Preis von einer Wohnung. Diese wurde mit vielen verscheidenen Attributen in die Datenbank eingespeichert (zum Beispiel Größe, Lage, ob sie von einem) der Endwert wäre dann zum Beispiel der konkrete Preis und nicht eine Klassifizierung in “teuer” , “mittelteuer”, “billig”.

### 1.2 Support Vector Machines

Support Vector Machines oder kurz SVMs sind ein Modell im Machine Learning. Dabei werden für alle Datensätze zuerst gewisse Features festgelegt. Diese Features entsprechen den Attributen des Datensatzes. Je nach Problem kann es unterschiedlich viele Features geben, doch allgemein kann man sagen, dass mit der Anzahl der Features auch die Genauigkeit des Programms wächst.

Diese Features werden normalerweise in einem Vektor $x_i \in \mathbb{R}^m$ für den entsprechenden Datensatz $i = 1, \ldots, n$ gespeichert. Zusätzlich hat jeder Datensatz noch einen weiteren Wert $y_i \in \mathbb{R}$, welcher das “Ergebnis” oder “Label” des Datensatzes beschreibt. Meist betrachtet man zwei Klassen $y_i \in \{-1, +1\}$.

Durch die Features ist es dann möglich den Datensätzen einen bestimmten Vektor im $m$-dimensionalen Koordinatensystem zuzuordnen. Diese können dann auf Grund ihrer räumlichen Anordnung klassifiziert werden. Dabei soll der Abstand von beiden Gruppen von Punkten zu dem Trennobjekt maximal sein. Dadurch kann die Klassifizierung optimal durchgeführt werden. Andernfalls würden schon geringe Abweichungen von den Test-Datensätzen reichen, das ein Datensatz falsch klassifiziert würde (siehe Abb. 1.2).

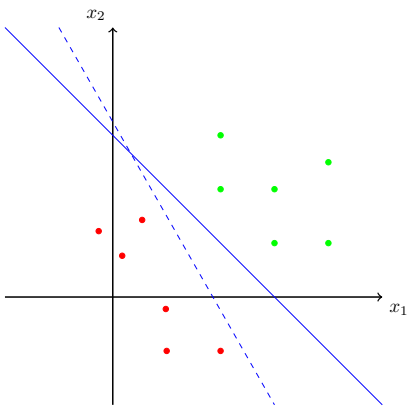


*Abb. 2.1: SVM mit verschiedenen möglichen Trenngeraden, wobei nur die durchgezogene Linie optimal ist.*

Für die mathematische Formulierung des zugehörigen Optimierungsproblems stellen wir zunächst fest dass $2/\|w\|$ der Abstand der Ebenen $\langle w, x_i\rangle - b = 1$ und $\langle w, x_i\rangle - b = -1$ ist (siehe Abb. 2.2). Man nennt diesen Abstand auch den *margin* der Klassifikationsebene $\langle w, x\rangle - b = 0$, $1/\|w\|$ ist der maximal Abstand der Testdatenpunkte von der Ebene. Das folgende Programm berechnet somit

eine Klassifikationsebene mit bester Trennung:

$$\begin{aligned} \underset{w,b}{\text{maximiere}} \quad & \frac{2}{\|w\|} \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 \quad \forall i = 1, \ldots, n \end{aligned}$$

Dieses Problem is äquivalent zu

$$\begin{aligned} \underset{w,b}{\text{minimiere}} \quad & \|w\| \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 \quad \forall i = 1, \ldots, n \end{aligned}$$

Die Klassifikation neuer Datenpunkte erfolgt dann nach der Regel

$$y_i = \begin{cases} 1 & \text{falls} \quad \langle w, x_i\rangle - b \geq 0 \\ -1 & \text{falls} \quad \langle w, x_i\rangle - b \leq 0 \end{cases}$$

Nachdem das Problem optimiert wurde, ist der Computer in der Lage, weitere Daten einzuordnen. Dennoch kann es bei dieser Art von SVMs zu einem Problemen kommen: Wenn die Datensätze nicht linear separierbar sind, das heißt es ist nicht möglich, die beiden Datensätze mit einem linearen Element zu trennen. Dann ist obiges Optimierungsproblem nicht lösbar.

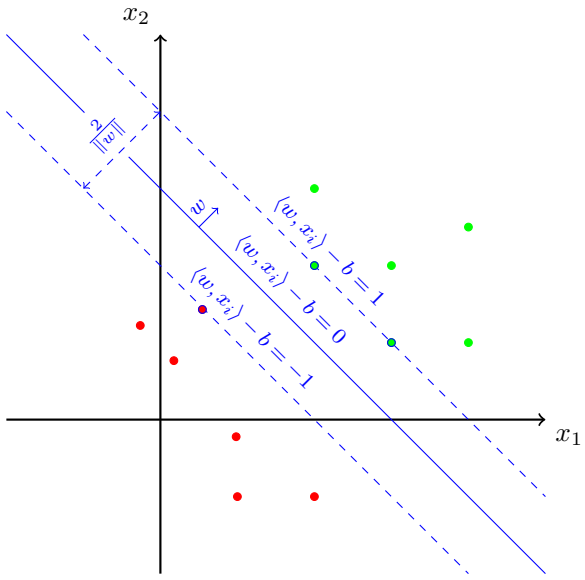


*Abb. 2.2: Links und rechts zur Trenngeraden mit dem Normalenvektor $w$ befinden sich die parallelen Grenzen (gestrichelte Geraden).*

**Soft Margin SVM.** Die Soft Margin SVMs können mit diesem Problem umgehen. Sie sind im Prinzip wie herkömmliche SVMs aufgebaut, nur dass sie eine gewisse Fehlertoleranz besitzen. Diese kommt durch die folgende Modifikation des Optimierungsproblems zustande:

$$\begin{aligned} \underset{w,b}{\text{minimiere}} \quad & \|w\| + C\sum_i z_i \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 - z_i \quad \forall i = 1, ..., n \end{aligned}$$

Dabei steht $z_i$ für die Größe des Fehlers des Punktes $x_i$ und $C$ ist eine Konstante, deren Größe bestimmt, wie stark die Fehler gewichtet werden. Da die gesamte Gleichung minimiert werden soll sorgt ein hohes $C$ dafür, dass das Problem schwerer optimiert werden kann. Die Konstante muss vom Mensch gewählt werden, je nachdem, wie schwer Fehler gewichtet werden sollen. Um die optimale Größe von $C$ für das entsprechende Problem zu finden wird ein Verfahren namens *crossvalidation* genutzt. Dabei wird der Algorithmus mehrmals auf den gleichen Testdatensatz angewendet, $C$ dabei aber variiert. Damit kann man herausfinden, bei welchem Wert von $C$ der Algorithmus optimal funktioniert.

# 2 Neuronale Netze

## 2.1 Einführung

Inspiriert von unserem Verständnis über das menschliche Gehirn benutzen neuronale Netze Lernalgorithmen, welche besonders für praktische Anwendungen geeignet sind. Dazu zählen Spracherkennung, Objekterkennung in Bildern und die Fähigkeit individuell passende Produkte vorzuschlagen, die dem Kunden gefallen könnten. Ein neuronales Netz ist aus mehreren Schichten aufgebaut. Ausgangsschicht ist dabei, die Datenschicht, auf die ein oder mehrere sogenannte *hidden layer* folgen. Als Ausgabewert erhält man schließlich einen Vektor, welcher eine Wahrscheinlichkeitsverteilung darstellt, die angibt wie wahrscheinlich es ist, dass das Ausgangsobjekt zu einer bestimmten Klasse gehört.

## 2.2 Deep Learning

Mit Deep Learning beschreibt man neuronale Netze, die über mehr als eine "versteckte Schicht" verfügen. Damit ist gemeint, dass sich zwischen Ein- und Ausgabeschicht weitere Schichten befinden.

Um die Anzahl an Parametern zu vergrößern, werden diese auch über nicht-lineare Funktionen miteinander verknüpft (die Verkettung von linearen Funktion wäre wieder linear und besäße deshalb nur begrenzte Representationskraft). Jedoch kann eine zu hohe Anzahl an Parametern auch dazu führen, dass das Netzwerk "overfitted", das heißt zu sehr an den Trainingsdatensatz angepasst wird.

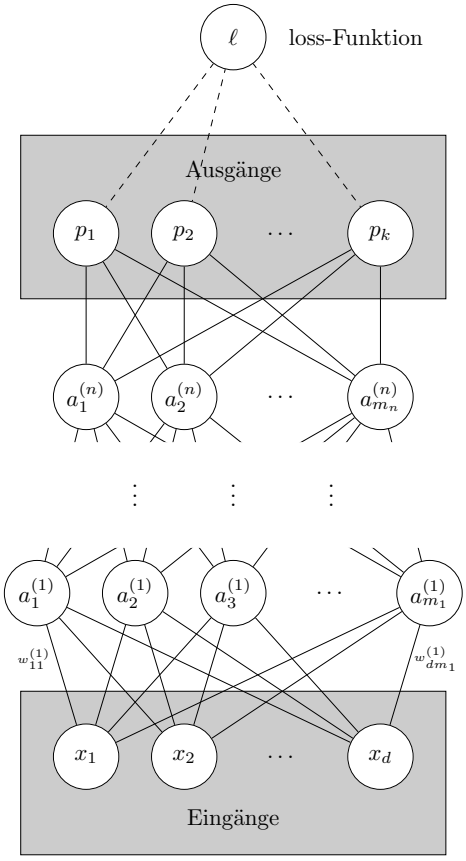


*Abb. 2.3: Ein vollständig verbundenes neuronales Netzwerk mit d Eingängen und k Ausgängen, bestehend aus n Schichten mit jeweils $m_n$ "Neuronen", wobei $m_n$ für die Anzahl an Neuronen der n-ten Schicht steht.*

Dann kann das Netzwerk neue Daten nicht mehr korrekt klassifizieren.

Wie wir in der Abbildung 2.3 sehen können ist ein typisches *fully connected* neuronales Netz abgebildet. Als Basis findet man unten die Datenschicht mit ihren Eingabewerten in Form eines Vektors $(x_1 \dots x_i)$. Diese Werte werden nun mit einem jeweiligen Faktor $w$ multipliziert. Hierbei handelt es sich um ein Skalarprodukt. Das Ergebnis wird nun im ersten hidden layer abgespeichert. Darauf wird eine nicht-lineare Funktion angewendet. Für die $k$-te hidden layer bekommen wir

$$\begin{aligned} a_j^{(k)} &= f^{(k)}(\langle w_j^{(k)}, a^{(k-1)} \rangle) \\ &= f^{(k)}\left( \sum_{i=1}^{m^{(k-1)}} w_{ji}^{(k)}, a_i^{(k-1)} \right) \end{aligned}$$

Das Ergebnis wird erneut mit einem Faktor $w$ multipliziert. Dieser Vorgang wiederholt sich so lange bis uns schließlich ein Vektor mit seinen Komponenten $(p_1 \dots p_k)$ zurückgegeben wird, welcher als Wahrscheinlichkeitsverteilung interpretiert werden kann.

Anhand eines konkreten Beispiels würde es folgendes bedeuten: Nehmen wir an wir haben ein Bild und wollen ermitteln zu welcher Klasse Haus, Tisch oder Stuhl das daurauf abgebildete Objekt gehört. Unsere inputs wären demnach die Pixel des Bildes. Diese durchlaufen nun das Neuronale Netz und wir erhalten eine Wahrscheinlichkeitsverteilung als Rückgabewert, welche uns mitteilt, dass es am wahrscheinlichsten ist, dass das abgebildete Objekt ein Objekt der Klasse Stuhl ist. Demnach ist die Klassifizierung vollzogen.

## 2.3 Convolutional Neural Networks

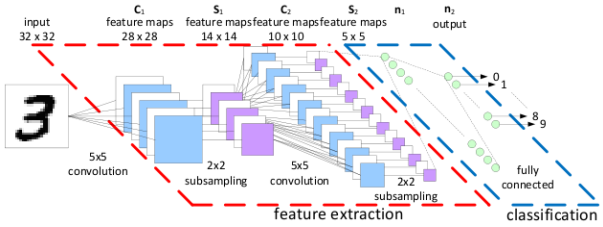


*Abb. 2.4: Ein Convolutional Neural Network (CNN) (deutsch "faltendes neurales Netzwerk")*

Bei *convolutional networks* handelt es sich um spezielle neuronale Netze, die besonders für die Bilderkennung geeignet sind.

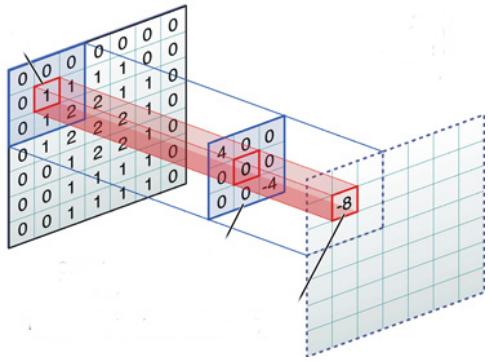

*Abb. 2.5: Ein Convolutional NN mit einer Schicht aus neun "Neuronen" dazwischen, die als Filter wirkt.*

Wie in Fig. 2.6 gezeigt wird der existierende Filter $w$ über die Eingabematrix geschoben. Dabei wird das Skalarprodukt berechnet und in der Endmatrix als Ergebnis festgehalten. Daraufhin wird der Filter immer um eine Stelle weiter nach rechts in der Eingabematrix verschoben und das neue Skalarprodukt berechnet bis das Ende der Eingabematrix erreicht wurde.

## 2.4 SVMs als Neuronale Netze

Wir haben bereits SVMs kennengelernt, die beschrieben werden durch das Optimierungsproblem

$$\begin{aligned} \underset{b,w,z}{\text{minimiere}} \quad & \|w\| + C\sum_{i=1}^{n} z_i \\ \text{sodass} \quad & y_i(\langle w, x\rangle - b) \geq 1 - z_i \text{ mit } z_i \geq 0 \end{aligned}$$

Die Nebenbedingung lässt sich umschreiben als

$$z_i \geq 1 - y_i(\langle w, x\rangle - b), z_i \geq 0$$

Da über $z_i$ minimiert wird und $C$ positiv ist, nehmen die Variablen $z_i$ immer die Grenzen an, wir bekommen also

$$\begin{aligned} &\min \|w\| + C \cdot \sum_{i=1}^{n} \max(0, 1 - y_i(\langle w, x_i\rangle - b) \\ &= \|w\| + C\sum_{i=1}^{n} \ell(1 - y_i(\langle w, x_i\rangle) - b) \\ &= \|w\| + C\sum_{i=1}^{n} \ell(1 - y_i a_i) \end{aligned}$$

Charakteristisch für dieses Neuronale Netz ist, dass es nur einen Layer besitzt.

## 2.5 Lineare Regression

Analog kann auch die lineare Regression in Form von Neuronalen Netzen ausgedrückt werden:

$$\min_{w,b} \frac{1}{2}\sum_{i=1}^{n} (a_i - y_i)^2, \quad a_i = \langle w, x_i\rangle - b$$

## 2.6 MNIST

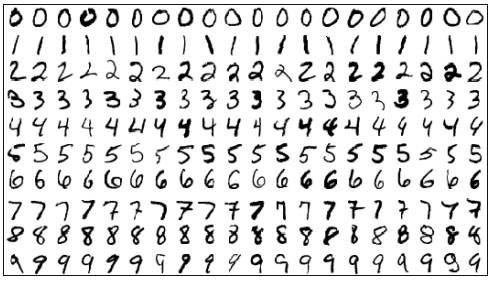

*Abb. 2.6: mnist Example*

MNIST ist ein freier Datensatz, der es erlaubt, SVMs und Neuronalen Netze zur Zeichenerkennung zu trainieren. Wir haben den Datensatz im Kurs verwendet, um unsere Modelle zu trainieren.