Contents

Machine Learning

SVMs und Neuronale Netzwerke

1 Machine Learning

Machine Learning ist ein Konzept, bei dem der Computer durch Algorithmen in der Lage ist Probleme selbstständig zu lösen. Dabei erkennt der Computer nach einer gewissen Lernphase Gesetzmässigkeiten, durch die er dann in der Lage ist, neue, ihm unbekannte Datensätze bezüglich eines bestimmten Problems zu lösen.

Dass der Computer die Gesetzmässigkeiten erkennen kann, werden verschiedene Systeme genutzt, doch zunächst einmal sollten wir uns mit den Begriffen Klassifikation und Regression vertraut machen.

1.1 Klassifikation und Regression

Bei Klassifikation und Regression handelt es sich um zwei verscheidene Arten von Problemen, beziehungsweise Möglichkeiten zur Problemlösung im Bereich Machine Learning.

Klassifikation Bei der Klassifikation sorgt der Algorithmus am Ende dafür, dass der Datensatz klassifiziert wird. Das heißt, der Datensatz wird vom Computer in verschiedene Klassen eingeteilt. Die Klassen müssen vorher vom Mensch entschieden werden. Der Computer klassifiziert die Datensätze dann anhand der Attribute des entsprechenden Datensatzes. Die klassifizierten Datensätze enthalten dann meistens eine weitere Dimension, in der die Klassifizierung 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 ein konkretes Ergebnis beinhaltet. Ein Beispiel wäre der Preis von einer Wohnung. Diese wurde mit vielen verscheidenen Attributen in die Datenbank eingespeichert (z. B. Größe, Lage ...) 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 Konzept zum Lösen von Machine Learning Problemen. 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, je mehr Features es sind die Genauigkeit des Programms genauer wird.

Diese Features werden normalerweise in einem Vektor für den entsprechenden Datensatz gespeichert. Dieser Vektor wird x_i genannt wobei $i = 1, ..., n$ den Datensatz beschreibt mit $x \in \mathbb{R}^m$. Zusätzlich hat jeder Datensatz noch einen weiteren Wert, welcher das »Ergebnis« des Datensatzes beschreibt, der $y_i \in \mathbb{R}$ oder auch »Label« genannt wird.

Die Features und das Label zusammen beschreiben einen Punkt im n-dimensionalen Koordinatensystem, wobei $n = m + 1$, da zu den m Dimensionen von x noch die eine Dimension von y dazukommt.

Durch die Features ist es dann möglich den Datensätzen einen bestimmten Ort im n-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).

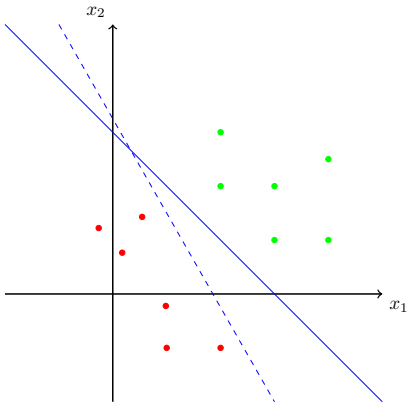

Abb. 2.1: Da der Abstand zwischen den unterschiedlich klassifizierten Datensätzen maximiert werden soll, gilt die durchgezogene und nicht die gestrichelte Linie als Trennelement.

Um die Trennlinie zu optimieren, gibt es auch eine mathematische Beschreibung. Ebenen kann man auch in der Form $\langle x, w\rangle = b$ beschreiben. Daraus folgt, dass das Optimierungsproblem wie folgt geschreiben werden kann.

$$\begin{aligned} \max_{w,b} \quad & \frac{2}{\|w\|} \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 \quad \forall i = 1, ..., n; \\ & x \in \mathbb{R}^n; \quad y \in \{1; -1\} \end{aligned}$$

Dabei wird der Abstand zwischen den innersten Ebenen durch die beiden Datensätze ($\frac{2}{\|w\|}$) (Siehe Abb. 1.2).

Unter der Vorraussetzung, dass alle vorhandenen Datensätze richtig klassifiziert sind. Dieses Problem kann man aber genauso schreiben als:

$$\begin{aligned} \underset{w,b}{\text{minimiere}} \quad & \|w\| \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 \quad \forall i = 1, ..., n; \\ & x \in \mathbb{R}^n; \quad y \in \{1; -1\} \end{aligned}$$

Klassifikation:

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

Nachdem das Problem optimiert wurde, ist der Computer in der Lage, weitere Daten einzuordnen, vorrausgesetzt, es ist richtig optimiert. Dennoch kann es bei dieser Art von SVMs zu einigen Problemen kommen.

- Wenn die Datensätze nicht linear separierbar sind, d.h., es ist nicht möglich die beiden Datensätze mit einem linearen Element zu trennen.
- Wenn die Daten nicht alle korrekt klassifiziert sind oder es Daten gibt, die in dem »Bereich« der anderen Seite liegen.

Soft Margin SVMs Die Soft Margin SVMs können mit beiden Problemen umgehen. Sie sind im Prinzip wie herkömmliche SVMs aufgebaut, nur dass sie eine gewisse Fehlertoleranz besitzen. Diese kommt durch eine Veränderung an der Formel zustande:

$$\begin{aligned} \underset{w,b}{\text{minimiere}} \quad & \|w\| + C\sum_i z_i \quad \forall i = 1, ..., n \\ \text{sodass} \quad & y_i(\langle w, x_i\rangle - b) \geq 1 - z_i \quad \forall i = 1, ..., n; \\ & x \in \mathbb{R}^n; \quad y \in \{1; -1\} \quad z_i \geq 0 \end{aligned}$$

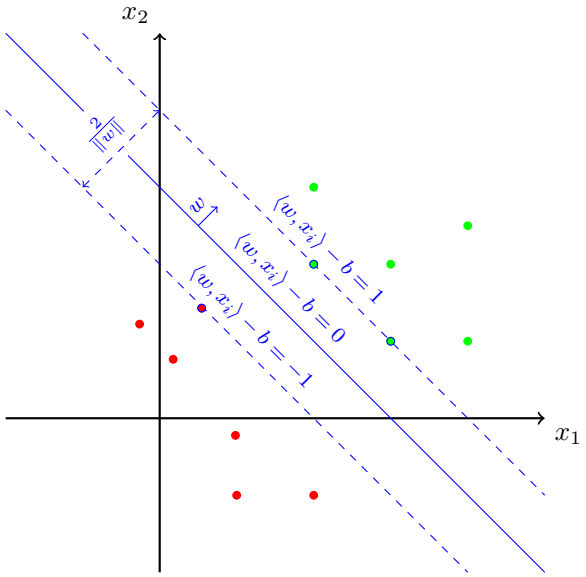

Abb. 2.2: Links und rechts zur Trenngeraden befinden sich die parallelen Grenzen (gestrichelte Geraden). Ziel der Optimierung ist es, den Abstand zwischen den Grenzen zu maximieren, um den Normalenvektor w zu bestimmen.

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 Konstante mit der Summe der Fehler multipliziert wird. Da die gesamte Gleichung aber minimiert werden soll sorgt ein hohes C dafür, dass das Problem schwerer optimiert werden kann. Die Konstante muss vom Mensch selbst gewählt werden, je nachdem, wie schwer Fehler gewichtet werden sollen. Um die optimale Größe von C für das entsprechende Problem zu finden wird ein System namens Crossvalidation genutzt. Bei diesem System 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, wie das menschliche Gehirn lernt, 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 wird von mehreren Schichten

aufgebaut. Ausgangsschicht ist dabei, die Datenschicht, auf die ein oder mehrere hidden layer folgen. Als Ausgabewert erhält man schließlich einen Vektor, welcher die Wahrscheinlichkeitsverteilung darstellt. Mit anderen Worten, wie wahrscheinlich es ist, dass das Ausgangsobjekt zu einer bestimmten Klasse gehört.

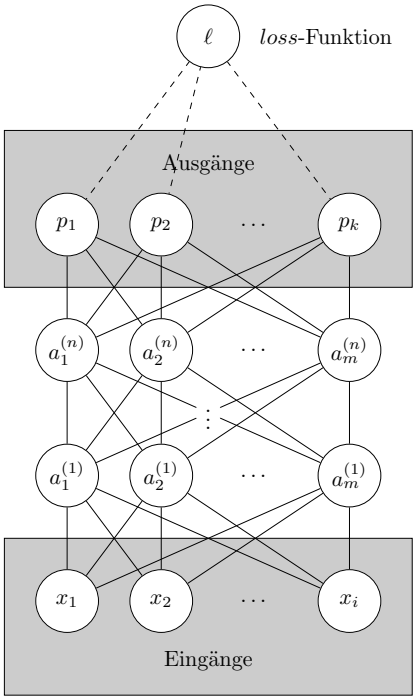

Abb. 2.3: Ein vollständig verbundenes neuronales Netzwerk mit i Eingängen und k Ausgängen, bestehend aus n Schichten mit jeweils m »Neuronen«.

2.2 Deep Learning

Mit Deep Learning beschreibt man die Neuronalen Netze, die über mehr als einen »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. Dadurch kann eine höhere Genauigkeit erzielt werden. Jedoch kann eine zu hohe Anzahl an Parametern auch dazu führen, dass das Netzwerk »overfitted«, d.h., zu sehr an den Trainingsdatensatz angepasst, wird. Dann kann das Netzwerk neue, fremde Daten nicht mehr korrekt klassifizieren.

Wie wir in der Abbildung 1.3 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 und das Ganze 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.

$$a_j^{(k)} = f^{(k)}(\langle w_j^{(k)}, a_i^{(k-1)} \rangle) = f^{(k)}(\sum_{i=1}^{m^{(k-1)}} w_{ji}^{(k)}, a_i^{(k-1)})$$

2.3 Convolutional Neural Networks

Abb. 2.4: Ein Convolutional Neural Network (CNN) (dt.: »faltendes neurales Netzwerk«)

Bei den Convolutional Networks handelt es sich um ein Neuronales Netz mit einer vereinfachten Struktur. Diese kommt dadurch zu Stande, dass nun kein fully connected Neuronales Netzwerk mehr vorliegt.

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

Der existierende Filter w wird auf die Eingabematrix angewendet. 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 durch das folgende Optimierungsproblem beschrieben werden:

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

Diese Nebenbedingung lässt sich umschreiben als:

$$\begin{aligned} & y_i(\langle w, x\rangle - b) \geq 1 - z_i \text{ mit } z_i \geq 0 \\ \Rightarrow & z_i \geq 1 - y_i(\langle w, x\rangle - b) \text{ mit } z_i \geq 0 \end{aligned}$$

Da über z_i minimiert wird und C positiv ist, nehmen die Variablen C immer die Grenzen an.

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

Charakteristisch für dieses Neuronale Netz ist, dass es nur einen Layer besitzt. Im folgenden wird verdeutlicht, wie man diese auch als Neuronales Netz ausdrücken kann.

2.5 Lineare Regression

Die lineare Regression kann auch in Form von Neuronalen Netzen ausgedrückt werden.

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

2.6 MNIST

Abb. 2.6: mnist Example

Das MNIST bietet Daten zur freien Verfügung, damit man seine SVMs und Neuronalen Netze mit den Daten trainieren kann. Mit diesen MNIST Trainingsets haben wir uns im Kurs beschäftigt, um unsere maschinellen Lernalgorithmen zu trainieren, damit sie am Ende Vorhersagen treffen können.