Einführung in die Künstliche Intelligenz

Übungszettel 6

Prof. Dr. Claudia Schon

C.Schon@hochschule-trier.de

Fachbereich Informatik
Hochschule Trier

1 Perzeptrontraining und lineare Separierbarkeit[1]

In dieser Aufgabe soll überprüft werden, ob ein einfaches Perzeptron eine bestimmte boolesche Funktion mit zwei Eingaben korrekt lernen kann. Die Wahrheitstabelle lautet:

x_1	x_2	$x_2 \to x_1$
0	0	1
0	1	0
1	0	0
1	1	1

Wir erweitern den Eingabevektor um einen zusätzlichen konstanten Wert $x_0 = 1$. Das Gewicht w_0 ist der Bias des Perzeptrons. Der vollständige (erweiterte) Eingabevektor ist also $\vec{x} = (x_0, x_1, x_2)$. Der Gewichtsvektor hat die Komponenten w_0, w_1 und w_2. Die Eingabedaten aus der obigen Wahrheitstabelle, ergänzt um $x_0 = 1$, sollen als Trainingsdaten verwendet werden.

a) Trainieren Sie mit diesen Daten ein Perzeptron.

Verwenden Sie die Lernrate $\eta = 1$ sowie die folgenden Startgewichte:

$$w_0 = 0; \quad w_1 = 0{,}5; \quad w_2 = -0{,}5$$

In der folgenden Tabelle ist der erste Trainingsschritt schon durchgeführt worden. Tragen Sie den weiteren Trainingsverlauf in der Tabelle ein:

	Erw. Eingabe $(x_0; x_1; x_2)$	Gewichte $(w_0; w_1; w_2)$	$x \cdot w$	Ausgabe y	Zielausgabe t	$t - y$	Neue Gewichte
1	(1; 0; 0)	(0; 0,5; −0,5)	0	0	1	1	(1,0; 0,5; −0,5)
2	(1; 0; 1)	(1,0; 0,5; −0,5)					
3	(1; 1; 0)						
4	(1; 1; 1)						
1	(1; 0; 0)						
2	(1; 0; 1)						
3	(1; 1; 0)						
4	(1; 1; 1)						

b) Was beobachten Sie?

c) Was sagt Ihnen das über die Lernfähigkeit eines einfachen Perzeptrons bei dieser Funktion?

[1] Die Aufgabe wurde mit Unterstützung von ChatGPT erstellt.

d) Warum ist eine lineare Trennung der Klassen in diesem Fall nicht möglich? Erklären Sie mit Hilfe einer Skizze.

2 Klassifikation von Pinguinen mit neuronalen Netzen

In dieser Übung wollen wir ein neuronales Netz zur Klassifikation von Pinguinarten trainieren. Dazu verwenden wir das sogenannte **Penguin-Dataset**, das Messwerte von drei verschiedenen Pinguinarten enthält (z. B. Körpermasse, Flipperlänge, Schnabellänge, Insel, Geschlecht usw.).

- Die Daten können von **Stud.IP** heruntergeladen werden: `penguin_size.csv`.
- Zudem steht dort das Python-Programm zur Klassifikation des Iris-Datasets zur Verfügung, das wir in der Vorlesung betrachtet haben: `iris.zip` (beachten Sie dort bitte die `readme.txt` Datei.)

Wir werden nun Pinguine klassifizieren:

a) Passen Sie das Programm des Iris-Datasets auf das Pinguin-Dataset an.

- Machen Sie sich mit dem Pinguin-Dataset vertraut.
- Da neuronale Netze nur numerische Werte verarbeiten können, müssen alle nicht-numerischen Spalten in numerische Kategorien umgewandelt werden. Dies kann analog zu den Zeilen 24 und 25 im Iris-Programm erfolgen, oder etwas kürzer durch:

```
data['spaltenname'] = data['spaltenname'].astype('category').cat.codes
```

- Teilen Sie die Daten auf:
  - 20 % Testdaten und
  - 80 % Trainingsdaten, von denen wiederum 20 % für die Validierung verwendet werden.
- Passen Sie das neuronale Netz an:
  - Passen Sie die Anzahl der Eingabeneuronen so an, wie sie im Pinguin-Dataset benötigt wird.
  - Wählen Sie nur eine versteckte Schicht mit 10 Neuronen und `sigmoid` als Aktivierungsfunktion.
  - Passen Sie die Anzahl der Neuronen in der Ausgabeschicht auf die Anzahl der Klassen an, die das Zielattribut im Pinguin-Dataset hat. Wählen Sie als Aktivierungsfunktion `softmax`.
  - Die restlichen Einstellungen (Optimizer, Loss, Callback usw.) können wie im Iris-Beispiel übernommen werden
- Trainieren Sie das Netz, erstellen Sie mit dem Programm eine Grafik des *Validation Loss* und geben Sie die *Test Accuracy* aus. Wie gut generalisiert Ihr neuronales Netz?

3 Skalierungsmethoden

In der vorherigen Aufgaben haben Sie vermutlich festgestellt, dass die Ergebnisse nicht zufriedenstellend sind. Dies kann daran liegen, dass die Werte der Merkmale unterschiedliche Größenordnungen haben (z. B. `body_mass_g` im Tausenderbereich, während `bill_depth_mm` im Zehnerbereich liegt). Neuronale Netze reagieren sensibel auf die Größenordnungen der Eingabewerte. In dieser Aufgabe wollen wir zwei Verfahren zur Skalierung von Daten kennenlernen und ausprobieren: **Min-Max-Skalierung** und **Standardisierung (Z-Transformation)**.

a) Min-Max-Skalierung

Bei der Min-Max-Skalierung werden die Werte eines Attributs so transformiert, dass sie in einem festen Bereich liegen – typischerweise im Intervall $[0, 1]$. Dies geschieht mit folgender Formel:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- x ist der ursprüngliche Wert,
- $x_{\min}$ ist der kleinste Wert des Attributs und
- $x_{\max}$ ist der größte Wert des Attributs.

In der folgenden Tabelle sehen Sie einige Attribute aus dem Pinguin-Dataset mit Werten. Berechnen Sie für die Attribute in der Tabelle die Min-Max-skalierten Werte.

bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
39.1	18.7	181	3750
38.8	17.2	180	3800
36.3	19.5	190	3800
38.2	20.0	190	3900
39.0	17.1	191	3050
43.3	14.0	208	4575

b) Anwendung im der Min-Max-Skalierung Programm:

Fügen Sie in Ihrem Python-Programm aus der vorherigen Aufgabe die folgende Min-Max-Skalierung ein:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)
```

Hinweis: Diese Skalierung soll direkt nach der Aufteilung in Trainings-, Validierungs- und Testdaten erfolgen.

- Führen Sie das Training erneut durch.
- Beobachten Sie den Verlauf des Validierungsfehlers (Validation Loss) und die Testgenauigkeit.
- Verbessert sich das Ergebnis durch diese Art der Skalierung?

c) Standardisierung (Z-Transformation)

Bei der Standardisierung wird jeder Wert eines Attributs wie folgt transformiert:

$$x' = \frac{x - \mu}{\sigma}$$

Dabei ist:

- x der ursprüngliche Wert,
- μ der Mittelwert des Attributs (aus dem Datensatz) und
- σ die Standardabweichung (aus dem Datensatz).

Für ein Attribut mit n Werten $x_1, x_2, \ldots, x_n$ berechnen sich:

- **Mittelwert**:
$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$
- **Standardabweichung** (empirisch, aus einer Stichprobe geschätzt):
$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)^2}$$

Hinweis: Da wir in der Regel nur eine *Stichprobe* aus einer größeren Grundgesamtheit betrachten (z. B. 6 Pinguine statt aller Pinguine weltweit), kennen wir den wahren Mittelwert der Population nicht. Wir verwenden stattdessen den Stichprobenmittelwert μ, der näher an den beobachteten Werten liegt. Um die dadurch entstehende *systematische Unterschätzung der Streuung* auszugleichen, teilen wir beim Berechnen der Standardabweichung durch $n - 1$ statt durch n.

Berechnen Sie den Mittelwert μ und die Standardabweichung σ für jede Spalte in der obigen Tabelle. Berechnen Sie dann die standardisierten Werte x' für alle Zellen.

d) Anwendung der Standardisierung im Programm:

Verwenden Sie wieder den folgenden Code in Ihrem Programm:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)
```

Hinweis: Auch hier sollte die Skalierung direkt nach der Aufteilung in Trainings-, Validierungs- und Testdaten erfolgen.

Führen Sie das Training mit dieser Skalierung erneut durch und vergleichen Sie die Ergebnisse mit jenen aus der vorherigen Aufgabe und Teil b) dieser Aufgabe. Beobachten Sie insbesondere den Validation Loss und die Test Accuracy.

e) Warum führen wir die Skalierung erst **nach der Aufteilung** in Trainings-, Validierungs- und Testdaten durch und nicht direkt nach dem Einlesen des gesamten Datensatzes?