KIP 23/24 - Praktikum 2: Supervised Learning

Rahmenbedingungen

Programmiersprache: Python.
Persönliche Empfehlung: PyCharm als IDE verwenden, andere Optionen wie z.B. Jupyter Notebooks aber auch legitim. Lösung, wenn z.B. kein Laptop vorhanden ist, welcher zum Praktikumstermin mitgebracht werden kann: Jupyter Notebooks auf Laborrechnern im 11. Stock oder Google Collab → Jupyter Notebooks in Google Cloud.
Implementierung: In den Aufgaben werden einige "Grundpfeiler" für die Implementierungen vorgegeben, davon ab werden euch jedoch sehr viele Freiheiten gelassen. Grundprinzipien aus Programmieren 1 und 2 und Software Engineering sollten natürlich auch in KIP Anwendung finden, so dass die Implementierung gut "lesbar", wartbar, erweiterbar etc. sein sollte.
LLM Policy: Die Verwendung von z.B. ChatGPT ist grundsätzlich erlaubt. Falls ihr dieses verwenden solltet, dokumentiert grob eure Prompts, wie gut das geklappt hat, wo ihr Verbesserungen vorgenommen habt und ob ihr meint, dadurch einen Zeitgewinn erhalten zu haben.
Persönliche Einschätzung: Der Zeitgewinn wird eher gering ausfallen, da der generierte Code auch verstanden und häufig verbessert werden muss. Dadurch, dass eben diese Konzepte auch verstanden und vermittelt werden müssen für die Abgabe (s.u.), bietet es sich für den Lernprozess ggf. eher an, den Code von Grund auf selbst zu implementieren.
Abnahme:
Grundsätzlich gilt: Beide Teammitglieder müssen sowohl den Code als auch die relevanten Konzepte vollständig erklären können. **Die Aufgaben sollten bis zu dem Praktikumstermin fertig sein und nicht erst im Praktikumstermin bearbeitet werden**.
Der generelle Ablauf ist wie folgt: Idealerweise redet ihr den größten Teil der Zeit über eure Aufgabenlösungen, führt grob durch den Code und stellt die dahinterstehenden Konzepte vor und beantwortet die Teilfragen der einzelnen Aufgaben, sodass nur noch wenige Rückfragen bleiben. Zwischendurch und danach werde ich ggf. einige Fragen zur Implementierung und den dazugehörigen Konzepten der Vorlesung stellen. Idealerweise dauert dieser Prozess etwa 20 Minuten - bereitet euch also vor!

Aufgabe 1: Minimalbeispiel Gradient Descent

Für diese Aufgabe giltt:
Die Verwendung von fertigen Implementierungen der kompletten Algorithmen, importiert aus Packages, ist nicht erlaubt. Die Implementierung soll generell mit möglichst wenigen Imports auskommen (also eine weitgehend eigenständige Implementierung der Algorithmen darstellen).

In dieser Aufgabe soll ein Minimalbeispiel für das Gradientenabstiegsverfahren für das *Fitten* einer Funktion auf einem Datensatz erstellt werden. Der Datensatz besteht für dieses Beispiel aus nur einem Sample, nämlich:
$X = (1.0, 1.5)$ und $y_{tar} = 2.0$.

Die Funktion, welche dieses Sample approximieren soll, d.h. gefittet wird, lautet: $y=\sin(w_1 \cdot x_1)+\cos(w_2 \cdot x_2)+w_2$ mit den lernbaren Parametern w_1 und w_2.

Auch wenn diese Funktion komplizierter aussieht als die lineare Regression, an welcher der Gradientenabstieg in der Vorlesung vorgestellt wurde, ändert sich am Grundprinzip des Gradientenabstieges nichts.

1. Definition einer Fehlerfunktion E

Der Abstand zwischen der aktuellen Vorhersage des Modells und den Soll-Werten des "Datensatzes" (d.h. des einen Samples) wird über eine Fehlerfunktion E ermittelt und soll im Rahmen des Gradient Descent Verfahrens minimiert werden. Für dieses Beispiel soll MSE als Fehlerfunktion verwendet werden:

$$E = \frac{1}{2}(y - y_{tar})^2 = \frac{1}{2}(sin(w_1 x_1) + cos(w_2 x_2) + w_2 - y_{tar})^2$$

Visualisiert die Fehlerfunktion, z.b. mit Matplotlib und dem folgendem Codesnippet, wobei die Funktion *function* noch leicht angepasst werden muss:

```
import matplotlib.pyplot as plt
import numpy as np

def loss_function(w1, w2, x1=1.0, x2=1.5, y_tar=2.0):
   0.5*(w1*x1+w2*x2-y_tar)**2 #Fehlerfunktion Lineare Regression,
ersetze durch Funktion und Datensample dieser Aufgabe

def viz(current_w1, current_w2, current_loss):
   x = np.linspace(-10, 10, 100) #Der fuer diese Aufgabe
interessante Definitionsbereich liegt zwischen -10 und 10
   y = np.linspace(-10, 10, 100)
   X, Y = np.meshgrid(x, y)
   Z = loss_function(X,Y)
   fig = plt.figure(figsize = (10,7))

   ax = plt.axes(projection='3d')
   ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
   cmap='jet', edgecolor='none')
   ax.plot(current_w1, current_w2, current_loss, marker="o",
markersize=10, markeredgecolor="black", markerfacecolor="white")
   ax.set_title("Fehlergebirge", fontsize = 13)
   ax.set_xlabel('w1', fontsize = 11)
   ax.set_ylabel('w2', fontsize = 11)
   ax.set_zlabel('E', fontsize = 11)
   plt.show()
```

Achtung: Visualisiert werden soll die Fehlerfunktion, welche ja minimiert werden soll, in Abhängigkeit von den Gewichten, also $E(w_1, w_2)$ und nicht das Modell $y(x_1, x_2)$. D.h. die drei visualisierten Achsen entsprechen E, w_1 und w_2 (x_1, x_2 und y_{tar} sind konstant, definiert durch das oben gegebene Datensample).

Der Funktion *viz* werden die Parameter current_w1, current_w2 und current_loss übergeben. An die Stelle der hier übergebenen 3 Koordinaten wird ein Punkt gezeichnet, mithilfe dessen später der Verlauf der Gewichtsanpassungen verfolgt werden kann.

2. Gradient der Fehlerfunktion ermitteln
Um schrittweise das Minimum der Funktion zu ermitteln, sind die partiellen Ableitungen nach den lernbaren Gewichten notwendig (***warum?***).
Ermittelt also:
$\frac{\partial E}{\partial w_1}$ und $\frac{\partial E}{\partial w_2}$. Dies könnt ihr z.B. entweder händisch machen oder unter Zuhilfenahme von z.B. Wolfram Alpha (oder ChatGPT) oder z.B. mit Sympy (https://www.askpython.com/python/examples/derivatives-in-python-sympy)

3. Schrittweise optimieren
Nähert euch schrittweise dem/einem Minimum, indem ihr iterativ kleine Schritte der Größe $\alpha = 0.05$ macht:

$$w_{1,neu} = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$w_{2,neu} = w_2 - \alpha \frac{\partial E}{\partial w_2}$$

Macht zwei Durchläufe eures implementierten Verfahrens mit 100 Schritten und den Startwerten:

1) $w_1 = -6.5, w_2 = -9.5$
2) $w_1 = 0.0, w_2 = -0.5$

Wie unterscheiden sich Fehlerwert, Gewichtswerte und Vorhersagen? Visualisiert (s.o.) die Startpositionen und Endpositionen der gefundenen Gewichte in der Fehlerfunktion $E(w_1, w_2)$.

Aufgabe 2: Preprocessing, Training, Evaluation

Für diese Aufgabe gilt:
Die Verwendung von fertigen Implementierungen ist explizit erlaubt :).
Im Praktikumsordner findet Ihr einen Datensatz in der CSV-Datei "Praktikum3_Datensatz.csv". Dieser soll für im Rahmen dieser Aufgabe für das Training einer Logistischen Regression verwendet werden.

1. Einlesen
Lest die Zeilen des Datensatzes ein, z.B. unter Verwendung des csv-Packages von Python. Am Ende sollte eine Liste von Listen mit den Datensamples stehen.

2. Train/Test Split
Teilt den Datensatz in einen Trainings- und einen Testdatensatz auf. Der Trainingsdatensatz soll aus den ersten 80% der Samples bestehen, der Testdatensatz aus den letzten 20%.

3. Preprocessing

Der Datensatz besteht aus 6 Spalten, die ersten 5 ("grundstuecksgroesse", "stadt", "hausgroesse", "kriminalitaetsindex", "baujahr") sollen den Input des Modells darstellen und “klasse” ist die Klasse, welche vorhergesagt werden soll. Von den Inputvariablen sind einige kontinuierlich und einige diskret. Wendet auf die kontinuierlichen eine Standardisierung an (“zero mean, unit variance”/”Z-norm”/”Z-score”) und auf die diskreten ein One-hot-encoding (vgl. Vorlesung).

Wichtig: Die Ermittlung von Mittelwert und Varianz für die Standardisierung erfolgt auf den Trainingsdaten alleine! Für die Testdaten, welche ja beim späteren Testdurchlauf auch standardisiert werden müssen, werden dann die (für die jeweiligen Spalten) auf dem Trainingsdatensatz ermittelten Mittelwert- und Varianzwerte für die Standardisierung verwendet (***warum?***).

Das Feature “baujahr” könnte vor der Standardisierung noch leicht umformuliert/umgerechnet werden, wie könnte dies aussehen und warum?

Außerdem muss auch die Zielvariable binär enkodiert werden.

Am Ende sollte X_train, eine Liste von Listen mit den Eingabewerten und y_train, eine Liste mit den zugehörigen Ausgabewerten, stehen. Analog für X_test und y_test.

4. Training

Trainiert eine logistische Regression mit dem Trainingsdaten unter Verwendung des Frameworks sklearn und der Default-Konfiguration.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Was ist mit dem Parameter “penalty” einstellbar?

5. Evaluation

Bewertet euer Modell auf dem Testdatensatz mit den Metriken

- Accuracy
- Precision
- Recall

Die Berechnung dieser Metriken stellt auch sklearn zur Verfügung! Z.B.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Was ist grundsätzlich die jeweilige Aussagekraft dieser Metriken?