1. Finance Projekt WS21/22

Version: V1.1
Created/changed: 29.10.2021
Status: draft
Autohor: Axel Roth, Aysegül Dursun

In diesem Projekt ist die Aufgabe ein Perceptron (=Künstliches Neuronales Netz KNN) in Python zu Implementieren. Gegeben ist ein Trainingsdatensatz, welches unseren Eingabedaten entspricht, das sieht wie folgt aus:

$$x_1 = (1,0,0), x_2 = (1,0,1), x_3 = (1,1,0), x_4 = (1,1,1)$$

Die Lösung von dem KNN soll $y = (0,1,1,1)$ sein. Jetzt stellt sich die Frage: Wie funktioniert das Perceptron? Die Antowrt ist ganz einfach, die Eingabe Daten kommen in das Perceptron rein und raus kommt ein Output. Sehen wir uns das Perceptron mal näher an. Das Perceptron besteht aus einem Vorwärtsflow und Rückwärtsflow. *Der Vorwärtsflow* Hier startet man mit den Eingabedaten. Dabei wird Zufällig ein Vektor x aus dem Trainingsdatensatz gewählt. Aus dem Vektor$x$ und dem Gewichtsvektor $w_0 = (0,0,0)$, wird die gewichtete Summe s gebildet. Das ist nichts anderes als das Skalarprodukt aus zwei Vektoren. Dann führen wir eine Stufenfunktion ein, diese ist wie folgt definiert: $step_0(s)$=die stufenfunktion Mithilfe der Stufenfunktion wird der Output berechnet. Anschließend wird das Perceptron noch traniert werden, dafür schauen wir uns den Rückwärtsflow an, dieser ist wie folgt: *Der Rückwärtsflow* Basierend auf dem ermittelten Output(=Fehler) werden die Gewichte angepasst und der Vorwärtsflow wird erneut durchlaufen. Die anpassung der Gewichte erfolgt durch folgende Formel:

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

mit

$$\Delta w_i(t) = \alpha * (y_i - \hat{y}_i(t)) * x_{i,j} * \alpha$$

ist die Lernrate, in unserem Fall ist $\alpha = 1$.

Funktionsdefinitionen

```
import numpy as np
import random as ra
import pandas as pd
import matplotlib.pyplot as pyplot
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)


def step(s):
    if( s >= 0 ):
        return(1)
    else:
        return(0)
```

```
def skalarprodukt(w, x):
    return(np.dot(w.transpose(), x))

def backwards(w, x, correct_answer, approx_answer, alpha):
    return(w + alpha * (correct_answer - approx_answer) * x)


def fit(alpha, iterations, training_data_set, wgts):

    df = pd.DataFrame(columns=["iteration", "input_wgts", "approx_answer",
↪   "correct_answer", "error", "new_wgts"])

    for i in range(1, iterations+1):
        random_index = ra.randint(0, len(training_data_set)-1)
        input_wgts = wgts
        input_x = training_data_set[random_index][0]
        correct_answer = training_data_set[random_index][1]

        approx_answer = step(skalarprodukt(w = input_wgts, x = input_x))
        wgts = backwards(w = input_wgts,
                         x = input_x,
                         correct_answer = correct_answer,
                         approx_answer = approx_answer,
                         alpha = alpha)
        df = df.append({'iteration': i,
                        'input_wgts': input_wgts,
                        'approx_answer': approx_answer,
                        'correct_answer': correct_answer,
                        'error': correct_answer - approx_answer,
                        'new_wgts': wgts},
                       ignore_index=True)
    return(df)


def test(wgts, test_data_set):

    df = pd.DataFrame(columns=["iteration", "dataset", "approx_answer", "correct_answer",
↪   "error"])

    for i in range(0, len(test_data_set)):

        x = test_data_set[i][0]
        correct_answer = test_data_set[i][1]

        approx_answer = step(skalarprodukt(w = wgts,
                                           x = x))

        df = df.append({'iteration': i,
                        "dataset": x,
                        'approx_answer': approx_answer,
                        'correct_answer': correct_answer,
                        'error': correct_answer - approx_answer},
```

```
                        ignore_index=True)
    return(df)
```

Inputs

```
alpha = 1
iterations = 30
biasVal = 1
training_data_set = [(np.array([1,0,0]), 0),
                     (np.array([1,0,1]), 1),
                     (np.array([1,1,0]), 1),
                     (np.array([1,1,1]), 1)]

wgts = np.zeros(len(training_data_set[0][0]))
wgts[0] = -biasVal
```

Aufruf

```
result_fit = fit(alpha = alpha, iterations = iterations, training_data_set =
↪  training_data_set, wgts = wgts)
```

```
py$result_fit
```

```
##    iteration input_wgts approx_answer correct_answer error new_wgts
## 1          1   -1, 0, 0             0              0     0 -1, 0, 0
## 2          2   -1, 0, 0             0              0     0 -1, 0, 0
## 3          3   -1, 0, 0             0              1     1  0, 1, 1
## 4          4    0, 1, 1             1              0    -1 -1, 1, 1
## 5          5   -1, 1, 1             1              1     0 -1, 1, 1
## 6          6   -1, 1, 1             0              0     0 -1, 1, 1
## 7          7   -1, 1, 1             0              0     0 -1, 1, 1
## 8          8   -1, 1, 1             0              0     0 -1, 1, 1
## 9          9   -1, 1, 1             0              0     0 -1, 1, 1
## 10        10   -1, 1, 1             0              0     0 -1, 1, 1
## 11        11   -1, 1, 1             1              1     0 -1, 1, 1
## 12        12   -1, 1, 1             1              1     0 -1, 1, 1
## 13        13   -1, 1, 1             1              1     0 -1, 1, 1
## 14        14   -1, 1, 1             0              0     0 -1, 1, 1
## 15        15   -1, 1, 1             1              1     0 -1, 1, 1
## 16        16   -1, 1, 1             1              1     0 -1, 1, 1
## 17        17   -1, 1, 1             0              0     0 -1, 1, 1
## 18        18   -1, 1, 1             1              1     0 -1, 1, 1
## 19        19   -1, 1, 1             0              0     0 -1, 1, 1
## 20        20   -1, 1, 1             1              1     0 -1, 1, 1
## 21        21   -1, 1, 1             1              1     0 -1, 1, 1
## 22        22   -1, 1, 1             1              1     0 -1, 1, 1
```

```
## 23        23   -1, 1, 1            1              1     0 -1, 1, 1
## 24        24   -1, 1, 1            1              1     0 -1, 1, 1
## 25        25   -1, 1, 1            1              1     0 -1, 1, 1
## 26        26   -1, 1, 1            1              1     0 -1, 1, 1
## 27        27   -1, 1, 1            0              0     0 -1, 1, 1
## 28        28   -1, 1, 1            0              0     0 -1, 1, 1
## 29        29   -1, 1, 1            0              0     0 -1, 1, 1
## 30        30   -1, 1, 1            0              0     0 -1, 1, 1
```