blatt07_nitschke_grisard

December 6, 2018

0.1 Aufgabe 19

siehe PDF

0.2 Aufgabe 20

a) Was beschreibt die Loss-Funktion?

b) Wie kann die Lossfunktion minimiert werden?

c) Welche Funktion haben die Aktivierungsfunktionen bzw. welches Problem wird durch diese gelöst? Nennen Sie drei gängige Aktivierungsfunktionen.

d) Was ist ein Neuron?

e) Nennen Sie drei Anwendungsbeispiele für Neuronale Netze und beschreiben Sie kurz warum sie für diese Beispiele besonders geeignet sind.

Literaturtipp: http://www.informatik.uni-ulm.de/ni/Lehre/SS04/ProsemSC/ausarbeitungen/Ruland.pdf

a) Die Lossfunktion beschreibt den Informationsverlust, der bei einem Schnitt hervorgerufen wird. Da die Lossfunktion unter anderem die Diskrepanz zwischen vorhergesagten und beobachteten Daten beschreibt, soll diese minimiert werden.

b) Allgemein durch analytisches (iteratives) Vorgehen. Zum Beispiel durch die lineare Klassifikation --> bestimme das W, welches die Kostenfunktion minimiert.

c) Die Aktivierungsfunktion ist wichtig für die Anregung eines weiteren Neurons. Die Aktivierungsfunktion, der Output eines Neurons sowie der verwendete Schwellenwert bestimmen, ob die Information an ein gewisses Neuron des nächsten Layers weitergeleitet wird. Dabei muss die Aktivierungsfunktion auch immer zum gewünschten Output des Neurons passen. Mithilfe der Aktivierungsfunktion kann der Raum verzerrt werden, sodass im verzerrten Raum gerade Schnitte gesetzt werden. Im unverzerrten Raum können dann zum Beispiel Werte getrennt werden, die gemäSS einer Spirale angeordnet sind.

- ReLu
- Leaky ReLu
- tanh

d) Ein Neuron besteht grundlegend aus 3 Teilen. Den ersten Teil bildet die sogenannte Propagierungsfunktion, welche mithilfe der Gewichte die Netzeingabe bestimmt. Mit der oben beschriebenen Aktivierungsfunktion wird dann der interne Zustand des Neurons festgelegt. Mithilfe der Ausgabefunktion wird letzendlich der Output eines Neurons bestimmt.

e)

- Bildverarbeitung: Mithilfe neuronaler Netze können die enormen Datenmengen durch Reduktion der Dimensionalität schneller und leichter verarbeitet werden.
- Klassifizierung: Mithilfe der neuronalen Netze kann die Trennung und Klassifizierung von Datenpunkten effizient durchgeführt. Ein mögliches Anwendungsgebiet wäre zum Beispiel die Teichenphysik, da aus eine riesigen Menge an Events nur bestimmte Events gefiltert werden sollen. Für diese Filterung kann dann zum Beispiel ein neuronales Netz trainiert werden.
- ProzeSSoptimierung: Neuronale Netze können zum Beispiel in der Wirtschaft genutzt werden, um bestimmte ProzeSSparameter zu optimieren um so die Effizienz zu steigern

0.3 Aufgabe 21

a) Dimensionen der GröSSen gekennzeichnet durch dim(...) = (m, n) - m Zeilen, n Spalten - dim(x_i) = (M, 1) - dim(C) = (1, 1) - dim(W) = (K, M) - dim(b) = (K, 1)

d) Implementierung der linearen Klassifikation:

```
In [1]: import numpy as np
        import pandas as pd
        from pandas import DataFrame
        import matplotlib.pyplot as plt
        np.random.seed(42)

In [2]: #load data
        P_0 = pd.read_hdf('populationen.hdf5', key = 'P_0')
        P_1 = pd.read_hdf('populationen.hdf5', key = 'P_1')

In [3]: #add labels
        P_0['label'] = np.zeros_like(P_0.x)
        P_1['label'] = np.ones_like(P_1.x)

In [4]: #combine data sets
        P = pd.concat([P_0, P_1], ignore_index=True)

In [5]: #linear classification model
        def f(x, W, b):
            return np.matmul(W, x.T) + b

In [6]: #update matrix and vector b, learning rate h = 0.5
        def update(W, grad_W, b, grad_b, h = 0.5):
            W = W - h * grad_W
            b = b - h * grad_b
            return W, b
```

Die Funktion 'grad_f' berechnet den Gradienten $\nabla_f C$, der für die Berechnung der Gradienten $\nabla_W C$ und $\nabla_b C$ benötigt wird.

```
In [7]: def grad_f(x, f, labels):
            labels = labels.values
            m = np.shape(x)[0] # number of examples
            x = np.matrix(x.values)

            #all different classes (here only 0 and 1)
            classes = np.unique(labels)

            #generate a matrix representing the real class distribution
            truth = np.zeros_like(x.T)
            mask = []
            for i in range(len(classes)):
                mask.append(labels == classes[i])
            truth[~np.array(mask)] = 1

            #calculate the gradient
            grad_f = 1 / m * (np.exp(f) / np.exp(f).sum(axis = 0) - truth)

            return grad_f
```

Funktionen zur Berechnung der Gradienten $\nabla_W C$ und $\nabla_b C$:

```
In [8]: def grad_W(x, f, labels):
            grad_F = grad_f(x, f, labels)
            grad_W = np.matmul(grad_F, x)
            return grad_W
```

```
In [9]: def grad_b(x, f, labels):
            grad_F = grad_f(x, f, labels)
            grad_b = grad_F.sum(axis = 1)
            return grad_b
```

Verwende nun die Funktionen um für 100 Epochen zu trainieren:

```
In [10]: #choose initial W and b randomly
         W = np.matrix(np.random.rand(2, 2))
         b = np.matrix(np.random.rand(2, 1))
         x = P.drop(columns = 'label')

         for i in range(100):
             f_init = f(x, W, b)
             W, b = update(W = W, grad_W = grad_W(x, f_init, P.label),
                           b = b, grad_b = grad_b(x, f_init, P.label))
```

e) Die Geradengleichung (hier nur Spezialfall für 2 Klassen):

$$y(x) = \frac{1}{W_{12} - W_{22}} \{(W_{21} - W_{11})x + b_2 - b_1\}$$

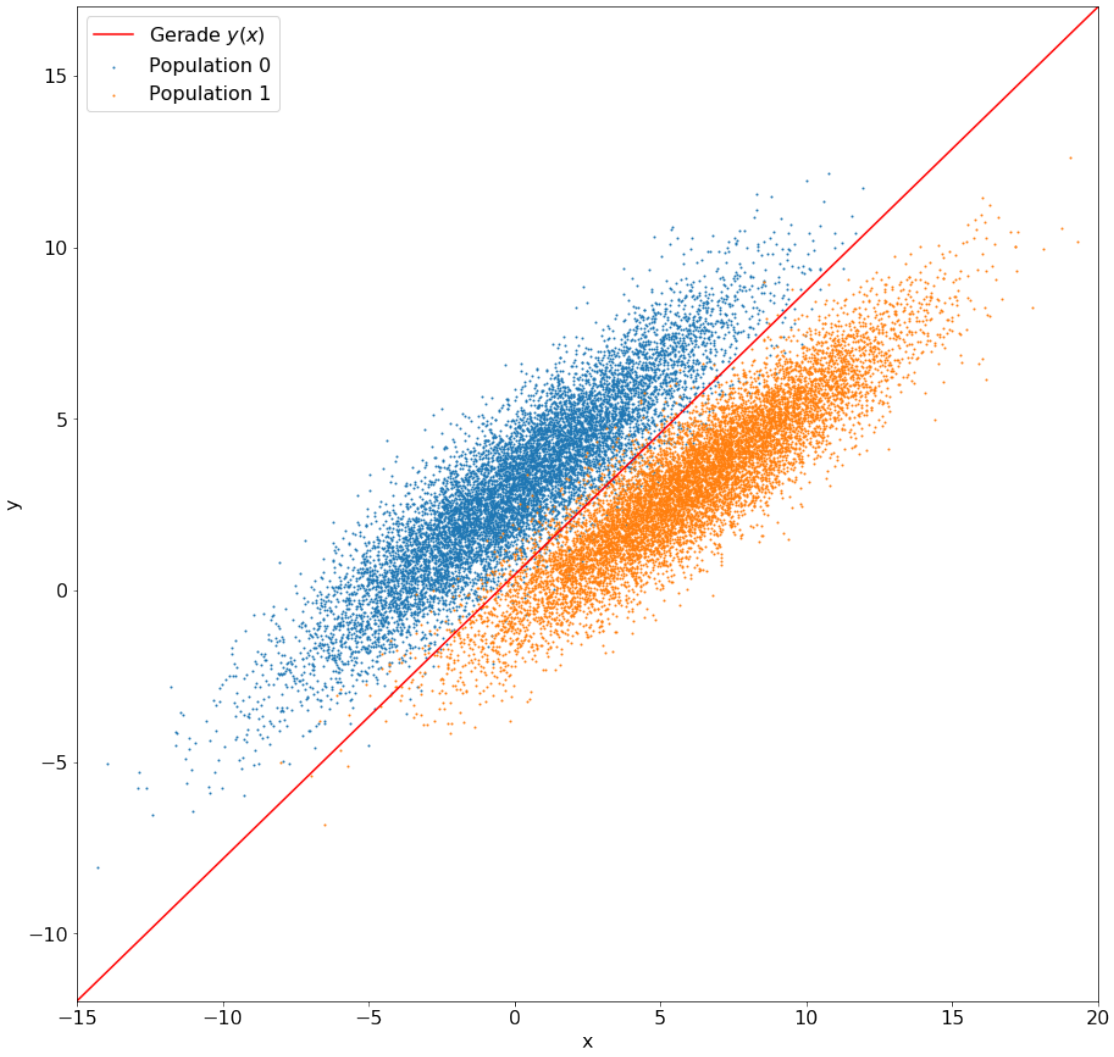
Ergibt sich aus der Bedingung $f_1 = f_2$, da entlang der Gerade der Score für beide Klassen gleich ist. Die Gerade trennt die beiden Populationen. Grafische Darstellung des Ergebnisses:

```
In [11]: def lin(x, W, b):
             return 1 / (W[0, 1] - W[1, 1]) * ((W[1, 0] - W[0, 0])*x + b[1] - b[0] )

In [12]: plt.figure(figsize=(15, 15))
         plt.rcParams.update({'font.size': 16})

         plt.scatter(P_0.x, P_0.y, s = 1, label = 'Population 0')
         plt.scatter(P_1.x, P_1.y, s = 1, label = 'Population 1')
         xplot = np.linspace(-15, 20, 100)
         plt.xlim(xplot[0], xplot[-1])
         plt.ylim(lin(xplot[0], W, b), lin(xplot[-1], W, b))
         plt.plot(xplot, lin(xplot, W, b).T, color = 'r', label = 'Gerade $y(x)$')
         plt.legend()
         plt.xlabel('x', )
         plt.ylabel('y')


         plt.show()
```


Gerade $y(x)$
Population 0
Population 1
15
10
5
0
−5
−10
y
−15
−10
−5
0
5
10
15
20
x

In [ ]: