aufgabe3

December 6, 2018

1 Aufgabe 21 (Dritte Aufgabe auf dem Blatt) - Lineare Klassifikation mit Softmax

1.1 Teilaufgabe a)

1.2 Teilaufgabe b)

1.3 Teilaufgabe c)

1.4 Teilaufgabe d)

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [26]: class net:
             def __init__(self):
                 np.random.seed(0) # Kann man verändern um etwas Abwechslung reinzukriegen
                 self.W = np.random.randn(2,2) # Zufällige Gewichte im Intervall [0,1]
                 self.b = np.zeros((2,1)) # Ist heir fast egal wie man es initialisiert

             def getPredictions(self, X):
                 scores = np.dot(self.W, X) + self.b
                 exp_scores = np.exp(scores)
                 probs = exp_scores / np.sum(exp_scores, axis=0, keepdims=True)
                 # Das ist quasi softmax. In extra Methode ging es irgendwie nicht...?
                 return np.argmax(probs,axis=0) # Das sind dann die predicted Labels

             def train(self, X, labels, stepsize, epochs):
                 m = X.shape[1] # So viele Punkte stecken wir rein
                 for i in range(epochs):
                     scores = np.dot(self.W, X) + self.b # Wir haben schlielich lineare Klassi
                     exp_scores = np.exp(scores)
                     probs = exp_scores / np.sum(exp_scores, axis=0, keepdims=True) # Softmax
                     logprobs = -np.log(probs[labels.astype('int'),range(m)])
                     loss = np.sum(logprobs)/m
                     if i%10==0: print(loss) # ausgeben als kontrolle
                     # Gradienten berechnen
```

```
                dscores = probs
                #for column in range(0,m):
                #    if labels[column] == 0:
                #        dscores[0,column] -= 1
                #    else:
                #        dscores[1,column] -= 1
                # Das ist Multiindexing und shorthand für das obere auskommentiere. Das i
                dscores[labels.astype('int'),range(m)] -= 1
                dscores /= m # Durchschnitt bilden
                dW = np.dot(dscores, X.T) # Der Gradient hat halt diese Form
                db = np.sum(dscores, axis=1, keepdims=True) # Bei dem bias nicht mal x ne
                # Parameter ein mal updaten
                self.W += -stepsize * dW
                self.b += -stepsize * db
```

1.5 Teilaufgabe e)

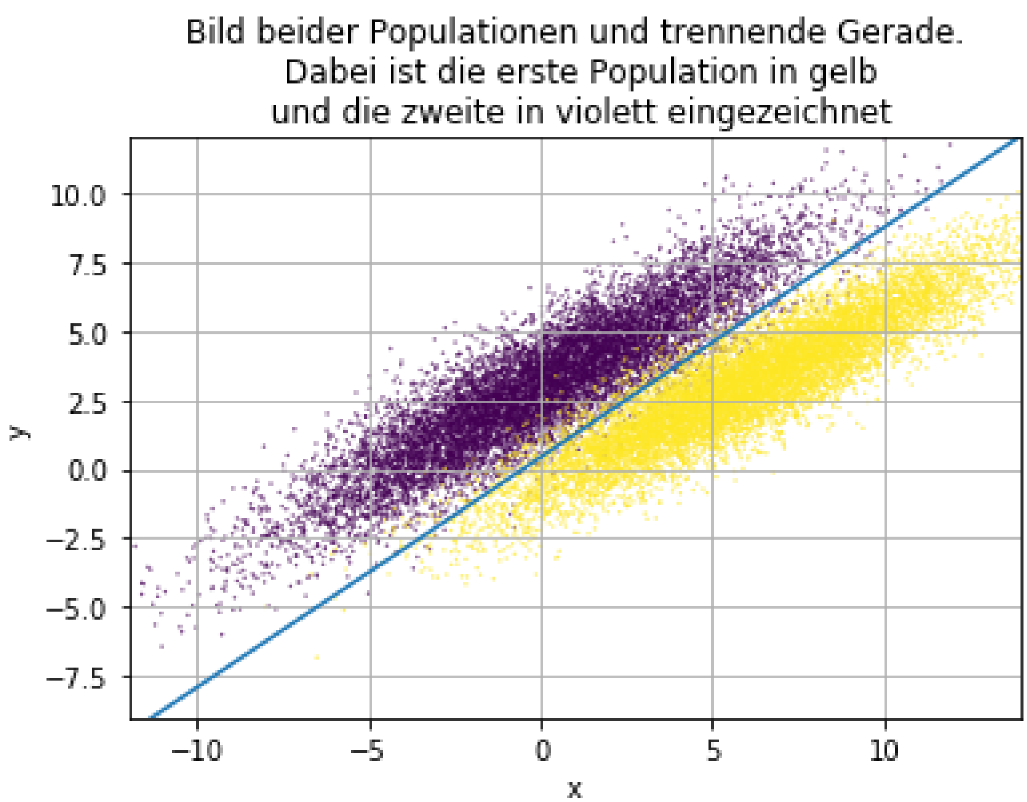
```
In [27]: df_P_0 = pd.read_hdf('populationen.hdf5', key = 'P_0')
         P_0 = df_P_0.values
         df_P_1 = pd.read_hdf('populationen.hdf5', key = 'P_1')
         P_1 = df_P_1.values
         P = np.concatenate((P_0,P_1))
         P = np.transpose(P)
         labels = np.concatenate((np.zeros(P_0.shape[0]),np.ones(P_1.shape[0])))

         testnet = net()
         testnet.train(P, labels, stepsize = 0.5, epochs = 100)
```

```
3.13839773788
0.114890691421
0.103156206518
0.0959353474805
0.0909401878947
0.0873077090349
0.0845770869684
0.0824739747747
0.0808229532658
0.0795062176105
```

```
In [28]: W = testnet.W
         b = testnet.b
         m = (W[1,0]-W[0,0])/(W[0,1]-W[1,1])
         n = (b[1]-b[0])/(W[0,1]-W[1,1])
         print('f(x)=',m,'x+',n)
         print('W = ', testnet.W)
         print('b = ', testnet.b)
```

```
x, y = zip(*np.transpose(P)) # Nicht nachfragen wie das funktioniert, bitte...
xlin = np.linspace(-20,25,1000)
plt.scatter(x, y, s = 0.1, c = labels)
plt.plot(xlin, m*xlin+n)
plt.grid()
plt.title('Bild beider Populationen und trennende Gerade.\n Dabei ist die erste Popula
plt.xlabel('x')
plt.ylabel('y')
plt.axis([-12, 14, -9, 12])
plt.show()
plt.clf()
```

```
f(x)= 0.833059889648 x+ [ 0.42652373]
W =  [[ 0.39213966  2.49601753]
 [ 2.35065067  0.14503288]]
b =  [[-0.50137538]
 [ 0.50137538]]
```



Der Ansatz $f_1 = f_2$ bedeutet, dass die Konfidenz für beide Klassen bei der Zuordnung des Punktes (x,y) gleich ist. Es liegt nahe, dass dies dann die trennende Gerade ist, weil sie Bereiche der beiden Zuordnungen trennt.