Übung 4 - Neuronale Netze

Tobias Hahn - 3073375

1 Perzeptionsalgorithmus

1.1 Implementierung

Ich habe den Algorithmus in zwei Klassen implementiert, perception.py und training.py. Die erste Klasse liefert dabei alle notwendigen Methoden, um aus einem Datenset mit mitgelieferter Klassifizierung einen Trennungsvektor zu ermitteln. Dabei wurde der verbesserte Additions- bzw. Subtraktionsalgorithmus verwendert, um eine verschnellerte Konvergenz zu erreichen.

Die zweite Klasse implementiert die Erstellung des zufälligen Datensets plus Ergebnissen, gibt einige Kenndaten über die Lösung aus und schlussendlich auch noch den Graphen.

Der Graph ist ein sogenannter Parallel Coordinates Graph, dabei wird für jede Dimension eine y-Achse erstellt, die einzelnen Datenpunkte sind Linien die durch die Werte auf den jeweiligen y-Achsen (Dimensionen) definiert werden. Jede Klasse ist hierbei durch die Farbe der Linien definiert. Um die Übereinstimmung des gelernten Vektors mit dem ursprünglichen zu zeigen, wurde außerdem noch eine Dimension eingeführt, die die ursprüngliche Klassifizierung anzeigt. Hier müsste ien guter Algorithmus dafür sorgen, dass alle Linien einer Farbe sich auf einen Punkt vereinen - der in der Vorlesung vorgestellte Algorithmus leistet dies auch, wie man an den Ergebnissen sieht.

Im folgenden zuerst der Quellcode für die beiden Klassen, und danach das Bild mit dem Graphen welcher die Ergebnisse dokumentiert.

../perceptron.py

```
1  import numpy as n
2
3  def add(w, x, epsilon):
4      return w + ((n.dot(-1*w, x)+epsilon)/n.power((n.linalg.norm(x)), 2))*x
5
6  def sub(w, x, epsilon):
7      return w - ((n.dot(1*w, x)+epsilon)/n.power((n.linalg.norm(x)), 2))*x
8
9  def test(w, x, epsilon):
10     if (x['res'] > 0):
11         if (n.dot(w,x['data']) > 0):
12             return (w,False)
13         else:
14             return (add(w, x['data'], epsilon),True)
15     else:
16         if (n.dot(w,x['data']) < 0):
17             return (w,False)
18         else:
19             return (sub(w, x['data'], epsilon),True)
20
21 def learn(data):
22     w = n.zeros(data['data'].shape[1])
23     num = data.shape[0]
24     wrong_count = 0
25     epsilon = 0.001
26     convergence = 0
27     total_count = 0
28
29     while (convergence < 1000):
30         random_int = n.random.randint(0, num)
31         res = test(w, data[random_int], epsilon)
32         w = res[0]
33         total_count += 1
34         convergence += 1
```

```
35          if (res[1]):
36              wrong_count += 1
37              convergence = 0
38
39      return (w,wrong_count,total_count)
```

../training.py

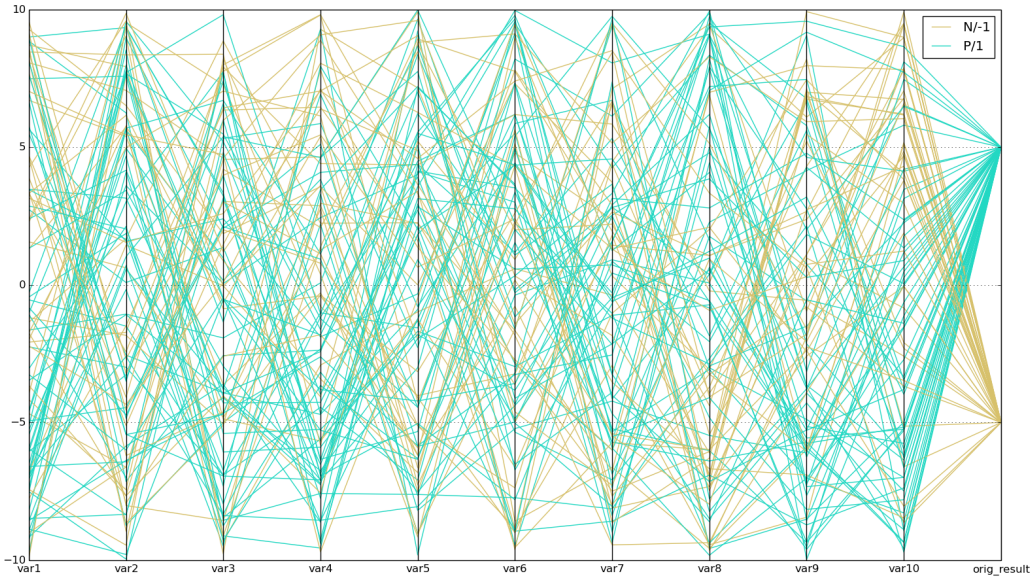
```
1  import numpy as n
2  import perceptron as p
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  from pandas.tools.plotting import parallel_coordinates
6
7  if __name__ == "__main__":
8      samples = 100
9      dimensions = 10
10     w = n.random.rand(dimensions) * 20 - 10
11     data = n.zeros(samples, dtype=[('data', 'f8', (dimensions)), ('res', 'f8'), ('names', '
           a4')])
12     data['data'] = n.random.rand(samples,dimensions) * 20 - 10
13     print("The original weight vector: {0}".format(w))
14
15     for i in range(0,samples):
16         if (n.dot(w, data['data'][i]) > 0):
17             data['res'][i] = 5
18         elif (n.dot(w, data['data'][i]) < 0):
19             data['res'][i] = -5
20         else:
21             del(data[i])
22
23     res = p.learn(data)
24     w = res[0]
25     error = 0
26
27     print("The learned vector: {0}".format(w))
28
29     for i in range(0,samples):
30         if (data['res'][i] > 0 and n.dot(data['data'][i], w) < 0 or data['res'][i] < 0 and n
               .dot(data['data'][i], w) > 0):
31             error += 1
32     print("The number of wrong classifications: {0}\nThe number of adjustments made to the
           vector: {1}\nThe total number of test iterations: {2}".format(error, res[1], res[2])
           )
33
34     for i in range(0,samples):
35         if (n.dot(data['data'][i], w) < 0):
36             data['names'][i] = "N/-1"
37         else:
38             data['names'][i] = "P/1"
39
40     data_columns = ['var1', 'var2', 'var3', 'var4', 'var5', 'var6', 'var7', 'var8', 'var9',
           'var10']
41     df = pd.concat([
42         pd.DataFrame(data['data'], columns=data_columns),
43         pd.DataFrame(data['res'], columns=['orig_result']),
44         pd.DataFrame(data['names'], columns=['names'])], axis=1)
45
46     plt.figure()
47
48     parallel_coordinates(df, "names")
49
50     plt.show()
```

Beispielausgabe

```
1  The original weight vector: [ 9.84658028  0.96802249 -8.5146963   1.333602   -3.55325839
       -4.77932809 -0.08857396 -0.15751875 -6.47169592  7.71299722]
2  The learned vector: [  4.22755294e-04   1.77767972e-05  -3.40020042e-04   6.74635191e-05
       -1.62867848e-04  -1.92920237e-04  -2.40298543e-05  -1.96774406e-05  -2.33433280e-04
       3.34287987e-04]
```

```
3  The number of wrong classifications: 0
4  The number of adjustments made to the vector: 212
5  The total number of test iterations: 5446
```

Figure 1: Graph des Datensets



1.2 Verständnisfrage

Der Perzeptionsalgorithmus braucht für Konvergenz recht lange, wenn zwei Datenpunkte antiparallel sind. Zwei Vektoren sind dann antiparallel, wenn einer der beiden Vektoren das Produkt des anderen mit einer negativen Zahl ist.

Um dies zu begründen stellen wir die beiden Vektoren zuerst als Summe dreier anderer Vektoren dar:

$$x = g + x_{diff}$$
$$y = -g + y_{diff}$$

Wobei wir festhalten können dass die beiden Anteile die hinter g stehen jeweils ziemlich klein sind, da die Vektoren ja fast antiparallell sind. Nun würde der Algorithmus bei jeder Iteration abwechselnd g zum Gewichtsvektor hinzufügen und wieder abziehen, was sich im Mittel ausgleicht. Lediglich der kleine Differenzvektor würde den Gewichtsvektor in die "richtige" Richtung bringen, da nur dieser den Unterschied zwischen x und y ausmacht. Je kleiner dieser Differenzvektor, je "antiparalleler" die Vektoren also sind, desto weniger bewegt sich der Gewichtsvektor also in die richtige Richtung mit jeder Iteration.

1.2.1 Beispieldaten

Dimensionen	Vektor x	Vektor y
x	1.01	-1.02
y	1.01	-1.02