Lernverfahren autonomer Roboter - Übung 4

Tobias Hahn - 3073375
André Osse - 3066368
Waldemar Stockmann - 3066
Markus Strehling - 3066373

December 4, 2016

Übung 4

1 K-Means

1.1 Implementierung von K-Means

../code/KMeans.py

```
1  import numpy as np
2
3  class KMeans:
4      codeBook=None
5
6      def iterate(self, k, sample, random=False):
7          self.codeBook = []
8
9          sample = self.createStartReproductionVecs(k, sample, random)
10
11         for i in range(len(sample)):
12             x = sample[i]
13
14             rIndex = self.getReproductionMinDistIndex(x)
15             r = self.codeBook[rIndex][1]
16
17             r.append(x)
18
19             cen = self.calcNewCentrum(rIndex)
20             self.codeBook[rIndex][0] = cen
21
22         return self.codeBook
23
24     def createStartReproductionVecs(self, k, sample, random):
25         if random:
26             for i in range(k):
27                 index = np.random.randint(0, high=len(sample))
28                 self.codeBook.append([sample[index], [sample[index]]])
29                 sample1 = sample[0:index-1]
30                 sample2 = sample[index+1:len(sample)]
31                 np.concatenate((sample1, sample2), axis=0)
32         else:
33             for i in range(k):
34                 self.codeBook.append([sample[0], [sample[0]]])
35                 sample = sample[1:len(sample)]
36
37         return sample
38
39     def calcNewCentrum(self, index):
40         means = self.calcMeanVector(self.codeBook[index][1])
41         cent = self.getVectorMinDistInCell(means, self.codeBook[index][1])
42         return cent
43
44     def getReproductionMinDistIndex(self, vec):
45         index = 0
46         distMin = self.calcEuclideanDistance(vec, self.codeBook[0][0])
47         for i in range(1, len(self.codeBook)):
48             distComp = self.calcEuclideanDistance(vec, self.codeBook[i][0])
49             if distMin > distComp:
50                 index = i
51                 distMin = distComp
52
53         return index
54
55     def getVectorMinDistInCell(self, vec, cell):
56         vecRet = cell[0]
57         distMin = self.calcEuclideanDistance(vec, cell[0])
58         for i in range(1, len(cell)):
59             distComp = self.calcEuclideanDistance(vec, cell[i])
60             if distMin > distComp:
61                 vecRet = cell[i]
```

```
62                distMin = distComp
63
64        return vecRet
65
66    def calcMeanVector(self, vectors):
67        vecLength = len(vectors[0])
68        means = [0 for x in range(vecLength)]
69        for i in range(len(vectors)):
70            for j in range(vecLength):
71                means[j] += vectors[i][j]
72
73        for i in range(vecLength):
74            means[i] = means[i] / len(vectors)
75
76        return means
77
78    def calcEuclideanDistance(self, vec1, vec2):
79        return np.linalg.norm(vec1 - vec2)
```

1.2 Tests von der K-Means-Implementierung

../code/Test.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  from KMeans import KMeans
5
6  class Tests:
7
8      def test(self, k, data, random=False):
9          kmeans = KMeans()
10         book = kmeans.iterate(k, data, random)
11         print('---Start---')
12         for i in range(len(book)):
13             b = book[i]
14             print(b[0])
15             listX = []
16             listY = []
17             repVecX = [b[0][0]]
18             repVecY = [b[0][1]]
19             for vec in b[1]:
20                 listX.append(vec[0])
21                 listY.append(vec[1])
22             plt.plot(listDX, listDY, 'ro', listX, listY, 'g^', repVecX, repVecY, 'bs')
23             plt.axis([-5, 30, -5, 30])
24             plt.show()
25             plt.clf()
26         print('---End---')
27
28 if __name__ == '__main__':
29     t = Tests()
30
31     data = np.loadtxt("./cluster_dataset2d.txt")
32
33     listDX = []
34     listDY = []
35
36     for d in data:
37         listDX.append(d[0])
38         listDY.append(d[1])
39
40     t.test(3, data)
41     t.test(6, data)
42     t.test(12, data)
43     t.test(3, data, True)
44     t.test(6, data, True)
45     t.test(12, data, True)
```

1.3 Clustering: Dataset2D

Wie in der Aufgabe vorgegeben wurde k=3 gewählt. Ebenfalls wurden die initialen Reproduktions-Vektoren der Cluster zufällig gewählt, da mit den ersten k-gewählten Vektoren als Reproduktions-Vektoren keine optimalen aber reproduzierbare Ergebnisse geliefert wurden.

1.3.1 Erklärung des Plots

Die roten Punkte repräsentieren die 2D-Vektoren aus dem Datensatz. Das blaue Viereck stellt den Reproduktions-Vektor des jeweiligen Clusters da. Die grünen Vierecke sind die genutzten Vektoren um den Reproduktions-Vektor zu bestimmen.
Dabei ist zu beachten, dass diese Vektoren sich ebenfalls in anderen Clustern befinden können, da sie nicht die Reproduktion, sondern die Findung des Reproduktions-Vektors dienen. Desweiteren wenn diese Vektoren zu diesem Zeitpunkt der Iteration weniger optimal liegen, kann es geschehen, dass ein Vektor einem anderen Reproduktions-Vektor zugewiesen wird, als angenommen werden würde. Deshalb kann es geschehen, dass grüne Dreiecke ebenfalls in anderen der 'Haufen' auffindbar wären.
In dem Durchlauf, welcher diese Plots darstellen, ist dies jedoch nicht geschehen.

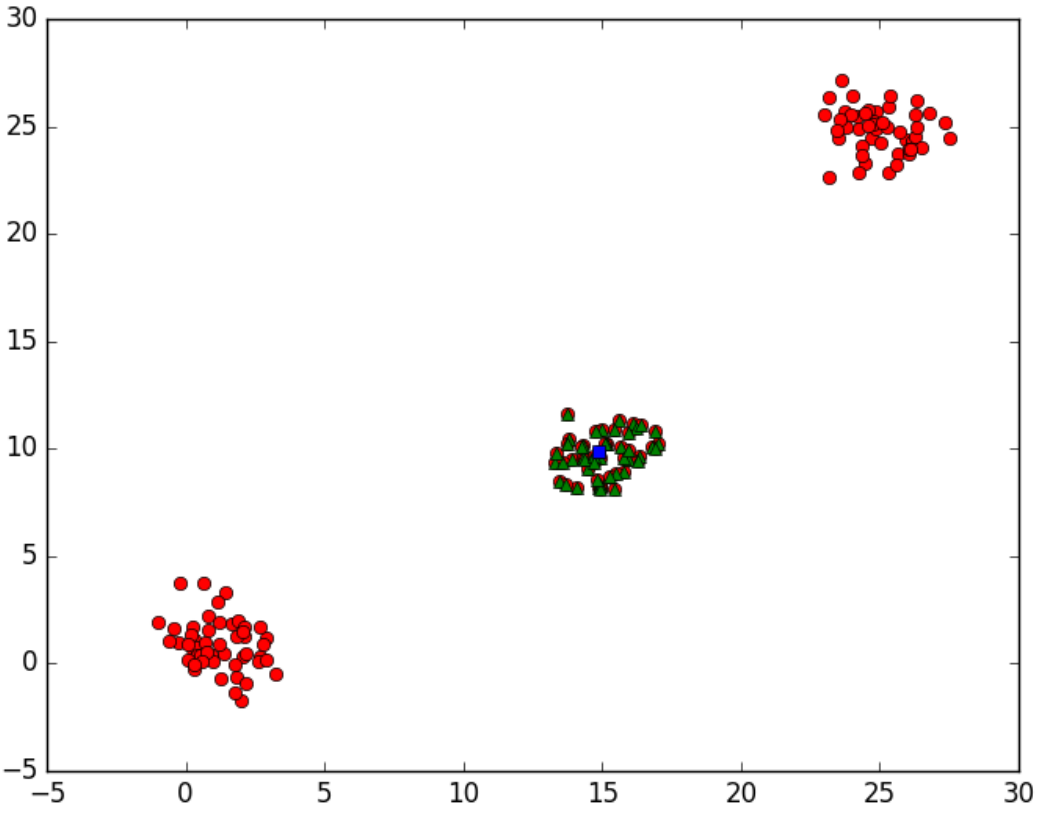

Figure 1: Cluster: 1

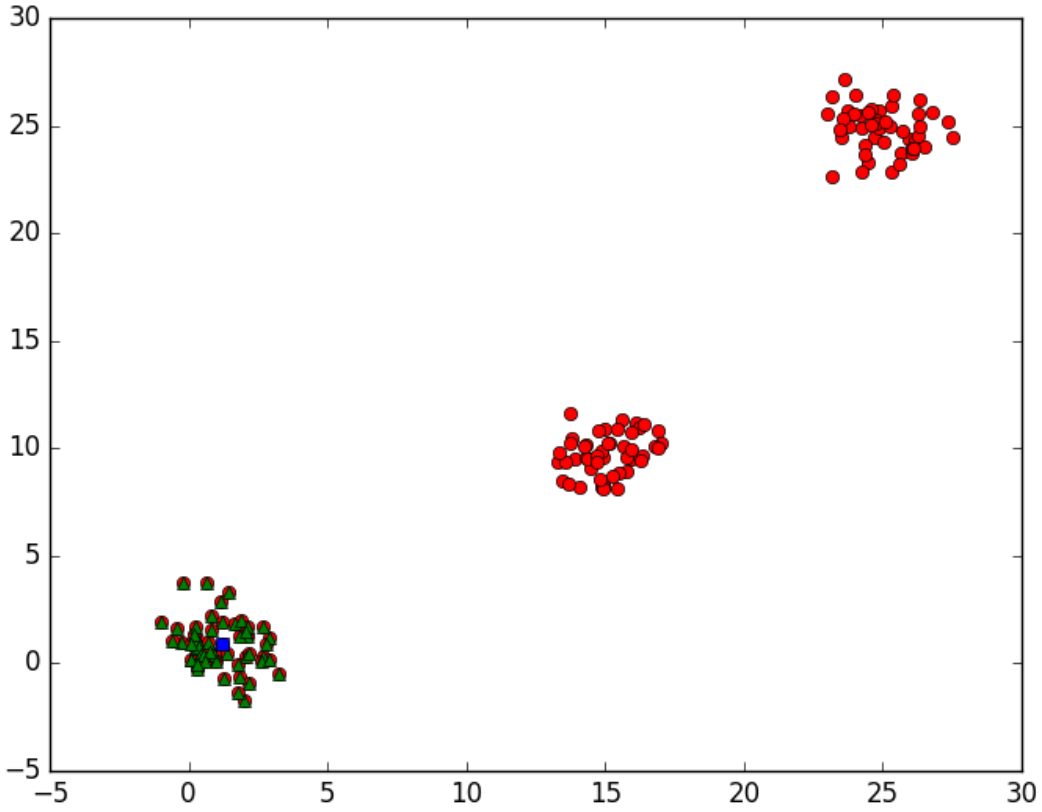

Figure 2: Cluster: 2

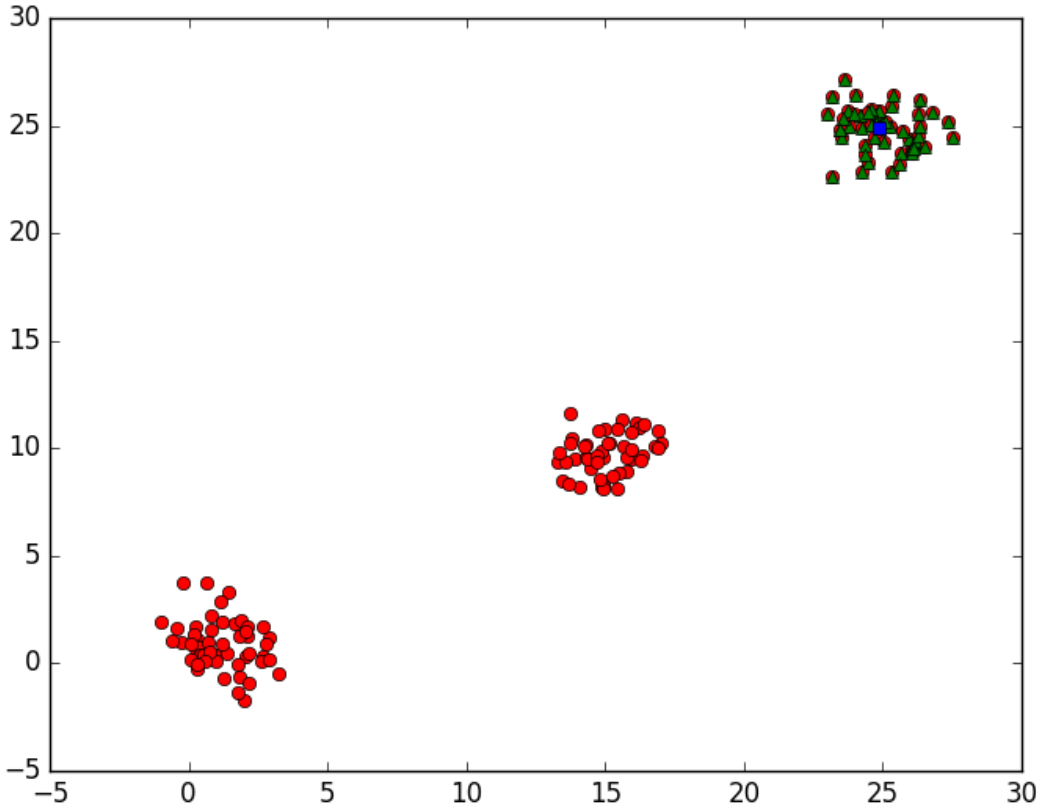

Figure 3: Cluster: 3