

Prof. R. Rojas

Mustererkennung, WS17/18

Übungsblatt 5

Boyan Hristov, Nedeltscho Petrov

22. November 2017

Link zum Git Repository: <https://github.com/BoyanH/FU-MachineLearning-17-18/tree/master/Solutions/Homework5>

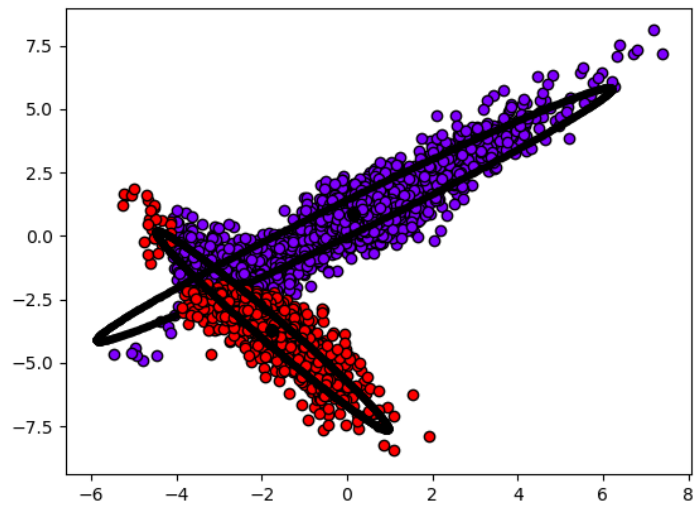
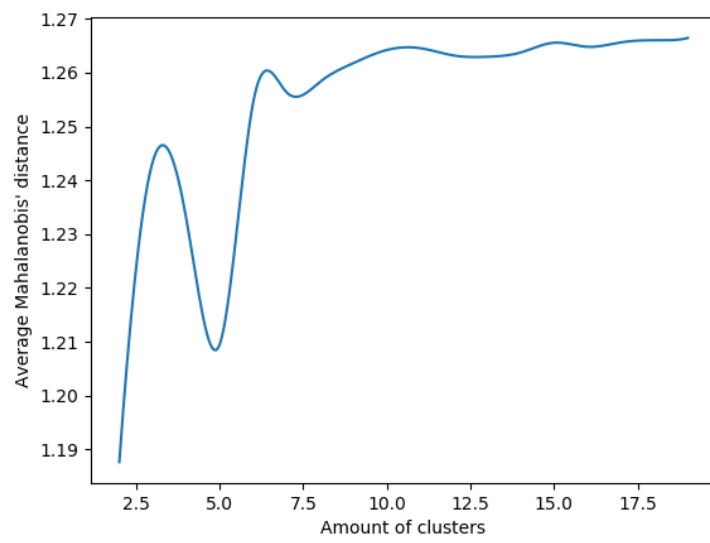
Expectation Maximization

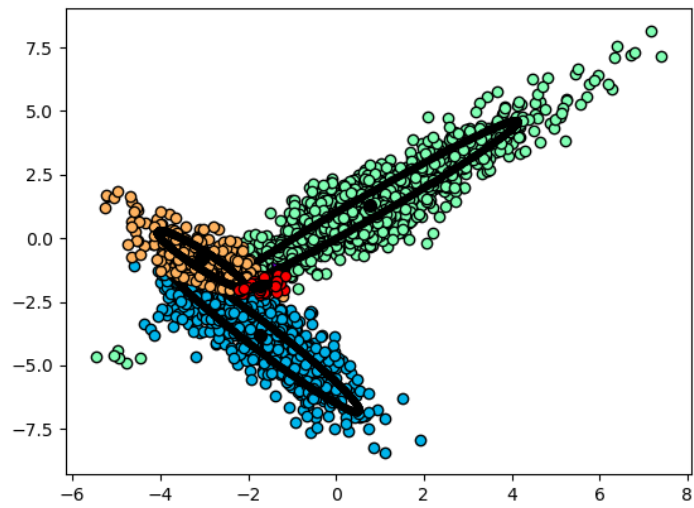
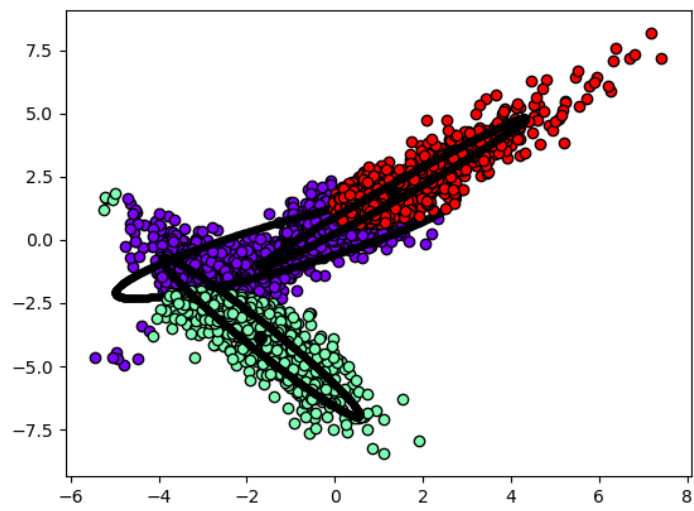
Das Verfahren ist für gestreute Datensätze eine deutlich bessere Clustering Methode als K-Means. Viel, womit wir das vergleichen können, haben wir noch nicht gelernt. Für den gegebenen Datensatz, der zwei sehr stark ausgeprägte Clusters hat, hat es aber super funktioniert. Wir erkennen aber die Clusters nur anhand der Streuung der Daten, d.h. also wir wurden Schwierigkeiten mit K-Means haben, selbe Ergebnisse zu bekommen.

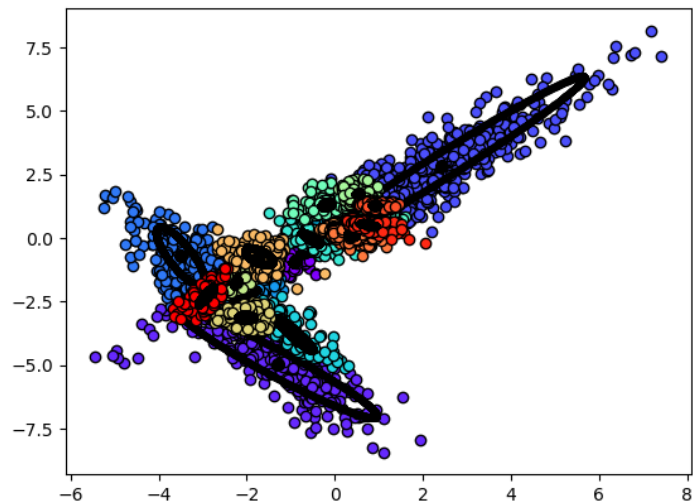
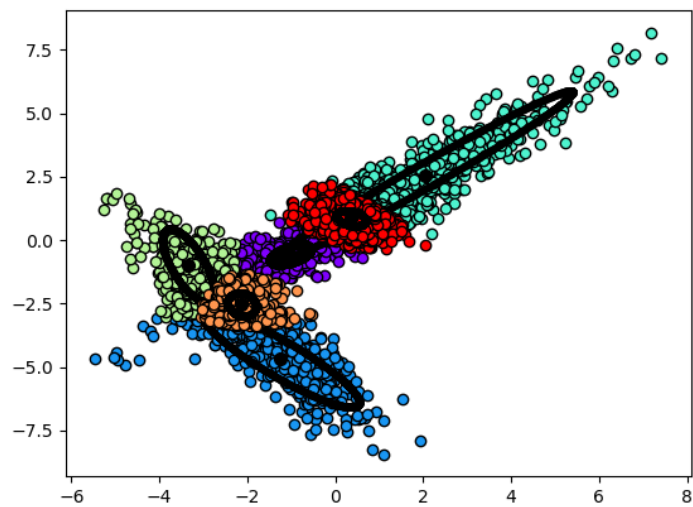
Plots

Wir haben die durchschnittliche Distanz zum Cluster (Mahalanobis) von allen Punkten abhängig von der Anzahl der Cluster. Damit die Graphik besser aussieht, haben wir scipy benutzt, es existiert aber natürlich kein Clustering mit z.B. 2.3 Cluster.

Auf dem Plot kann man die im Tutorium besprochene "Elbow"-Leider nicht sehen. Es war nur in einer früheren Version des Programms zu sehen, die ähnlicher zu K-Means war. Es liegt vermutlich daran, dass der Datensatz eine große und leicht erkennbare Streuung der Daten hat. Nachdem man mehr als 2 Cluster versucht zu finden, werden viele Punkte außerhalb des Streuungsbereichs eines Clusters liegen, aber trotzdem zu dem Cluster gehören. Das ist so, da je mehr man die Daten splittet in mehreren Clusters, desto weniger wichtig wird die Streuung eines Clusters verglichen mit einem anderen. Naja, "long story short" wird denken 2 Clusters sind am besten geeignet für diesen Datensatz.







Details zur Implementierung

Interessant für die Implementierung sind wahrscheinlich die Berechnung von dem Abstand und die Entscheidung, wann der Algorithmus eigentlich fertig ist.

Mahalanobis Abstand - wir könnten uns hier den Wurzel sparen, so haben wir aber deutlich präziseren durchschnittlichen Abstand bekommen, der besser zu plotten war. Da sonst die Großteil der Implementierung vektorisiert ist, hatten wir nicht zu viele Sorgen wegen Performance. Sehr interessant ist es eigentlich nicht, nur die Formel aus der Vorlesung..

```

1  def get_distance_mahalanobis_to_cluster(self, x, i_cluster):
2      # the square root could be removed, but helps for better plotting
3      return math.sqrt((x - self.cluster_centers[i_cluster]).dot(
4          self.inv_covariances_per_cluster[i_cluster]).dot((x - self.cluster_centers[
            i_cluster]).T))

```

Cluster Zentren berechnen, vektorisiert

```
1 def calculate_cluster_centers(self):
2     get_cluster_centers = np.vectorize(lambda x, points_per_center: points_per_center[x
3     ].mean(0),
4     signature='(,), (m)->(n)')
5     self.cluster_centers = get_cluster_centers(self.cluster_indexes, self.
6     points_per_cluster)
```

Berechnung von Kovarianzmatrizen - dafür haben wir Numpy benutzt, interessant es aber der Fall, wenn wir nicht genug Punkte haben. Dann nehmen wir bloß die Identitätsmatrix. ägainßtete im Kommentare, da wir bei der Initialisierung auch die Identitätsmatrix nehmen.

```
1 def calculate_covariances(self):
2     # if we only have the center in our cluster (empty cluster) then just use the
3     identity
4     # matrix as covariance matrix again
5     get_covariances = np.vectorize(lambda x, points_for_cluster: np.cov(
6     points_for_cluster[x],
7     rowvar=False, bias=True) if len(points_for_cluster[
8     x]) > 1 else
9     np.identity(len(points_for_cluster[x][0])),
10    signature='(,), (m)->(n,n)')
11    self.covariances_per_cluster = get_covariances(self.cluster_indexes, self.
12    points_per_cluster)
13    self.inv_covariances_per_cluster = np.vectorize(lambda x: np.linalg.pinv(x),
14    signature='(m,n)->(m,n)')(self.
15    covariances_per_cluster)
```

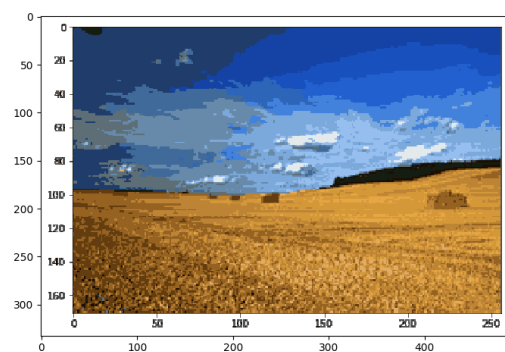
”Main”Methode - wir haben hier anhand von Flags zwie Vorgehensweisen implementiert. Entweder terminiert man, wenn die Clusterzentren sich nicht so viel bewegen, oder falls man 1. schlechtere Ergebnisse bekommt oder 2. ein Wunschergebniss erreicht hat. In beiden Implementierungen gibt es eine maximale Anzahl von Iterationen (30, vermutlich wäre 10 besser, aber... i

```
1 def apply_expectation_maximization(self, k=0):
2     if VERBOSE:
3         print('iteration: {}'.format(k))
4     old_centers = np.copy(self.cluster_centers)
5     old_distance = np.copy(self.mean_distance)
6     self.reset_points_per_cluster()
7     self.assign_points_to_clusters()
8     self.calculate_cluster_centers()
9     self.calculate_covariances()
10    self.update_mean_distance_to_cluster_centers()
11
12    # for some reason old_distance is None or old_distance > ...
13    # was throwing errors cannot compare NoneType with int
14    # therefore the less readable not old_distance :X
15
16    # but basically, if judging on distance for when to stop,
17    # if the average distance gets worse, we reach the max amount of iterations or we
18    reach our desired
19    # threshold, stop
20    if USE_DISTANCE_THRESHOLD:
21        if (self.mean_distance > DISTANCE_THRESHOLD and
22            k < self.max_iterations and (not old_distance or old_distance >
23            self.mean_distance)):
24            self.apply_expectation_maximization(k + 1)
25
26    # other method to determine when to stop is by simply checking if the cluster
27    centers still move enough
28    elif abs((old_centers - self.cluster_centers).sum()) > MOVEMENT_THRESHOLD and k <
29    self.max_iterations:
```

Bildkompression

Nicht sicher was ich sagen soll, außer es ist seeeehr langsam (nicht so viel wie erste Hausaufgabe mit k-NN, aber die war auch nicht vektorisiert). Ich habe nur 30 Farben genommen, es hat aber ziemlich gut geklappt auf dem Bild. Also wenn man nur die Ergebnisse analysiert, ist es gar nicht so schlecht, leider aber zu wenig performant.

Before & After



Erklärung Bildkompression

Ich werde mir hier lange Erklärungen sparen, wir haben EM benutzt, um alle Pixel (damit Farben) in dem Bild zu Clustern. So wird z.B. nur die mittlere und häufigste Grüne genommen von allen Grünen (bei uns nicht genau, bei 30 Farben muss es mindestens 2 Grüne geben (war Beispiel, es gibt in dem Bild gar nicht grün i). Danach haben wir je Pixel mit der am nächsten zu der Farbe liegenden Cluster ausgetauscht. Vollständiges Code am Ende.

Vollständiges Code zu Expectation Maximization

```
1 from Parser import parse_data
2 import numpy as np
3 from Helpers import save_plot, plot_covariance
4 from random import random
5 import matplotlib.pyplot as plt
6 import matplotlib.cm as cm
7 from scipy.interpolate import spline
8 import random
9 import math

11 CHOOSE_INITIAL_CENTERS_RANDOMLY = True
12 USE_DISTANCE_THRESHOLD = True
13 MOVEMENT_THRESHOLD = 0.002
14 DISTANCE_THRESHOLD = 0.01

16 VERBOSE = False
17 PLOT_MEAN_FOR_CLUSTERS_COUNT = True
18 PLOT_CLUSTERING_FOR_SOME_K = True
19 SAVE_PLOTS = True

22 class ExpectationMaximization:
23     @staticmethod
24     def get_initial_centers_from_data_set(data, k):
25         if CHOOSE_INITIAL_CENTERS_RANDOMLY:
26             random.seed(8)
27             return np.array(random.choices(data, k=k), dtype=np.float64)

29         min_point = data.min(0)
30         max_point = data.max(0)
31         centers = []

33         for i in range(k):
34             centers.append(min_point + (max_point - min_point) / k)

36         return centers

38     def __init__(self):
39         self.data = None
40         self.k_clusters = None
41         self.sigma = None
42         self.cluster_centers = None
43         self.points_per_cluster = None
44         self.inv_covariances_per_cluster = []
45         self.covariances_per_cluster = []
46         self.cluster_indexes = None
47         self.last_diff = None
48         self.mean_distance = None
49         self.max_iterations = None

51     def reset_points_per_cluster(self):
52         self.points_per_cluster = [[x] for x in self.cluster_centers]
```

```

54     def cluster(self, data, k_clusters, max_iterations=30):
55         self.data = data
56         self.mean_distance = None
57         self.last_diff = None
58         self.k_clusters = k_clusters
59         self.cluster_indexes = [x for x in range(self.k_clusters)]
60         self.cluster_centers = ExpectationMaximization.get_initial_centers_from_data_set(
61             data, k_clusters)
62         self.covariances_per_cluster = [[np.identity(len(x))] for x in self.cluster_centers
63 ]
64         self.inv_covariances_per_cluster = self.covariances_per_cluster
65         self.reset_points_per_cluster()
66         self.max_iterations = max_iterations
67         self.apply_expectation_maximization()
68
69     def apply_expectation_maximization(self, k=0):
70         if VERBOSE:
71             print('iteration: {}'.format(k))
72         old_centers = np.copy(self.cluster_centers)
73         old_distance = np.copy(self.mean_distance)
74         self.reset_points_per_cluster()
75         self.assign_points_to_clusters()
76         self.calculate_cluster_centers()
77         self.calculate_covariances()
78         self.update_mean_distance_to_cluster_centers()
79
80         # for some reason old_distance is None or old_distance > ...
81         # was throwing errors cannot compare NoneType with int
82         # therefore the less readable not old_distance :X
83
84         # but basically, if judging on distance for when to stop,
85         # if the average distance gets worse, we reach the max amount of iterations or we
86         # reach our desired
87         # threshold, stop
88         if USE_DISTANCE_THRESHOLD:
89             if (self.mean_distance > DISTANCE_THRESHOLD and
90                 k < self.max_iterations and (not old_distance or old_distance >
91 self.mean_distance)):
92                 self.apply_expectation_maximization(k + 1)
93
94         # other method to determine when to stop is by simply checking if the cluster
95         # centers still move enough
96         elif abs((old_centers - self.cluster_centers).sum()) > MOVEMENT_THRESHOLD and k <
97 self.max_iterations:
98             self.apply_expectation_maximization(k + 1)
99
100     def update_mean_distance_to_cluster_centers(self):
101         # 1. get distance for every point in each cluster in a single array
102         # 2. get mean of that
103
104         dis_for_x_in_k = np.vectorize(lambda x, i: self.get_distance_mahalanobis_to_cluster
105 (x, i),
106                                     signature='(m),(n)->()')
107
108         # didn't manage to vectorize this one ;/ somehow, numpy doesn't like jagged arrays
109         distances_for_points = list(map(lambda x: dis_for_x_in_k(self.points_per_cluster[x
110 ], x),
111                                     self.cluster_indexes))
112
113         flattened_distances = []
114         for cluster_distances in distances_for_points:
115             flattened_distances = flattened_distances + list(cluster_distances)
116
117         self.mean_distance = np.array(flattened_distances, dtype=np.float64).mean()
118
119         if VERBOSE:
120             print('Average Mahalanobis\' distance to cluster center: {}'.format(self.
121 mean_distance))

```



```

114     def assign_points_to_clusters(self):
115         assign_points_to_clusters = np.vectorize(lambda x: self.assign_point_to_cluster(x),
116                                                  signature='(m)->()')
117         assign_points_to_clusters(self.data)

119         for idx, list in enumerate(self.points_per_cluster):
120             self.points_per_cluster[idx] = np.array(self.points_per_cluster[idx])

122     def assign_point_to_cluster(self, point):
123         distances_to_clusters = [self.get_distance_mahalanobis_to_cluster(point, i) for i
124 in self.cluster_indexes]
125         closest_cluster_idx = np.argmin(distances_to_clusters)
126         self.points_per_cluster[closest_cluster_idx].append(point)

127     def get_distance_mahalanobis_to_cluster(self, x, i_cluster):
128         # the square root could be removed, but helps for better plotting
129         return math.sqrt((x - self.cluster_centers[i_cluster]).dot(
130             self.inv_covariances_per_cluster[i_cluster]).dot((x - self.cluster_centers[
131 i_cluster])).T))

132     def calculate_cluster_centers(self):
133         get_cluster_centers = np.vectorize(lambda x, points_per_center: points_per_center[x
134 ].mean(0),
135                                           signature='(),(m)->(n)')
136         self.cluster_centers = get_cluster_centers(self.cluster_indexes, self.
137 points_per_cluster)

138     def calculate_covariances(self):
139         # if we only have the center in our cluster (empty cluster) then just use the
140 identity
141         # matrix as covariance matrix again
142         get_covariances = np.vectorize(lambda x, points_for_cluster: np.cov(
143 points_for_cluster[x],
144 rowvar=False, bias=True) if len(points_for_cluster[
145 x]) > 1 else
146 np.identity(len(points_for_cluster[x][0])),
147 signature='(),(m)->(n,n)')
148         self.covariances_per_cluster = get_covariances(self.cluster_indexes, self.
149 points_per_cluster)
150         self.inv_covariances_per_cluster = np.vectorize(lambda x: np.linalg.pinv(x),
151                                                         signature='(m,n)->(m,n)')(self.
152 covariances_per_cluster)

153 data = parse_data()
154 em = ExpectationMaximization()

155 # Depending on the cluster centers that are chosen, results differ
156 # Though mostly between 3 and 4 clusters fit best for the current data set
157 if PLOT_MEAN_FOR_CLUSTERS_COUNT:
158     cluster_count_experiments = [x for x in range(2, 20)]
159     cluster_count_mean_distance_results = []

160     for cluster_count in cluster_count_experiments:
161         em.cluster(data, cluster_count)
162         cluster_count_mean_distance_results.append(np.copy(em.mean_distance))

163         if VERBOSE:
164             for idx, points in enumerate(em.points_per_cluster):
165                 print('Points in cluster #{}: {}'.format(idx, len(points)))

166     x = np.linspace(min(cluster_count_experiments), max(cluster_count_experiments), 300)
167     y = spline(cluster_count_experiments, cluster_count_mean_distance_results, x)

168     figure = plt.figure()
169     plt.plot(x, y)
170     plt.xlabel('Amount of clusters')

```

```

171     plt.ylabel('Average Mahalanobis\' distance')
173
174     if SAVE_PLOTS:
175         save_plot(figure, './plots/avrg_distance_for_k.png')
176     else:
177         plt.show()
178
179 if PLOT_CLUSTERING_FOR_SOME_K:
180     plot_for_k_s = [2,3,5,6, 20]
181     # plot_for_k_s = [2]
182
183     for k in plot_for_k_s:
184         em.cluster(data, k)
185         colors = cm.rainbow(np.linspace(0, 1, k))
186
187         fig = plt.figure()
188         ax1 = fig.add_subplot(111)
189
190         for cl_idx in em.cluster_indexes:
191             X = em.points_per_cluster[cl_idx]
192             x, y = zip(*X)
193             # ax1.figure(figsize=(15, 10))
194             ax1.scatter(x, y, edgecolors="black", c=colors[cl_idx])
195
196             center = em.cluster_centers[cl_idx]
197             covariance = em.covariances_per_cluster[cl_idx]
198             plot_covariance(ax1, center[0], center[1], covariance)
199
200         if SAVE_PLOTS:
201             save_plot(fig, './plots/plot_for_k_{}.png'.format(k))
202         else:
203             plt.show()
204
205 # Parser.py
206
207 import csv
208 import numpy as np
209 import os
210 from sklearn.model_selection import train_test_split
211
212 def parse_data():
213     file_name = os.path.join(os.path.dirname(__file__), './Dataset/2d-em.csv')
214     csv_file = open(file_name, 'rt')
215     reader = csv.reader(csv_file, delimiter=',', quoting=csv.QUOTE_NONE)
216
217     return np.array([row for row in reader], dtype=np.float64)
218
219 # Helpers.py
220
221 import os
222 import pandas as pd
223 from numpy import pi, sin, cos
224 import numpy as np
225 import matplotlib.pyplot as plt
226
227 RGB_BLACK = [0, 0, 0]
228
229 def save_plot(fig, path):
230     fig.savefig(os.path.join(os.path.dirname(__file__), path))

```

```

241 def plot_covariance(ax1, x_initial, y_initial, cov):
242     num_points = 1000
243     radius = 1.5 # adjusted radius, seems more correct this way
244
245     # plot a circle
246     arcs = np.linspace(0, 2 * pi, num_points)
247     x = radius * sin(arcs)
248     y = radius * cos(arcs)
249
250     # stretch it according to the covariance matrix
251     xy = np.array(list(zip(x, y)))
252     x, y = zip(*xy.dot(cov))
253
254     # move it in the space so it's center is above the cluster's center
255     x = x + x_initial
256     y = y + y_initial
257
258     ax1.scatter(x, y, c=RGB_BLACK, s=10) # plot covariance
259     ax1.scatter([x_initial], [y_initial], c=RGB_BLACK, s=50) # plot center
260
261
262 def show_img(img, save=False):
263     figure = plt.figure(figsize=(8, 8))
264     plt.imshow(img)
265
266     if save:
267         save_plot(figure, './plots/compressed.png')
268     else:
269         plt.show()

```

Vollständiges Code zu Bildkompression

```

1 import numpy as np
2 import numpy as np
3 import os
4 import matplotlib.image as mpimg
5 from Helpers import show_img
6 from ExpectationMaximization import ExpectationMaximization
7 import cv2
8
9 class ImageCompression(ExpectationMaximization):
10     def compress(self, img, colours):
11         try:
12             reshaped = img.reshape(img.shape[0] * img.shape[1], 4) # in my case it was 4,
13             rgb + alpha
14         except:
15             reshaped = img.reshape(img.shape[0] * img.shape[1], 3) # could be 3 as well,
16             who cares
17
18             self.cluster(reshaped, colours)
19
20             for r_idx, row in enumerate(img):
21                 for c_idx, pixel in enumerate(row):
22                     img[r_idx][c_idx] = self.get_color(pixel)
23
24             return img
25
26     def get_color(self, pixel):
27         distances_to_clusters = [self.get_distance_mahalanobis_to_cluster(pixel, i) for i
28         in self.cluster_indexes]
29         closest_cluster_idx = np.argmin(distances_to_clusters)

```

```

28         return self.cluster_centers[closest_cluster_idx]

30     # performance is important here, overwrite to remove math.sqrt
31     def get_distance_mahalanobis_to_cluster(self, x, i_cluster):
32         # the square root could be removed, but helps for better plotting
33         return (x - self.cluster_centers[i_cluster]).dot(
34             self.inv_covariances_per_cluster[i_cluster]).dot((x - self.cluster_centers[
35                 i_cluster]).T)

39 ic = ImageCompression()
40 img = mpimg.imread('./Dataset/image.png')
41 show_img(img)

43 compressed = ic.compress(img, 30) # 30 colors
44 show_img(compressed, True) # True for save

```