

```

1  #
2  # svd_faces.py
3  #
4
5  import matplotlib.pyplot as plt
6
7  import numpy as np
8  import numpy.linalg as LA
9  import argparse
10 import scipy.misc as misc
11 import glob, os
12 import cv2
13 from sklearn import decomposition
14 import time
15
16 def reconstr_matrix(U,D,V,k):
17     rec_mat=np.dot(U[:, :k], np.dot(D[:k, :k], V[:k, :]))
18     return rec_mat
19
20 def rgb2gray(rgb):
21
22     if(len(rgb.shape)>2):
23         r, g, b = rgb[:, :, 0], rgb[:, :, 1], rgb[:, :, 2]
24         L = 0.2989 * r + 0.5870 * g + 0.1140 * b
25     else:
26         L = rgb
27     return L
28
29 def ParseArguments():
30     parser = argparse.ArgumentParser(description="Project ")
31     parser.add_argument('--data-dir', default="", required=True, help='data dir (default: %(default)s)')
32     parser.add_argument('--alg', default="pca", required=False, help='pca, nmf (default: %(default)s)')
33     args = parser.parse_args()
34
35     return args.data_dir, args.alg
36
37
38 data_dir, alg = ParseArguments()
39
40 classes=[]
41
42 for file in glob.glob(data_dir+"/*"):
43     tmp=file.split('/')
44     print(tmp)
45     classes.append(tmp[len(tmp)-1])
46
47 print("classes =", classes)
48
49 data_classes=np.zeros(0,dtype=np.int)
50
51 classes_names=classes;
52
53 counter_tmp=0;
54
55 points=np.array([])
56
57 for classs in classes:
58     list1=glob.glob(data_dir+"/"+classs+"/*");
59     for file in list1:
60
61
62         face0 = plt.imread(file)
63         face = rgb2gray(face0)
64
65         h=face.shape[0]
66         w=face.shape[1]
67
68
69         face2 = np.reshape(face, face.shape[0]*face.shape[1]);
70
71
72         print("class = ", classs, "file = ", file, " , shape = ", face.shape, " , shape2 = ", face2.shape)
73
74         #data_classes=np.append(data_classes,int(classs));
75         data_classes=np.append(data_classes,int(counter_tmp));
76
77
78         if(points.size==0):
79             points=face2;
80     else:

```

```

81         points=np.vstack((points,face2))
82
83         counter_tmp=counter_tmp+1
84
85     X=points.transpose()
86
87
88     U, Sigma, VT = np.linalg.svd(X, full_matrices=False)
89     D=np.diag(Sigma)
90     print("X:", X.shape)
91     print("U:", U.shape)
92     print("Sigma:", Sigma.shape)
93     print("V^T:", VT.shape)
94
95
96
97     ile=5
98     nr=1
99
100
101     fig0 = plt.figure(1)
102
103
104
105     fig = plt.figure(1)
106     st = fig.suptitle("Eigenfaces ", fontsize="x-large")
107
108
109
110     for i in range(0,ile):
111         for j in range(0, ile):
112             plt.subplot(ile,ile,nr);
113             #plt.title("Eigenface " + str(nr))
114             eigenface=U[:,nr-1].reshape(h,w)
115             #if(nr==1):
116             eigenface=-eigenface
117             #eigenface=X[:,nr-1].reshape(h,w)
118             plt.imshow(eigenface, cmap=plt.get_cmap('gray'))
119
120             plt.axis('off')
121             nr=nr+1
122
123
124     nr_rec = 4
125     nr_rec2 = 6
126
127     #reconstruct
128     k=15
129
130
131
132     nr=1
133
134
135
136     fig2 = plt.figure(2)
137     st = fig2.suptitle("Reconstructed faces, max dim="+str(D.shape[0]), fontsize="x-large")
138
139
140     for i in range(0,nr_rec):
141         im_nr = np.random.randint(X.shape[1])
142
143         face_orig = X[:,im_nr].reshape(h,w)
144         plt.subplot(nr_rec,nr_rec2+1,nr)
145
146         plt.imshow(face_orig, cmap=plt.get_cmap('gray'))
147         plt.title("Person "+str(im_nr))
148         plt.axis('off')
149         nr=nr+1
150         dims=2
151         for j in range(0,nr_rec2):
152             plt.subplot(nr_rec,nr_rec2+1,nr)
153             if(j==nr_rec2-2):
154                 dims=int(0.6*D.shape[0]-1)
155
156             if(j==nr_rec2-1):
157                 dims=int(0.8*D.shape[0]-1)
158
159
160         X_rek=np.dot(U[:, :dims], np.dot(D[:dims, :dims], VT[:dims, :]))

```

```

161
162     face_rek = X_rek[:,im_nr].reshape(h,w)
163     plt.imshow(face_rek, cmap=plt.get_cmap('gray'))
164
165     plt.title("dim = "+str(dims))
166     plt.axis('off')
167     nr=nr+1
168     dims=dims+2
169
170
171
172
173
174
175 # ~ model = decomposition.NMF(n_components=10, init='random', random_state=0)
176 # ~ W=model.fit_transform(X)
177 # ~ H=model.components_
178
179
180
181 if(alg=="nmf" or alg=="NMF"):
182
183
184     p_red=min(int(h/2),int(w/2))
185
186     print("Calculating NMF...", end="", flush=True)
187     start_time = time.time()
188
189     model = decomposition.NMF(n_components=p_red, init='random', random_state=0)
190     W=model.fit_transform(X)
191     H=model.components_
192     print("\t\t took %s seconds " % round((time.time() - start_time),5))
193
194
195
196     nr=1;
197     ile=4
198     how_much_rec=0
199     for i in range(0,ile):
200         for j in range(0, ile):
201             how_much_rec+=8;
202             fig5=plt.figure(5);
203             plt.subplot(ile,ile,nr);
204             k=how_much_rec
205             #rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
206             M_recons=np.dot(W[:,0:k],H[0:k,:])
207             plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
208             plt.title("i = " + str(how_much_rec))
209             plt.axis('off')
210             nr=nr+1
211
212     fig5.suptitle("NMF reconstructed")
213
214
215     nr=1;
216     ile=4
217     how_much_rec=0
218     for i in range(0,ile):
219         for j in range(0, ile):
220             how_much_rec+=1;
221             fig6=plt.figure(6);
222             plt.subplot(ile,ile,nr);
223             k=how_much_rec
224             #rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
225             M_recons=np.dot(W[:,(k-1):k],H[(k-1):k,:])
226             plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
227             plt.title("i = " + str(how_much_rec))
228             plt.axis('off')
229             nr=nr+1
230
231     fig6.suptitle("NMF eigenimages")
232
233 plt.show()
234
235
236 quit()
237
238
239
240

```