1. 课程内容
   * + 1. 了解人脸识别的**训练**和**测试**过程.
       2. 以Python实现人脸识别的训练和测试.
       3. 学习使用mglearn, skimage, matplotlib, numpy, face\_recognition, os, sklearn等Python库.
2. 人脸识别原理回顾
   1. 机器学习

机器学习分为两大类:**监督学习和非监督学习**. 非监督学习中的数据**没有标签**,可用的算法主要有**降维和聚类**. 监督学习中的数据**具有标签**,即机器学习算法利用已有的结果去预测新出现的数据.监督学习可分为两类:**分类算法**和**回归算法**. 分类和回归,二者既有区别,本质上又是统一的.

人脸识别属于分类算法. 分类算法有很多种,例如:逻辑回归算法,决策树算法,支持向量机算法,**k-近邻算法(KNN)**等.

对数千人的数百万张照片重复这个过程数百万次, 这个神经网络学会了可靠地对每一个人产生这128个测量值. 同一个人的任意数十张不同照片将会给出基本相同的测量值! **将图片这样的复杂数据简化为由数字构成的列表**这样的思想在机器学习领域常常出现．

* 1. 为人脸编码

我们需要亲自做的就是：使我们的人脸照片输入至他们已经训练好的神经网络中以得到那128个测量值. 这些数值分别测量的是什么不是我们关心的.重要的是这个网络在看同一个人的两张不同的照片时，产出**几乎一样的数**.

对图片中的人脸编码, 函数 face\_recognition.face\_encodings()

模块face\_recognition.api中的函数face\_encodings() .

1. **def** face\_encodings(face\_image, known\_face\_locations=None, num\_jitters=1)
2. """
3. 对给定的图片中的每一张人脸, 返回128-dimension人脸编码.  X
4. :param face\_image: 包含一张或多张人脸之图片(一个numpy数组)
5. :param known\_face\_locations: 可选 - 已经认识的每张脸的bounding boxes.
6. :param num\_jitters: 计算编码时,re-sample的次数. 其值越高, 越精确, 但更慢 (i.e. 100 is 100x slower)
7. :return: 128-dimensional人脸编码的列表(对图片中每一张脸都会生成这样一个列表)
8. """
9. ...

这个人脸编码的列表会加到特征列表X中, 同时把其所在的文件夹名(即人名)加入到列表y中. 这样,一条记录就产生了! 对每一个文件夹中的图片文件进行同样的操作,就可以对每一张图片中的人脸都得到一条记录(样本).这就有了一个行128+1=129列的**数据集**,其中为有效照片的张数.

* 1. KNeighborsClassifier类的使用

knn\_clf = neighbors.KNeighborsClassifier(n\_neighbors=n\_neighbors, algorithm=knn\_algo, weights='distance')

knn\_clf.fit(X, y)

1. 人脸识别和应用:
   1. 训练模型

训练模型. 定义一个函数train()

1. **def** train(train\_dir, model\_save\_path='trained\_knn\_model.clf', n\_neighbors=3,
2. knn\_algo='ball\_tree'):
3. """
4. 功能: 训练一个KNN分类器.
5. :param train\_dir: 训练目录.其下对每个已知的人,分别以其名字,建立一个文件夹. 该目录的结构如下:
6. <train\_dir>/
7. ├── <甲>/
8. │   ├── <甲图1>.jpeg
9. │   ├── <甲图2>.jpeg
10. │   ├── ...
11. ├── <乙>/
12. │   ├── <乙图1>.jpeg
13. │   └── <乙图2>.jpeg
14. └── ...
15. :param model\_save\_path: (optional)
16. :param n\_neighbors: (可选) 邻居的数.
17. 有默认值.
18. :param knn\_algo: (可选) 支持KNN的数据结构.
19. :return: KNN分类器.
20. """
21. …
    1. 预测

定义一个函数predict(),用于预测**分类**结果.

1. **def** predict(X\_img\_path, knn\_clf=None, model\_path=None, distance\_threshold=0.35):
2. """
3. 利用KNN分离器识别给定照片中的人脸
4. :param X\_img\_path: 待识别照片的路径
5. :param knn\_clf: (可选) KNN分离器对象.如果没有指定其值, 则必须指定model\_save\_path的值.
6. :param model\_path: (可选) 放置KNN分离器字节流的路径.如果没有指定, model\_save\_path的值必须为 knn\_clf. (“Pickling” 意思是将Python对象转换成字节流. )
7. :param distance\_threshold: (可选)人脸分类的距离阈值. 其值越大,则对于一个未知人脸的识别的错误率就越高. (如果一个训练集足够大,我们可以降低这个参数的值,以增加识别准确率.
8. :return: 返回这张照片中的人名和边界盒子的元组构成的列表: [(人名1, 边界盒子1),(人名2,边界盒子2), ...]. 未识别的人脸,返回名字'unknown'.
9. """
10. …
    1. 显示人脸识别的结果

定义函数show\_names\_on\_image(),显示人脸识别结果.

1. **def** show\_names\_on\_image(img\_path, predictions):
2. """
3. 显示人脸识别结果(可视化).
4. :param img\_path: 待识别图片的位置
5. :param predictions:预测的结果
6. :return:
7. """
8. ...

例.**人脸识别**作为自动点名系统代码.

1. # -\*- coding: utf-8 -\*-
2. #file:05\_face\_recog\_knn.py
4. #===========
5. #1.导入模块
6. #===========
7. **from** sklearn **import** neighbors
8. **import** os
9. **import** os.path
10. **import** pickle
11. **from** PIL **import** Image, ImageDraw
12. **import** face\_recognition as fr
13. **from** face\_recognition.cli **import** image\_files\_in\_folder
15. #=========
16. #2.函数定义
17. #=========
18. **def** train(train\_dir, model\_save\_path='trained\_knn\_model.clf', n\_neighbors=3,
19. knn\_algo='ball\_tree'):
20. """
21. 训练一个KNN分类器.
22. :param train\_dir: 训练目录.其下对每个已知的人,分别以其名字,建立一个文件夹.
23. :param model\_save\_path: (optional)
24. :param n\_neighbors:
25. 有默认值.
26. :param knn\_algo: (optional) 支持KNN的数据结构.
27. :return: KNN分类器.
28. """
30. #生成训练集
31. X = []
32. y = []
34. #遍历训练集中的每一个人
35. **for** class\_dir **in** os.listdir(train\_dir):
36. **if** **not** os.path.isdir(os.path.join(train\_dir, class\_dir)):
37. **continue** #结束当前循环, 进入下一个循环
39. # 遍历这个人的每一张照片
40. **for** img\_path **in** image\_files\_in\_folder(os.path.join(train\_dir, class\_dir)):
41. image = fr.load\_image\_file(img\_path)
42. boxes = fr.face\_locations(image)
44. # 对于当前图片,增加编码到训练集
45. X.append(fr.face\_encodings(image,
46. known\_face\_locations=boxes)[0])
47. y.append(class\_dir)
49. # 决定k值for weighting in the KNN classifier
50. **if** n\_neighbors **is** None:
51. #n\_neighbors = int(round(math.sqrt(len(X))))
52. n\_neighbors = 3
54. # 创建并训练分类器
55. knn\_clf = neighbors.KNeighborsClassifier(n\_neighbors=n\_neighbors)
56. knn\_clf.fit(X, y)
58. # 保存训练好的分类器
59. **if** model\_save\_path **is** **not** None:
60. with open(model\_save\_path, 'wb') as f:
61. pickle.dump(knn\_clf, f)
63. **return** knn\_clf
65. **def** predict(X\_img\_path, knn\_clf=None, model\_path=None, distance\_threshold=0.45):
66. """
67. 利用KNN分离器识别给定照片中的人脸
68. :return: [(人名1, 边界盒子1), ...]
69. """
70. **if** knn\_clf **is** None **and** model\_path **is** None:
71. **raise** Exception("必须提供KNN分类器:可选方式为 knn\_clf 或 model\_path")
73. # 加载训练好的KNN模型(如果有)
74. # rb 表示要读入二进制数据
75. **if** knn\_clf **is** None:
76. with open(model\_path, 'rb') as f:
77. knn\_clf = pickle.load(f)
79. # 加载图片,发现人脸的位置
80. X\_img = fr.load\_image\_file(X\_img\_path)
81. X\_face\_locations = fr.face\_locations(X\_img)

84. # 对测试图片中的人脸编码
85. encodings = fr.face\_encodings(X\_img,
86. known\_face\_locations=X\_face\_locations)
88. # 利用KNN model 找出与测试人脸最匹配的人脸
89. # encodings: 128个人脸特征构成的向量
90. closest\_distances = knn\_clf.kneighbors(encodings, n\_neighbors=1)
91. are\_matches = [closest\_distances[0][i][0] <= distance\_threshold
92. **for** i **in** range(len(X\_face\_locations))]
94. # 预言类别,并 remove classifications that aren't within the threshold
95. **return** [(pred, loc) **if** rec **else** ("unknown", loc)
96. **for** pred, loc, rec **in** zip(knn\_clf.predict(encodings),
97. X\_face\_locations, are\_matches)]

100. **def** show\_names\_on\_image(img\_path, predictions):
101. """
102. 人脸识别可视化.
103. :param img\_path: 待识别图片的位置
104. :param predictions:预测的结果
105. """
106. pil\_image = Image.open(img\_path).convert("RGB")
107. draw = ImageDraw.Draw(pil\_image)
109. **for** name, (top, right, bottom, left) **in** predictions:
110. # 用Pillow模块画出人脸边界盒子
111. draw.rectangle(((left, top), (right, bottom)), outline=(255, 0, 255))
113. # pillow里可能生成非UTF-8格式,所以这里做如下转换
114. name = name.encode("UTF-8")
115. name = name.decode("ascii") #L add
117. # 在人脸下写下名字,作为标签
118. text\_width, text\_height = draw.textsize(name)
119. draw.rectangle(((left, bottom - text\_height - 10), (right, bottom)),
120. fill=(255, 0, 255), outline=(255, 0, 255))
121. draw.text((left + 6, bottom - text\_height - 5), name, fill=(255, 255, 255))
123. # 追加名字到列表li\_names
124. li\_names.append(name)
126. # 从内存删除draw
127. **del** draw
129. # 显示结果图
130. pil\_image.show()

133. #========
134. #统计分析
135. #========
136. # 为了打印名字的集合
137. li\_names = []
139. # 计算总人数
140. **def** count(train\_dir):
141. """
142. Counts the total number of the set.
143. """
144. path = train\_dir
145. count = 0
146. **for** fn **in** os.listdir(path): #fn 表示的是文件名
147. count = count + 1
148. **return** count
150. # 获取所有名字的列表
151. **def** list\_all(train\_dir):
152. """
153. Determine the list of all names.
154. """
155. path = train\_dir
156. result = []
157. **for** fn **in** os.listdir(path): #fn 表示的是文件名
158. result.append(fn)
159. **return** result
161. # 输出结果
162. **def** stat\_output():
163. s\_list = set(li\_names)
164. s\_list\_all = set(list\_all("examples/train"))
165. **if** "unknown" **in** s\_list:
166. s\_list.remove("unknown")
168. tot\_num = count("examples/train")
169. s\_absent = set(s\_list\_all - s\_list)
170. **print**("\n")
171. **print**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")
172. **print**("全体名单:",s\_list\_all)
173. **print**("已到名单:",s\_list)
174. **print**("应到人数:",tot\_num)
175. **print**("已到人数:",len(s\_list))
176. **print**("出勤率:{:.2f}".format(float(len(s\_list))/float(tot\_num)))
177. **print**("未到:",s\_absent)

180. **if** \_\_name\_\_ == "\_\_main\_\_":
181. # 1 训练KNN分类器(它可以保存,以便再用)
182. **print**("正在训练KNN分类器...")
183. classifier = train("examples/train", model\_save\_path="trained\_knn\_model.clf",
184. n\_neighbors=2)
185. **print**("完成训练!")
187. # 2 利用训练好的分类器,对新照片进行预测
188. **for** image\_file **in** os.listdir("examples/test"):
189. full\_file\_path = os.path.join("examples/test", image\_file)
191. **print**("在{}中寻找人脸...".format(image\_file))
193. # 利用分类器,找出所有的人脸;
194. # 要么传递一个classifier文件名,要么一个classifier模型实例
195. predictions = predict(full\_file\_path, model\_path="trained\_knn\_model.clf")
197. # 打印结果
198. **for** name, (top, right, bottom, left) **in** predictions:
199. **print**("发现{}, 位置: ({}, {},{},{})".format(name, top, right, bottom, left))
201. # 在图片上显示预测结果
202. show\_names\_on\_image(os.path.join("examples/test", image\_file), predictions)
204. # 3.输出统计结果
205. stat\_output()
     1. 人脸识别可能的应用:
        * 1. 点名系统,统计名单和人数.
          2. 侦察,防盗.
          3. 推广到**目标识别**后,可以识别行人,路牌提示,用于自动驾驶技术.
206. 扩展
     1. 参考文献

[1] Dalal, N. and Triggs, B., “Histograms of Oriented Gradients for Human Detection,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA.

[2] David G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

[3] Adam Geitgey, Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning

[4] http://dlib.net/python/index.html

[5] 部分函数功能介绍.

pickle.dump(): pickle模块中的内置函数dump() .它可将对象的表示存入文件对象.

函数load\_image\_file()

1. **def** load\_image\_file(file, mode='RGB')
2. """
3. 加载一个图片文件(.jpg, .png, etc)到一个numpy列表.
4. :param file: 图片名称或待加载的文件对象.
5. :param mode: 该参数指明将图片转换成何种格式(format). 仅支持 'RGB'(8-bit RGB, 3 channels) 和 'L' (black and white)两种格式.
6. :return: 一个numpy列表.
7. """
8. ...

函数 face\_recognition.face\_locations(): 模块face\_recognition.api中的函数face\_locations().

1. **def** face\_locations(img, number\_of\_times\_to\_upsample=1, model='hog')
2. """
3. 返回输入图片中的所有人脸的边界盒子(为一个数组)
4. :param img: 一张图片(作为一个numpy数组)
5. :param number\_of\_times\_to\_upsample: 寻找人脸时对图片进行上采样的次数.此数值越高,则可以探测到越小的人脸. (影像金字塔,往上走)
6. :param model: 所用的探测模型. "hog",或 "cnn". “cnn” is a more accurate deep-learning model which is GPU/CUDA accelerated (if available). The default is "hog".
7. :return: 元组构成的列表,每个元组给出找到的一张人脸的位置.位置按照(top, right, bottom, left) 顺序以css格式表示.
8. """
9. ...