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

2.1. 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.

2.2. 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() .

```
def face_encodings(face_image, known_face_locations=None, num_jitters=1)
    """
    对给定的图片中的每一张人脸,返回 128-dimension 人脸编码. X
    :param face_image: 包含一张或多张人脸之图片(一个 numpy 数组)
    :param known_face_locations: 可选 - 已经认识的每张脸的 bounding boxes.
    :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)
    :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)
    """
```

```
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N行 128+1=129 列的**数据集**，其中N为有效照片的张数.

2.3. KNeighborsClassifier 类的使用

```
knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors, algorithm=knn_algo, weights='distance')
knn_clf.fit(X, y)
```

3. 人脸识别和应用:

3.1. 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.     ...
```

3.2. 预测

定义一个函数 predict(),用于预测**分类**结果.

```
def predict(X_img_path, knn_clf=None, model_path=None, distance_threshold=0.35):
    """
    利用 KNN 分离器识别给定照片中的人脸
    :param X_img_path: 待识别照片的路径
    :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.
    :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. )
    :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.
    :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'.
    """
    ...
```

3.3. 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.

```
def show_names_on_image(img_path, predictions):
    """
    显示人脸识别结果(可视化).
    :param img_path: 待识别图片的位置
    :param predictions:预测的结果
    :return:
    """
    ...
```

例. **人脸识别**作为自动点名系统代码.

```
# -*- coding: utf-8 -*-
#file:05_face_recog_knn.py

#============
#1.导入模块
#============
from sklearn import neighbors
import os
```

```
import os.path
import pickle
from PIL import Image, ImageDraw
import face_recognition as fr
from face_recognition.cli import image_files_in_folder

#==========
#2.函数定义
#==========
def train(train_dir, model_save_path='trained_knn_model.clf', n_neighbors=3,
          knn_algo='ball_tree'):
    """
    训练一个 KNN 分类器.
    :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.
    :param model_save_path: (optional)
    :param n_neighbors:
    有默认值.
    :param knn_algo: (optional) 支持 KNN 的数据结构.
    :return: KNN 分类器.
    """

    #生成训练集
    X = []
    y = []

    #遍历训练集中的每一个人
    for class_dir in os.listdir(train_dir):
        if not os.path.isdir(os.path.join(train_dir, class_dir)):
            continue #结束当前循环, 进入下一个循环

        # 遍历这个人的每一张照片
        for img_path in image_files_in_folder(os.path.join(train_dir, class_dir)):
            image = fr.load_image_file(img_path)
            boxes = fr.face_locations(image)

            # 对于当前图片,增加编码到训练集
            X.append(fr.face_encodings(image,
                    known_face_locations=boxes)[0])
            y.append(class_dir)

    # 决定 k 值 for weighting in the KNN classifier
    if n_neighbors is None:
        #n_neighbors = int(round(math.sqrt(len(X))))
        n_neighbors = 3
```

```
53.
54.     # 创建并训练分类器
55.     knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors)
56.     knn_clf.fit(X, y)
57.
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)
62.
63.     return knn_clf
64.
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")
72.
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)
78.
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)
82.
83.
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)
87.
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))]
93.
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),
```

```
                X_face_locations, are_matches)]

def show_names_on_image(img_path, predictions):
    """
    人脸识别可视化.
    :param img_path: 待识别图片的位置
    :param predictions:预测的结果
    """
    pil_image = Image.open(img_path).convert("RGB")
    draw = ImageDraw.Draw(pil_image)

    for name, (top, right, bottom, left) in predictions:
        # 用 Pillow 模块画出人脸边界盒子
        draw.rectangle(((left, top), (right, bottom)), outline=(255, 0, 255))

        # pillow 里可能生成非 UTF-8 格式,所以这里做如下转换
        name = name.encode("UTF-8")
        name = name.decode("ascii") #L add

        # 在人脸下写下名字,作为标签
        text_width, text_height = draw.textsize(name)
        draw.rectangle(((left, bottom - text_height - 10), (right, bottom)),
                       fill=(255, 0, 255), outline=(255, 0, 255))
        draw.text((left + 6, bottom - text_height - 5), name, fill=(255, 255, 255))

        # 追加名字到列表 li_names
        li_names.append(name)

    # 从内存删除 draw
    del draw

    # 显示结果图
    pil_image.show()


#========
#统计分析
#========
# 为了打印名字的集合
li_names = []

# 计算总人数
def count(train_dir):
```

```
    """
    Counts the total number of the set.
    """
    path = train_dir
    count = 0
    for fn in os.listdir(path): #fn 表示的是文件名
        count = count + 1
    return count

# 获取所有名字的列表
def list_all(train_dir):
    """
    Determine the list of all names.
    """
    path = train_dir
    result = []
    for fn in os.listdir(path): #fn 表示的是文件名
        result.append(fn)
    return result

# 输出结果
def stat_output():
    s_list = set(li_names)
    s_list_all = set(list_all("examples/train"))
    if "unknown" in s_list:
        s_list.remove("unknown")

    tot_num = count("examples/train")
    s_absent = set(s_list_all - s_list)
    print("\n")
    print("*****************************************\n")
    print("全体名单:",s_list_all)
    print("已到名单:",s_list)
    print("应到人数:",tot_num)
    print("已到人数:",len(s_list))
    print("出勤率:{:.2f}".format(float(len(s_list))/float(tot_num)))
    print("未到:",s_absent)


if __name__ == "__main__":
    # 1 训练 KNN 分类器(它可以保存,以便再用)
    print("正在训练 KNN 分类器...")
    classifier = train("examples/train", model_save_path="trained_knn_model.clf",
                       n_neighbors=2)
```

```
print("完成训练!")

# 2 利用训练好的分类器,对新照片进行预测
for image_file in os.listdir("examples/test"):
    full_file_path = os.path.join("examples/test", image_file)

    print("在{}中寻找人脸...".format(image_file))

    # 利用分类器,找出所有的人脸;
    # 要么传递一个 classifier 文件名,要么一个 classifier 模型实例
    predictions = predict(full_file_path, model_path="trained_knn_model.clf")

    # 打印结果
    for name, (top, right, bottom, left) in predictions:
        print("发现{}, 位置: ({}, {},{},{})".format(name, top, right, bottom, left))

    # 在图片上显示预测结果
    show_names_on_image(os.path.join("examples/test", image_file), predictions)

# 3.输出统计结果
stat_output()
```

3.4. 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.

4. 扩展

4.1. 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()

```
def load_image_file(file, mode='RGB')
    """
    加载一个图片文件(.jpg, .png, etc)到一个 numpy 列表.
    :param file: 图片名称或待加载的文件对象.
    :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.
    :return: 一个 numpy 列表.
    """
    ...
```

函数 face_recognition.face_locations(): 模块 face_recognition.api 中的函数 face_locations().

```
def face_locations(img, number_of_times_to_upsample=1, model='hog')
    """
    返回输入图片中的所有人脸的边界盒子(为一个数组)
    :param img: 一张图片(作为一个 numpy 数组)
    :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)
    :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".
    :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.
    """
    ...
```