1. 课程内容

1. 了解人脸校正和编码的基本过程.
2. K-近邻算法.
3. 学习使用 mglearn, skimage, matplotlib, numpy 等 Python 库.

2. 人脸校正(face alignment)和变换

前面已经知道,我们分四个步骤来处理人脸识别.

第一,人脸探测;

第二,人脸校正和变换;

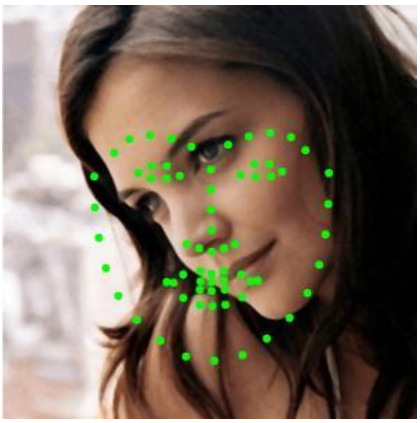
第三,人脸编码;

第四,区分出人脸.

对每一步, 我们将学习一个不同的机器学习算法. 我们将会学习每一个算法背后的基本思想. 我们已经可以找出所有的人脸. 下面介绍后面的步骤. 第二步: 人脸校正和变换.

基本思想:我们将选出每张人脸上的 68 个特殊点 (**landmarks**) ---- 如眼睛的外边界, 眉毛的内边界, 等. 然后我们将运用一个机器学习算法在任意一张人脸上找出这些特殊点:

当我们知道人脸的双眼和嘴的位置, 我们将简单地旋转, 伸缩和剪切(shear)这张图片, 使得双眼和嘴尽可能地在每张图中都在**相同**的位置. 我们运用基本的图形变换, 如保持平行线仍然平行的旋转变换和伸缩变换 (仿射变换(affine transformations)). 这一步操作为下一步提供了很大的方便.



达到的目标: 无论人脸如何转动, 我们都能将眼睛和嘴基本上置于照片中心固定的位置. 这将使我们下一步更精确. 这里可以用 Kazemi 和 Sullivan 等人在 2014 发明的技术[0]:

One Millisecond Face Alignment with an Ensemble of Regression Trees

Vahid Kazemi and Josephine Sullivan
KTH, Royal Institute of Technology
Computer Vision and Active Perception Lab
Teknikringen 14, Stockholm, Sweden
{vahidk,sullivan}@csc.kth.se

Abstract

This paper addresses the problem of Face Alignment for a single image. We show how an ensemble of regression trees can be used to estimate the face's landmark positions directly from a sparse subset of pixel intensities, achieving super-realtime performance with high quality predictions. We present a general framework based on gradient boosting for learning an ensemble of regression trees that optimizes the sum of square error loss and naturally handles missing or partially labelled data. We show how using appropriate priors exploiting the structure of image data helps with efficient feature selection. Different regularization strategies and its importance to combat overfitting are also investigated. In addition, we analyse the effect of the quantity of training data on the accuracy of the predictions and explore the effect of data augmentation using synthesized data.

Figure 1. Selected results on the HELEN dataset. An ensemble

2.1.1. Boosting 方法 (了解)

Boosting 方法最初来源于**分类问题**，但也可以拓展到**回归问题**. （正常邮件,垃圾邮件）(0, r)
Boosting 方法的动机：是将许多**弱分类器组合**起来构成一个强大的分类器集体.
由此，boosting 方法与 bagging 方法以及其他基于 " 分类器集体 " 的方法相似.
但是这些联系是表面的，boosting 方法和它们有根本的不同.

最流行的 boosting 算法(Freund and Schapire, 1997)： “AdaBoost.M1”

举例，对二分类问题,分类器的**错误率**定义为：

$$err = \frac{1}{N}\sum_{\{i=1\}}^{N} I\left(y_i \neq G(x_i)\right)$$

其中，$y_i \in \{-1,1\}$，分类器用$G(x_i)$表示.

弱分类器(weak classifier)：错误率仅仅比随机猜测略好一点的分类器.

boosting 方法的目标： 相继应用弱分类器算法以不断地修改数据的版本， 因此产生一系列分类器 $G_m(x)$, $m = 1,2,\ldots,M$.

最终的预测器：

$$G(x) = sign\left[\sum_{1}^{M} \alpha_m G_{m(x)}\right]$$

此处$\alpha_1, \alpha_2, \ldots, \alpha_M$. 由 boosting 算法计算而得，并为各个分类器$G_m(x)$提供权重.

该算法的目标:

以一种有效的计算方法来精确地估算人脸(facial landmarks)的位置.

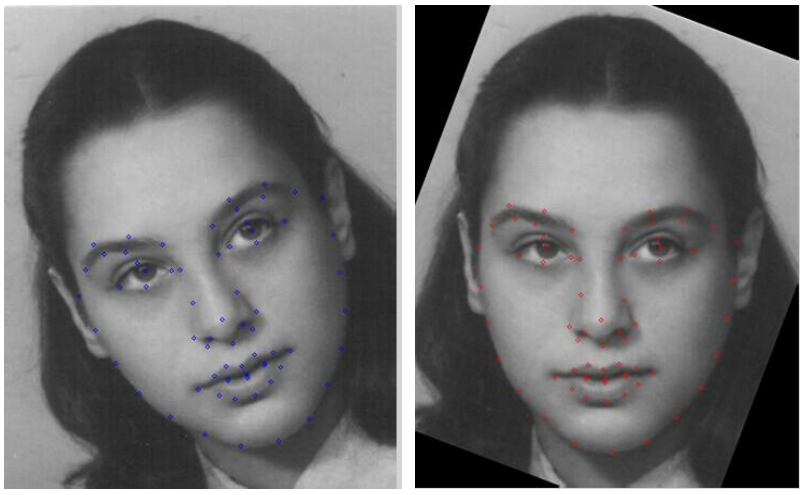
2.1.2. 仿射变换后人脸图像及关键点

仿射变换将原坐标(x, y)变换为新坐标(x', y'):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} A_{00}x + A_{01}y + A_{02} \\ A_{10}x + A_{11}y + A_{12} \\ 1 \end{pmatrix}$$

通过这公式，可计算出原图像经过变换后的新图像.

根据眼睛坐标进行人脸对齐,计算变换后对应坐标. 图片如下:



人脸图像变换前后及 68 个面部关键点

<src='http://www.mcudiy.cn/attachment/Mon_1407/33_8_eca362c3f10fea4.jpg?29'>

<src="http://www.mcudiy.cn/attachment/Mon_1407/33_8_c1ecd47da715b3e.jpg?28">

3. 人脸的编码

现在开始区分不同的人脸！最简单的人脸识别方法是直接将第二步中**找出的未知人脸**与我们已经带有标签的**所有**照片进行对照． 当我们发现一张已带有标签的人脸与未知人脸非常相似时,那它们一定是同一个人的脸!

但是,这个方法有一个问题． 具有上亿用户和照片的网站**不可能**遍历每一张有标签的人脸来与新上传的照片进行对比．那样太耗时． 人脸识别应该在数毫秒内完成．

3.1. 测量人脸的可靠方法

我们需要**提取人脸的特征值**！ 我们的方法是：提取每一张人脸的几个基本测量值．然后我们可以以同样的方式测量**新的**人脸照片，并找出与之有**最接近**的测量值的已知人脸． 例如，我们可能测量每一只耳朵的尺寸，两眼间距，鼻子长度等.

我们应该从每一张脸上收集哪些测量值来建立我们的带有标签的＂熟人＂数据库呢？ 耳朵大小？眉毛长度？ 眼睛大小？ 鼻子宽度？ 最精确的方法是**让计算机自己去决定该收集哪些测量值**．对于人脸的哪些测量值最重要这一问题，深度学习算法是一种很好的方法！ 具体的解决办法是训练一个**深度卷积神经网络**，并训练它对每张人脸产生 128 个测量值.

3.2. 为人脸编码

即使有大型计算机， 训练神经网络也非常耗时．但是一旦神经网络训练完成，它便可以对任意一张从未见过的人脸产生出测量值．所以，训练只需要运行一次！ OpenFace 的研究人员已经训练除了一些神经网络，我们可以直接使用 (参考 Brandon Amos and team)！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**.

问：这 128 个数值**分别**测量的是什么呢?
我们关注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 .

```python
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.
    :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)
    """

    ...
```

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.

3.3. KNeighborsClassifier 类的使用

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:回归算法和分类算法. 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. 我们今天用到 k-近邻算法.

先从具体的一个例子开始.

例1. 训练集{X,y}. 本例展示数据集的特征的散点图.

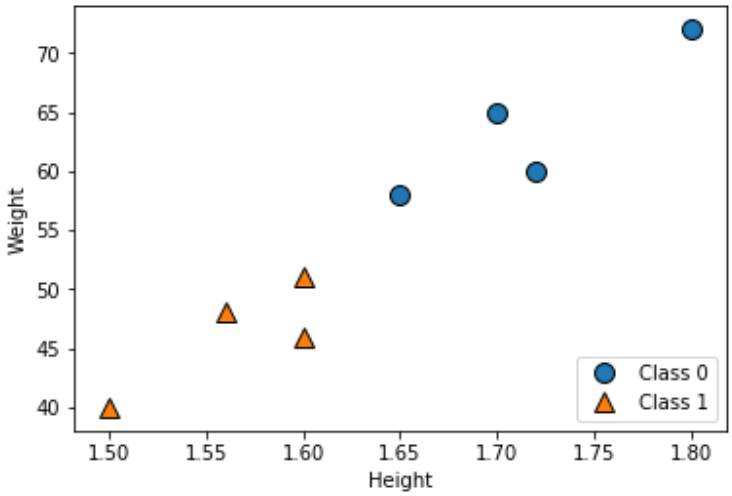
```
1.  # -*- coding: utf-8 -*-
2.  # 本例将具有两个特征的数据集作散点图,并用 knn 分类
3.  # 安装 mglearn: pip install mglearn
4.  import mglearn
5.  import matplotlib.pyplot as plt
6.  import numpy as np
7.  # 生成身高和体重的数据集
8.  # X,y 都表示为 numpy 数组,方便用 mglearn 作散点图
9.  X = np.array([[1.5, 40], [1.56, 48], [1.6,51], [1.6, 46], [1.7,65], [1.65,58], [1.72,60], [1.8, 72]])
10. y = np.array([1, 1, 1,1, 0, 0, 0, 0])
11. # 对数据集作图
12. mglearn.discrete_scatter(X[:,0], X[:,1],y)
13. plt.legend(["Class 0", "Class 1"], loc=4)
14. plt.xlabel("Height")
15. plt.ylabel("Weight")
16. print("X.shape:{}".format(X.shape))
17. plt.show()
```

运行:

```
python ex_scattering.py
```

结果如下图:

<img src="https://github.com/hg08/tututu/blob/master/fig_ex_scattering.png?raw=true">



训练和测试:

```
(base) C:\WINDOWS\system32>python
Python 3.5.5 |Anaconda custom (64-bit)| (default, Apr  7 2018, 04:52:34) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from sklearn.neighbors import KNeighborsClassifier
>>> X = [[1.5, 40], [1.56, 48], [1.6, 51], [1.7, 65], [1.72,60], [1.8,72]]
>>> y = [1,1,1,0,0,0]
>>> neigh =KNeighborsClassifier(n_neighbors=1)
>>> neigh.fit(X,y)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=1, p=2,
          weights='uniform')
>>> print(neigh.predict([[1.5,43]]))
[1]
```

例 3.　我们从一个表示我们的数据集的数组出发,我们构建一个 NeighborsClassifier 类, 并提问: 哪一个点离[1, 1, 0.5] 最近?

Python 交互式代码如下:

```
>>> from sklearn.neighbors import NearestNeighbors
>>> samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .8]]
>>> neigh = NearestNeighbors(n_neighbors=1)
>>> neigh.fit(samples)
NearestNeighbors(algorithm='auto', leaf_size=30, metric='minkowski',
         metric_params=None, n_jobs=1, n_neighbors=1, p=2, radius=1.0)
```

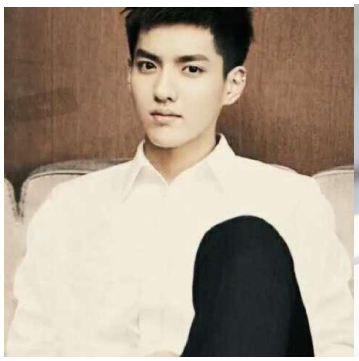
```
>>> print(neigh.kneighbors([[1., 1., .5]]))
(array([[ 0.3]]), array([[2]]))
>>>
>>> X = [[0., 1., 0.], [1., 0., .6]]
>>> neigh.kneighbors(X, return_distance=False)
array([[1],
       [2]])
```

结果:
程序返回[[0.3]]和 [[2]].这意味着这个元素 is at distance 0.3, 是第三个元素.(注意:从 0 开始计数).

3.4. 训练过程的工作方式(了解)

训练过程通过同时看 3 张人脸图片来工作:

1. 加载一个熟人的训练人脸照片(左);
2. 加载同一个熟人的另一张照片(中);
3. 加载另外一个人的一张照片(右).



照片 1　　照片 2　　照片 3

然后, 这个算法考查这些三张照片中在当前产生的各自测量值. 然后,算法轻微地调整神经网络以确保照片 1 和照片 2 的测量值微微接近,而使得照片 2 和照片 3 的测量值微微远离:

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! 在机器学习领域,人们称每张人脸中的这 128 个测量值为 一个 " 嵌入 " (embedding). **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. 我们这里用的方法是 2015 年谷歌的研究人员发明的.

4. 课程总结

4.1. 重点

1. 了解人脸校正/对齐的算法：Boosting 算法的优点
2. 了解编码的目的和基本原理
3. KNN 分类器的使用

4.2. 难点

1. KNN 分类器的使用

4.3. 扩展

4.3.1. 参考文献

[0]Vahid 和 Sullivan 的技术主要贡献：基于一系列回归树(regression trees)的新的对齐方法,实现了不依赖于形状的特征选择.

[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