华东师范大学数据科学与工程学院实验报告

课程名称：统计学习与机器方法	年级：2019	实践成绩：
指导教师：董启文	**姓名**：周辛娜	**学号**：10195501442
上机实践： finalproject：人脸识别	**上机实践时间**：	2021.12.25

任务：

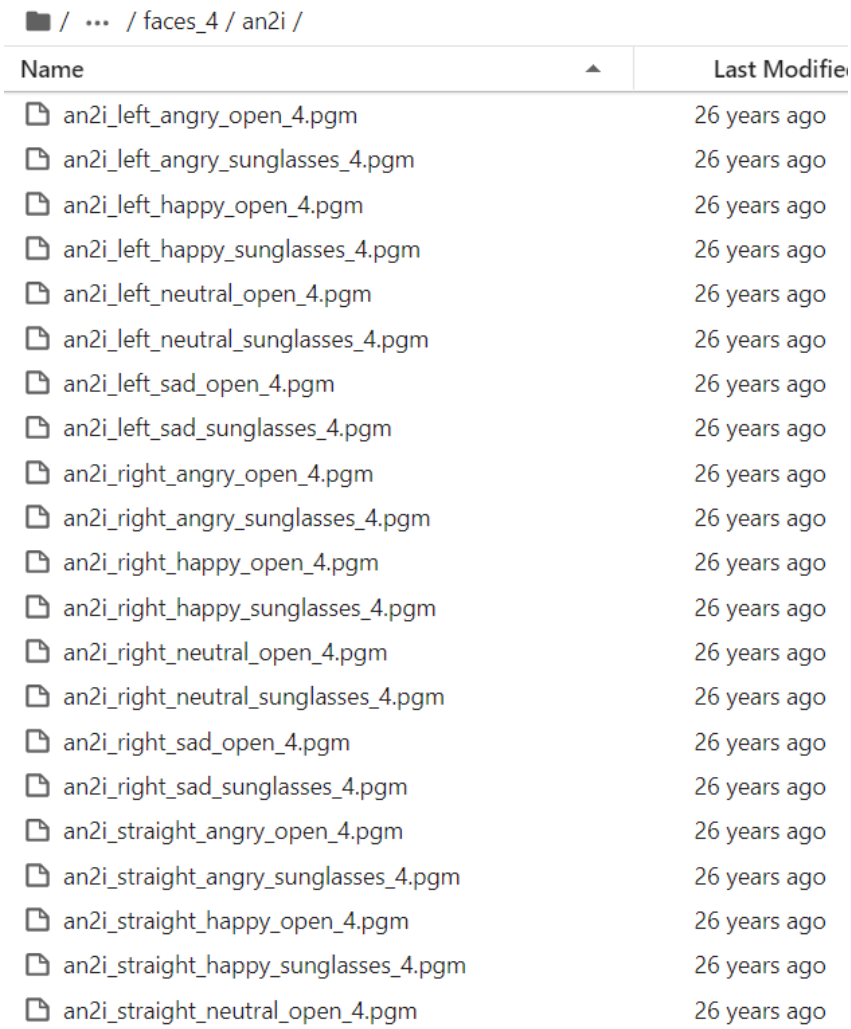
任务1：使用机器学习进行人类分类识别，给出识别准确率。

任务2：使用聚类或分类算法发现表情相似的脸图。

数据集：

来源于CMU Machine Learning Faces，Neural Networks for Face Recognition (cmu.edu)，里面包含20个人，每人32张脸图(含表情)。

一、探索数据集



1.得到name，direction，expression，wear_sunglasses

下载的数据集即faces_4如上图所示，其中包含了20个人的不同图片，进入第一个文件夹an2i，里面共有32张图片，图片的文件名表名了这张图片记录的是这个人的名字、方向direction、表情、有没有戴sunglasses，那么可以根据这个进行数据整理。如an2i_straight_neutral_open_4.pgm就可以被拆成an2i，straight，neutral，open，4.pgm，进而遍历faces_4中的每个人的32个文件名，得到4个分类标签，分别是name，direction，expression，wear_sunglasses，其中name，代表的人名，共有20类，direction代表脸的朝向，共有4类，expression代表表情，共有4类，wear_sunglasses代表有没有戴sunglasses，有2类。

最后得到的name，direction，expression，wear_sunglasses如下图所示：

```
['an2i', 'at33', 'boland', 'bpm', 'ch4f', 'cheyer', 'choon', 'danieln', 'glickman', 'karyadi', 'kawamura', 'kk49', 'megak', 'mitch
ell', 'night', 'phoebe', 'saavik', 'steffi', 'sz24', 'tammo']
['left', 'right', 'straight', 'up']
['angry', 'happy', 'neutral', 'sad']
['open', 'sunglasses']
```

2. 构造可以被用来训练的数据

这里借鉴了fetch_lfw_people这一个sklearn中自带的人脸数据集，它的返回对象sklearn.utils.Bunch，该对象类似于字典，分为5个属性：

'data'，'images'，'target'，'target_names'，'DESCR'。

这里我构造的数据分为5个属性：

data 图像特征、name、direction、expression、wear_sunglasses。

采用字典的方式。

这里一个重要的函数就是construct_data()

```
def construct_data(path):
    im,names,directions,expressions,sunglasses = [],[],[],[],[]
    for file in path:
        #构造data
        image = Image.open(file).convert("L")
        #如果图像的模式是“1”，“L”，“P”等模式，因为这些模式是8bit表示一个像素，
        #所以这些模式的颜色值的范围是在0~255，所以getdata（）的返回值的元素就不是上述的（R,G,B）的形式了，而是0~255中的任意一个数
        im.append(np.array(list(image.getdata())))
        #im.getdata()函数的返回值是一个sequence对象，sequence对象的每一个元素对应一个像素点R、G、B值
        #可以用list（）将sequence对象转为普通的sequence对象
        str1 = file.split('_')
        #因为这里包括了faces_4/，所以要把“/”分割开
        str2=str1[1].split('/')
#         print(str2)
        a=0
        for i in range(len(name)):
            if name[i]==str2[2]:
                a=i
        b=0
        for i in range(len(direction)):
            if direction[i]==str1[2]:
                b=i
        c=0
        for i in range(len(expression)):
            if expression[i]==str1[3]:
                c=i
        d=0
        for i in range(len(wear_sunglasses)):
            if wear_sunglasses[i]==str1[4]:
                d=i

        names.append(a)
        directions.append(b)
        expressions.append(c)
        sunglasses.append(d)

    return {'data': np.array(im), 'name': np.array(names), 'direction': np.array(directions), 'expression': np.array(expressions),'wear_sunglasses': np.array(sunglasses)}
```

而且还要将对应的数字对应到对应的类别，而不应该是字符串。

最后得到的数据如下图所示：

```
'name': array([ 1, 15, 10,  6,  8,  5,  9,  7, 15, 11,  6, 10,  6,  9,  4, 11, 16,
        4,  3, 13, 15, 16, 14, 10,  2,  5, 18, 18, 18,  7,  8,  7,  8, 10,
       15, 15,  3, 10,  0, 17, 14,  3,  6, 18,  5, 11, 11, 17, 18, 18,  6,
        5,  0, 13, 11, 10,  9,  4,  8,  7, 12,  1, 10, 14,  7,  5, 13,  2,
       12, 19,  2, 13,  8,  6, 18, 13,  8,  8,  2, 18,  0, 17, 12,  0, 11,
       16,  3,  5, 19,  0,  1, 12,  7, 10,  3,  1, 11,  9, 13, 13,  8,  0,
        6, 13, 16, 11, 13, 14, 10, 14,  4,  9, 17, 19, 18, 17, 15, 12, 15,
        1, 12,  8,  0, 12,  3,  5, 19,  2, 18, 13,  3,  4,  4,  1, 14, 17,
       16, 16,  4, 11,  1,  8, 14, 15, 12, 19, 10,  3,  8,  0,  8,  0,  9,
        1, 11,  0,  2,  9,  9, 14, 14,  5, 19, 12,  1, 18, 19,  7,  3,  3,
       10, 15, 15, 10, 14,  5,  2,  6, 10,  5,  6,  5,  6,  2, 14,  5, 15,
       17,  9, 19,  9,  0, 14, 12,  5, 12, 10, 18, 11, 11, 19,  9,  9,  9,
        3, 16,  2,  5,  4, 16, 17, 11, 12,  0,  7, 17,  9,  7, 12,  9,  6,
        1, 14,  3, 15, 14,  5,  1, 13,  9, 17, 19, 16, 14, 17,  4,  3,  7,
        7, 12,  2,  0, 18,  4, 18,  2,  5, 19, 11,  1,  2,  3, 17, 13,  1,
        6, 17,  8,  0,  7, 15, 16, 13,  8,  4, 12,  0,  1,  6,  7,  6, 12,
        0,  0, 10,  3, 16, 14,  0,  9, 19, 15,  8, 12,  3, 18, 17, 11,  4,
       16,  6, 18,  7, 19, 17, 16, 10, 17,  6, 10,  7, 12, 13, 16, 11]),
'direction': array([0, 0, 1, 0, 3, 2, 1, 3, 1, 2, 3, 2, 1, 3, 0, 0, 3, 1, 3, 3, 2, 0,
       1, 0, 3, 0, 0, 2, 2, 2, 3, 0, 2, 1, 2, 3, 3, 2, 0, 1, 3, 3, 0, 3,
       1, 3, 1, 3, 0, 0, 2, 2, 1, 2, 1, 0, 3, 3, 0, 1, 0, 2, 2, 3, 0, 1,
       1, 3, 3, 3, 1, 1, 3, 3, 3, 0, 0, 0, 2, 3, 3, 3, 1, 2, 1, 3, 1, 0,
       3, 0, 3, 3, 1, 0, 0, 3, 1, 0, 3, 2, 1, 1, 0, 1, 1, 3, 2, 3, 0, 3,
       0, 2, 1, 0, 0, 0, 1, 0, 1, 3, 2, 0, 2, 2, 2, 3, 0, 2, 0, 3, 3, 3,
       2, 0, 0, 1, 2, 2, 1, 1, 0, 1, 3, 3, 0, 2, 1, 1, 3, 1, 3, 3, 1, 2,
       2, 3, 2, 2, 3, 3, 0, 2, 0, 0, 2, 3, 0, 1, 0, 1, 0, 3, 1, 0, 3, 3,
       0, 0, 3, 3, 2, 3, 3, 3, 1, 0, 2, 0, 1, 1, 2, 1, 0, 1, 0, 0, 2, 1,
       3, 0, 3, 0, 0, 2, 2, 1, 2, 0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 1, 2, 1,
       1, 1, 2, 0, 0, 0, 2, 3, 1, 3, 2, 2, 0, 2, 0, 0, 0, 2, 1, 0, 3, 3,
       1, 0, 0, 3, 3, 2, 2, 1, 0, 0, 3, 3, 0, 2, 0, 1, 3, 0, 3, 2, 0, 1,
       0, 3, 2, 0, 1, 1, 3, 2, 3, 0, 0, 1, 1, 2, 0, 0, 1, 2, 0, 3, 3, 2,
       1, 0, 3, 3, 1, 3, 3, 1, 0, 0, 3, 3, 1, 1, 0, 2, 1, 3, 0]),
'expression': array([0, 2, 3, 3, 3, 1, 2, 1, 1, 1, 2, 1, 0, 3, 2, 1, 2, 1, 0, 0, 1, 1,
       0, 0, 1, 1, 1, 2, 1, 3, 3, 0, 2, 0, 2, 2, 0, 3, 2, 0, 3, 3, 2, 1,
       0, 3, 1, 3, 0, 2, 1, 3, 3, 1, 1, 1, 0, 3, 3, 3, 0, 0, 2, 0, 2, 3,
       2, 0, 2, 1, 0, 1, 0, 2, 0, 3, 2, 0, 0, 0, 0, 1, 1, 1, 2, 3, 1, 2,
       0, 0, 1, 3, 1, 2, 0, 2, 3, 2, 3, 2, 0, 2, 3, 1, 2, 0, 0, 3, 2, 0,
       3, 1, 3, 3, 1, 2, 0, 3, 1, 1, 1, 1, 3, 0, 1, 0, 1, 1, 0, 0, 1, 3,
       2, 1, 1, 0, 0, 2, 3, 0, 3, 1, 1, 3, 2, 1, 0, 1, 1, 0, 2, 0, 0, 0,
       0, 3, 3, 3, 1, 2, 3, 2, 2, 3, 2, 2, 0, 0, 3, 2, 3, 2, 2, 0, 2, 0,
       2, 2, 2, 1, 1, 2, 1, 2, 2, 0, 2, 2, 3, 3, 2, 1, 2, 2, 1, 2, 0, 0,
       2, 3, 2, 1, 1, 0, 0, 1, 0, 0, 3, 2, 0, 1, 3, 2, 1, 1, 2, 2, 3, 2,
       2, 1, 0, 3, 2, 0, 3, 3, 0, 0, 3, 0, 0, 1, 0, 0, 2, 3, 0, 0, 0, 1,
       1, 1, 3, 3, 2, 3, 3, 3, 1, 1, 2, 2, 1, 3, 1, 1, 1, 2, 0, 2, 2, 2,
       2, 0, 0, 2, 3, 2, 1, 0, 2, 1, 3, 0, 3, 1, 1, 3, 2, 1, 2, 2, 3, 0,
       2, 3, 1, 1, 1, 3, 3, 0, 3, 3, 0, 1, 2, 1, 3, 1, 3, 2, 0]),
'wear_sunglasses': array([0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0])}
```

可以看到，name有20个类别，所以取值范围是0-19，direction有4个类别，取值范围是0-3，expression有4个类别，取值范围0-3，wear_sunglasses有2个类别，取值分别是0,1。

二、数据预处理及训练集测试集划分

由于是顺序读取，故需要random.shuffle打乱数据，然后通过定义的construct_data函数分别得到训练集和测试集，这里选择前305个为训练集，剩下的图片为测试集。

在一开始进行训练的时候，并没有意识到需要用random.shuffle打乱数据，造成训练集的种类全集不到20类，而测试集的种类全集不到20类，而且之间的重合项很少，所以在最后模型的实验结果非常差，最后才发现是在划分训练集合测试集的时候没有考虑的样本均衡的问题，经过random.shuffle打乱后的数据集经过划分后，训练集合测试集的种类全集都有20类了，而且分布相对比较均匀了。

三、任务一，人脸分类识别

第一个任务：使用机器学习进行人脸分类识别，给出识别准确率，其实name就是对应的标签

使用两种机器学习方法完成

分别是SVM和神经网络，SVM采用的kernel为rbf，根据网格搜索选取最优超参数，神经网络借助tensorflow搭建模型。

支持向量机(SVM) 在有限的数据量下性能非常好。给定训练数据，SVM将得到一个最优超平面，从而对训练数据进行分类。本实验中了，是调用了sklearn中的库，即from sklearn.svm import SVC

1）SVM模型

通过网格搜索选取最佳超参数：

```
SVC(C=1)
```

得到C=1

代入模型，进行训练

仅仅使用SVM效果就很好了，如果再加上PCA降维准确率反而降低了，最后就直接使用SVM，至于为什么会降低，会在解释中阐述。

实验结果：

	precision	recall	f1-score	support
an2i	1.00	1.00	1.00	17
at33	1.00	0.88	0.94	17
boland	0.94	1.00	0.97	15
bpm	0.88	1.00	0.94	15
ch4f	1.00	1.00	1.00	16
cheyer	1.00	1.00	1.00	20
choon	1.00	1.00	1.00	14
danieln	1.00	1.00	1.00	14
glickman	1.00	1.00	1.00	16
karyadi	1.00	1.00	1.00	16
kawamura	1.00	1.00	1.00	19
kk49	1.00	0.95	0.97	19
megak	1.00	1.00	1.00	16
mitchell	1.00	1.00	1.00	13
night	1.00	1.00	1.00	18
phoebe	1.00	1.00	1.00	14
saavik	1.00	1.00	1.00	12
steffi	1.00	1.00	1.00	14
sz24	1.00	1.00	1.00	19
tammo	1.00	1.00	1.00	15
accuracy			0.99	319
macro avg	0.99	0.99	0.99	319
weighted avg	0.99	0.99	0.99	319

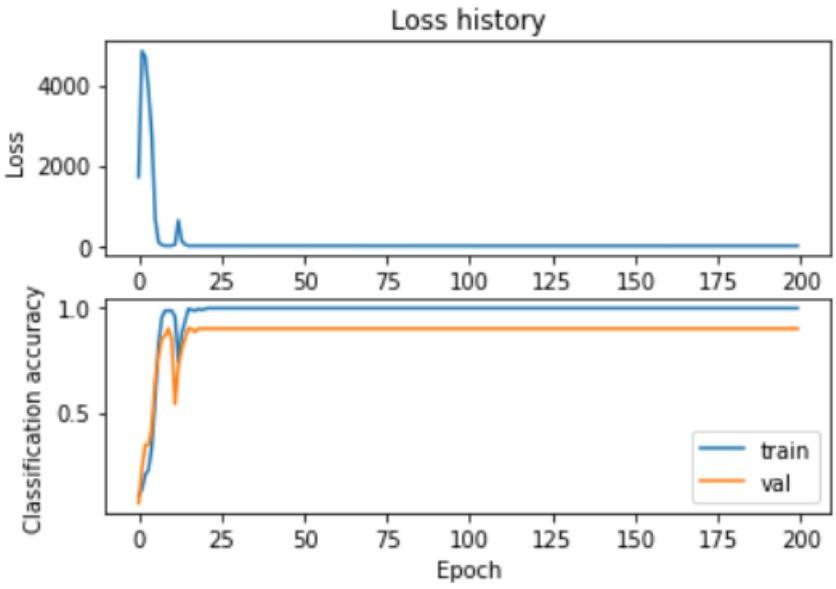
最终准确率达到99%，效果不错

2) 神经网络

这里使用models.Sequential()来搭建神经网络，使用model.compile()方法来配置训练方法，使用model.fit()方法来执行训练过程。

采用一层全连接层，20个神经元，softmax激活函数，L2正则化，使用随机梯度下降SGD，学习率为0.009，使用交叉熵损失函数。

训练过程:



训练好了以后，进行预测，由于人名有20类，最后网络输出的是独热向量编码，需要将它转变成对应的第几类，

最终得到的实验结果：

	precision	recall	f1-score	support
an2i	1.00	1.00	1.00	17
at33	0.94	1.00	0.97	17
boland	0.79	1.00	0.88	15
bpm	1.00	1.00	1.00	15
ch4f	1.00	1.00	1.00	16
cheyer	1.00	0.95	0.97	20
choon	0.93	1.00	0.97	14
danieln	0.93	1.00	0.97	14
glickman	1.00	1.00	1.00	16
karyadi	1.00	1.00	1.00	16
kawamura	1.00	1.00	1.00	19
kk49	1.00	0.84	0.91	19
megak	1.00	1.00	1.00	16
mitchell	0.93	1.00	0.96	13
night	1.00	1.00	1.00	18
phoebe	1.00	0.86	0.92	14
saavik	1.00	1.00	1.00	12
steffi	1.00	0.86	0.92	14
sz24	0.95	1.00	0.97	19
tammo	1.00	0.93	0.97	15
accuracy			0.97	319
macro avg	0.97	0.97	0.97	319
weighted avg	0.98	0.97	0.97	319

最后达到97%的准确率，效果不错

四、任务二，发现表情相似的图

expression就是对应的标签

使用三种方法，分别是SVM，神经网络和kmeans

1）SVM

通过网格搜索选取最佳超参数：

```
SVC(C=0.1)
```

得到C=0.1

代入模型，进行训练

实验结果：

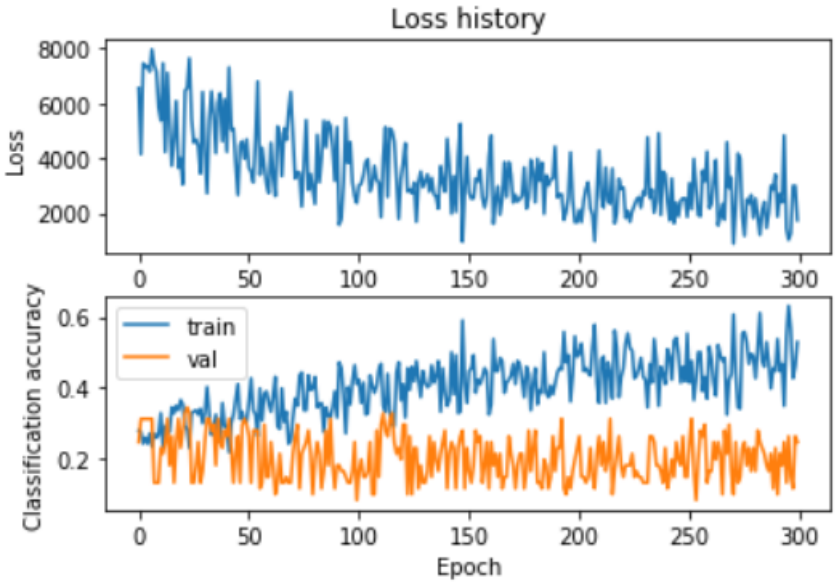
	precision	recall	f1-score	support
angry	0.00	0.00	0.00	83
happy	0.23	1.00	0.37	73
neutral	0.00	0.00	0.00	87
sad	0.00	0.00	0.00	76
accuracy			0.23	319
macro avg	0.06	0.25	0.09	319
weighted avg	0.05	0.23	0.09	319

最后准确率才23%，效果不佳，基本上就等于盲猜了。

2）神经网络

这里使用models.Sequential()来搭建神经网络，使用model.compile()方法来配置训练方法，使用model.fit()方法来执行训练过程，采用一层全连接层，4个神经元，softmax激活函数，l2正则化，使用随机梯度下降SGD，学习率为0.01，使用交叉熵损失函数。

训练过程:



可以看到loss history波动非常大，而且下降幅度也并不大，classification accuracy这一块，不管是在测试集上还是验证集上，准确率都是波动非常大，而且基本上就是在盲猜。

训练好了以后，进行预测，由于表情有4类，最后网络输出的是独热向量编码，需要将它转变成对应的第几类。

最终得到的实验结果：

	precision	recall	f1-score	support
angry	0.26	0.18	0.21	83
happy	0.16	0.37	0.22	73
neutral	0.18	0.18	0.18	87
sad	0.00	0.00	0.00	76
accuracy			0.18	319
macro avg	0.15	0.18	0.15	319
weighted avg	0.15	0.18	0.16	319

最后效果很差，准确率才18%。

3）kmeans

聚为4类，表情有4类。

实验结果：

	precision	recall	f1-score	support
angry	0.26	0.17	0.21	83
happy	0.21	0.42	0.28	73
neutral	0.24	0.15	0.18	87
sad	0.26	0.22	0.24	76
accuracy			0.24	319
macro avg	0.24	0.24	0.23	319
weighted avg	0.24	0.24	0.23	319

最后准确率才24%，效果不佳，基本上等于盲猜。

解释：

这三种方法，SVM，神经网络，kmeans聚类算法，不管是分类方法还是聚类方法效果都很差，和盲猜差不多，盲猜的概率是0.25，而有的甚至比盲猜表现还差。

猜测主要原因是在于数据集本身质量不是很好，该数据集年代较为久远，它不仅包含人脸，还包含了背景，会有很大的噪声。该数据集规模小，损失了很多人脸的特征，并且是两通道的灰度图。

这也解释了为什么经过PCA降维后再进行SVM，分类的准确率反而降低了，因为本身数据就没有包含很多特征，降维使得特征更少了。

五、进行朝向和有没有戴墨镜的区别

1）区别人脸朝向：

通过网格搜索选取最佳超参数：

```
SVC(C=10)
```

得到C=10

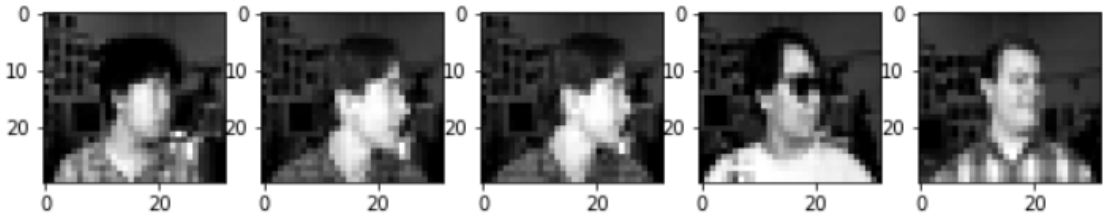
代入模型，进行训练

最后得到的实验结果：

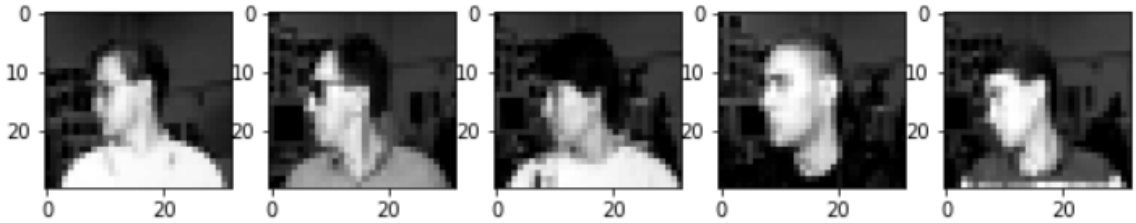
	precision	recall	f1-score	support
left	0.99	0.95	0.97	84
right	1.00	1.00	1.00	87
straight	0.93	0.97	0.95	71
up	0.96	0.96	0.96	77
accuracy			0.97	319
macro avg	0.97	0.97	0.97	319
weighted avg	0.97	0.97	0.97	319

最后准确率达到97%，效果不错

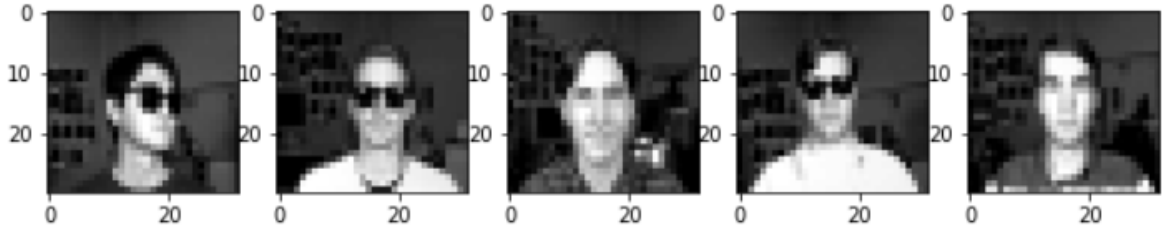
可视化查看朝left的前5个样本：



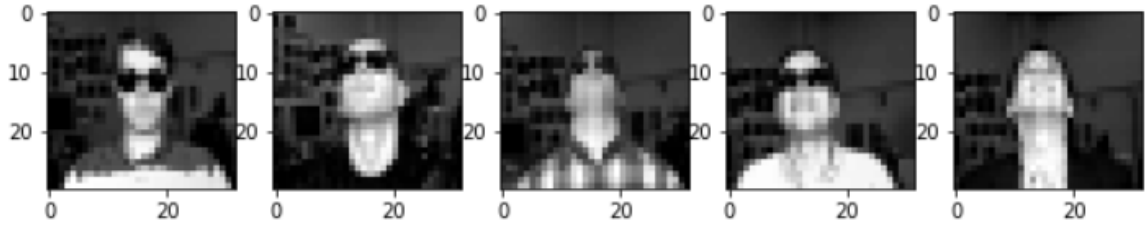
可视化查看朝right的前5个样本：



可视化查看朝straight的前5个样本：



可视化查看朝up的前5个样本：



可以看到，分类效果不错。

2）区分人脸是否戴sunglasses

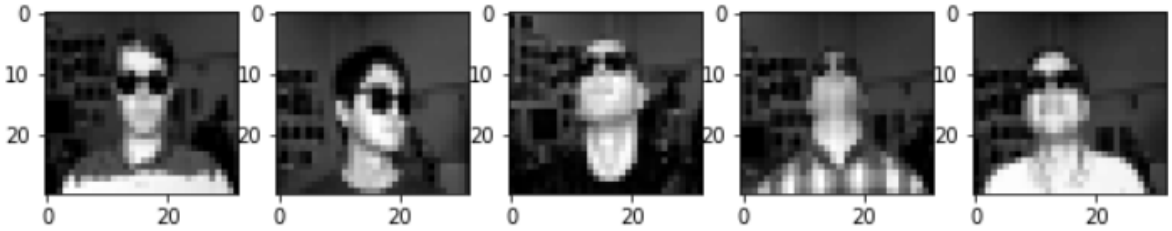
使用SVM

最后得到的实验结果：

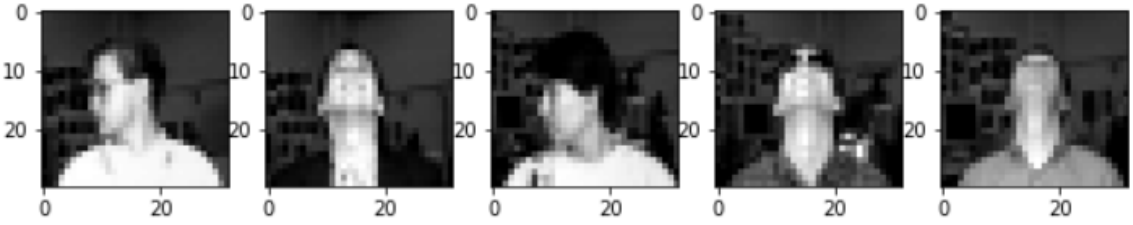
	precision	recall	f1-score	support
open	0.90	0.92	0.91	162
sunglasses	0.92	0.89	0.90	157
accuracy			0.91	319
macro avg	0.91	0.91	0.91	319
weighted avg	0.91	0.91	0.91	319

准确率达到91%，效果不错

可视化查看被分类到戴sunglasses的前5个样本：



可视化查看被分类到没戴sunglasses的前5个样本：



可以看到，分类效果不错。

六、总结

本次实验针对CMU Machine Learning Faces数据集，使用了两种机器学习方法完成任务一（人脸分类识别），准确率再97%以上，效果很不错，使用三种方法完成任务二（发现表情相似的图），但是最后的效果不太好，基本等同于盲猜，猜测这主要是同数据集本身的质量不高，特征少，噪声大，样本数量少有关。此外，还使用SVM区别了人脸朝向和人脸是否戴sunglasses，最后的分类准确率都在90%以上，效果不错。