# Lab\_3 基于监督学习的表情识别

## 一、实习目的与要求

1、结合实际应用理解监督学习分类过程；

2、理论结合实践，采用至少一种分类方法实现EmotionNet 自发表情数据预测；

3、通过进一步查阅文献，了解相关研究方向的最新研究进展。

## 二、实习题目

基于\*\*分类方法的人脸表情识别。

#### 【实验数据】

2479张人脸表情图像，要求能够预测人脸的七类表情：happy， angry， sad, disgust, surprise, fear, neutral

#### 【过程描述】

1. 数据预处理，清洗，提取特征（至少选一种颜色直方图、sift、hog、lbp等）
2. 要求采用至少一种分类器（决策树、贝叶斯、SVM、神经网络等），自动识别人脸自发表情
3. 要求采用5-折交叉验证
4. 要求绘出混淆矩阵、计算准确率、召回率和F1.

#### 【源代码】

数据预处理：去除xlsx中不存在的图片以及错误的图片：

对存在的图像：

1：针对LBP特征：先压缩成（200，200，3），在转化成灰度图像。从xlsx中提取图像对应的标签：

dataSet=[];dataLabel=[];  
info=xlrd.open\_workbook(**"URLsWithEmotionCat.xlsx"**);  
info\_sheet1=info.sheet\_by\_index(0);  
*#print(info\_sheet1.cell\_value(1,0));  
#print(info\_sheet1);*rows,columns=info\_sheet1.nrows,info\_sheet1.ncols;  
*#count=0;  
#print(rows,columns);***for** i **in** range(1,rows):  
 *#根据xlsx中第一列的数据获取图像名称,为字符串中最后一个/后面的字符串* url=info\_sheet1.cell\_value(i,0);  
 temp=re.split(**'/'**,url);  
 imageName=temp[len(temp)-1];  
  
 *#读取图像* **try**:  
 *#获取初始图像* init\_image=mpimg.imread(**"urlf/"**+imageName);  
 *#缩放减少计算量* image=transform.resize(init\_image,(200,200));  
 *#灰度化* gravity = np.array([0.299, 0.587, 0.114])  
 data=np.array(image);  
 data=np.dot(data,gravity.T)  
 *#将其转为一维数组  
 #data=data.flatten()* **for** j **in** range(1,columns):  
 **if**(int(info\_sheet1.cell\_value(i,j))!=-1):  
 dataLabel.append(j-1);  
 **break**;  
  
 dataSet.append(data);  
  
 *#判断图像是否缩放成功  
 #plt.figure('resize')  
 #plt.subplot(111)  
 #plt.title('before resize')  
 #plt.imshow(image, plt.cm.gray)  
 #picture.reshape(48,48);  
 #print("he")* **except**:  
 **continue**;  
 print(i,imageName);  
 print(**"This picture isn't exist"**);  
print(len(dataSet));  
  
np.save(**"docData.npy"**,dataSet);  
np.save(**"docLabel.npy"**,dataLabel)

之后:

dataSet=np.load(**"docData.npy"**);

dataLabel=np.load(**"docLabel.npy"**);  
**return** dataSet,dataLabel;

2:提取LBP特征：对所有的图像特征直接取LBP特征，中的local\_binary\_pattern直接分析，因为值在0到255之间的整数，故用直方统计图将其划分为256份，进行归类统计：

radius=1;  
points=8;  
**def** LBP\_doc(docMat):  
 hist\_doc=np.zeros((len(docMat),256));  
 *#np.save("LBPData.npy",hist\_doc);* **for** i **in** range(len(docMat)):  
 *#print(np.array(trainMat[i]).shape)* lbp=skft.local\_binary\_pattern(docMat[i],points,radius,**"default"**);  
 *#print(np.array(lbp).shape);* max\_bin=int(lbp.max()+1);  
 hist\_doc[i],\_=np.histogram(lbp,normed=**True**,bins=max\_bin,range=(0,max\_bin));  
 **return** hist\_doc;

3：提取SIFT特征：先提取出有效图像后，做灰度处理（不压缩）,直接通过opencv中的xfeatures2d.SIFT\_creat提取创建SIFT的对象，通过对象的detectAndCompute获取特征点，每个特征点的维度为128，然后使用PCA降维降到15维，减少运算量。最后提取图像对应标签：

**def** SIFT():  
 dataSet = [];  
 dataLabel = [];  
  
 info = xlrd.open\_workbook(**"URLsWithEmotionCat.xlsx"**);  
 info\_sheet1 = info.sheet\_by\_index(0);  
  
 rows, columns = info\_sheet1.nrows, info\_sheet1.ncols;  
 Array\_des=[];  
 **for** i **in** range(1, rows):  
 *# 根据xlsx中第一列的数据获取图像名称,为字符串中最后一个/后面的字符串* url = info\_sheet1.cell\_value(i, 0);  
 temp = re.split(**'/'**, url);  
 imageName = temp[len(temp) - 1];  
  
 *# 读取图像* **try**:  
 *# 获取初始图像* init\_image = cv2.imread(**"urlf/"** + imageName);  
 *# 灰度化* gray=cv2.cvtColor(init\_image,cv2.COLOR\_RGB2GRAY);  
 *#siftDetector=cv2.xfeatures2d.SIFT\_create(100);* siftDetector =cv2.xfeatures2d.SIFT\_create(23);  
 kp,des=siftDetector.detectAndCompute(gray, **None**);  
 *#des=des[:24,:];* new\_des=pca(des,15);  
 new\_des=new\_des[:23,:];  
  
 print(np.array(new\_des).shape)  
 new\_des=np.array(new\_des).flatten();  
**for** j **in** range(1, columns):  
 **if** (int(info\_sheet1.cell\_value(i, j)) != -1):  
 dataLabel.append(j - 1);  
 **break**;  
 Array\_des.append(new\_des)  
 *#dataSet.append(data);* **except**:  
 **continue**;  
 print(len(Array\_des));  
 *#Array\_des=np.load("SIFT.npy");* np.save(**"docLabel.npy"**,dataLabel);  
 dataLabel=np.load(**"docLabel.npy"**);  
 print(len(dataLabel));  
 np.save(**"SIFT.npy"**,Array\_des);  
 print(np.array(Array\_des).shape);  
 **return** Array\_des,dataLabel;

4：混淆矩阵及其回归率，准确率，精确率的计算：使用sklearn.metrics中的confusion\_matrix求得混淆矩阵，然后依据混淆矩阵计算出回归率，准确率，精确率。

*#根据testLabel和predict来制作混淆矩阵***def** my\_confusion\_matrix(testLabel,predict):  
 labels=list(set(testLabel)); *#获得类别数* con\_mat=confusion\_matrix(testLabel,predict);  
 print(type(con\_mat))  
 print(len(con\_mat),len(con\_mat[0]))  
 print(**"confusion\_matrix:"**);  
 print(**"label:\t"**,end=**''**);  
 **for** i **in** range(len(labels)):  
 print(str(labels[i])+**"\t"**,end=**''**);  
 print();  
 **for** i **in** range(len(labels)):  
 print(str(i)+**"\t\t"**,end=**''**);  
 **for** j **in** range(len(con\_mat)):  
 print(str(con\_mat[i][j])+**"\t"**,end=**''**);  
 print();  
 print();  
  
 *#准确率计算* true\_num=0; *#预测正确的个数* **for** i **in** range(len(con\_mat)):  
 true\_num+=con\_mat[i,i];  
 Arrucate=float(true\_num)/float(np.array(con\_mat).sum());  
 print(**"准确率:"**+str(Arrucate));  
 *#召回率计算，所有值的召回率求平均* Recall=0.0;  
 **for** i **in** range(len(con\_mat)):  
 *#print(con\_mat[:,i])* **if** sum(con\_mat[:,i])!=0:  
 Recall+=float(con\_mat[i][i])/float(sum(con\_mat[:,i]));  
  
 Recall=Recall/float(len(con\_mat[0]));  
 print(**"召回率:"**+str(Recall));  
  
 *#求精确率，所有精确率求平均* Precise=0;  
 **for** i **in** range(len(con\_mat)):  
 *#print(con\_mat[i,:])* Precise+=float(con\_mat[i][i])/float(sum(con\_mat[i,:]));  
 Precise=Precise/float(len(con\_mat[0]));  
 print(**"精确率:"**+str(Precise))  
  
 **return** Arrucate;

5:k折交叉运算：

使用sklearn.model\_selection中的Kfold划分数据集和标签。分成K份，每次都进行决策树、贝叶斯、SVM、神经网络的计算。求出各种方法对应的混淆矩阵，回归率，准确率，精确率。最后在求出每种方法对应的平均准确率；

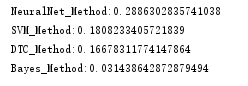
**def** K\_Cross\_Validation(k):  
 docLabel=np.load(**"docLabel.npy"**);  
 print(len(docLabel));  
 docLabel = np.array(docLabel);  
 kf=KFold(n\_splits=k);  
 Accurate\_NN=0.0;Accurate\_SVM=0.0;Accurate\_DTC=0.0;Accurate\_Bayes=0.0;  
 *#hist\_data=np.array(LBP\_doc(docData));  
 #np.save("LBPData.npy",hist\_data);* hist\_data=np.load(**"LBPData.npy"**);  
 *#hist\_data=np.load("SIFT.npy");* print(np.array(hist\_data).shape)  
 **for** trainIdex,testIndex **in** kf.split(hist\_data):  
  
 *#hist\_data,docLabel必须为np.array型* trainData,testData=hist\_data[trainIdex],hist\_data[testIndex];  
 trainLabel,testLabel=docLabel[trainIdex],docLabel[testIndex];  
  
 *#神经网络* print(**"NeuralNet\_Method:"**)  
 predict=NeuralNet\_Method(trainData,trainLabel,testData,testLabel);  
 Ac=my\_confusion\_matrix(testLabel,predict);  
 Accurate\_NN+=Ac;  
  
 *#SVM* print(**"SVM\_Method:"**);  
 predict=SVM\_Method(trainData,trainLabel,testData,testLabel);  
 Ac=my\_confusion\_matrix(testLabel,predict);  
 Accurate\_SVM+=Ac;  
 *#决策树* print(**"DTC\_Method:"**);  
 predict=DTC\_Method(trainData,trainLabel,testData,testLabel);  
 Ac=my\_confusion\_matrix(testLabel,predict);  
 Accurate\_DTC+=Ac;  
  
 *#贝叶斯方法* print(**"Bayes\_Method"**);  
 predict=Bayes\_Method(trainData,trainLabel,testData,testLabel);  
 Ac=my\_confusion\_matrix(testLabel,predict)  
 Accurate\_Bayes+=Ac;  
 *#n\_jobs是多线程,-1代表全部cpu资源都用上  
 # result = OneVsRestClassifier(dtc, -1).fit(trainData, trainLabel).score(testData, testLabel)  
  
 # dtc = DecisionTreeClassifier();  
 # dtc.fit(trainData,trainLabel);  
 # result=dtc.score(testData,testLabel);  
  
 #Accurate+=result;* print(**"NeuralNet\_Method:"**+str(float(Accurate\_NN/float(k))));  
 print(**"SVM\_Method:"** + str(float(Accurate\_SVM / float(k))));  
 print(**"DTC\_Method:"** + str(float(Accurate\_DTC / float(k))));  
 print(**"Bayes\_Method:"** + str(float(Accurate\_Bayes / float(k))));

关于神经网络，自己尝试基于Keras编写了一个3层网络，迭代400次，相比于sklearn中的库函数，计算出的准确率相差不大，但时间复杂度很大。下面附上自己编写的神经网络（其实也是调用别人的东西，嘻嘻）：

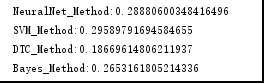
**from** keras.models **import** Sequential  
**from** keras.layers.core **import** Dense  
**from** keras.utils.np\_utils **import** to\_categorical  
**import** numpy **as** np  
  
*#构建神经网络,训练模型***def** NueralNet(trainData,trainLabel,testData,testLabel):  
 *#将三位矩阵转化为2维，每一组值由灰度图像变为一维向量* trainData=trainData.reshape(trainData.shape[0],-1);  
 testData=testData.reshape(testData.shape[0],-1);  
 *#需要转化为维矩阵* col=np.array(trainData).shape[1];  
 model=Sequential();  
 model.add(Dense(100,input\_dim=col,activation=**"relu"**));  
 model.add(Dense(50,activation=**"relu"**));  
 model.add(Dense(16,activation=**"softmax"**));  
  
 model.compile(optimizer=**"sgd"**,loss=**"categorical\_crossentropy"**,metrics=[**'accuracy'**]);  
 categorical\_train\_labels = to\_categorical(trainLabel, num\_classes=**None**);  
 categorical\_test\_labels=to\_categorical(testLabel, num\_classes=**None**);  
 model.fit(trainData,categorical\_train\_labels,batch\_size=20,epochs=400);  
  
 predict=model.predict(testData);  
 **return** predict;

结果：

LBP特征：

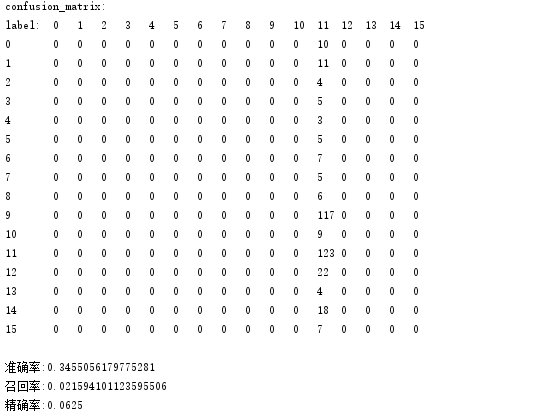


SIFT特征：



可以看到：SVM效果最好（单次测试有时可以达到0.34）神经网络次之（单次测试有时可以达到0.33）其次巨册书，贝叶斯效果最差。LBP五折一般为

某一次的（神经网络方法：LBP特征提取）



（附上源代码）

#### 【改进设想】

1：关于Hog:此特征提取多用于与SVM结合的行人监测，sklearn库有封装，但自己没看懂该方法，就没敢乱尝试。后面可以使用该方法测试。

2：SIFT与LBP的结合：要转化成两个一维向量，之后与对应方法做分类处理，但结合的方法自己一直不理解。后面可以继续尝试。

（或查阅文献提出改进设想；或考虑与第三方机器学习库分类结果的比较；或与采用其他分类算法实现的结果比较等）。

## 三、本次实习小结

通过本次实习，能够有机会小小尝试一下神经网络的编写，原来也不是太过复杂\*当然我是基于Keras,TensorFlow自己还无法理解)。然后时图像中的处理方法，hog,lbp,sift，每种方法都有自己的特点。最后发现sklearn十分强大，从降维到各种分类如果再结合skimage，几乎完成了所有的工作，并且效率很高，时间复杂度也很低。只有你想不到，没有做不到的。富有趣味。但是使用别人的东西，也让自己对于这些模型没有真正的理解。自己无法真正理解轮子的原理，而冒然使用轮子。可能终会翻车。这是自己有待改进的。

=======================================================

我的联系方式：

电话：13349830890

邮箱：liuyy@cug.edu.cn