# 表情识别系统

# 项目介绍

## 应用背景

表情识别在机器人的应用上具有重要意义，机器人通过识别人的面部表情，能够提升它与人类的沟通能力，为机器的下一步行动作出决策。对于孤寡老人和幼童等社交能力较弱的群体而言，机器人通过识别他们的表情，可以更加细致地与他们进行交流。在刑侦领域，机器通过识别犯人的实时表情，对于案件的侦破能够起到一定的帮助。

## 功能介绍

该项目是一个简单的面部表情识别系统，与人脸识别系统相结合，在标注处人物身份的同时，能标注出其表情。

## 环境依赖

opencv\_python、numpy、matplotlib、pandas、PIL、scipy、sklearn、xgboost、skimage

## 前置条件

Python语言

numpy、matplotlib、scipy等科学计算类库

机器学习

计算机视觉

# 系统概述

该项目构建了一个gabor滤波器，用滤波器对图像进行特征提取，使用一个或多个分类器对训练数据进行情感分类，并保存训练好的模型。当有一张图片等待识别时，会首先经过训练好的分类器得到类别，然后输出这个类别。

# 功能实现

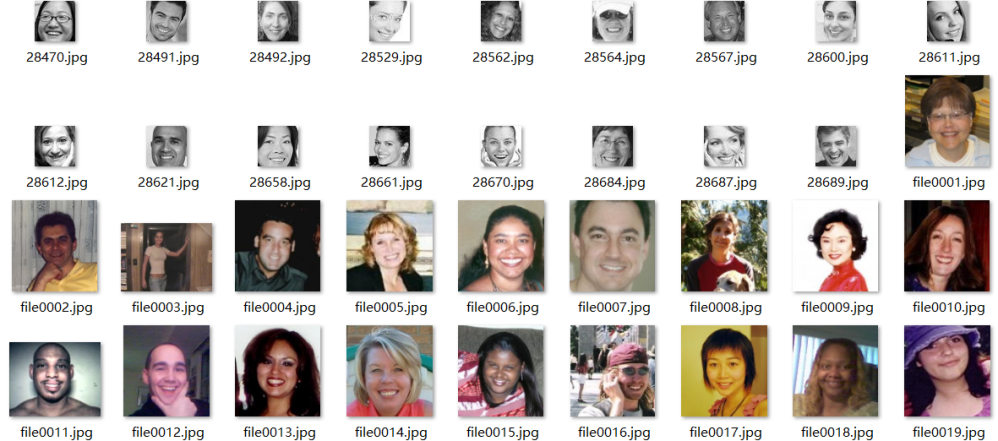
## 特征提取

特征提取是图像识别的重要步骤，原始图像中包含大量不必要的信息，包括很多噪音。特征提取的目的是将对于分类决策有帮助的信息抽取出来，减少受到的噪音的干扰。

### 数据集准备

数据集是机器学习模型必不可少的部分，在表情识别中，需要标注好的表情图片作为训练数据。使用的数据集包括以下几个：fer2013、jaffe、originalPics、cohn-kanade，以及一些来自互联网的图片。

训练数据展示如下：

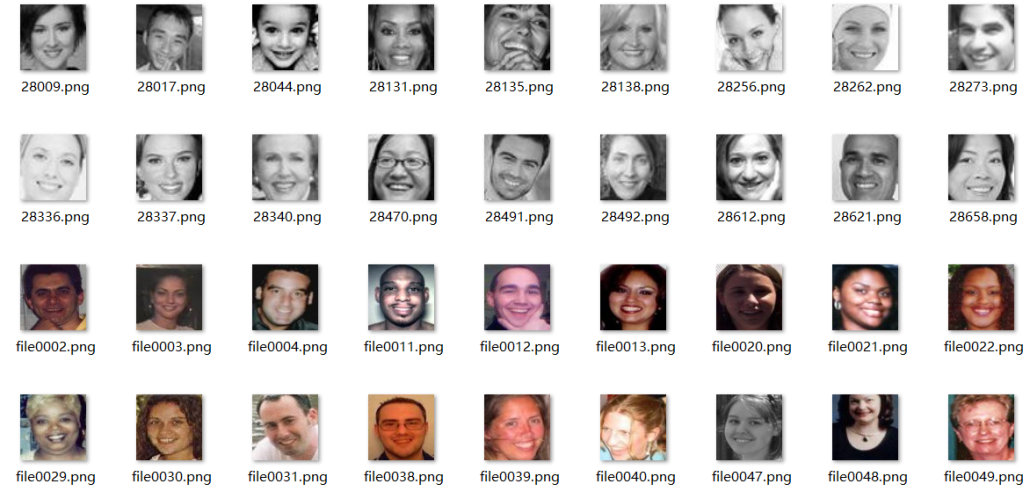


### 图片预处理

下载好的图片并不能直接进行处理，因为它们的格式不尽相同，尺寸也大小各异，有的图片中包含较多的背景，这都会对分类结果产生影响。因此，必须首先对图片进行预处理。

进入分类器的图片必须保证特征数量一致，即每张图片的像素要相同。我们使用mtcnn对图片进行人脸检测与裁剪，然后统一它们的尺寸。使用mtcnn的步骤详情参见《人脸识别系统》的人脸检测部分。

经过预处理之后的图片如下所示：



预处理的第二步是将图像由三通道的彩色图转化为单通道的灰度图，这样能够有效地压缩特征。

代码如下：

im\_gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

预处理的第三步是进行灰度直方图的均衡化，目的是使各个灰度级的分布更加均匀，避免特征提取时受到极端值的影响。

代码如下：

im\_gray = cv2.equalizeHist(im\_gray)

### 构建gabor滤波器

Gabor变换所采用的核（Kernels）与哺乳动物视觉皮层简单细胞2D感受野剖面非常相似，具有优良的空间局部性和方向选择性，能够抓住图像局部区域内多个方向的空间频率和局部性结构特征。在该步骤中，我们通过构建一个3x3的滤波核来提取图像在不同尺度和方向上的纹理特征。

opencv提供了构建滤波器的函数，代码如下：

ksize = [1,2,3]

lamda = np.pi / 2.0

filters = []

for theta in np.arange(0, np.pi\*15/8, np.pi/8):

for K in range(3):

kern = cv2.getGaborKernel((ksize[K], ksize[K]), 1.0, theta, lamda, 0.5, 0, ktype=cv2.CV\_32F)

kern /= 1.5 \* kern.sum()

filters.append(kern)

return filters

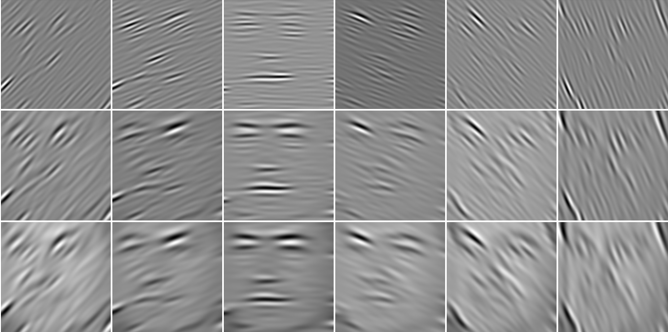
### gabor变换

一个二维的gabor滤波器是正弦平面波和高斯核函数的乘积，由于该项目在3个尺度和6个不同方向上做gabor变换，因此共能得到18个不同的特征图。

如果对下面这张人脸做gabor变换：



得到的特征图如下所示：



代码如下：

def process(self,img,filters):

accum = np.zeros\_like(img)

for kern in filters:

fimg = cv2.filter2D(img, cv2.CV\_8UC3, kern)

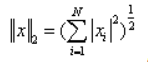
np.maximum(accum, fimg, accum)

return accum

### 计算L2范数

经过gabor滤波，总共得到了18个特征图，每个特征图的特征数量都与原图像像素个数相同。为了再次简化特征，我们将这18个特征图叠加到一起，然后在每个像素点上对它的第三个维度取L2范数。

L2范数的计算方式为：



它与欧几里得距离的计算方式一样。通过计算L2范数，将18个特征图压缩为了一个特征图。该特征图中的特征数量与原始图像的像素个数相同。

代码如下：

(x,y)=img\_ndarray.shape

for i in range(len(filters)):

res1 = self.process(img\_ndarray , filters[i])

res.append(res1)

for i in range(x):

for j in range(y):

res = np.asarray(res)

res2=res[:,i,j]

res3 = np.linalg.norm(res2, ord=2)

norm.append(res3)

res = np.asarray(norm)

return res

## 图片分类

在前一个步骤中，我们对于每一张图片都进行特征提取操作，得到了一个特征图。这个特征图可以看作一个矩阵，该矩阵对应着图片的类别。在该步骤中，使用两种方式来对图片进行多分类。

第一种是采用一对多的方式，训练多个模型，每个模型都进行二分类：其中一种类别和剩下的所有类别，当有新的数据等待识别时，让它通过所有的模型，根据每个模型输出概率的投票来进行决策。

第二种是训练一个模型，用softmax的方式进行多分类。

### 读取数据和标签

每个类别的数据放在各自的文件夹中，用文件夹的名字作为图片的类别。这样在读取数据的时候可以方便地获得它的标签。读取数据时，将数据的特征放在一个矩阵中，而标签存放在另一个矩阵中，这两个矩阵将会作为参数传入到模型当中。

其中标签[0]代表“高兴”，[1]代表“不高兴”，[2]代表“惊讶”，[3]代表“愤怒”。

数据和标签的变量如下：

|  |  |  |
| --- | --- | --- |
| 变量名 | 说明 | shape |
| X | 训练集特征矩阵 | 训练集图片总数x每张图片的特征数量 |
| y | 训练集数据对应的标签矩阵 | 训练集图片总数x1 |

相应的代码如下：

label=[]

total=[]

if not os.path.isdir(imagePath):

raise IOError("The folder " + imagePath + " doesn't exist")

for root, dirs, files in os.walk(imagePath):

for filename in (x for x in files if x.endswith(('.tiff','.jpg','.png'))):

filepath = os.path.join(root, filename)

object\_class = filepath.split(os.sep)[-2]

if object\_class == '0':

label.append(0)

elif object\_class == '1':

label.append(1)

elif object\_class == '2':

label.append(2)

elif object\_class == '3':

label.append(3)

elif object\_class == '012':

label.append(120)

elif object\_class == '013':

label.append(130)

elif object\_class == '023':

label.append(230)

else:

label.append(123)

image = np.array(Image.open(filepath))

newData = self.getGabor(image,self.filters)

total.append(newData)

total = np.array(total)

print(total.shape)

if random\_split == False:

X = total[:,:]

y = np.squeeze(label)

y = y.reshape((-1, 1))

return X,y

### 切分训练集和测试集

切分训练集和测试集也有两种方式：手动切分和随机切分。在手动切分的条件下，需要把训练数据分为两部分：训练集和测试集，每个部分下面都有四种类别的表情。在随机切分的条件下，只需要将所有的类别全部放到同一个目录下，程序会根据预设的比例进行随机切分。

代码如下：

if random\_split == False:

X = total[:,:]

y = np.squeeze(label)

y = y.reshape((-1, 1))

return X,y

else:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(total, label, test\_size=0.1, random\_state=0)

return X\_train,X\_test,y\_train,y\_test

### 初始化分类器

sklearn提供了多个分类器，如决策树、支持向量机、随机森林等，此外我们还使用了对梯度提升树做了优化的xgboost，他们各有如下特点：

|  |  |  |
| --- | --- | --- |
| 分类器 | 特点 | 不足 |
| 决策树 | 采用贪心算法，理解简单，在数据特征较少或数据规模大时均有较好效果。 | 预测结果不稳定，容易过拟合。 |
| SVM | 采用非线性映射，使用支持向量进行决策，具有较好的鲁棒性。 | 对于多分类问题难以实施，不适合大规模训练样本。 |
| 随机森林 | 多棵树集体决策，改善过拟合。可处理高维数据，且不需要进行特征筛选。 | 回归问题中的表现不好，不适合处理特征较少的数据。 |
| xgboost | 在梯度提升树基础上引入剪枝和正则化，改善过拟合。对loss函数做二阶泰勒展开，提升梯度下降的准确性和速度。 | 迭代前需要在树的节点对特征做预排序，算法复杂度高，内存开销较大。 |

在这个方法中，利用sklearn和xgb提供的接口初始化了一个模型，使用fit函数进行模型训练，将训练好的模型返回。

初始化分类器的方法如下：

def load\_model(self,X,y,cla):

if cla == 'xgb':

clf = xgb.XGBClassifier(silent=False,

learning\_rate = 0.07,

n\_estimators=160,

colsample\_bytree = 0.7,

max\_depth = 5,

min\_child\_weight = 5,

gamma = 0,

reg\_lambda = 0.0125,

subsample = 0.8,

objective = "multi:softmax")

elif cla == 'linear':

clf = svm.SVC(kernel='linear', C = 0.1)

elif cla == 'rbf':

clf = svm.SVC(kernel = 'rbf', C = 0.1, gamma = 0.0001)

elif cla == 'poly':

clf = svm.SVC(kernel = 'poly',degree = 6,gamma = 0.0001,C = 0.5)

elif cla == 'dtree':

clf = DecisionTreeClassifier(criterion = 'gini',splitter = 'random' ,max\_depth = 20,max\_leaf\_nodes = 50)

elif cla == 'randomforest':

clf = RandomForestClassifier(n\_estimators = 140 , criterion = 'entropy' ,n\_jobs = -1)

model = clf.fit(X, y)

return model

### 训练多个模型

从硬盘中读取图片数据并提取特征以后，就可以将特征传入模型进行训练了。在二分类情况下，模型分类的效果通常会比较好。但是当分类的数量增多时，由于类别之间特征区分不明确等原因，分类的效果有所下降。此时，可以尝试训练多个二分类模型，每个模型都用来区分其中一类和剩下所有类。

训练完毕以后，使用joblib将模型保存起来。

相关的代码如下：

def train\_clf\_model(self):

paths = os.listdir(self.train\_path\_fix)

start\_time = time.time()

for i,path in enumerate(paths):

path = os.path.join(self.train\_path\_fix,path)

i += 1

print("正在加载第%d个模型的数据..." % i)

train\_x,train\_y = self.load\_training\_data(path,random\_split = False)

print("正在训练第%d个XGB模型..." % i )

model = self.load\_model(train\_x,train\_y, cla = 'xgb')

clf\_model\_path = os.path.join(self.model\_path,("clf\_model\_" + "%04d"%int(i) + ".pkl"))

print("正在保存第%d个模型..." % i )

joblib.dump(model,clf\_model\_path)

cost\_time = (time.time() - start\_time)

print("模型训练完成！用时%4.5f秒。" % cost\_time)

### 3.2.5.测试模型

使用joblib的load方法从磁盘中载入所有保存好的模型，然后从读取测试集中的数据，对这些数据进行特征提取，并传入到训练好的模型当中。这时，每一个模型都会输出一个n行两列的数组，这个数组包含测试集中n张图片的两个概率：待识别图片是该分类的概率和待识别图片是剩下所有分类的概率。

|  |  |  |
| --- | --- | --- |
| 变量 | shape | 数组每一行含义 |
| y\_pred | 测试集图片总数x2 | [P(class = [0]),P(class = [1,2,3])] |
| y\_pred2 | 测试集图片总数x2 | [P(class = [1]),P(class = [0,2,3])] |
| y\_pred3 | 测试集图片总数x2 | [P(class = [2]),P(class = [0,1,3])] |
| y\_pred4 | 测试集图片总数x2 | [P(class = [3]),P(class = [0,1,2])] |

之后会比较P(class = [0])、P(class = [1])、P(class = [2])、P(class = [3])这四个数值，选择最高的一个作为输出的类别。

然后，用sklearn提供的classification\_report和confusion\_matrix方法输出预测结果的准确率、召回率和混淆矩阵。

代码如下：

def test\_model(self):

print("正在载入模型...")

model = joblib.load(r".\model\clf\_model\_0001.pkl")

model2 = joblib.load(r".\model\clf\_model\_0002.pkl")

model3 = joblib.load(r".\model\clf\_model\_0003.pkl")

model4 = joblib.load(r".\model\clf\_model\_0004.pkl")

print("\n" + "正在载入测试数据...")

X, y = self.load\_training\_data(self.test\_path)

y\_pred = model.predict\_proba(X)

y\_pred2 = model2.predict\_proba(X)

y\_pred3 = model3.predict\_proba(X)

y\_pred4 = model4.predict\_proba(X)

feature = []

for i in range(y\_pred.shape[0]):

photo = np.array([y\_pred[i,0],y\_pred2[i,0],y\_pred3[i,0],y\_pred4[i,0]])

feature.append(photo)

index = np.argmax(np.array(feature),axis = 1)

print("召回率和准确度：")

print(classification\_report(y.ravel(), index, target\_names=['0','1','2','3']))

print("混淆矩阵：")

print(confusion\_matrix(y.ravel(), index, labels=range(4)))

print(str(index) + "\n" + str(y.ravel()))

### 训练单个模型并测试

第二种方法只训练一个多分类模型，直接分成四个类别。训练集目录下包含了四个目录，分别是[0],[1],[2],[3]四种表情的数据。采用随机切分训练集-测试集的模式读取数据以后，进行特征提取的操作，然后把提取过的特征传入到模型当中训练。

训练完毕以后，保存训练好的模型，并使用该模型进行测试。

代码如下：

def train\_test\_clf\_model\_2(self,\*\*kwargs):

for key,value in kwargs.items():

if value not in ['xgb','linear','rbf','poly','dtree','randomforest']:

raise ValueError("please choose a portable classifier in 'linear','rbf','poly','dtree','randomforest'!")

if len(kwargs) == 0:

cla = 'xgb'

else:

cla = kwargs['classifier']

path = self.train\_path

start\_time = time.time()

print("正在载入训练数据...")

train\_x,test\_x,train\_y,test\_y = self.load\_training\_data(path,random\_split = True)

print("数据载入完毕，用时%4.5f秒。"% (time.time()-start\_time))

start\_time = time.time()

print("正在训练模型...")

model = self.load\_model(train\_x,train\_y,cla)

clf\_model\_path = os.path.join(self.model\_path,("clf\_model.pkl"))

print("模型训练完毕！用时%4.5f秒，正在保存模型..." % (time.time()-start\_time))

joblib.dump(model,clf\_model\_path)

print("正在评估模型...")

pred\_y = model.predict(test\_x)

print("召回率和预测准确度:"+"\n")

print(classification\_report(test\_y, pred\_y, target\_names=['0','1','2','3']))

print("混淆矩阵：")

print(confusion\_matrix(test\_y, pred\_y, labels=range(4)))

print(str(pred\_y) + "\n" + str(test\_y))

acc\_train = model.score(train\_x,train\_y)

acc\_test = model.score(test\_x,test\_y)

print(model.\_\_class\_\_.\_\_name\_\_, "训练集预测精度:", acc\_train)

print(model.\_\_class\_\_.\_\_name\_\_, "测试集预测精度:", acc\_test)

## 表情标注

训练好的模型保存完毕后，可以载入模型对图片进行预测，并与人脸识别系统进行对接，可以在一副画面中同时识别出人脸与表情。

### 初始化表情类

表情识别的代码定义在一个class里面，使用它需要先初始化这个类，然后调用其中的方法。这个类包含了以下几个方法，可以根据需要调用。

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| build\_filters | 构建滤波核 |
| process | 滤波过程 |
| getGabor | 图像预处理和提取特征的L2范数 |
| load\_training\_data | 载入训练数据 |
| Load\_model | 初始化模型 |
| feature\_visualization | 特征可视化 |
| train\_clf\_model | 训练一对多模型 |
| test\_model | 测试一对多模型 |
| train\_test\_clf\_model\_2 | 训练并测试单个模型 |
| image\_recognise | 与人脸识别对接 |

这个类需要如下几个参数：

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| train\_path\_fix | 一对多方式训练模型的训练集存放位置 |
| train\_path | 训练单个模型的训练集存放位置 |
| test\_path | 测试集存放位置 |
| model\_path | 模型保存位置 |
| image\_size | 训练数据使用的图片尺寸 |

实例代码：

emo = Emotion(train\_path\_fix = r'source\fix',train\_path = r'source\train',test\_path = r'source\val',model\_path = r'source\model',image\_size = 75)

### 3.3.2调用方法进行训练

train\_clf\_model和train\_test\_clf\_model\_2都是用来训练模型的方法，调用这两个方法的其中一个即可开始训练。

实例代码：

emo.train\_test\_clf\_model\_2()

### 3.3.3.待识别图片预处理

待识别图片需要经过处理提取特征才能作为参数传入到训练好的模型中。

step1：进行人脸检测与裁剪

待识别图片从文件读取以后，需要使用mtcnn对其进行人脸检测与裁剪的工作，mtcnn会将人脸区域框选并裁剪下来。如果一张图片中包含多个人脸，那么裁剪出来的将会是多个人脸区域的图像矩阵。

示例代码：

name\_list,bounding\_boxes = rec.get\_name\_and\_box(cv\_img)

images,\_ = rec.face\_det.find\_faces(cv\_img)

Step2：对人脸区域的图像矩阵做特征提取

裁剪出来的人脸区域需要和训练集做同样的特征提取操作，并将尺寸修改为和训练数据相同，才能传入模型。

示例代码：

total = []

for i,image in enumerate(images):

resized = misc.imresize(image, (self.image\_size, self.image\_size), interp='bilinear')

newData = self.getGabor(resized,self.filters)

total.append(newData)

total = np.array(total)

X = total[:,:]

### 3.3.4.标注表情

将待识别图片的特征输入模型以后，会得到模型对它的分类，用一个字典把该分类和类别所代表的表情相对应，就可以得到一张图片中人脸表情的名称。如果一张图片中有多张人脸，得到的将会是包含多张表情的列表。

实例代码：

model = joblib.load(r".\source\model\clf\_model.pkl")

predict = model.predict(X)

emotion\_dict = {0:'happy',1:'unhappy',2:'surprise',3:'angry'}

emotion\_list = [emotion\_dict[p] for p in predict]

return emotion\_list

# 模型调优

Gabor滤波和模型初始化都需要设置参数，开始我们并不知道什么样的参数是合适的，因此需要通过实验来找到最佳参数。为了减少载入数据的时间，以下实验仅使用[0]和[1]两种类别。

## 调节滤波参数

实验条件：

|  |  |
| --- | --- |
| 训练集样本数 | [0]类别1000张，[1]类别1000张 |
| 训练集样本比例 | 1:1 |
| 分类器 | svm |

实验对象：

滤波核尺度、高斯函数标准差、滤波方向

实验过程：

根据文献指导，尝试不同的滤波尺度、波长和方向的组合，查看测试集准确率。

|  |  |  |  |
| --- | --- | --- | --- |
| 尺度 | 标准差 | 方向 | 准确率 |
| [1,2,3] | 1 | π/4-7π/4以π/4为间隔 | 76.3% |
| [1,2,3] | 1 | π/8-15π/8以π/8为间隔 | 77.8% |
| [0,1,2] | 1 | π/8-15π/8以π/8为间隔 | 75.9% |
| [1,2,3,4,5] | 1 | π/8-15π/8以π/8为间隔 | 76.7% |
| [1,2,3,4,5] | 1 | π/5-6π/5以π/5为间隔 | 78.7% |
| [1,2,3,4,5] | 2π | π/5-6π/5以π/5为间隔 | 78.2% |
| [1,2,3] | 2π | π/4-7π/4以π/4为间隔 | 77.6% |
| [1,2,3] | 2π | π/8-15π/8以π/8为间隔 | 79.4% |

结论：通过尝试不同参数的组合，发现当滤波尺度为[1,2,3]，高斯函数标准差为2π，滤波方向为π/8到15π/8时，效果最好。

## 调节样本数量

实验条件：

|  |  |
| --- | --- |
| 滤波核参数 | 前项实验调节的最佳参数 |
| 分类器 | svm、随机森林 |

实验对象：

样本数量、样本比例

实验过程：

Step1：调节样本数量，观察svm和随机森林在测试集上的准确率。

|  |  |  |
| --- | --- | --- |
| 样本数量 | svm（poly核函数） | 随机森林 |
| 类别[0]500，类别[1]500 | 69.6% | 70.5% |
| 类别[0]550，类别[1]550 | 72.4% | 73.2% |
| 类别[0]600，类别[1]600 | 71.8% | 72.9% |
| 类别[0]650，类别[1]650 | 74.8% | 74.6% |
| 类别[0]700，类别[1]700 | 74.1% | 75.3% |
| 类别[0]750，类别[1]750 | 75.3% | 76.6% |
| 类别[0]800，类别[1]800 | 76.3% | 77.2% |
| 类别[0]850，类别[1]850 | 76.3% | 77.7% |
| 类别[0]900，类别[1]900 | 77.5% | 78.6% |
| 类别[0]950，类别[1]950 | 77.9% | 79.3% |
| 类别[0]1000，类别[1]1000 | 78.9% | 80.2% |

结论：随着训练集数量的增加，测试集准确率呈上升趋势。

Step2：调节正负样本比例，观察测试集准确率和召回率，使用svm分类器。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 正负样本比 | 召回率（正） | 准确率（正） | 召回率（负） | 准确率（负） |
| 1:1 | 80% | 77% | 77% | 79% |
| 2:1 | 84% | 71% | 73% | 84% |
| 3:1 | 87% | 65% | 67% | 88% |
| 4:1 | 93% | 59% | 57% | 91% |
| 5:1 | 97% | 51% | 52% | 95% |

结论：当正负样本比例为1:1时，预测结果处于最平衡的状态。如果加大某样本的比重，那么预测结果的召回率会向那一方倾斜，而准确率随之降低。

## 选择分类模型

找到最佳的滤波参数和训练数据比例以后，需要确定哪一种分类器更加适合该任务。该实验将会比较sklearn的各种分类器和xgboost的效果。

实验条件：

|  |  |
| --- | --- |
| 滤波核参数 | 前项实验调节的最佳参数 |
| 正负样本比 | 1:1 |
| 样本数量 | [0]类别1000张，[1]类别1000张 |

实验过程：

Step1：因为rbf核的svm在图像领域具有一定优势，我们先选择该核函数作为分类模型，并调节它的参数。

实验对象：

rbf核的svm的正则项系数C和gamma

|  |  |  |
| --- | --- | --- |
| C | gamma | 测试集准确率 |
| 1 | 0.1 | 61.6% |
| 0.5 | 0.1 | 62.3% |
| 0.2 | 0.1 | 62.3% |
| 0.1 | 0.1 | 62.5% |
| 0.1 | 0.01 | 64.6% |
| 0.1 | 0.001 | 64.9% |
| 0.1 | 0.0005 | 65.7% |
| 0.1 | 0.0001 | 66.7% |

结论：当选择正则项系数为0.1，gamma为0.0001时，rbf核函数的svm在该项任务中的表现最好。

Step2：先选择svm作为默认分类器，比较该分类器的三种常用核函数：linear核、rbf核和poly核。

实验对象：

svm的核函数类型

|  |  |  |
| --- | --- | --- |
| 核函数类型 | 训练集准确率 | 测试集准确率 |
| linear | 100% | 70.7% |
| rbf | 100% | 66.8% |
| poly | 100% | 79.2% |

结论：poly核函数在该任务中的表现更好。

Step3：比较决策树、svm（poly核）、随机森林和xgboost的效果。

实验对象：

四种不同分类器

|  |  |  |
| --- | --- | --- |
| 分类器 | 训练集准确率 | 测试集准确率 |
| 决策树 | 100% | 74.5% |
| svm | 100% | 79.2% |
| 随机森林 | 100% | 81.6% |
| xgboost | 100% | 83.3% |

结论：xgboost和随机森林的表现要领先于决策树和svm

Step4：调节随机森林的参数

实验对象：

随机森林的决策树个数

|  |  |  |
| --- | --- | --- |
| 决策树个数 | 训练集准确率 | 测试集准确率 |
| 100 | 100% | 81.3% |
| 120 | 100% | 82.2% |
| 140 | 100% | 82.9% |
| 160 | 100% | 82.4% |
| 180 | 100% | 81.8% |
| 200 | 100% | 81.1% |

结论：当随机森林的决策树个数在140的时候，模型的表现最好，并且与xgboost相接近。

## 解决过拟合

我们发现，无论使用什么模型，训练集的准确率都为100%，而测试集的表现与训练集相差甚远。这是因为模型训练时迭代到默认迭代到最大次数，对训练集中的局部特殊数据也进行了拟合。可以通过调节参数来解决这个问题。

Step1：调节随机森林参数

随机森林中与过拟合相关的参数较少，能调节的只有n\_estimators，即决策树的数量。但是通过调节这个参数并不能改善模型过拟合的现状，于是我们考虑通过调节xgboost的参数来解决。

Step2：调节xgboost参数

xgboost的参数比较多，与过拟合相关的参数有以下几个：

|  |  |
| --- | --- |
| 参数 | 说明 |
| **min\_child\_weight** | 最小叶子节点样本权重和 |
| max\_depth | 树的最大深度 |
| gamma | 节点分裂所需的最小损失函数下降值 |
| subsample | 每棵树，随机采样的比例 |
| lambda | 权重的L2正则项 |

经过多次测试，发现gamma和subssample仅会略微影响测试集的准确率，对训练集准确率没有影响。max\_depth和min\_child\_weight两个参数对于结果的影响相反。增大max\_depth会使模型更容易学习到局部样本，而增大min\_child\_weigh则会使得模型更加保守。

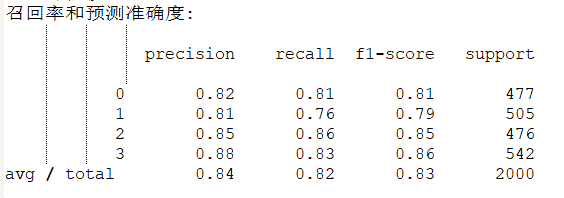
调整结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| max\_depth | min\_child\_weight | 训练集准确率 | 测试集准确率 |
| 6 | 1 | 100% | 80.8% |
| 5 | 5 | 100% | 83.4% |
| 4 | 6 | 100% | 83.6% |
| 10 | 20 | 96.9% | 82.2% |
| 12 | 50 | 89.8% | 81.2% |
| 20 | 100 | 85.4% | 77.6% |

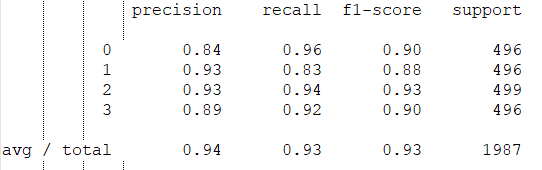
结论：当两个参数都比较小的时候，调节它们没有明显的效果。因为训练数据的特征维度比较大，所以将这两个值都相应地放大，在12和50的时候，能使得训练集和测试集的差距较小且测试集准确率不会太低。

## 验证结果

将每个类别的样本数量扩大至5000个，训练模型以后保存。使用一个模型测试的结果如下：



使用多个模型测试的结果如下：



# 5.参考资料

有关支持向量机（svm）的原理可以参见https://www.cnblogs.com/spoorer/p/6551220.html

有关决策树的原理可以参见

<http://www.aboutyun.com/thread-22370-1-1.html>

有关随机森林的原理可以参见

<https://www.jiqizhixin.com/articles/2017-07-31-3>

有关梯度提升树的原理可以参见

https://blog.csdn.net/program\_developer/article/details/79404581

xgboost的原理可以参见以下这篇论文：

https://arxiv.org/pdf/1603.02754v1.pdf