【华子】机器学习011-分类模型的评估：准确率，精确率，召回率，F1值 -

(本文所使用的Python库和版本号: Python 3.5, Numpy 1.14, scikit-learn 0.19, matplotlib 2.2)

在前面的（**[机器学习004-岭回归器的构建和模型评估]**）中，讲解了回归模型的评估方法，主要有均方误差MSE，解释方差分，R方得分等指标。

同样的，对于分类模型，也有很多评估指标来判断该分类模型是否达到我们的要求，这几个评估指标主要是指：准确率（accuracy），精确率（precision），召回率（recall），F1值（F1 measure）

1. 指标的基本概念和计算方法

1.1 准确率（Accuracy）

准确率的定义是：对于给定的测试集，分类模型正确分类的样本数与总样本数之比。举个例子来讲，有一个简单的二分类模型（暂时叫做Classifier_A），专门用于分类苹果和梨，在某个测试集中，有30个苹果+70个梨，这个二分类模型在对这个测试集进行分类的时候，得出该数据集有40个苹果（包括正确分类的25个苹果和错误分类的15个梨）和60个梨（包括正确分类的55个梨和错误分类的5个苹果）。画成矩阵图为：

	预测类别 1-苹果	预测类别 2-梨
实际类别 1-苹果	25	5
实际类别 2-梨	15	55

从图中可以看出，行表示该测试集中实际的类别，比如苹果类一共有25+5=30个，梨类有15+55=70个。其中被分类模型正确分类的是该表格的对角线所在的数字。在sklearn中，这样一个表格被命名为混淆矩阵（Confusion Matrix），所以，按照准确率的定义，可以计算出该分类模型在测试集上的准确率为：

$$Accuracy = \frac{25 + 55}{25 + 5 + 55 + 15} * 100$$

即，该分类模型在测试集上的准确率为80%。

但是，准确率指标并不总是能够评估一个模型的好坏，比如对于下面的情况，假如有一个数据集，含有98个苹果，2个梨，而分类器（暂时叫做Classifier_B）是一个很差劲的分类器，它把数据集的所有样本都划分为苹果，也就是不管输入什么样的样本，该模型都认为该样本是苹果。那么这个表格会是什么样的了？

	预测类别 1-苹果	预测类别 2-梨
实际类别 1-苹果	98	0
实际类别 2-梨	2	0

则该模型的准确率为98%，因为它正确地识别出来了测试集中的98个苹果，只是错误的把2个梨也当做苹果，所以按照准确率的计算公式，该模型有高达98%的准确率。

可是，这样的模型有意义吗？一个把所有样本都预测为苹果的模型，反而得到了非常高的准确率，那么问题出在哪儿了？只能说准确率不可信。特别是对于这种样品数量偏差比较大的问题，准确率的“准确度”会极大的下降。所以，这时就需要引入其他评估指标评价模型的好坏了。

1.2 精确率（Precision）

精确率的定义是：对于给定测试集的某一个类别，分类模型预测正确的比例，或者说：分类模型预测的正样本中有多少是真正的正样本，其计算公式是：

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} = \frac{TP}{TP + FP}$$

所以，根据定义，精确率要区分不同的类别，比如上面我们讨论有两个类别，所以要分类来计算各自的精确率。对于上面提到的Classifier_A和Classifier_B分类模型，我们可以分别计算出其精确率：

精确率计算	Classifier_A	Classifier_B
类别 1-苹果	25/(25+15)=62.5%	98/(98+2)=98%
类别 2-梨	55/(55+5)=91.7%	0/(0+0)

在很多文章中，都讲到把某一个类别当做正类（Positive），把其他类别当做负类（Negative），然后只关注正类的精确率。但是，有的时候我们不知道哪一个类作为正类更合适，比如此处的二分类问题，我们可以把苹果当做正类，也可以把梨当做正类，两者计算出来的精确率明显是不一样的。

1.3 召回率（Recall）

召回率的定义为：对于给定测试集的某一个类别，样本中的正类有多少被分类模型预测正确，其计算公式为：

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} = \frac{TP}{TP + FN}$$

同样的，召回率也要考虑某一个类别，比如，下面我们将苹果作为正类，在Classifier_A模型下得到的表格为：

苹果为正类	预测类别 1-苹果 Positive	预测类别 2-梨 Negative
实际类别 1-苹果	TP=25	FN=5
实际类别 2-梨	FP=15	TN=55

如果我们将梨作为正类，在Classifier_A模型下得到的表格为：

梨为正类	预测类别 1-苹果 Negative	预测类别 2-梨 Positive
实际类别 1-苹果	TN=25	FP=5
实际类别 2-梨	FN=15	TP=55

同样的，可以计算出上面两个模型对给定测试集的召回率，如下表所示：

召回率计算	Classifier_A	Classifier_B
类别 1-苹果	25/(25+5)=83.3%	98/(98+0)=100%
类别 2-梨	55/(55+15)=78.6%	0/(0+2)=0%

1.4 F1值（F1-Measure）

在理想情况下，我们希望模型的精确率越高越好，同时召回率也越高越高，但是，现实情况往往事与愿违，在现实情况下，精确率和召回率像是坐在跷跷板上一样，往往出现一个值升高，另一个值降低，那么，有没有一个指标来综合考虑精确率和召回率了，这个指标就是F值。F值的计算公式为：

$$F = \frac{(a^2+1)*P*R}{a^2*(P+R))}$$

式中：P: Precision， R: Recall, a：权重因子。

当a=1时，F值便是F1值，代表精确率和召回率的权重是一样的，是最常用的一种评价指标。F1的计算公式为：

$$F1 = \frac{2*P*R}{P+R}$$

所以根据上面的精确率和召回率，可以轻松的计算出这两个模型的F1值：

F1 值计算	Classifier_A	Classifier_B
类别 1-苹果	2*62.5%*83.3%/(62.5%+83.3%)=71.4%	99.0%
类别 2-梨	2*91.7%*78.6%/(91.7%+78.6%)=84.6%	0%

2. 用sklearn计算精确率，召回率，F1值

上面第一部分，我们都是手动计算出每种分类模型的各种评价指标，但是上帝告诉我们， 他已经帮我们造好了轮子，我们只需要直接调用即可。废话少说，直接把计算这些评价指标的代码贴过来：

2.1 用sklearn计算分类结果的混淆矩阵

```
# 假如有一个模型在测试集上得到的预测结果为：
y_true = [1, 0, 0, 2, 1, 0, 3, 3, 3] # 实际的类别
y_pred = [1, 1, 0, 2, 1, 0, 1, 3, 3] # 模型预测的类别

# 使用sklearn 模块计算混淆矩阵
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_true, y_pred)
print(confusion_mat) #看看混淆矩阵长啥样
```

----------**输**--------**出**----------

[[2 1 0 0] [0 2 0 0] [0 0 1 0] [0 1 0 2]]

----------完----------

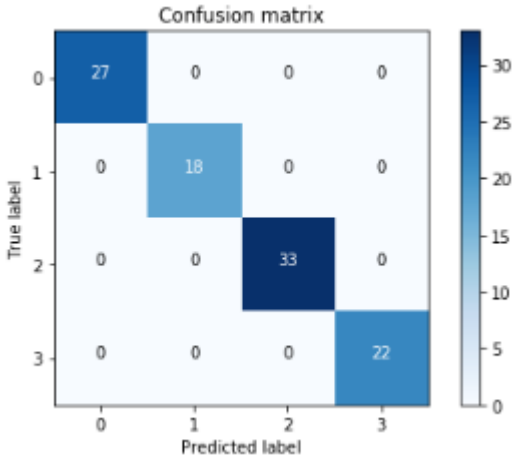
其实这个矩阵的产生过程很简单，比如实际类别是3（y_true=3）一共有三个样本，其中两个预测正确，一个预测错误成1，结果如表格所示：

	预测类别 0	预测类别 1	预测类别 2	预测类别 3
实际类别 0	2	1	0	0
实际类别 1	0	2	0	0
实际类别 2	0	0	1	0
实际类别 3	0	1	0	2

2.2 混淆矩阵可视化

上面虽然把混淆矩阵打印出来了，但是结果很难直接观察对比，下面将混淆矩阵可视化为图表，便于观察。

```
from matplotlib import pyplot as plt
%matplotlib inline
import numpy as np
# 可视化混淆矩阵
def plot_confusion_matrix(confusion_mat):
    '''将混淆矩阵画图并显示出来'''
    plt.imshow(confusion_mat, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(confusion_mat.shape[0])
    plt.xticks(tick_marks, tick_marks)
    plt.yticks(tick_marks, tick_marks)
    thresh = confusion_mat.max() / 2.
    for i, j in itertools.product(range(confusion_mat.shape[0]), range(confusion_mat.shape[1])):
        plt.text(j, i, confusion_mat[i, j],
                 horizontalalignment="center",
                 color="white" if confusion_mat[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

plot_confusion_matrix(confusion_mat)
```



2.3 打印模型的性能报告

Sklearn模块中已经有成熟的对于模型评估的性能报告，里面已经集成了分类模型的精确率，召回率和F1值信息，可以说，是我们分析模型的必备良器。

```
# 打印该模型的性能报告
# 直接使用sklearn打印精度，召回率和F1值
from sklearn.metrics import classification_report
target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3']
print(classification_report(y_true, y_pred, target_names=target_names))
```

----------**输**--------**出**----------

precision recall f1-score support

Class-0 1.00 0.67 0.80 3 Class-1 0.50 1.00 0.67 2 Class-2 1.00 1.00 1.00 1 Class-3 1.00 0.67 0.80 3 avg / total 0.89 0.78 0.79 9

----------**完**----------

这个性能报告表中的评估指标的计算方法如前面第一节讲述的，比如对于Precision, 对于Class_1, Precision=TP/(TP+FP)=2/(2+1+0+1)=0.50，而对于Class_2: Precision=1/（1+0+0+0）=1.0.

对于Recall的计算方法，比如，对于Class_0=TP/(TP+FN)=2/(2+1+0+0)=0.67.所以可以看出，Precision的算法是在混淆矩阵的纵向上运算，而Recall的算法是在混淆矩阵的横向上运算。

F1-score的计算便是直接套用公式即可。后面的support是真实的各个类别的样本数量。

##########**小**********结##########

1，sklearn中已经有成熟的方法可以直接计算混淆矩阵和打印性能报告，这些函数对于模型的评估非常有帮助。

2，精确率，召回率，F1值的计算其实很简单，只要弄明白各自对应的公式，弄明白公式所代表的具体含义即可。

##########

3. 评估朴素贝叶斯多分类模型

在[**机器学习010-用朴素贝叶斯分类器解决多分类问题**]中我们使用了高斯朴素贝叶斯分类器构建了一个多分类模型，将该数据集成功分成了四个类别，貌似结果很不错，可是该文章没有提高用可以量化的性能指标来评估该分类模型的好坏。此处我们可以用上面讲到的各种评价指标来评估该分类模型。

3.1 用交叉验证检验模型的准确性

先加载数据集，然后划分为train set 和testset。构建朴素贝叶斯分类模型，用trainset进行训练，然后用sklearn中的cross_val_score来输出该分类模型在test set上的整体表现，代码如下：

```
# 评估朴素贝叶斯多分类模型
# 1，  准备数据集
data_path='D:\PyProjects\DataSet/NaiveBayers/data_multivar.txt'
df=pd.read_csv(data_path,header=None)
dataset_X,dataset_y=df.iloc[:,:-1],df.iloc[:,-1]
dataset_X=dataset_X.values
dataset_y=dataset_y.values
# 将整个数据集划分为train set和test set
from sklearn.cross_validation import train_test_split
train_X, test_X, train_y,
test_y=train_test_split(dataset_X,dataset_y,test_size=0.25,random_state=42)

# 构建朴素贝叶斯分类模型
from sklearn.naive_bayes import GaussianNB
gaussianNB_new=GaussianNB()
gaussianNB_new.fit(train_X,train_y)

# 2 用交叉验证来检验模型的准确性，只是在test set上验证准确性
from sklearn.cross_validation import cross_val_score
num_validations=5  #将原始数据分成5份进行验证，scoring:调用的方法
# 分数只有在分类器从未见过的数据上计算时才有意义。
accuracy=cross_val_score(gaussianNB_new,test_X,test_y,
scoring='accuracy',cv=num_validations)
print('准确率：{:.2f}%'.format(accuracy.mean()*100))
precision=cross_val_score(gaussianNB_new,test_X,test_y,
 scoring='precision_weighted',cv=num_validations)
print('精确度：{:.2f}%'.format(precision.mean()*100))
recall=cross_val_score(gaussianNB_new,test_X,test_y,
                       scoring='recall_weighted',cv=num_validations)
print('召回率：{:.2f}%'.format(recall.mean()*100))
f1=cross_val_score(gaussianNB_new,test_X,test_y,
                       scoring='f1_weighted',cv=num_validations)
print('F1  值：{:.2f}%'.format(f1.mean()*100))
```

----------**输**---------**出**----------

准确率：99.00% 精确度：99.17% 召回率：99.00% F1 值：98.97%

----------**完**----------

3.2 查看该模型在测试集上的混淆矩阵和性能报告

直接上代码，没什么好讲的。

```
# 使用sklearn 模块计算混淆矩阵
y_pred=gaussianNB_new.predict(test_X)
confusion_mat = confusion_matrix(test_y, y_pred)
print(confusion_mat) #看看混淆矩阵长啥样

print('*'*50)
# 打印该模型的性能报告
# 直接使用sklearn打印精度，召回率和F1值
target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3']
print(classification_report(test_y, y_pred, target_names=target_names))
```

----------**输**---------**出**----------

[[27 0 0 0] [0 18 0 0] [0 0 33 0] [0 0 0 22]]

precision recall f1-score support

Class-0 1.00 1.00 1.00 27 Class-1 1.00 1.00 1.00 18 Class-2 1.00 1.00 1.00 33 Class-3 1.00 1.00 1.00 22

avg / total 1.00 1.00 1.00 100

----------**完**----------

##########**小**************结**##########

1，对于朴素贝叶斯解决这里的多分类问题，从模型检测结果可以看出，该分类器可以很好地将这个数据集的四个类别区分开来，得到的准确率，精确率，召回率，F1值等都比较好。

2，该分类器在测试集上的混淆矩阵都只有TP，没有FP或FN，表明该分类器可以100%的将这测试集中的四个类别区分开来。

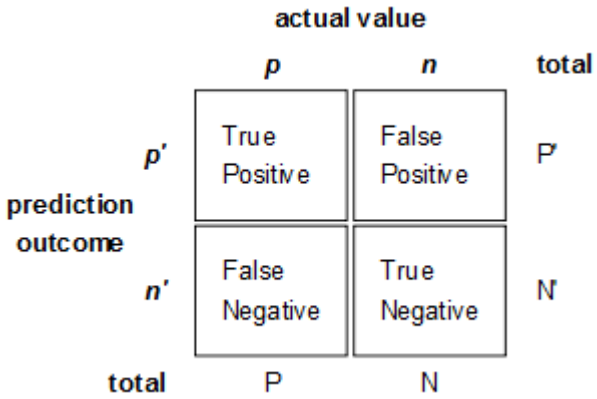
3，在交叉验证时，得到的结果并不是100%，而是99%以上，原因很有可能是交叉验证是采用多步验证来计算这些指标，评估得到的结果可能更可靠普一些。

##########

补充

1、混淆矩阵

混淆矩阵是监督学习中的一种可视化工具，主要用于比较分类结果和实例的真实信息。矩阵中的每一行代表实例的预测类别，每一列代表实例的真实类别。




```
真正(True Positive , TP)：被模型预测为正的正样本。
假正(False Positive , FP)：被模型预测为正的负样本。
假负(False Negative , FN)：被模型预测为负的正样本。
真负(True Negative , TN)：被模型预测为负的负样本。

真正率(True Positive Rate,TPR)：TPR=TP/(TP+FN)，即被预测为正的正样本数 /正样本实际数。
假正率(False Positive Rate,FPR) ：FPR=FP/(FP+TN)，即被预测为正的负样本数 /负样本实际数。
假负率(False Negative Rate,FNR) ：FNR=FN/(TP+FN)，即被预测为负的正样本数 /正样本实际数。
真负率(True Negative Rate,TNR)：TNR=TN/(TN+FP)，即被预测为负的负样本数 /负样本实际数/2
```

2、准确率（Accuracy）

准确率是最常用的分类性能指标。 Accuracy = (TP+TN)/(TP+FN+FP+TN)

即正确预测的正反例数 /总数

3、精确率（Precision）

精确率容易和准确率被混为一谈。其实，精确率只是针对预测正确的正样本而不是所有预测正确的样本。表现为预测出是正的里面有多少真正是正的。可理解为查准率。 Precision = TP/(TP+FP)

即正确预测的正例数 /预测正例总数

4、召回率（Recall）

召回率表现出在实际正样本中，分类器能预测出多少。与真正率相等，可理解为查全率。 Recall = TP/(TP+FN)，

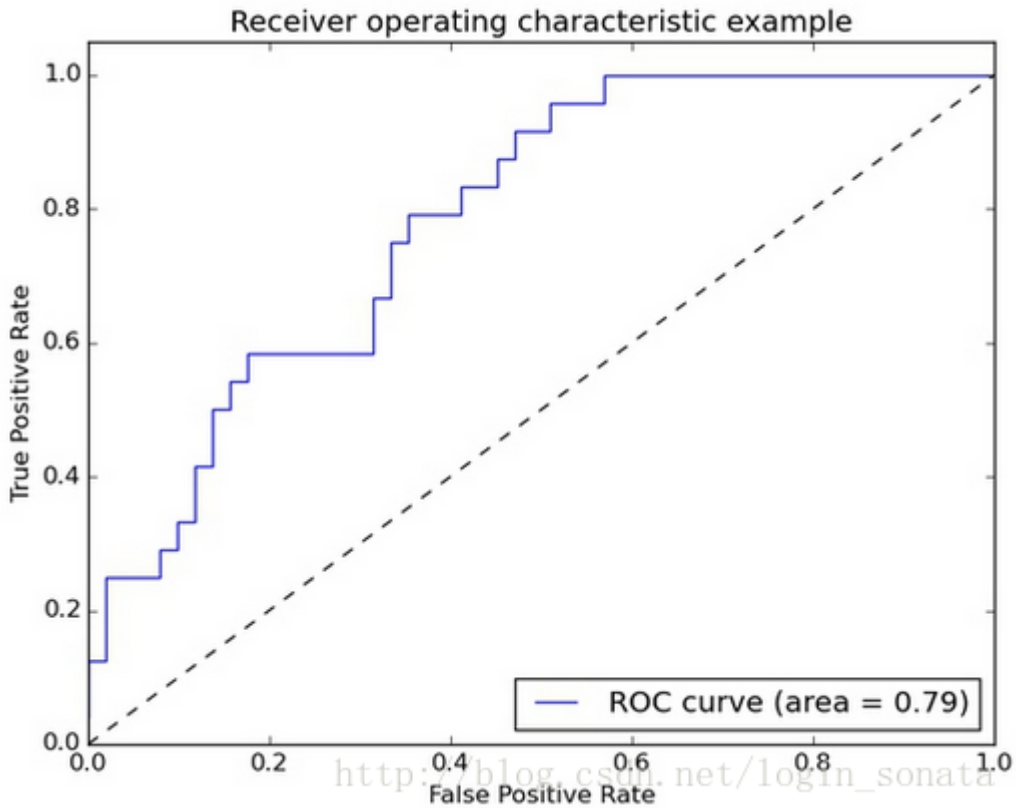
即正确预测的正例数 /实际正例总数

5、F1 -score

F值是精确率和召回率的调和值，更接近于两个数较小的那个，所以精确率和召回率接近时，F值最大。很多推荐系统的评测指标就是用F值的。 2/F1 = 1/Precision + 1/Recall

6、ROC曲线

逻辑回归里面，对于正负例的界定，通常会设一个阈值，大于阈值的为正类，小于阈值为负类。如果我们减小这个阈值，更多的样本会被识别为正类，提高正类的识别率，但同时也会使得更多的负类被错误识别为正类。为了直观表示这一现象，引入ROC。根据分类结果计算得到ROC空间中相应的点，连接这些点就形成ROC 曲线，横坐标为False Positive Rate(FPR假正率)，纵坐标为True Positive Rate(TPR真正率)。一般情况下，这个曲线都应该处于(0,0)和(1,1)连线的上方.如图：

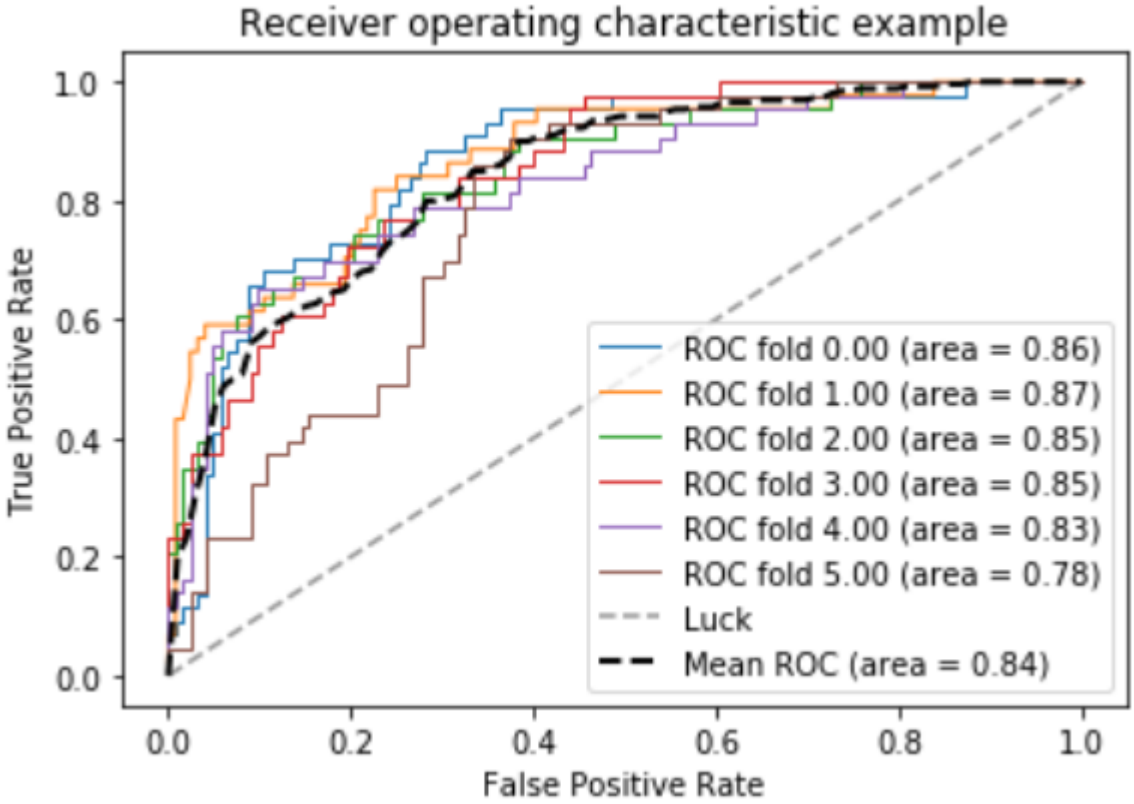


ROC曲线中的四个点和一条线：
点(0,1)：即FPR=0，TPR=1，意味着FN＝0且FP＝0，将所有的样本都正确分类。
点(1,0)：即FPR=1，TPR=0，最差分类器，避开了所有正确答案。
点(0,0)：即FPR=TPR=0，FP＝TP＝0，分类器把每个实例都预测为负类。
点(1,1)：分类器把 每个实例都预测为正类。
总之：ROC曲线越接近左上角，该分类器的性能越好。而且一般来说，如果ROC是光滑的，那么基本可以判断没有太大的overfitting

```
#绘制ROC曲线函数
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
def drawROC(classifier,X,y):
    #X:训练集/测试集
    #y:训练集标签/测试集标签
    print(X.shape)
    print(y.shape)
    # 画平均ROC曲线的两个参数
    mean_tpr = 0.0     # 用来记录画平均ROC曲线的信息
    mean_fpr = np.linspace(0, 1, 100)
    cnt = 0
    cv = StratifiedKFold(n_splits=6)  #导入该模型，后面将数据划分6份
    for i, (train, test) in enumerate(cv.split(X,y)):  #利用模型划分数据集和目标变量 为一一对应的下标
        cnt +=1
        probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test]) # 训练模型后预测每条样本得到两种结果的概率
        fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])  # 该函数得到伪正例、真正例、阈值，这里只使用前两个

        mean_tpr += np.interp(mean_fpr, fpr, tpr)   # 插值函数 interp(x坐标,每次x增加距离,y坐标) 累计每次循环的总值后面求平均值
        mean_tpr[0] = 0.0  # 将第一个真正例=0 以0为起点

        roc_auc = auc(fpr, tpr)  # 求auc面积
        plt.plot(fpr, tpr, lw=1, label='ROC fold {0:.2f} (area = {1:.2f})'.format(i, roc_auc))
 # 画出当前分割数据的ROC曲线

    plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck') # 画对角线

    mean_tpr /= cnt   # 求数组的平均值
    mean_tpr[-1] = 1.0   # 坐标最后一个点为 (1,1)   以1为终点
    mean_auc = auc(mean_fpr, mean_tpr)

    plt.plot(mean_fpr, mean_tpr, 'k--',label='Mean ROC (area = {0:.2f})'.format(mean_auc), lw=2)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')    # 可以使用中文，但需要导入一些库即字体
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
#调用
drawROC(test_X[:1000,:],test_y[:1000])
```

效果：



7、AUC值（Area Under Curve）

定义为ROC曲线下的面积(ROC的积分)，通常大于0.5小于1。

随机抽出一对样本（一个正样本，一个负样本），然后用训练得到的分类器来对这两个样本进行预测，预测得到正样本的概率大于负样本概率的概率，就是AUC。（详细参考：https://blog.csdn.net/qq_22238533/article/details/78666436）

AUC值(面积)越大的分类器，性能越好。

从AUC判断分类器（预测模型）优劣的标准：

- 0.90~1 = very good (A)
- 0.80~0.90 = good (B)
- 0.70~0.80 = not so good (C)
- 0.60~0.70 = poor (D)
- 0.50~0.60 = fail (F)

```
# 打印ROC曲线
drawROC(gaussianNB,test_X,test_y)
```

参考资料:

1, Python机器学习经典实例，Prateek Joshi著，陶俊杰，陈小莉译