



**数据挖掘实验报告**

**姓 名**

**学 号**

**专业班级 物联网工程19-2**

**指导教师 张玉红、吕俊伟**

**院系名称 计算机与信息学院**

**2022年 06月 19 日**

目录

[1. 实验目的 3](#_Toc12341)

[2. 实验要求 3](#_Toc22521)

[3. 实验环境 3](#_Toc12932)

[4. 实验结果要求 3](#_Toc719)

[5. 实验步骤 3](#_Toc12760)

[5.1. 读取数据 3](#_Toc17900)

[5.2. 数据清洗 4](#_Toc25431)

[5.3. 数据划分 4](#_Toc1680)

[5.4. 数据向量化及降维 5](#_Toc20775)

[5.5. 分类器实现 5](#_Toc8333)

[5.6. 分类器调参 7](#_Toc270)

[5.7. 结果评估 8](#_Toc7904)

[6. 实验结果分析 11](#_Toc17694)

[7. 实验代码 15](#_Toc23091)

[8. 参考文献 20](#_Toc11954)

### 不平衡的分类任务之垃圾短信分类

1. 实验目的

1）理解数据不平衡度对分类性能的影响，以及不平衡分类方法的框架；

2）基模型可以调用已有的包，训练学生熟悉数据挖掘的基本框架；

3）学会多维度的对模型进行评估以及模型中参数的讨论。

1. 实验要求

1）要有针对数据特点的预处理步骤，包括去停用词，降维等。重点关注其中的上/下采样，自行划分 train 和 test，一般按 7：3 划分。

2）原则上不限制模型，决策树，NB，NN，SVM，random forest 均可，且不限于上述方法，可选择不平衡的代价敏感模型。

3）可采用 BOW，主题模型以及词向量等多种表示方式，图像数据集可采用 LBP，HOG，SURF 等特征表示方式。

1. 实验环境

实验平台：Windows11操作系统

实验工具：PyCharm、Python3.8

1. 实验结果要求

1）实现一个或多个基本分类模型，并计算其评估指标如准确率，召回率等，重点关注不平衡数据分类的代价以及 ROC 等指标；

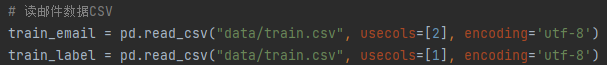
2）对模型中的关键参数，（如决策树中停止分裂条件，NN 中层数等参数）进行不同范围的取值，讨论参数的最佳取值范围。

3）对比分析不同的特征表示方法对结果的影响。

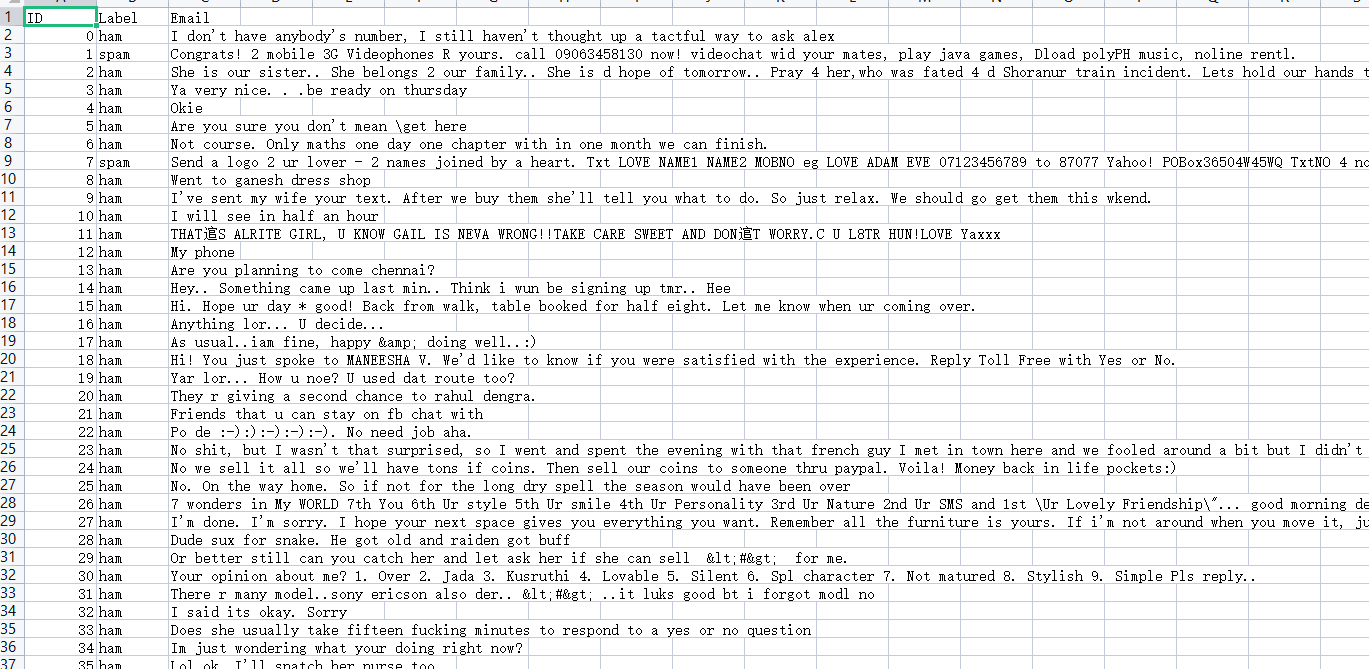
4）考察数据标签的不平衡度对分类模型的影响。

5）选做：针对不平衡分类问题，改进数据或模型，以提高当前的精度。

1. 实验步骤
   1. 读取数据



第一列为短信类型，第二列为短信内容，原始数据集如下：



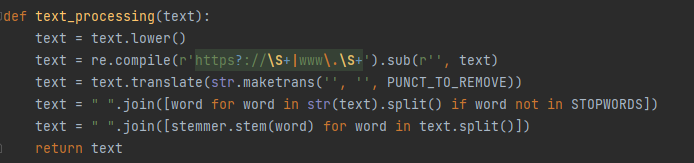
* 1. 数据清洗

导入nltk包中的停词表



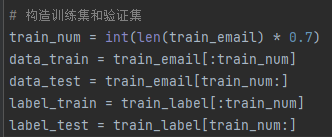


进行数据清洗，正则去除垃圾邮件中附带的网址链接，去除停用词

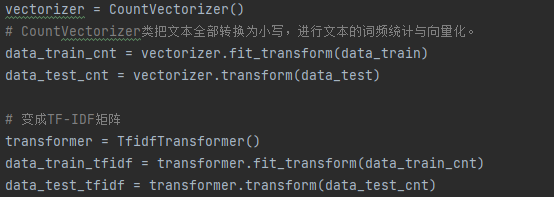


* 1. 数据划分

训练集和测试集划分为7:3



* 1. 数据向量化及降维

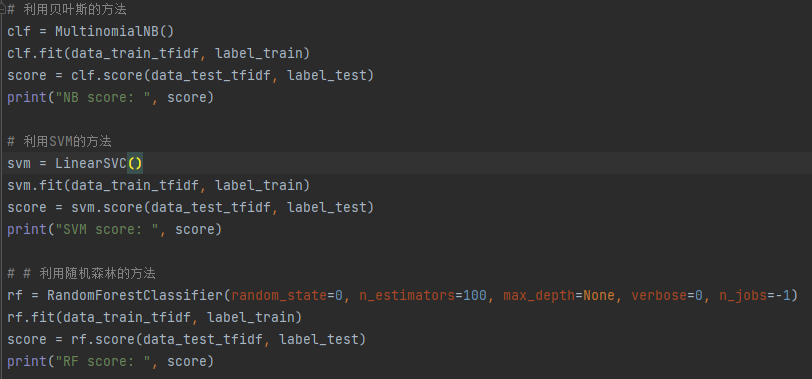
 TfidfVectorizer、CountVectorizer和TfidfTransformer是sklearn中处理自然语言常用的工具。

TfidfVectorizer相当于CountVectorizer+TfidfTransformer。

CountVectorizer 的作用是将文本文档转换为计数的稀疏矩阵。

TfidfTransformer用于计算tf-idf，tf表示词频率，而tf-idf表示词频率乘以逆文档频率。

* 1. 分类器实现

 调用包内函数构造分类器，分别实现朴素贝叶斯、支持向量机SVM、随机森林三种分类器。

MultinomialNB()函数全称是先验为多项式分布的朴素贝叶斯。它假设特征是由一个简单多项式分布生成的。多项分布可以描述各种类型样本出现次数的概率，因此多项式朴素贝叶斯非常适合用于描述出现次数的特征。该模型常用于文本分类，特征表示的是次数，例如某个词语的出现次数。IMG_256

|  |  |
| --- | --- |
| 参数 | 说明 |
| alpha | 浮点型可选参数，默认为1.0，其实就是添加拉普拉斯平滑，即为上述公式中的λ ，如果这个参数设置为0，就是不添加平滑 |
| fit\_prior | 布尔型可选参数，默认为True。布尔参数fit\_prior表示是否要考虑先验概率，如果是false，则所有的样本类别输出都有相同的类别先验概率。否则可以自己用第三个参数class\_prior输入先验概率，或者不输入第三个参数class\_prior，MultinomialNB自己从训练集样本来计算先验概率，此时的先验概率为 P ( Y = C k ) = m k / m P(Y=C\_k)=m\_k/mP(Y=Ck​)=mk​/m。其中m为训练集样本总数量，m k m\_kmk​为输出为第k类别的训练集样本数。 |
| class\_prior | 可选参数，默认为None。 |

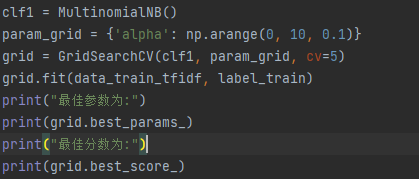
LinearSVC实现了线性分类支持向量机，它是给根据liblinear实现的，可以用于二类分类，也可以用于多类分类，LinearSVC使用 （One-vs-All）方式来实现多分类问题。

|  |  |
| --- | --- |
| 参数 | 说明 |
| Penalty | string,‘l1’or‘l2’(default=l2)指定惩罚中使用的规范。l2惩罚是SVC中使用的标准。 l1导致稀疏的coef\_向量。 |
| loss | string,‘hinge’or‘squared\_hinge’(default=squared\_hinge) 指定损失函数。“hinge”是标准的[SVM](https://so.csdn.net/so/search?q=SVM&spm=1001.2101.3001.7020" \t "_blank)损失（例如由SVC类使用），而“squared\_hinge”是hinge损失的平方。 |
| dual | bool, (default=True) 选择算法以解决双优化或原始优化问题。当n\_samples> n\_features时，首选dual=False。 |
| tol | float, optional (default=1e-4)公差停止标准。 |
| C | float, optional (default=1.0)错误项的惩罚参数 |
| multi\_class | string,‘ovr’or‘crammer\_singer’(default=’ovr’) 如果y包含两个以上的类，则确定多类策略。“ovr”训练n\_classes one-vs-rest分类器，而“crammer\_singer”优化所有类的联合目标 |
| fit\_intercept | boolean, optional (default=True)是否计算此模型的截距 |
| random\_state | 如果是int，则random\_state是随机数生成器使用的种子; 如果是RandomState实例，则random\_state是随机数生成器; 如果为None，则随机数生成器是np.random使用的RandomState实例。 |
| max\_iter | int, (default=1000)要运行的最大迭代次数。 |

随机森林是非常具有代表性的Bagging集成算法，它的所有基评估器都是决策树，分类树组成的森林就叫做随机森林分类器，森林中树木的数量，即基评估器的数量越多，模型的效果往往越好。

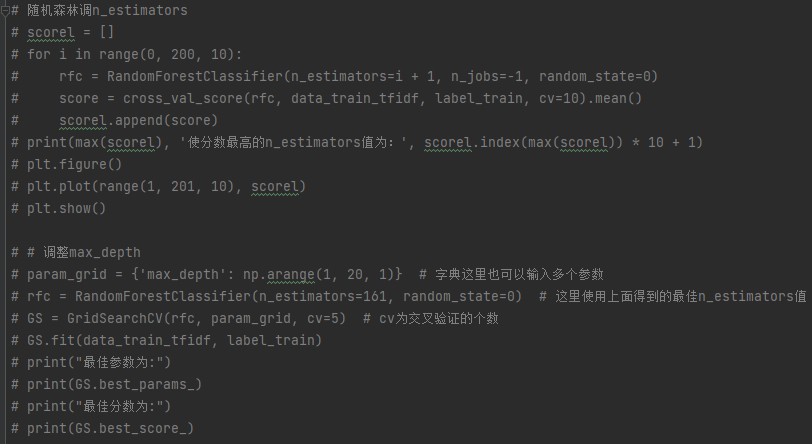
| 参数 | 说明 |
| --- | --- |
| n\_estimators | 决策树的个数，越多越好，但是性能会越差。 |
| bootstrap | Bool,是否有放回的采样。 |
| oob\_score | Bool,oob（out of band，带外）数据，即 在某次决策树训练中没有被bootstrap选中的数据。 |
| n\_jobs | 并行job个数，并行可以提高性能。n=1不并行；n= n n个并行；n= -1 CPU有多少core，就启动多少job |
| class\_weight=None | 各个label的权重。 |
| max\_depth： | 树的最大[深度](https://so.csdn.net/so/search?q=%E6%B7%B1%E5%BA%A6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_42479987/article/details/_blank)。如果为None，则将节点展开，直到所有叶子都是纯净的(只有一个类)，或者直到所有叶子都包含少于min\_samples\_split个样本。默认是None |

* 1. 分类器调参
     1. 朴素贝叶斯



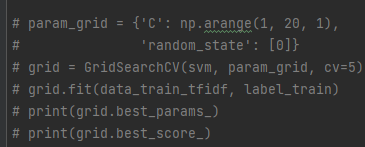
每0.1步长检测一次分数，输出最优。

* + 1. 随机森林



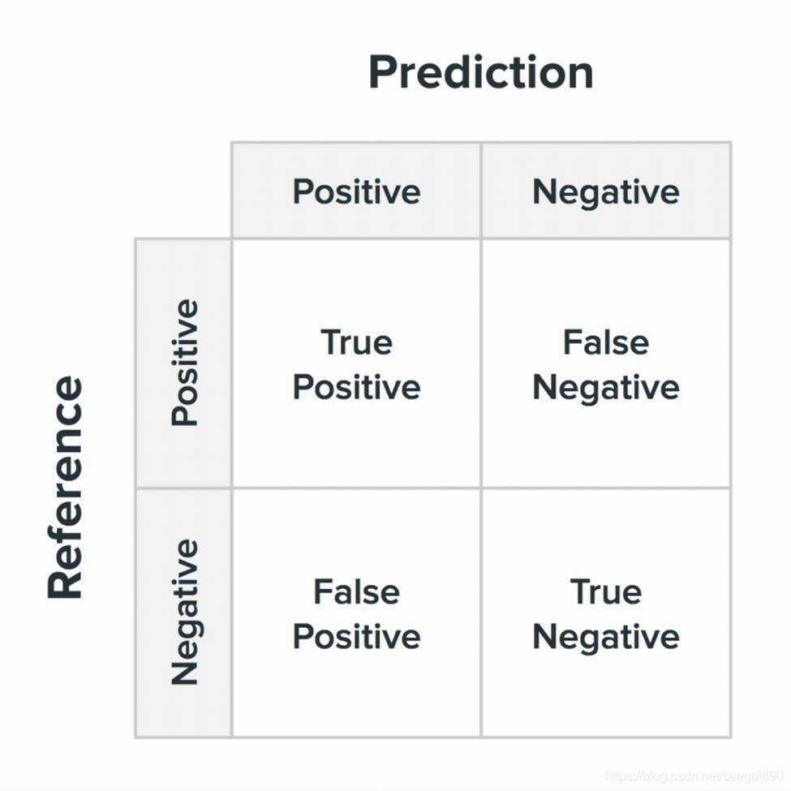
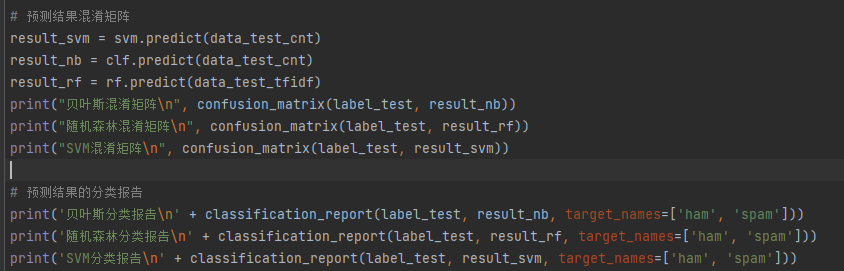
先调n\_estimators，寻找最优参数，然后带入到其中，调整max\_depth

* + 1. SVM



调错误项的惩罚参数C

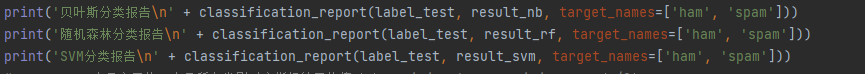
* 1. 结果评估
     1. 混淆矩阵



TP == True Postive ==真阳性 ：样本的真实类别是正例，并且模型预测的结果也是正例；  
FP = = False Positive == 假阳性 ：样本的真实类别是负例，但是模型将其预测成为正例；  
FN == False Negative == 假阴性 ：样本的真实类别是正例，但是模型将其预测成为负例；  
TN == True Negative == 真阴性 ：样本的真实类别是负例，并且模型将其预测成为负例；

对于二分类问题来说，精确率Precision=TP/(TP+FP)，召回率recall=TP/(TP+FN)，准确率accuracy=(TP+FN+FP+TN)，准确率中的分子值就是矩阵对角线上的值。

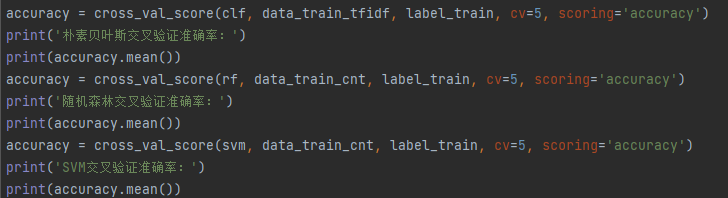
* + 1. 分类报告



sklearn中的classification\_report函数用于显示主要分类指标的文本报告．在报告中显示每个类的精确度，召回率，F1值等信息。

accuracy表示准确率，也即正确预测样本量与总样本量的比值，；macro avg表示宏平均，表示所有类别对应指标的平均值；weighted avg表示带权重平均，表示类别样本占总样本的比重与对应指标的乘积的累加和。

* + 1. 交叉验证



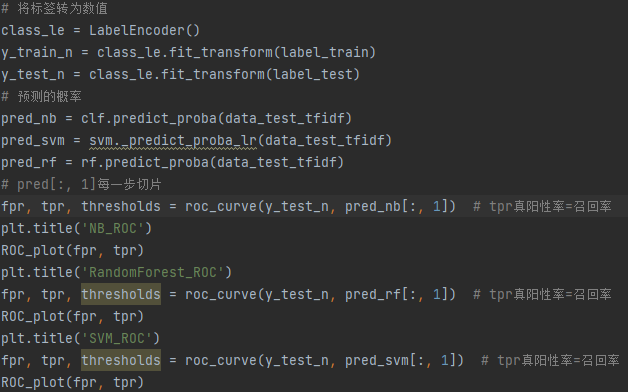
**k折交叉验证：**

1、首先，将全部样本划分成k个大小相等的样本子集；

2、依次遍历这k个子集，每次把当前子集作为验证集，其余所有样本作为训练集，进行模型的训练和评估；

3、最后把k次评估指标的平均值作为最终的评估指标。

* + 1. ROC图绘制



 ROC的全名叫做Receiver Operating Characteristic，中文名字叫“受试者工作特征曲线”，平面的横坐标是false positive rate(FPR)，纵坐标是true positive rate(TPR)。对某个分类器而言，我们可以根据其在测试样本上的表现得到一个TPR和FPR点对。



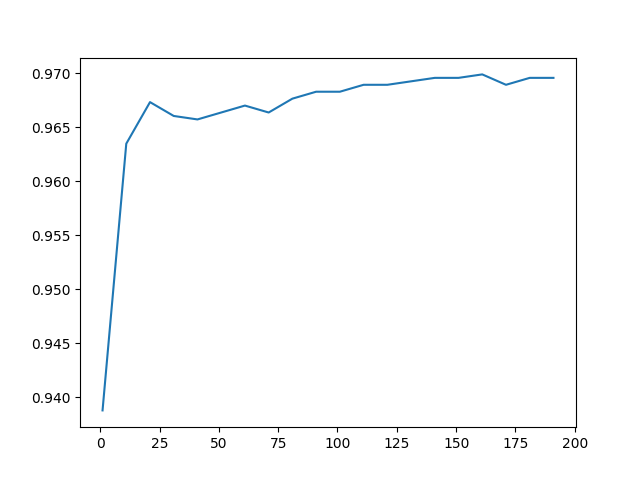
AUC表示ROC曲线下的面积，主要用于衡量模型的泛化性能，即分类效果的好坏。

优化AUC就是希望同时优化TPR和(1-FPR)。TPR又被称为sensitivity(灵敏度)，(1-FPR)又被成为specificity(特异度),优化F1就是希望同时优化recall和precision。AUC的计算方法同时考虑了学习器对于正例和负例的分类能力，在样本不平衡且可能小幅变化的情况下，依然能够对分类器做出合理的评价，是对模型整体性能的评价。AUC不容易受样本不平衡的影响，所以对于不平衡分类的情况优先使用AUC。

1. 实验结果分析

数据中ham有3866条，spam有592条，总数4458。

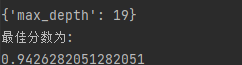
随机森林调参：



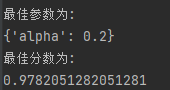


将以上参数代入分类器，再次调参max\_depth

使用了网格搜索之后，发现分数下降，所以不调整max\_depth。



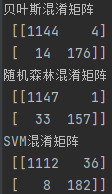
朴素贝叶斯调参



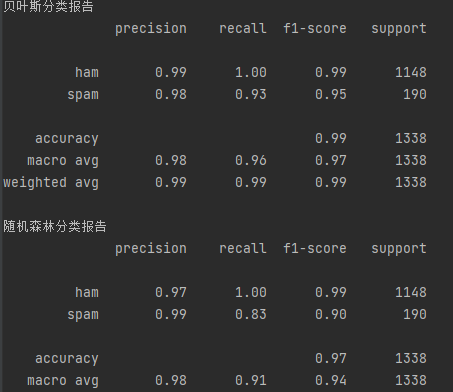
SVM调参

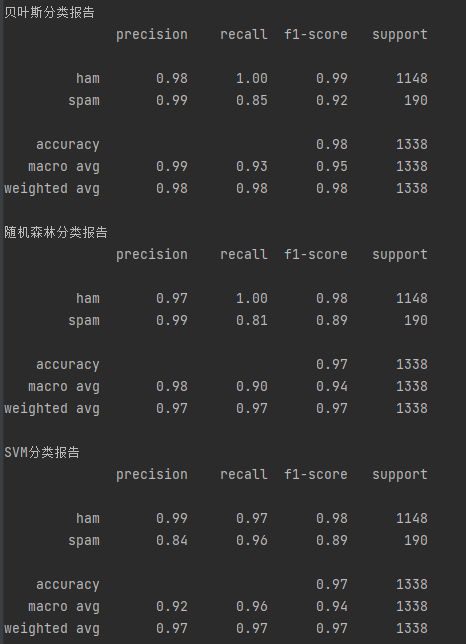


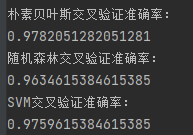
三种分类器的混淆矩阵：

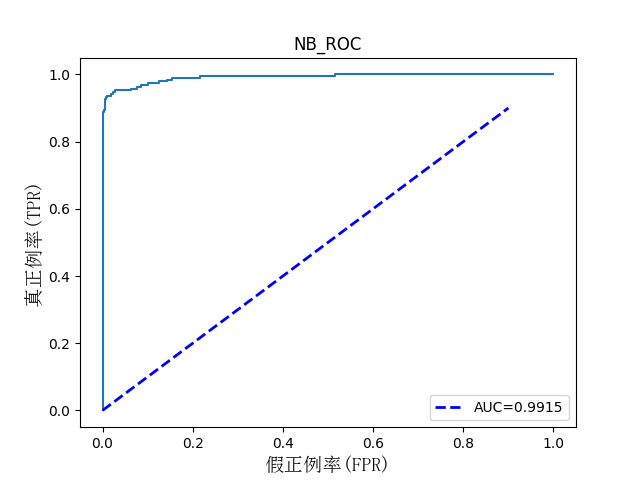


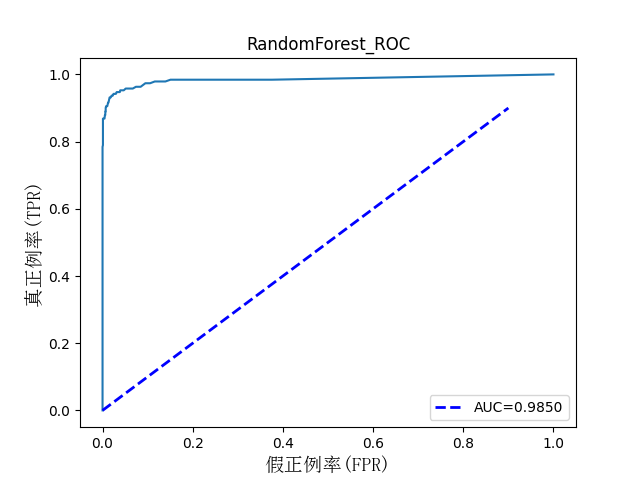
更加详细的报告：

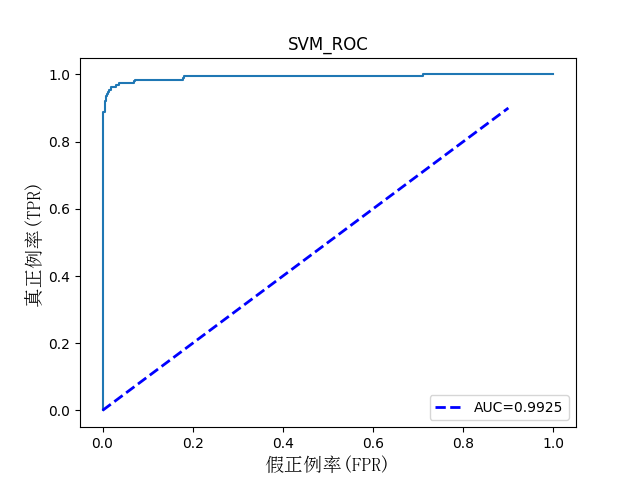






下面是绘制的ROC曲线图：





AUC(ROC曲线下的面积)越大，分类器效果越好，可以得出结论：

SVM>朴素贝叶斯>随机森林

1. 实验代码

import warnings  
import numpy as np  
import pandas as pd  
import string  
from sklearn.preprocessing import LabelEncoder  
from matplotlib import pyplot as plt  
from matplotlib.font\_manager import FontProperties  
from nltk.corpus import stopwords  
from nltk.stem.porter import PorterStemmer  
from sklearn.metrics import roc\_curve, classification\_report, auc  
import re  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.svm import LinearSVC  
from sklearn.metrics import confusion\_matrix  
from sklearn.feature\_extraction.text import TfidfTransformer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import cross\_val\_score, GridSearchCV  
warnings.filterwarnings("ignore")  
stemmer = PorterStemmer()  
PUNCT\_TO\_REMOVE = string.punctuation  
STOPWORDS = set(stopwords.words("english"))  
  
  
def text\_processing(text):  
 text = text.lower()  
 text = re.compile(r'https?://\S+|www\.\S+').sub(r'', text)  
 text = text.translate(str.maketrans('', '', PUNCT\_TO\_REMOVE))  
 text = " ".join([word for word in str(text).split() if word not in STOPWORDS])  
 text = " ".join([stemmer.stem(word) for word in text.split()])  
 return text  
  
  
def ROC\_plot(fpr, tpr): # 画出函数图像  
 font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)  
 plt.xlabel('假正例率(FPR)', fontproperties=font)  
 plt.ylabel('真正例率(TPR)', fontproperties=font)  
 x = np.arange(0, 1.1, 0.2)  
 y = np.arange(0, 1.1, 0.2)  
 auc1 = auc(fpr, tpr)  
 plt.xticks(x)  
 plt.yticks(y)  
 plt.plot(fpr, tpr)  
 x1 = np.arange(0, 1.0, 0.1)  
 plt.plot(x1, x1, color='blue', linewidth=2, linestyle='--', label='AUC=%.4f' % auc1)  
 plt.legend(loc='lower right')  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 读邮件数据CSV  
 train\_email = pd.read\_csv("data/train.csv", usecols=[2], encoding='utf-8')  
 train\_label = pd.read\_csv("data/train.csv", usecols=[1], encoding='utf-8')  
 # 数据中ham有3866条，总数4458，属于不平衡数据集  
  
# 数据预处理  
train\_email['Email'] = train\_email['Email'].apply(text\_processing)  
# 将内容转为list类型  
train\_email = np.array(train\_email).reshape((1, len(train\_email)))[0].tolist()  
train\_label = np.array(train\_label).reshape((1, len(train\_email)))[0].tolist()  
  
# 构造训练集和验证集  
train\_num = int(len(train\_email) \* 0.7)  
data\_train = train\_email[:train\_num]  
data\_test = train\_email[train\_num:]  
label\_train = train\_label[:train\_num]  
label\_test = train\_label[train\_num:]  
  
# 使用词袋模型  
vectorizer = CountVectorizer()  
# CountVectorizer类把文本全部转换为小写，进行文本的词频统计与向量化。  
data\_train\_cnt = vectorizer.fit\_transform(data\_train)  
data\_test\_cnt = vectorizer.transform(data\_test)  
  
# 变成TF-IDF矩阵  
transformer = TfidfTransformer()  
data\_train\_tfidf = transformer.fit\_transform(data\_train\_cnt)  
data\_test\_tfidf = transformer.transform(data\_test\_cnt)  
  
# 以上两步可用下面替代，TfidfVectorizer 相当于 CountVectorizer 和 TfidfTransformer 的结合使用  
# vectorizer\_tfidf = TfidfVectorizer(sublinear\_tf=True)  
# data\_train\_tfidf = vectorizer\_tfidf.fit\_transform(data\_train)  
# data\_test\_tfidf = vectorizer\_tfidf.transform(data\_test)  
  
  
# 利用贝叶斯的方法  
clf = MultinomialNB(alpha=0.2)  
clf.fit(data\_train\_tfidf, label\_train)  
score = clf.score(data\_test\_tfidf, label\_test)  
print("NB score: ", score)  
  
# 利用SVM的方法  
svm = LinearSVC(C=4, random\_state=0)  
svm.fit(data\_train\_tfidf, label\_train)  
score = svm.score(data\_test\_tfidf, label\_test)  
print("SVM score: ", score)  
  
# # 利用随机森林的方法，n\_estimators是森林中树木的数量，即基评估器的数量，越大，模型的效果往往越好。  
# random\_state固定时，随机森林中生成是一组固定的树，但每棵树依然是不一致的。max\_depth如果为None，则将节点展开，直到所有叶子都是纯净的  
rf = RandomForestClassifier(random\_state=0, n\_estimators=161, max\_depth=None, verbose=0, n\_jobs=-1)  
rf.fit(data\_train\_tfidf, label\_train)  
score = rf.score(data\_test\_tfidf, label\_test)  
print("RF score: ", score)  
  
# [0.001, 0.01, 0.1, 1, 10, 100]  
# clf1 = MultinomialNB()  
# param\_grid = {'alpha': np.arange(0, 10, 0.1)}  
# grid = GridSearchCV(clf1, param\_grid, cv=5)  
# grid.fit(data\_train\_tfidf, label\_train)  
# print("最佳参数为:")  
# print(grid.best\_params\_)  
# print("最佳分数为:")  
# print(grid.best\_score\_)  
  
  
# 随机森林调n\_estimators  
# scorel = []  
# for i in range(0, 200, 10):  
# rfc = RandomForestClassifier(n\_estimators=i + 1, n\_jobs=-1, random\_state=0)  
# score = cross\_val\_score(rfc, data\_train\_tfidf, label\_train, cv=10).mean()  
# scorel.append(score)  
# print(max(scorel), '使分数最高的n\_estimators值为：', scorel.index(max(scorel)) \* 10 + 1)  
# plt.figure()  
# plt.plot(range(1, 201, 10), scorel)  
# plt.show()  
  
# # 调整max\_depth  
# param\_grid = {'max\_depth': np.arange(1, 20, 1)} # 字典这里也可以输入多个参数  
# rfc = RandomForestClassifier(n\_estimators=161, random\_state=0) # 这里使用上面得到的最佳n\_estimators值  
# GS = GridSearchCV(rfc, param\_grid, cv=5) # cv为交叉验证的个数  
# GS.fit(data\_train\_tfidf, label\_train)  
# print("最佳参数为:")  
# print(GS.best\_params\_)  
# print("最佳分数为:")  
# print(GS.best\_score\_)  
  
# param\_grid = {'C': np.arange(1, 20, 1),  
# 'random\_state': [0]}  
# grid = GridSearchCV(svm, param\_grid, cv=5)  
# grid.fit(data\_train\_tfidf, label\_train)  
# print(grid.best\_params\_)  
# print(grid.best\_score\_)  
  
# 预测结果混淆矩阵  
result\_svm = svm.predict(data\_test\_cnt)  
result\_nb = clf.predict(data\_test\_cnt)  
result\_rf = rf.predict(data\_test\_tfidf)  
print("贝叶斯混淆矩阵\n", confusion\_matrix(label\_test, result\_nb))  
print("随机森林混淆矩阵\n", confusion\_matrix(label\_test, result\_rf))  
print("SVM混淆矩阵\n", confusion\_matrix(label\_test, result\_svm))  
  
# 预测结果的分类报告  
print('贝叶斯分类报告\n' + classification\_report(label\_test, result\_nb, target\_names=['ham', 'spam']))  
print('随机森林分类报告\n' + classification\_report(label\_test, result\_rf, target\_names=['ham', 'spam']))  
print('SVM分类报告\n' + classification\_report(label\_test, result\_svm, target\_names=['ham', 'spam']))  
# macro avg表示宏平均，表示所有类别对应指标的平均值（（precision.ham+precision.spam）/2）  
# weighted avg表示带权重平均，表示类别样本占总样本的比重与对应指标的乘积的累加和  
# （（precision.ham \* support.ham /support +precision.spam \* support.spam /support））  
# 验证模型的性能  
# 交叉验证用于评估模型的预测性能，尤其是训练好的模型在新数据上的表现，可以在一定程度上减小过拟合。cv：交叉验证生成器或可迭代的次数  
# 进行交叉验证数据评估, 数据分为5部分, 每次用一部分作为测试集，输出5次交叉验证的准确率  
accuracy = cross\_val\_score(clf, data\_train\_tfidf, label\_train, cv=5, scoring='accuracy')  
print('朴素贝叶斯交叉验证准确率：')  
print(accuracy.mean())  
accuracy = cross\_val\_score(rf, data\_train\_cnt, label\_train, cv=5, scoring='accuracy')  
print('随机森林交叉验证准确率：')  
print(accuracy.mean())  
accuracy = cross\_val\_score(svm, data\_train\_cnt, label\_train, cv=5, scoring='accuracy')  
print('SVM交叉验证准确率：')  
print(accuracy.mean())  
# 将标签转为数值  
class\_le = LabelEncoder()  
y\_train\_n = class\_le.fit\_transform(label\_train)  
y\_test\_n = class\_le.fit\_transform(label\_test)  
# 预测的概率  
pred\_nb = clf.predict\_proba(data\_test\_tfidf)  
pred\_svm = svm.\_predict\_proba\_lr(data\_test\_tfidf)  
pred\_rf = rf.predict\_proba(data\_test\_tfidf)  
# pred[:, 1]每一步切片  
fpr, tpr, thresholds = roc\_curve(y\_test\_n, pred\_nb[:, 1]) # tpr真阳性率=召回率  
plt.title('NB\_ROC')  
ROC\_plot(fpr, tpr)  
plt.title('RandomForest\_ROC')  
fpr, tpr, thresholds = roc\_curve(y\_test\_n, pred\_rf[:, 1]) # tpr真阳性率=召回率  
ROC\_plot(fpr, tpr)  
plt.title('SVM\_ROC')  
fpr, tpr, thresholds = roc\_curve(y\_test\_n, pred\_svm[:, 1]) # tpr真阳性率=召回率  
ROC\_plot(fpr, tpr)

1. 参考文献

TfidfVectorizer、CountVectorizer 和TfidfTransformer

<https://zhuanlan.zhihu.com/p/166636681>

sklearn.naive\_bayes.MultinomialNB()函数解析

<https://blog.csdn.net/qq_41551450/article/details/105926001>

SVM有监督学习LinearSVC, LinearSVR,SVC,SVR -- 024

<https://blog.csdn.net/u010986753/article/details/105021495>

【机器学习-RFC】sklearn.ensemble.RandomForestClassifier参数说明

<https://blog.csdn.net/weixin_41990278/article/details/93342321>

sklearn 中 predict 方法和 predict\_proba 方法的区别和使用

<https://blog.csdn.net/qq_43468807/article/details/105740396>

ROC曲线学习总结

<https://blog.csdn.net/qq_30992103/article/details/99730059>