

**项目开发报告**

**题目：互联网虚假新闻检测**

**课程名称： \_\_\_机器学习\_ \_\_**

**专业班级： \_\_\_ \_ \_\_\_**

**学 号： \_ \_\_**

**姓 名： \_\_ \_\_\_**

**指导教师： \_\_ \_ \_\_\_**

**报告日期： \_\_2021年11月29日**

**计算机科学与技术学院**

# 项目开发报告

## 1.1项目目的

互联网的深度普及加速了“信息时代”的到来，网络中的每个人都可以极低甚至“零”成本的方式创造信息，同时每个人也都可以成为信息传播路径上的一个节点。这种获取、创造和传播信息的便捷给社会的进步和人类的发展带来了巨大的福祉。然而凡事终有两面，互联网也开始为人类社会带来很大挑战——网络中的信息鱼龙混杂，充斥大量的虚假信息。

本次实验，旨在通过机器学习的方式，训练适合的模型，进而检测网络中的虚假新闻。

## 1.2问题分析

此次实验，我们需要根据给定的train.csv文件里面的新闻文本，评论和新闻所属的类别，训练合适的模型，然后对于test.csv里面给出的新闻以及评论，我们需要用训练好的模型给出预测结果。

## 1.3设计与分析

### 1.3.1 数据分析

1.新闻的类别（label）：-1：虚假新闻，0：无需判断的新闻，1：真实新闻。由此可见该问题是一个三分类的问题。

2.训练数据集train.csv：包含39928条数据，各个字段及其含义分别为content:新闻文本，comment\_all：评论，label：新闻所属的类别。

3.测试数据集test.csv：包含9982条数据，各个字段及其含义分别为content：新闻文本，comment\_all：评论。

4.停用词词库：本次实验数据清洗时所用到的停用词词库hit-stopwords.txt。

### 1.3.2 算法流程设计

由于本次实验要解决的问题的数据都来自于真实的生产环境，因此往往面临着数据噪声大，难以处理，脏数据多等特征，所以在设计算法的过程中对数据的预处理就显得尤其重要。基于上述考虑，整个算法的流程分为三步：清洗数据，将文本转化为词向量，以及最终的学习算法。

本次实验使用的编程语言是Python，集成开发环境为Pycharm，主要用到的第三方库有pandas，jieba，sklearn和numpy。总体的算法流程图描述如下：



图1：实验总体流程图

### 1.3.3 机器学习算法设计

1.输入数据：原始数据保存在train.csv和test.csv中，因此使用pandas库里面的read\_csv方法，读入数据。并使用fillna方法填充缺失数据。具体实现过程如图2所示，结果如图3所示。

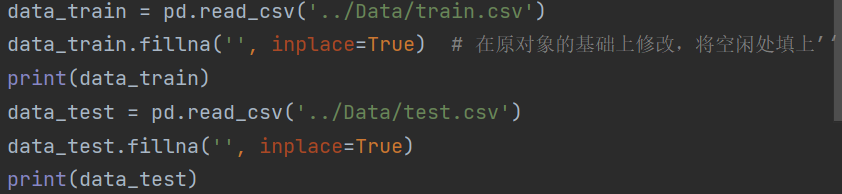


图2：数据读入处理

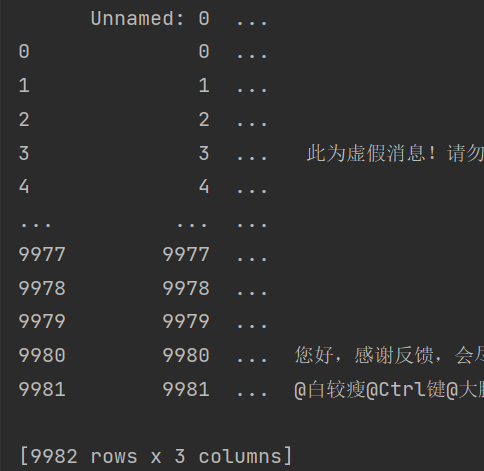
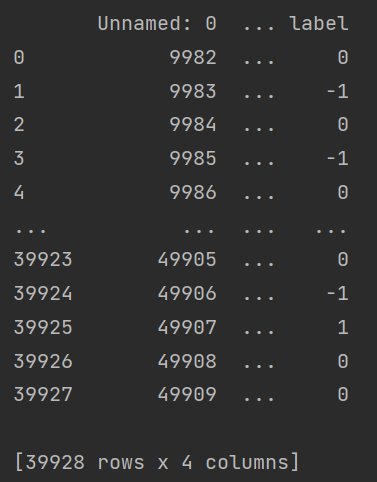
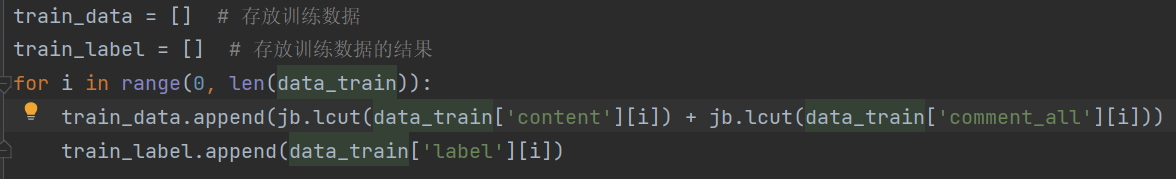


图3：train.scv（左）和test.csv（右）里面数据读取结果

2.从文件中将所要的数据分别提取出来（即train\_data,train\_label,test\_data):使用jieba库里面的lcut方法，将每一条文本和comment\_all进行分词，该方法返回一个将文本切分好的列表。具体实现过程如图4所示。



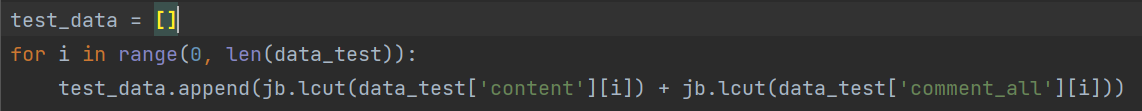


图4：使用jieba分词将文本进行划分

3.清洗数据：使用hit\_stopwords停用词表，将文本中的影响实际结果但实际上却不能判断新闻真假的词语去除。

首先open hit\_stopwords.txt文件，使用read方法读入停词后，再使用splitlines方法将所有停词存在一个列表中。如图5所示。

然后遍历训练数据train\_data和测试数据test\_data，如果包含停词列表中的词语，则在训练数据和测试数据中将该词删除，具体实现过程如图6所示，流程图如图7所示。

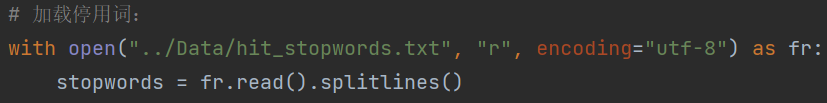
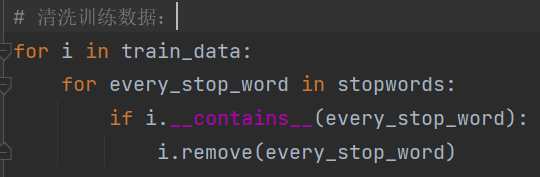


图5：处理文件得到停用词列表



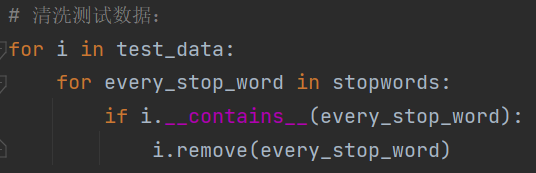


图6：清洗训练数据和测试数据



图7：清洗数据流程图

4.模型使用1：CountVectorizer + MultinomialNB

使用sklearn中的CountVectorizer类来提取文本特征，对于每一个训练的数据，他只考虑该种词汇在训练文本中出现的频率。

创建该类的实例对象，使用fit\_transform()方法来将文本中的词语转化为词频矩阵，该方法的参数是形式化列表，列表元素为代表一篇文章的字符串，字符串是已经分割好的。

假设转化后的词频矩阵定义为a[i][j],那么其值则表示j词在第i个文本下的词频，可以通过get\_feature\_names()方法看到所有文本的关键字，通过toarray()方法可以看到词频矩阵的结果。

然后使用多项式分布的朴素贝叶斯，训练样本，然后进行结果预测。具体实现过程如图8所示：

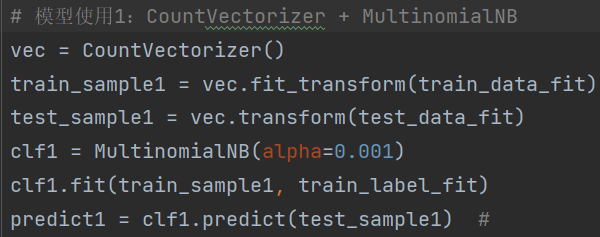


图8：模型1的具体实现

5.模型使用2：使用TfidfTransformer + MultinomialNB

TfidfTransformer是另一种构建词向量的方式，使用方式与CountVectorizer大致相同，这里不做过多的赘述，具体实现过程如图9所示：

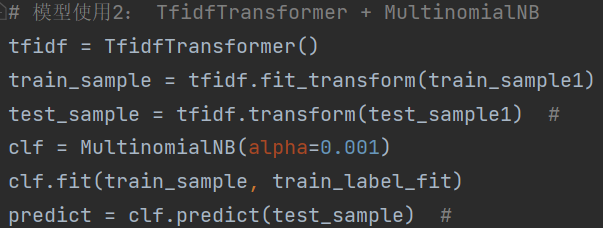


图9：模型2的具体实现

## 1.4结果分析

1.首先，使用numpy中的savetxt方法将模型1的结果保存在result1.txt中，将模型2的结果保存在result.txt中，如下图10所示：

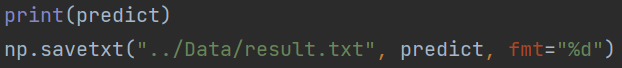


图10：保存在相应文件中

2.将不同模型测试的结果提交到educoder中进行测试，得到的测试结果如图11所示：

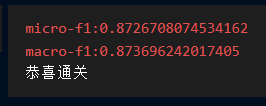
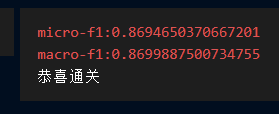


图11：结果CountVectorizer （左），TfidfTransformer （右）

从图中可以看出，两种不同的词向量构建方式对结果有一定的影响，二者的区别在于：使用CountVectorizer对原始文本数据进行处理的时候，直接忽略词汇的长度小于2的的单词，不计入统计范围；而TfidfTransformer不会删除长度小于2 的词汇。TfidfTransformer的评价指标略高于CountVectorizer，由此可见，某些长度为1的单词对于虚假新闻的检测具有一定的意义。

3.模型评估：F1 score中的Micro和Macro

对某问题的预测结果共有下图12几种情况，准确率和召回率的公式如图13所示，由此可知，Precision表示的是我的预测为正例的样本中我答对的比例，Recall则表示实际中正例样本中我答对的比例。

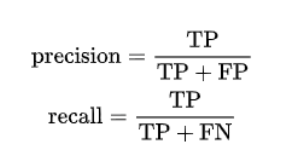
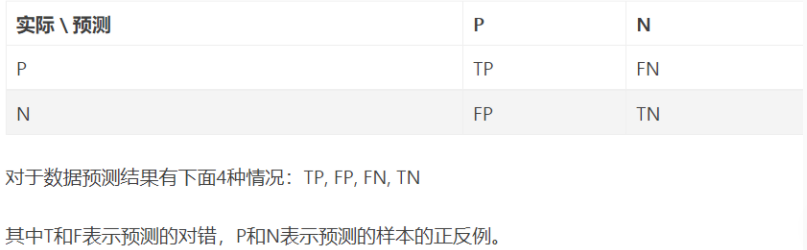


图12：数据结果预测的情况 图13：准确率和召回率公式

F1 score是一个权衡Precision和Recall 的指标，他表示为这两个值的调和平均。当任务为多分类任务时，precision和recall的计算方式就需要权衡每一类的precision\_i和recall\_i，Micro和Macro就是两种不同的权衡方式。

计算方法：将所有类别的Precision和Recall求平均，然后计算F1值作为macro-F1；先计算所有类别的总的Precision和Recall，然后计算出来的F1值即为micro-F1。

根据测试结果，最高的micro-f1为0.8726708074534162，最高的macro-f1为0.873696242017405，结果较为可观。

## 1.5思考与总结

实验中遇到的问题以及思考：

1.停词的另一种去除方法直接在CountVectorizer参数列表里面给出，可以达到和遍历数据去除停用词同样的效果。如下图14：

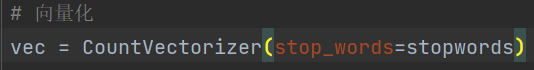


图14：停用词去除

2.fit\_transformer和transformer的使用方式：（重要）

fit(): 就是求得训练集X的均值，方差，最大值，最小值,这些训练集X固有的属性。

transform(): 在fit的基础上，进行标准化，降维，归一化等操作。

fit\_transform(): fit\_transform是fit和transform的组合，既包括了训练又包含了转换。

transform()和fit\_transform()二者的功能都是对数据进行某种统一处理，这里同意非常重要，如果对训练数据和测试数据都使用fit\_transformer方法，将会按照不同的标准对二者数据进行训练，导致控制台报错（会出现文本特征值的数目不匹配的错误）。

因此，正确的使用方式是：fit\_transform(trainData)对部分数据先拟合fit，找到该part的整体指标，如均值、方差、最大值最小值等等，然后对该trainData进行转换transform，从而实现数据的标准化、归一化等等。

3.fit\_transformer与transformer里面的参数的使用问题：

查阅官方文档，得知该参数是形式化的一维的list，而我处理之后的训练数据和测试数据均为list嵌套list，因此，需要将嵌套的每一个list转化为字符串，然后将这些字符串组成一个一维的list，具体实现如下图15所示：

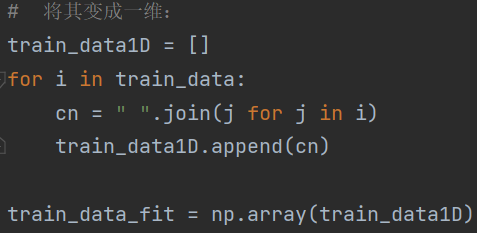


图15：参数形式化

4.朴素贝叶斯的选择：

在sklearn中，一共有3个朴素贝叶斯的分类算法，分别是GaussianNB,MultinomialNB和BernoulliNB，经查阅资料，得知：如果样本特征的分布大部分是连续值，使用GaussianNB会比较好；如果样本特征的分布大部分是多元离散值，使用MultinomialNB会比较好；如果样本特征的分布是二元离散值或者很稀疏的多元离散值，应该使用BernoulliNB。

而此次虚假新闻检测，样本特征类型更贴近于多元离散值，因此选择MultinomialNB。

5.朴素贝叶斯的优缺点：

优点：生成式模型，通过计算概率来进行分类，可以用来处理多分类问题，

对小规模的数据表现很好，算法简单，适合增量式训练。

缺点：对输入数据的表达形式很敏感，由于朴素贝叶斯的朴素特点，所以会带来精确率上的损失，同时还需要计算先验概率，分类决策也存在一定的错误率。

6.感想：此次机器学习实验，对于初学者的我来说既友好又不是太友好，首先，友好在于，对于虚假新闻检测这样的三分类问题，可以使用简单的朴素贝叶斯模型，或者支持向量机模型等等就可以得到一个相对可观的结果，但是对于现阶段的我来说，想要进一步提高准确率是比较困难的，因为自己所掌握的并没有超出课堂范围多少，自己之前也并没有深入了解过机器学习。

其次，不友好的点在于自己之前并没有学习过python，对于各种库的使用较为生涩，因此在实验的过程中，反复查找了大量的官方文档，花费了大量的时间，而且有的时候选择的方法并不能得到我想要的结果，因此，这个过程是非常痛苦的。但是，当最后经过反复测试，终于实现了某种功能的时候，自己还是非常有成就感的，自己也对python语言的掌握得到了加强。

总的来说，我对此次实验是非常满意的，该实验适用范围面较广，初学者可以通过实践掌握更多机器学习知识，而老手则可以对模型做进一步的优化，得到更高的准确率。同时，学习知识并能真正解决日常生活中离我们很近的问题是一个非常快乐的过程。最后，对于整个机器学习课程的设置，我希望可以增加实验课时，对于逻辑回归等相应知识也设置一些实验，强化我们的学习。

附录：

import pandas as pd  
import jieba as jb  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.feature\_extraction.text import TfidfTransformer  
import numpy as np  
# 加载停用词：  
with open("../Data/hit\_stopwords.txt", "r", encoding="utf-8") as fr:  
 stopwords = fr.read().splitlines()  
print(stopwords)  
data\_train = pd.read\_csv('../Data/train.csv')  
data\_train.fillna('', inplace=True) # 在原对象的基础上修改，将空闲处填上’‘  
print(data\_train)  
data\_test = pd.read\_csv('../Data/test.csv')  
data\_test.fillna('', inplace=True)  
print(data\_test)  
train\_data = [] # 存放训练数据  
train\_label = [] # 存放训练数据的结果  
for i in range(0, len(data\_train)):  
 train\_data.append(jb.lcut(data\_train['content'][i]) + jb.lcut(data\_train['comment\_all'][i]))  
 train\_label.append(data\_train['label'][i])  
# 清洗训练数据：  
for i in train\_data:  
 for every\_stop\_word in stopwords:  
 if i.\_\_contains\_\_(every\_stop\_word):  
 i.remove(every\_stop\_word)  
# 将其变成一维：  
train\_data1D = []  
for i in train\_data:  
 cn = " ".join(j for j in i)  
 train\_data1D.append(cn)  
train\_data\_fit = np.array(train\_data1D)  
train\_label\_fit = np.array(train\_label)  
test\_data = []  
for i in range(0, len(data\_test)):  
 test\_data.append(jb.lcut(data\_test['content'][i]) + jb.lcut(data\_test['comment\_all'][i]))  
# 清洗测试数据：  
for i in test\_data:  
 for every\_stop\_word in stopwords:  
 if i.\_\_contains\_\_(every\_stop\_word):  
 i.remove(every\_stop\_word)  
# 将二维转换为一维  
test\_data1D = []  
for i in test\_data:  
 cn = " ".join(j for j in i)  
 test\_data1D.append(cn)  
test\_data\_fit = np.array(test\_data1D)  
# 模型使用1：CountVectorizer + MultinomialNB  
vec = CountVectorizer()  
train\_sample1 = vec.fit\_transform(train\_data\_fit)  
test\_sample1 = vec.transform(test\_data\_fit)  
clf1 = MultinomialNB(alpha=0.001)  
clf1.fit(train\_sample1, train\_label\_fit)  
predict1 = clf1.predict(test\_sample1) #  
print(predict1)  
np.savetxt("../Data/result1.txt", predict1, fmt="%d")  
# 查看所有文本的关键字，显示词频矩阵  
# print(vec.get\_feature\_names())  
# print(train\_sample.toarray())  
# 模型使用2： TfidfTransformer + MultinomialNB  
tfidf = TfidfTransformer()  
train\_sample = tfidf.fit\_transform(train\_sample1)  
test\_sample = tfidf.transform(test\_sample1) #  
clf = MultinomialNB(alpha=0.001)  
clf.fit(train\_sample, train\_label\_fit)  
predict = clf.predict(test\_sample) #  
print(predict)  
np.savetxt("../Data/result.txt", predict, fmt="%d")