西安电子科技大学

安全前沿讨论班（Ⅰ）课程实验报告

实验名称 基于朴素贝叶斯的垃圾短信过滤

学　院 网络与信息安全学院　　班 级 1618019

姓　名 曹寅峰　　学 号 16020610025

实验日期 2019 年 6 月 4 日

指导教师评语：   指导教师： 年 月 日

实验报告内容基本要求及参考格式

一、实验目的

通过垃圾短信过滤案例，加深对 pandas 库，matplotlib 库，sklearn 库的了解，掌握这些库中常见的数据分析方法，理解数据分析的基本步骤。

1. 任务 1：运行并完成实验中的所有代码；
2. 任务 2：对实验中的所有库，函数，方法和代码做笔记，进行注解。

二、实验环境

Python3.5 jupyter

三、实验基本原理及步骤（或方案设计及理论计算）

1. 朴素贝叶斯算法的内涵；

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。而朴素贝叶斯分类是贝叶斯分类中最简单，也是常见的一种分类方法，它基于多种假设条件，使得其易于使用，准确率高。

算法主要核心是贝叶斯公式

$$p(A \mid B) = P(A)\frac{P(B \mid A)}{P(B)}$$

这里左式代表**后验概率（在 B 的条件下 A 的概率）**，P(A)代表**先验概率（A 独自发生的概率）**，分式代表**可能因子（现有结果对先验概率的增强）**

而右式分母不变，是固定值，往往可以根据训练集得出，而分子可以转化成若干个条件概率相乘。这就是朴素的含义，它假设了各个条件相互独立（实际上并不是这样）

如果能掌握一个事情的全部信息，我当然能计算出一个客观概率（古典概率、正向概率）。

可是生活中绝大多数决策面临的信息都是不全的，我们手中只有有限的信息。既然无法得到全面的信息，我们就在信息有限的情况下，尽可能做出一个好的预测。也就是，在主观判断的基础上，可以先估计一个值（先验概率），然后根据观察的新信息不断修正(可能性函数)。

一般来说，可以分成三个步骤解题：

第 1 步. 分解问题

1）要求解的问题是什么？

识别出哪个是贝叶斯中的事件 A（一般是想要知道的问题），哪个是事件 B（一般是新的信息，或者实验结果）

2）已知条件是什么？

第 2 步.应用贝叶斯定理

第 3 步，求贝叶斯公式中的 2 个指标

1）求先验概率

2）求可能性函数

3）带入贝叶斯公式求后验概率

2. 邮件过滤机制；

把邮件过滤代入到上述算法，我们可以得出 c 代表邮件的 type，x,y 代表文件的 text

	type	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

所以对与邮件过滤，我们需要计算 p1(ham|text)和 p2(spam|text) ，其中 text 为某一个单词，选出两者中较大的值作为分类结果。

所以按照上述三个步骤解答问题：(以 p2 为例)

1. 先验概率

在这里先验概率 p(spam)为训练集中的垃圾邮件概率，即 745/(4822+745)

```
]: # 观察变量 type 的分布情况
   sms_raw.groupby('type').count()
```

]:

type	text
0	4822
1	745

2. 可能性函数

可能性函数需要用到全概率公式

计算垃圾邮件中出现 text 的概率 p(text|spam)，和非垃圾邮件中出现 text 的比例 p(text|ham)

则可能性函数为 p(spam)/ p(text|spam)* p(spam)+ p(text|ham)*p(ham)

3. 后验概率

代入贝叶斯公式

p(spam|text)=p(spam)* p(text|spam)/ p(text|spam)* p(spam)+ p(text|ham)*p(ham)

3. 朴素贝叶斯算法的独立性假设；

朴素贝叶斯的关在之处在于它假设各个条件的独立性，例如垃圾邮件中 free，discount 等等单词的出现是相互独立的（实际上不是），这样做可以大大减少运算难度。

比如说实际上一封邮件会包含很多个单词 w1,w2…wn

那么我们需要计算 p(w1,w2…wn|spam)，这样需要训练集中**出现同时包含这些单词的邮件**，这样的邮件是很少的，所以朴素贝叶斯假设各个单词的出现两两独立，如果是独立的，那么

p(w1,w2…wn|spam)=p(w1|spam)* p(w2|spam)*…p(wn|spam)

这样的运算，训练集可以满足，速度很快。当然，如果没有某一个单词，不会把它计算为 0，而是设成一个较小值，符合现实情况（**训练集没有覆盖完全**）

4. 朴素贝叶斯算法应用于文本分类的三种模型。

目前主要有三种模型伯努利模型，高斯模型，多项式模型

1. 高斯朴素贝叶斯

但如果对于连续数据的处理，也就是特征值可能为连续值，不一定出现在训练集中的情况，我们通常假设连续特征为高斯分布，根据训练集数据求均值和方差获得其高斯分布的概率密度参数。在分类过程中需要求解某个特征值 text 的时候，以其对应的高斯密度来代替其概率值。

2. 多项式朴素贝叶斯

如果是离散的训练数据集，比如垃圾邮件分类问题中。P(text|ci)的计算是通过计数，即特征 text（在垃圾邮件分类中为每个字）在每个类别（垃圾/非垃圾）中出现的的次数和总字数的比较得出。

值得注意的是，在离散样本也就是基于频率的估计中，如果某个特征 text 未在训练集的类别 ci 中出现过，那么 P(text|ci)项为 0 会导致整个估计为 0 而忽略了其他的特征信息。这样的估计显然是不准确的，所以通常需要对于样本进行样本修正保证不会有 0 概率出现。比如采用 laplace 校准，对没类别下所有划分的计数加 1，这样如果训练样本集数量充分大时，并不会对结果产生影响。listone 修正则是加一个 0-1 之间的数。

3. 伯努利朴素贝叶斯

和多元朴素贝叶斯中通过特征出现频率来计算 P(text|ci)不同，伯努利模型只考虑出现不出现的二值问题。

伯努利模型中，条件概率 P(text|ci)的计算方式是：

当特征值 text 为 1 时，P(text|ci)=P(text=1|ci)；

当特征值 text 为 0 时，P(text|ci)=1−P(text=1|ci)；

以垃圾邮件分类为例，P(text|ci)=(类 ci 下包含单词 text 的文件数+1)/(类 ci 下单词总数+2)

实验过程

首先利用 pd 库读取 csv 文件，转换成 dataframe（类似表格的类型），展示出来

```
]: import pandas as pd
   # read the content of the csv file
   # and save it to sms_raw
   sms_raw=pd.read_csv("./sms_spam.csv")
```

```
]: # return the dimensionality of the dataframe sms_raw
   smsdim=pd.DataFrame(sms_raw)
```

```
]: # look through the first 5 rows of sms_raw
   smsdim.head(5)
```

]:

	type	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

可以查看某一列

```
9]: # look for the type of sms_raw
    smsdim['type']
```

```
9]: 0        ham
    1        ham
    2       spam
    3        ham
    4        ham
    5       spam
    6        ham
    7        ham
    8       spam
    9       spam
    10       ham
    11      spam
    12      spam
    13       ham
    14       ham
```

利用 describe 可以查看表格的统计信息

Factorize 方法可以把分类变量转化为因子变量，便于处理，直接把 ham/spam 转化成 0/1

```
1: # your code is here
```

数据预处理

语料库现在包含5567条短信的原始文本内容。在将文本内容分解成单词之前，我们需要进行一些清理步骤以去除标点符号和可能会影响结果的其他字符。例如我们将把单词hello!、HELLO和Hello等都作为单词hello的样本。

首先我们利用 `map` 方法去除文本内容中的标点符号和数字，并将文本中所有的字符串都转换成小写形式：

```
]: def tolower(text):
       return text.lower()
```

```
]: def removePunctuation(text):
       trantab = str.maketrans('','',string.punctuation + string.digits)

       return text.translate(trantab)
```

```
]: sms_raw['text'] = sms_raw['text'].map(removePunctuation).map(tolower)
```

```
]: # look through the first 5 rows of sms_raw
   sms_raw.head(5)
```

在文本分类中，预处理非常重要，这里利用了 map 执行函数 remove 和 tolower，去除了标点

```
12]: pd.Series(sms_raw['type'].factorize())
```

```
12]: 0    [0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, ...
     1               Index(['ham', 'spam'], dtype='object')
     dtype: object
```

当前的特征type是一个字符串向量。由于它是一个分类变量，所以将其转换成一个因子变量将有利于后续的数据处理，其中 1 表示垃圾短信，0 表示非垃圾短信，如下面的代码所示：

```
21]: # factorize函数可以将Series中的标称型数据映射称为一组数字，相同的标称型映射为相同的数字
     sms_raw['type'] = pd.Series(sms_raw['type'].factorize()).iloc[0]
```

```
22]: # 观察变量 type 的分布情况
     sms_raw.groupby('type').count()
```

```
22]:
```

type	text
0	4822
1	745

符号并转化成小写

:

	type	text
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in a wkly comp to win fa cup final...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

最后再去掉 to and 等连词，统计预处理之后的数据。

在分析文本数据时，一个常见的做法就是去除填充词，比如to、and、but和or，这些词称为停用词（stop word）。我们将使用 `skle` `CountVectorizer` 方法来去除停止词并计算文本的词频分布情况：

31]:
```
count_vect = CountVectorizer(stop_words="english",decode_error='ignore')
sms_counts = count_vect.fit_transform(sms_raw['text'])
sms_counts.shape
```

31]: (5567, 8333)

32]:
```
len(count_vect.get_feature_names())
```

32]: 8333

TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率). 是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。引用总结就是，一个词语在一篇文章中出现次数越多，同时在所有文档中出现次数越少，越能够代表该文章.

词频 (term frequency, TF) 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件。(同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否。)

33]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(stop_words="english",decode_error='ignore',min_df=5)
sms_tfidf = tfidf_vect.fit_transform(sms_raw['text'])
sms_tfidf.shape
```

33]: (5567, 1490)

建立训练数据集和测试数据集

由于已经为分析准备好了数据，所以现在我们需要将数据分成训练数据集和测试数据集，从而可以把垃圾短信分类器应用到之前没有学习过的数分类器的性能进行评估。我们将数据分成两部分：75%的训练数据和25%的测试数据。因为短信的排序是随机的，所以我们可以简单地取前41练，剩下的1391条用于测试。

34]:
```
sms_trainx = sms_tfidf[:4176,]
sms_trainy = sms_raw['type'][:4176]
```

35]:
```
sms_testx = sms_tfidf[4176:5567,]
sms_testy = sms_raw['type'][4176:5567]
```

39]:
```
# 查看训练集的数据大小
sms_trainx.shape
```

39]: (4176, 1490)

41]:
```
# 查看测试集的数据大小
sms_testx.shape
```

41]: (1391, 1490)

最后我们还利用了词语作为特征的直观展示，分别是垃圾词云和非垃圾词云

最后调用 mutinomanialNB 进行训练即可

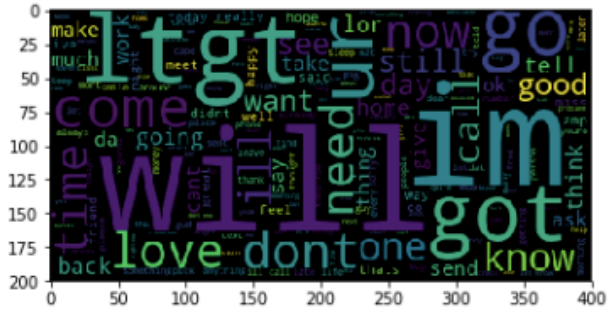
```
plt.imshow(wc)
```

因为我们已经将原始短信转换成了可以用一个统计模型代表的形式，所以此时是应用朴素贝叶斯算法的时候了。该算法将根据单词的存在与否来统计一条给定的短信是否是垃圾短信的概率。

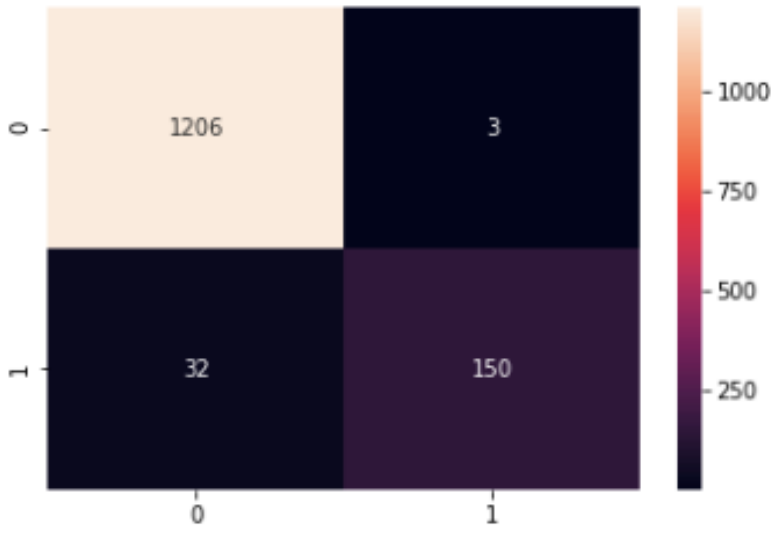
我们采用 sklearn 包中的朴素贝叶斯算法实现。

```
from sklearn.naive_bayes import MultinomialNB
sms_classifier = MultinomialNB().fit(sms_trainx,sms_trainy)
```


```
wc.generate(''.join(sms_raw['text'][sms_raw['type']==0]))
plt.imshow(wc)
plt.show()
```



统计最后的训练结果




```
print(metrics.classification_report(sms_testy,sms_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1209
           1       0.98      0.82      0.90       182

    micro avg       0.97      0.97      0.97      1391
    macro avg       0.98      0.91      0.94      1391
weighted avg       0.97      0.97      0.97      1391
```

可以看到，有三条被误归为垃圾邮件，32 条被误归为非垃圾邮件，朴素贝叶斯模型的精度和召回率都达到 97%，这在我们没有调整任何参数的情况下，性价比相当高。

四、问题与结论

这次实验代码上并没有什么问题，主要是贝叶斯算法的理解上，虽然之前概率论学过相关的内容，但这次实际使用仍然需要重新理解，感谢老师给我发的笔记，感觉已经理解地比较好了