使用sklearn的贝叶斯分类器进行文本分类

1、sklearn简介

sklearn是一个Python第三方提供的非常强力的机器学习库，它包含了从数据预处理到训练模型的各个方面。在实战使用scikit-learn中可以极大的节省我们编写代码的时间以及减少我们的代码量，使我们有更多的精力去分析数据分布，调整模型和修改超参。

2、朴素贝叶斯在文本分类中的常用模型：多项式、伯努利

朴素贝叶斯分类器是一种有监督学习，常见有两种模型，多项式模型(multinomial model)即为词频型和伯努利模(Bernoulli model)即文档型。二者的计算粒度不一样，多项式模型以单词为粒度，伯努利模型以文件为粒度，因此二者的先验概率和类条件概率的计算方法都不同。计算后验概率时，对于一个文档d，多项式模型中，只有在d中出现过的单词，才会参与后验概率计算，伯努利模型中，没有在d中出现，但是在全局单词表中出现的单词，也会参与计算，不过是作为“反方”参与的。这里暂不虑特征抽取、为避免消除测试文档时类条件概率中有为0现象而做的取对数等问题。

2.1、多项式模型

在多项式模型中， 设某文档d=(t1,t2,…,tk)，tk是该文档中出现过的单词，允许重复，则

先验概率P(c)= 类c下单词总数/整个训练样本的单词总数

类条件概率P(tk|c)=(类c下单词tk在各个文档中出现过的次数之和+1)/(类c下单词总数+|V|)

V是训练样本的单词表（即抽取单词，单词出现多次，只算一个），|V|则表示训练样本包含多少种单词。 P(tk|c)可以看作是单词tk在证明d属于类c上提供了多大的证据，而P(c)则可以认为是类别c在整体上占多大比例(有多大可能性)。

2）举例

给定一组分好类的文本训练数据，如下：

docId	doc	类别 In c=China?
1	Chinese Beijing Chinese	yes
2	Chinese Chinese Shanghai	yes
3	Chinese Macao	yes
4	Tokyo Japan Chinese	no

给定一个新样本Chinese Chinese Chinese Tokyo Japan，对其进行分类。该文本用属性向量表示为d=(Chinese, Chinese, Chinese, Tokyo, Japan)，类别集合为Y={yes, no}。

类yes下总共有8个单词，类no下总共有3个单词，训练样本单词总数为11，因此P(yes)=8/11, P(no)=3/11。类条件概率计算如下：

P(Chinese | yes)=(5+1)/(8+6)=6/14=3/7

P(Japan | yes)=P(Tokyo | yes)= (0+1)/(8+6)=1/14

P(Chinese|no)=(1+1)/(3+6)=2/9

P(Japan|no)=P(Tokyo| no) =(1+1)/(3+6)=2/9

分母中的8，是指yes类别下textc的长度，也即训练样本的单词总数，6是指训练样本有Chinese,Beijing,Shanghai, Macao, Tokyo, Japan 共6个单词，3是指no类下共有3个单词。

有了以上类条件概率，开始计算后验概率：

$P(yes \mid d)=(3/7)^3\times1/14\times1/14\times8/11=108/184877\approx0.00058417$

$P(no \mid d)= (2/9)^3\times2/9\times2/9\times3/11=32/216513\approx0.00014780$

比较大小，即可知道这个文档属于类别china。

2.2、伯努利模型

1）基本原理

P(c)= 类c下文件总数/整个训练样本的文件总数

P(tk|c)=(类c下包含单词tk的文件数+1)/(类c下单词总数+2)

2）举例

使用前面例子中的数据，模型换成伯努利模型。

类yes下总共有3个文件，类no下有1个文件，训练样本文件总数为11，因此P(yes)=3/4, P(Chinese | yes)=(3+1)/(3+2)=4/5，条件概率如下：

P(Japan | yes)=P(Tokyo | yes)=(0+1)/(3+2)=1/5

P(Beijing | yes)= P(Macao|yes)= P(Shanghai |yes)=(1+1)/(3+2)=2/5

P(Chinese|no)=(1+1)/(1+2)=2/3

P(Japan|no)=P(Tokyo| no) =(1+1)/(1+2)=2/3

P(Beijing| no)= P(Macao| no)= P(Shanghai | no)=(0+1)/(1+2)=1/3

有了以上类条件概率，开始计算后验概率，

P(yes|d)=P(yes)×P(Chinese|yes)×P(Japan|yes)×P(Tokyo|yes)×(1-P(Beijing|yes))×(1-P(Shanghai|yes))×(1-P(Macao|yes))=3/4×4/5×1/5×1/5×(1-2/5)×(1-2/5)×(1-2/5)=81/15625≈0.005

P(no|d)= 1/4×2/3×2/3×2/3×(1-1/3)×(1-1/3)×(1-1/3)=16/729≈0.022

因此，这个文档不属于类别china。

2.3、两个模型的区别

在多项式模型中：

> 在多项式模型中， 设某文档d=(t1,t2,…,tk)，tk是该文档中出现过的单词，允许重复，则
> 先验概率P(c)= 类c下单词总数/整个训练样本的单词总数
> 类条件概率P(tk|c)=(类c下单词tk在各个文档中出现过的次数之和+1)/(类c下单词总数+|V|)
> V是训练样本的单词表（即抽取单词，单词出现多次，只算一个），|V|则表示训练样本包含多少种单词。
> P(tk|c)可以看作是单词tk在证明d属于类c上提供了多大的证据，而P(c)则可以认为是类别c在整体上占多大比例(有多大可能性)。

在伯努利模型中：

> P(c)= 类c下文件总数/整个训练样本的文件总数
> P(tk|c)=(类c下包含单词tk的文件数+1)/(类c下单词总数+2)

3、实战演练

使用在康奈尔大学下载的2M影评作为训练数据和测试数据，里面共同、共有1400条，好评和差评各自700条，我选择总数的70%作为训练数据，30%作为测试数据，来检测sklearn自带的贝叶斯分类器的分类效果。

> 读取全部数据，并随机打乱

In [5]:

```
import os
import random
def get_dataset():
    data = []
    for root, dirs, files in os.walk('../dataset/aclImdb/neg'):
        for file in files:
            realpath = os.path.join(root, file)
            with open(realpath, errors='ignore') as f:
                data.append((f.read(), 'bad'))
    for root, dirs, files in os.walk(r'../dataset/aclImdb/pos'):
        for file in files:
            realpath = os.path.join(root, file)
            with open(realpath, errors='ignore') as f:
                data.append((f.read(), 'good'))
    random.shuffle(data)

    return data
```

In [9]:

```
data = get_dataset()
data[:2]
```

Out[9]:

```
[("An awful film! It must have been up against some real stinkers to be nominated for the G
olden Globe. They've taken the story of the first famous female Renaissance painter and m
angled it beyond recognition. My complaint is not that they've taken liberties with the fact
s; if the story were good, that would perfectly fine. But it's simply bizarre -- by all accounts t
he true story of this artist would have made for a far better film, so why did they come up
with this dishwater-dull script? I suppose there weren't enough naked people in the factual
version. It's hurriedly capped off in the end with a summary of the artist's life -- we could h
ave saved ourselves a couple of hours if they'd favored the rest of the film with same brevit
y.",
  'bad'),
 ("I used to LOVE this movie as a kid but, seeing it again 20+ years later, it actually sucks. U
p The Academy might have been ahead of it's time back in 1980, but it has almost nothing t
o offer today! Movies like Caddyshack and Stripes hold-up much better today than this stea
ming dogpile. No T&A. No great jokes except for the one-liners we've all heard a million tim
es by now.<br /><br />I recently bought the DVD in hopes that it would be the gem I reme
mbered it being. Well, I was WAY off! The soundtrack had only 2-3 widely-recognizable hits
 (not the smash compilation others had mentioned) and the frequent voice-overs were terr
ible. The only thing that was interesting, to me, was predicting what the character's lines w
ere before they said them. Yep, I watched this movie that much back then! <br /><br />The
 only reason I am writing this review is to give my two cents on why this movie should be fo
rgotten, sorry to say. :(",
  'bad')]
```

按照7:3的比例划分训练集和测试集

In [10]:

```
def train_and_test_data(data_):
    filesize = int(0.7 * len(data_))
    # 训练集和测试集的比例为7:3
    train_data_ = [each[0] for each in data_[:filesize]]
    train_target_ = [each[1] for each in data_[:filesize]]

    test_data_ = [each[0] for each in data_[filesize:]]
    test_target_ = [each[1] for each in data_[filesize:]]

    return train_data_, train_target_, test_data_, test_target_
```

In [13]:

```
train_data, train_target, test_data, test_target = train_and_test_data(data)
```

使用多项式贝叶斯分类器

In [18]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer, HashingVectorizer, CountVectorizer
from sklearn import metrics
from sklearn.naive_bayes import BernoulliNB

nbc = Pipeline([
   ('vect', TfidfVectorizer()),
   ('clf', MultinomialNB(alpha=1.0)),
])
nbc.fit(train_data, train_target)    #训练我们的多项式模型贝叶斯分类器
predict = nbc.predict(test_data)  #在测试集上预测结果
count = 0                         #统计预测正确的结果个数
for left , right in zip(predict, test_target):
    if left == right:
        count += 1
print(count/len(test_target))
```

0.8835274542429284

使用伯努利模型分类器

In [21]:

```
nbc_1= Pipeline([
   ('vect', TfidfVectorizer()),
   ('clf', BernoulliNB(alpha=0.1)),
])
nbc_1.fit(train_data, train_target)
predict = nbc_1.predict(test_data)  #在测试集上预测结果
count = 0                         #统计预测正确的结果个数
for left , right in zip(predict, test_target):
    if left == right:
        count += 1
print(count/len(test_target))
```

0.8818635607321131

从分类结果可以看出，和多项式模型相比，使用伯努利模型的贝叶斯分类器，在文本分类方面的精度相比，差别不大，我们可以针对我们面对的具体问题，进行实验，选择最为合适的分类器。

作业

sklearn中一共提供了四种贝叶斯分类器：

- 高斯朴素贝叶斯
- 多项式朴素贝叶斯
- 补充朴素贝叶斯
- 伯努利朴素贝叶斯

从四种贝叶斯分类器模型中找出具有最佳分类效果的分类器，并用直方图直观表示其分类准确率。

参考资料

sklearn官方网站 (https://scikit-learn.org/stable/index.html)
https://scikit-learn.org/stable/index.html (https://scikit-learn.org/stable/index.html)

sklearn:朴素贝叶斯 (https://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes)
https://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes (https://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes)