Dating Mining

Experiment 2 Naïve Bayes Model

姓名：包琛

学号：201814800

班级：2018级计算机学硕班

指导老师：尹建华

时间：2018年11月25日

1 实验要求

- 实现朴素贝叶斯分类器，测试其在20 Newsgroups数据集上的效果。

数据集：http://qwone.com/~jason/20Newsgroups/

2 实验背景

2.1 Naïve Bayes Model

若一个样本有n个特征，分别用x1,x2,…,xn表示，将其划分到类yk的可能性P(yk|x1,x2,…,xn)为：

P(yk|x1,x2,…,xn)=P(yk)∏P(xi|yk)

上式中等号右侧的各个值可以通过训练得到。根据上面的公式可以求的某个数据属于各个分类的可能性（这些可能性之和不一定是1），该数据应该属于具有最大可能性的分类中。

朴素贝叶斯的两种模型：多项式模型和伯努利模型。

（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)。

（1） 伯努利模型：

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 实验内容

1.1 实验流程

数据集包含20个类的共18828篇文档。本次实验中使用的是多项式模型。

1. 首先，要分别取每个类的80%的文档作为训练集，每个类的20%作为测试集。
2. 对文档进行预处理，预处理之后可以得到每篇文档的关键词。我使用了NLTK这个Python库。预处理主要进行了以下几步：
   (a)分句并分词
   (b)标注词性标签，且只保留名词（标签为NN的词）
   (c)全部转化为小写字母
   (d)去掉标点，特殊符号和数字
   (e)词干还原
3. 计算文档的先验概率，类c下单词总数/整个训练样本的单词总数。经测试，正确率比直接使用类c下文档总数/整个训练集的文档总数要高。
4. 因为选用的是多项式模型，因此需要统计不同类别下每个单词出现的次数，以及每篇文档的总单词数（计重复单词)。
5. 平滑处理：需要建立一个全部训练集的词典，以统计整个训练集中不重复的词的个数。即|V|,用来做平滑处理。
6. 构建朴素贝叶斯多项式模型，输入测试集进行分类。
7. 将测试集的真实类别与算法测出的类别相比较，计算正确率。

3.2 实验环境

处理器：Intel(R) Core(TM) i7-8700K CPU @ 3.70Ghz 3.70Ghz
RAM：16.0 GB
系统： Windows 10，64位
编程语言：Python 3.7
IDE: PyCharm

1.2 核心代码及注释

1. 预处理 和上一个实验区别不大，去掉计算 tf 和 idf 的部分即可。

```
#stopwords
from nltk.corpus import stopwords
stop=stopwords.words('english')

#the stemming method
from nltk.stem import WordNetLemmatizer
lemma=WordNetLemmatizer()

#the tokenization method
def splitIntoWords(article):
    documentWords=[]
    tokenizer=nltk.data.load('english.pickle')
    for paragraph in article:
        sentences = tokenizer.tokenize(paragraph) #first split into sentences

        for sentence in sentences:
            words=nltk.word_tokenize(sentence) #then split each sentence into words
            documentWords+=words
    return documentWords

# the pre-processing method
def preProcessing(document):
    processedDocument=[]
    iter_d=iter(document)
    for word in iter_d:
        #lowerWord=word
        lowerWord=word.lower()# change to lowercase
        word=""
        for char in lowerWord:# delete punctuation #delete numbers
            if (char not in string.punctuation) and (char not in string.digits):
                word+=char
        lowerWord=word

        #print(lowerWord)
        if len(lowerWord)==0: #the word is deleted because it only contains punctuation or numbers
```

```
            continue
        else:
            lowerWord = lemma.lemmatize(lowerWord)    #
stemming
        if lowerWord in stop: # delete stop words
            continue
        elif len(lowerWord)<2: # delete words whose length is
lower than 2
            continue
        else:
            processedDocument.append(lowerWord)
    return processedDocument

def docProcess(path,fileList,flag):
    files = os.listdir(path)    # obtain all file names under path
[atl.atheism,comp.graphics,...]
    documents = []
    document_count=0
    category_index=0

    for file in files:
        documents=os.listdir(path+"/"+file)
        countTraining=int(len(documents)*0.8)# take the first 80%
documents as the training set
        documentList=[] # the list of documents within one single
file
        if flag==0:
            documentRange=range(countTraining) #training
        else:
            documentRange=range(countTraining,len(documents))

        for i in documentRange:
            document_count+=1
            currentDoc = open(path + "/" +
file+"/"+documents[i],encoding='ISO-8859-1')

            wordList=splitIntoWords(currentDoc.readlines())
#tokenization

            text = nltk.Text(wordList)
            tags = nltk.pos_tag(text)
            wordVec = []
            for tag in tags:
                if "NN" in tag[1]:
                wordVec.append(tag[0])
            #print(wordVec)

            #print(wordList)
            processedDocument=preProcessing(wordVec)
#preprocessing

            documentLength = len(processedDocument) #number
of words in one document
            count=Counter(processedDocument) #count the
number of each word
            #print(count.most_common(10))


processedDocument=[category_index]+processedDocument

            documentList.append(processedDocument)

        fileList.append(documentList)
        category_index+=1

    return document_count
```

2. 计算类条件概率和先验概率

```
#calculate the conditional probability of each word in each category
wordCountPath = "wordCount.txt"
wordCount=[] # the number of words of all documents in a single
category
ff=open(wordCountPath,'w',encoding='ISO-8859-1')

totalLength=0

for i in range(20):
    filePath="processedDoc/processedDoc_"+str(i)+".txt"
    f1=open(filePath,encoding='ISO-8859-1')
    documents=f1.readlines()
    f1.close()

    docVec=[] # the list of the split documents
    wordCount.append(0)
    for document in documents:
        document=document.strip("\n").split()
        docVec.append(document[1:])
        wordCount[i]+=len(document)-1
    totalLength+=wordCount[i]

    proDic={} # the dictionary of a category
    for wordVec in docVec:
        for word in wordVec:
            if word not in proDic:
                proDic[word]=1
            else:
                proDic[word]+=1

    ff.write("%d\n" % wordCount[i])

    # output the number of each word in one single category
    proPath="probability/probability_"+str(i)+".txt"
    f2=open(proPath,'w',encoding="ISO-8859-1")
    probability={}
    for word in proDic:
        if proDic[word]>=5:
            probability[word]=proDic[word]
            f2.write("%s %f\n" % (word,probability[word]))
    f2.close()
ff.close()

# compute prior probability of each category
priorProb=[]
for i in range(20):
    categoryPath = "cateProbability.txt"
    f1 = open(categoryPath, 'w', encoding="ISO-8859-1")
    priorProb.append(wordCount[i]/totalLength)
    for cate in priorProb:
        f1.write("%f\n" % cate)
#print(priorProb)
```

3. 贝叶斯公式部分

```
import os
import math

def takeSecond(elem):
    return elem[1]

f1=open("test_processedDoc.txt",encoding='ISO-8859-1')
```

```
processedTest=f1.readlines() # the processed testing document that were split in to words

f2=open("cateProbability.txt",encoding='ISO-8859-1')
cateProbability=f2.readlines() # the prior probability of each category
newCate=[]
for cate in cateProbability:
    newCate.append(float(cate.strip('\n')))
cateProbability=newCate

f3=open("wordCount.txt",encoding='ISO-8859-1')
wordCount=f3.readlines() # the number of words in each category
newcount=[]
for count in wordCount:
    newcount.append(int(count.strip("\n")))
wordCount=newcount

##################################
filePath="D:/Coding/Data Mining/NBM/probability"
dictionary={}
probList=[] # the number of each word in each category of the training data. (word,categoryFrequency)
for i in range(20):
    f4=open(filePath+"/probability_"+str(i)+".txt",encoding='ISO-8859-1')
    probabilities=f4.readlines()
    docProb={}
    for probability in probabilities:
        probability=probability.strip("\n").split()
        if probability[0] not in dictionary:
            dictionary[probability[0]]=1
            #print(probability[0])
        docProb[probability[0]]=float(probability[1])
    probList.append(docProb)
    f4.close()
#print(len(dictionary))
##################################
classifiedCate=[]
initialCate=[]
f5=open("test_result.txt","w")
for document in processedTest:
    document=document.strip("\n").split()
    #print(document)
    documentProb=[]
    initialCate.append(int(document[0]))
    for i in range(len(probList)):
        P_doc = 0
        for word in document[1:]:
            if word in probList[i]:

P_doc+=math.log((probList[i][word]+1)/(wordCount[i]+len(dictionar
y)))
            else:

P_doc+=math.log(1/(wordCount[i]+len(dictionary)))
            #print(P_doc)
        #print("HELLO")
        P_doc=P_doc+math.log(cateProbability[i])
        documentProb.append((i,P_doc))
    documentProb.sort(key=takeSecond,reverse=True)
    classifiedCate.append(documentProb[0][0])
    #print(documentProb[0][0])
    f5.write("%d\n" % documentProb[0][0])
f5.close()

accurate=0
for i in range(len(initialCate)):
    if initialCate[i]==classifiedCate[i]:
        accurate+=1
print(accurate/len(initialCate))
```

4 实验结果

正确率 81.9%

5 分析与讨论

此次实验的结果比上次使用 KNN 进行分类好了许多，这给了我很大鼓舞。而且使用贝叶斯分类器，即使处理像 20news 这样规模比较大的数据，也非常迅速，可以快速得到结果，不必像上次实验那样等待超过 5 个小时。

实际代码过程中要注意的地方：

1. 平滑问题，上文已经进行了描述。
2. 使用 log 函数进行贝叶斯公式的计算。由于数据规模较大，实际操作时如果使用概率相

乘，结果会因为数据太小而向下溢出变为 0。为了避免这个问题，对每一项取 log，将相乘转化为相加：

$$logC = logP(\text{“我”}|S) + logP(\text{“司”}|S) + logP(\text{“可”}|S) + logP(\text{“办理”}|S)$$