DM Homework Report

----NBC 算法的实现

姓名：　曹　瑞　

学号：　201834856

一、实验要求

（一）预处理文本数据集，并且得到每个文本的VSM表示。

（二）实现NBC分类器，测试其在20Newsgroups上的效果。

二、程序设计思路

（一）数据的预处理

1. 数据集的产生

将每一类数据前75%的文件作为训练数据，后25%的数据作为完全测试数据集。训练数据用来构建词典以及建立模型。在训练数据中，将每一类数据前50%的文件作为构建VSM的数据，后25%的数据作为训练数据中的预测试数据。进行词典建立的时候我们预设词典的大小为500维，即选取500个最具代表力的单词来构建词典。

2. 数据类别过滤

本次实验采用Stanford CoreNLP作为分词工具，并通过其对分词的标签功能进行单词类别的过滤。试验中主要考虑动词及其各种时态，名词，地名，书名等与有代表性的词语，其余的单词，例如物主代词，介词，副词等均不列入考虑范围，都需要过滤掉。

（二）算法实现思路

1. 词典的建立

扫描每一类下前75%的文件，对于每个文件中出现的单词，计算其在当前文档中出现的频率TF，以及在其他文档中出现的次数IDF，通过TF-IDF衡量单词的好坏，最终选取500个最具分类能力的单词构建词典。其中IDF算法采用如下公式：

$$IDF(\mathrm{w}) = \log\left(\frac{D}{1+D_w}\right)$$

其中D表示文档总数，D_w表示文档中出现单词w的文档数。

2. 根据词典描述预测试数据

根据建立出来的词典，先将每一类前50%的每一个文件描述为一个500维的向量。其次将每一类50%-75%的文件进行预测试分类，分别计算每一类每一个属性上的均值与方差。其中概率计算采用如下公式：

$$\mathrm{p}(x_i \mid c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

其中$\mu_{c,i}$表示均值，$\sigma_{c,i}$表示方差。

三、实验过程

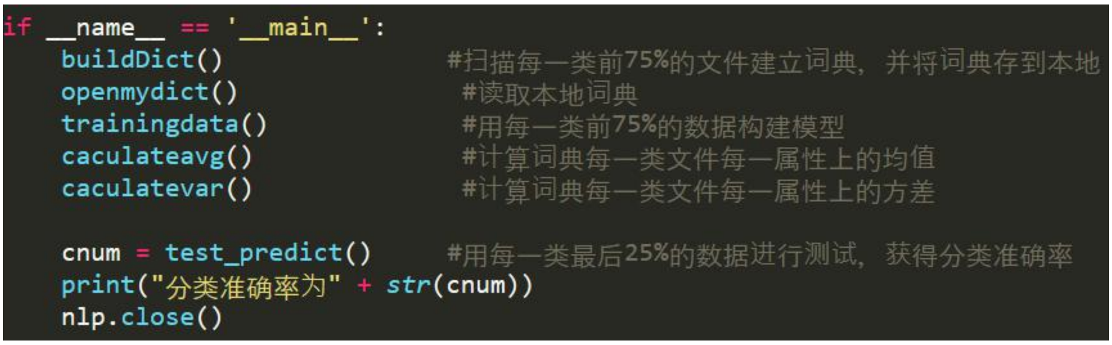
在试验过程中遇到了很多问题，主要是两方面的问题，一个是实验的设计上，一个是算法的实现上。

实验的设计方面，由于一开始将每个词都列入了考虑范围，导致词典规模太大，后来在和同学的交流下，采取了词频过滤与词类过滤结合的方法，既将低频词过滤掉，又将分类中效果不好的词类过滤掉，大大减小了词典规模。其次一开始实验只计算了词频，所以分类效果非常不好，后来发现是自己对于 TF-IDF 的原理没有弄清楚，所以重新编写了词典建立的函数。

算法实现上，因为以前没接触过 python，所以在文件读取时遇到了一些问题，对于文件的编码有了更深的认识，同时对于 python 语言对格式要求的严格性也有了体会。

四、代码说明

此处仅对 main 函数进行说明，具体的代码详见附录或者源代码



```
if __name__ == '__main__':
    buildDict()                #扫描每一类前75%的文件建立词典，并将词典存到本地
    openmydict()                #读取本地词典
    trainingdata()              #用每一类前75%的数据构建模型
    caculateavg()               #计算词典每一类文件每一属性上的均值
    caculatevar()               #计算词典每一类文件每一属性上的方差

    cnum = test_predict()      #用每一类最后25%的数据进行测试，获得分类准确率
    print("分类准确率为" + str(cnum))
    nlp.close()
```

附录

```
from stanfordcorenlp import StanfordCoreNLP
import os
import math
nlp = StanfordCoreNLP(r'D:\000-software\corenlp\stanford-corenlp-full-2018-10-05') #corenlp包存放路径

valuewords = ["NN","NNS","NNP","NNPS"]   #有效单词类别
words = []                               #记录候选词典
wordscount = []                          #记录候选词典里每个单词的 TF-IDF
dictwords = []                           #保存词典
trainingtags = []                        #训练数据的类别-500
trainingvector = []                      #训练数据的 VSM-500
avg = []                                 #训练数据每一类的均值信息
var = []                                 #训练数据每一类的方差信息
path = "20news-18828"                    #data路径


def getSum(path):
    ##BEGIN
    ##统计path路径中文件的个数并返回
    allFileNum = 0
    files = os.listdir(path)
    for f in files:
        if(os.path.isfile(path+'/'+f)):
            allFileNum = allFileNum + 1
    return allFileNum
    ##END


def orderwords():
    ##BEGIN
    ##将词典中单词按照TF-IDF降序进行排序
    print("开始排序\n")
    for r1 in range(0,200):
        for r2 in range(r1+1,len(wordscount)):
            if(wordscount[r1] < wordscount[r2]):
                wordscount[r1],wordscount[r2] = wordscount[r2],wordscount[r1]
                words[r1],words[r2] = words[r2],words[r1]
    print("结束排序\n")
    ##END

def SaveDict():
    ##BEGIN
    ##选前500个TF-IDF最高的,将字典内容保存为一个txt, 命名为mydict
    filename = 'mydict.txt'
    orderwords()
    len = 0
    print("开始保存\n")
    with open(filename,'a',encoding='gb18030') as p:
        for q1 in words:
            if(len<200):
                p.write(q1)
                p.write("\n")
                len = len + 1
            else:
                pass
        len = 0
        for q2 in wordscount:
            if(len<200):
                p.write(str(q2))
                p.write("\n")
                len = len + 1
            else :
                pass
    print("结束保存\n")
    ##END

def haveword(d,j):
    ##BEGIN
    ##扫描文档d是否含有单词j, 返回1代表含有, 0代表不含有
    try:
        f = open(d,encoding='gb18030',errors='ignore')
        for line in f.readlines():
            types = nlp.pos_tag(line.strip())
            for t in types:
                if(j==t[0]):
                    return 1
                else:
                    pass
        return 0
    finally:
        if f:
            f.close()
        ##END


def getidf(j):
        ##BEGIN
        ##计算单词 j 的 idf 并返回,总共用来构建词典的文件数为 14121
        jc = 0
        nd = 14121
        t = []
        filelist = os.listdir(path)
        for fi in filelist:
                if(os.path.isdir(path+'\\'+fi)):
                        t.append(path+'\\'+fi)
        for k in t:
                count = 0
                filesum = getSum(k)
                everyfile = []
                fh = os.listdir(k)
                for m in fh:
                        if(os.path.isfile(k+'\\'+m)):
                                everyfile.append(k+'\\'+m)
                for d in everyfile:
                        if(count < int(filesum*0.75)):
                                jc = jc + haveword(d,j)
                                count = count + 1
                        else:
                                pass
        jc = jc + 1
        return math.log10(nd/jc)

        ##END

def analyzefile(d):
        ##BEGIN
        ##统计当前文件中的每一行, 进行词频统计与记录, 其中 words 记录单词, wordscount 记录 tf-idf
        types = []
        tempwords = []
        tempwordscount = []
        wordsum = 0
        try:
                f = open(d,encoding='gb18030',errors='ignore')
                for line in f.readlines():
                        types = nlp.pos_tag(line.strip())
                        wordsum = wordsum + len(types)
                for t in types:
                    if(t[1] in valuewords and t[0].isalpha() and len(t[0])>2):
                        if(t[0].lower() not in tempwords):
                            tempwords.append(t[0].lower())
                            tempwordscount.append(1)
                        else:
                            tempwordscount[tempwords.index(t[0].lower())] =
tempwordscount[tempwords.index(t[0].lower())] + 1
                    else:
                        pass
            for i in tempwordscount:
                i = i / wordsum          ##得到单词的TF
            l = 0
            for j in tempwords:
                tempwordscount[l] = tempwordscount[l] * getidf(j)     #得到单词的TF-IDF
                l = l + 1
            for s in tempwords:
                if s not in words:
                    words.append(s)
                    wordscount.append(tempwordscount[tempwords.index(s)])
                else:
                    wordscount[words.index(s)] = ( wordscount[words.index(s)] +
tempwordscount[tempwords.index(s)] ) / 2  #如果词典中已经有这个单词，则对TF-IDF取平均
        finally:
            if f:
                f.close()
        ##END


    def buildDict():
        ##BEGIN
        ##根据每个文件夹前75%的文件创建词典
        t = []
        filelist = os.listdir(path)
        for fi in filelist:
            if(os.path.isdir(path+'\\'+fi)):
                t.append(path+'\\'+fi)
        for k in t:
            count = 0;
            print("正在分析"+k+'\n')
            filesum = getSum(k)
            everyfile = []
            fh = os.listdir(k)
            for m in fh:
```

```
            if(os.path.isfile(k+'\\'+m)):
                everyfile.append(k+'\\'+m)
        for d in everyfile:
            if(count < int(filesum*0.75)):
                analyzefile(d)
                count = count + 1
            else:
                pass
        print("分析完毕"+k+"\n"+"总共："+str(int(count*0.75))+"个文件"+"\n")
    SaveDict()
    print(len(words))
    #nlp.close()
    ##END


def openmydict():
    ##BEGIN
    ##将词典文件读取进来
    try:
        g = open("mydict.txt",encoding='gb18030',errors='ignore')
        p = 0
        for k in g:
            if(p<200):
                dictwords.append(k.strip('\n'))
                p = p + 1
            else:
                pass
    finally:
        if g:
            g.close()
    ##END


def buildfilevector(d):
    ##BEGIN
    ##将当前文件表示成向量
    vec = []
    for i in range(0,200):
        vec.append(0)
    try:
        f = open(d,encoding='gb18030',errors='ignore')
        for line in f.readlines():
            types = nlp.pos_tag(line.strip())
            for t in types:
                    if(t[0].lower() in dictwords):
                        vec[dictwords.index(t[0].lower())] = vec[dictwords.index(t[0].lower())] +
1
                    else:
                        pass
        finally:
            if f:
                f.close()
        return vec
        ##END



    def trainingdata():
        ##BEGIN
        ##用75%的数据集建立模型
        t = []
        filelist = os.listdir(path)
        for fi in filelist:
            if(os.path.isdir(path+'\\'+fi)):
                t.append(path+'\\'+fi)
        typenum = 0
        for k in t:
            typenum = typenum + 1
            count = 0;
            print("正在建立模型，当前文件夹为："+k)
            filesum = getSum(k)
            everyfile = []
            fh = os.listdir(k)
            for m in fh:
                if(os.path.isfile(k+'\\'+m)):
                    everyfile.append(k+'\\'+m)
            for d in everyfile:
                if(count < int(filesum*0.75)):
                    vector = buildfilevector(d)
                    trainingvector.append(vector)
                    trainingtags.append(str(typenum))
                    count = count + 1
                else:
                    pass
            print("当前文件夹完毕"+k+"\n"+"总共："+str(int(count*0.75))+"个文件"+"\n")
        ##END
```

```
def getdis(filevec,i):
    ##BEGIN
    ##计算向量filevec与trainingvector中下标为i的向量之间的距离
    sum = 0
    for h in range(0,200):
        sum = sum + (int(filevec[h]) - int(trainingvector[i][h])) * (int(filevec[h]) - int(trainingvector[i][h]))
    return sum
    ##END


def dismin(dis,k):
    ##BEGIN
    ##返回数组中第k小的元素下标
    dis1 = dis
    le = len(dis)
    for i in range(0,k):
        for j in range(i,le):
            if(dis1[i] > dis1[j]):
                mid = dis1[i]
                dis1[i] = dis1[j]
                dis1[j] = mid
    return dis.index(dis1[k-1])
    ##END


def test_predict():
    ##BEGIN
    ##测试数据
    t = []
    filelist = os.listdir(path)
    for fi in filelist:
        if(os.path.isdir(path+'\\'+fi)):
            t.append(path+'\\'+fi)
    type5 = 0
    testsum = 0
    for k in t:
        type5 = type5 + 1
        count = 0;
        print("正在预测测试数据"+k+'\n')
        filesum = getSum(k)
        testsum = testsum + int(filesum*0.25)
        correct = 0
        everyfile = []
        fh = os.listdir(k)
        for m in fh:
            if(os.path.isfile(k+'\\'+m)):
                everyfile.append(k+'\\'+m)
        for d in everyfile:
            if(count < int(filesum*0.75)):
                count = count + 1
            else:
                if (count < int(filesum)):
                    m = predicttype(d,type5)   #0表示错误，1表示正确
                    correct = correct + m
                else:
                    pass
        print("预测完毕"+k+"\n"+"总共："+str(count)+"个文件"+"\n")
    s = str(correct/testsum)
    return s
    ##END



def predicttype(d,t):
    ##BEGIN
    ##对于文件d，文件真实的类别为t，判断是否分类正确
    filevector = buildfilevector(d)
    num = []
    for i in range(0,20):
        num.append(1)
    for i in range(0,20):
        for p in range(0,200):
            num[i] = num[i] * (math.exp((-1)*(int(filevector[p])-int(avg[i]))*(int(filevector[p])-int(avg[i])) / (var[i] * var[i])) ) / ((var[i])*sqrt(6.28))
    max = num[0]
    flag = 0
    for y in range(0,20):
        if(max < num[y]):
            max = num[y]
            flag = y
    if(int(y+1) == t):
        return 1
    else:
        return 0
    ##END
```

```
def caculateavg():
    ##BEGIN
    ##计算训练数据均值
    for t in range(0,20):
        num = []
        n = -1
        for i in range(0,200):
            num.append(0)
        for r in trainingvector:
            n = n + 1
            if(int(trainingtags[n]) == t+1):
                for k in range(0,200):
                    num[k] = (num[k] + int(trainingvector[n][k])) / 2
        avg.append(num)
    ##END

def caculatevar():
    ##BEGIN
    ##计算训练数据方差
    for t in range(0,20):
        num = []
        n = -1
        ssum = 0
        for i in range(0,200):
            num.append(0)
        for r in trainingvector:
            if(int(trainingtags[n]) == t+1):
                n = n + 1
                ssum = ssum + 1
                for k in range(0,200):
                    num[k] = num[k] + (int(trainingvector[n][k])-int(avg[t][k])) * (int(trainingvector[n][k])-int(avg[t][k]))
            else:
                n = n + 1
        for d in range(0,200):
            num[d] = num[d] / ssum
        avg.append(num)
    ##END

if __name__ == '__main__':
    #buildDict()                #扫描每一类前75%的文件建立词典，并将词典存到本地
    openmydict()                #读取本地词典
    trainingdata()              #用每一类前75%的数据构建模型
    caculateavg()               #计算词典每一类文件每一属性上的均值
```

```
caculatevar()              #计算词典每一类文件每一属性上的方差

cnum = test_predict()      #用每一类最后25%的数据进行测试，获得分类准确率
print("分类准确率为" + str(cnum))
nlp.close()
```