

DM Homework Report

----KNN 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 KNN 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优化 KNN 算法中 K 的值。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 KNN 参数的优化工作。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。

3. KNN 参数优化

对于 KNN 算法，参数 K 对于分类的准确率有着重要的作用，在 75% 的训练数据中，考虑分理处一部分数据，来预先测试模型，通过改变 K 值来观察分类准确率，取得最优 K 值。

（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：

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

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，将每个文件与建立出来的向量矩阵进行距离计算，确定使得 KNN 分类效果最好的 K 值。

3. 根据确定的最优 K 值对测试数据进行测试

对于每一类后 25% 的数据，在最优值 K 的确定下进行 KNN 算法的测试。

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50%的数据构建模型
    k = getfittestk()      #用每一类50%-75%的数据预测试，获取最优K值
    cnum = test_predict(k) #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

一、全局变量声明

```
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
trainingvector = []                        #训练数据的 VSM
path = "20news-18828"                     #data 路径
```

二、函数汇总

```
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(0,500):
        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<500):
                p.write(q1)
                p.write("\n")
                len = len + 1
            else:
                pass
        len = 0
        for q2 in wordscount:
            if(len<500):
                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 valuelwords 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<500):
                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(0,500):
        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 50%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*0.5)):
                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*0.5))+ "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(0,500):
        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 predicttype(d,j,t):
    ##BEGIN
    ##对于文件 d, K 的值取 j, 文件真实的类别为 t, 判断是否分类正确
    filevector = buildfilevector(d)
    dis = []
    for i in range(0,500):
        d = getdis(filevector,i)
        dis.append(str(d))
    t = []
    for k in range(1,j+1):
        t.append(str(trainingtags[dismin(dis,k)]))
    m = []
    for q in range(0,j):
        m.append(0)
    for q1 in range(0,j):
        for q2 in range(0,j):

```

```

        if(t[q1] == t[q2]):
            m[q1] = m[q1] + 1
ptype = int(t[m.index(max(m))])
if(ptype == t):
    return 1
else:
    return 0
##END

```

```

def train_predict(j):
    ##BEGIN
    ##测试数据，其中 K 取 j 维
    t = []
    filelist = os.listdir(path)
    for fi in filelist:
        if(os.path.isdir(path+'\\'+fi)):
            t.append(path+'\\'+fi)
    type5 = 0
    for k in t:
        type5 = type5 + 1
        count = 0;
        print("正在预测训练数据"+k+'\n')
        filesum = getSum(k)
        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*0.5)):
                count = count + 1
            else:
                if (count < int(filesum*0.75)):
                    m = predicttype(d,j,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")
    return correct
##END

```

```

def getfittestk():
    ##BEGIN
    ##寻找最优 K 值
    m = []
    for i in range(1,6):
        t = train_predict(i)
        m.append(str(t))
    return m.index(max(m)) + 1
    ##END

```

```

def test_predict(j):
    ##BEGIN
    ##测试数据，其中 K 取 j 维
    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,j,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
```

三、主函数展示

```
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 50%的数据构建模型
    k = getfittestk()     #用每一类 50%-75%的数据预测试，获取最优 K 值
    cnum = test_predict(k) #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()
```