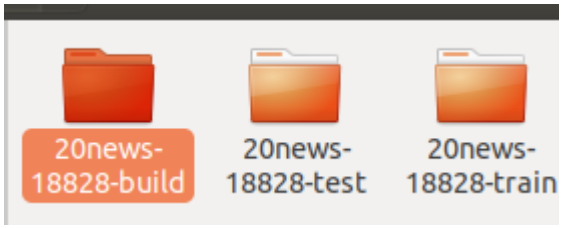
实验步骤：将数据集解压后，将其中的 80%作为训练集，20%作为测试集。



预处理时报错，缺少所需的 stopwords：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/nltk/corpus/util.py", line 80, in __load
    try: root = nltk.data.find('{}/{}'.format(self.subdir, zip_name))
  File "/usr/local/lib/python3.6/dist-packages/nltk/data.py", line 675, in find
    raise LookupError(resource_not_found)
LookupError:
**********************************************************************
  Resource stopwords not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('stopwords')

  Searched in:
    - '/home/liubuntu/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
    - '/usr/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/lib/nltk_data'
**********************************************************************
```

解决办法：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/liubuntu/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

python 读取文件时发生了错误

```
    main()
  File "knn.py", line 199, in main
    preProcess()
  File "knn.py", line 25, in preProcess
    preProcessFile(dirs[i],files[j])
  File "knn.py", line 34, in preProcessFile
    data = open(src,"r").readlines()
  File "/usr/lib/python3.6/codecs.py", line 321, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfd in position 293: invalid start byte
```

解决办法：

将

```
data = open(src,"r").readlines()
```

改为：

```
data = open(src,"rb").readlines()
```

单个文件的 VSM 结果：

运行结果：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/liubuntu/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
correctness is  0.7659574468085106
```

代码原理：

1、 预处理：

```
#创建新文件夹，存放预处理后的文本数据
def preProcess():
    dirs=os.listdir(rootDir)
    for i in range(len(dirs)):
        if i == 0:
            continue
        fileDir=rootDir+dirs[i]
        buildFileDir=buildDir+dirs[i]

        files=os.listdir(fileDir)

        if os.path.exists(buildFileDir)==False:
            os.mkdir(buildFileDir)

        for j in range(len(files)):
            preProcessFile(dirs[i],files[j])

#建立目标文件夹，生成目标文件
def preProcessFile(dir,fileName):
    src = rootDir+ "/" + dir + "/" + fileName
    dst = buildDir + "/" + dir + "/" + fileName

    dstFile = open(dst,'w')
    data = open(src,"rb").readlines()

    for line in data:
        result = processLine(line)
        for word in result:
            dstFile.write('%s/n' % word)
```

```
            dstFile.close()



        #对每行字符串进行处理，主要是去除非字母字符，转换大写为小写，去除停用词
        def processLine(line):
            stopwords = nltk.corpus.stopwords.words('english') #去停用词
            porter = nltk.PorterStemmer()  #词干分析
            splitter = re.compile('[^a-zA-Z]')  #去除非字母字符，形成分隔
    for i in range(len(line)):
                if chr(line[i]).isalpha():
                string+=chr(line[i])
            else:
                string+=" "

            words = [porter.stem(word.lower()) for word in splitter.split(string)/
                if len(word)>0 and/
                word.lower() not in stopwords]
        return words

2、 建立 vsm
# 统计每个词的总的出现次数
def countWords():
    wordMap = {}
    wordMap2 = {}

    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')
                wordMap[word]=wordMap.get(word,0) + 1
    for key, value in wordMap.items():
        if value > 10:#去掉出现次数小于 10 的词
            wordMap2[key]=value

    return wordMap2


#计算 idf
```

```
def IDF():
    wordFileMap = {}
    IDFMap = {}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')

                if wordFileMap.get(word)== None:#该词在所有文件里第一次出现
                    wordFileMap[word]=[]
                    wordFileMap[word].append(filePath)
                elif filePath in wordFileMap[word] == False: #该词在该文件里第一次出现
                    wordFileMap[word].append(filePath)


    for word in wordFileMap:
        IDFMap[word] = 20000.0/len(wordFileMap[word]) #文件总数/该词出现的文件个数
    return IDFMap

#计算一个文件的向量
def Vector(file,wordMap,IDFMap):
    fileVector = {}
    for line in open(file).readlines():
        word=line.strip('/n')
        if word not in wordMap:
            continue
        if word not in fileVector:
            fileVector[word]=1
        else:
            fileVector[word]+=1

    for key in fileVector:
        fileVector[key]=fileVector[key]*IDFMap[word]
       return fileVector
```

3、 KNN 分类器

#建立训练集的 map{向量：分类}

```
def vectorMap(wordMap,IDFMap):
    vectorMap={}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileVector = Vector(filePath,wordMap,IDFMap)
            vectorMap[fileVector]=filedir

    return vectorMap

#计算两个向量的相似度
def similarity(trainVector,testVector):
    trainlist = []
    testlist = []
    for word in testVector:
        if word in trainVector:
            trainlist.append(trainVector[word])
            testlist.append(testlist[word])

    trainVect = numpy.mat(trainlist)
    testVect = numpy.mat(testlist)

    num = float(testVect * trainVect.T)
    denom = numpy.linalg.norm(testVect) * numpy.linalg.norm(trainVect)
    return float(num)/(1.0+float(denom))

#根据模型预测测试文件的分类
def testFileSimilarity(testFile,wordMap,IDFMap,trainVectors):
    similarityMap = {}
    testVectoer = Vector(testFile,wordMap,IDFMap)
    resultMap = {}
    for vector in trainVectors:
        sim = similarity(vector,testVectoer)
        similarityMap[sim] = trainVectors[vector]

    keys = similarityMap.keys()
    keys.sort()
    sortedResult =  [(k,similarityMap[k]) for k in sorted(similarityMap.keys())]
    count = 0

    for i in sortedResult:
```

```
        if i[1] not in resultMap:
            resultMap[i[1]] = 1
        else:
            resultMap[i[1]] += 1
        count = count + 1
        if count > 100:#选择的 K 值为 100
            break

    max = 0
    doctype = ""
    for key,value in resultMap:
        if value>max:
            doctype = key
            max = value

        return doctype
```

4、观察在测试集上运行的准确率

```
#预测文件的分类并计算正确度
def predictTestFileCorr(wordMap,IDFMap,trainVectors):
    dirs = os.listdir(testDir)

    for i in range(len(dirs)):
        filedir = testDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileNumber = fileNumber + 1
            predict = testFileSimilarity(filePath,wordMap,IDFMap,trainVectors)
            if predict == dirs[i]:
                correctCount = correctCount + 1

    return float(correctCount/fileNumber)

def main():
        preProcess()
        wordMap = countWords()
        IDFMap= IDF()
        vectors = vectorMap(wordMap,IDFMap)
        correctness = predictTestFileCorr(wordMap,IDFMap,vectors)
        print(correctness)
```