作业四 文本纠错
任务描述：给出三个文件，词典库vocab.txt（任何不在词典中出现的词都认为是拼写错误），spell-errors.txt（给出正确单词与常见的拼写错误）与测试数据testdata.txt（1000个存在拼写错误的句子），利用bigram语言模型与Noisy Channel Model进行文本纠错训练。
步骤（每一步注释标注）：
一、读取词典库vocab.txt；
二、对于词库里没有的词，取编辑距离为1和2，生成候选词集合；
三、读取spell-errors.txt，计算概率p(错误的单词|正确的单词)；
四、选用nltk语料库movie_reviews，构建bigram语言模型；
五、读取testdata.txt，找出拼错词，生成候选词，并计算每个候选词的概率；
六、选出概率最高的候选词，作为修改词。
注：为便于计算，可采用似然对数，将概率乘积转化为求和。注意平滑。
输出：对每个句子，输出句子编号，拼错词，修改词（若没有则输出False），候选词列表及概率。最后输出总拼错词数。
例如：
第1句：
protectionst, protectionist
{'protectionist': -30.484951869432177}
第2句：
Tkyo's, False
第3句：
retaliation, retaliation
{'retaliation': -30.97301670912914}
Japan's, Japan
{'Japan': -30.748299246173307, 'Japanese': -30.748299246173307}
……
Total mistakes: 1351

一.读取vocab.txt
注意到在词典中是分大小写的，为了编辑距离部分的正常撰写
这里并没有做大小写统一的处理，而是保留原词典中的大小写

In [12]:

```
dictionary = set(open("vocab.txt").read().split()) #生成字典,统一小写，以免区分大小写
#print(dictionary)
```

二.生成候选词集合
本部分一共编写了三个函数
(1) 生成编辑距离为1的函数,通过枚举替换，取代与删除然后一并去重，返回在字典中的元素
(2) 生成编辑距离为0-2的函数，通过调用(1)的函数，将(1)的结果再次调用，生成与原单词编辑距离为0-2的单词
(3) 同时调用(1)与(2)的函数,对于编辑距离为0-1的，将通过去重剔除，最终保留结果即为编辑距离为1和2的集合/列表
最终请通过调用 candidate(word) 来获取一个词的候选词

In [27]:

```
def getWordsSetWithDistanceOne(word): #生成编辑距离为1的候选词
    alphabet = "abcdefghijklmnopqrstuvwxyz"  #字母表
    splits = [(word[:i],word[i:]) for i in range (len(word)+1)] #生成所有分割组
    insert = [a+x+b for a,b in splits for x in alphabet]
    delete = [a+b[1:] for a,b in splits]
    replace = [a+x+b[1:] for a,b in splits for x in alphabet]
    candidate = set(delete+insert+replace)
    return [word for word in candidate if word in dictionary]

def getWordsSetWithDistanceTwo(word): #生成编辑距离为2的候选词
    tmp = getWordsSetWithDistanceOne(word);  #生成距离为1的候选词
    result = []
    for string in tmp:
        result += getWordsSetWithDistanceOne(string)
    return result

def candidate(word):   #候选词集合
    allWords = getWordsSetWithDistanceOne(word)+getWordsSetWithDistanceTwo(word)
    return list(set(allWords))
```

三.读取spell-errors.txt，计算概率p(错误的单词|正确的单词)
查看spell-errors.txt中的内容,发现单词以 word:error1,error2,……,errorn的形式存储
但是除了普通的陈列展示外，还存在errorx*N的形式，因此要计算概率，首先需要处理每一行的字符串
最终保留结果是一个多层嵌套字典
errorDictionary是包含了 {key1:key1Dictionary,key2:key2Dictionary}字典
其中每个元素是一个{key1:key1Dictionary}形式的字典
而keyDictionary则是 {w1:p1,w2:p2,……,wn:pn}的字典

In [43]:

```
allErrors = open("spell-errors.txt").read().split("\n")#读取元素
errorDictionary = {}    #最终的存储结果
for Error in allErrors:
    key,val = Error.split(":")  #获取单词与错误列表
    errorList = val.split(",")  #生成错误单词的列表
    probability = {}            #概率词典
    wordsCount = 0;
    for e in errorList:         #个数计数
        if "*" in e:            #errorx*N的格式
            a,b = e.split("*")
            probability[a.strip()] = int(b)
            wordsCount += int(b)
        else:                   #普通格式
            probability[e.strip()] = 1
            wordsCount += 1

        for p in probability:
            probability[p] /= wordsCount #将个数转换成频率
    errorDictionary[key] = probability
#print(errorDictionary)
```

四.选用nltk语料库movie_reviews，构建bigram语言模型
总的来说就是需要计算每个单词的频率和两个连续单词出现的频率
我们通过 oneWordDict来存储单个单词的频率
然后通过 twoWordDict来存储两个单词连续在一起的频率

In [57]:

```
from nltk.corpus import movie_reviews
comments =movie_reviews.sents(categories=movie_reviews.categories()) #获取所有评论的关键分词
oneWordDict = {}
twoWordsDict = {}

for comment in comments:
    for i in range(len(comment)-1):
        bigram = comment[i] + " " + comment[i+1]
        #计数
        if comment[i] in oneWordDict:
            oneWordDict[comment[i]] += 1
        else:
            oneWordDict[comment[i]] = 1
        #计数
        if bigram in twoWordsDict:
            twoWordsDict[bigram] += 1
        else:
            twoWordsDict[bigram] = 1
wordsOfComments = len(oneWordDict)
```

五.读取testdata.txt，找出拼错词，生成候选词，并计算每个候选词的概率
选出概率最高的候选词，作为修改词。

大致思路如下:
以一行为一个处理单元，检索这行中的未出现在dictionary,即错误单词
随后对于每个错误单词，首先生成其候选词列表，若不存在任何候选词，那么输出False
否则对于每个候选词，设置一个概率P,通过spell-error.txt中的信息与movie_review中建立的词典
综合考虑，通过log运算加权两个参考值，对于不出现在词典的情况，设置一个极小值来平滑
然后以概率为唯一关键词，找出具有最大p值的候选词，选择为纠正词
最后输出所有信息

In [69]:

```
import numpy as np
mistakeCount = 0
for line in open("testdata.txt"):
    line = line.split()
    string = line[2:]
    for i in range (len(string)):
        if string[i][-1] in ".,":
            string[i] = string[i][:len(string[i])-1]
    print("第"+line[0]+"句:")
    for word in string:
        #去除末尾的标点
        if word not in dictionary:
            mistakeCount += 1          #错误词汇总数加1
            candidateList = candidate(word)#生成候选词

            if(len(candidateList) == 0):#如果不存在候选词
                print(word+","+"False")
            else:
                probabilityList = []
                for c in candidateList:
                    #如果这一组(候选-错误)在错误词典中
                    if c in errorDictionary and word in errorDictionary[c]:
                        p = np.log(errorDictionary[c][word])
                    else:
                        p = np.log(1e-6) #等价于log一个极小值

                    #与前一个单词组合
                    pos = string.index(word)
                    if(pos != 0):
                        bigram = string[pos-1]+" "+word;
                    else:
                        bigram = word;
                    if bigram in twoWordsDict:
                        up = twoWordsDict[bigram]+1
                        if pos == 0:
                            down = wordsOfComments
                        elif string[pos-1] not in oneWordDict:
                            down = wordsOfComments
                        else:
                            down = oneWordDict[string[pos-1]]+wordsOfComments
                        p += np.log(up/down)
                    else:
                        p += np.log(1/wordsOfComments)
                    probabilityList.append(p)
                print(word+","+candidateList[probabilityList.index(max(probabilityList))])#输出
                #输出每个候选词的概率
                print("{",end = "")
                for i in range(len(candidateList)):
                    print(candidateList[i]+":"+str(probabilityList[i]),end = "")
                    if(i == len(candidateList)-1):
                        print("}")
                    else:
                        print(",",end="")
```

```
第1句:
protectionst,protectionism
{protectionism:-24.406227817296312,protectionist:-24.406227817296312}
第2句:
```

```
Tkyo's,False
第3句:
retaiation,retaliation
{retaliation:-24.406227817296312}
Japan's,False
第4句:
tases,taxis
{taxis:-24.406227817296312,cased:-24.406227817296312,case:-24.406227817296312,til
es:-24.406227817296312,stakes:-24.406227817296312,lakes:-24.406227817296312,biase
s:-24.406227817296312,tusks:-24.406227817296312,capes:-24.406227817296312,taps:-2
4.406227817296312,vases:-24.406227817296312,wales:-24.406227817296312,tapis:-24.4
06227817296312,tanks:-24.406227817296312,dales:-24.406227817296312,tale:-24.40622
7817296312,bases:-24.406227817296312,cages:-24.406227817296312,cases:-24.40622781
7296312,axes:-24.406227817296312,caves:-24.406227817296312,types:-24.406227817296
312,tapes:-24.406227817296312,taken:-24.406227817296312,bales:-24.40622781729631
```

六.最后输出总拼错词数

In [70]:

```
print("Total mistakes: "+str(mistakeCount))
```

```
Total mistakes: 1347
```