Assignment 3

Information Retrieval

CS 834

Fall 2017

Mohammed Nauman Sididque

November 9, 2017

Contents

Chapter 1

Problem 6.2

1.1 Problem Statement

Create a simple spelling corrector based on the noisy channel model. Use a single-word language model, and an error model where all errors with the same edit distance have the same probability. Only consider edit distances of 1 or 2. Implement your own edit distance calculator (example code can easily be found on the Web).

1.2 Solution

1.2.1 Noisy Channel Model

This problem has been solved using Wiki small dataset provided by the text book. Initially we need to find the words in the corpus. The corpus contains 3902115 words of which 225744 are unique. The output of the total words is in file "word_list.txt" which has been added to the Assignment3 folder of Github.

Code parsing out words from the wiki small dataset

```
'''
Created on Nov 7, 2017

@author: nauman
'''
import os
from bs4 import BeautifulSoup
import codecs
from string import punctuation
import re

def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

## Calculate Words in wiki small dataset
def createDictionary():
    file_write=codecs.open("word_list.txt","w","utf-8")
    exclude = set(punctuation)
    doc_count = 0
```

```
    for root, dirs, files in os.walk("F:\Fall2017\InformationRetreival\Assignment2\en"):
        for file in files:
            if file.endswith(".html"):
                with codecs.open(root+"\\" + file, "r", "utf-8") as file_open:
                    doc_count = doc_count +1
                    count = 0
                    soup = BeautifulSoup(file_open,'html.parser')
                    for words in soup.findAll(text=True):
                        if words.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
                            continue

                        s = ''.join(ch for ch in str(words) if ch not in exclude)
                        words = s.split(" ")
                        for word in words:
                            count = count+1
                            if not RepresentsInt(word.rstrip()):
                                if re.search('[a-zA-Z]', word.rstrip()):
                                    file_write.write(word.rstrip().lower())
                                    file_write.write("\n")

    file_write.close()
    print("WordsOver")
```

The next step was to find the unique words of from the list of total words. I used unix commands as mentioned below:

```
$  sort word_list.txt | uniq -c > word_list_unique.txt
```

This command sorts my word_list file and counts all the unique words to save it to the file word_list_unique.txt

```
$  awk '{print $1}' word\_list_unique.txt > word_list_unique_frequency.txt
```

This command cuts the frequency of unique words from word_list_unique.txt and saves to word_list_unique_frequency.txt

```
$  awk '{print $2}' word_list_unique.txt > word_list_unique_.txt
```

This command cuts the unique words listed from word_list_unique.txt and saves to word_list_unique_.txt

The above two files were then used by the code to save them to lists.

```
'''
Created on Nov 7, 2017

@author: nauman
'''
import pickle
import codecs

def countWordFrequency():
    file = codecs.open("word_list.txt","r","utf-8")
    file_unique = open("word_unique.pkl", "w")
    file_frequency = open("word_frequency.pkl","w")
```

```python
    word = []
    frequency_word = []
    for line in file:
        if line in word:
            index = word.index(line)
            temp = frequency_word[index]
            frequency_word[index] = (temp+1)
        else:
            word.append(line.rsplit())
            frequency_word.append(1)
      print(line)
    file.close()
    file_unique.write(pickle.dumps(word))
    file_frequency.write(pickle.dumps(frequency_word))
    file_unique.close()
    file_frequency.close()

#countWordFrequency()

def writeToList():
    file_unique = open("word_unique.pkl", "w")
    file_frequency = open("word_frequency.pkl","w")
    file_word = open("word_list_unique_.txt","r")
    file_frequency1 = open("word_list_unique_frequency.txt", "r")
    word = []
    frequency = []
```

```
    for line in file_word:
        word.append(line.rstrip())
    for line in file_frequency1:
  frequency.append(line)
    file_unique.write(pickle.dumps(word))
    file_frequency.write(pickle.dumps(frequency))
    file_unique.close()
    file_frequency.close()

writeToList()
```

In caculating edit distance between strings insetion, deletion and replace have been considered. Transformation has not been implemented.

Code for calculating edit distance between two strings.

```
'''
Created on Nov 7, 2017

@author: nauman
'''

def editDistDP(str1, str2, m, n):

    # Create a table to store results of subproblems
    dp = [[0 for x in range(n+1)] for x in range(m+1)]

    # Fill d[][] in bottom up manner
```

```python
    for i in range(m+1):
        for j in range(n+1):

            # If first string is empty, only option is to
            # insert all characters of second string
            if i == 0:
                dp[i][j] = j    # Minimum operations = j

            # If second string is empty, only option is to
            # remove all characters of second string
            elif j == 0:
                dp[i][j] = i    # Minimum operations = i

            # If last characters are same, ignore last char
            # and recur for remaining string
            elif str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-1][j-1]

            # If last character are different, consider all
            # possibilities and find minimum
            else:
                dp[i][j] = 1 + min(dp[i][j-1],        # Insert
                                   dp[i-1][j],        # Remove
                                   dp[i-1][j-1])    # Replace

    return dp[m][n]
```

Code for NoisyChannel Model

```python
'''
Created on Nov 7, 2017

@author: nauman
'''

from EditDistance import editDistDP
from WordList import createDictionary
from WordFrequency import countWordFrequency
import pickle
from decimal import *

## Check for words that start with same capital letter
def getCorrectWord(word):
    getcontext().prec = 28
    suggestionList = []
    probability_list = []
    with open('word_unique.pkl', 'rb') as f:
        wordlist = pickle.load(f)
    with open('word_frequency.pkl', 'rb') as f:
        frequency = pickle.load(f)
    for i in range(0, len(wordlist)):
        if checkLength(wordlist[i], word) and wordlist[i][:1] == word[:1]:
            editDistance = editDistanceCalculator(wordlist[i],word)
            if editDistance == 1:
```

```
                probability = Decimal(int(frequency[i]))/Decimal(3902115* 0.6)
    suggestionList.append(wordlist[i])
    probability_list.append(probability)
            elif editDistance == 2:
    probability = Decimal(int(frequency[i]))/Decimal(3902115* 0.2)
                suggestionList.append(wordlist[i])
    probability_list.append(probability)
      elif editDistance == 0:
    print(wordlist[i])
    exit(1)

    maximum_probability = max(probability_list)
    index = probability_list.index(maximum_probability)
    print(suggestionList[index])

## Check for words than that are in length of +1 and -1

def editDistanceCalculator(str1,str2):
    editDistance = editDistDP(str1, str2,len(str1) , len(str2))
    return editDistance

def checkLength(word1,word2):
    len1 = len(word1)
    len2 = len(word2)
    if len1 == len2 or (len1 - len2) ==1 or (len2 - len1) == 1:
        return True
```

```
    else:
        return False

def main():
    createDictionary()
    countWordFrequency()
    getCorrectWord("collectionz")


main()
```

Noisy Channel Code

$$W = argmax_{w \epsilon V} P(x|w)P(w)$$

where W is predicted word

$P(x|w)$ is the error model

P(w) is the document mode

The above equation states that the word which has the highest product of document model and error model is the predicted word in noisy model channel.

1.2.2 Steps for generating the predicted correct word using Noisy Channel Model

- Parse out all posibble words from the corpus.

- Identify all the unique words with their frequency from the word list.
- Calculate the edit distance of the input word with each unique word present in the word list of the corpus.
- For each unique word in the corpus compute the probability product using equation 1 .
- Find the word with highest probability and that is your predicted correct word from corpus for the word entered by the user.

1.2.3 Assumptions

$*$ Consider only those unique words which have an edit distance of 0, 1 or 2.

$*$ Calculate the edit distnace if the first character of the unique word from corpus and word entered by user match.

$*$ Calculate edit distance if the difference of the length if word entered by user and unique word from corpus is 1.

$*$ The probability of error model have been provided with constant values. For edit distance 1 , the error model probability is set to 0.6 and for edit distance 2, the error model probability is set to 0.2.

1.2.4 Outputs

```
msiddique@wsdl-3102-03:~/Information Retrieval/Assignment3/Assignment3/Problem6_
2$ python NoisyModel.py
canadian
```

Figure 1.1: Output for input:canadai

```
msiddique@wsdl-3102-03:~/Information Retrieval/Assignment3/Assignment3/Problem6_
2$ python NoisyModel.py
0 Distance
collection
```

Figure 1.2: Output for input:collection

```
msiddique@wsdl-3102-03:~/Information Retrieval/Assignment3/Assignment3/Problem6_
2$ python NoisyModel.py
collection
```

Figure 1.3: Output for input:collectionz

```
msiddique@wsdl-3102-03:~/Information Retrieval/Assignment3/Assignment3/Problem6_
2$ python NoisyModel.py
cup
```

Figure 1.4: Output for input:copr

```
msiddique@wsdl-3102-03:~/Information Retrieval/Assignment3/Assignment3/Problem6_
2$ python NoisyModel.py
victoria
```

Figure 1.5: Output for input:victora

Chapter 2

Problem 6.4

2.1 Problem Statement

Assuming you had a gazetteer of place names available, sketch out an algorithm for detecting place names or locations in queries. Show examples of the types of queries where your algorithm would succeed and where it would fail.

2.2 Solution

2.2.1 Assumption about the Location Gazetteer

It does not accept any argument other than strings. It can not translate latitude and longitude, zipcodes and other aspects of address which rely on numerical value for translating address or location.

2.2.2 Description of the Algorithm

It parses the query term and splits them to query terms on the basis of space, colon and other punctuation marks used frequently in the address. Intially the algorithm sends all the query terms to the location gazatter and if it is a hit it processes the query. In case of miss, it extracts each query term and sends them individually to the location gazatteer. In case of a hit, the index of the query term is saved and then the query term is split in two blocks with index term as mid point. For example, a sample query is Sushi in Norfolk Found. Each query term is passed individually to th location gazatteer. In case of Norfolk it is a hit. Now the query is split as Sushi in as first block , Found as second block and Norfolk is kept as index variable. The first half of the query terms are evaluated in reverse order starting from index location to zero position query term. Each time a query is formed by concatenating with the previous query term and send it to location gazatteer. In case of hit, the query tasks are perfomed but in case of a miss, the process is repeated again the previous query term is added to the query and checked with location gazatteer. This continues till all the elements prior to the index term have not been part of the query and been checked in the location gazatteer. If it is a overall miss, this step is then performed for query terms index to the end of the query terms. Each time a query term is appended to the query which has index query term to check for hit or miss with location gazatteer. If both the conditions fail then the index term is sent to location gazatteer and further actions are performed based on it.

```
//Function location dictionary return Trues in case of hit or False if it is a miss.
```

```
locationDictionary(word)


//Splitting query terms for a query


splitQuery(query){

if(query.length == 1){

if("=" in query){

queryTerms <- split query by =

}

else if("&" in query){

queryTerms <- split query by &

}

else if("-"in query){

queryTerms <- split query by -

}

else if(":"in query){

queryTerms <- split query by :

}

else if(","in query){

queryTerms <- split query by ,

}
```

```
}
else{
queryTerms <- split query by whitespace
if "=" or "&" or "-" or ":" or "," in queryTerms{
Deleted from the list of queryTerms
}
}
return queryTerms
}


//Find locations in queryTerms


findLocation(queryTerms){
if (locationDictionary(queryTerms)){
//Location identified in query
//Use the location in indexing results for the query
exit 0
}
else{
for(i in 0 to queryTerms.length){
if(locationDictionary(queryTerms[i])){
```

```
index = i
break
}
}
for(i in index-1 to 0){
if(queryModified == "" ){
queryModified = queryTerms[i] + queryTerms[index]
if (locationDictionary(queryModified)){
//Location identified in query
//Use the location in indexing results for the query
exit 0
}
}
else{
queryModified = queryTerms[i] + queryModified
if (locationDictionary(queryModified)){
//Location identified in query
//Use the location in indexing results for the query
exit 0
}
}
```

```
}
for(i in index+1 to queryTerms.length){
if(queryModified == "" ){
queryModified = queryTerms[index] + queryTerms[i]
if (locationDictionary(queryModified)){
//Location identified in query
//Use the location in indexing results for the query
exit 0
}
}
else{
queryModified = queryModified + queryTerms[i]
if (locationDictionary(queryModified)){
//Location identified in query
//Use the location in indexing results for the query
exit 0
}
}
}
if(locationDictionary(queryTerms(index))){
//Location identified in query
```

```
//Use the location in indexing results for the query

}


}

}
```

Queries where it works:

- New York City
- Royal Bazaar
- Norfolk
- Walmart Norfolk

Queries where it fails:

- IA 264
- ZIP 23508
- lat 88.0 long 12.34
- ODU Address

Chapter 3

Problem 6.5

3.1 Problem Statement

Describe the snippet generation algorithm in Galago. Would this algorithm work well for pages with little text content? Describe in detail how you would modify the algorithm to improve it.

3.2 Solution

Dataset Used

I am using wiki-small corpus downloaded from the book website to analyze the snippet generation algorithm in Galago.

3.2.1 Description of Galago Snippet Algorithm

As shown in figure 3.1, Galago Search Engine presents search results with snippets for each indexed search result containing a title, URL and a short web summary to help the users find their desired result in quickly.

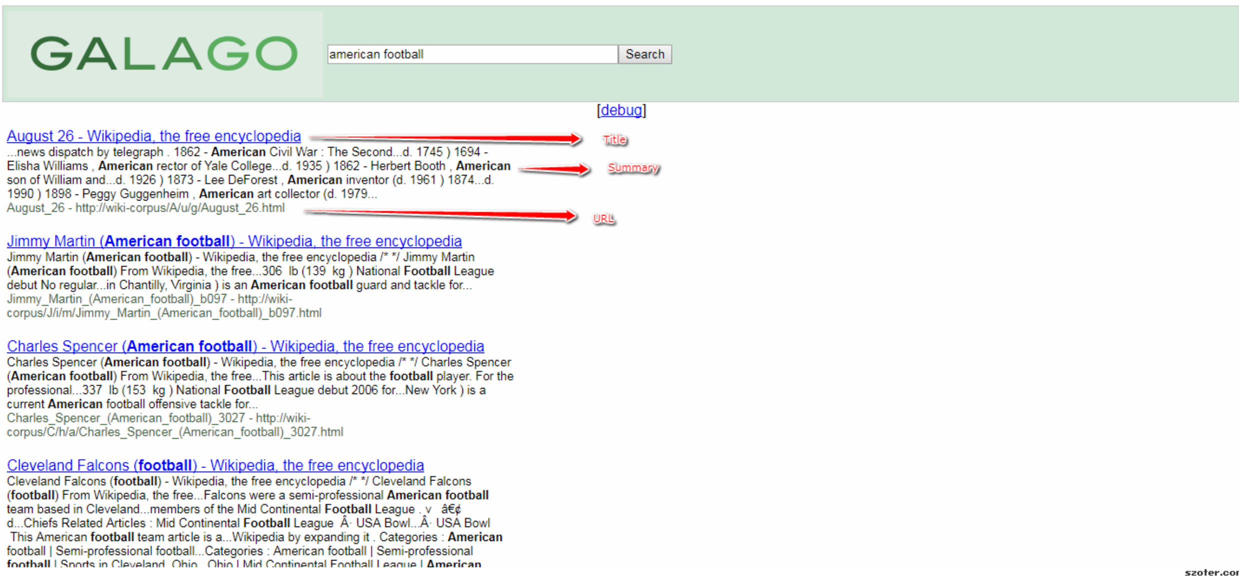

Figure 3.1: Web Snippet In Galago

The title displayed in the snippets are extracted from header (h1, h2, h3....) tags of the body of the webpage. The URLs displayed on search results have been modified to show standard URLs rather than the localhost extended URLs. The text for web summaries are extracted from the paragraph (p) tags of the body of the webpage.

The text generated for web summaries are fragmented sentences rather than complete English language sentences. The texts extracted to form sentences for web summary contain the query terms in it. They do not stem the query terms for searching the word in the document. For query term gem stone, they search for the exact query terms and do not

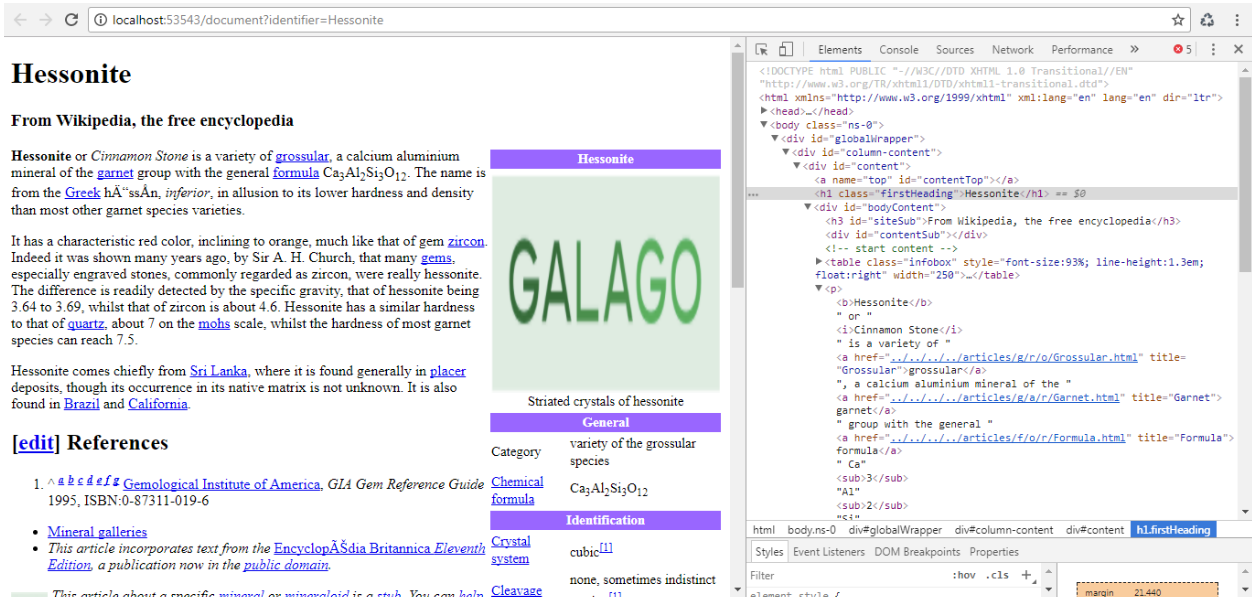

Figure 3.2: Description of Web Snippet Generation

consider the words like gems or stones for forming web summaries. They roughly form fragmented sentences of ten words where the query term is the sixth term. In case, another query term falls within the range of fragmented sentence they extend the fragmented sentence by one. So, the general rule for the length of fragmented senteces is for each query term falling in the range of fragmented sentence the length increases by the number of query terms. For example, if three query terms lie in the fragmented sentence then the length will be tweleve as two extra words have been added for two extra query terms showing up in the fragmented sentence. When framing fragmented sentences they follow the order in the webpage.For example, if the webpage has three query terms and the web summary is of two fragmented sentence then, the fragmented sentences are constructed of the first two occurences of the query term. The length of the web summary in terms of fragmented sentences can be determined as roughly equal to half the query terms present in the webpage must occur in the web summary. For example, if a page contains fifteen query terms, then the web summary must contain seven to eight query terms in their web summary. But this approach could mean that the number of fragmented sentences could be in range of one to eight.

3.2.2 Little Text Content Problem

One of the major problems with the Galago Snippet Generation algorithm with little text content is it might not produce any web summary if the query terms do not exactly match the words in the documents. As shown in figure 3.3, a web page with just a paragraph of text to validate our problem for little text content problem. For a search query gem stone the search result does not contain any web summary for this page as shown in figure 3.4. Although, the webpage does have one occurence of word gems which is a plural form of gem which can be seen in figure 3.3. So, with little text content it is very tricky for the Galago snippet generation algorithm to find the exact query terms each time in the document but without stemming of query terms it is bound to produce no web summary almost every time on my search request.

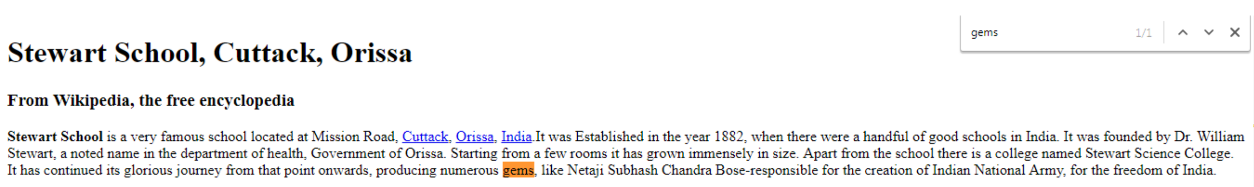

Figure 3.3: Document containing query term

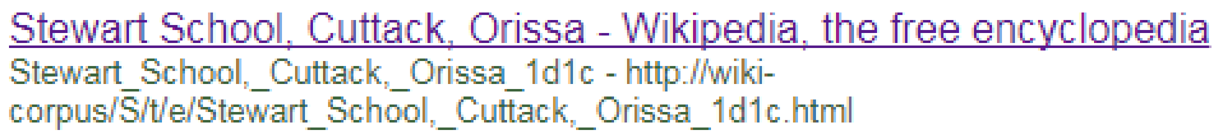

Figure 3.4: No Web summary generated for Query Term: Gem Stone

3.2.3 Improvements to the Algorithm

Firstly, the query terms need to be stemmed for incorporating their plural and other verb forms, which will help in generating web summaries for even little text content webpages.Secondly, the web summaries are supposed to be an abstract to the document, so the algorithm could extract some fragmented sentences from the metadata or the links which refer to the page rather than every time relying on the query term search in the socument. Thirdly, the length of web summaries needs to have some minimum and maximum threshold as it becomes a story when the search result contains five or greater sentences in web summary. The algorithm could even implement a feedback mechanism or analyze the clickthrough logs to underdstand the reasons for clickthrough inversions related to web summaries. The length of fragmented sentences do not need to be hard bound by any length but rather should convey meaningful information relating the the page.

Chapter 4

Problem 6.9

4.1 Problem Statement

Give five examples of web page translation that you think is poor. Why do you think the translation failed?

4.2 Solution

I chose to translate webpages from Hindi to English and from Urdu to English. This was done due to my familiarity with these languages which help me understand in detail the problems happening in translated version of the webpage.

4.2.1 Aaj Tak Website

It is a hindi news website. I loaded the webpage in the hindi and translated it to english by selection the option of translate to english on right click of the webpage. The reason for choosing this website as a bad translation is due to one news in particular. In figure 4.2, there is a news on the left side of the figure saying "Hardline never reached Sibal's meeting". The original news in hindi as shown in figure 4.1, says "The deadline on reservation ended by Patels, Hardik did not attend meeting with Sibal".

Hardline never reached Sibal's meeting - English Version

The deadline on reservation ended by Patels, Hardik did not attend meeting with Sibal - Hindi Version translated to English by me

If I look at both the news they seem very different to me because in english they converted Hardik to Hardline where Hardik is name of a person and hardline is something no where related to a name but it changes the whole narrative to the news now reading the headlines.

4.2.2 Aaj Tak News

Figure 4.3 is an English translation of a news from Aaj Tak, a Hindi news daily. The paragraph shown in the figure is gramatically incorrect and looks like the news in Hindi has been literally converted to English leaving out the rules of English language grammer.

Figure 4.1: News website: Aaj Tak in Hindi

4.2.3 Dainik Bhaskar

It is a Hindi news website. The paragraph of news translated in English as shown in figure 4.5 is not grammatically acceptable. There are numerous instances of wrong usage of capital and small letters and punctuations.

4.2.4 Urdu Newspaper

Figure 4.6 is an English translation of a news from Urdu newspaper. The texts in English are right to left aligned which is just a copy of the way urdu is written. Few proper nouns containing names of people mentioned in the news are wrong.

4.2.5 Urdu Newspaper from Pakistan

Figure 4.7 is an English translation of a news from Pakistani Urdu newspaper. The texts in English are right to left aligned which is just a copy of the way urdu is written. The most

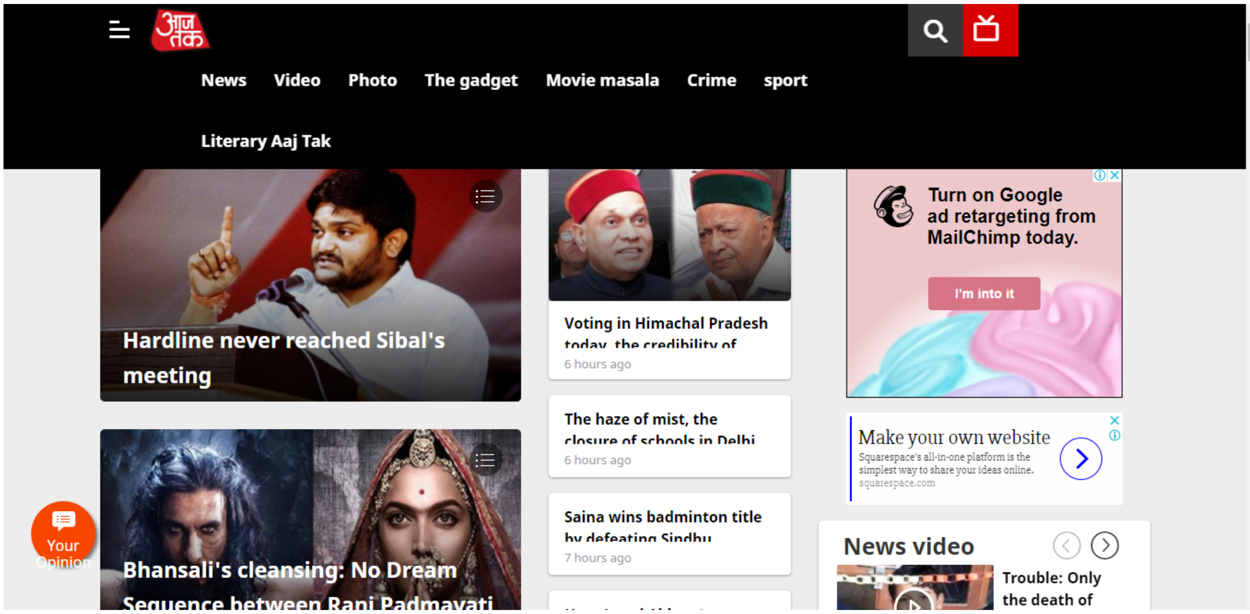

Figure 4.2: News website: Aaj Tak in English

standout things to notice int he figure are the commas(,). They have been inverted in the document making the translated version look very rudimentary.

4.2.6 Possible reason for Wrong Translation

Firstly, a major flaw in the translation of a webpage to English was with proper nouns containing names of people which are unique to the geographical area. Google might not have those names in its dictionary and translated it incorrectly. Second, most standout flaw was translating from a language that has a right to left writing stlye to English. It did not have any understanding of the layout of the text. Thirdly, a number of punctuations appeared wierd in Urdu to English translation and it could be due to just reversing their direction and not keeping in to consideration that the writing style had been changed from

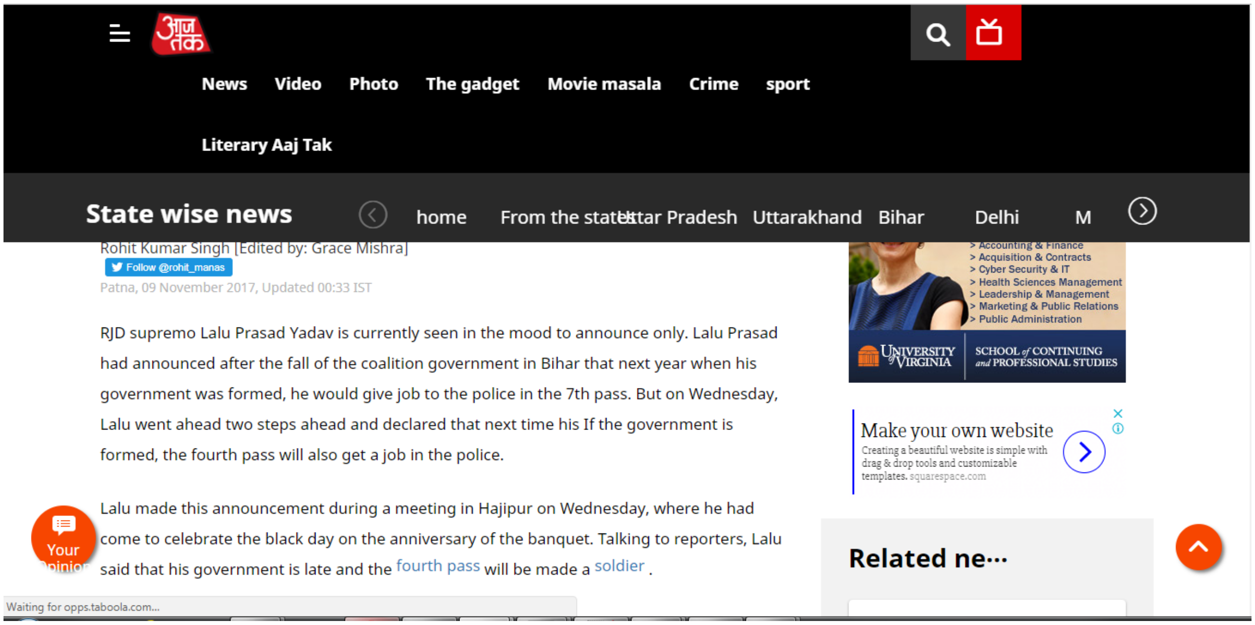

Figure 4.3: News website: Aaj Tak News translated to English

right - left to left- right. Thirdly, one of the pages had a literal translation from Hindi to English. The reason could due to word to word text translation from Hindi to English. Machines lack the understanding to look at sentences and paragraphs in context and often translate them as they encounter on word by word or sentence by sentence. The reason for grammatically incorrect translation in English from Hindi could be due togeneral flow of English language which does not follow a strict rule and above all the machines translating it must be trained to form sentences out of ideas rather than words. The reason even we are having this conversation of improvement in translation is due to Google which does a fairly good job at translating webpages but it is difficult to train machines to understand the behaviour of natural languages which are influenced by people who use it rather than any harlined grammer.

Figure 4.4: News website: Dainik Bhaskar in Hindi

Figure 4.5: News website: Dainik Bhaskar in English

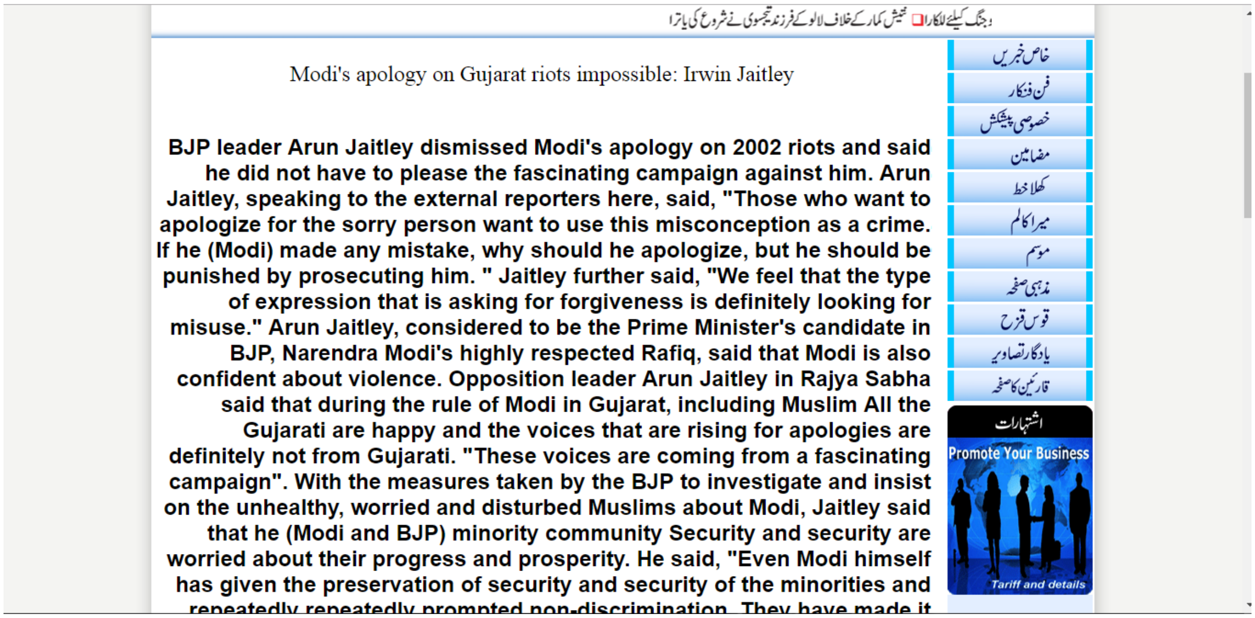

Figure 4.6: News website: Urdu Newspaper translated to English

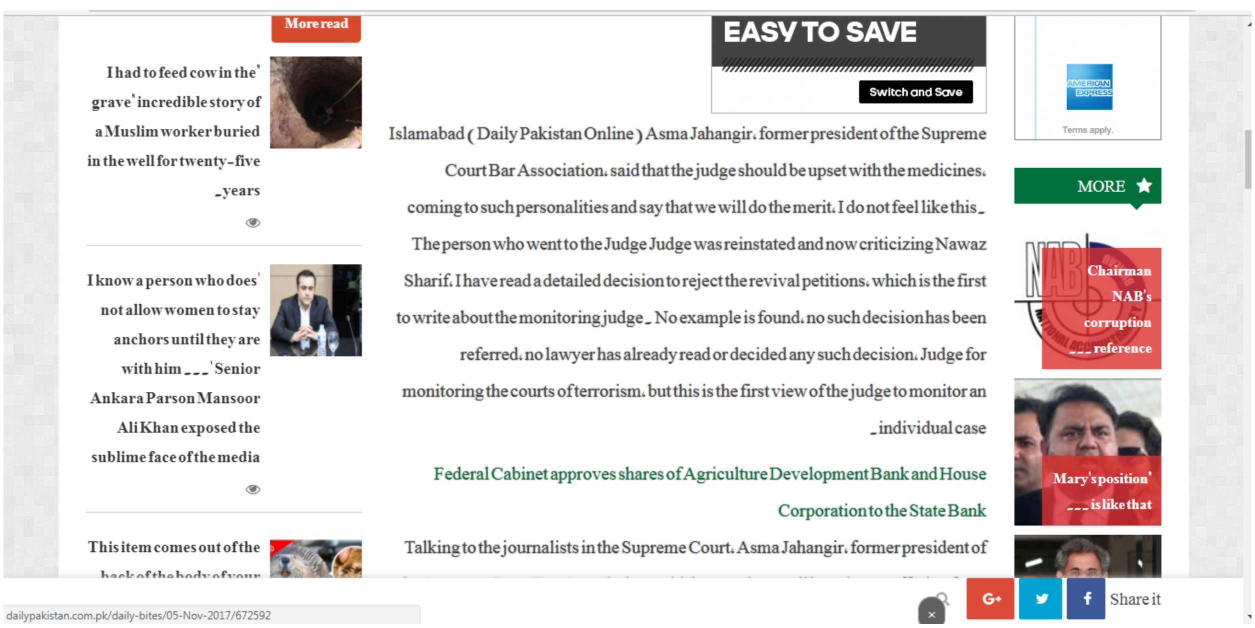

Figure 4.7: News website:Pakistani Urdu Newspaper translated to English

Chapter 5

Problem 7.2

5.1 Problem Statement

Can you think of another measure of similarity that could be used in the vector space model? Compare your measure with the cosine correlation using some example documents and queries with made-up weights. Browse the IR literature on the Web and see whether your measure has been studied (start with van Rijsbergens book).

5.2 Solution

We can use Manhattan Distance to find similarity in vector space model.Manhattan distance is the distance between two points is the sum of the absolute differences of their Cartesian coordinates. The lesser the distance the closer they are or similar the two points are. We can use Manhattan Distance as the closer a value is to zero the more the document and query models.

Example:

In a plane with p1 at (x1, y1) and p2 at (x2, y2).

$ManhattanDistance = |x1 - x2| + |y1 - y2|$

Suppose we have three documents D1,D2,D3 and three queries with q1,q2,q3.

The document vectors are assumed to be:

D1 = [0.8 ,0.7 ,0.6]

D2 = [1,0.5,0.3]

D3 = [0.7, 0.4, 0.9]

The query vectors are assumed to be:

q1 = [0.4, 0.2, 0.8]

q2 = [0.8, 0,3, 0.9]

q3 = [0.7, 0.9, 0.8]

Cosine Similarity

For Query q1

$\text{D1} = (0.8 * 0.4) + (0.7 * 0.2) + (0.6 * 0.8) / \sqrt{(0.8^2 + 0.7^2 + 0.6^2) + (0.4^2 + 0.2^2 + 0.8^2)}$

$\text{D1} = (0.32 + 0.14 + 0.48) / 1.59 = 0.59$

$\text{D2} = (1 * 0.4) + (0.5 * 0.2) + (0.3 * 0.8) / \sqrt{(1^2 + 0.5^2 + 0.3^2) + (0.4^2 + 0.2^2 + 0.8^2)}$

$\text{D2} = (0.4 + 0.1 + 0.24) / 1.086 = 0.681$

$\text{D3} = (0.7 * 0.4) + (0.4 * 0.2) + (0.9 * 0.8) / \sqrt{(0.7^2 + 0.4^2 + 0.9^2) + (0.4^2 + 0.2^2 + 0.8^2)}$

$\text{D3} = (0.28 + 0.08 + 0.72) / 1.517 = 0.712$

Manhattan Distance

For Query q1

D1 = $|0.8 - 0.4| + |0.7 - 0.2| + |0.6 - 0.8|$

D1 = $0.4 + 0.5 + 0.2 = 1.1$

D2 = $|1 - 0.4| + |0.5 - 0.2| + |0.3 - 0.8|$

D2 = $0.6 + 0.3 + 0.5 = 1.4$

D3 = $|0.7 - 0.4| + |0.4 - 0.2| + |0.9 - 0.8|$

D3 = $0.3 + 0.2 + 0.1 = 0.6$

For query q1 the cosine similarity index for documents D1, D2 and D3 are 0.59, 0.681 and 0.712. The ranking of documents in respect to the query q1 is $D3 > D2 > D1$. For query q2 the Manhattan Distance for documents D1, D2 and D3 are 1.1, 1.4 and 0.6. The ranking of documents in respect to the query q1 is $D3 > D1 > D2$.

Problems with Manhattan Distance The major flaw in the Manhattan Distance is the non-normalized value. In the above example the distance could vary from 0 to 3 because there are three co-ordinates to each vector. So to normalize the values of Manhattan Distance we can divide it by the number of co-ordinates in each vector which results in Manhattan Distance in range of 0 to 1.

$ModifiedManhattanDistance = (|x1 - x2| + |y1 - y2|)/N$

where N is the number of co-ordinates in each vector

Cosine Similarity

For Query q2

D1 $= (0.8 * 0.8) + (0.7 * 0.3) + (0.6 * 0.9)/\sqrt{(0.8^2 + 0.7^2 + 0.6^2) + (0.8^2 + 0.3^2 + 0.9^2)}$

D1 $= (0.64 + 0.21 + 0.54)/1.74 = 0.903$

D2 $= (1 * 0.8) + (0.5 * 0.3) + (0.3 * 0.9)/\sqrt{(1^2 + 0.5^2 + 0.3^2) + (0.8^2 + 0.3^2 + 0.9^2)}$

D2 $= (0.8 + 0.15 + 0.27)/1.697 = 0.719$

D3 $= (0.7 * 0.8) + (0.4 * 0.3) + (0.9 * 0.9)/\sqrt{(0.7^2 + 0.4^2 + 0.9^2) + (0.8^2 + 0.3^2 + 0.9^2)}$

D3 $= (0.56 + 0.12 + 0.81)/1.732 = 0.86$

Modified Manhattan Distance

For Query q2

D1 $= (|0.8 - 0.8| + |0.7 - 0.3| + |0.6 - 0.9|)/3$

D1 $= (0 + 0.4 + 0.3)/3 = 0.234$

D2 $= (|1 - 0.8| + |0.5 - 0.3| + |0.3 - 0.9|)/3$

D2 $= (0.2 + 0.2 + 0.6)/3 = 0.334$

D3 $= (|0.7 - 0.8| + |0.4 - 0.3| + |0.9 - 0.9|)/3$

D3 $= (0.1 + 0.1 + 0)/3 = 0.067$

For query q1 the cosine similarity index for documents D1, D2 and D3 are 0.903, 0.719 and 0.86. The ranking of documents in respect to the query q1 is $D1 > D3 > D2$. For query q2 the Manhattan Distance for documents D1, D2 and D3 are 0.234, 0.334 and 0.067. The ranking of documents in respect to the query q1 is $D3 > D1 > D2$.

Research Papers on calculating similarity using Manhattan Distance for Vector Space

Model (used Google Scholar)

1. Evaluation of texture features for content-based image retrieval, by P Howarth, SM Rger - CIVR, 2004 - Springer

There are a few other research papers related to Manhattan Distance for Vector Space Model but they are either a modified version of Manhattan Distance or they are talking about improvements over Manhattan Distance.

Chapter 6

Problem 7.7

6.1 Problem Statement

What is the bucket analogy for a bigram language model? Give examples.

6.2 Solution

In context of Information Retreival, Language Model is the ability to predict next word on the prior knowledge of the previous word. A language model for documents is the total vocabulary of the document which can be used to form query terms. A unigram language model is predicting the next word on the basis of their frerquency in the document. For example, if all the words present in the document are in a bucket and it is needed to contruct a new text with the bucket of words. A word is randomly chosen from the bucket without seeing in the bucket written down and the word is place back again in the bucket and then the search for next word happens in the similar fashion. It is the unigram language model where each word is independent of each other but are only just dependent on their occurence in the document. But, a bigram language model uses the prior knowledge of the previous word to predict the next word. The main advantage of bigram language model over unigram language model is that it considers the previous word in context to predict the next word unlike the unigram model which treats each word individually on their frequency. It can be shown by the below equation.

$$P_{bi}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3)$$

In respect to the bucket analogy, if all the words in the document are in a bucket and a new text is to be constructed using bigram language model,initially a word is picked from the bucket of words at random and writen down. Further on the bucket of words is replaced by the bucket of the words occuring with the prior word to select the next word and this process is continued where each time the bucket of words coming after the previous word is used to predict the next word.

For example, suppose there is a document that has following text.

i live in osaka .

i am a graduate student .

my school is in nara .

We want to find the probability of framing in osaka from the lanuage model using bigram model. It is defined by conditional probability of osaka occuring if in has already occured. The below equation says thatthe conditional probability of osaka with in is equal to the frequencies of occurence of term in osaka divided by frequency of in.

$$P(osaka|in) = c(inosaka)/c(in)$$

$$P(osaka|in) = 1/2 = 0.5$$

Similarly we want to predict the probability of nara occuring after in.

$$P(nara|in) = c(innara)/c(in)$$

$$P(nara|in) = 1/2 = 0.5$$

We want to predict school occuring after in.

$$P(school|in) = c(inschool)/c(in)$$

$$P(school|in) = 0/2 = 0$$

Chapter 7

Problem MLN1

7.1 Problem Statement

using the small wikipedia example, choose 10 words and create stem classes as per the algorithm on pp. 191-192..

7.2 Solution

7.2.1 Description of Code

The code uses a random list of words extracted from the word list generated by parsing out the wiki-small dataset. The code for parsing words from wiki-small dataset has been reused from problem 6.2 and is not part of the code for this problem. Porter The codde used Stemmer library of nltk package. The extacted words from the word list were stemmed to result in forming of 10 stems which will be used to build stem classes. The 10 stems, extarcted word list and the overall word list of the wiki-small dataset was passed as an argument createPorterStemmerClass() function which returns the stem classes by parsing out all of the word list and placing them in a stem class if it matches them. The stem classes are passed as an argument to the function countCooccurence() to return a 2-D matrix of co-occurence of stem words of each class in a window size of 50.

```
'''
Created on Nov 9, 2017

@author: nauman
'''
from nltk.stem.porter import *
import codecs
import pickle
import os
from bs4 import BeautifulSoup
import codecs
from string import punctuation
import re

def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False
```

```
def checkDigits(word):
  return word.isalpha()


def createPorterStemmerClass(wordlist,stem_words):
    word_frequency_matrix = []
    for i in range(0,10):
        new = [stem_words[i]]
        word_frequency_matrix.append(new)
    stemmer = PorterStemmer()
    for plural in wordlist:
  containsDigits = checkDigits(plural)
  if containsDigits:
          stem = stemmer.stem(str(plural))
          if stem in stem_words:
                index = stem_words.index(stem)
                word_frequency_matrix[index].append(plural)
    return word_frequency_matrix

def countCooccurence(word_frequency_matrix):
    stem_cooccurence_matrix = []
    for i in range(0,10):
        new = []
        for j in range(0,len(word_frequency_matrix[i])):
            new.append(0)
```

```
        stem_cooccurence_matrix.append(new)

    exclude = set(punctuation)
    for root, dirs, files in os.walk("F:\Fall2017\InformationRetreival\Assignment2\en"):
        for file in files:
            if file.endswith(".html"):
                with codecs.open(root+"\\" + file, "r", "utf-8") as file_open:
                    count_range = [0,0,0,0,0,0,0,0,0,0]
                    flag_wordFound = [False,False,False,False,False,False,False,False,False,False]
                    temp_word = ["","","","","","","","","",""]
                    soup = BeautifulSoup(file_open,'html.parser')
                    for words in soup.findAll(text=True):
                        if words.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
                            continue

                        s = ''.join(ch for ch in str(words) if ch not in exclude)
                        words = s.split(" ")
                        for word in words:
                            if not RepresentsInt(word.rstrip()):
                                if re.search('[a-zA-Z]', word.rstrip()):

                                    for i in range(0,10):
```

```
                                        if word.rstrip().lower() in word_frequency_matrix[i]:
                                            if count_range[i] == 0:
                                                temp_word[i] = word.rstrip().lower()
                                                flag_wordFound[i] = True
                                            elif count_range[i] < 200 and flag_wordFound[i]:
                                                index_prev = word_frequency_matrix[i].index(temp_word[i])
                                                index_next = word_frequency_matrix[i].index(word.rstrip().lower())
                                                temp = stem_cooccurence_matrix[i][index_next]
                                                stem_cooccurence_matrix[index_prev][index_next] = temp +1
                                                temp = stem_cooccurence_matrix[i][index_prev]
                                                stem_cooccurence_matrix[index_prev][index_prev] = temp +1
                                                temp_word[i] = word.rstrip().lower()
                                                count_range[i] = 0
                                        elif count_range < 200:
                                            count_range[i] = count_range[i] + 1
```

```
                                        else:
                                            count_range[i] = 0
              flag_wordFound[i] = False

    print("WordsOver")
    return stem_cooccurence_matrix

def main():
    word_list =["canada","candidates","collections","competitive","composition","very", "couples","victoria", "weapon", "defence"]
    stemmer = PorterStemmer()
    stem_words = [stemmer.stem(plural) for plural in word_list]
    with open("word_unique.pkl", 'rb') as f:
        wordlist = pickle.load(f)
    word_frequency_matrix = createPorterStemmerClass(wordlist,stem_words)
    file = open ("StemClass.txt", "w")
    file.write("Intital Stem Class without Co-occurence" + "\n")
    for i in range(0,10):
  file.write("Stem Class: ")
        for j in range(0,len(word_frequency_matrix[i])):
            file.write(word_frequency_matrix[i][j] + "  ")
    file.write("\n")
    file.write("\nFinal Stem classes after considering co-occurence\n\n")

    stem_cooccurence_matrix = countCooccurence(word_frequency_matrix)
    for i in range(0,10):
```

```
        file.write("Stem Class \n" )
        for j in range(0,len(stem_cooccurence_matrix[i])):
            if stem_cooccurence_matrix[i][j]> 0:
                file.write(word_frequency_matrix[i][j] + "  ")
        file.write("\n")
    file.close()



main()
```

```
Intital Stem Class without Co-occurence

Stem Class: canada  canada  canadas

Stem Class: candid  candid  candidate  candidates  candidating

Stem Class: collect  collect  collectable  collected  collectible  collectibles  collect

Stem Class: competit  competiting  competition  competitions  competitive  competitively

Stem Class: composit  composite  composites  compositing  composition  compositional  co

Stem Class: veri  veri  very

Stem Class: coupl  couple  coupled  couples  coupling  couplings

Stem Class: victoria  victoria  victoriae  victorias

Stem Class: weapon  weapon  weapons

Stem Class: defenc  defence  defences


Final Stem classes after considering co-occurence
```

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

Stem Class

7.2.2 Results

The first stem result is for stems class generated by parsing the word list of the wiki-small dataset. It does not account for co-occurence of the words in a stem class. The second output is empty, as none of the words in each stem class lied in a window size of 50,100 or 200.

Chapter 8

Problem MLN2

8.1 Problem Statement

Using the small wikipedia example, choose 10 words and compute MIM, EMIM, chi square, dice association measures for full document & 5 word windows (cf. pp. 203-205).

8.2 Solution

8.2.1 Description of Code

The code uses a random list of words extracted from the word list generated by parsing out the wiki-small dataset. The code for parsing words from wiki-small dataset has been reused from problem 6.2 and is not part of the code for this problem. I extracted the random words keeping an assumption that their frequencies must be greater than 100, so that I can get frequency values for the extracted words in a window size of five. I have provided constant values for the extracted word list, their frequencies and the totoal word list size of the wiki-small dataset. The extracted word list is passed as an argument to my function findWordOccurence() which creates a 10*10 matrix for storing frequency of two words occuring in a window size of five. The 2-D matrix of frequency, frequency of each extracted word list and total word count for wiki-small dataset have been used to calculate Dice Co-efficient, MIM, EMIM, Chi-square for each pair of words in the extracted word list. The formulas for calculating each measure have been extracted from our text book.

```
'''
Created on Nov 9, 2017

@author: nauman
'''
import os
from bs4 import BeautifulSoup
import codecs
from string import punctuation
import re
import math

def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
```

```
        return False

def findWordOccurence(word_list):
    ####Create a 2D matrix for word frequency
    word_frequency_matrix = []
    for i in range(0,10):
        new = []
        for j in range(0,10):
            new.append(0)
        word_frequency_matrix.append(new)

    exclude = set(punctuation)
    for root, dirs, files in os.walk("F:\Fall2017\InformationRetreival\Assignment2\en"):
        for file in files:
            if file.endswith(".html"):
                with codecs.open(root+"\\" + file, "r", "utf-8") as file_open:
                    count_range = 0
                    flag_wordFound = False
                    soup = BeautifulSoup(file_open,'html.parser')
                    for words in soup.findAll(text=True):
                        if words.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
                            continue

```

```
                    s = ''.join(ch for ch in str(words) if ch not in exclude)
                    words = s.split(" ")
                    for word in words:
                        if not RepresentsInt(word.rstrip()):
                            if re.search('[a-zA-Z]', word.rstrip()):

                                if word.rstrip().lower() in word_list and count_range == 0:
                                    flag_wordFound = True
                                    temp_word = word.rstrip().lower()
                                elif word.rstrip().lower() in word_list and count_range <= 5:
                                    index_temp = word_list.index(temp_word)
                                    index_next = word_list.index(word.rstrip().lower())
                                    temp_frequency = word_frequency_matrix[index_temp][index_next]
                                    word_frequency_matrix[index_temp][index_next] = temp_frequency+1
                                    count_range = 0
                                    temp_word = word.rstrip().lower()
                                elif count_range < 5 and flag_wordFound:
                                    count_range = count_range+1
                                else:
```

```
                    flag_wordFound = False
                    count_range = 0

    print("WordsOver")
    return word_frequency_matrix

def findDiceCoefficient(word_frequency_matrix, word_frequency_list):
    dice_coefficient = []
    for i in range(0,len(word_frequency_list)-1):
        for j in range (i+1,len(word_frequency_list)):
            dice_temp = (word_frequency_matrix[i][j] +word_frequency_matrix[j][i] )/(word_frequency_list[i]+word_frequency_list[j])
            dice_coefficient.append(dice_temp)
    return dice_coefficient

def findMutualInforamtion(word_frequency_matrix, word_frequency_list):
    mim = []
    for i in range(0,len(word_frequency_list)-1):
        for j in range (i+1,len(word_frequency_list)):
            mim_temp = (word_frequency_matrix[i][j] +word_frequency_matrix[j][i] )/(word_frequency_list[i]*word_frequency_list[j])
            mim.append(mim_temp)
    return mim

def findExpectedMutualInforamtion(word_frequency_matrix, word_frequency_list, N):
```

```
    emim = []
    for i in range(0,len(word_frequency_list)-1):
        for j in range (i+1,len(word_frequency_list)):
            nab = word_frequency_matrix[i][j] +word_frequency_matrix[j][i]
            if nab>0:
                emim_temp = nab* math.log((N*(nab/(word_frequency_list[i]*word_frequency_list[j]))))
            else:
                emim_temp = 0
            emim.append(emim_temp)
    return emim

def findChiSquare(word_frequency_matrix, word_frequency_list, N):
    chi_square = []
    for i in range(0,len(word_frequency_list)-1):
        for j in range (i+1,len(word_frequency_list)):
            nab = word_frequency_matrix[i][j] +word_frequency_matrix[j][i]
            x = nab-((word_frequency_list[i]*word_frequency_list[j])/N)
            chi_square_temp =  math.pow(x,2)/ (word_frequency_list[i]*word_frequency_list[j])
            chi_square.append(chi_square_temp)
    return chi_square

def main():
    N = 3902115
    word_frequency_list = [1090,106,105,104,104,1062,108,510,100,210]
```

```
    word_list =["canada","candidates","collections","competitive","composition
","very", "couples","victoria", "weapon", "defence"]
    word_frequency_matrix = findWordOccurence(word_list)

    for i in range(0,10):
        for j in range(0,10):
            print(word_frequency_matrix[i][j])
    diceCoefficient = findDiceCoefficient(word_frequency_matrix,
word_frequency_list)
    mim = findMutualInforamtion(word_frequency_matrix, word_frequency_list)
    emim = findExpectedMutualInforamtion(word_frequency_matrix,
word_frequency_list, N)
    chiSquare = findChiSquare(word_frequency_matrix, word_frequency_list, N)
    file = open("Result.txt", "w")
    file.write("Query Term1"+ " " + "Query Term2" + " " + "Dice Coefficient" +
 " " +"MIM" + " " + "EMIM" + " " + "Chi-Square" + "\n")
    count =0;
    for i in range(0,len(word_list)-1):
        for j in range (i+1,len(word_list)):
            file.write(word_list[i] + " " + word_list[j] + " " + str(
diceCoefficient[count]) + " " + str(mim[count]) + " " + str(emim[count]) +
 " " + str(chiSquare[count]) + "\n")
            count = count+1
    file.close()
    print("End")

```

```
main()
```

```
94 0 0 0 0 2 0 3 0 0

0 3 0 0 0 0 0 0 0 0

0 0 4 0 0 0 0 0 0 0

0 0 0 2 0 2 0 0 0 0

0 0 0 0 3 0 0 0 0 0

0 0 0 0 0 13 0 0 0 0

0 0 0 0 0 0 0 2 0 0

3 0 0 0 0 1 1 48 0 0

0 0 0 0 0 0 0 0 3 0

0 0 0 0 0 0 0 0 0 1


QueryTerm1 QueryTerm2 DiceCoefficient MIM EMIM ChiSquare

canada candidates 0.0 0.0 0 7.588085825438693e-09

canada collections 0.0 0.0 0 7.516500110104367e-09

canada competitive 0.0 0.0 0 7.444914394770038e-09

canada composition 0.0 0.0 0 7.444914394770038e-09

canada very 0.0009293680297397769 1.727742359059417e-06 3.81666855066276 2.5064236337055

canada couples 0.0 0.0 0 7.731257256107347e-09

canada victoria 0.00375 1.0793308148947653e-05 22.442670240498014 6.172110226621129e-05

canada weapon 0.0 0.0 0 7.158571533432729e-09
```

canada defence 0.0 0.0 0 1.5033000220208733e-08

candidates collections 0.0 0.0 0 7.309623960284979e-10

candidates competitive 0.0 0.0 0 7.240008493996552e-10

candidates composition 0.0 0.0 0 7.240008493996552e-10

candidates very 0.0 0.0 0 7.393162519831094e-09

candidates couples 0.0 0.0 0 7.518470359150264e-10

candidates victoria 0.0 0.0 0 3.550388780709847e-09

candidates weapon 0.0 0.0 0 6.961546628842838e-10

candidates defence 0.0 0.0 0 1.4619247920569958e-09

collections competitive 0.0 0.0 0 7.171706527072056e-10

collections composition 0.0 0.0 0 7.171706527072056e-10

collections very 0.0 0.0 0 7.323415703606272e-09

collections couples 0.0 0.0 0 7.447541393497904e-10

collections victoria 0.0 0.0 0 3.516894546929565e-09

collections weapon 0.0 0.0 0 6.895871660646208e-10

collections defence 0.0 0.0 0 1.4481330487357034e-09

competitive composition 0.0 0.0 0 7.103404560147559e-10

competitive very 0.0017152658662092624 1.810806895552658e-05 8.515752702826394 3.5198306

competitive couples 0.0 0.0 0 7.376612427845543e-10

competitive victoria 0.0 0.0 0 3.483400313149284e-09

competitive weapon 0.0 0.0 0 6.830196692449576e-10

```
competitive defence 0.0 0.0 0 1.4343413054144112e-09

composition very 0.0 0.0 0 7.25366888738145e-09

composition couples 0.0 0.0 0 7.376612427845543e-10

composition victoria 0.0 0.0 0 3.483400313149284e-09

composition weapon 0.0 0.0 0 6.830196692449576e-10

composition defence 0.0 0.0 0 1.4343413054144112e-09

very couples 0.0 0.0 0 7.532656152280738e-09

very victoria 0.0006361323155216285 1.8463129131125143e-06 1.9747093442762524 1.36934123

very weapon 0.0 0.0 0 6.974681622482164e-09

very defence 0.0 0.0 0 1.4646831407212545e-08

couples victoria 0.0048543689320388345 5.446623093681917e-05 16.07729882286608 0.0001618

couples weapon 0.0 0.0 0 7.0928965652361e-10

couples defence 0.0 0.0 0 1.4895082786995808e-09

victoria weapon 0.0 0.0 0 3.349423378028158e-09

victoria defence 0.0 0.0 0 7.03378909385913e-09

weapon defence 0.0 0.0 0 1.3791743321292416e-09
```

8.2.2 Results

N = 3902115

word_list =["canada","candidates","collections","competitive","composition","very", "couples","victoria", "weapon", "defence"] word_frequency_list = [1090,106,105,104,104,1062,108,510,100,210]

The above three lines show the values that have been provided to the code based on my previous problem. First line contains N which represents the total number of words in the wiki-small dataset. Second line contains the extarcted word list and the last line has all the frequency values for each extarcted word list.The file for results shows values a 10*10 matrix which is the frequency of co-ocurrence of two query terms in a window size of 5. Further the file shows for every two query terms their values in different measure. The results queries have been shown only for queries which have non-zero values for each measure of query expansion. The list of frequent occuring query terms in a window size of 5 which can be used for query expansion are:

canada very

canada victoria

competitive very

very victoria

couples victoria