

Old Dominion University
Department of Computer Science
CS834: Introduction to Information Retrieval
Fall 2017
Assignment 3
Professor: Dr. Michael Nelson

Hussam Hallak
CS Master's Student

November 10, 2017

Contents

List of Figures

1	Running correct.py	4
2	Google Web Translation for hussam.us	8
3	Google Web Translation for odu.edu	9
4	Google Web Translation for Dr. Nelson	10
5	Google Web Translation for Dr. Weigle	10
6	Google Web Translation for Mr. Kennedy	11
7	Euclidean Distance Similarity	12
8	Euclidean Distance Similarity Between Unit Vectors	13
9	Euclidean Distance Similarity Between Unit Vectors	13

List of Tables

Question 1:

Exercise 6.2:

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).

Answer:

Summary of the noisy channel model approach:

Let M be the misspelled word we want to correct.

Let W be the array of possible corrections to M where $W = \{w_i | i = 1, 2, 3, \dots, n\}$.

Computing conditional probability CP for each possible correct word, element in W , gives us: $CP_i = P(M|w_i) \times P(w_i)$ for $i = 1, 2, 3, \dots, n$

Choose w_i that maximizes CP_i

Implementation:

I found an example code on the web and modified it as the question suggested. The program file is named “correct.py”. It takes one command line argument, the word we are trying to correct, and it prints out the correct spelling and exits. The program, by design, will have to be launched again to correct another word.

```
1 import sys
2 import re
3 from collections import Counter
4
5 class Correct:
6     def __init__(self):
7         self.document = open('big.txt').read()
8         self.words()
9         self.count_words()
10
11     def correction(self, word):
12         "Most probable spelling correction for word."
13         word = word.lower()
14         candidates = self.candidates(word)
15         return max(candidates, key=self.P)
16
17     def candidates(self, word):
18         "Generate possible spelling corrections for word."
19         word = word.lower()
20         return (self.known([word]) or self.known(self.edits1(word)) or self.known(self.edits2(word)) or [word])
21
22     def known(self, words):
23         "The subset of 'words' that appear in the dictionary of WORDS."
24         return set(w for w in words if w in self.word_count)
25
26     def edits1(self, word):
27         "All edits that are one edit away from 'word'."
28         word = word.lower()
29         letters = 'abcdefghijklmnopqrstuvwxyz'
30         splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
31         deletes = [L + R[1:] for L, R in splits if R]
32         transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
33         replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
34         inserts = [L + c + R for L, R in splits for c in letters]
35         return set(deletes + transposes + replaces + inserts)
36
37     def edits2(self, word):
38         "All edits that are two edits away from 'word'."
39         word = word.lower()
40         return (e2 for e1 in self.edits1(word) for e2 in self.edits1(e1))
```

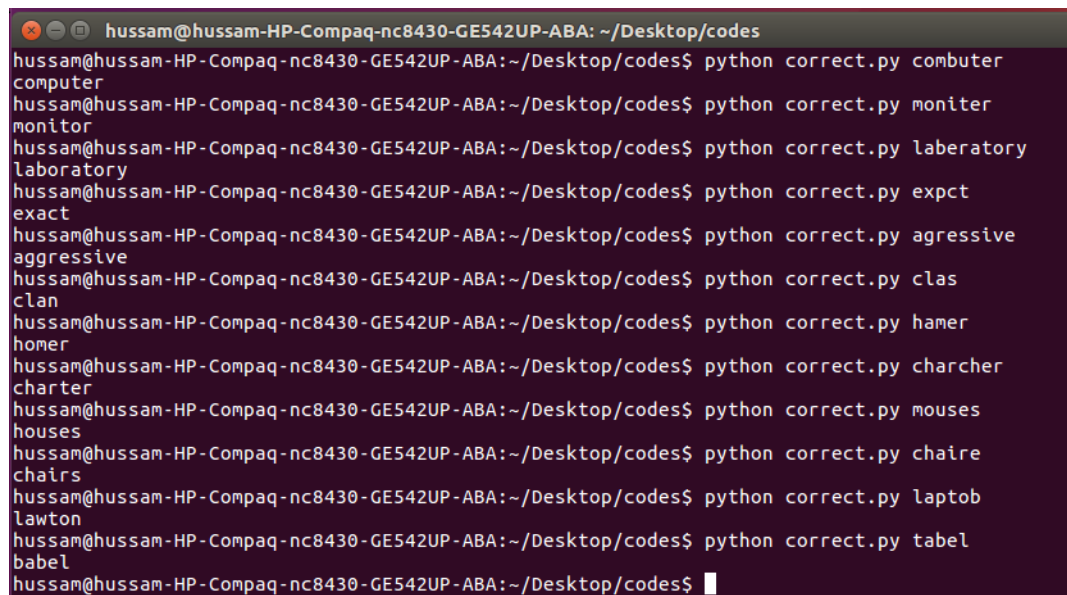
```

41
42 def words(self):
43     self.words = re.findall(r'\w+', self.document.lower())
44
45 def count_words(self):
46     self.word_count = Counter(self.words)
47
48 def P(self, word):
49     "Probability of 'word'."
50     word = word.lower()
51     N = sum(self.word_count.values())
52     return self.word_count[word] / N
53
54
55 if __name__ == '__main__':
56     correct = Correct()
57
58     if len(sys.argv) != 2:
59         print "Usage: python correct.py <word>"
60     print "Example: python correct.py camputer"
61     exit()
62
63     word = sys.argv[1]
64     corrected = correct.correction(word)
65
66     print corrected

```

Listing 1: The content of correct.py

Figure 1: Running correct.py



```

hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA: ~/Desktop/codes
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py combuter
computer
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py moniter
monitor
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py laberatory
laboratory
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py expct
exact
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py agressive
aggressive
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py clas
clan
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py hamer
homer
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py charcher
charter
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py mouses
houses
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py chaire
chairs
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py lapto
lawton
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$ python correct.py tabel
babel
hussam@hussam-HP-Compaq-nc8430-GE542UP-ABA:~/Desktop/codes$

```

Question 2:

Exercise 6.4:

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.

Answer:

The easiest way to detect place names or locations in a query is to tokenize the query and run all tokens against the place names dictionary, gazetteer. This method would be computationally expensive if the gazetteer has a large number of place names. It will sometimes fail, generate some false positives. For example:

Query: Virginia Woolf

The term “Virginia” will be falsely identified as a place name, but the user meant Virginia Woolf, the writer.

A better, less expensive, algorithm can be implemented by defining patterns that will help detecting place names. These patterns can be defined using prepositions or other words that are used with places in English language. Examples of these prepositions and words include: From, in, near, to, of, ...etc.

Pattern: * + ('in' || 'from' || 'near' || 'to' || 'of') + < *placename* >

Example queries:

Celebrities born in Virginia

When did MalcomX go to Egypt?

Universities in Hampton

Homes for sale near Suffolk

Is Michael Nelson from Norfolk?

Does Syria lie north of Jordan?

The approach would begin by scanning the query to find these words or prepositions from, in, to, ...etc. If one of the words is detected, only the following word is run through the gazetteer. This method will sometimes fail, generate some false positives as well as false negatives.

False positive example:

Query: A recipe for a pie made out of avocado

The term “avocado” will be falsely detected as a place name, Avocado, California. However, the user meant avocado, the fruit.

False negative example:

Query: Where is Avocado located in California?

Unless the defined patterns are advanced enough to detect place names in such queries, “Avocado”, the place in California will not be detected.

Assuming that the defined patterns are advanced enough to identify all possible place names, practically impossible, a large amount of false positives will be generated.

Pattern: < *placename* > + *location*

Query 1: Washington location on the map

Query 2: George Washington location of birth

In query 1, the term “Washington” will be detected as a place name, which is true. However, the same term “Washington” in query 2 will also be identified as a place name, which is false.

Also using patterns will have to be accompanied by a rule to handle single term queries. This will also generate a large number of false positives. For example:

Query: Lincoln

The term “Lincoln” will be identified as the place name Lincoln in Alabama, while the user’s intention was to look up Lincoln, the president, or Lincoln, the car, or Lincoln, the welding machine.

One more rule must be added to the patterns to handle place names that are not uni-grams. Examples include place names like “Virginia Beach”, “Newport News”, “Washington DC”, ...etc.

Example:

Pattern: $* + ('in' || 'from' || 'near' || 'to' || 'of') + < placename >$

Query: Hospitals in San Francisco

A rule to handle queries similar to the query above must be added. The first word of each place name that is n-gram where $n > 1$ must be added to the gazetteer and then mapped to the next term in the place name.

Here is my simple algorithm/implementation to detect place names in python-like code:

```
1 def gazetteerLookup(term):
2     placeNames = []
3     placeNames.append('Norfolk')
4     placeNames.append('Suffolk')
5     placeNames.append('Chesapeake')
6     .
7     .
8     .
9     if term in placeNames:
10         return True
11     else:
12         return False
13
14 def getPlaceNames(query):
15     result = []
16     tokens = query.split()
17     preps = ['in', 'from', 'near', 'to', 'of']
18     for i in range(0, len(tokens)):
19         for prep in preps:
20             if prep == tokens[i]:
21                 if gazetteerLookup(tokens[i+1]):
22                     result.append(tokens[i+1])
23     return result
```

Listing 2: The content of placenames.txt

Success example:

Query: Schools in Norfolk

Failure example:

Query: Casinos in Las Vegas

Question 3:

Exercise 6.5: 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. .

Answer:

Approach:

I found the snippet generation code for Galago in Github. The file “SnippetGenerator.java” can be downloaded [here](#).

Galago’s snippet generation algorithm is rather simple. It can be summarized by the following steps:

1. Find words in the returned document that match the query words.
2. Divide the document into regions where the matching words were found including the five words before the matching word and the five words after it.
3. Merge overlapped regions.
4. Generate the snippet from these regions when the threshold for the snippet size is reached.

I noticed that the file actually has the code for generating multiple snippets, scoring them, then returning the snippet with the highest score. However, the comment above the code stated that this method is too slow to be useful.

Galago’s snippet generation algorithm will work well for pages with little text content as long as query terms has exact matches in the document. The problem starts when no matches are found!

Possible improvements:

1. Performing query expansion before searching for matches. Query expansion can be performed using terms from other returned documents.
2. Stemming the query terms and looking for words that share the same stems in the document.
3. If no matches were found in a small document, summarize the document and return the summary. Significant Terms could be utilized to generate a summary of the returned document.

Question 4:

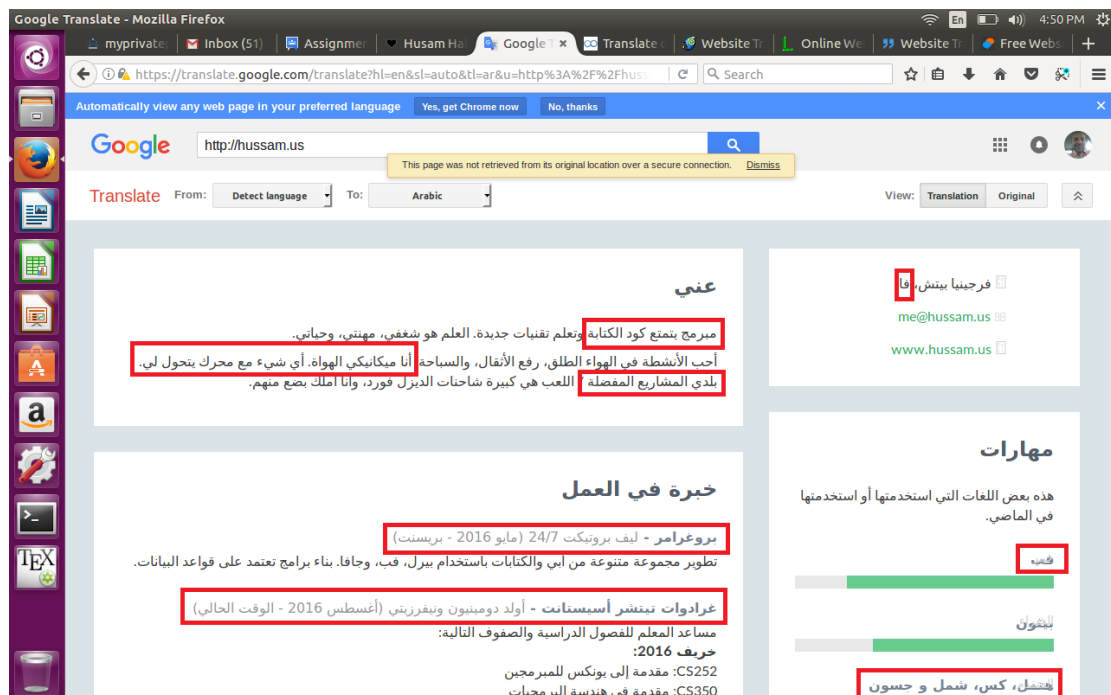
Exercise 6.9:

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

Answer:

The five web pages I have chosen are my personal page, ODU's home page, Dr. Nelson's home page, Dr. Weigle's home page, and Mr. Kennedy's home page.

Figure 2: Google Web Translation for hussam.us



I translated the pages using Google web pages translation service. The translation was from English to Arabic, my first language. I put a red rectangle around the parts where the translation is poor. It is clear than the translation was poor and inconsistent on all five web pages due to multiple reasons.

1. Acronym unknown to the translator such as CS, ODU, VA, PHP, HTML, ...etc
2. Capitalization: Sometimes "Spring" is translated as spring, the season in Arabic, and sometimes it is transliterated as if "Spring" is a name for a person or business, ...etc. The translation was also inconsistent in this case. The same happened to the word "Old" in "Old Dominion University"
3. Titles translation was always poor. The word "Programmer" was transliterated when it was in the title, but it was translated when it was in the paragraph.
4. Poor/informal English: One of the cases is my introduction. The first sentence is a fragment that ends with a period. I did not use first person and made it sound like someone else is introducing me to the visitor. "A programmer who enjoys writing code and learning new technologies." is a fragment ending with a period, which is probably what caused the translator to generate a very bad translation.

Figure 3: Google Web Translation for odu.edu



5. Words have multiple meanings depending on the context. The translator does not consider that when translating a web page. One example is the word “Publications” in Dr. Nelson’s page. The Arabic translation make it sound like it means published Facebook posts or tweets instead of published research papers. The reason is that the translator does not categorize the page before translating it. If Google classified pages before translating them, the translation would be much better. The word publications on a professor’s home page could not mean published social media posts. It sure means something like published research papers, other scholarly documents, or at least articles.

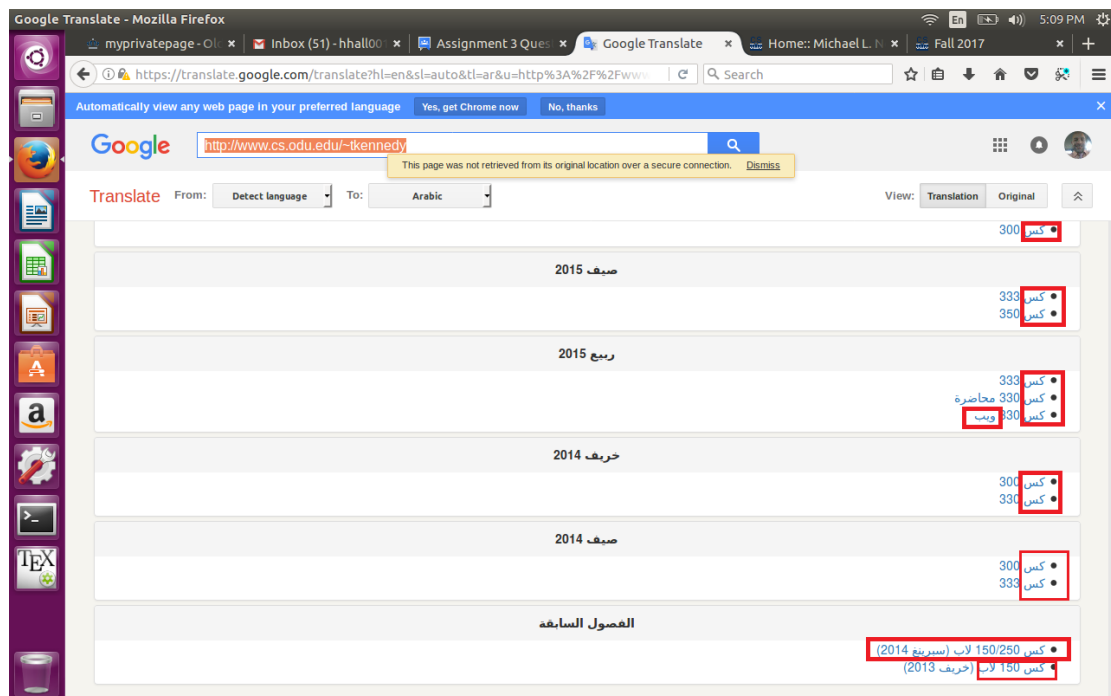
Figure 4: Google Web Translation for Dr. Nelson



Figure 5: Google Web Translation for Dr. Weigl



Figure 6: Google Web Translation for Mr. Kennedy



Question 5:

Exercise 7.2:

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 Rijsbergen's book).

Answer:

In order for the similarity measure to be correct and accurate when used in the vector space model, it must be based on the angle between the two vectors. Any measure that is based on the Euclidean distance will give wrong results unless both vectors, query & document vectors, are reduced to the unit vectors, which is what cosine similarity does.

Example:

We have a query q = "Do jealous people gossip?". Let's assume that the two terms "do" and "people" are stop words. The two terms that affect the results are "jealous" and "gossip".

We also have three documents to examine, d_1 , d_2 , and d_3 .

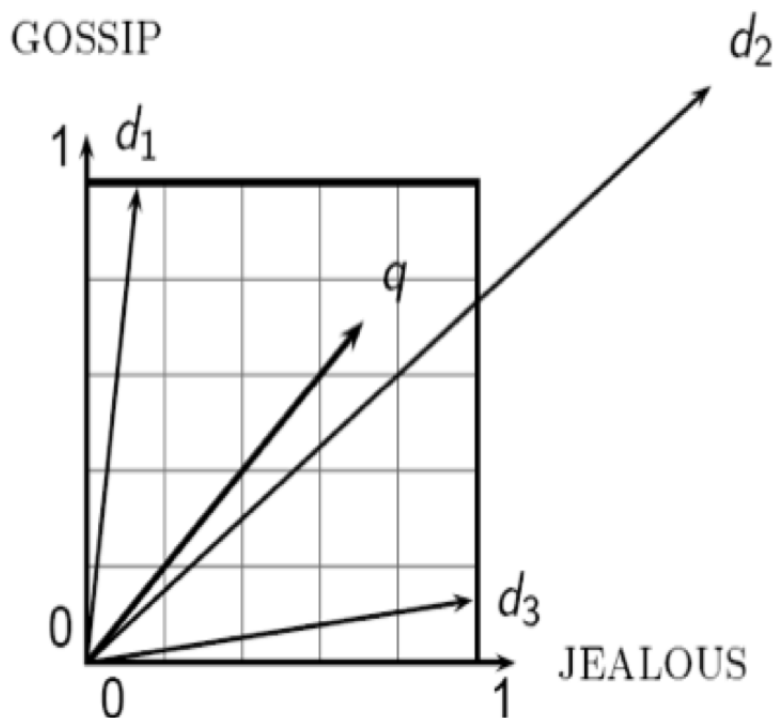
d_1 : A document that is mostly about gossip, but has a little about jealousy.

d_2 : A document about both gossip and jealousy. It has a little more about jealousy than it does about gossip. It is the most relevant document out of all three documents.

d_3 : A document that is mostly about jealousy, but has a little about gossip.

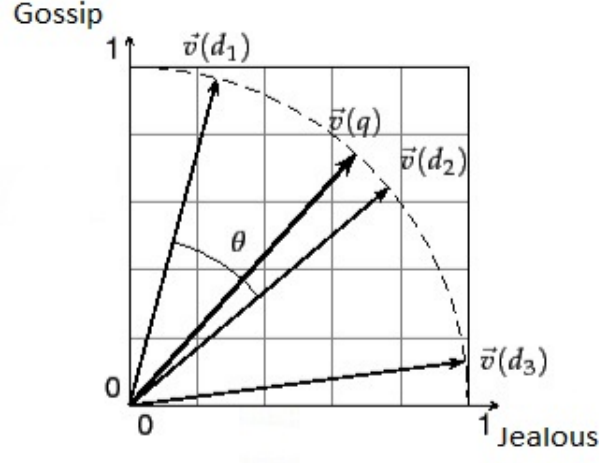
Euclidean distance similarity will cause d_1 and d_3 to be ranked higher than d_2 with respect to the query q because the distance between q and d_2 is larger than the distance between q and d_1 as well as the distance between q and d_3 . The result is clear in Figure 1

Figure 7: Euclidean Distance Similarity



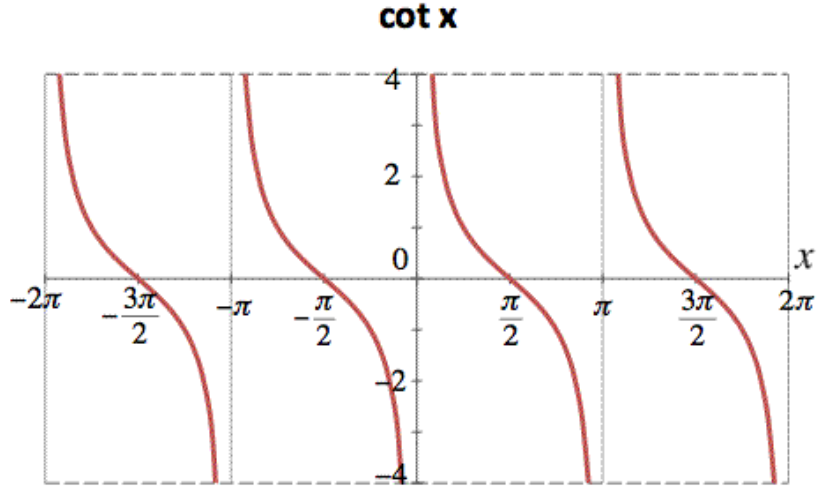
The only case where Euclidean distance can be used as a similarity measure is when all vectors are reduced to the unit vector. This reduction can be performed by dividing each component in the vector by the vector's magnitude. The result is clear in Figure 2

Figure 8: Euclidean Distance Similarity Between Unit Vectors



Cotangent of the angle between vectors could be used as a measure of similarity in the vector space model. Cotangent θ is the reciprocal of tangent θ . The graph of the function $y = \cot(x)$ is shown in Figure 3.

Figure 9: Euclidean Distance Similarity Between Unit Vectors



The period of the function cotangent that should be used lies in the interval $\theta = [0, \pi]$. From the graph, three statements can be made:

1. If the angle between the vectors is zero, the value of the cotangent is ∞ .
2. If the angle between the vectors is $\frac{\pi}{2}$, the value of the cotangent is zero.
3. If the angle between the vectors is π , the value of the cotangent is $-\infty$.

This similarity measure agrees with the cosine similarity measure, but it is very steep since its value spans the interval $(-\infty, +\infty)$.

The log function can be wrapped around the cotangent function to reduce its aggression.
 $similarity = \log_{10}(\cot(\theta))$

The base of the log is not important. Base 10 is not empirically determined. The base is there for completeness.

Question 6:

Exercise 7.5:

Implement a BM25 module for Galago. Show that it works and document it.

Answer:

I wrote the code for BM25 in Java since that is the language in which Galago is written. The file is named “BM25.java”. Some variables should be passed as arguments, but for the purpose of demonstration, I manually set them within the class BM25. Here is the code I wrote:

I downloaded the source code for Galago. BM25 is already implemented in the file “BM25Scorer.java”. I noticed that IDF is calculated using a different formula. Here is the formula used in their implementation:

```
1 idf = Math.log(documentCount / (df + 0.5));
```

Question 7:

Exercise 7.7:

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

Answer:

I want to build the definition of bigram, two terms, language model by explaining few concepts then giving some examples:

Language Model:

A topic in a document or query can be represented as a language model. Words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model. If I have a document that consists of 1000 words, and 50 of them are “coffee”, then the document is most likely to be about coffee. The word coffee has a high probability unless the corpus has 25,000,000 documents and the word “coffee” is in most of them, stop word.

Unigram Language Model:

It is the probability distribution over the words in a language. The generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them. In this model, the probability of each word only depends on that word’s own probability in the document. It splits the probabilities of different terms in a context, e.g., from

$$P(t_1 t_2 t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2)$$

to

$$P_{uni}(t_1 t_2 t_3) = P(t_1)P(t_2)P(t_3)$$

Bigram Language Model:

Some applications use bigram language model where probabilities depend on the previous word. It is assumed that the probability of observing the word w_2 in the context history of the preceding word can be approximated by the probability of observing it in the shortened context history of the preceding word.

Example:

In a bigram language model, the probability of the sentence “I lived in Virginia Beach” is approximated as:

$$P_{bi}(I, lived, in, Virginia, Beach) \approx$$

$$P(I | < s >) P(lived | I) P(in | lived) P(Virginia | in) P(Beach | Virginia) P(< /s > | Beach)$$

Where:

$< s >$ denotes the beginning of the sentence.

$< /s >$ denotes the end of the sentence.

In this example, the probability of the word “Beach” being pulled out of the “bucket” given that the previous word is “Virginia” is higher than its probability in a unigram language model where context is not considered.

Question 8:

Exercise 7.8: Using the Galago implementation of query likelihood, study the impact of short queries and long queries on effectiveness. Do the parameter settings make a difference?

Answer:

After doing some search for Galago query likelihood, I was able to find some useful information on the wiki pages for Galago on Sourceforge.

Galago uses Latent Dirichlet allocation model, LDA, which is a way of automatically discovering topics that queries belong to.

The impact of short queries and long queries is that short queries can be difficult to find out which topics they contain because of their ambiguity due to being short.

Long queries should not cause a problem as they will have enough terms to determine which topics they contain.

References

- [1] Stackoverflow. <https://stackoverflow.com/questions/tagged/python>.
- [2] <http://norvig.com/spell-correct.html>
- [3] <https://sourceforge.net/p/lemur/wiki/Galago%20Query%20Language/>
- [4] <https://sourceforge.net/p/lemur/wiki/Galago%20Query%20Language/?version=2>
- [5] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation