Assignment 3

CS834-F16: Introduction to Information Retrieval

Fall 2016

Erika Siregar

CS Department - Old Dominion University

November 10, 2016

Question 6.1

Using the Wikipedia collection provided at the book website, create a sample of stem clusters by the following process:

1. Index the collection without stemming.
2. Identify the first 1,000 words (in alphabetical order) in the index.
3. Create stem classes by stemming these 1,000 words and recording which words become the same stem.
4. Compute association measures (Dice's coefficient) between all pairs of stems in each stem class. Compute co-occurrence at the document level.
5. Create stem clusters by thresholding the association measure. All terms that are still connected to each other form the clusters.

Compare the stem clusters to the stem classes in terms of size and the quality (in your opinion) of the groupings.

Answer

For indexing the collection, I modified the code that I used in assignment 2, sort the index alphabetically, and take the first 1000 words. The code for this is written in the file '1_2_index.py'. For creating the stem classes, I use Krovetz Stemmer because it produces a better stemming results than Porter Stemmer. Listing 1 shows the code to create the stem classes that utilize python library for Krovetz Stemmer [1].

```
#!/usr/bin/python
import json
from krovetzstemmer import Stemmer as KrovetzStemmer
import unicodecsv as csv
from prettyprint import prettyprint


# Instantiate krovetz stemmer
krovetz = KrovetzStemmer()


# Read result of 1_index
with open('1_2_index.txt', 'rb') as f:
    str_word_files_index = f.read()
    word_files_index =  json.loads(str_word_files_index)

    stem_word_index = {}
    for word, files in word_files_index.items():
        # Stem word using krovetz
        stemmed_word = krovetz.stem(word)

        # Group by stemmed word
        stem_word_index.setdefault(stemmed_word, [])
        stem_word_index[stemmed_word].append(word)


    filename = '3_stemmed_words.csv'
    with open(filename, 'wb') as f:
        print('Writing to file {}'.format(filename))

        writer = csv.writer(f)
        for stemmed_word, words in stem_word_index.items():
            writer.writerow((stemmed_word, ', '.join(words)))

        print('Done!')
```

Listing 1: Creating Stem Classes with Krovetz Stemmer

Table 1 shows the snippet of the stem classes created. The complete list of the stem classes is available in '3_stemmed_words.csv' which is uploaded on github.

stem class	**terms**
academician	academicians, academician
adamant	adamantly, adamant
abundance	abundance
account	account, accounted, accounts, accounting
abdelkader	abdelkader
achter	achter
abednego	abednego
abortion	abortion, abortions
aboot	aboot
abrahamsson	abrahamsson
abdeali	abdeali
abandon	abandonment, abandon, abandoning

Table 1: A snippet of the stem classes

Next step is to create the stem clusters using Dice's Coefficient [2] as the term association measure. Dice's Coefficient works based on this formula:

$$2.\frac{n_{ab}}{n_a + n_b}$$

Using this formula, compute the Dice's Coefficient for each pair of terms in every stem classes. With a threshold $= 0.01$, create a graph in which every pair of terms that has Dice's Coefficient greater than 0.01 will be connected with an edge. Figure 1 shows the graph for the stem class 'activate' that can be grouped into two clusters: '*activate, activating, activator*' and '*activates, activation*'. The graph for other stem classes are available on github in a folder named *'graph'*. From this graph, we only need to extract the connected components to form the clusters.

Listing 3 shows the code used to compute the Dice's Coefficient, create the graphs, and extract the connected components of the graphs.

```

#!/usr/bin/python
import json
import nltk as nltk
from tabulate import tabulate
import unicodecsv as csv
from prettyprint import prettyprint
import networkx as nx
import matplotlib.pyplot as plt

dice_coef_threshold = 0.01
stem_clusters = []

```

```python
# Read result of 1_2_index.txt
with open('1_2_index.txt', 'rb') as f1:
    word_files_index = json.loads(f1.read())

    # Read result of 3_stemmed_words.csv
    with open('3_stemmed_words.csv', 'rb') as f3:
        for stemmed_word, words in csv.reader(f3):
            words = words.split(', ')

            # create bigrams from words
            bigrams = list(nltk.bigrams(words))
            for word_a, word_b in bigrams:
                # Lookup filename in word_files_index
                files_a = word_files_index[word_a]
                files_b = word_files_index[word_b]
                files_a_sliced_b = list(set(files_b) & set(files_a))

                dice_coef = float(2 * len(files_a_sliced_b)) / (len(files_a) + len(files_b))

                if(dice_coef > dice_coef_threshold):
                    stem_clusters.append((stemmed_word, word_a, word_b, dice_coef))


stem_clusters = sorted(stem_clusters, key=lambda x: x[3], reverse=True)
# print tabulate(stem_clusters, headers=['stemmed_word', 'word_a', 'word_b', 'dice_coef'])

filename = '4_dice_coeficient.csv'
with open(filename, 'wb') as f:
    print('Writing to file {}'.format(filename))

    writer = csv.writer(f)
    for stemmed_word, word_a, word_b, dice_coef in stem_clusters:
        writer.writerow((stemmed_word, word_a, word_b, dice_coef))


# Create graph
stemmed_word_data = {}
for stemmed_word, word_a, word_b, dice_coef in stem_clusters:
    stemmed_word_data.setdefault(stemmed_word, [])
    stemmed_word_data[stemmed_word].append((word_a, word_b, dice_coef))

stemmed_word_clusters = {}
for stemmed_word, data in stemmed_word_data.items():
    G=nx.MultiGraph()

    labels = {}
    for word_a, word_b, dice_coef in data:
        G.add_edge(word_a, word_b, weight=dice_coef, label=dice_coef)
        labels[(word_a, word_b)] = dice_coef

    # export connected components into list
    stemmed_word_clusters[stemmed_word] = list(nx.connected_components(G))

    nx.draw(G, with_labels=True)
    nx.draw_networkx_edge_labels(G, pos=nx.spring_layout(G), edge_labels=labels)

    filename = '4_graph_{}.png'.format(stemmed_word)
    print('Saving graph {}'.format(filename))
    plt.savefig(filename, format='PNG')
    plt.clf()

print('Draw graphics done!')

print('Print stem clusters...')

for stemmed_word, connected_nodes in stemmed_word_clusters.items():
    for connected_node in connected_nodes:
        print(u'{}\t: {}'.format(stemmed_word, ', '.join(connected_node)))

print('Print stem clusters done')
```

Listing 2: Creating Cluster using Dice's Coefficient

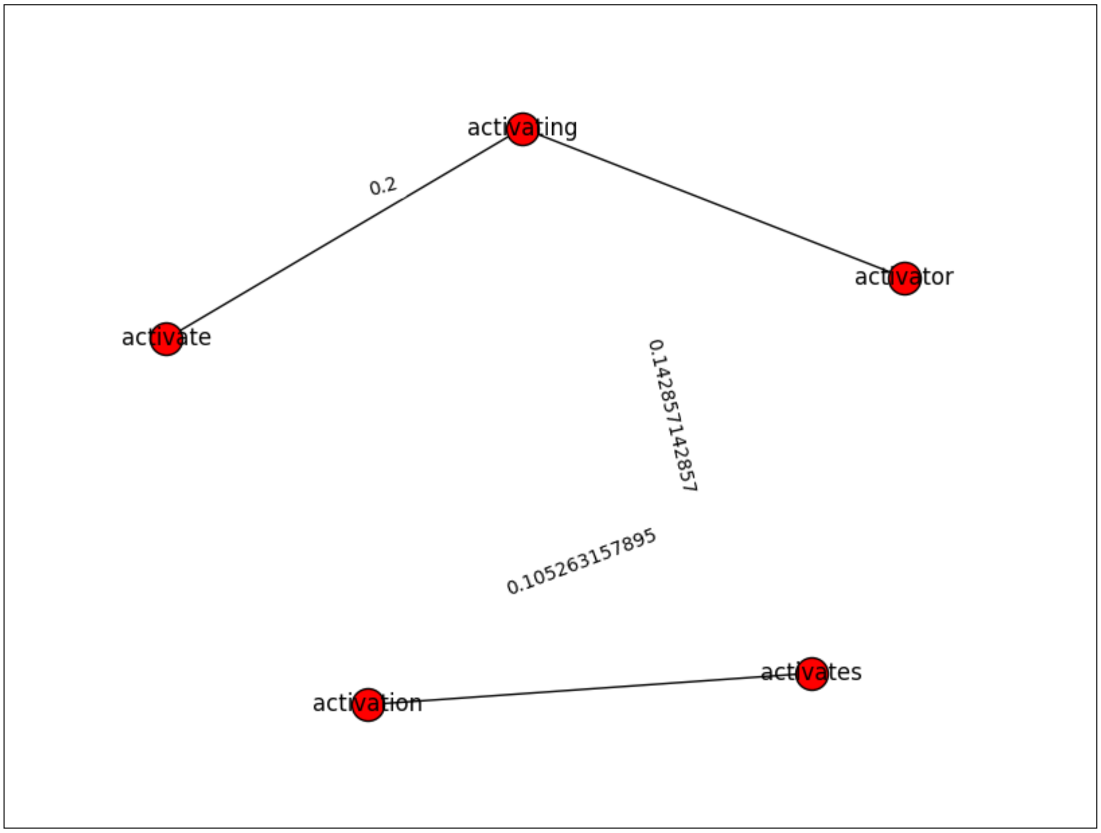


Figure 1: Graph of the connected component for the stem class ‘activate’

Table 2 shows the Dice's Coefficient for some pair of terms in the Wikipedia collection. The complete list of the term pairs with their Dice's Coefficient value is available on Github in a file named ‘4_dice_coeficient.csv’.

No	**Class**	**Term 1**	**Term 2**	**Dice's Coefficient**
1	abomination	abominationes	abomination	0.6666666667
2	abut	abuts	abutting	0.6666666667
3	abjad	abjads	abjad	0.6666666667
4	academician	academicians	academician	0.5714285714
5	aberration	aberrations	aberration	0.5
6	adapter	adapters	adapter	0.5
7	actor	actors	actor	0.4984025559
8	abridge	abridged	abridges	0.4
9	absolve	absolve	absolved	0.4
10	acoustic	acoustically	acoustical	0.4

Table 2: Dice's Coefficient for some pair of terms in small Wikipedia collection

Listing 3 shows the clusters resulted for the small Wikipedia Collection.

```
accessible  : accessible , accessibility
activate  : activator , activate , activating
activate  : activation , activates
accelerator : accelerators , accelerator
abridge : abridges , abridged
accurate  : accurate , accurately
address : addressing , addresses , addressed , address
abomination : abomination , abominationes
accept  : accepting , accepted , accept
abbreviation  : abbreviation , abbreviations
acclaim : acclaim , acclaimed
adaptation  : adaptations , adaptation
abrogate  : abrogation , abrogated
accrete : accreting , accrete , accreted
acid  : acidic , acids , acid
accommodate : accommodate, accommodated
absolute  : absoluter , absolute
acceleration  : acceleration , accelerations
additional  : additional , additionally
acknowledge : acknowledges, acknowledged
addition  : addition , additions
accent  : accent , accented
actor : actors , actor
access  : access , accessed , accessing , accessor
acyltransferase : acyltransferase , acyltransferases
add : adding, add, added, adds
activist  : activist , activists
adapt : adaption , adapt
adapt : adaptive , adapted
acre  : acres , acre
achieve : achieves , achieve
achieve : achieving , achieved
abstraction : abstraction , abstractions
accompany : accompany, accompanying
activity  : activities , activity
accidental  : accidentally , accidental
aberration  : aberrations , aberration
acronym : acronym, acronyms
academy : academy, academies
acquire : acquire , acquiring
```

```
academician : academician , academicians
abut  : abutting , abuts
abuse : abused , abuse
accompaniment : accompaniment , accompaniments
actress : actresses , actress
accuse  : accusing , accuses
acute : acute , acutely
accumulate  : accumulate , accumulated
abugida : abugidas , abugida
abduct  : abducted , abductors
achievement : achievements , achievement
accredit  : accrediting , accredited , accreditation
accusation  : accusation , accusations
account : accounting , accounted , accounts
accident  : accident , accidents
actual  : actual , actualized
adapter : adapter , adapters
accomplishment  : accomplishment , accomplishments
absolve : absolved , absolve
abjad : abjads , abjad
academic  : academic , academically
abrupt  : abrupt , abruptly
abolition : abolitionism , abolition
act : acted , act
action  : action , actions
acoustic  : acoustical , acoustic , acoustically
abbey : abbeys , abbey
acquisition : acquisitions , acquisition
abbot : abbots , abbot
```

Listing 3: Cluster for the small wikipedia collection

Question 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:

Spelling corrector based on the noisy channel model works using this approach [3]:

1. Let's say x is a mispelled word and $w = w_1, w_2, w_3, ..., w_n$ is an array of possible corrected words.
2. Our task is to compute the conditional probability and take a word w_i that has the maximum value for $P(x|w_i).P(w_i)$.

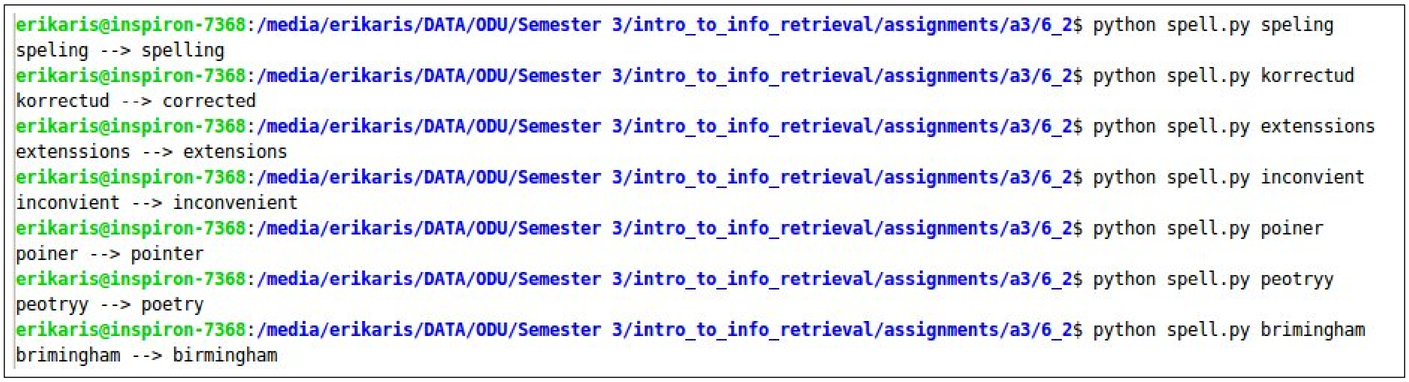
It looks complicated. But, fortunately, Peter Norvig [4] has provided a nice python code for spelling corrector, which is worked based on the noisy channel model. To compute the probability, Norvig uses word frequencies taken from a predefined dataset 'big.txt'. For this assignment, I modified Norvig's code as can be seen in listing 4.

```python

#!/usr/bin/python

import re
from collections import Counter

import sys


class Spell:
    def __init__(self, train_file):
        self.document = open(train_file).read()
        self.to_words()
        self.count_words()

    def to_words(self):
        self.words = re.findall(r'\w+', self.document.lower())

    def count_words(self):
        self.word_count = Counter(self.words)

    def probability(self, word):
        "Probability of `word`."
        word = word.lower()
        N = sum(self.word_count.values())
        return self.word_count[word] / N

    def edits1(self, word):
        "All edits that are one edit away from `word`."
        word = word.lower()
        letters = 'abcdefghijklmnopqrstuvwxyz'
        splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
        deletes = [L + R[1:] for L, R in splits if R]
        transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
        replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
        inserts = [L + c + R for L, R in splits for c in letters]
        return set(deletes + transposes + replaces + inserts)

    def edits2(self, word):
        "All edits that are two edits away from `word`."
        word = word.lower()
        return (e2 for e1 in self.edits1(word) for e2 in self.edits1(e1))

    def known(self, words):
        "The subset of `words` that appear in the dictionary of WORDS."
        return set(w for w in words if w in self.word_count)

    def candidates(self, word):
        "Generate possible spelling corrections for word."
        word = word.lower()
        return (self.known([word]) or self.known(self.edits1(word)) or self.known(self.edits2(word)) or [word])

    def correction(self, word):
        "Most probable spelling correction for word."
        word = word.lower()
        candidates = self.candidates(word)
        return max(candidates, key=self.probability)

```

```

if __name__ == '__main__':
    spell = Spell('big.txt')

    if len(sys.argv) != 2:
        print('python spell.py <word>')
        exit()

    word = sys.argv[1]
    spelled_word = spell.correction(word)

    print('{} --> {}'.format(word, spelled_word))
```

Listing 4: Spelling corrector

Figure 2 shows the example of spelling correction using some words taken from the textbook [5] and some words taken from Norvig's website [4].



```
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py speling
speling --> spelling
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py korrectud
korrectud --> corrected
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py extenssions
extenssions --> extensions
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py inconvient
inconvient --> inconvenient
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py poiner
poiner --> pointer
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py peotryy
peotryy --> poetry
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a3/6_2$ python spell.py brimingham
brimingham --> birmingham
```

Figure 2: Spelling corrector using noisy channel model with edit distance of 1 and 2

Question 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

For this question, I downloaded the source code for Galago version 3.10 [6] from https://sourceforge.net/p/lemur/galago/ci/release-3.10/tree/. It is a part of The Lemur Project [7]. The code for the snippet generator can be found in a file named *'SnippetGenerator.java'* under the directory 'galago-3.10/core/src/main/java/org/lemurproject/galago/core/index/corpus'. This is how the code works:

1. Given 'documentText' and a set of 'queryTerms', tokenize the 'documentText' into terms and its position. Listing 1 shows the code for this step.

```

public String getSnippet(String documentText, Set<String> queryTerms) throws
    IOException {
```

```
    ArrayList<IntSpan> positions = new ArrayList<IntSpan>();
    Document document = parseAsDocument(documentText, positions);
    return generateSnippet(document, positions, queryTerms);
  }
```

2. Stem each term using a defined stemmer. By default, Galago uses Krovetz stemmer.

```

  private Document parseAsDocument(String text, ArrayList<IntSpan> positions) throws IOException {
    Document document = new Document();
    document.text = text;

    // Tokenize the document
    TagTokenizer tokenizer = new TagTokenizer();
    tokenizer.process(document);

    if (positions != null) {
      positions.addAll(tokenizer.getTokenPositions());
    }
    if (stemming) {
      document = stemmer.stem(document);
    }

    return document;
  }
```

3. Iterate each stemmed term in documentText and find matches with each term in queryTerm.

4. For each matched term, make snippet region containing match term (original term) and maximum 5 terms before and 4 terms after original term in document. So, the snippet region will contain maximum 10 terms including the matched term.

```

Private ArrayList<SnippetRegion> findMatches(final Document document, final Set<String> queryTerms) {
    // Make a snippet region object for each term occurrence in the document,
    // while also counting matches
    ArrayList<SnippetRegion> regions = new ArrayList<SnippetRegion>();

    for (int i = 0; i < document.terms.size(); i++) {
      String term = document.terms.get(i);
      if (queryTerms.contains(term)) {
        regions.add(new SnippetRegion(term, i, width, document.terms.size()));
      }
    }
    return regions;
  }
```

5. Check the snippet regions and resolve if there are overlapped regions.

6. Remove snippet regions that overflow the maxSize. In the code, the maxSize is set to 40.

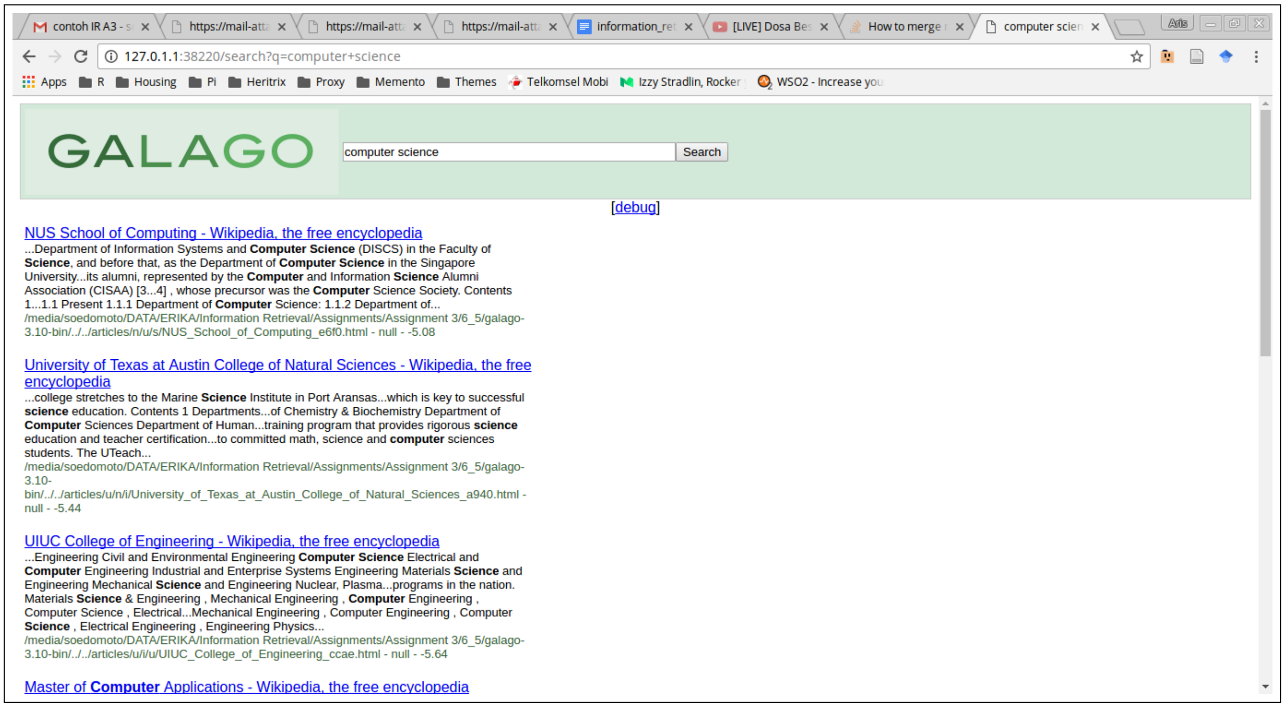
```

  public ArrayList<SnippetRegion> combineRegions(final ArrayList<SnippetRegion> regions) {
    ArrayList<SnippetRegion> finalRegions = new ArrayList<SnippetRegion>();
    SnippetRegion last = null;
    int snippetSize = 0;
    int maxSize = 40;

    for (SnippetRegion current : regions) {
      if (last == null) {
        last = current;
      } else if (last.overlap(current)) {
        SnippetRegion bigger = last.merge(current);

        if (bigger.size() + snippetSize > maxSize) {
          finalRegions.add(last);
          last = null;
        } else {
          last = bigger;
        }
      } else if (last.size() + snippetSize > maxSize) {
        break;
      } else {
        finalRegions.add(last);
        snippetSize += last.size();
        last = current;
      }
    }

    if (last != null && snippetSize + last.size() < maxSize) {
      finalRegions.add(last);
    }

    return finalRegions;
  }
```

7. Combine all snippet regions in each document into a single snippet splitted by '...' (three dots). Make the matched terms displayed in a bold format.

```
public String buildHtmlString(Snippet best, Document document, ArrayList<IntSpan> positions) {
  StringBuilder builder = new StringBuilder();

  for (SnippetRegion region : best.regions) {
    if (region.start != 0) {
      builder.append("...");
    }
    int startChar = positions.get(region.start).start;
    int endChar = positions.get(region.end - 1).end;
    int start = 0;

    // section string
    String section = document.text.substring(startChar, endChar);

    for (Match m : region.matches) {
      int startMatchChar = positions.get(m.start).start - startChar;
      int endMatchChar = positions.get(m.end - 1).end - startChar;

      String intermediate = stripTags(section.substring(start, startMatchChar));
      builder.append(intermediate);
      builder.append("<strong>");
      builder.append(stripTags(section.substring(startMatchChar, endMatchChar)));
      builder.append("</strong>");
```

```
        start = endMatchChar;
      }

      if (start >= 0) {
        builder.append(stripTags(section.substring(start)));
      }

      // terminate matches once we reached a max length.
      int maxSnippetSize = 500;
      if (builder.length() > maxSnippetSize) {
        break;
      }
    }

    if (best.regions.size() > 1 && best.regions.get(best.regions.size() - 1).end
     != document.terms.
            size()) {
      builder.append("...");
    }
    return builder.toString();
  }
```

Figure 3 shows the example of snippets generated for the query terms ‘computer science’.



NUS School of Computing - Wikipedia, the free encyclopedia
...Department of Information Systems and **Computer Science** (DISCS) in the Faculty of **Science**, and before that, as the Department of **Computer Science** in the Singapore University...its alumni, represented by the **Computer** and Information **Science** Alumni Association (CISAA) [3...4] , whose precursor was the **Computer** Science Society. Contents 1...1.1 Present 1.1.1 Department of **Computer** Science: 1.1.2 Department of...
/media/soedomoto/DATA/ERIKA/Information Retrieval/Assignments/Assignment 3/6_5/galago-3.10-bin/../../articles/n/u/s/NUS_School_of_Computing_e6f0.html - null - -5.08

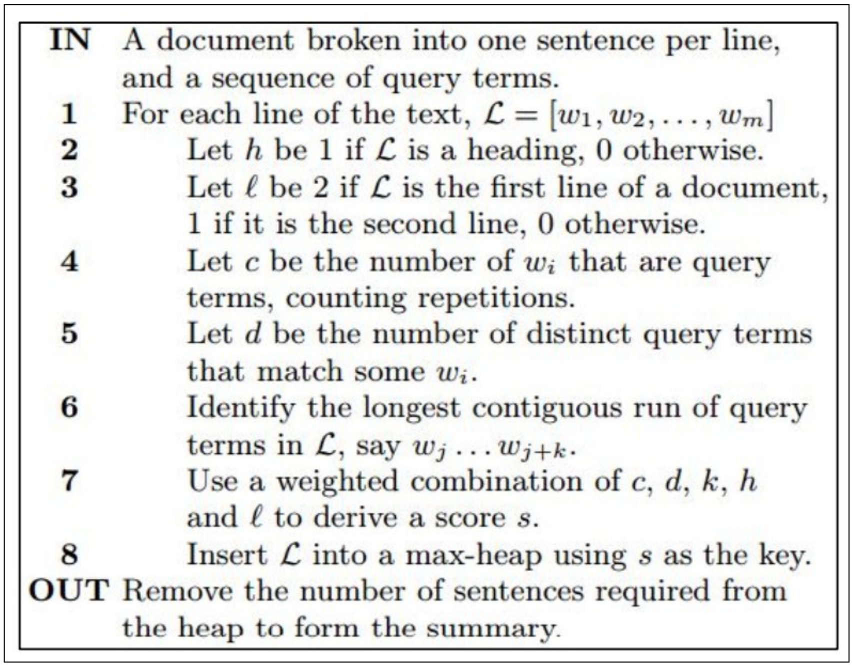
University of Texas at Austin College of Natural Sciences - Wikipedia, the free encyclopedia
...college stretches to the Marine **Science** Institute in Port Aransas...which is key to successful **science** education. Contents 1 Departments...of Chemistry & Biochemistry Department of **Computer** Sciences Department of Human...training program that provides rigorous **science** education and teacher certification...to committed math, science and **computer** sciences students. The UTeach...
/media/soedomoto/DATA/ERIKA/Information Retrieval/Assignments/Assignment 3/6_5/galago-3.10-bin/../../articles/u/n/i/University_of_Texas_at_Austin_College_of_Natural_Sciences_a940.html - null - -5.44

UIUC College of Engineering - Wikipedia, the free encyclopedia
...Engineering Civil and Environmental Engineering **Computer Science** Electrical and **Computer** Engineering Industrial and Enterprise Systems Engineering Materials **Science** and Engineering Mechanical **Science** and Engineering Nuclear, Plasma...programs in the nation. Materials **Science** & Engineering , Mechanical Engineering , **Computer** Engineering , Computer Science , Electrical...Mechanical Engineering , Computer Engineering , Computer **Science** , Electrical Engineering , Engineering Physics...
/media/soedomoto/DATA/ERIKA/Information Retrieval/Assignments/Assignment 3/6_5/galago-3.10-bin/../../articles/u/i/u/UIUC_College_of_Engineering_ccae.html - null - -5.64

Master of **Computer** Applications - Wikipedia, the free encyclopedia

Figure 3: Snippet generated by Galago for query terms ‘computer science’

Based on my analysis, I think Galago’s snippet algorithm will not work well for pages with little text content. It is because the algorithm works by matching the query terms with the page content. If there is only little content on the page, then the probability to generate a good snippet will also decrease. One thing that we can use to improve the algorithm is to adopt the algorithm explained by Turpin [8]. Turpin’s algorithm does not only depends on the ‘matched terms’. It also uses the combination of several weights to improve the rank of the sentences such as the longest contiguous run of query terms and the position of the sentences. Figure 4 illustrates the algorithm used by Turpin [8] to generate snippets.



```
IN   A document broken into one sentence per line,
     and a sequence of query terms.
1    For each line of the text, $\mathcal{L} = [w_1, w_2, \ldots, w_m]$
2        Let $h$ be 1 if $\mathcal{L}$ is a heading, 0 otherwise.
3        Let $\ell$ be 2 if $\mathcal{L}$ is the first line of a document,
         1 if it is the second line, 0 otherwise.
4        Let $c$ be the number of $w_i$ that are query
         terms, counting repetitions.
5        Let $d$ be the number of distinct query terms
         that match some $w_i$.
6        Identify the longest contiguous run of query
         terms in $\mathcal{L}$, say $w_j \ldots w_{j+k}$.
7        Use a weighted combination of $c$, $d$, $k$, $h$
         and $\ell$ to derive a score $s$.
8        Insert $\mathcal{L}$ into a max-heap using $s$ as the key.
OUT  Remove the number of sentences required from
     the heap to form the summary.
```

Figure 4: Turpin's algorithm for ranking the sentences. Adapted from [8]

Question MLN1

`MLN1: using the small wikipedia example, choose 10 words and create stem classes as per the algorithm on pp. 191-192.`

Answer

The steps to solve this problem is similar to what I have done in question 6.1. There are some differences:

1. There are 4 metrics that are use to measure the association between the terms, which are Dice's Coefficient, EMIM, MIM, and Chi-square.

2. Window size = 100 words

3. Ten chosen words, which are *daughter, army, provide, type, feature, free, standard, street, fin, and unit.*

Listing 5 shows the python code for this problem.

```

#!/usr/bin/python
import io
import math
import os
```

```python
from krovetzstemmer import Stemmer as KrovetzStemmer

import html2text
import matplotlib.pyplot as plt
import networkx as nx
import nltk


# Create graph function
def create_graph(stem_data, metric_name):
    stemmed_word_data = {}
    for stemmed_word, word_a, word_b, coef in stem_data:
        stemmed_word_data.setdefault(stemmed_word, [])
        stemmed_word_data[stemmed_word].append((word_a, word_b, coef))

    stemmed_word_clusters = {}
    for stemmed_word, data in stemmed_word_data.items():
        G = nx.MultiGraph()

        labels = {}
        for word_a, word_b, coef in data:
            G.add_edge(word_a, word_b, weight=coef, label=coef)
            labels[(word_a, word_b)] = coef

        # export connected components into list
        stemmed_word_clusters[stemmed_word] = list(nx.connected_components(G))

        nx.draw(G, with_labels=True)
        nx.draw_networkx_edge_labels(G, pos=nx.spring_layout(G), edge_labels=labels)

        filename = '{}_graph_{}.png'.format(metric_name, stemmed_word)
        print('Saving graph {}'.format(filename))
        plt.savefig(filename, format='PNG')
        plt.clf()

    return stemmed_word_clusters


# Print stem cluster function
def print_stem_cluster(stemmed_word_clusters, metric_name):
    print('\n')
    print('Clusters using metric {}'.format(metric_name))
    print('===================================')

    for stemmed_word, connected_nodes in stemmed_word_clusters.items():
        for connected_node in connected_nodes:
            print(u'{}\t: {}'.format(stemmed_word, ', '.join(connected_node)))


# List all files ====================================================================================================

html_files = []
for root, dirs, files in os.walk(os.path.abspath('/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/assignments/a2/code_report/articles/z')):
    for file in files:
        if file.endswith('.html'):
            filepath = os.path.join(root, file)
            html_files.append(filepath)
```

```python


file_words_index = {}
all_words = set()
window_text_number = 100

# Index all words and files ==========================================================================
for idx, file in enumerate(html_files):
    print('{} of {}. Processing file {}'.format(idx+1, len(html_files), file))
    print('=' * 30)

    # get text only from each file -> remove all tags
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="utf-8").readlines()]))

    # get all words from text by splitting by whitespace
    words = [word.lower() for word in text.split() if word.isalpha()]

    # make text window containing 50 - 100 words
    for w in range(1, int(len(words)/window_text_number)):
        start = (w-1)*window_text_number
        end = w*window_text_number
        if end > len(words): end = len(words)

        window_words = words[start:end]

        all_words |= set(window_words)
        file_words_index[os.path.basename(file) + '_part_' + str(w)] = window_words


# # Remove stopword to purify result ================================================================
# nltk.download('stopwords')
# all_words = [word for word in all_words if word not in stopwords.words('english')]
#
#
# # Select random 10 words ===========================================================================

# all_words = random.sample(all_words, 10)


# Invert words and files index =====================================================================
word_files_index = {}
for idx, word in enumerate(all_words):
    print('{} of {}. Processing word {}'.format(idx + 1, len(all_words), word.encode('utf-8')))
    print('=' * 30)

    files = []
    for file, words in file_words_index.items():
        if word in words:
            files.append(file)
    word_files_index[word] = sorted(set(files))

```

```

# Stem words ==============================================================================================
krovetz = KrovetzStemmer()
stem_word_index = {}
for word, files in word_files_index.items():
    # Stem word using krovetz
    stemmed_word = krovetz.stem(word)

    # Group by stemmed word
    stem_word_index.setdefault(stemmed_word, [])
    stem_word_index[stemmed_word].append(word)


# Calculate coefficient ===================================================================================
coef_threshold = 0.0

dice_stemmed_word_data = []
mim_stemmed_word_data = []
emim_stemmed_word_data = []
chi_sqr_stemmed_word_data = []

counter = 0
for stemmed_word, words in stem_word_index.items():
    # create bigrams from words
    bigrams = list(nltk.bigrams(words))
    for word_a, word_b in bigrams:
        # Lookup filename in word_files_index
        files_a = word_files_index[word_a]
        files_b = word_files_index[word_b]
        files_a_sliced_b = list(set(files_b) & set(files_a))

        # Using dice coef
        dice_coef = float(len(files_a_sliced_b)) / (len(files_a) + len(files_b))
        if (dice_coef > coef_threshold):
            dice_stemmed_word_data.append((stemmed_word, word_a, word_b, dice_coef))

        # Using MIM coef
        mim_coef = float(len(files_a_sliced_b)) / (len(files_a) * len(files_b))
        if (mim_coef > coef_threshold):
            mim_stemmed_word_data.append((stemmed_word, word_a, word_b, mim_coef))

        # Using EMIM coef
        if len(files_a_sliced_b) > 0:
            emim_coef = len(files_a_sliced_b) * \
                        math.log(len(file_words_index) * len(files_a_sliced_b) / float(len(files_a) * len(files_b)))
        else:
            emim_coef = 0.0

        if (emim_coef > coef_threshold):
            emim_stemmed_word_data.append((stemmed_word, word_a, word_b, emim_coef))

        # Chi-square
        chi_sqr_coef = math.pow((len(files_a_sliced_b) - (float(len(files_a) * len(files_b)) / len(file_words_index))), 2) / \
                        float(len(files_a) * len(files_b))
        if (chi_sqr_coef > coef_threshold):
            chi_sqr_stemmed_word_data.append((stemmed_word, word_a, word_b,
    chi_sqr_coef))

    if len(bigrams) > 0: counter+=1
    if counter >= 10: break



# Create graph
dice_stemmed_word_clusters = create_graph(dice_stemmed_word_data, 'dice')
mim_stemmed_word_clusters = create_graph(mim_stemmed_word_data, 'mim')
emim_stemmed_word_clusters = create_graph(emim_stemmed_word_data, 'emim')
chi_sqr_stemmed_word_clusters = create_graph(chi_sqr_stemmed_word_data, 'chi_sqr')

# Print clusters
print_stem_cluster(dice_stemmed_word_clusters, 'dice')
print_stem_cluster(mim_stemmed_word_clusters, 'mim')
print_stem_cluster(emim_stemmed_word_clusters, 'emim')
print_stem_cluster(chi_sqr_stemmed_word_clusters, 'chi_sqr')
```

Listing 5: Python code for MLN1

I create the cluster using the graphs and extract the connected components from the graph. Figure 5 shows the example of the graph for the word 'feature'.

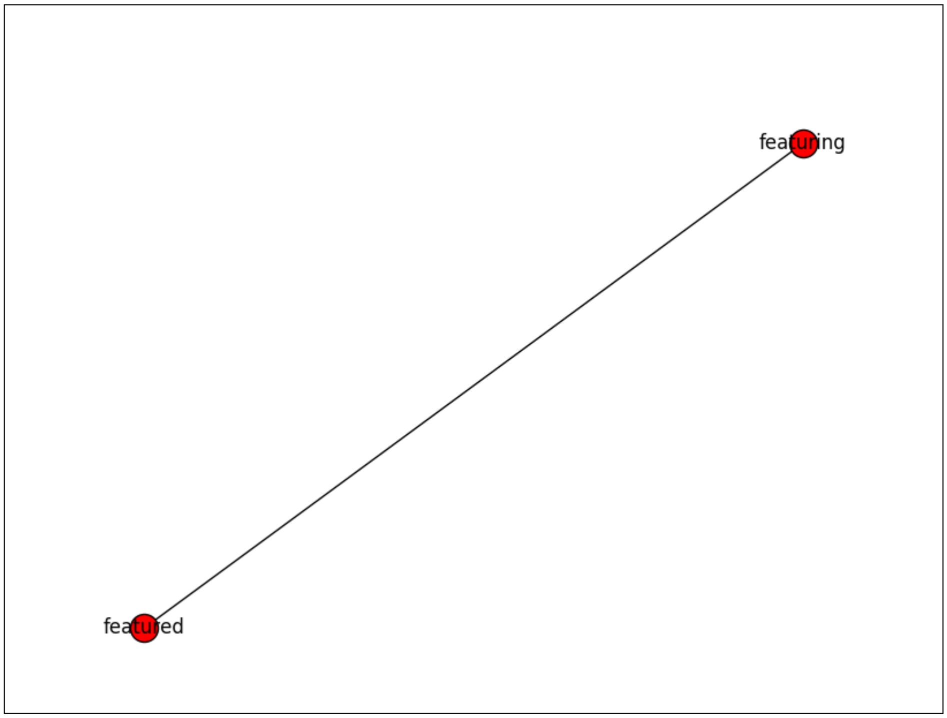
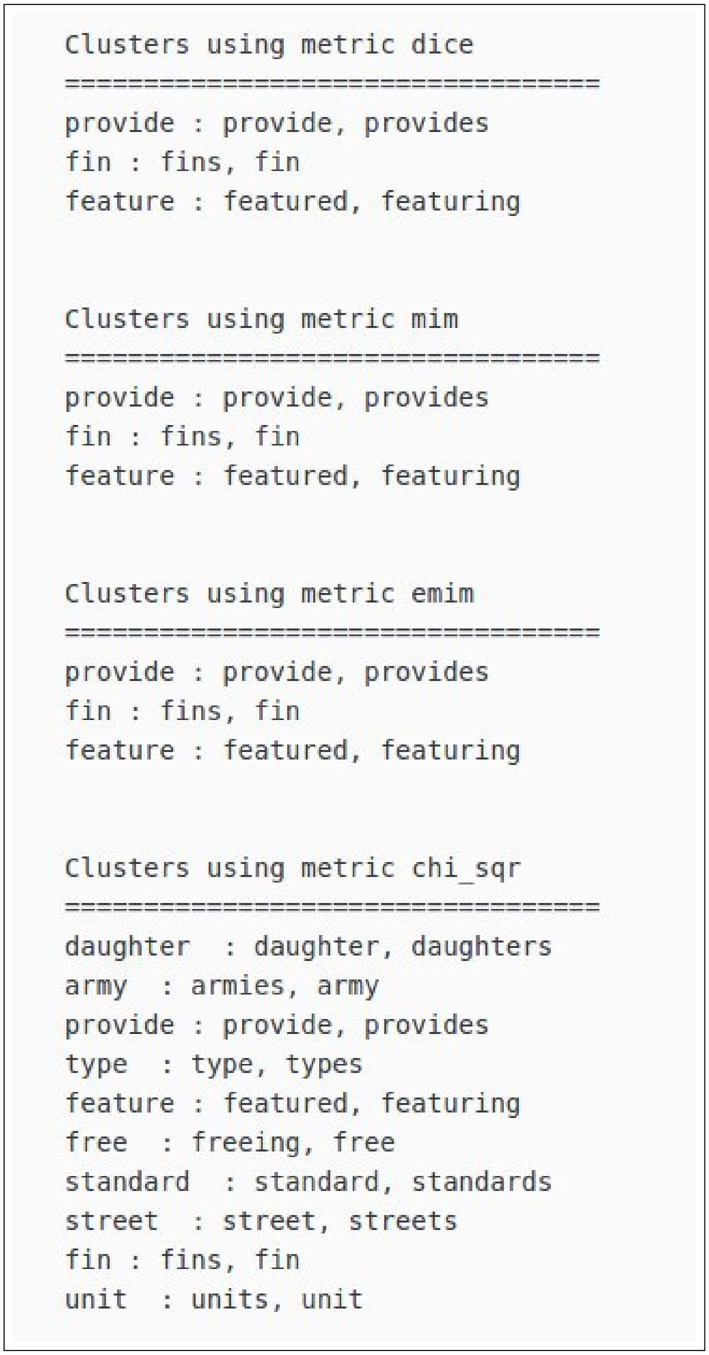


Figure 5: Connected component for the word 'feature'

From the output that I got, I see that difference association measurement return different clustering result. I also noticed that several terms do not have any other components connected to it. Figure 6 shows the cluster resulted from using Dice's Coefficient, MIM, EMIM, and Chi-square as the association measures.



```
Clusters using metric dice
==================================
provide : provide, provides
fin : fins, fin
feature : featured, featuring


Clusters using metric mim
==================================
provide : provide, provides
fin : fins, fin
feature : featured, featuring


Clusters using metric emim
==================================
provide : provide, provides
fin : fins, fin
feature : featured, featuring


Clusters using metric chi_sqr
==================================
daughter  : daughter, daughters
army  : armies, army
provide : provide, provides
type  : type, types
feature : featured, featuring
free  : freeing, free
standard  : standard, standards
street  : street, streets
fin : fins, fin
unit  : units, unit
```

Figure 6: The clustering result using Dice's Coefficient, MIM, EMIM, and Chi-square

---

Question MLN2

`MLN2: 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)`

Answer

Listing 6 shows the code to find the most strongly associated words for all 10 words that I have chosen.

```

#!/usr/bin/python

import json
import os
import random
import html2text
import io
import math
import nltk

from krovetzstemmer import Stemmer as KrovetzStemmer
from nltk.corpus import stopwords


class AssociationMeasure:
    _document_words_index = {}
    _window_words_index_cache = {}
    _word_windows_index_cache = {}

    def __init__(self, dir_name):
        '''
        List all files in dir_name, and create document-words index
        :param dir_name:
        '''

        self._document_words_index = json.loads(io.open(dir_name).read())

    def get_window_words_index(self, window_size=None):
        '''
        Get window-words index
        :param window_size: number of words in a window, None or False if use document
        :return:
        '''

        # Re-build word_windows_index if window_size is not exists in _word_windows_index_cache
        if window_size not in self._window_words_index_cache:
            if not window_size:
                window_words_index = self._document_words_index
            else:
                window_words_index = {}
                for document, words in self._document_words_index.items():
                    for w in range(1, int(len(words) / window_size)):
                        start = (w - 1) * window_size
                        end = w * window_size
                        if end > len(words): end = len(words)

                        window_words = words[start:end]
                        window_words_index[document + '_part_' + str(w)] = window_words

            # Cache word_windows_index for each window_size
            self._window_words_index_cache[window_size] = window_words_index
```

```python


        return self._window_words_index_cache[window_size]

    def get_all_words(self, window_words_index):
        '''
        Get all words in a window-words index
        :param window_words_index:
        :return:
        '''

        all_words = []
        for window, words in window_words_index.items():
            all_words += words

        all_words = set(all_words)
        return all_words

    def get_stemmed_words_index(self, window_words_index):
        '''
        Get stemmed-words index from window-words index
        :param window_words_index:
        :return:
        '''

        all_words = self.get_all_words(window_words_index)
        stem_words_index = {}

        krovetz = KrovetzStemmer()
        for word in all_words:
            # Stem word using krovetz
            stemmed_word = krovetz.stem(word)

            # Group by stemmed word
            stem_words_index.setdefault(stemmed_word, [])
            stem_words_index[stemmed_word].append(word)

        return stem_words_index

    def get_word_windows_index(self, window_size=None):
        '''
        Get word-windows index
        :param window_size: number of words in a window, None or False if use document
        :return:
        '''

        return json.loads(io.open('../6_1/1_2_index.txt').read())

        # # Re-build word_windows_index if window_size is not exists in _word_windows_index_cache
        # if window_size not in self._word_windows_index_cache:
        #     window_words_index = self.get_window_words_index(window_size)
        #     all_words = self.get_all_words(window_words_index)
        #
        #     word_windows_index = {}
        #     for idx, word in enumerate(all_words):
        #         print('{} of {}. Indexing word {}'.format(idx + 1, len(all_words), word.encode('utf-8')))
        #          print('=' * 30)
        #
        #          files = []
        #          for file, words in window_words_index.items():
        #              if word in words:
        #                  files.append(file)
        #          word_windows_index[word] = sorted(set(files))
        #
        #
        #     # Cache word_windows_index for each window_size
        #     self._word_windows_index_cache[window_size] = word_windows_index
        #
        #
        # return self._word_windows_index_cache[window_size]

    def dice(self, word_a, word_b, window_size=None):
        '''
        Use dice coefficient
        :param word_1:
        :param word_2:
        :param window_size:
        :return:
        '''

        # Create window-word index
        word_windows_index = self.get_word_windows_index(window_size)

        # Lookup filename in word_files_index
        files_a = word_windows_index[word_a]
        files_b = word_windows_index[word_b]
        files_a_sliced_b = list(set(files_b) & set(files_a))

        dice_coef = float(len(files_a_sliced_b)) / (len(files_a) + len(files_b))
        return dice_coef

    def mim(self, word_a, word_b, window_size=None):
        '''
        User Mutual Information Measure
        :param word_a:
        :param word_b:
        :param window_size:
        :return:
        '''

        # Create window-word index
        word_windows_index = self.get_word_windows_index(window_size)

        # Lookup filename in word_files_index
        files_a = word_windows_index[word_a]
        files_b = word_windows_index[word_b]
        files_a_sliced_b = list(set(files_b) & set(files_a))

        mim_coef = float(len(files_a_sliced_b)) / (len(files_a) * len(files_b))
        return mim_coef

    def emim(self, word_a, word_b, window_size=None):
        '''
        Use Expected Mutual Information Measure
        :param word_a:
```

```
        :param word_b:
        :param window_size:
        :return:
        '''

        # Get window-words and word-windows index
        window_words_index = self.get_window_words_index(window_size)
        word_windows_index = self.get_word_windows_index(window_size)

        # Lookup filename in word_files_index
        files_a = word_windows_index[word_a]
        files_b = word_windows_index[word_b]
        files_a_sliced_b = list(set(files_b) & set(files_a))

        if len(files_a_sliced_b) > 0:
            emim_coef = len(files_a_sliced_b) * \
                        math.log(len(window_words_index) * len(files_a_sliced_b) /
    float(len(files_a) * len(files_b)))
        else:
            emim_coef = 0.0

        return emim_coef

    def chi_square(self, word_a, word_b, window_size=None):
        '''
        Chi-Square Measure
        :param word_a:
        :param word_b:
        :param window_size:
        :return:
        '''

        # Get window-words and word-windows index
        window_words_index = self.get_window_words_index(window_size)
        word_windows_index = self.get_word_windows_index(window_size)

        # Lookup filename in word_files_index
        files_a = word_windows_index[word_a]
        files_b = word_windows_index[word_b]
        files_a_sliced_b = list(set(files_b) & set(files_a))

        chi_sqr_coef = math.pow((len(files_a_sliced_b) -
                                 (float(len(files_a) * len(files_b)) / len(
    window_words_index))), 2) / \
                       float(len(files_a) * len(files_b))
        return chi_sqr_coef


def get_most_associates(most_associated):
    most_associated_sorted = {}
    for word, associates in most_associated.items():
        associates = sorted(associates, key=lambda x: x[1], reverse=True)
        if len(associates) > 10: associates = associates[:10]

        most_associated_sorted[word] = associates


    return most_associated_sorted
```

```
if __name__ == '__main__':
    selected_words = ['abolishes', 'access', 'accommodate', 'accredited', 'sky',
                      'railroad', 'calendar', 'airplane', 'airplane', 'bicycle']

    measure = AssociationMeasure('../6_1/1_2_file_words_index.txt')

    # Get all words and create all possible bigrams
    all_word = measure.get_word_windows_index().keys()
    all_bigrams = bigrams = list(nltk.bigrams(all_word))

    # Calculate coocurence
    most_associated_all_d = {}
    most_associated_all_m = {}
    most_associated_all_e = {}
    most_associated_all_c = {}

    # most_associated_5_d = {}
    # most_associated_5_m = {}
    # most_associated_5_e = {}
    # most_associated_5_c = {}

    print all_word
    for word in selected_words:
        for word_a, word_b in all_bigrams:
            if word == word_a or word == word_b:
                dice_window_size_all = measure.dice(word_a, word_b)
                mim_window_size_all = measure.mim(word_a, word_b)
                emim_window_size_all = measure.emim(word_a, word_b)
                chi_square_window_size_all = measure.chi_square(word_a, word_b)

                # Use window-size = 5
                dice_window_size_5 = measure.dice(word_a, word_b, 5)
                mim_window_size_5 = measure.mim(word_a, word_b, 5)
                emim_window_size_5 = measure.emim(word_a, word_b, 5)
                chi_square_window_size_5 = measure.chi_square(word_a, word_b, 5)

                most_associated_all_d.setdefault(word, [])
                most_associated_all_m.setdefault(word, [])
                most_associated_all_e.setdefault(word, [])
                most_associated_all_c.setdefault(word, [])

                # most_associated_5_d.setdefault(word, [])
                # most_associated_5_m.setdefault(word, [])
                # most_associated_5_e.setdefault(word, [])
                # most_associated_5_c.setdefault(word, [])

                most_associated_all_d[word].append((word_a if word_b == word else
    word_b, dice_window_size_all))
                most_associated_all_m[word].append((word_a if word_b == word else
    word_b, mim_window_size_all))
                most_associated_all_e[word].append((word_a if word_b == word else
    word_b, emim_window_size_all))
                most_associated_all_c[word].append((word_a if word_b == word else
    word_b, chi_square_window_size_all))

                # most_associated_5_d[word].append((word_a if word_b == word else
    word_b, dice_window_size_5))
```

```
            # most_associated_5_m[word].append((word_a if word_b == word else word_b, mim_window_size_5))
            # most_associated_5_e[word].append((word_a if word_b == word else word_b, emim_window_size_5))
            # most_associated_5_c[word].append((word_a if word_b == word else word_b, chi_square_window_size_5))


    # Sort ascending and get top 10
    most_associated_all_d = get_most_associates(most_associated_all_d)
    most_associated_all_m = get_most_associates(most_associated_all_m)
    most_associated_all_e = get_most_associates(most_associated_all_e)
    most_associated_all_c = get_most_associates(most_associated_all_c)

    # most_associated_5_d = get_most_associates(most_associated_5_d)
    # most_associated_5_m = get_most_associates(most_associated_all_d)
    # most_associated_5_e = get_most_associates(most_associated_5_e)
    # most_associated_5_e = get_most_associates(most_associated_5_e)

    print most_associated_all_d

# # a. Find 10 words randomly ==============================================================
# stemmed_words_index = measure.get_stemmed_words_index(measure.get_window_words_index())
#
# # List only non-stop-words
# nltk.download('stopwords')
# stemmed_words= [stemmed_word for stemmed_word, words in stemmed_words_index.items()
#                  if stemmed_word not in stopwords.words('english')]
#
# # Find 10 stemmed-words randomly
# stemmed_words = random.sample(stemmed_words, 10)
#
#
# # b. Calculate co-occurence measure with window_size = 5 and window_size = full-document ====
# for word in stemmed_words:
#     # get words in stemmed-word-class
#     words = stemmed_words_index[word]
#     # create bigrams for words
#     bigrams = list(nltk.bigrams(words))
#
#     for word_a, word_b in bigrams:
#         # Use window-size = all document
#         dice_window_size_all = measure.dice(word_a, word_b)
#         mim_window_size_all = measure.mim(word_a, word_b)
#         emim_window_size_all = measure.emim(word_a, word_b)
#         chi_square_window_size_all = measure.chi_square(word_a, word_b)
#
#         # Use window-size = 5
#         dice_window_size_5 = measure.dice(word_a, word_b, 5)
#         mim_window_size_5 = measure.mim(word_a, word_b, 5)
#         emim_window_size_5 = measure.emim(word_a, word_b, 5)
#         chi_square_window_size_5 = measure.chi_square(word_a, word_b, 5)
#
#         print(u'Word class {} : {} - {} :'.format(word, word_a, word_b))
#
```

```
#         print('\tUsing dice coefficient')
#         print('\t\tWindow size = all --> {}'.format(dice_window_size_all))
#         print('\t\tWindow size = {} --> {}'.format(5, dice_window_size_5))
#
#         print('\tUsing MIM coefficient')
#         print('\t\tWindow size = all --> {}'.format(mim_window_size_all))
#         print('\t\tWindow size = {} --> {}'.format(5, mim_window_size_5))
#
#         print('\tUsing EMIM coefficient')
#         print('\t\tWindow size = all --> {}'.format(emim_window_size_all))
#         print('\t\tWindow size = {} --> {}'.format(5, emim_window_size_5))
#
#         print('\tUsing Chi-square coefficient')
#         print('\t\tWindow size = all --> {}'.format(chi_square_window_size_all))
#         print('\t\tWindow size = {} --> {}'.format(5, chi_square_window_size_5))
```

Listing 6: Association Measure

Tables below show the strong associated words for the chosen words.

Table 3: Strong associated words for 'school'

MIM	EMIM	Chi Square	Dice
renaissance	poets	new	renaissance
sons	renaissance	poets	sons
formalism	new	group	poets
others	group	renaisaance	formalism
poets	son	southern	others
group	others	others	group
agrarians	formalism	sons	agrarians
artists	southern	formalism	artists
club	agrarians	club	club
southern	artists	agrarians	southern

Table 4: Strong associated words for 'city'

MIM	EMIM	Chi Square	Dice
tackle	back	back	tackle
area	tackle	new	back
founded	new	green	founded
county	green	bay	green
shopping	bay	end	bay
metropolitan	rams	packers	rams
privileges	packers	edit	packers
back	end	events	county
green	area	founded	new

Dice	MIM	EMIM	Chi-square
division	oxford	united	import
army	army	division	edit
oxford	divisions	army	united
united	division	oxford	id
th	birth	th	th
university	transportation	university	class
sovereign	allegiance	sovereign	division
press	sovereign	press	free
divisions	press	divisions	army
disambiguation	united	disambiguation	events

Table 5: Strong associated words for ‘states’

Dice	MIM	EMIM	Chi-square
zero	zero	zero	zero
use	element	poetry	poetry
programming	programming	poets	poets
poets	systems	element	value
systems	array	may	also
would	however	would	index
however	would	also	use
array	wishing	programming	would
array	poets	however	would
element	use	use	may

Table 6: Strong associated words for ‘language’

Dice	MIM	EMIM	Chi-square
zero	zero	zero	id
count	empty	count	zero
parser	set	parser	count
categories	defined	categories	main
expensive	shopping	expensive	value
bytes	comet	bytes	parser
poetry	particular	empty	bytese
size	zeros	set	categories
set	cardinality	poetry	expensive
asteroid	constant	defined	see

Table 7: Strong associated words for ‘function’

Dice	MIM	EMIM	Chi-square
tackle	tackle	back	back
back	regiment	tackle	lions
regiment	soviet	lions	redskins
lions	artillery	redskinsd	giants
redskins	back	mm	guard
mm	lions	regiment	id
soviet	volunteer	giants	mm
infantry	redskins	guard	new
artillery	allegiance	infantry	tackle
giants	mount	soviet	infantry

Table 8: Strong associated words for ‘union’

Dice	MIM	EMIM	Chi-square
limit	howitzer	limit	edit
preprocessor	song	preprocessor	count
node	area	node	limit
count	fellowships	count	preprocessor
kingdom	preprocessor	howitzer	node
championships	limit	kingdom	see
post	node	song	article
howitzer	gunfighter	championships	th
expanding	workshop	area	also
seasons	fas	expanding	league

Table 9: Strong associated words for ‘report’

Dice	MIM	EMIM	Chi-square
zero	zero	zero	zero
buy	buy	buy	buy
limited	limited	limited	limited
exponentiation	exponentiation	exponentiation	exponentiation
four	four	four	four
sleep	sleep	sleep	sleep
oldest	oldest	oldest	oldest
whose	whose	whose	whose
calculate	calculate	calculate	calculate
segments	segments	segments	segments

Table 10: Strong associated words for ‘fish’

Dice	MIM	EMIM	Chi-square
singles	singles	singles	edit
championships	elections	championships	championships
elections	procterate	elections	round
protectorate	permitted	act	main
act	parliamentary	protectorate	article
candidadtes	stand	candidates	singles
allowing	championships	allowing	act
permitted	candidates	permitted	defeated
defeated	allowing	defeated	elections
parliamentary	act	parliamentary	candidates

Table 11: Strong associated words for 'women'

References

[1] Ruey-Cheng Chen. KrovetzStemmer 0.4. `https://pypi.python.org/pypi/KrovetzStemmer/0.4`, 2016. [Online; accessed 12-October-2016].

[2] Wikipedia. Algorithm Implementation/Strings/Dice's coefficient. `https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Dice%27s_coefficient/`, 2016. [Online; accessed 8-November-2016].

[3] Dan Jurafsky. Spelling Correction and the Noisy Channel . `https://web.stanford.edu/class/cs124/lec/spelling.pdf`, 2012. [Online; accessed 8-November-2016].

[4] Peter Norvig. How to Write a Spelling Corrector . `http://norvig.com/spell-correct.html`, 2016. [Online; accessed 9-November-2016].

[5] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[6] The Lemur Project. The Lemur Project - Galago 3.10 . `https://sourceforge.net/p/lemur/galago/ci/release-3.10/tree/`, 2016. [Online; accessed 5-November-2016].

[7] The Lemur Project. The Lemur Project . `http://www.lemurproject.org/`, 2016. [Online; accessed 5-November-2016].

[8] Andrew Turpin, Yohannes Tsegay, David Hawking, and Hugh E. Williams. Fast generation of result snippets in web search. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 127–134, New York, NY, USA, 2007. ACM.