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 = []

```

```
# 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

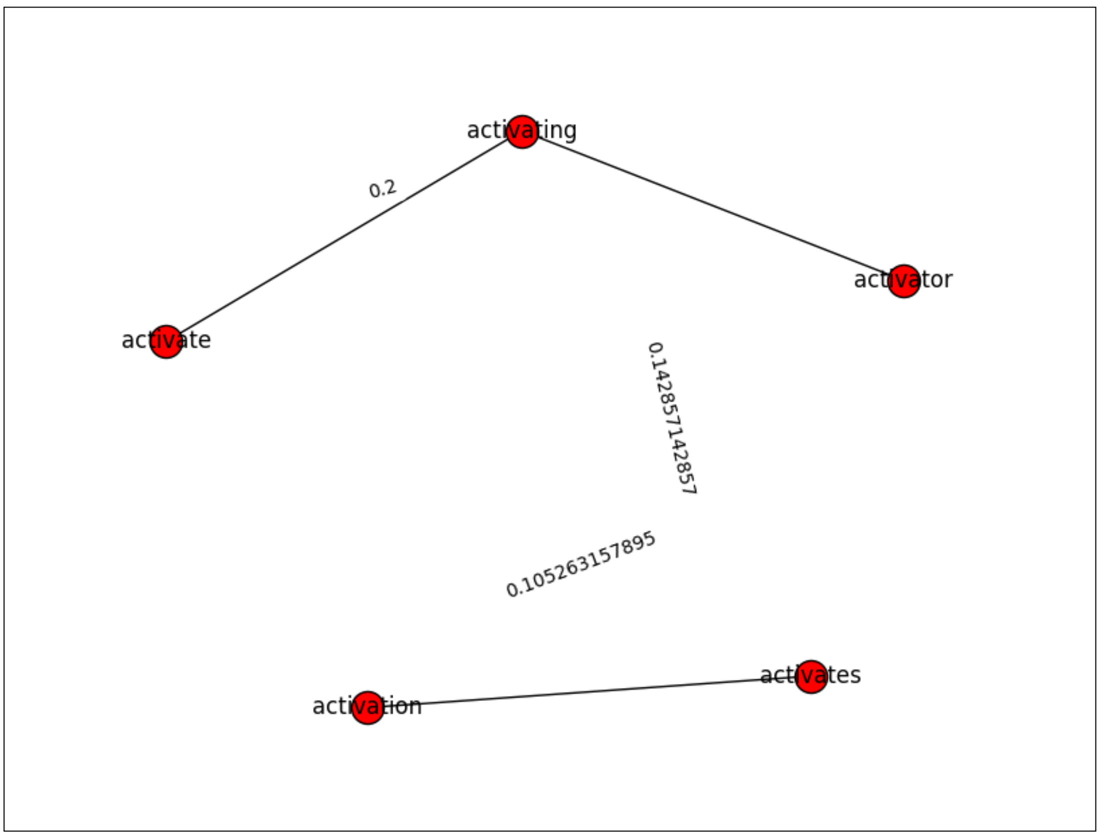

Figure 1: Graph of the connected component for the stem class 'activate'

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)

```

```python

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.

```java

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;
```

```java
    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.

```java
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’.

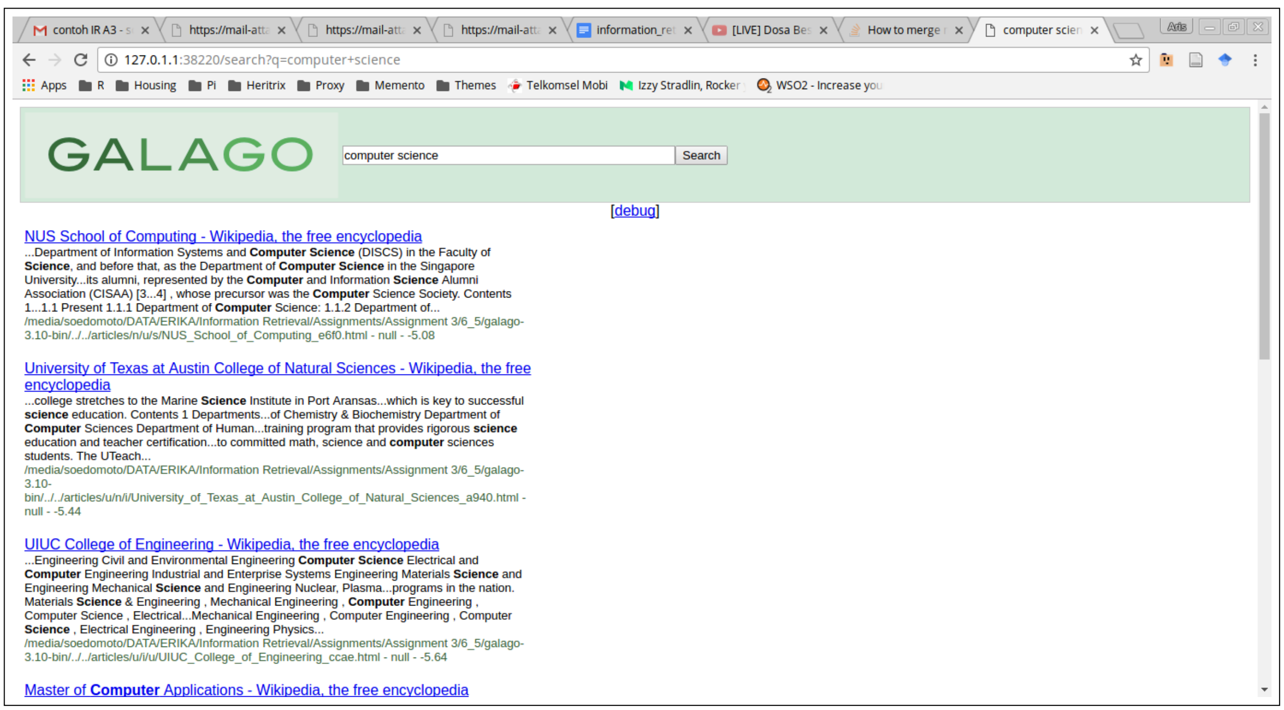

Figure 3: Snippet generated by Galago for query terms ‘computer science’

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 ℓ 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 ℓ 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

D

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

e

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.