# Semantic Similarity

Noah Buchanan
Computer Science Department
University of Arkansas - Fort Smith

July 2, 2021

## Introduction

The following are programs designed to extract contextual and semantic properties of words using a pre-tokenized corpus of files and using the aggregation of the words in said corpus. A pruned lexicon of a specific amount is used and the use of Cosine Similarity and Positive Point-wise Mutual Information(PPMI) is used to complete these tasks.

## Background

Mutual information is weighting of the mutual dependence between two variables, in short measuring how similar the meaning or contextual use of the words are. Things like hot and dog should theoretically have a high mutual information because they are used in conjunction frequently as "hot dog" so they would be used in similar context often. Point-wise mutual information is a measure of how often two words occur compared to what we could expect if they were independent. Lastly PPMI is just an adaptation of the prior where if the value is below zero we simply truncate at 0.

n-grams are a way of creating word combinations with a certain window size, say we have a window size of 2 we would refer to it as a bi-gram and the bi-grams produced from the following "this is fun right" would be "this is", "is fun", and "fun right". For this implementation we use bi-grams for simplicity and the count aggregation of a specific bi-gram combination would be used to calculate the PPMT using the following formula:

$$PPMI(w1, w2) = max\left[log_2\left(\frac{Pr(w1, w2)}{Pr(w1) * Pr_a(w2)}\right), 0\right]$$

w1 and w2 denoting words of interest $Pr()$ denoting the probability of given arguments and $Pr_a()$ is a weighted form of probability that increases the probability assigned to rare context words, this lowers the PMI.

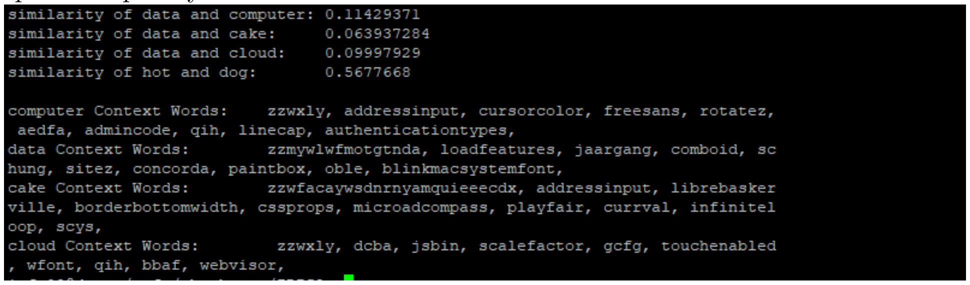
## Specification

The process begins by first obtaining some corpus that has been tokenized so that we can use the words from said corpus. We must remove stop words from the file that we will be using. Next we aggregate the count of each individual word in the file, we truncate everything to lowercase so there are no capitalized duplicates. Once we have the individual counts of words we must now aggregate the count of the unique bi-grams. Once all the prior tasks have been complete we can move to the algorithm that actually constructs the context matrix using PPMI and the counts already aggregated. The algorithm is extremely simple, we must first obtain a total for the regular probabilities(sum of all counts of bi-grams) and a weighted total for the aformentioned $Pr_a()$ which is just the summation of the sum of each column raised to the power of 0.75. Once you have those you simply perform the PPMI calculation above for each word combination of word and context word and fill a matrix of Lexicon length by Lexicon length with these values and store it to disk.

In UAWordSimilarity we use the matrix and the Lexicon created from the prior programs execution to use the data. We read in the lexicon and cmatrix from disk, calculate the cosine similarity of the words passed in and get the top k context words ranked by cosine similarity of each word passed in relative to all other words in the Lexicon.

## Implementation

The UAContextMatrix program runs in $O(n^2)$ time complexity and with $O(2n)$ space complexity. UAWordSimilarity runs in $O(nlog(n))$ with and with $O(n)$ space complexity.



```
similarity of data and computer: 0.11429371
similarity of data and cake:     0.063937284
similarity of data and cloud:    0.09997929
similarity of hot and dog:       0.5677668

computer Context Words:    zzwxly, addressinput, cursorcolor, freesans, rotatez,
 aedfa, admincode, qih, linecap, authenticationtypes,
data Context Words:        zzmywlwfmotgtnda, loadfeatures, jaargang, comboid, sc
hung, sitez, concorda, paintbox, oble, blinkmacsystemfont,
cake Context Words:        zzwfacaywsdnrnyamquieeecdx, addressinput, librebasker
ville, borderbottomwidth, cssprops, microadcompass, playfair, currval, infinitel
oop, scys,
cloud Context Words:        zzwxly, dcba, jsbin, scalefactor, gcfg, touchenabled
, wfont, qih, bbaf, webvisor,
```

These specific context words do not seem exactly correct there would need to be better tokenization because a lot of these look like code variables or etc but there are relevant context word.

UAContextMatrix ran in 17,755.501 seconds.

## Evaluation

The dataset used was a tokenized version of 839,000 html files that were scraped from the web. Pre-processing included removing stop words from lexicon and word file. The size of the Lexicon was 69,876 words. Word window size used was 2, and bigrams were used in the construction of n-grams. The overall performance I think worked very well, the only problems were with the tokenization and this was not apart of the task assigned for this. The Cosine Similarities work and semantically make sense in my testing.

## Conclusion

This concludes my explanation of the process I took and how UAContextMatrix and UAWordSimilarity extract semantic similarity from words using PPMI and Cosine Similarity.

## UAContextMatrix

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;

public class UAContextMatrix {

  public static HashMap<String, Integer> Freq;

  public static void main(String[] args) throws IOException {

    Freq = new HashMap<>();// 69500 roughly for actual thing
    // filling words to keep and words to drop
    HashMap<String, Integer> StopWords = new HashMap<>();
    HashMap<String, Integer> KeepWords = new HashMap<>();
    KeepWords.put("data", 0);
    KeepWords.put("computer", 0);
    KeepWords.put("cloud", 0);
    KeepWords.put("cake", 0);
    KeepWords.put("hot", 0);
    KeepWords.put("dog", 0);
    int count = 0;
    BufferedReader br = new BufferedReader(new FileReader("stopwords.txt"));
```

```
String line = "";
while ((line = br.readLine()) != null) {
  StopWords.put(line, 0);
}
br.close();
// filling freq with only the words we want based on keep and drop words
// provided
br = new BufferedReader(new FileReader("../IRPS1/frequency.txt"));
while ((line = br.readLine()) != null) {
  String[] split = line.split(" ");
  if (KeepWords.get(split[split.length - 1].toLowerCase()) != null
      || (count <= 70000 && StopWords.get(split[split.length - 1].toLowerCas
          && Integer.parseInt(split[split.length - 2]) > 3)) {
    Freq.put(split[split.length - 1].toLowerCase(), Integer.parseInt(split[s
  }
  count++;
}
br.close();
// getting alphabetically sorted Array of Lexicon
count = 0;
String[] Lexicon = new String[Freq.keySet().size()];
for (String key : Freq.keySet()) {
  Lexicon[count] = key;
  count++;
}
Arrays.sort(Lexicon);
// filtering word.txt data into cleaned.txt and filling hashmap with bigrams
// respective frequencies
HashMap<String, Integer> Bigrams = new HashMap<>();
br = new BufferedReader(new FileReader(args[0]));
BufferedWriter bw = new BufferedWriter(new FileWriter("cleaned.txt"));
while ((line = br.readLine()) != null) {
  line = line.toLowerCase();
  if (Freq.get(line) != null) {
    StringBuilder build = new StringBuilder();
    build.append(line);
    build.append("\n");
    bw.write(build.toString());
  }
}
br.close();
bw.close();
br = new BufferedReader(new FileReader("cleaned.txt"));
String prev = br.readLine();
while ((line = br.readLine()) != null) {
  StringBuilder build = new StringBuilder();
```

```
      build.append(prev);
      build.append(" ");
      build.append(line);
      if (Bigrams.get(build.toString()) == null) {
        Bigrams.put(build.toString(), 1);
      } else {
        Bigrams.put(build.toString(), Bigrams.get(build.toString()) + 1);
      }
      prev = line;
    }
    br.close();

    // writing cmatrix to file and lexicon to file
    float[][] Cmatrix = BuildTermContextMatrix(Lexicon, Bigrams);
    bw = new BufferedWriter(new FileWriter(args[1] + "/cmatrix.txt"));
    for (int i = 0; i < Cmatrix.length; i++) {
      for (int j = 0; j < Cmatrix.length; j++) {
        StringBuilder string = new StringBuilder();
        string.append(Cmatrix[i][j]);
        string.append("\t");
        bw.write(string.toString());
      }
      bw.newLine();
    }
    bw.close();
    bw = new BufferedWriter(new FileWriter(args[1] + "/lexicon.txt"));
    for (int i = 0; i < Lexicon.length; i++) {
      bw.write(Lexicon[i]);
      bw.newLine();
    }
    bw.close();
  }

  /**
   * Builds a Term Context Matrix of PPMI values using Pr(w1) Pr_a(w2) and
   * Pr(w1,w2) for each word combination
   *
   * @param Lexicon List of words being used post removal of stop words
   * @param Bigrams list of unique bigrams and their respecting frequencies
   * @return returns a context matrix of PPMI values
   */
  public static float[][] BuildTermContextMatrix(String[] Lexicon, HashMap<Strin

    float total = 0;
    float weightedtotal = 0;
    float[][] Cmatrix = new float[Lexicon.length][Lexicon.length];
  // filling Cmatrix with frequency of each bigram and calculating the unweigh
  // total for probabilities later
  for (int i = 0; i < Lexicon.length; i++) {
    float subtotal = 0;
    for (int j = 0; j < Lexicon.length; j++) {
      StringBuilder string = new StringBuilder();
      string.append(Lexicon[i]);
      string.append(" ");
      string.append(Lexicon[j]);
      if (Bigrams.get(string.toString()) != null) {
        Cmatrix[i][j] = Bigrams.get(string.toString());
      } else {
        Cmatrix[i][j] = 0;
      }
      subtotal += Cmatrix[i][j];
    }
    total += subtotal;
  }

  // calculating the weighted probabilities of the columns
  for (int i = 0; i < Lexicon.length; i++) {
    float subtotal = 0;
    for (int j = 0; j < Lexicon.length; j++) {
      subtotal += Cmatrix[j][i];
    }
    weightedtotal += Math.pow(subtotal, .75);
  }

  // finishing the math using our precalculated unweighted and weightedtotal f
  // pr_a(w2) and pr(w1) and pr(w1,w2)
  for (int i = 0; i < Lexicon.length; i++) {
    for (int j = 0; j < Lexicon.length; j++) {
      // Pr(w1,w2) //Pr(w1) //Pr_a(w2)
      Cmatrix[i][j] = (Cmatrix[i][j] / total) / ((float) (Freq.get(Lexicon[i])
          * (float) ((float) Math.pow(Freq.get(Lexicon[j]), 0.75) / weightedto
      // log_2
      Cmatrix[i][j] = (float) (Math.log10(Cmatrix[i][j]) + 0.00000000000000001
      // max function
      if (Cmatrix[i][j] < 0) {
        Cmatrix[i][j] = 0;
      }
    }
  }

  return Cmatrix;
}
}
```

## UAWordSimilarity

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

public class UAWordSimilarity {

  public static ArrayList<String> Lexicon = new ArrayList<>();

  public static void main(String[] args) throws IOException {

    BufferedReader br = new BufferedReader(new FileReader(args[0]));
    String line = "";
    while ((line = br.readLine()) != null) {
      Lexicon.add(line);
    }
    br.close();
    br = new BufferedReader(new FileReader(args[1]));
    float[][] Cmatrix = new float[Lexicon.size()][Lexicon.size()];
    int count = 0;
    while ((line = br.readLine()) != null) {
      String[] split = line.split("\t");
      for (int i = 0; i < split.length; i++) {
        Cmatrix[count][i] = Float.parseFloat(split[i]);
      }
      count++;
    }
//.................................................................
    String[] context;
    /*
     * System.out.println("Word 1:    " + args[2] + "\t\t Word 1 Lexicon Index:\t
     * GetWordIndex(Lexicon, args[2])); System.out.println("Word 2:
" + args[3] +
     * "\t\t Word 2 Lexicon Index:\t" + GetWordIndex(Lexicon, args[3]));
     * System.out.println(); System.out.println("Cosine Similarity Score: " +
     * CalculateSimilarity(Cmatrix, GetWordIndex(Lexicon,
     * args[2]),GetWordIndex(Lexicon, args[3]))); System.out.println();
```

```
 * System.out.print("Word 1 Context Words:    "); context =
 * GetContext(Cmatrix,GetWordIndex(Lexicon, args[2]),Integer.parseInt(args[4
 * for(int i = 0; i < context.length; i++) { if(context[i] != null) {
 * System.out.print(context[i] + ", "); } } System.out.println();
 * System.out.print("Word 2 Context Words:    "); context =
 * GetContext(Cmatrix,GetWordIndex(Lexicon, args[3]),Integer.parseInt(args[4
 * for(int i = 0; i < context.length; i++) { if(context[i] != null) {
 * System.out.print(context[i] + ", "); } } System.out.println();
 * System.out.println();
 */
/// *
System.out.println("similarity of data and computer: "
    + CalculateSimilarity(Cmatrix, GetWordIndex(Lexicon, "data"), GetWordInd
System.out.println("similarity of data and cake:     "
    + CalculateSimilarity(Cmatrix, GetWordIndex(Lexicon, "data"), GetWordInd
System.out.println("similarity of data and cloud:    "
    + CalculateSimilarity(Cmatrix, GetWordIndex(Lexicon, "data"), GetWordInd
System.out.println("similarity of hot and dog:       "
    + CalculateSimilarity(Cmatrix, GetWordIndex(Lexicon, "hot"), GetWordInde
System.out.println();
System.out.print("computer Context Words:    ");
context = GetContext(Cmatrix, GetWordIndex(Lexicon, "computer"), 10);
for (int i = 0; i < context.length; i++) {
  if (context[i] != null) {
    System.out.print(context[i] + ", ");
  }
}
System.out.println();
System.out.print("data Context Words:        ");
context = GetContext(Cmatrix, GetWordIndex(Lexicon, "data"), 10);
for (int i = 0; i < context.length; i++) {
  if (context[i] != null) {
    System.out.print(context[i] + ", ");
  }
}
System.out.println();
System.out.print("cake Context Words:        ");
context = GetContext(Cmatrix, GetWordIndex(Lexicon, "cake"), 10);
for (int i = 0; i < context.length; i++) {
  if (context[i] != null) {
    System.out.print(context[i] + ", ");
  }
}
System.out.println();
System.out.print("cloud Context Words:        ");
context = GetContext(Cmatrix, GetWordIndex(Lexicon, "cloud"), 10);
```

```
  for (int i = 0; i < context.length; i++) {
    if (context[i] != null) {
      System.out.print(context[i] + ", ");
    }
  }
  System.out.println();
  // */

}

/**
 * Method to return the index of a word based on alphabetically sorted Index
 *
 * @param Lexicon Lexicon of words alphabetically sorted
 * @param word    word of interest that we want the index of
 * @return Index of word passed into method
 */
public static int GetWordIndex(ArrayList<String> Lexicon, String word) {
  int found = -1;
  for (int i = 0; i < Lexicon.size(); i++) {
    if (word.equals(Lexicon.get(i))) {
      found = i;
    }
  }
  return found;
}

/**
 * Calculates Cosine Similarity of two words given their index in the lexicon
 * and the Context Matrix
 *
 * @param Cmatrix Context Matrix of PPMI values
 * @param index1  index of first word
 * @param index2  index of second word
 * @return returns Cosine Similarity of the two words of the given indexesa
 */
public static float CalculateSimilarity(float[][] Cmatrix, int index1, int ind

  float numer = 0;
  for (int i = 0; i < Cmatrix.length; i++) {
    numer += Cmatrix[index1][i] * Cmatrix[index2][i];
  }
  float denom1 = 0;
  float denom2 = 0;
  for (int i = 0; i < Cmatrix.length; i++) {
    denom1 += Cmatrix[index1][i];
```

```
      denom2 += Cmatrix[index2][i];
    }
    return numer / ((float) Math.sqrt(denom1) * (float) Math.sqrt(denom2));
  }

  /**
   * This method extracts the context words by performing Cosine Similarity on
   * each combination of word with the word of the given index
   *
   * @param Cmatrix Context Matrix of PPMI values
   * @param index   index of word we want the context words for
   * @param k       number of context words we want to return(returns null spots
   *                if there are not enough context words)
   * @return String vector of context words related to the word of the given ind
   */
  public static String[] GetContext(float[][] Cmatrix, int index, int k) {

    String[] context_vector = new String[k];
    HashMap<Float, Integer> valueindex = new HashMap<>();
    for (int i = 0; i < Lexicon.size(); i++) {
      valueindex.put(CalculateSimilarity(Cmatrix, index, i), i);
    }
    float[] sims = new float[valueindex.keySet().size()];
    int count = 0;
    for (float sim : valueindex.keySet()) {
      sims[count] = sim;
      count++;
    }
    Arrays.sort(sims);
    for (int i = 0; i < k; i++) {
      context_vector[i] = Lexicon.get(valueindex.get(sims[i]));
    }
    return context_vector;
  }

}
```