Assignment 3

CS834 Introduction to Information Retrieval
Fall 2017

Polykarpos Thomadakis

November 8, 2017

Question 6.1

Using theWikipedia 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 (Dices 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 this assignment, I used the code I used in assignment 2 to produce the inverted index. Then, I sort the index alphabetically and pick the first 1000 words and create stems from each one of them using the krovetz stemmer. Once I have created the stems, I create bigrams from them in order to find co-occurences in the given collection and apply Dice's coefficient to create the stemming clusters. I set the threshold to 0.01 and then extracted the connected subgraphs produced by the code. The code used can be seen in listing 1.

```
import os
# import os.path
import csv
from bs4 import BeautifulSoup
import nltk
import argparse
import krovetzstemmer

def hasNumbers(inputString):
  return any(char.isdigit() for char in inputString)

tokenizer = nltk.RegexpTokenizer(r'\w+')

parser = argparse.ArgumentParser('invindex')
group = parser.add_mutually_exclusive_group(required= True)
```

```
group.add_argument('-l','--list', nargs='+', help='List of documents to parese',
    required=False)
group.add_argument('-p','--path', help='Path to directory to parse all of its files',
     required=False)
args = parser.parse_args();

html_files = []
if(args.path):
  root_dir = args.path
  for root,dirs,files in os.walk(root_dir):
    for file in files:
      if file.endswith('.html'):
        html_files.append(os.path.join(root, file))
else:
  html_files = [x for x in args.list if x.endswith('.html')]

inv_index = {}

if not os.path.exists("./inv_file.csv"):

  for file in html_files:
    print 'Indexing file: '+file
    with open(file) as f:
      soup = BeautifulSoup(f.read(),'html.parser')
      tokens=  tokenizer.tokenize(soup.get_text())
      for token in tokens:
        token = token.encode("utf-8")
        if token not in inv_index:
          inv_index[token] = set()
        inv_index[token].add(os.path.split(file)[1])

  outfile = open("inv_file.csv","w")
  for token in sorted(inv_index.iterkeys()):
    if  hasNumbers(token) or token.isupper():
      continue;
    outfile.write(token+",")
    i=0
    outfile.write('"')
    for file in inv_index[token]:
      i+=1
      outfile.write(file)
      if(i != len(inv_index[token])):
        outfile.write(",")
    outfile.write('"\n')

krovetz = krovetzstemmer.Stemmer()

stem_words = {}
index = {}
count = 0;
with open('inv_file.csv') as csvfile:
  for c in csvfile:
    word,files = c.split(",",1)
    files = files.replace("\"","")
    files = files.strip()
    index[word] = files.split(",")
    if count == 1000:
      continue;
    stem = krovetz.stem(word)
    if stem not in stem_words:
      stem_words[stem] = []
    stem_words[stem].append(word)
    count+=1

coef_thres = 0.001
before_dice = []
```

```
after_dice = {}
for stem in sorted(stem_words):
  bigrams = nltk.bigrams(stem_words[stem])
  for a,b in bigrams:
    f_a = index[a]
    len_f_a = len(f_a)
    f_b = index[b]
    len_f_b = len(f_b)
    f_ab = set(index[a]) & set (index[b])
    len_f_ab = len(f_ab)
    before_dice.append((stem,a,b))
    coef = float(2*len_f_ab)/(len_f_a+len_f_b)
    if coef > coef_thres:
      if stem not in after_dice:
        after_dice[stem] = set()
      after_dice[stem].add(a)
      after_dice[stem].add(b)


for stem in after_dice:
  print stem +": {",
  for word in after_dice[stem]:
    print word,
  print "}"
```

Listing 1: Script to extract the stem clusters using Dice's coefficient

The clusters created by the algorithm are presented in table 1.

Table 1: Clusters created using the Dice's coefficient on the small wikipedia collection

academician	Academician,Academicians
abbott	Abbotts,Abbott
abstraction	Abstraction,Abstractions
abugida	Abugidas,Abugida
acquisition	Acquisitions,Acquisition
ace	Aces,Acer
actor	Actors,Actor
acronym	Acronym,Acronyms
access	Access,Accessed
account	Accounting,Account,Accounts
abbey	Abbeys,Abbey
abbreviation	Abbreviation,Abbreviations
accident	Accident,Accidents
abt	Abts,Abt
absurd	Absurdity,Absurd
acre	Acres,Acre
atpase	ATPase,ATPases
activity	Activities,Activity
achievement	Achievements,Achievement
adaboost	AdaBoost,Adaboost

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

For the edit distance calculator I used the python code found in *http://norvig.com/spell-correct.html*, specifically edits1 and edits2, that which given a misspelled word apply 1 and 2 letter transformations to the misspelled word with the purpose of generating a valid one. Once I collect the set of words generated this way, I keep only those that are valid words by checking if they exist in a big collection of words in file *big.txt* found in the repository (adopted from the same site noted above). Once the valid words have been collected, I compute the probability the frequency that each of those words appears, and finally correct the misspelled word with the most frequent word of the set of valid words collected. The code for this assignment appears in listing 2.

```
import re
import sys
from collections import Counter

text = open('big.txt').read()
words = re.findall(r'\w+',text.lower())
word_freq = Counter(words)
num_words =0

if len(sys.argv) !=2:
  print 'Please input word to correct'
  exit()

input_word = sys.argv[1].lower()


for counter in word_freq.values():
  num_words += counter

def edits1(word):
    "All edits that are one edit away from `word`."
    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(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))

def propability(word):
  return float(word_freq[word])/num_words


correct_words = set()

for x in edits1(input_word) or edits2(input_word):
  if x in word_freq:
    correct_words.add(x)

max_ = 0
corrected_word = input_word
for p in correct_words:
  if word_freq[p]>max_:
    max_ = word_freq[p]
    corrected_word = p
```

```
print corrected_word
```

Listing 2: Script to correct a misspelled word based on the noisy channel model

Figure 1 demonstrates the spell correction program listed above.

```
pthomadakis@e-3100-ubuntu:~/Desktop/CS834/A3$ python spell_correction.py lodon
london
pthomadakis@e-3100-ubuntu:~/Desktop/CS834/A3$ python spell_correction.py virgninia
virginia
pthomadakis@e-3100-ubuntu:~/Desktop/CS834/A3$ python spell_correction.py wasington
washington
pthomadakis@e-3100-ubuntu:~/Desktop/CS834/A3$ python spell_correction.py cicago
chicago
pthomadakis@e-3100-ubuntu:~/Desktop/CS834/A3$ python spell_correction.py milwauke
milwaukee
```

Figure 1: Sample output of the spell correction script

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

The code for the Galago search engine can be found in *https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/galagosearch/galagosearch-1.04-src.tar.gz*. The code for the snippet generation resides in the file:
galagosearch-1.04/galagosearch-core/src/main/java/org/galagosearch/core/store/SnippetGenerator.java.
The algorithm can be described as the following 6 steps:

1. Tokenize the document into terms and maintain them and their positions.
2. Find all the terms in the document that match the ones of the query.
3. When a term is matched, retrieve a small region of the document that contains the term.
4. Check for overlapping regions and merge them if any.
5. Keep adding document regions to the snippet, until the maximum size is reached.
6. Combine all regions to a single html string.

Galago includes snippets only when the exact term is found in the document. Thus, when a small document is indexed there is a lower probability that a term will appear and as a result a lower probability that a snippet will be generated. Also, Galago does not seem to use stemming, meaning that it is even less probable for a snippet to be generated. In my oppinion, stemming would increase the quality of snippet generation as it does for searching. If this also fails, it would be a good attempt to just display the first few sentences of the document. Since it has already been marked as relevant, it makes sense that at the introduction of the document there will be some reference relevant to the query term.

Question MLN1

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

Answer

This assignment can be answered directly using the code from question 6.1. To add more value to the code used there I added the stem clusters produced when other association metrics are used, namely Dice's coefficient, EMIM, MIM and Chi-square.
The ten words I randomly chose are:
"academician","university", "mind", "synonym", "monitor", "owe", "top","charge","hit","diverge".
Figure 2 shows the classes created from each metric using threshold 0.01. The code used for this assignment is shown in listing 3

```
import os
# import os.path
import csv
from bs4 import BeautifulSoup
import nltk
import argparse
import krovetzstemmer
import math
import random

chosen_words = [ "academician" ,"university", "mind", "synonym", "monitor", "owe", "
    top" ,"charge" ,"hit" ,"diverge"]
def hasNumbers(inputString):
  return any(char.isdigit() for char in inputString)

def printIndex(index):
  for stem in chosen_words:
    print stem +": {",
    if( stem in index):
      for word in index[stem]:
        print word,
      print "}"
    else:
      print "}"

def dice(a,b):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  # before_dice.append((stem,a,b))
  coef = float(len_f_ab)/(len_f_a+len_f_b)
  if coef > coef_thres:
    return True
  return False

def mim(a,b):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  coef = float(len_f_ab)/(len_f_a*len_f_b)
  if coef > coef_thres:
    return True
  return False

def emim(a,b,N):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
```

```
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  if len_f_ab == 0:
    return False
  coef = len_f_ab*math.log(N*(float(len_f_ab)/(len_f_a*len_f_b)))
  if coef > coef_thres:
    return True
  return False

def chi_sqr(a,b,N):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  coef = len_f_ab-1/float(N)*len_f_a*len_f_b
  coef = coef*coef/(len_f_a*len_f_b)
  if coef > coef_thres:
    return True
  return False

tokenizer = nltk.RegexpTokenizer(r'\w+')

parser = argparse.ArgumentParser('invindex')
group = parser.add_mutually_exclusive_group(required= True)
group.add_argument('-l','--list', nargs='+', help='List of documents to parese',
    required=False)
group.add_argument('-p','--path', help='Path to directory to parse all of its files',
     required=False)
args = parser.parse_args();

html_files = []
if(args.path):
  root_dir = args.path
  for root,dirs,files in os.walk(root_dir):
    for file in files:
      if file.endswith('.html'):
        html_files.append(os.path.join(root, file))
else:
  html_files = [x for x in args.list if x.endswith('.html')]

inv_index = {}

if not os.path.exists("./inv_file.csv"):

  for file in html_files:
    print 'Indexing file: '+file
    with open(file) as f:
      soup = BeautifulSoup(f.read(),'html.parser')
      tokens=  tokenizer.tokenize(soup.get_text())
      for token in tokens:
        token = token.encode("utf-8")
        if token not in inv_index:
          inv_index[token] = set()
        inv_index[token].add(os.path.split(file)[1])

  outfile = open("inv_file.csv","w")
  for token in sorted(inv_index.iterkeys()):
    if  hasNumbers(token) or token.isupper():
      continue;
    outfile.write(token+",")
    i=0
    outfile.write('"')
    for file in inv_index[token]:
      i+=1
```

```
        outfile.write(file)
        if(i != len(inv_index[token])):
          outfile.write(",")
      outfile.write('"\n')

krovetz = krovetzstemmer.Stemmer()

stem_words = {}
index = {}
count = 0;
with open('inv_file.csv') as csvfile:
  for c in csvfile:
    word,files = c.split(",",1)
    files = files.replace("\"","")
    files = files.strip()
    index[word] = files.split(",")
    # if count == 1000:
      # continue;
    stem = krovetz.stem(word)
    if stem not in stem_words:
      stem_words[stem] = []
    stem_words[stem].append(word)
    # count+=1
# print chosen_words
coef_thres = 0.01
dice_index = {}
mim_index = {}
emim_index = {}
chi_sqr_index = {}
for stem in sorted(stem_words):
  bigrams = nltk.bigrams(stem_words[stem])
  for a,b in bigrams:
    if dice(a,b):
      if stem not in dice_index:
        dice_index[stem] = set()
      dice_index[stem].add(a)
      dice_index[stem].add(b)
    if mim(a,b):
      if stem not in mim_index:
        mim_index[stem] = set()
      mim_index[stem].add(a)
      mim_index[stem].add(b)
    if emim(a,b,len(html_files)):
      if stem not in emim_index:
        emim_index[stem] = set()
      emim_index[stem].add(a)
      emim_index[stem].add(b)
    if  chi_sqr(a,b,len(html_files)):
      if stem not in chi_sqr_index:
        chi_sqr_index[stem] = set()
      chi_sqr_index[stem].add(a)
      chi_sqr_index[stem].add(b)
# print "N = ",len(html_files)
print "========== Dice ==========="
printIndex(dice_index)
print "========== MIM ==========="
printIndex(mim_index)
print "========== EMIM ==========="
printIndex(emim_index)
print "========== CHI_SQR ==========="
printIndex(chi_sqr_index)
```

Listing 3: Script to generate stem classes for the 10 chosen words

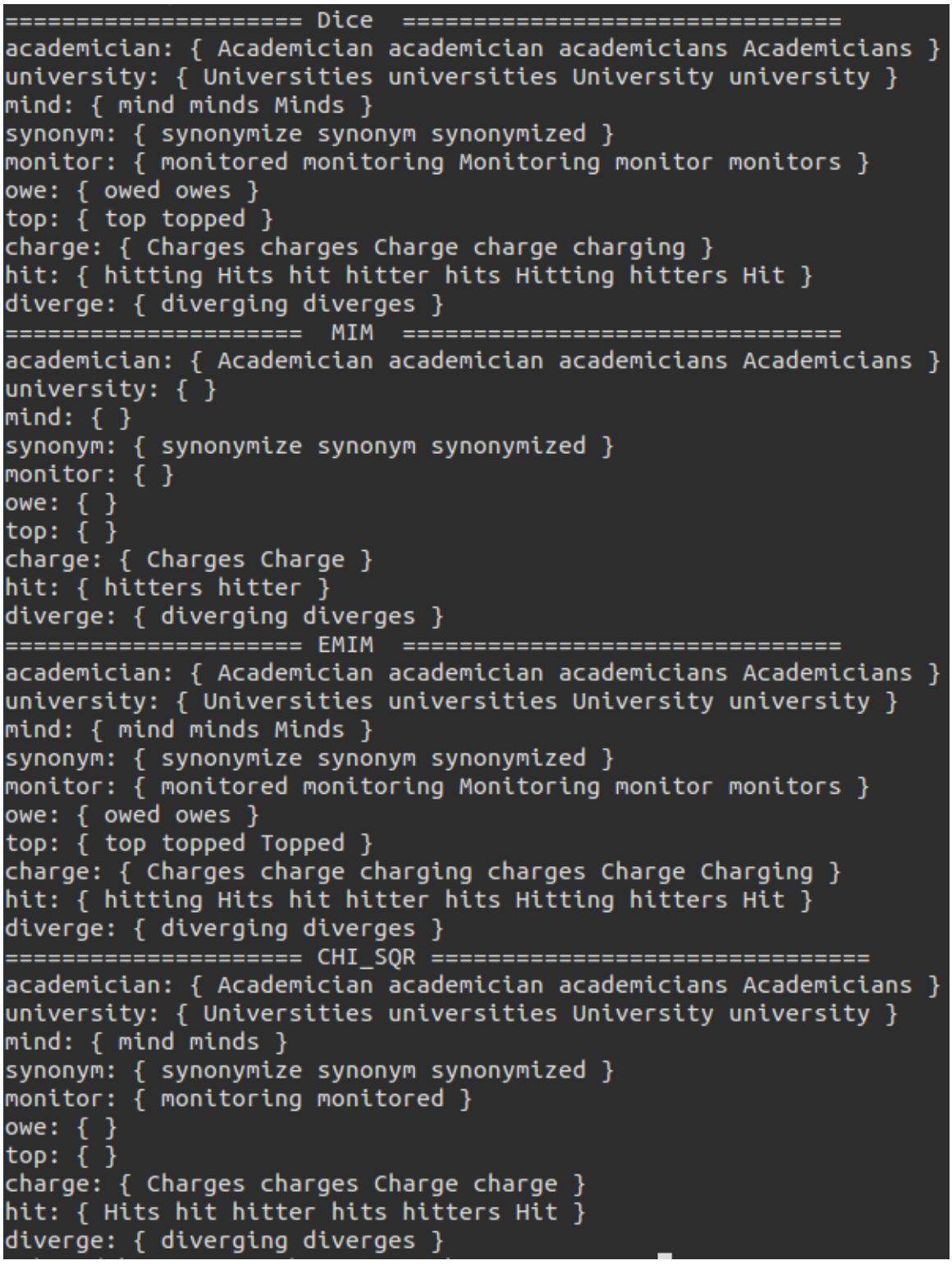

Figure 2: Stem classes produced for different association metrics

Question MLN2

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

Answer

The script that was written for this assignment can be found in listing 4. It uses the inverted index of the previous assignments (or regenerates it if it does not exist) to calculate the association measures using the metrics: MIM, EMIM, Chi-squared and Dice's coefficient. The ten words I randomly chose are: "instrument","university","book","America","fish","dinosaur","car","food","hit","soccer".

```
import os
import csv
from bs4 import BeautifulSoup
import nltk
import argparse
import krovetzstemmer
import math

chosen_words = [ "instrument" ,"university", "book", "America", "fish", "dinosaur", "
    car" ,"food" ,"hit" ,"soccer"]

def hasNumbers(inputString):
  return any(char.isdigit() for char in inputString)

def sortByValue(tup):
  return tup[1]

def insertAssociate(a,b,metric,coef):
  if a not in associates[metric]:
    associates[metric][a] = []
  associates[metric][a].append([b,coef])

def printIndex(index):
  for stem in chosen_words:
    print stem +": {",
    if( stem in index):
      for word in index[stem]:
        print word,
      print "}"
    else:
      print "}"

def dice(a,b):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  coef = float(len_f_ab)/(len_f_a+len_f_b)
  insertAssociate(a,b,'dice',coef)

def mim(a,b):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  coef = float(len_f_ab)/(len_f_a*len_f_b)
  insertAssociate(a,b,'mim',coef)

```

```
def emim(a,b,N):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  if len_f_ab == 0:
    coef = 0.0
    insertAssociate(a,b,'emim',coef)
    return
  coef = len_f_ab*math.log(N*(float(len_f_ab)/(len_f_a*len_f_b)))
  insertAssociate(a,b,'emim',coef)

def chi_sqr(a,b,N):
  f_a = index[a]
  len_f_a = len(f_a)
  f_b = index[b]
  len_f_b = len(f_b)
  f_ab = set(index[a]) & set (index[b])
  len_f_ab = len(f_ab)
  coef = len_f_ab-1/float(N)*len_f_a*len_f_b
  coef = coef*coef/(len_f_a*len_f_b)
  insertAssociate(a,b,'chi',coef)

tokenizer = nltk.RegexpTokenizer(r'\w+')

parser = argparse.ArgumentParser('invindex')
group = parser.add_mutually_exclusive_group(required= True)
group.add_argument('-l','--list', nargs='+', help='List of documents to parese',
    required=False)
group.add_argument('-p','--path', help='Path to directory to parse all of its files',
     required=False)
args = parser.parse_args();

html_files = []
if(args.path):
  root_dir = args.path
  for root,dirs,files in os.walk(root_dir):
    for file in files:
      if file.endswith('.html'):
        html_files.append(os.path.join(root, file))
else:
  html_files = [x for x in args.list if x.endswith('.html')]

inv_index = {}

if not os.path.exists("./inv_file.csv"):

  for file in html_files:
    print 'Indexing file: '+file
    with open(file) as f:
      soup = BeautifulSoup(f.read(),'html.parser')
      tokens=  tokenizer.tokenize(soup.get_text())
      for token in tokens:
        token = token.encode("utf-8")
        if token not in inv_index:
          inv_index[token] = set()
        inv_index[token].add(os.path.split(file)[1])

  outfile = open("inv_file.csv","w")
  for token in sorted(inv_index.iterkeys()):
    if  hasNumbers(token) or token.isupper():
      continue;
    outfile.write(token+",")
    i=0
```

```
    outfile.write('"')
    for file in inv_index[token]:
      i+=1
      outfile.write(file)
      if(i != len(inv_index[token])):
        outfile.write(",")
    outfile.write('"\n')

krovetz = krovetzstemmer.Stemmer()

stem_words = {}
index = {}
count = 0;
with open('inv_file.csv') as csvfile:
  for c in csvfile:
    word,files = c.split(",",1)
    files = files.replace("\"","")
    files = files.strip()
    index[word] = files.split(",")

words = index.keys()

associates = {}
associates['dice']  = {}
associates['mim']   = {}
associates['emim']  = {}
associates['chi'] = {}


for choice in chosen_words:
  for word in words:
    if word != choice:
      dice(choice,word)
      mim(choice,word)
      emim(choice,word,len(html_files))
      chi_sqr(choice,word,len(html_files))

per_word_associates = {}
out = open("associates","w")
for choice in chosen_words:
  per_word_associates[choice] = {}
  per_word_associates[choice]['dice'] = []
  per_word_associates[choice]['mim']  = []
  per_word_associates[choice]['emim'] = []
  per_word_associates[choice]['chi']  = []
  per_word_associates[choice]['dice'] = sorted(associates['dice'][choice],key=
    sortByValue,reverse=True)[:10]
  per_word_associates[choice]['mim'] = sorted(associates['mim'][choice],key=
    sortByValue,reverse=True)[:10]
  per_word_associates[choice]['emim'] = sorted(associates['emim'][choice],key=
    sortByValue,reverse=True)[:10]
  per_word_associates[choice]['chi'] = sorted(associates['chi'][choice],key=
    sortByValue,reverse=True)[:10]
  out.write(choice+"\n")
  out.write("Dice,MIM,EMIM,Chi\n")
  for i in range(0,10):
    out.write(per_word_associates[choice]['dice'][i][0]+","+per_word_associates[
    choice]['mim'][i][0]+","+per_word_associates[choice]['emim'][i][0]+","+
    per_word_associates[choice]['chi'][i][0]+"\n")
```

Listing 4: Script to generate top 10 associated words for the 10 chosen words

The resulting associated words are shown in tables 2-11. We can see that the associated words can differ significantly depending on the metric that is used to measure association. The most significant difference in the table is noticed on the MIM metric, where almost none of the words is common with the other ones. That is probably caused by the fact that this metric favors low-frequency terms,

since $n_a * n_b$ will increase very fast for frequent words a,b but their co-occurrence remains the same. As a result, the total value will be very low compared to low-frequency.

Table 2: Associated words for word "instrument"

instrument			
Dice	MIM	EMIM	Chi
instruments	Mortem	instruments	instruments
signature	Serwaczynski	musical	timbres
Perspectives	conductive	music	Buggles
Furrykef	Jbonneau	such	ney
scores	Desinicization	some	Postmodernism
Instrument	Arhielanto	used	disclosed
sensitive	Dentine	up	Ulmanor
musical	Artisan	heavily	halts
Face	Dannycali	different	Crazed
timbres	SatellitesHidden	various	Maxillofacial

Table 3: Associated words for word "university"

university			
Dice	MIM	EMIM	Chi
institution	Tootie	University	institution
universities	Pamplemousse	education	universities
campus	Mortem	college	education
academic	Velocisaurus	institution	campus
education	Cheiro	students	academic
college	Babineau	campus	college
student	Wingwangwo	academic	postgraduate
colleges	Malfatti	universities	Faculties
Engineering	majr	College	University
students	Thegreyanomaly	student	colleges

Table 4: Associated words for word "book"

book			
Dice	MIM	EMIM	Chi
published	Simka	published	published
books	Mortem	books	books
story	SparrowsWing	story	story
written	fiction	his	novels
novel	Pvasiliadis	written	novel
author	Mucks	he	comic
wrote	Norilana	that	author
Books	Grecque	novel	fiction
series	author	they	Publication
Book	equilateral	author	wrote

Table 5: Associated words for word "America"

America			
Dice	MIM	EMIM	Chi
North	Tootie	North	North
American	Mortes	American	Europe
Europe	Mortem	New	Asia
New	seamier	show	American
York	Sergeants	States	Oceania
through	Pinkerton	hide	Mexico
States	Mucko	var	Brazil
most	Zizaniopsis	tocShowText	York
South	Velocisaurus	showTocToggle	Africa
began	Amirthanayagam	tocHideText	New

Table 6: Associated words for word "fish"

fish			
Dice	MIM	EMIM	Chi
Actinopterygii	hukim	species	Actinopterygii
Fish	Keulemans	Actinopterygii	FishBase
Chordata	desiccant	water	Ranier
freshwater	Jabr	Chordata	Osteichthyes
species	Sennet	Fish	Fishes
Phylum	Percopsiformes	Phylum	Fish
fins	bellow	classification	Pauly
Animalia	arsēn	Animalia	fins
FishBase	Protista	Scientific	fin
Ranier	bobwhite	Family	Froese

Table 7: Associated words for word “dinosaur”

dinosaur			
Dice	MIM	EMIM	Chi
Dinosaurs	Velocisaurus	dinosaurs	Dinosaurs
dinosaurs	Cucumber	Dinosaurs	dinosaurs
theropod	Gurche	Firsfron	theropod
Dinosauria	jil	theropod	Dinosauria
Cretaceous	candelariensis	Dinosauria	Saurischia
Sauropsida	sauropodomorph	Spencer	Cretaceous
Saurischia	edwardsorum	Cretaceous	Megalosaurus
Superorder	Zanclodon	Sauropsida	Theropoda
Jurassic	Baricco	Dinosaur	Sauropsida
Dodson	Diplodocus	Superorder	Dodson

Table 8: Associated words for word “car”

car			
Dice	MIM	EMIM	Chi
cars	Mortem	into	cars
vehicle	seamier	cars	vehicle
Car	Sergeants	they	Car
driver	SparrowsWing	through	driver
driving	Hafid	their	driving
vehicles	Mwstorlie	such	vehicles
hotel	dna	been	wheels
road	equilateral	new	hotel
get	titanium	have	automobile
drive	Alor	two	Shoppes

Table 9: Associated words for word “food”

food			
Dice	MIM	EMIM	Chi
drink	Behaalotecha	have	drink
water	water	they	consume
themselves	Sergeants	there	diet
eat	canes	than	foods
roughly	Food	their	diarrhea
Food	Hif	more	prey
source	eat	most	roughly
significant	hukim	well	Food
low	Sameerkale	them	eating
areas	Vernooykill	are	water

Table 10: Associated words for word “hit”

hit			
Dice	MIM	EMIM	Chi
song	comically	song	song
re	Haynos	re	re
behind	Valli	second	hits
debut	Sedaka	released	getting
next	Shockey	their	fans
getting	Mucks	his	behind
featured	Mwstorlie	single	next
single	Vipiteno	off	single
fans	Akroyd	next	Billboard
video	Alou	up	debut

Table 11: Associated words for word "soccer"

soccer			
Dice	MIM	EMIM	Chi
Football	Karaliopolous	football	Football
Goals	Hig	Football	football
football	Visvliet	league	Goals
Gls	Bobae	club	Gls
App	Estay	team	App
league	NellieMcKay	Club	league
footballers	Ide	Goals	footballers
domestic	Spudders	Cup	domestic
goals	Thosol	domestic	club
Playing	Seade	goals	goals