Assignment 3

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

November 10, 2016

Contents

1 Question 6.1 **3**
1.1 Question . . . 3
1.2 Approach . . . 3
1.3 Results . . . 5

2 Question 6.2 **7**
2.1 Question . . . 7
2.2 Approach . . . 7
2.3 Results . . . 7

3 Question 6.5 **8**
3.1 Question . . . 8
3.2 Answer . . . 8

4 Question MLN1 **9**
4.1 Question . . . 9
4.2 Approach . . . 9
4.3 Results . . . 9

5 Question MLN2 **11**
5.1 Question . . . 11
5.2 Approach . . . 11
5.3 Results . . . 11

6 Appendix **16**
6.1 Code listings . . . 16

7 References **22**

List of Figures

1 Graph Visualization of Stem Clusters . . . 10

List of Tables

1 Calculated values for “running” . . . 12
2 Calculated values for “calculation” . . . 12
3 Calculated values for “color” . . . 12
4 Calculated values for “horse” . . . 13
5 Calculated values for “sky” . . . 13
6 Calculated values for “railroad” . . . 13
7 Calculated values for “calendar” . . . 14
8 Calculated values for “airplane” . . . 14
9 Calculated values for “ocean” . . . 14
10 Calculated values for “bicycle” . . . 15

1 Question 6.1

1.1 Question

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.

1.2 Approach

The `stem.py` and `cluster.py` scripts, found in Listings 2 and 4, were used to define the initial stem classes from the list of 1,000 words. This list was created using words from a previously constructed inverted index in the small Wikipedia collection.

After the stem classes were compiled, Dice's coefficient was calculated for all pairs of terms within each class, and then pairs of co-occurring stemmed words that had a resulting score below a threshold of 0.1 were dropped from their class, resulting in a tight cluster of co-occurring terms.

1.2.1 Initial Classes

The first 1,000 words stemmed by using the nltk SnowballStemmer [1] resulted in the following stem classes:

affair: Affair, Affaires, Affairs
ae: Aes, Ae
ad: Adding, Added, Adly
advantag: Advantages, Advantage
agil: Agile, Agilent, Agil
ah: Ah, Ahli, Ahly
address: Addressed, Addressing, Addresses, Address
agent: Agents, Agent
ahr: AhR, Ahr
ador: Adored, Adoration, Adoring, Adore
aggreg: Aggregate, Aggregation
agri: Agris, Agri
aesthet: Aesthetics, Aesthetes, Aesthetic
agenc: Agency, Agencies, Agence
afil: Afiler, Afil
affin: Affine, Affinity
afghan: Afghan, Afghans
adz: Adzeds, Adze, Adz

aflatoxin: Aflatoxin, Aflatoxins
aeronaut: Aeronautical, Aeronautics
affleck: Affleck, Afflecks
agricultura: Agriculturae, Agricultura
advanc: Advances, Advance, Advancement, Advanced, Advancing
agricultur: Agricultural, Agriculture
adolesc: Adolescents, Adolescent, Adolescence
aeon: Aeon, Aeons
advert: Advert, Adverts
adob: Adobe, Adobes
ag: Agly, Ag
advoc: Advocate, Advocating, Advocates
aflah: Aflah, AflaH
advis: Advisers, Adviser
advers: Adverse, Adversity
admir: Admirals, Admiral
agit: Agitation, Agitator, Agitators
addi: Addis, Addie, Addy
adi: Adi, Ady
adhes: Adhesion, Adhesive
adher: Adherence, Adherents
agricola: Agricolae, Agricola
affili: Affiliations, Affiliation, Affiliate, Affiliates, Affiliated
afford: Affordable, Affordance
aero: Aeros, Aero
adolph: Adolph, Adolphe
agreement: Agreements, Agreement
agua: Aguas, Agua
adept: Adepts, Adeption, Adept
adder: Adderly, Addere, Adders, Adder
aeroplan: Aeroplanes, Aeroplane
advisor: Advisors, Advisor
adri: Adri, Adrie
affect: Affected, Affects, Affecting, Affect, Affections, Affection
advertis: Advertiser, Advertisements, Advertising, Advertised
adventur: Adventure, Adventurers, Adventurous, Adventures
adil: Adil, Adils
afterward: Afterward, Afterwards
addit: Additive, Additional, Addition, Additionally, Additions, Additives
agn: Agnes, Agne
agi: Agis, Agy
administr: Administrator, Administratively, Administration, Administrators, Administred, Administrations, Administrative
affirm: Affirms, Affirmation, Affirmed, Affirmative
age: Aging, Ageing, Age, Ages, Agee, Aged
adopt: Adoptive, Adopt, Adoption, Adopted
adult: Adults, Adult, Adultism
admiss: Admissions, Admission
admit: Admits, Admittedly, Admitting, Admitted
adjust: Adjustment, Adjusting, Adjustable, Adjusted, Adjust
agglomer: Agglomerations, Agglomeration

afternoon: Afternoon, Afternoons
agenda: Agendas, Agenda
african: Africans, African, Africanism
adel: Adel, Adele
aerosol: Aerosol, Aerosoles
addam: Addams, Addam
addict: Addiction, Addict

1.2.2 Stem Clusters

With a Dice's coefficient threshold value of 0.1 applied to filter out the weakly-linked stem class elements, the following are the remaining stem clusters:

ah: Ahli, Ahly
adolesc: Adolescents, Adolescence
adopt: Adopt, Adoption
agua: Aguas, Agua
agricultura: Agriculturae, Agricultura
aflah: Aflah, AflaH
address: Addressing, Addresses
ador: Adoration, Adoring
adventur: Adventure, Adventures
age: Aging, Ageing
agit: Agitators, Agitator, Agitation

1.3 Results

It seems clear that applying a co-occurrence association measure to the stem class creation process reduces the number of classes created overall, even breaking apart some groups that are logically similar. For example, the *addict* class featured the two words Addiction and Addict. One being a different capitalization of the root stem and the other being a modified form of the stem, which shows that the class could have remained together, but this method dropped them from the initial stem classes.

This observation casts some doubt on the usage of a Dice's coefficient threshold of 0.1 to create semantically related stem classes. There are other examples of related word groups being broken apart by the coarseness of this threshold. To test the tuning of this parameter, another test run was conducted with a value of 0.00001 for the threshold. This experimental run produced the following stem clusters:

affair: Affair, Affairs
ah: Ahli, Ahly
agricultura: Agriculturae, Agricultura
agenc: Agency, Agencies
agit: Agitators, Agitator, Agitation
affili: Affiliation, Affiliate, Affiliates, Affiliations
adventur: Adventurers, Adventures, Adventure
agricultur: Agricultural, Agriculture
advoc: Advocate, Advocates
aflah: Aflah, AflaH
advanc: Advance, Advanced

agreement: Agreements, Agreement
agua: Aguas, Agua
adult: Adults, Adult
address: Addressing, Addresses, Address
ador: Adoration, Adoring
afghan: Afghan, Afghans
addit: Addition, Additionally, Additive, Additional, Additives
administr: Administrator, Administratively, Administration, Administrators, Administrations, Administrative
adolesc: Adolescents, Adolescence
adopt: Adopt, Adoption
admir: Admirals, Admiral
african: Africans, Africanism, African
age: Aging, Age, Ages, Ageing
aesthet: Aesthetics, Aesthetic

These also seem to show good grouping for the stem classes but there are still a significant number of classes that were broken up that reasonably could have been kept together in a single cluster. Upon inspecting the data at run time, it appears that some of the terms that were put into stem classes together do not actually co-occur within any documents at all, even though they are semantically linked to the same root word, which explains why they were separated as part of the Dice's coefficient threshold filtering operation. Future use of association measures such as this should take this and other related corpus-specific phenomena into account when attempting to perform this type of contextual analysis.

2 Question 6.2

2.1 Question

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)

2.2 Approach

Peter Norvig's noisy channel spelling correction algorithm [2] was used as the basis for this solution. The `spelling.py` script, found in Listing 6, was created as an implementation of this algorithm. It was written with the Python programming language [3].

A large text file was downloaded from Mr. Norvig's website to calculate language model probability function $P(W)$. The words in the text file were counted and stored in a map that was compressed and saved on disk using the pickle python library [4].

$P(W)$ is calculated with the following formula:

$$P(W) = \frac{C_W}{N}$$

where C_W is the word count for word W and N is the sum of all word counts.

The process of determining a spelling correction is as follows:

1. Take the input word and determine all existing (correctly spelled) words with edit distance one and two.
2. With the assumption that shorter edit distances equate to a higher probability of being the correct intended word, select from the set of words from the previous step the one with the shortest edit distance and highest value for $P(W)$.

2.3 Results

Here is some sample output from the spelling.py script:

```
[mchaney@mchaney-l spelling]$ ./spelling splling
selling
[mchaney@mchaney-l spelling]$ ./spelling sweling
swelling
[mchaney@mchaney-l spelling]$ ./spelling aacck
back
[mchaney@mchaney-l spelling]$ ./spelling panaceu
palace
[mchaney@mchaney-l spelling]$ ./spelling plaec
place
[mchaney@mchaney-l spelling]$ ./spelling intrmdiate
intermediate
[mchaney@mchaney-l spelling]$ ./spelling informatino
information
[mchaney@mchaney-l spelling]$ ./spelling pretende
pretended
[mchaney@mchaney-l spelling]$ ./spelling teh
the
```

Listing 1: spelling.py example output

It is clear that edit distances of one or two cover a great deal of spelling mishaps.

3 Question 6.5

3.1 Question

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.

3.2 Answer

Snippet creation is done by the `SnippetGenerator` class. This class takes as parameters to it's `getSnippet` method the document text as a `String` and a `Set` of `String` query terms, and returns a `String` that is a query-relevant snippet, or summary, of the document.

The snippet generator begins by turning the document text into a list of tokens for processing. The generator then parses these tokens, looking for query term matches, and when it finds a match, it creates a `SnippetRegion` object that stores the location within the document where the query term matched, plus five contextual terms preceding and following each term match. This equates to storing sentence fragments containing query terms.

After collecting all of the regions in the document containing a query term the generator begins constructing the final snippet by adding the SnippetRegions found from the previous step, combining those regions that overlap each other into larger regions, until a final list of `SnippetRegions` is created with total length in terms is no greater than 40 + the length of the last `SnippetRegion` added.

With the final list of `SnippetRegions` the algorithm builds an HTML string containing all the snippets concatenated together for rendering the snippet in a browser while adding `<strong>` tags around each query term match for emphasis.

This approach does not seem like it would work very well with pages with little text content because it requires matching query terms to snippet regions in order to build the snippet, and if there is little text the opportunity for finding good contextual information related to queries drops.

This approach favors regions at the beginning of the document without regard to query context. One way to improve upon this method is to favor regions that contain more query terms. This can be done by counting the number of query terms found in the combined regions and then ordering the snippet generation based on the regions with the highest contained query term counts. This method could cut down the size of the final snippet by choosing regions that contain more query words within the normal extent of 5 terms per query word match, which would allow for a more concise summary of the website as it relates to the user query.

4 Question MLN1

4.1 Question

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

1. For all pairs of words in the stem classes, count how often they co-occur in text windows of W words. W is typically in the range 50-100.
2. Compute a co-occurrence or association metric for each pair. This measures how strong the association is between the words.
3. Construct a graph where the vertices represent words and the edges are between words whose co-occurrence metric is above a threshold T .
4. Find the connected components of this graph. These are the new stem classes.

4.2 Approach

The following were selected as the words to be stemmed using the approach outlined on pp. 191-192 of the textbook:

- affiliates
- affiliation
- affiliated
- fire
- firing
- fired
- changing
- changed
- changes
- change

The python scripts `stem.py` and `mln1.py` were written to complete this task and can be found in Listings 2 and 5, respectively. After performing initial stemming using nltk's SnowballStemmer [1], Dice's coefficient was used as a measure of word co-occurrence with a window size of the entire document. A threshold of 0.1 was used to determine which pairs of words should have edges in the resulting word co-occurrence graph.

4.3 Results

The NetworkX library [5] was used to construct the graph found in Figure 1 and to find the connected components.

The initial stem classes after applying the Dice's coefficient threshold of 0.1:

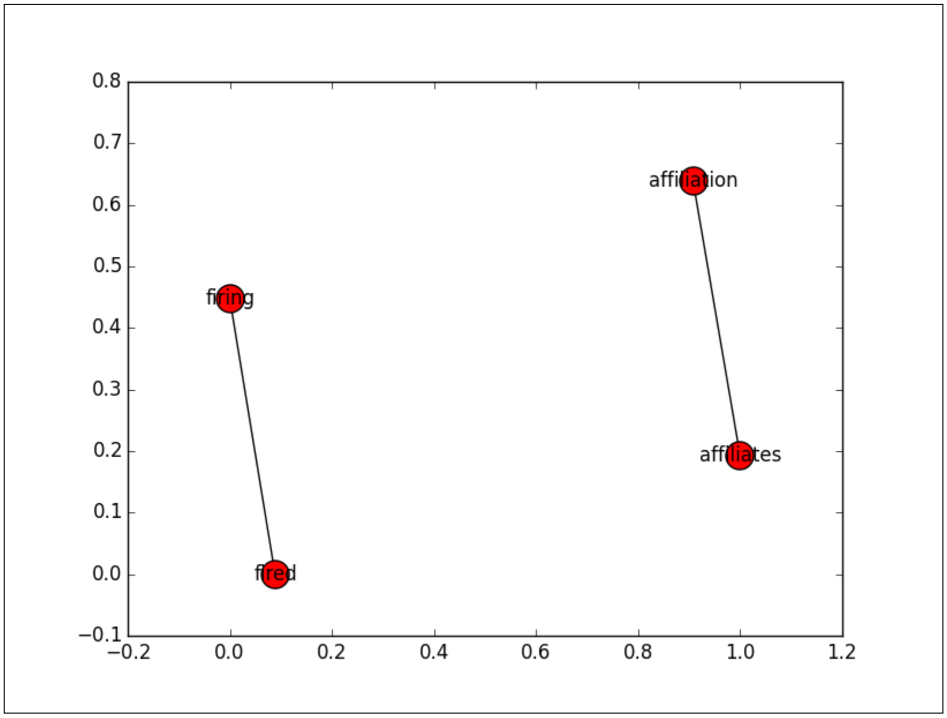

Figure 1: Graph Visualization of Stem Clusters

fire: fire, firing, fired
affili: affiliation, affiliates, affiliated
chang: changing, changed, changes, change

The final connected components from the resulting graph:

fire: firing, fired
affili: affiliation, affiliates

5 Question MLN2

5.1 Question

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)

5.2 Approach

The python script `calc.py`, found in Listing 7, was used to complete this task. It initially uses a previously constructed simple inverted index to calculate the association measures Mutual Information (MIM), Expected Mutual Information ($EMIM$), Chi-squared (χ^2), and Dice's coefficient, for the following words:

- running
- calculation
- color
- horse
- sky
- railroad
- calendar
- airplane
- ocean
- bicycle

5.3 Results

For many of these terms Dice's coefficient and Chi-squared perform admirably, resulting in words that are topically similar, while the MIM and $EMIM$ seem to find only proper nouns. This echoes the author's observation from the textbook that these two formulas favor rare words over more common terms, which explains the occurrence of many proper nouns in their results.

After calculating the measures for all terms that co-occur with the chosen terms the top ten words for each measure were compiled into the tables below:

running			
MIM	*EMIM*	χ^2	*Dice*
Tootie	Tootie	long	ran
Mortes	Mortes	only	long
Mortem	Mortem	but	could
Alsab	Alsab	over	run
Titulus	Titulus	two	started
Cruguet	Cruguet	could	ever
defensed	defensed	had	changed
Vipiteno	Vipiteno	time	old
Velocisaurus	Velocisaurus	In	opening
Pedophilia	Pedophilia	into	end

Table 1: Calculated values for “running”

calculation			
MIM	*EMIM*	χ^2	*Dice*
unknot	unknot	proleptic	usefulness
Jabr	Jabr	Casull	Spoon
humbler	humbler	Exiguus	computed
Bcbell	Bcbell	usefulness	compute
Marxschen	Marxschen	Spoon	calculate
Ethiopic	Ethiopic	computed	formulas
reconciling	reconciling	falsify	proleptic
anthropologie	anthropologie	compute	Casull
dampens	dampens	calculate	Exiguus
provable	provable	formulas	falsify

Table 2: Calculated values for “calculation”

color			
MIM	*EMIM*	χ^2	*Dice*
roadrunners	roadrunners	Depreciated	Depreciated
Tootie	Tootie	param	param
SparrowsWing	SparrowsWing	Alter	red
equilateral	equilateral	Abilities	colors
Sleepwalking	Sleepwalking	ego	black
Editorials	Editorials	red	Comics
Alor	Alor	colors	infobox
Antaheen	Antaheen	white	white
mutantsHidden	mutantsHidden	black	image
Caucasoids	Caucasoids	NGV17	ego

Table 3: Calculated values for “color”

horse			
MIM	*EMIM*	χ^2	*Dice*
Alsab	Alsab	thoroughbred	Horse
Cruguet	Cruguet	Equestrianism	thoroughbred
haoma	haoma	Zafonic	Stakes
pompeux	pompeux	Stakes	Equestrianism
iro	iro	racehorse	Zafonic
Awaystay	Awaystay	racehorses	racehorse
Beaurepaire	Beaurepaire	Thoroughbred	racehorses
Jardim	Jardim	Horse	Thoroughbred
Agnihotra	Agnihotra	Harness	Trainer
Legate	Legate	Slipper	racing

Table 4: Calculated values for “horse”

sky			
MIM	*EMIM*	χ^2	*Dice*
mailings	mailings	binoculars	Astronomy
Hig	Hig	ChristalPalace	bright
Alor	Alor	calvus	wind
Jeremywn	Jeremywn	Arcus	items
Kert01	Kert01	incus	eclipse
Chikubasho	Chikubasho	mackerel	visible
Jabr	Jabr	æŰĞ	speeds
Sennen	Sennen	Achiu31	gravity
iro	iro	Colares	Telescope
Cucumber	Cucumber	Cycles	objects

Table 5: Calculated values for “sky”

railroad			
MIM	*EMIM*	χ^2	*Dice*
Timken	Timken	railroads	Railroad
Hegins	Hegins	Railroad	railroads
Sameerkale	Sameerkale	Railroads	Slambo
ContrÃťle	ContrÃťle	Slambo	rail
Friedensburg	Friedensburg	trackage	freight
WLVN	WLVN	freight	Railroads
Harrisonville	Harrisonville	rail	Railway
C420	C420	Railway	gauge
C425	C425	gauge	Lines
C424	C424	mae	train

Table 6: Calculated values for “railroad”

calendar			
MIM	*EMIM*	χ^2	*Dice*
27a	27a	Gregorian	Gregorian
SÃűrenstam	SÃűrenstam	liturgics	liturgical
Jabr	Jabr	Lunisolar	calendars
escalade	escalade	Tixity	lunar
Tankersley	Tankersley	Calendarists	Persia
Desinicization	Desinicization	commemorations	Dionysius
Kikadue	Kikadue	calendars	Calendar
Munaishy	Munaishy	liturgical	Frysk
Mandarina999	Mandarina999	Calendars	leap
Ethiopic	Ethiopic	alms	Babylonian

Table 7: Calculated values for “calendar”

airplane			
MIM	*EMIM*	χ^2	*Dice*
USAFE	USAFE	MiG	MiG
Hiu	Hiu	maneuverability	plane
Alor	Alor	canopy	altitude
Plegovini	Plegovini	motherships	jets
bellow	bellow	Thunderstreak	maneuverability
RandalSchwartz	RandalSchwartz	underwing	pilots
Ufology	Ufology	84F	canopy
jib	jib	wrinkling	jet
Zhaoguo	Zhaoguo	Filmsite	Aviation
fashionably	fashionably	Maneuver	fuselage

Table 8: Calculated values for “airplane”

ocean			
MIM	*EMIM*	χ^2	*Dice*
Cheiro	Cheiro	Anstey	Antarctic
Tracysurf	Tracysurf	Bruticus	sail
Alvarolima	Alvarolima	DMeyering	floating
Dejima	Dejima	adverb	biodiversity
Sennet	Sennet	Paukrus	Fishing
iro	iro	tusk	ecosystems
Rockheights	Rockheights	bodyboarding	oceans
barque	barque	Orinoco	locked
bellow	bellow	plankton	temporarily
Ryanjunk	Ryanjunk	shack	seal

Table 9: Calculated values for “ocean”

bicycle			
MIM	*EMIM*	χ^2	*Dice*
Sergeants	Sergeants	racer	racer
Backhuys	Backhuys	cyclists	cyclists
Moetus	Moetus	PalmarÃĺs	Discipline
Spudders	Spudders	Drunt	PalmarÃĺs
Spilsby	Spilsby	Discipline	Drunt
Dockx	Dockx	Giro	cycling
MountainBikes	MountainBikes	U23	Giro
Klostergaard	Klostergaard	ProTeam	Friis
Lengerhane	Lengerhane	Friis	UCI
Khari	Khari	cycling	Rider

Table 10: Calculated values for “bicycle”

6 Appendix

6.1 Code listings

```
#!/usr/bin/env python

import collections
import itertools
from data import words
from nltk.stem import *

class Result(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b
        self.sa = set(words[a])
        self.sb = set(words[b])
        self.sab = self.sa.intersection(self.sb)
        self.na = float(len(self.sa))
        self.nb = float(len(self.sb))
        self.nab = float(len(self.sab))
        self.dice = self.nab / (self.na + self.nb)

    def getdice(self):
        return self.dice

    def __repr__(self):
        return '({},{}) Dice {}'.format(self.a, self.b, self.dice)


def getstems(wordlist):
    # stem the first 1k words
    stemmer = SnowballStemmer('english')
    stems = {word: stemmer.stem(unicode(word, 'utf-8')) for word in wordlist}

    # count the stems to find duplicates
    vals = collections.Counter(stems.values())

    # reduce stem map to those that stemmed to the same stem
    dupkeys = {key: val for key, val in stems.items() if vals[val] > 1}

    # create new map that is the stem pointing to all terms that stemmed to it
    classes = {}
    for pair in itertools.combinations(dupkeys.items(), 2):
        k1 = pair[0][0]
        k2 = pair[1][0]
        v1 = pair[0][1]
        v2 = pair[1][1]
        if v1 == v2:
            if not classes.has_key(v1):
                classes[v1] = set()
            classes[v1].add(k1)
            classes[v1].add(k2)
    print '%d duplicate stems' % len(classes)

    # calculate Dice's coefficient for each term with the same stem
    results = {}
    for stem, terms in classes.items():
        for pair in itertools.combinations(terms, 2):
            t1 = pair[0]
            t2 = pair[1]
            if not results.has_key(stem):
                results[stem] = set()
            results[stem].add(Result(t1, t2))
    return classes, results


def convert(filtered):
    converted = {}
    for stem, fres in filtered.items():
        if len(fres) > 0:
            res = set()
            for result in fres:
                res.add(result.a)
                res.add(result.b)
```

```python
                converted[stem] = res
    return converted


def printtables(fname, resultset, mode='w'):
    head = '\\begin{table}[h!]\n\\centering\n\\begin{tabular}{ | c | c | }\n'
    foot = '\\hline\n\\end{tabular}\n\\caption{``%s\'\' stems}\n\\label{tab:%s}\n\\end{table}\n'
    print 'writing tables'
    with open(fname, mode) as outfile:
        outfile.write('\n\n\n\\noindent\n')
        for stemclass in resultset.items():
            outfile.write(stemclass[0])
            outfile.write(': ')
            outfile.write(', '.join(stemclass[1]))
            outfile.write('\\\\\n')
```

Listing 2: stem.py

```python
import cPickle

try:
    print 'loading cached word map'
    words = cPickle.load(open('wordcount.p', 'rb'))
except IOError:
    words = {line.split()[0]: line.split()[1:] for line in open('invidx.dat').readlines()}
    cPickle.dump(words, open('wordcount.p', 'wb'))
    words = cPickle.load(open('wordcount.p', 'rb'))
N = float(sum(len(docs) for docs in words.values()))
```

Listing 3: data.py

```python
#!/usr/bin/env python

from stem import *

# skipping first 13,000 terms because they are all numeric
# and won't meet language probability expectations
first1k = sorted(words.keys())[13000:14000]

classes, results = getstems(first1k)
printtables('clustertab.tex', classes)

filtered = {stem: [result for result in rset if result.getdice() > 0.1] for stem, rset in results.items()}
converted = convert(filtered)
printtables('clustertab.tex', converted, mode='a')

filtered = {stem: [result for result in rset if result.getdice() > 0.00001] for stem, rset in results.items()}
converted = convert(filtered)
printtables('clustertab.tex', converted, mode='a')
```

Listing 4: cluster.py

```python
#!/usr/bin/env python

from stem import *
import networkx as nx
import matplotlib.pyplot as plt

wordlist = [
    'affiliates',
    'affiliation',
    'affiliated',
    'fire',
    'firing',
    'fired',
    'changing',
    'changed',
    'changes',
    'change'
]

def buildgraph(filtered):
    graph = nx.Graph()
    for stem, results in filtered.items():
        for result in results:
            graph.add_edge(result.a, result.b)
    return graph

classes, results = getstems(wordlist)
printtables('mln1tab.tex', classes)

filtered = {stem: [result for result in rset if result.getdice() > 0.1] for stem, rset in results.items()}
converted = convert(filtered)
printtables('mln1tab.tex', converted, mode='a')

graph = buildgraph(filtered)
figure = plt.figure()
nx.draw_networkx(graph)
figure.savefig('figure1.png')
```

Listing 5: mln1.py

```python
#!/usr/bin/env python

import re
import sys
import cPickle
from collections import Counter


def get_words():
    try:
        return cPickle.load(open('words.p', 'rb'))
    except IOError:
        wordmap = Counter(re.findall(r'\w+', open('big.txt').read().lower()))
        cPickle.dump(wordmap, open('words.p', 'wb'))
        return cPickle.load(open('words.p', 'rb'))


words = get_words()
N = sum(words.values())


def exists(wordset):
    return set([word for word in wordset if word in words])


def prob(word):
    return float(words[word]) / float(N)


def edit1(w):
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    deletes    = [w[:i]+w[i+1:]                for i in range(len(w))]
    transposes = [w[:i]+w[i+1]+w[i]+w[i+2:]    for i in range(len(w)-1)]
    replaces   = [w[:i]+l+w[i+1:]              for i in range(len(w)) for l in letters]
    inserts    = [w[:i]+l+w[i:]                for i in range(len(w)+1) for l in letters]
    return set(deletes + transposes + replaces + inserts)


def edit2(word):
    e2 = [edit1(w) for w in edit1(word)]
    return [item for sublist in e2 for item in sublist]


def parse(word):
    return exists([word]) or exists(edit1(word)) or exists(edit2(word)) or [word]


def correct(word):
    return max(parse(word), key=prob)


if __name__ == '__main__':
    print correct(sys.argv[1])
```

Listing 6: spelling.py

```python
#!/usr/bin/env python

import cPickle
import math
from data import words, N

class Result(object):
    def __init__(self, a, b):
        """calculate MIM, EMIM, Chi-square, and Dice's coefficient for words a and b.
        mim  = nab / (na * nb)
        emim = nab * log [ N * nab / ( na * nb ) ]
        x2   = ( nab - ( 1 / N ) * na * nb )^2 / ( na * nb )
        dice = nab / ( na + nb )"""
        self.a = a
        self.b = b
        sa = set(words[a])
        sb = set(words[b])
        sab = sa.intersection(sb)
        na = float(len(sa))
        nb = float(len(sb))
        nab = float(len(sab))
        self.mim = nab / (na * nb)
        try:
            self.emim = nab * math.log(N * nab / (na * nb))
        except Exception as e:
            self.emim = 0.0
        self.x2 = (nab - (1/N) * na * nb)**2 / (na * nb)
        self.dice = nab / (na + nb)

    def getmim(self):
        return self.mim

    def getemim(self):
        return self.emim

    def getx2(self):
        return self.x2

    def getdice(self):
        return self.dice

    def __repr__(self):
        return '({},{})\n  MIM  {}\n  EMIM {}\n  X2   {}\n  Dice {}'.format(
            self.a, self.b, self.mim, self.emim, self.x2, self.dice)

def calc(choices):
    print 'calculating...'
    return {choice: [Result(choice, word) for word in words.keys() if choice != word] for choice in choices}


def gethighest(results, choice, keyfunc):
    return sorted(results[choice], key=keyfunc, reverse=True)[:10]


def printresults(results, choices):
    print 'writing tables.tex'
    with open('tables.tex', 'wb') as outfile:
        for choice in choices:
            mim  = [res.b for res in gethighest(results, choice, Result.getmim)]
            emim = [res.b for res in gethighest(results, choice, Result.getmim)]
            x2   = [res.b for res in gethighest(results, choice, Result.getx2)]
            dice = [res.b for res in gethighest(results, choice, Result.getdice)]
            printtab(outfile, choice, mim, emim, x2, dice)


head = """\\begin{table}[h!]
\centering
\\begin{tabular}{ l | c | c | c }
\hline
"""

foot = '\\hline\n\\end{tabular}\n\\caption{Calculated values for ``%s\'\'}\n\\label{tab:words}\n\\end{table}\n'

def printtab(outfile, choice, mim, emim, x2, dice):
```

```
    outfile.write(head)
    outfile.write('\\multicolumn{4}{c}{'
        + choice + '}\\\\\n\\hline\n\\textit{MIM} & \\textit{EMIM} & \\textit{\(\chi^2\)} &
            \\textit{Dice}\\\\\n\\hline\n')
    for i in range(10):
        outfile.write(row(i, mim, emim, x2, dice))
    outfile.write(foot % choice)

def row(r, mim, emim, x2, dice):
    return mim[r] + ' & ' + emim[r] + ' & ' + x2[r] + ' & ' + dice[r] + '\\\\\n'

choices = [
    'running',
    'calculation',
    'color',
    'horse',
    'sky',
    'railroad',
    'calendar',
    'airplane',
    'ocean',
    'bicycle']
results = calc(choices)
printresults(results, choices)
```

Listing 7: calc.py

7 References

[1] Team NLTK http://www.nltk.org/team.html. Natural Language Toolkit. Available at: https://www.nltk.org/. Accessed: 2016/10/11.

[2] Peter Norvig. How to Write a Spelling Corrector. Available at: http://norvig.com/spell-correct.html. Accessed: 2016/11/08.

[3] The Python Programming Language. Available at: https://www.python.org/. Accessed: 2016/09/17.

[4] Python.org. Python object serialization. Available at: https://docs.python.org/2/library/pickle.html. Accessed: 2016/11/06.

[5] NetworkX Team. NetworkX Library. Available at: http://networkx.readthedocs.io/en/networkx-1.10/. Accessed: 2016/11/11.