Assignment 3

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

November 8, 2016

Contents

1 Question 6.1 **3**
1.1 Question . . . 3
1.2 Approach . . . 3

2 Question 6.2 **4**
2.1 Question . . . 4
2.2 Approach . . . 4
2.3 Results . . . 4

3 Question 6.3 **5**
3.1 Question . . . 5
3.2 Approach . . . 5

4 Question 6.5 **6**
4.1 Question . . . 6
4.2 Answer . . . 6

5 Question MLN2 **7**
5.1 Question . . . 7
5.2 Approach . . . 7
5.3 Results . . . 7

6 Appendix **11**

7 References **15**

List of Figures

Listings

1 spelling.py example output . . . 4
2 spelling.py . . . 11
3 stem.py . . . 12
4 calc.py . . . 13

List of Tables

1 wordz . . . 7
2 wordz . . . 7
3 wordz . . . 8
4 wordz . . . 8
5 wordz . . . 8
6 wordz . . . 9
7 wordz . . . 9
8 wordz . . . 9
9 wordz . . . 10
10 wordz . . . 10

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` script, found in Listing 3, was used to solve this problem.

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

The `spelling.py` script, found in Listing 2, was created using the Python programming language [1]. Downloaded `big.txt` to calculate an example language model. These words were counted and stored in a map that was compressed on disk using the pickle python library [2].

The process of determining a spelling correction is as follows:

1. Count all the words contained in the example text file. This count is used to determine the language model $P(W)$ calculation.
2. Take the input word and determine all existing (correctly spelled) words with edit distance one and two.
3. With the assumption that shorter edit distances equate to a higher probability of being the correct spelling, select from the remaining set of (valid) words the one with the shortest edit distance and highest value for $P(W)$.

$P(W)$ 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.

2.3 Results

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

```
[mchaney@mchaney-l spelling]$ ./spelling.py werds
words
[mchaney@mchaney-l spelling]$ ./spelling.py flewerz
flower
[mchaney@mchaney-l spelling]$ ./spelling.py splling
selling
[mchaney@mchaney-l spelling]$ ./spelling.py sweling
swelling
[mchaney@mchaney-l spelling]$ ./spelling.py aacck
back
```

Listing 1: spelling.py example output

3 Question 6.3

3.1 Question

Implement a simple pseudo-relevance feedback algorithm for the Galago search engine. Provide examples of the query expansions that your algorithm does, and summarize the problems and successes of your approach.

3.2 Approach

Here's a formula.

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

4 Question 6.5

4.1 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.

4.2 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 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.

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 4, was used to complete this task.

5.3 Results

Here is the output from running the `calc.py` script:

walking			
MIM	*EMIM*	*X2*	*Dice*
Behaalotecha	Behaalotecha	Ortolan	pulled
roadrunners	roadrunners	necks	habit
Mortem	Mortem	Goshawk	covered
Hafid	Hafid	McStub	trails
Alor	Alor	Goddesses	neck
Eux	Eux	Megumi	car
Heathert	Heathert	Baobab	beyond
Jamesaustinhilll	Jamesaustinhilll	abductions	generations
FulÉŞe	FulÉŞe	Gab	bus
gopher	gopher	Chaffinch	walk

Table 1: wordz

bathroom			
MIM	*EMIM*	*X2*	*Dice*
Bookermorgan	Bookermorgan	maze	evicted
Revered	Revered	overdosing	maze
Veracities	Veracities	couches	overdosing
DurinsBane87	DurinsBane87	evicted	couches
Senor	Senor	Bleed	Bleed
Lapdog2908	Lapdog2908	Bookermorgan	owes
Vicks	Vicks	Revered	kitchens
Purvey	Purvey	Veracities	poisons
Dundeady	Dundeady	DurinsBane87	underage
GeForce	GeForce	Senor	seizes

Table 2: wordz

slam			
MIM	*EMIM*	*X2*	*Dice*
SÃűrenstam	SÃűrenstam	consecutively	consecutively
irb	irb	haka	haka
escalade	escalade	Ansbaradigeidfran	Ansbaradigeidfran
gnashing	gnashing	RBIs	RBIs
Gatland	Gatland	Koufax	Koufax
Kubek	Kubek	SÃűrenstam	Mytildebang
AER	AER	irb	McGwire
MRQE	MRQE	escalade	Slams
ArmadilloProcess	ArmadilloProcess	gnashing	characteristically
Pauldanon	Pauldanon	Gatland	Mtmelendez

Table 3: wordz

horse			
MIM	*EMIM*	*X2*	*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: wordz

sky			
MIM	*EMIM*	*X2*	*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	æŨĞ	speeds
Sennen	Sennen	Achiu31	gravity
iro	iro	Colares	Telescope
Cucumber	Cucumber	Cycles	objects

Table 5: wordz

hamstring			
MIM	*EMIM*	*X2*	*Dice*
CharlusIngus	CharlusIngus	CharlusIngus	CharlusIngus
Tibialis	Tibialis	Tibialis	Tibialis
condyle	condyle	condyle	condyle
Moyo	Moyo	Moyo	Moyo
Espino	Espino	Espino	Espino
Muscles	Muscles	Muscles	Muscles
fasciae	fasciae	fasciae	fasciae
Longus	Longus	Longus	Longus
quadratus	quadratus	quadratus	quadratus
valiantly	valiantly	valiantly	valiantly

Table 6: wordz

calendar			
MIM	*EMIM*	*X2*	*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: wordz

airplane			
MIM	*EMIM*	*X2*	*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: wordz

ocean			
MIM	*EMIM*	*X2*	*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: wordz

unpleasant			
MIM	*EMIM*	*X2*	*Dice*
Heathert	Heathert	unites	unites
humbler	humbler	Heathert	coping
reabsorption	reabsorption	humbler	compensates
reconciling	reconciling	reabsorption	Knutux
Rehabilitiation	Rehabilitiation	reconciling	Fluid
dampens	dampens	Rehabilitiation	evoke
Botodo	Botodo	dampens	anticipation
saucy	saucy	Botodo	Vary
Veracities	Veracities	saucy	remedies
Hollins	Hollins	Veracities	bodily

Table 10: wordz

6 Appendix

```
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 2: spelling.py

```
import cPickle

def cachewords():
    global words
    try:
        words = cPickle.load(open('words.p', 'rb'))
    except IOError:
        words = [line.split()[1] for line in open('wordcount.dat').readlines() if not line.
            split()[1].isdigit()]
        words = sorted(words)
        cPickle.dump(words, open('words.p', 'wb'))
        words = cPickle.load(open('words.p', 'rb'))


cachewords()
```

Listing 3: stem.py

```
import cPickle
import math


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 init():
    global words
    try:
        print 'loading cached word map'
        words = cPickle.load(open('words.p', 'rb'))
    except IOError:
        print 'cached word map not found, building now'
        words = {line.split()[0]: line.split()[1:] for line in open('invidx.dat').readlines
            ()}
        cPickle.dump(words, open('words.p', 'wb'))
        words = cPickle.load(open('words.p', 'rb'))
    global N
    N = float(sum(len(docs) for docs in words.values()))


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
\end{tabular}
\caption{wordz}
\label{tab:words}
\end{table}
"""

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{X2} & \\textit{Dice}\\\\\n\\hline\n')
    for i in range(10):
        outfile.write(row(i, mim, emim, x2, dice))
    outfile.write(foot)

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



init()
choices = [
    'walking',
    'bathroom',
    'slam',
    'horse',
    'sky',
    'hamstring',
    'calendar',
    'airplane',
    'ocean',
    'unpleasant']
results = calc(choices)
printresults(results, choices)
```

Listing 4: calc.py

7 References

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

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