

Assignment 3
Introduction to Information Retrieval
CS734/834

John Berlin

November 10, 2016

Note

As was mentioned in the this section for assignment two the size considerations for output files apply generated for this assignment. The trec files produces by *batch-search* and the indexes created by *build* functions of galago prohibit them from inclusion in this report. Also used in this report the shell script *rungalago.sh*, Source Code 7, was used to streamline the usage of galago by providing the argument boiler plate to command which required more than path arguments and to run the bin script for galago which resides in the nested directory structure of the application. The two commands which required the argument boiler plate are build and build-window both create a search index whereas the latter creates a windowed version of the first.

The version of galago used in this report is 3.10, which was rebuilt from source, whereas the version available via the link found on the textbooks [web-site](#) is version 1.04 built in 2009. Even tho galago is a Java application, it was found that this version suffered from performance issues due to its age. The biggest difference in 3.10 from 1.04 is that batch queries are in JSON format not XML and all other changes are minor feature additions and performance increases.

Q.1 Question 6.1

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.

Q.1 Answer

The first step was to create the index for the Wikipedia collection by executing the command `./runGalago.sh build index htmls` where the first argument `index` is the location to where the index is to be placed and `htmls` is the location of the input. The html files of the Wikipedia collection were moved to a single directory as `galago` took some time to traverse the original directory structure of this data set which contained 6,042 files. As was mentioned in the note section the build command required argument boilerplate and was executed with the following additional arguments supplied by the script:

`--nonStemmedPostings=true --stemmedPostings=true --stemmer=krovetz --corpus=true`. By passing the arguments `nonStemmedPostings` and `stemmedPostings` `galago` created separate index files for both index types. Indexing the collection was surprisingly quick as shown in the report generated once indexing has finished Figure 1. After the build was complete the python file `indexer.py`, Source Code 1, was used to dump the index file and extract terms.

```
Stage parsePostings completed with 0 errors.
Stage writeFields completed with 0 errors.
Stage writeNames completed with 0 errors.
Stage writeCorpusKeys completed with 0 errors.
Stage writeLengths completed with 0 errors.
Stage writeNamesRev completed with 0 errors.
Stage writePostings completed with 0 errors.
Stage writePostings-krovetz completed with 0 errors.
Done Indexing.
  - 0.03 Hours
  - 1.72 Minutes
  - 103.42 Seconds
Documents Indexed: 6042.
```

Figure 1: Indexing Report

```

/usr/bin/python3 /home/john/Documents/cs834-f16/assignments/a3/code/q1_6dot1.py
stemming index with WordNetLemmatizer
stemming index with Lancaster
stemming index with PorterStemmer
stemming index with SnowballStemmer
building queries for WordNetLemmatizer
executing queries for WordNetLemmatizer
b'Nov 08, 2016 1:52:48 PM org.lemurproject.galago.core.tools.apps.ThreadedBatchS
building queries for SnowballStemmer
executing queries for SnowballStemmer
b'Nov 08, 2016 1:53:26 PM org.lemurproject.galago.core.tools.apps.ThreadedBatchS
building queries for PorterStemmer
executing queries for PorterStemmer
b'Nov 08, 2016 1:54:59 PM org.lemurproject.galago.core.tools.apps.ThreadedBatchS
building queries for Lancaster
executing queries for Lancaster
b'Nov 08, 2016 1:56:44 PM org.lemurproject.galago.core.tools.apps.ThreadedBatchS
calculating dice for Lancaster
calculating dice for WordNetLemmatizer
calculating dice for PorterStemmer
calculating dice for SnowballStemmer
writing report for Lancaster
writing report for WordNetLemmatizer
writing report for PorterStemmer
writing report for SnowballStemmer
writing final report section

```

Figure 2: Python Output

After the index was built the python file `q1_6dot1run.py`, Source Code 2, was used to generate the results for this question. Running the file produced the output seen in Figure 2. This python script expects the directories *output_files* and *pickled* to be present in the working directory the script is executed at using `python3 q1_6dot1run.py`. The first 1000 words in the collection only began with the letter *a* which I found to lead to un-interesting results. To augment this I stemmed the entire index using four different stemmers: Porter, Snowball, WordnetLemmatizer, and Lancaster which are available via the `nlTK.stem` package. This was achieved by using the `pseq` (parallel sequence) function from the `pyfunctional` package which made the processing timely. Processing the data used the *flatmap* function in parallel to transform the was utilized to transform the list of tuples (StemmerName, Stem, Word) to single list then `foldleft` was used to accumulate the tuples into `Stemmer[Stem][words]` structure. After which the stem classes for a stemmer were written to files with name `idx_[Stemmer].txt` .

Computing Dice's coefficient and co-occurrence at document level was done by constructing queries run via galago threaded-batch-search. After the queries were written and serialized to disk after which they were executed via the python code for this question and the trec file format returned by galago were also written to disk. A total of 1,453,423 queries were created of which Lancaster had 694,985, WordNetLemmatizer had 184,843, Porter had 286,350, and Snowball had 287,245. Calculation for Dice's coefficient required a slight modification due to the combine operator being an *or operation* which was to subtract the counts for a and b from the addition of the individual counts for the word pairs. The filter threshold was set at 0.001 due to the relatively small number of documents in this collection and each stemmers results were written to files with name `Stemmer_dice.txt` and `Stemmer_dice_filtered.txt`. The final step was to generate the report for the new stem classes Listing 1.

To my surprise each stem class created each one contained the same number of words as the original stemming contained. The scores seen in Listing 1 are from the cumulative scores for all stems classes which was calculated by taking the sum of the each word pairs Dice score and dividing it by the number of pairs contained in the new stem class if the stem class had more than one pair. Also of note the stems not in the new stem classes the number of words for each was one to two.

Listing 1: Stem Class Dice Filtered Report

```

Lancaster
The new stem classes had 11127 stems in common
116536 stems not in the new stem classes
For the stems not in the new stem classes number of words
Mean: 1 Median: 1 Mode: 1 Max: 2
For all stem classes cumulative Dice score
Mean: 0.56 Median: 0.53 Mode: 0.50
-----
WordNetLemmatizer

```

The new stem classes had 5884 stems in common
 158691 stems not in the new stem classes
 For the stems not in the new stem classes number of words
 Mean: 1 Median: 1 Mode: 1 Max: 2
 For all stem classes cumulative Dice score
 Mean: 0.14 Median: 0.09 Mode: 0.50

 PorterStemmer
 The new stem classes had 10863 stems in common
 134928 stems not in the new stem classes
 For the stems not in the new stem classes number of words
 Mean: 1 Median: 1 Mode: 1 Max: 2
 For all stem classes cumulative Dice score
 Mean: 0.42 Median: 0.50 Mode: 0.50

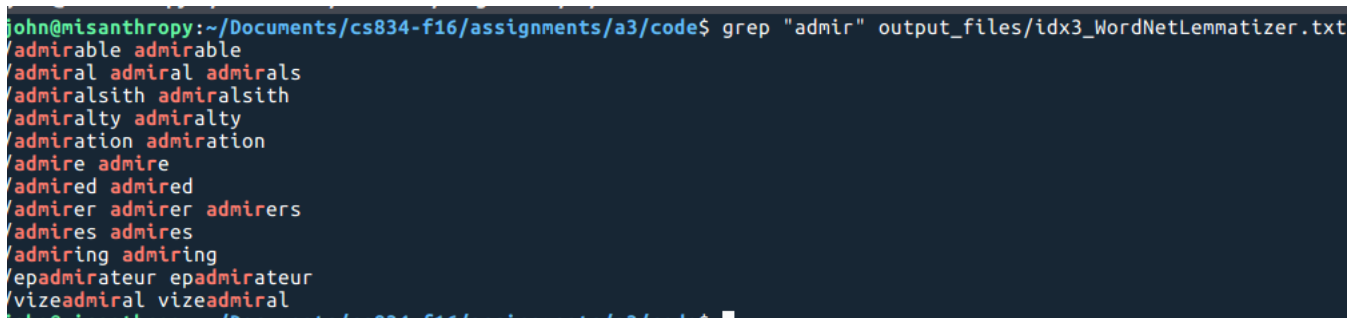
 SnowballStemmer
 The new stem classes had 10665 stems in common
 135437 stems not in the new stem classes
 For the stems not in the new stem classes number of words
 Mean: 1 Median: 1 Mode: 1 Max: 2
 For all stem classes cumulative Dice score
 Mean: 0.42 Median: 0.50 Mode: 0.50

Now to ensure this was not off I looked for the stem *admir* in all the generated classes. All but WordNetLemmatizer contained this stem as seen in Listing 2.

Listing 2: Stem Class Dice Filtered Report

```
Lancaster /admir 0.890 admirable admiral admirals admiration admire admired admirer admirers
  admires admiring
PorterStemmer /admir 0.890 admirable admiral admirals admiration admire admired admirer
  admirers admires admiring
SnowballStemmer /admir 0.890 admirable admiral admirals admiration admire admired admirer
  admirers admires admiring
```

But when grepping for that stem, Figure 3, it is clear to see WordNet does not do what one might think if not looking into the kind of stemming it does. For instance */do do dos* the lemmatization works well. But for */doje doje* and */dojo dojo* it does not as both would be expected to stem to *doj* at least. But lemmatisation¹ is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form



```
john@misanthropy:~/Documents/cs834-f16/assignments/a3/code$ grep "admir" output_files/idx3_WordNetLemmatizer.txt
/admirable admirable
/admiral admiral admirals
/admiralsith admiralsith
/admiralty admiralty
/admiration admiration
/admire admire
/admired admired
/admirer admirer admirers
/admires admires
/admiring admiring
/epadmirateur epadmirateur
/vizeadmiral vizeadmiral
```

Figure 3: Grep WordNet admir

¹<https://en.wikipedia.org/wiki/Lemmatisation>

Source Code 1: Extract Terms From Index

```
import re
import shlex
from collections import defaultdict
from subprocess import Popen, PIPE
from util import dump_pickle

isWord = re.compile('[a-zA-Z]+$')

class Idx(dict):
    def __missing__(self, key):
        res = self[key] = Term(key)
        return res

class Term(object):
    def __init__(self, term):
        self.term = term
        self.docwhere = defaultdict(list)

    def add_doc_where(self, doc, where):
        self.docwhere[doc].extend(where)

    def is_word(self):
        return isWord.match(self.term) is not None

    def __str__(self):
        return '%s %s' % (self.term, self.docwhere)

    def __repr__(self):
        return self.__str__()

def galago_postingd_csv():
    cline = './rungalago.sh dump-index index'
    with open('output_files/idx3.csv', 'w') as retOut:
        runner = Popen(shlex.split(cline), stdout=retOut, stderr=PIPE)
        print(runner.stderr.read())
    idx = Idx()
    with open('idx3.csv', 'r') as gal:
        for line in gal:
            lsplit = line.rstrip().lstrip().split(',')
            word = lsplit[0]
            doc = lsplit[1]
            at = lsplit[2:]
            idx[word].add_doc_where(doc, at)
    dump_pickle(idx, 'pickled/idx3.pickle')
    with open('output_files/idx3_terms.txt', 'w') as termOut:
        termOut.write(' '.join(sorted(filter(lambda x: isWord.match(x) is not None, list(idx.keys())))))

if __name__ == '__main__':
    galago_postingd_csv()
```

Source Code 2: Stem Cluser Generation Dice

```
import json
import shlex
import statistics
from decimal import *
from subprocess import Popen, PIPE
import re
from nltk.stem.porter import PorterStemmer
```

```

from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer
from functional import pseq
from util import read_pickle, dump_pickle
from itertools import combinations
from collections import Counter

portStemmer = PorterStemmer()
snowBall = SnowballStemmer('english')
wordNet = WordNetLemmatizer()
lancaster = LancasterStemmer()

class StemmerIdx(dict):
    def __missing__(self, key):
        ret = self[key] = StemIdx()
        return ret

class StemIdx(dict):
    def __missing__(self, key):
        ret = self[key] = Stem(key)
        return ret

    def write_class(self, out):
        for stem in sorted(self.values(), key=lambda stem: stem.stem):
            stem.write_stem(out)

class Stem(object):
    def __init__(self, stem):
        self.stem = stem
        self.stemsTo = []

    def add_term(self, term):
        self.stemsTo.append(term)

    def write_stem(self, out):
        out.write('%s %s\n' % (self.stem, ' '.join(self.stemsTo)))

    def __str__(self):
        return '%s %s' % (self.stem, self.stemsTo)

    def __repr__(self):
        return self.__str__()

class StemClazzIdx(dict):
    def __missing__(self, key):
        val = self[key] = StemClazz(key)
        return val

class StemClazz(object):
    def __init__(self, stem):
        self.stem = stem
        self.pair_scores = {}
        self.dice_score = None
        self.pairs = 0
        self.words = set()

    def add_pair_score(self, w1, w2, score):
        if self.pair_scores.get((w2, w1), None) is None:
            self.pair_scores[w1, w2] = float(score)
            self.pairs += 1
        self.words.add(w1)
        self.words.add(w2)

    def cumulative_dice(self):

```

```

    if self.dice_score is None:
        wc = len(self.words)
        if wc > 2:
            self.dice_score = sum(self.pair_scores.values()) / self.pairs
        else:
            self.dice_score = sum(self.pair_scores.values())
    return self.dice_score

def clazz(self):
    return '%s %.3f %s' % (self.stem, self.cumulative_dice(), ' '.join(sorted(self.words)))

def __str__(self):
    return '%s %d %s' % (self.stem, len(self.words), self.pair_scores)

def __repr__(self):
    return self.__str__()

def stem_map(x):
    ps = portStemmer.stem(x)
    nb = snowBall.stem(x)
    wn = wordNet.lemmatize(x)
    lan = lancaster.stem(x)
    return [('PorterStemmer', ps, x), ('SnowballStemmer', nb, x), ('WordNetLemmatizer', wn, x), ('Lancaster', lan,
↪ x)]

def fold_fun(accum, val):
    stemmer, stem, term = val
    accum[stemmer][stem].add_term(term)
    return accum

def stem_index():
    with open('output_files/idx3_terms.txt', 'r') as alpha:
        words = alpha.readline()
        stemmer_idx = pseq(words.split(' ')) \
            .flat_map(stem_map) \
            .fold_left(StemmerIdx(), fold_fun)
    dump_pickle(stemmer_idx, 'pickled/stemmerIdx.pickle')
    for stemmer, stemdic in stemmer_idx.items():
        print('stemming index with %s' % stemmer)
        with open('output_files/idx_%s.txt' % stemmer, 'w') as stemOut:
            stemdic.write_class(stemOut)

def build_stem_queries():
    stemmer_idx = read_pickle('pickled/stemmerIdx.pickle')
    for stemmer, stemdic in stemmer_idx.items():
        print('building queries for %s' % stemmer)
        queries = []
        c = 0
        for stem in stemdic.values():
            if len(stem.stemsTo) > 1:
                c += 3
                for c1, c2 in combinations(stem.stemsTo, 2):
                    if c1 is None or c2 is None:
                        raise Exception('bad field')
                    queries.append({
                        'number': '%s,%s' % (stem.stem, c1),
                        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % c1
                    })
                    queries.append({
                        'number': '%s,%s' % (stem.stem, c2),
                        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % c2
                    })
                    queries.append({
                        'number': '%s,%s,%s' % (stem.stem, c1, c2),
                        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) #dirichlet(
                            '#extents:@/%s/:part=postings() ) )' % (c1, c2)
                    })

```



```

    })
else:
    c += 1
    queries.append({
        'number': '%s,%s' % (stem.stem, stem.stemsTo[0]),
        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % stem.stemsTo[0]
    })
qloc = 'galagoqueries/%s.json' % stemmer
with open(qloc, 'w') as qout:
    json.dump({
        'queries': queries,
        'index': 'index3',
        'queryType': 'complex'
    }, qout, indent=2)
print('executing queries for %s' % stemmer)
cline = './runagalago.sh threaded-batch-search %s' % qloc
with open('output_files/%s_query_ret.trec' % stemmer, 'w') as retOut:
    runner = Popen(shlex.split(cline), stdout=retOut, stderr=PIPE)
    print(runner.stderr.read())

def write_dice():
    tol = Decimal(0.001)
    for stemmer in ['Lancaster', 'WordNetLemmatizer', 'PorterStemmer', 'SnowballStemmer']:
        count = {1: Counter(), 2: Counter()}
        print('calculating dice for %s' % stemmer)
        with open('output_files/%s_query_ret.trec' % stemmer, 'r') as trec, \
            open('output_files/%s_dice.txt' % stemmer, 'w') as out, \
            open('output_files/%s_dice_filtered.txt' % stemmer, 'w') as outf:
            for line in trec:
                it = line.rstrip().split(' ')[0]
                count[it.count(',')][it] += 1
            for stemC1C2, counts in sorted(count[2].items(), key=lambda x: x[0]):
                stem, c1, c2 = stemC1C2.split(',')
                a = count[1]['%s,%s' % (stem, c1)]
                b = count[1]['%s,%s' % (stem, c2)]
                # cause combine is or https://sourceforge.net/p/lemur/wiki/Belief%20Operations/
                ab = a + b - counts
                dice = ab / (a + b)
                out.write('%s %.3f\n' % (stemC1C2, dice))
                if Decimal(dice) >= tol:
                    outf.write('%s %.3f\n' % (stemC1C2, dice))

def write_report():
    stemmer_idx = read_pickle('pickled/stemmerIdx3.pickle')
    look_for = 'admir'
    keep = []
    with open('output_files/q1_report.txt', 'w') as stemr:
        for stemmer in ['Lancaster', 'WordNetLemmatizer', 'PorterStemmer', 'SnowballStemmer']:
            print('writing report for %s' % stemmer)
            clazIdx = StemClazzIdx()
            old_stemmer_idx = stemmer_idx[stemmer]
            with open('output_files/%s_dice_filtered.txt' % stemmer, 'r') as sin:
                for line in sin:
                    stemw1w2, score = line.rstrip().split(' ')
                    stem, w1, w2 = stemw1w2.split(',')
                    clazIdx[stem].add_pair_score(w1, w2, score)
            oks = set(old_stemmer_idx.keys())
            nks = set(clazIdx.keys())
            diff = oks.difference(nks)
            old_lens = []
            for old in diff:
                old_lens.append(len(old_stemmer_idx[old].stemsTo))
            with open('output_files/%s_new_stem_class.txt' % stemmer, 'w') as classOut:
                stemr.write('%s\n' % stemmer)
                stemr.write('The new stem classes had %d stems in common\n' % len(oks.intersection(nks)))
                stemr.write('%d stems not in the new stem classes\n' % len(diff))
                stemr.write('For the stems not in the new stem classes number of words\n')
                stemr.write('Mean: %d Median: %d Mode: %d Max: %d\n' % (

```

```

        statistics.mean(old_lens), statistics.median(old_lens), statistics.mode(old_lens),
        ↪ max(old_lens)))
coverage = []
for sclazz in sorted(clazzIdx.values(), key=lambda clazz: clazz.stem):
    classOut.write('/%s,%.3f,%s\n' % (sclazz.stem, sclazz.cumulative_dice(), '
    ↪ '.join(sclazz.words)))
    if look_for == sclazz.stem:
        keep.append((stemmer, sclazz))
    coverage.append(sclazz.cumulative_dice())
    old_stem = old_stemmer_idx[sclazz.stem]
    oldwords = set(old_stem.stemsTo)
    ondif = oldwords.difference(sclazz.words)
    if len(ondif) > 0:
        stemr.write('the original stem class %s had %d more words in it' % (sclazz.stem,
        ↪ len(ondif)))
        stemr.write('the new stem class has a cumulative dice score of %d' %
        ↪ sclazz.cumulative_dice())
    stemr.write('For all stem classes cumulative Dice score\n')
    stemr.write('Mean: %.2f Median: %.2f Mode: %.2f\n' % (
        statistics.mean(coverage), statistics.median(coverage), statistics.mode(coverage)))
    stemr.write('-----\n')
print('writing final report section')
with open('same.txt', 'w') as rout:
    for s, so in keep:
        rout.write('%s %s\n' % (s, so.clazz()))

if __name__ == '__main__':
    stem_index()
    build_stem_queries()
    write_dice()
    write_report()

```

Q.2 MLN1

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

Q.2 Answer

This question builds heavily on the work done in question one and uses the work done by section Q.1 up to the generation of the stem classes which is where I will focus on in answering this question except that the queries used `#uw:50` as the operator for the co-occurrence query. The code for this question can be seen in Source Code 3. Like section Q.1 all stemmers used in it were considered when creating the graphs. The graphs were creating using networkx library and the edges were stem to word and word to stem to ensure the connection. Both connected components and the strong connected components directed graph counterpart were generated. In order to make presenting the results of this work I combined the connected and strongly connected data points into a single graph per stemmer. The code used to produce the graphs `graphStemClass.py` can bee seen in Source Code 3 and the R code `stemClassClusterReport.R` can bee seen in Source Code 4.

The colors of each graph represent the percentage of how well these new stem class fair in comparison to the old. For instance when the label states 100% fully covered it means that the new stem class contained the original stem i.e /demi and all the words covered by the original likewise 100% only stem means the new stem class only contained the stem not any of the words contained in the old. All other percentages represent the fractional amount of the previous two measures. The different graph types are encoded in the shape attribute with the y axis representing how many times a stem class occurred for the x axis which is the size of the class i.e how many words are contained in it.

As seen in Figure 4 lancaster, Figure 5 Porter, and Figure 6 Snowball the new stem classes contained more words than the original and the graphs generated take the shape of a power law distribution. The Porter and Snowball stemmer had the least number of new stem classes created and generally had less of the original stem classes words. As stated in the answer to section Q.1 the WordNetLemmatizer, Figure 7, performed the worst out of the four.

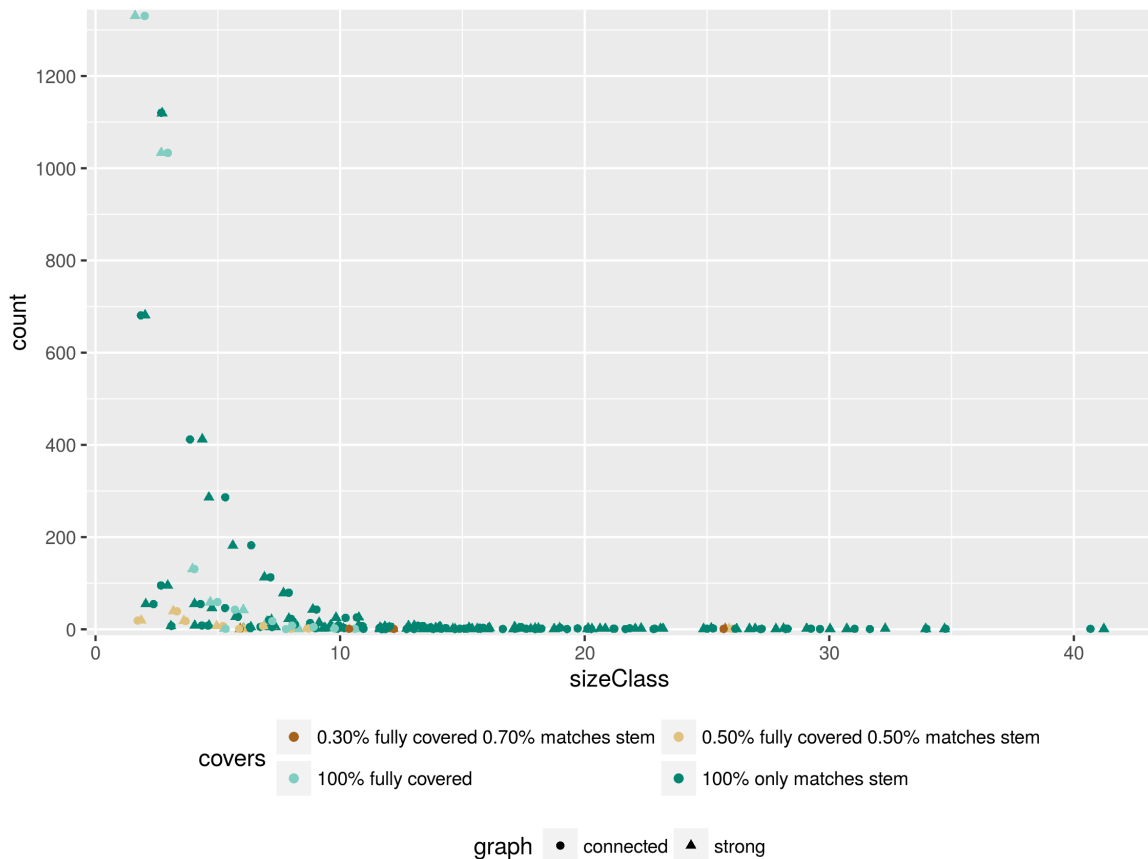


Figure 4: Lancaster Connected

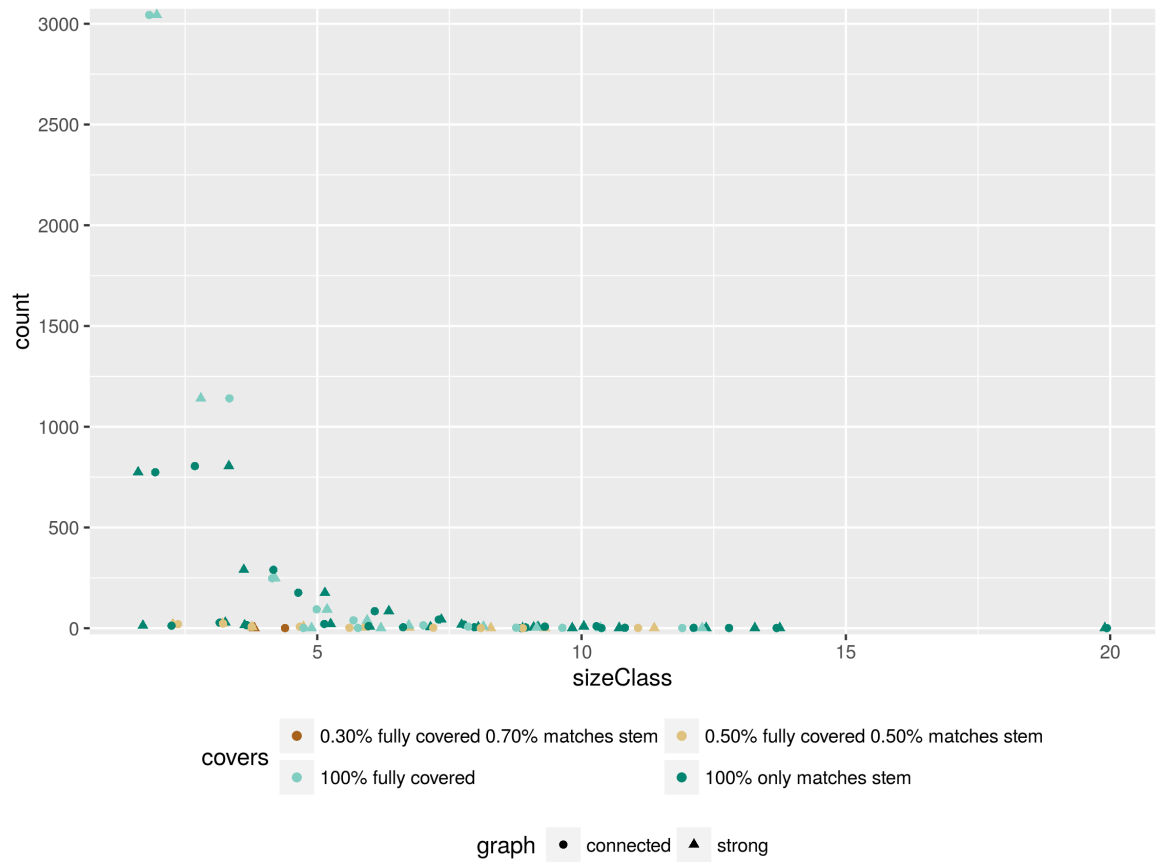


Figure 5: Porter Connected

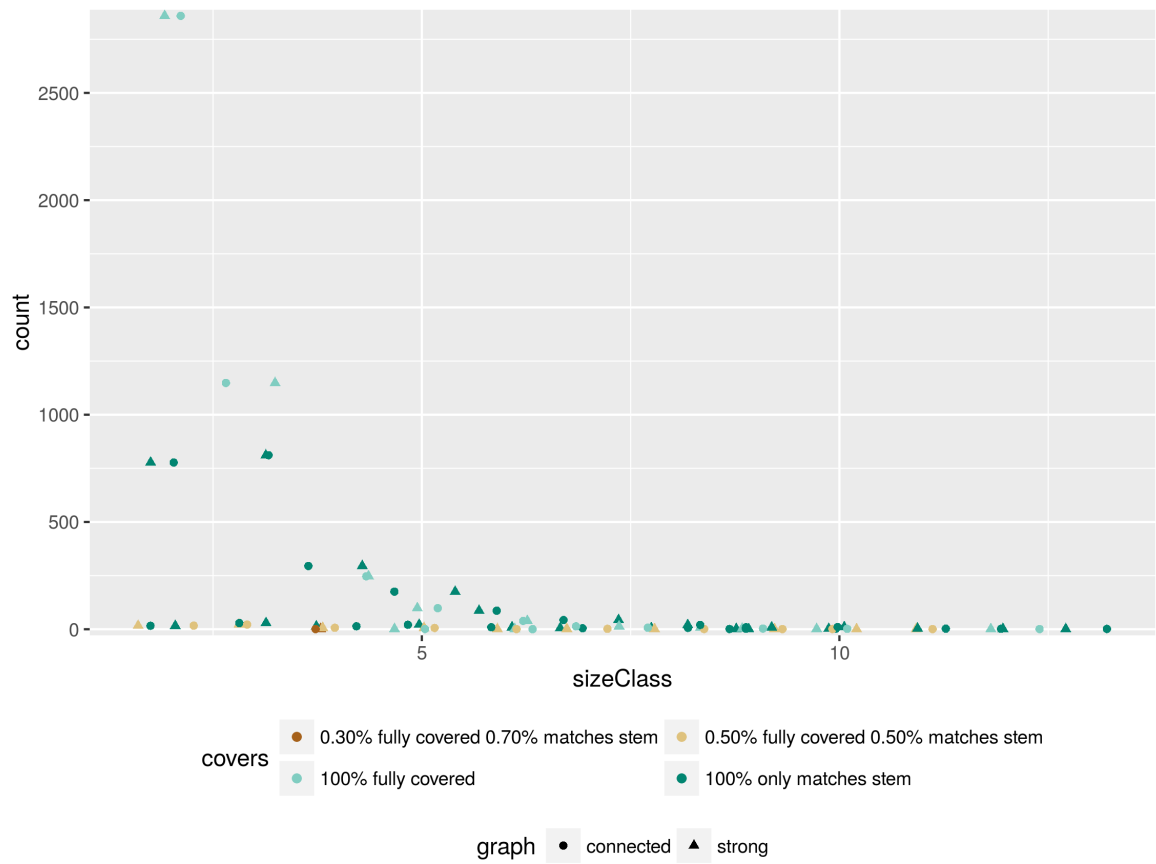


Figure 6: Snowball Connected

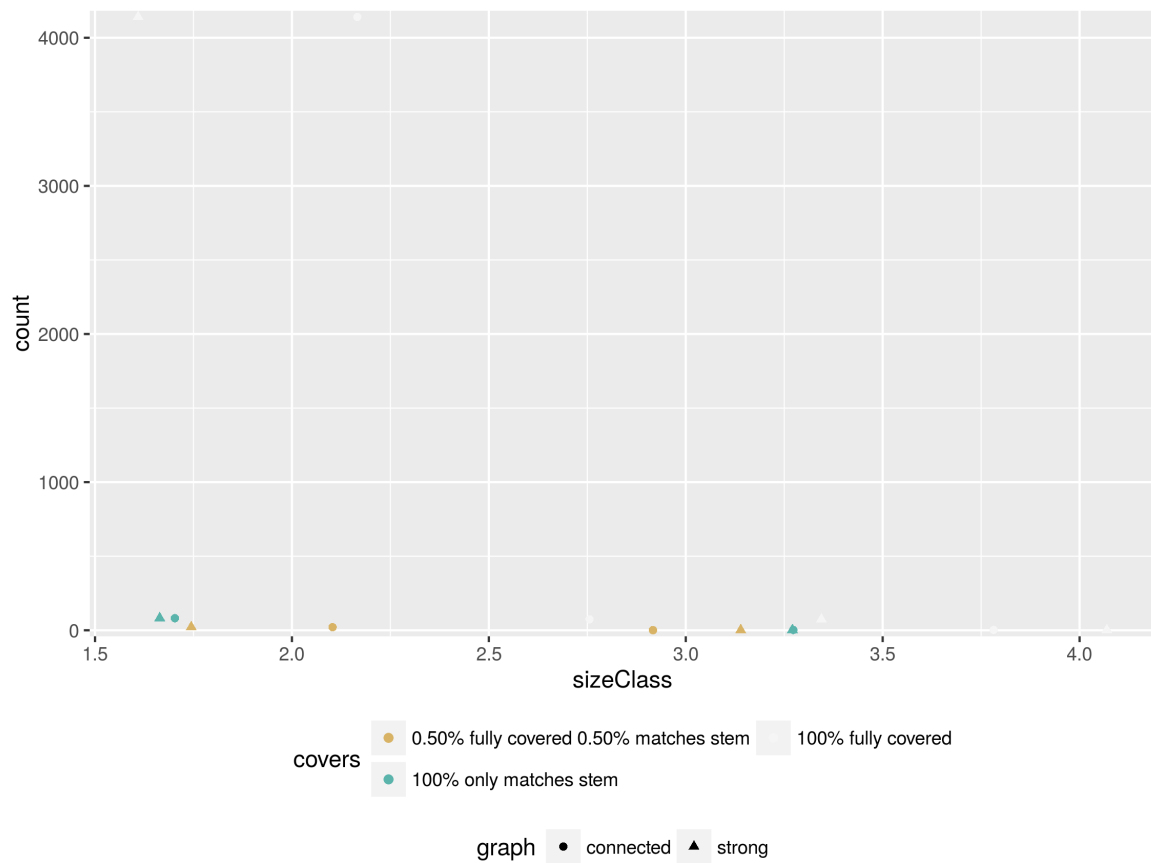


Figure 7: WordNet Lemmatizer Connected

Source Code 3: Stem Class From Graph Using Ordered Window

```
import json
import re
import shlex
from collections import Counter
from collections import defaultdict
from decimal import *
from itertools import combinations
from subprocess import Popen, PIPE
import networkx as nx
from networkx import Graph, DiGraph
from util import read_pickle

only_words = re.compile('[a-zA-Z]+[a-zA-Z]+$')

class StemmerIdx(dict):
    def __missing__(self, key):
        ret = self[key] = StemIdx()
        return ret

class StemIdx(dict):
    def __missing__(self, key):
        ret = self[key] = Stem(key)
        return ret

    def write_class(self, out):
        for stem in sorted(self.values(), key=lambda stem: stem.stem):
            stem.write_stem(out)

class Stem(object):
    def __init__(self, stem):
```

```

        self.stem = stem
        self.stemsTo = []

def add_term(self, term):
    self.stemsTo.append(term)

def write_stem(self, out):
    out.write('%s %s\n' % (self.stem, ' '.join(self.stemsTo)))

def __str__(self):
    return '%s %s' % (self.stem, self.stemsTo)

def __repr__(self):
    return self.__str__()

class StemClazz(object):
    def __init__(self, line):
        self.clazz = line.split(' ')
        self.matchesOld = 0
        self.matchesStem = []
        self.coverage = ''
        self.cboth = 0
        self.ostem = 0

    def search(self, alphaList):
        for alphaStem in alphaList:
            if alphaStem.stem in self.clazz:
                self.matchesOld += 1
                matchCount = 0
                for term in alphaStem.stemsTo:
                    if term in self.clazz:
                        matchCount += 1
                if matchCount == len(alphaStem.stemsTo):
                    self.matchesStem.append((alphaStem.stem, 'contains stem & terms'))
                else:
                    self.matchesStem.append((alphaStem.stem, 'matches only stem'))

    def cal_coverage(self):
        for stem, howMuch in self.matchesStem:
            if howMuch == 'contains stem & terms':
                self.cboth += 1
            else:
                self.ostem += 1
        if self.matchesOld == 1:
            if self.cboth > 0:
                self.coverage = '100% fully covered'
            else:
                self.coverage = '100% only matches stem'
        else:
            if self.cboth > 0 and self.ostem > 0:
                bothp = self.cboth / self.matchesOld
                ostemp = self.ostem / self.matchesOld
                self.coverage = '%.2f' % round(bothp, 1) + '% fully covered' + ' %.2f' % round(ostemp,
                                                                                               1) + '% matches
                                                                                               ↪ stem'

            elif self.ostem == 0:
                self.coverage = '100% fully covered'
            else:
                self.coverage = '100% only matches stem'

    def __str__(self):
        return '%s %d %s %s' % (self.clazz, self.matchesOld, self.coverage, self.matchesStem,)

    def __repr__(self):
        return self.__str__()

class CountDict(dict):
    def __missing__(self, key):
        ret = self[key] = CoverageDict()

```

```

return ret

class CoverageDict(dict):
    def __missing__(self, key):
        ret = self[key] = Counter()
        return ret

def build_stem_queries():
    stemmer_idx = read_pickle('pickled/stemmerIdx3.pickle')
    for stemmer, stemdic in stemmer_idx.items():
        print('building queries for %s' % stemmer)
        queries = []
        c = 0
        for stem in stemdic.values():
            if len(stem.stemsTo) > 1:
                c += 3
                for c1, c2 in combinations(stem.stemsTo, 2):
                    if c1 is None or c2 is None:
                        raise Exception('bad field')
                    queries.append({
                        'number': '%s,%s' % (stem.stem, c1),
                        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % c1
                    })
                    queries.append({
                        'number': '%s,%s' % (stem.stem, c2),
                        'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % c2
                    })
                    # #max( #extents:@/replicate/:part=postings() #extents:@/replicating/:part=postings())
                    queries.append({
                        'number': '%s,%s,%s' % (stem.stem, c1, c2),
                        'text': '#combine(#uw:50( #extents:@/%s/:part=postings() #extents:@/%s/:part=postings()))'
                               ↪ % (
                                   c1, c2)
                    })
            else:
                c += 1
                queries.append({
                    'number': '%s,%s' % (stem.stem, stem.stemsTo[0]),
                    'text': '#combine( #dirichlet( #extents:@/%s/:part=postings() ) )' % stem.stemsTo[0]
                })
        qloc = 'galagoqueries/window50/%s.json' % stemmer
        with open(qloc, 'w') as qout:
            json.dump({
                'queries': queries,
                'index': 'index3',
                'queryType': 'complex'
            }, qout, indent=2)
        print('executing queries for %s' % stemmer)
        cline = './rungalago.sh threaded-batch-search %s' % qloc
        with open('output_files/window50/%s_query_ret.trec' % stemmer, 'w') as retOut:
            runner = Popen(shlex.split(cline), stdout=retOut, stderr=PIPE)
            print(runner.stderr.read())

def write_dice():
    tol = Decimal(0.001)
    for stemmer in ['Lancaster', 'WordNetLemmatizer', 'PorterStemmer', 'SnowballStemmer']:
        count = {1: Counter(), 2: Counter()}
        print('calculating dice for %s' % stemmer)
        with open('output_files/window50/%s_query_ret.trec' % stemmer, 'r') as trec, \
            open('output_files/window50/%s_dice.txt' % stemmer, 'w') as out, \
            open('output_files/window50/%s_dice_filtered.txt' % stemmer, 'w') as outf:
            for line in trec:
                it = line.rstrip().split(' ')[0]
                count[it.count(',')] [it] += 1
            for stemC1C2, counts in sorted(count[2].items(), key=lambda x: x[0]):
                stem, c1, c2 = stemC1C2.split(',')
                a = count[1]['%s,%s' % (stem, c1)]

```

```

        b = count[1]['%s,%s' % (stem, c2)]
        # cause combine is or https://sourceforge.net/p/lemur/wiki/Belief%20Operations/
        ab = (a + b) - counts
        dice = ab / (a + b)
        out.write('%s %.3f\n' % (stemC1C2, dice))
        if Decimal(dice) >= tol:
            outf.write('%s %.3f\n' % (stemC1C2, dice))

def find_connected():
    for stemmer in ['Lancaster', 'WordNetLemmatizer', 'PorterStemmer', 'SnowballStemmer']:
        with open('output_files/window50/%s_dice_filtered.txt' % stemmer, 'r') as sin, \
            open('output_files/window50/%s_dice_connected.txt' % stemmer, 'w') as out, \
            open('output_files/window50/%s_dice_sconnected.txt' % stemmer, 'w') as out2:
            print('building clusters for %s' % stemmer)
            g = DiGraph()
            g2 = Graph()
            nodes = set()
            edges = set()
            for line in sin:
                stemC1C2, dice = line.rstrip().split(' ')
                stem, c1, c2 = stemC1C2.split(',')
                nodes.add(stem)
                nodes.add(c1)
                nodes.add(c2)
                edges.add((stem, c1, float(dice)))
                edges.add((stem, c2, float(dice)))
                edges.add((c1, stem, float(dice)))
                edges.add((c2, stem, float(dice)))
            for n in nodes:
                g.add_node(n)
                g2.add_node(n)
            for stem, term, dice in edges:
                g.add_edge(stem, term, weight=dice)
                g2.add_edge(stem, term, weight=dice)
            for connected in sorted(nx.connected_components(g2), key=lambda s: list(s)[0]):
                out.write('%s\n' % ' '.join(connected))
            for connected in sorted(nx.strongly_connected_components(g), key=lambda s: list(s)[0]):
                out2.write('%s\n' % ' '.join(connected))

def coverageReport():
    stemmer_idx = read_pickle('pickled/stemmerIdx3.pickle')
    for stemmer in ['Lancaster', 'WordNetLemmatizer', 'PorterStemmer', 'SnowballStemmer']:
        with open('output_files/window50/%s_dice_connected.txt' % stemmer, 'r') as sin, \
            open('output_files/window50/%s_dice_sconnected.txt' % stemmer, 'r') as sin2:
            print('generating coverage report for %s' % stemmer)
            stemdic = stemmer_idx[stemmer]
            writeCoverage(stemmer, stemdic, 'connected', sin)
            writeCoverage(stemmer, stemdic, 'sconnected', sin2)

def writeCoverage(stemmer, stemdic, clust, sin):
    alphaStem = defaultdict(list)
    for stem in stemdic.values():
        alphaStem[stem.stem[0]].append(stem)
    alphaStemClazz = defaultdict(list)
    for line in sin:
        line = line.rstrip()
        alphaStemClazz[line[0]].append(StemClazz(line))
    for alpha, stemclzss in alphaStemClazz.items():
        for sclzz in stemclzss:
            sclzz.search(alphaStem[alpha])
    alphaClazzCover = defaultdict(CoverageDict)
    for alpha, stemclzss in alphaStemClazz.items():
        for sclzz in stemclzss:
            sclzz.cal_coverage()
            alphaClazzCover[len(sclzz.clazz)][sclzz.matchesOld][sclzz.coverage] += 1
    with open('output_files/window50/%s_stemclass_%s.csv' % (stemmer, clust), 'w') as covOut:
        covOut.write('sizeClass,matches,covers,count\n')

```



```

    for sizeClazz, mathnum in sorted(alphaClazzCover.items(), key=lambda x: x[0], reverse=True):
        for matches, coverages in sorted(mathnum.items(), key=lambda x: x[0], reverse=True):
            for covers, count in sorted(coverages.items(), key=lambda x: x[1], reverse=True):
                covOut.write('%d,%d,%s,%d\n' % (sizeClazz, matches, covers, count))

if __name__ == '__main__':
    build_stem_queries()
    write_dice()
    find_connected()
    coverageReport()

```

Source Code 4: Stem Class Cluster Graphs

```

library(scales)
library(ggplot2)
library(RColorBrewer)

names <- c('Lancaster', 'PorterStemmer', 'SnowballStemmer', 'WordNetLemmatizer')
for(name in names) {
    lc <- read.csv(paste('output_files/window50/', name, '_stemclass_connected.csv', sep = ''))
    lsc <- read.csv(paste('output_files/window50/', name, '_stemclass_sconnected.csv', sep = ''))
    lsc$graph <- 'strong'
    lc$graph <- 'connected'
    ggplot(lsc, aes(sizeClass, count, shape=graph)) +
        geom_jitter(aes(color=covers, shape=graph, height = .005)) +
        geom_jitter(data=lc, aes(sizeClass, count, color=covers, height = .005)) +
        scale_y_continuous(breaks = pretty_breaks(n=5, min.n=4), expand=c(0.01, 0.25)) +
        scale_color_brewer(palette = 'BrBG') + theme(legend.position="bottom") + guides(col = guide_legend(ncol = 2,
        ↪ byrow = TRUE))
    ggsave(paste(name, 'connected.png', sep = ''))
}

```

Q.3 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)

Q.3 Answer

In order to complete this question a 5 word window was created using `./runGalago.sh build-window 5 false index3` where `false` was to create unordered window. The python file `associations.py` Source Code 5 was used to generate the results which were over the all the words contained in the wikipedia collection. Table 1 shows the results after I hand picked ten most interesting ones of which 5 were looked for using `grep`. The first word `rascal` has high dice and `mi` with `dizze` but the context in which this happens is unknown. `Sportscar` is highly associated with `boxster` along `Ozzy Osbourne` and `Lynyrd Skynyrd` both of which I was unsure would be in this collection. `Coder well robocoder` has an association with `lesfer` this association I am completely unsure off. The others were the most recognizable pairs when glancing in the file. I am for sure that the χ^2 measure was completely miscalculated along with `emi`. I can only attribute this to human error or the number of occurrences to number of words was so small it threw off the calculations.

Table 1: 10 Word Associations

w1	w2	dice	mi	emi	chi2
dizze	rascal	1.0	1.0	0.0000017097	88184514016920.98
sportscar	boxster	1.00000	1.00000	0.0000017097	88184514016920.98
ozzy	osbourne	1.00000	0.50000	0.0000032718	88184514016920.98
lynnyrd	skynyrd	1.00000	0.50000	0.0000032718	88184514016920.98
jeepers	creepers	1.00000	1.00	0.0000017097	88184514016920.98
robocoder	lesfer	0.50000	0.125	0.0000029765	22046128504230.24
spiffy	sperry	0.42857	0.0375	0.0000081603	19841515653807.22
pyrokinetic	thermokinetic	0.40	0.16667	0.0000015189	14697419002820.16
friends	video	0.00068	0.00	0.0000002062	49448444.52
career	research	0.00068	0.00	0.0000000783	45144302.65

Source Code 5: Association Measures

```
import shlex
import re
import math
from subprocess import Popen, PIPE
from util import dump_pickle, read_pickle
from collections import Counter

only_words = re.compile('[a-zA-Z]+[a-zA-Z]+$')

def dice(a, b, ab_wins, ab_count):
    return (2 * ab_wins[a, b]) / (ab_count[a] + ab_count[b])

def mi(a, b, ab_wins, ab_count, nwin):
    return ab_wins[a, b] / (ab_count[a] * ab_count[b])

def emi(a, b, ab_wins, ab_count, nwin):
    return (ab_wins[a, b] / nwin) * math.log(nwin * (ab_wins[a, b] / (ab_count[a] * ab_count[b])))

def chi(a, b, ab_wins, ab_count, nwin):
    top = (ab_wins[a, b] - ((1 / nwin) * ab_count[a] / nwin * ab_count[b] / nwin))
    return (math.pow(top, 2)) / ((ab_count[a] / nwin) * (ab_count[b] / nwin))
```

```

class Win5Ret(object):
    def __init__(self, a, b, ab_wins, ab_count, nwin):
        self.a = a
        self.b = b
        self.dice = dice(a, b, ab_wins, ab_count)
        self.mi = mi(a, b, ab_wins, ab_count, nwin)
        self.emi = emi(a, b, ab_wins, ab_count, nwin)
        self.chi = chi(a, b, ab_wins, ab_count, nwin)

    def __str__(self):
        return '%s, %s dice: %.5f mi: %.5f emi: %.5f chi %.5f' % (
            self.a, self.b, self.dice, self.mi, self.emi, self.chi)

    def __repr__(self):
        return self.__str__()

    def write_csv(self, out):
        out.write('%s, %s, %.5f, %.5f, %.5f, %.5f\n' % (
            self.a, self.b, self.dice, self.mi, self.emi, self.chi))

def dump_window_idx():
    cline = './rungalago.sh dump-index windowIdx/od.n2.w5.h2'
    with open('output_files/window5/ordered5idx.txt', 'w') as retOut:
        runner = Popen(shlex.split(cline), stdout=retOut, stderr=PIPE)
        print(runner.stderr.read())
    ab_count = Counter()
    ab_count_wins = Counter()
    wins = 0
    with open('output_files/window5/ordered5idx.txt', 'r') as oin:
        for line in oin:
            splitted = line.rstrip().split(',')
            if only_words.match(splitted[0]) is not None:
                a, b = splitted[0].split('~')
                ab_count[a] += 1
                ab_count[b] += 1
                wins += 1
                ab_count_wins[a, b] += 1

    dump_pickle((ab_count, ab_count_wins, wins), 'pickled/window5Counts.pickle')

def association_measures():
    ab_count, ab_count_wins, wins = read_pickle('pickled/window5Counts.pickle')
    rets = []
    for (a, b) in ab_count_wins.keys():
        rets.append(Win5Ret(a, b, ab_count_wins, ab_count, wins))
    with open('output_files/window5/ordered5associationret.csv', 'w') as out:
        for ret in sorted(rets, key=lambda w5r: w5r.dice, reverse=True):
            ret.write_csv(out)

if __name__ == '__main__':
    dump_window_idx()
    association_measures()

```

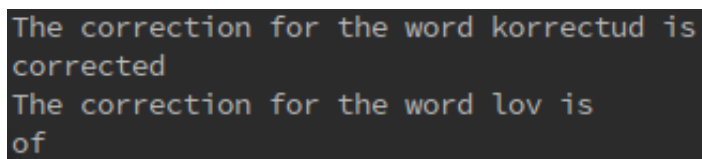
Q.4 Question 6.2

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).

Q.4 Answer

The correct spelling of words was provided using the brown, gutenbergreuters, inaugural and words corpus's available from the nltk library giving a total of 285,673 words. Calculating the edit distance was done using the Damerau Levenshtein distance (my third implementation, the first two were native C++ implementations for [Java](#)). I must confess I am not sure I meet the error model requirement but make up for that in the search space. Finding the correction is done by getting the edit distance for 285,673 words to the misspelled word and then filtering the weights to keep only those who are within a distance of two all in parallel. The probability calculation is the `within2Word / sum(wordCounts)`. The correction is the maximum probability of all the words remaining after filtering, if no words are returned 'No Spellz Korrektion Found' is returned ;)

The code for this noisy.py Source Code 6. Running the code `python3 noisy.py --wordz korrektud lov` produced the following output Figure 8.



```
The correction for the word korrektud is
corrected
The correction for the word lov is
of
```

Figure 8: Noisy Spell Corrector Run

Source Code 6: Noisy Spell Corrector

```
import re
import os.path
import argparse
from nltk.corpus import brown, gutenbergreuters, inaugural, words
from collections import Counter
from functional import pseq as parseq
from util import read_pickle, dump_pickle

only_words = re.compile('[a-z]+')

def build_word_count():
    if os.path.isfile('pickled/wcount.pickle'):
        return read_pickle('pickled/wcount.pickle')
    wcount = Counter()
    for fid in words.fileids():
        for word in words.words(fid):
            word = word.lower()
            if only_words.match(word) is not None:
                wcount[word] += 1
    for fid in gutenbergreuters.fileids():
        for word in gutenbergreuters.words(fid):
            word = word.lower()
            if only_words.match(word) is not None:
                wcount[word] += 1
    for fid in brown.fileids():
        for word in brown.words(fid):
            word = word.lower()
            if only_words.match(word) is not None:
                wcount[word] += 1
    for fid in reuters.fileids():
        for word in reuters.words(fid):
```

```

        word = word.lower()
        if only_words.match(word) is not None:
            wcount[word] += 1
    for fid in inaugural.fileids():
        for word in inaugural.words(fid):
            word = word.lower()
            if only_words.match(word) is not None:
                wcount[word] += 1
    dump_pickle(wcount, 'pickled/wcount.pickle')
    return wcount

def damerau_levenshtein(a, b):
    table = {}
    alphabet = {}
    for ac in a:
        alphabet[ac] = 0
    for bc in b:
        alphabet[bc] = 0
    lena = len(a)
    lenb = len(b)
    for i in range(lena + 1):
        table[i, 0] = i
    for j in range(lenb + 1):
        table[0, j] = j
    for i in range(1, lena + 1):
        db = 0
        for j in range(1, lenb + 1):
            k = alphabet[b[j - 1]]
            l = db
            cost = 0
            if a[i - 1] == b[j - 1]:
                db = j
            else:
                cost = 1
            table[i, j] = min(table[i - 1, j - 1] + cost, # substitution
                             table[i, j - 1] + 1, # insertion
                             table[i - 1, j] + 1) # deletion
            if k > 0 and l > 0:
                table[i, j] = min(table[i, j], table[k - 1, l - 1] + (i - k - 1) + 1 + (j - l - 1)) #
                ↪ transposition
            alphabet[a[i - 1]] = i
    return table[lena, lenb]

def dist_away(mispell, wc, dist=2):
    return parseq(wc.keys()) \
        .map(lambda w: (w, damerau_levenshtein(mispell, w))) \
        .filter(lambda wdist: wdist[1] <= dist) \
        .map(lambda wdist: wdist[0]).to_set()

def probability(word, wc):
    return wc[word] / sum(wc.values())

def correct(mispell, wc):
    try:
        return max(dist_away(mispell, wc), key=lambda w: probability(w, wc))
    except Exception:
        return 'No Spellz Korrektion Found'

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Korreckt Yo Spellz')
    parser.add_argument('--wordz', type=str, nargs='+',
                        help='wordz to make spellz good', required=True)
    args = parser.parse_args()
    wc = build_word_count()
    for badd in args.wordz:

```

```
print('The correction for the word',badd,'is')
print(correct(badd, wc))
```

Q.5 Not Attempted

Source Code 7: Run Galago

```
#!/usr/bin/env bash

if [ $# -eq 0 ]; then
    ./galago-3.10/contrib/target/appassembler/bin/galago --help
elif [ "$1" = "help" ]; then
    ./galago-3.10/contrib/target/appassembler/bin/galago --help
elif [ "$1" = "build" ]; then
    shift
    ./galago-3.10/contrib/target/appassembler/bin/galago build --nonStemmedPosting=true --stemmedPostings=true \
    --stemmer+krovetz --corpus=true --indexPath=$1 --inputPath+$2
elif [ "$1" = "dump-idx" ]; then
    shift
    ./galago-3.10/contrib/target/appassembler/bin/galago dump-index $@
elif [ "$1" = "search" ]; then
    shift
    ./galago-3.10/contrib/target/appassembler/bin/galago search --port=9000 $@
elif [ "$1" = "build-window" ]; then
    shift
    ./galago-3.10/contrib/target/appassembler/bin/galago build-window --width=$1 --ordered=$2 \
    --stemming=false --spaceEfficient=true --inputPath=$3 --indexPath=$4
else
    ./galago-3.10/contrib/target/appassembler/bin/galago $@
fi
```
