INTRO. TO INFO RETRIEVAL: CS 734: A3

Due on Thursday, November 9, 2017

Dr. Nelson

Udochukwu Nweke

Contents

Problem 1

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)

Solution 1:

Listing 1: Association Measure for 10 Words and 5 Word Windows

```python
import math
from P2 import getHTMLPaths
from datetime import datetime

from common import getTextFromHTML
from common import readTextFromFile
from common import dumpJsonToFile
from common import getDictFromFile
from common import writeTextToFile
from common import genericErrorInfo
from common import getTokenizer

from sklearn.feature_extraction.text import CountVectorizer
import re
import sys

def getKwordWindows(text, k):

    tokens = []
    try:
        tokens = getTokenizer(text)
    except:
        genericErrorInfo()

    kWordWindows = []
    for i in range(len(tokens)):

        if( i % k == 0 ):
            kWordWindows.append([])

        kWordWindows[-1].append( tokens[i] )


    return kWordWindows

def getKwordWindowsOpt(text, k):

    tokens = []
    try:
        tokens = getTokenizer(text)
    except:
        genericErrorInfo()

    kWordWindows = []
```

Problem 1 continued on next page. . .

```
    for i in range(len(tokens)):

        if( i % k == 0 ):
            kWordWindows.append([])

        kWordWindows[-1].append( tokens[i] )


    return kWordWindows


def transformDocToWindow(vocabDict, vocab):

    if( vocab not in vocabDict ):
        print('term:', vocab, 'not in vocab')
        return

    allWindows = []
    for i in range(len(vocabDict[vocab]['f'])):
        f = vocabDict[vocab]['f'][i] + '.txt'
        f = readTextFromFile(f)

        allWindows += getKwordWindows(f, 5)

    vocabDict[vocab]['f'] = allWindows

def transformDocToWindowOpt(vocabDict, vocab):

    if( vocab not in vocabDict ):
        print('term:', vocab, 'not in vocab')
        return

    allWindows = {'tot': 0, 'windows': []}
    for i in range(len(vocabDict[vocab]['f'])):
        f = vocabDict[vocab]['f'][i] + '.txt'
        f = readTextFromFile(f)

        allWindows['windows'] += getKwordWindowsOpt(f, 5)


    allWindows['tot'] = len(allWindows['windows'])
    windowsWithVocab = []

    for win in allWindows['windows']:
        if(vocab in win):
            windowsWithVocab.append(win)

    allWindows['windows'] = windowsWithVocab
    vocabDict[vocab]['f'] = allWindows



def countTerms(windows, left, right):
```

Problem 1 continued on next page. . .

```
    count = {'left': 0, 'both': 0}

    for window in windows:

        if(left in window):
            count['left'] += 1

            if(right in window):
                count['both'] += 1

    return count



def getAssocMeasuresWindow(a, N, filename, k=10):

    prev = datetime.now()

    vocabDict = getDictFromFile(filename)
    a = a.lower()

    if( a not in vocabDict ):
        print('term:', a, 'not in vocab')
        return

    transformDocToWindowOpt(vocabDict, a)
    totalVocab = len(vocabDict)
    pos = 0

    vocabDict[a]['MIM'] = -1
    vocabDict[a]['EMIM'] = -1
    vocabDict[a]['CHI-SQUARE'] = -1
    vocabDict[a]['DICE'] = -1

    for b, bDict in vocabDict.items():

        pos += 1

        if( b == a ):
            continue


        count = countTerms( vocabDict[a]['f']['windows'], a, b )
        Na = count['left']
        Nab = count['both']

        transformDocToWindowOpt(vocabDict, b)
        count = countTerms( vocabDict[b]['f']['windows'], b, a )
        Nb = count['left']

        MIM = -1
        EMIM = -1
```

```
            dice = -1
            chiSquare = -1

            if( pos % 100 == 0 ):
                print(pos, 'of', totalVocab)
                print('\tNa:', Na, a)
                print('\tNb:', Nb, b)
                print('\tNab:', Nab)
                delta = datetime.now() - prev
                print('\ttotal seconds:', delta.seconds)

            NaTimesNb = Na * Nb

            if( Nab != 0 ):
                MIM = Nab / (Na * Nb)
                dice = Nab / (Na + Nb)
                EMIM = Nab * math.log(N * MIM, 10)

            if( NaTimesNb != 0 ):
                numer = Nab - (NaTimesNb/N)
                chiSquare = (numer * numer) / NaTimesNb


            bDict['MIM'] = MIM
            bDict['EMIM'] = EMIM
            bDict['CHI-SQUARE'] = chiSquare
            bDict['DICE'] = dice

        for sortCriteria in ['MIM', 'EMIM', 'CHI-SQUARE', 'DICE']:

            print()

            sort = sorted( vocabDict.items(), key=lambda x: x[1][sortCriteria],
            reverse=True)
            sort = sort[:k]

            print(a, 'vs')
            for termDict in sort:
                term, termDict = termDict
                print('\tterm:', term, sortCriteria + ':', termDict[sortCriteria])

def getAssocMeasuresDocs(a, N, k = 10):

    vocabDict = getDictFromFile('wiki-small-vocab.json')
    a = a.lower()

    if( a not in vocabDict ):
        print('term:', a, 'not in vocab')
        return

    aFileSet = set(vocabDict[a]['f'])
    vocabDict[a]['MIM'] = -1
    vocabDict[a]['EMIM'] = -1
    vocabDict[a]['CHI-SQUARE'] = -1
    vocabDict[a]['DICE'] = -1

    Na = len(aFileSet)

    for b, bDict in vocabDict.items():

            if( b == a ):
                continue

            bFileSet = set(bDict['f'])
            Nb = len(bFileSet)
            intersect = aFileSet & bFileSet

            MIM = -1
            EMIM = -1
            dice = -1
            chiSquare = -1

            Nab = len(intersect)
            NaTimesNb = Na * Nb

            if( Nab != 0 ):

                MIM = Nab / (Na * Nb)
                dice = Nab / (Na + Nb)
                EMIM = Nab * math.log(N * MIM, 10)

            if( NaTimesNb != 0 ):
                numer = Nab - (NaTimesNb/N)
                chiSquare = (numer * numer) / NaTimesNb

            bDict['MIM'] = MIM
            bDict['EMIM'] = EMIM
            bDict['CHI-SQUARE'] = chiSquare
            bDict['DICE'] = dice


    for sortCriteria in ['MIM', 'EMIM', 'CHI-SQUARE', 'DICE']:

            print()

            sort = sorted( vocabDict.items(), key=lambda x: x[1][sortCriteria],
             reverse=True)
            sort = sort[:k]

            print(a, 'vs')
            counter = 1
            for termDict in sort:
                term, termDict = termDict
                print('\t', counter, 'term:', term, sortCriteria + ':',
                termDict[sortCriteria])
                counter += 1
```

```
def getVocabFreqDict(filenames, stop, ngramTup=(1, 1)):

    vocabDict = {}

    for i in range( len(filenames) ):
        f = filenames[i].strip()

        html = readTextFromFile(f)
        text = getTextFromHTML(html)
        #writeTextToFile(f + '.txt', text)

        if( len(text) == 0 ):
            continue

        countVectorizer = CountVectorizer( min_df=1, stop_words='english',
        ngram_range=ngramTup )
        termFreqMat = countVectorizer.fit_transform([text])

        for term in list(countVectorizer.vocabulary_.keys()):
            vocabDict.setdefault(term, {'f': []})
            vocabDict[term]['f'].append(f)

        if( i % 100 == 0 ):
            print( i, 'of', len(filenames) )

        if( i > stop ):
            break

    return vocabDict



word = 'hospital'
N = 6042
k = 20
getAssocMeasuresDocs(word, N, k)


'''
if( len(sys.argv) > 1 ):
    filename = 'wiki-small-vocab.json'
    word = sys.argv[1]
    N = 15103
    k = 20
    getAssocMeasuresWindow(word, N, filename, k)
'''
```

In order to compute term association measures for 10 words from the wiki small corpus I took the following steps:

1. I read the text files that were downloaded from wiki small corpus from to get HTML text

Problem 1 continued on next page...

2. I used the *boilerplate* removal library (Justext) to extract plain text.

3. I used sklearn *CountVectorizer* to get n-gram vocabulary. This is demonstrated in listing 1, line 271-278.

4. I created a dictionary with key as the term in the collection and value as the files in which the terms occur in (index)

5. I used *getAssocMeasuresDocs(a, N, k = 10)* to compute asscionation measures with the formula in Figure 1. For each term, I calculated association measures by referencing the index to find the number of times a term a occurs (Na), the number of times a term b occurs, (Nb), the number of time both terms occur in the index (Nab).

6. I computed association measures for the five word windows. Some of the examples for the association measures for my chosen word and five word windows are in Table 8-16 the complete file is in *10doc-5wind-words* folder.

7. For five word windows, I processed text and performed the same operations as before but each document was segmented into a group of five word sentences. Some of the examples for my ten chosen word and the top associated terms are in Table 1-8, The complete example file is in *10doc-5wind-words.*

Table 1: Top 5 MIM Association for "Election"

Associated Terms	Mutual Information Measure
parti	0.008771929824561403
patriote	0.008771929824561403
southport	0.008771929824561403
thefa	0.008771929824561403
fabricationfrom	0.008771929824561403

Table 2: Top 5 EMIM Association for "Food"

Associated Terms	Expected Mutual Information
time	36.988498354212695
large	32.71109290456123
water	31.139744388037478
long	30.34090215895875
people	29.50174435168828

Table 3: Top 5 Chi-square Association for "hospital"

Associated Terms	Chi-square
hospitals	0.11998634333107323
health	0.04862136324886152
mental	0.04351811544676538
medical	0.04278767992513581
care	0.03901883295625831

Table 4: Top 5 Dice's Association for "Sports"

Associated Terms	DiceâĂŹs coefficient
sport	0.13450292397660818
football	0.12334801762114538
championship	0.11940298507462686
basketball	0.11842105263157894
stadium	0.11258278145695365

Table 5: Top 5 MIM Association for "Crime" (5 Word Window)

Associated Terms	Mutual Information Measure
contentscomparison	0.009900990099009901
interpenetration	0.009900990099009901
dougbleday	0.009900990099009901
bionic	0.009900990099009901
decavalcante	0.009900990099009901

Table 6: Top 5 EMIM Association for "River" (5 Word Window)

Associated Terms	Expected Mutual Information Measure
nelson	8.012471932229197
tributary	7.524054612498825
wabash	6.855612505897404
banks	6.060661537344706
rats	5.511385944434505

Table 7: Top 5 Chi-square Association for "University" (5 Word Window)

Associated Terms	Chi-square Measure
time	0.006962271180089269
years	0.005781562236763527
new	0.0056631784106439065
wikipedia	0.005068187476213846
used	0.005016678300261902

Table 8: Top 5 Dices' Association for "Book" (5 Word Window)

Associated Terms	Dices' Measure
published	0.030461270670147953
comic	0.024759284731774415
movie	0.01794616151545364
lena	0.015759312320916905
wrote	0.015400410677618069

Problem 2

6.1. Using theWikipedia collection provided at the book website, create a sample of stem clusters by the following process:

Problem 2 continued on next page. . .

Measure	*Formula*
Mutual information (MIM)	$\frac{n_{ab}}{n_a.n_b}$
Expected Mutual Information ($EMIM$)	$n_{ab}.\log(N.\frac{n_{ab}}{n_a.n_b})$
Chi-square (χ^2)	$\frac{(n_{ab}-\frac{1}{N}.n_a.n_b)^2}{n_a.n_b}$
Dice's coefficient ($Dice$)	$\frac{n_{ab}}{n_a+n_b}$

Figure 1: Term association measures

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 (DiceâĂŹs 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

Solution 2:

Listing 2: Extract text from wikismall

```
from common import getDictFromFile
from Porter import PorterStemmer
import itertools

import networkx as nx

def getPairs(l):
    return list(itertools.combinations(l, 2))

def isWord(term):

    term = term.strip()
    for t in term:
        if( t.isalpha() == False ):
            return False

    return True

def getKAlphabeticalWords(k = 1000):

    index = getDictFromFile('wiki-small-vocab.json')
    sortedKeys = list(index.keys())
    sortedKeys.sort()

    counter = 0
    for i in range(len(sortedKeys)):

        if( isWord(sortedKeys[i]) == True ):
            counter += 1

            print(sortedKeys[i])


        if( counter == 1000 ):
            break

def getStemclasses():

    stemClasses = {}
    infile = open('good-1000-words.txt', 'r')
    terms = infile.readlines()
    infile.close()

    for voc in terms:
        voc = voc.strip()

        stem = PorterStemmer.useStemer(voc)
        stemClasses.setdefault(stem, [])
        stemClasses[stem].append(voc)

    return stemClasses

def getAssociationForPair(vocabDict, pair):

    a, b = pair

    Na = 0
    Nb = 0
    Nab = 0

    if( vocabDict[a] and vocabDict[b] ):

        aFileSet = set(vocabDict[a]['f'])
        bFileSet = set(vocabDict[b]['f'])

        Na = len(aFileSet)
        Nb = len(bFileSet)
        Nab = len(aFileSet & bFileSet)

    if( Nab != 0 ):
        return Nab / (Na + Nb)
    else:
        return 0
```

Problem 2 continued on next page. . .

```
def compAssocForPairsInStemClass(stemClasses):

    vocabDict = getDictFromFile('wiki-small-vocab.json')
    counter = 0
    total = len(stemClasses)

    for stem, classList in stemClasses.items():


        if( len(classList) < 2 ):
            continue

        pairs = getPairs(classList)
        print('stem:', stem)
        print('\tstem class:', classList, '\n')
        for i in range(len(pairs)):


            dice = getAssociationForPair(vocabDict, pairs[i])

            print('\tpair:', pairs[i])
            print('\tdice:', dice)
            print('\t', counter, 'of', total, '\n')

        print()
        counter += 1

def compAssocForPairsInStemClassThreshold(stemClasses, threshold):

    vocabDict = getDictFromFile('wiki-small-vocab.json')
    counter = 0
    total = len(stemClasses)

    for stem, classList in stemClasses.items():


        if( len(classList) < 2 ):
            continue


        G = nx.Graph()
        G.add_nodes_from(classList)
        pairs = getPairs(classList)
        stemDice = 0
        for i in range(len(pairs)):


            dice = getAssociationForPair(vocabDict, pairs[i])

            if( dice >= threshold ):
                G.add_edge( pairs[i][0], pairs[i][1] )
                stemDice = dice

```

Problem 2 continued on next page. . .

```
              if( len(G.edges()) != 0 ):
130                conComp = list(nx.connected_component_subgraphs(G))

                   print('stem:', stem)
                   print('\tdice:', stemDice)
                   print('\told stem class:', classList, '\n')
135                print('\tNew stem class for stem:')

                   for subgraph in conComp:
                        subgraph = subgraph.nodes()
                        if( len(subgraph) > 1 ):
140                          print('\t', subgraph)

              print()
              counter += 1

145


    stemClasses = getStemclasses()


150 #compAssocForPairsInStemClass(stemClasses)

    #threshold = 0.002
    #compAssocForPairsInStemClassThreshold(stemClasses, threshold)
```

In order to solve the above problem, I took the following steps:

1. I read the text files to get HTML text
2. I used the *boilerplate* removal library (Justext) to extract plain text.
3. I created a dictionary (from inverted index), with the key as the term in the wiki small collection, and the value as the list of files that include the term. The output is the index called *wiki-small-vocab.json*.
4. I used *getKAlphabeticalWords(k = 1000)* in Listing 2 to extract in alphabetical order, 1,000 terms from *wiki-small-vocab.json* that are words: That is terms that have only letters of the alphabets: e.g "gold" and not "1gold". The output is *good-1000-words.txt*
5. I used *getStemclasses()* in Listing 2 to create stem classs for the 1,000 words from *good-1000-words.txt* and the words that become the same stem class is in *good-1000-words-stem-classes.txt*.
6. I used *compAssocForPairsInStemClass(stemClasses)* in Listing 2 to compute association measures (DiceâĂŹs coefficient) between all pairs of stems in each stem class and the result is in *good-1000-words-dice.txt*. The stem classes are represented as a dictionary. The key is the stem, the value is a list of words that map to the stem
7. I used *compAssocForPairsInStemClassThreshold(stemClasses, threshold)* in Listing 2 to create a stem cluster by thresholding the association measure. I used networkx python library to create a graph with nodes as the stems. I added an edge if the Dices' association exceeded the threshold. I used the connected components of the graph to generate the new stem classes. Listing 2, line 115-140

Problem 3

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.

Solution 3:

Listing 3: Extract Galago Snippet Extraction Code

```


   private String generateSnippet(
           final Document document,
5          final ArrayList<IntSpan> positions,
           final Set<String> queryTerms) {
     ArrayList<SnippetRegion> regions = findMatches(document, queryTerms);
     ArrayList<SnippetRegion> finalRegions = combineRegions(regions);
     Snippet best = new Snippet(finalRegions);
10
     return buildHtmlString(best, document, positions);
   }
```

The Galago algorithm in Listing 3 can be summarized as follows:

1. Find a region of text in the document that contains any query term, return a region (left and right) of text based on a predefined text width from the match - *findMatches(), Listing 3 line 6*
2. Combine all regions of text which matched the query term by merging regions that overlap, but break on sentences when predefined merge window size is reached - *combineRegions(), Listing 3, line 6*
3. Highlight terms that match query in regions - *buildHtmlString()*

Would this algorithm work well for pages with little text content?

This algorithm set the region for a match to 5 words to the left and 5 words to the right of the position of the query term. Since documents are not typically very small, this threshold is reasonable. The performance of the algorithm is based on finding region that match the query, not the size of the document. Based on this, the content of the document ought not to affect the performance of the algorithm. The content will only affect the number of candidates to be matched. If the document is too small, there will have lesser candidates to match. The algorithm will work well for pages with lesser content

Ways to improve the Galago algorithm

1. Exact match should be supplemented with associated words in the same stem class (with high dice or chi-square association), for example, we should match tenses such as: “fast” and “fasting”
2. The algorithm should include regions of alias terms. For example, if the query includes “Barack,” we should also generate snippets that match “Obama”. Also for “NBA”, we should match regions with “National Football Association.” The algorithm should include regions of synonyms terms. For example, if the query include “Beautiful,” we should also match regions with “attractive”

Problem 4

7.7. What is the "bucket" analogy for a bigram language model? Give examples.

Solution 4:

Introduction
A language model is a probability distribution over sequences of words. In other words, a language model assigns probabilities to sequences of words. This shows us how likely a sequence occurs (or generated). Language models are useful in a variety of problems such as spell correction. In Information Retrieval, we use language models to rank each document in a collection by their respective probabilities to a given query. This is called Query likelihood model. This tells us how relevant a document is to a query P(Q|D). In this language model, every document in the collection is a separate language model.

Bucket analogy

The bucket or bag of words analogy simply means that each document in a collection is represented as a collection of words. There is no order in a bucket. If we represent the document as a collection of single words - we would get a bucket of unigrams, but if we represent the words as a collection of two words - we would get a bucket of bigrams. For example here is a document with the following buckets of unigrams and bigrams:

Document: "The quick brown fox jumped over the lazy dog"

Unigram bucket:
The (occurs twice)
quick
brown
fox
jumped
over
lazy
dog
Bigram bucket:
The quck
quick brown
brown fox
fox jumped
jumped over
over the
the lazy
lazy dog

Bigram bucket analogy

The bucket analogy for a bigram language model illustrates how we assign probabilities to bigrams with a bigram language model. In other words we measure the probabilities of extracting bigrams from a bucket consisting of bigrams (a document). This means every document is represented as a collection of bigrams. The probability of a given term in a query in the bigram language model depends on the probability of the previous term. Generally, the probability of a single word wi is in Figure 2.

Problem 4 continued on next page. . .

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Figure 2: Measuring n-gram probability

```
Document:
<start>the quick brown fox jumped over</end>
<start>fox jumped over us yesterday</end>
<start>fox jumped over the tent</end>


P(the quick brown) =
P(the | <start>) . P(quick | the) . P(brown | quick) . P(</end> | brown)

P(the | <start>) = 1/3  (1 sentence where âĂIJtheâĂİ after âĂIJstartâĂİ/ 3 sentences with s
P(quick | the) = 1/2
P(brown | quick) = 1/1

P(the quick brown) = 1/3 . 1/3 . 1 = 0.16666666666
```

Problem 5

MLN1: using the small wikipedia example, choose 10 words and create stem classes as per the algorithm on pp. 191-192

Algorithm

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

Solution 5:

Problem 5 continued on next page. . .

Listing 4: Generate stem class based on given algorithm

```python
from Porter import PorterStemmer
import itertools

from common import getDictFromFile
from common import dumpJsonToFile
from common import genericErrorInfo
from common import getTokenizer
from common import readTextFromFile

import networkx as nx

#CAUTION: duplicate with A3.P1.py
def searchKwordWindowsOpt(text, k, left, right, skipBothFlag=False):

     tokens = []
     try:
          tokens = getTokenizer(text)
     except:
          genericErrorInfo()

     counts = {'left': 0, 'both': 0}

     kWordWindows = []
     for i in range(len(tokens)):

          if( i % k == 0 ):
               kWordWindows.append([])

          kWordWindows[-1].append( tokens[i] )

     counts['left'] = len(kWordWindows)

     if( skipBothFlag == False ):
          for win in kWordWindows:
               if( left in win and right in win ):
                    counts['both'] += 1

     return counts

def getPairs(l):
     return list(itertools.combinations(l, 2))

def getAssociationForPair(vocabDict, pair, windowSize):

     a, b = pair

     Na = 0
     Nb = 0
     Nab = 0

     if( vocabDict[a] and vocabDict[b] ):
          for f in vocabDict[a]['f']:
               f = f + '.txt'
               f = readTextFromFile(f)

               counts = searchKwordWindowsOpt(f, windowSize, a, b)

               Na += counts['left']
               Nab += counts['both']

          for f in vocabDict[b]['f']:
               counts = searchKwordWindowsOpt(f, windowSize, b, a, True)
               Nb += counts['left']


     if( Nab != 0 ):
          return Nab / (Na + Nb)
     else:
          return -1



def optimizeStemClass(oldStemClass, windowSize, threshold):


     vocabDict = getDictFromFile('wiki-small-vocab.json')
     counter = 0
     total = len(oldStemClass)
     for stem, classList in oldStemClass.items():

          pairs = getPairs(classList)


          G = nx.Graph()
          G.add_nodes_from(classList)

          for i in range(len(pairs)):


               dice = getAssociationForPair(vocabDict, pairs[i], windowSize)

               if( dice >= threshold ):
                    G.add_edge( pairs[i][0], pairs[i][1] )

          if( counter % 10 == 0 ):
               print(counter, 'of', total, 'dice:', dice, '\n')

          if( len(G.edges()) != 0 ):

               print('Graph:')
               print('nodes:', G.nodes())
               print('edges:', G.edges())

               conComp = list(nx.connected_component_subgraphs(G))
```

Problem 5 continued on next page. . .

```
                print()
                print('New stem class for stem:', stem, ':')
                for subgraph in conComp:

                    subgraph = subgraph.nodes()
                    if( len(subgraph) > 1 ):
                        print('\t', subgraph)

          counter += 1


def getStemsClassesSizeKPlus(k=2):

     stemClasses = getDictFromFile('wiki-small-vocab-stem-classes.json')
     chosenStemClasses = {}

     for stem, classList in stemClasses.items():
          if( len(classList) >= k ):
               chosenStemClasses[stem] = classList

     diff = len(stemClasses) - len(chosenStemClasses)

     print('old:', len(stemClasses))
     print('new:', len(chosenStemClasses))
     print('getStemsClassesSizeKPlus() - diff:', diff, '\n')
     return chosenStemClasses

def getStemclasses():

     stemClasses = {}
     vocabDict = getDictFromFile('wiki-small-vocab.json')
     counter = 0

     for voc, vocDict in vocabDict.items():

          stem = PorterStemmer.useStemer(voc)
          stemClasses.setdefault(stem, [])
          stemClasses[stem].append(voc)

          if( counter % 10000 == 0 ):
               print('\t', counter, voc)

          counter += 1

     dumpJsonToFile('wiki-small-vocab-stem-classes.json', stemClasses, False)



getStemclasses()


sizeOfStemClass = 2
```

Problem 5 continued on next page. . .

```
chosenStemClasses = getStemsClassesSizeKPlus(sizeOfStemClass)

windowSize = 80
threshold = 0.003
optimizeStemClass(chosenStemClasses, windowSize, threshold)
```

Listing 5: Stem class snippet

```
stem: abducte
     stem class: ['abductee', 'abductees']

     pair: ('abductee', 'abductees')
     dice: 0
      11 of 617


stem: abid
     stem class: ['abide', 'abiding']

     pair: ('abide', 'abiding')
     dice: 0
      12 of 617


stem: abil
     stem class: ['abilities', 'ability']

     pair: ('abilities', 'ability')
     dice: 0.10614525139664804
      13 of 617


stem: abl
     stem class: ['able', 'abled', 'ables']

     pair: ('able', 'abled')
     dice: 0
      14 of 617

     pair: ('able', 'ables')
     dice: 0.004405286343612335
      14 of 617

     pair: ('abled', 'ables')
     dice: 0
      14 of 617
```

1. I read the file to get HTML
2. I used *boilerplate* removal (Justext) to get plain text.
3. I applied Porter to plain text to get stem classes. This is achieved with *getStemclasses()* in Listing 4
4. I used *getStemclasses()* in Listing 4 to generate a dictionary with stem as key, and value as list of terms that map to the stem (stem classes)

Problem 5 continued on next page. . .

5. I used *getAssociationForPair(vocabDict, pair, windowSize)* in Listing 4 to compute association measures (Dices coefficient) between all pairs of stems in each stem class. The stem classes are represented as a dictionary. The key is the stem, the value is a list of words that map to the stem

6. I used *optimizeStemClass(oldStemClass, windowSize, threshold))* in Listing 4 to create a stem cluster by thresholding the association measure. I used networkx python library to create a graph with nodes as the stems. I added an edge if the Dice association exceeded the threshold. I used the connected components of the graph to generate the new stem classes with *getStemsClassesSizeKPlus(k=2)* in Listing 4. Snippet of the new stem class is shown in Listing 5 while the complete stem class is saved in *6.1-stem-classes.txt*

Problem 6

7.2. Can you think of another measure of similarity that could be used in the vector space model? Compare your measure with the cosine correlation using some example documents and queries with made-up weights. Browse the IR literature on the Web and see whether your measure has been studied (start with van RijsbergenâĂŹs book).
Solution

I came up with a measure of similarity based on multiset - a generalization of sets which allows duplicates. The similarity can be defined as ratio of the size of the multiset intersection and size of the vocabulary. This is very similar to Jaccard, however, Jaccard is meant for sets (no duplicates). My similarity metric generalizes Jaccard because it takes into account the frequency of occurrence of terms.
Example1

Doc1: “Tropical Freshwater Aquarium Fish”
Doc2: “Tropical Fish, Aquarium Care, Tank Setup”

Vocabulary vector: tropical, freshwater, aquarium, fish, care, tank, setup
Doc1 vector: [1, 1, 1, 1, 0, 0, 0]
Doc2 vector: [1, 0, 1, 1, 1, 1, 1],

Doc1 multiset: Tropical, Freshwater, Aquarium, Fish
Doc2 multiset: Tropical, Fish, Aquarium, Care, Tank, Setup

Cosine (Doc1, Doc2) = 0.612372

```
Multiset similarity = {Tropical, Freshwater, Aquarium, Fish} intersect {Tropical,
Fish, Aquarium, Care, Tank, Setup}| / |{tropical, freshwater, aquarium, fish,
care, tank, setup}|

Multiset similarity = 3/6 = 0.5
```

Problem 7

6.4. Assuming you had a gazetteer of place names available, sketch out an algorithm for detecting place names or locations in queries. Show examples of the types of queries where your algorithm would succeed

Problem 7 continued on next page. . .

and where it would fail.

Solution

Locations of places are hierarchical in order. For example Old Dominion University is located as follows;

Country: USA
State: Virginia
City: Norfolk
Place: Old Dominion University

Let my gazetteer be structured hierarchically, for example here is a small portion of the gazetteer:

```
Gazetteer:
USA
    Virginia
        Norfolk
            Hampton
            Old Dominion University
            Norfolk State University
        Chesapeake
            Hampton

         ...Tide Water Community College
    West Virginia
    ...
...
Canada
```

Listing 6: Algorithm to identify locations in query

```
Algorithm for identifying locations in queries

tokens = tokenize(query)

matches = {}
for token in query:

     if( isCountry(token) == True ):
          matches[token] = 'Country'

     else if( isState(token) == True ):
          matches[token] = 'State'

     else if( isCity(token) == True ):
          matches[token] = 'City'

     else if( isPlace(token) == True ):
          matches[token] = 'Place'
```

Positive examples:

Query1: “Where is Virginia located?”
Result: “Where is [Virginia, State] located?”

Query2: “Directions for Hampton Norfolk”
Result: “Directions for [Hampton, Place] [Norfolk, City]”
Negative examples:

Query1: “Where is West Virginia located?”
Result: “Where is West [Virginia, State] located?”
Failure: Due to multiple words in state name, the tokenizer algorithm only considers single word locations which is not very practical.

Query1: “Where is Hampton Inn located?”
Result: “Where is [Hampton, Place] Inn located?”
Failure: Due to ambiguity in the query

As we can see this algorithm is very simplistic and suffers from multiple problems including failure in recognizing aliases of locations. Also it is computationally very expensive to go through the entire gazetteer for each token.

Problem 8

6.9. Give five examples of web page translation that you think is poor. Why do you think the translation failed?

Solution
Five example of poor web page translations are:

1. http://www.cnn.com/2014/01/15/politics/obamacare-spanish-language-site/index.html
2. http://english.cntv.cn/program/learnchinese/specialchinese/index.shtml

References

[1] Bigram Language Model. http://www.phontron.com/slides/nlp-programming-en-02-bigramlm.pdf. Accessed: 2017-11-05.

[2] Countvectorizer. http://scikitlearn.org/stable/modules/feature_extraction.html. Accessed: 2017-10-10.

[3] Information Retrieval in Practice. http://ciir.cs.umass.edu/downloads/SEIRiP.pdf. Accessed: 2017-10-10.

[4] Jaccard index. https://en.wikipedia.org/wiki/Multiset. Accessed: 2017-11-05.

[5] Justext. https://pypi.python.org/pypi/jusText/2.1.1. Accessed: 2017-10-10.

[6] Language Modeling. https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf. Accessed: 2017-11-05.

[7] Lectureslide. https://raw.githubusercontent.com/phonedude/cs834-f17/. Accessed: 2017-11-05.

[8] Porterstemmer. https://pypi.python.org/pypi/PorterStemmer. Accessed: 2017-10-10.

[9] Query likelihood model. https://en.wikipedia.org/wiki/Query_likelihood_model. Accessed: 2017-11-05.