INTRO. TO INFO RETRIEVAL: CS 734: A4

Due on Monday, December 4, 2017

Dr. Nelson

Udochukwu Nweke

Contents

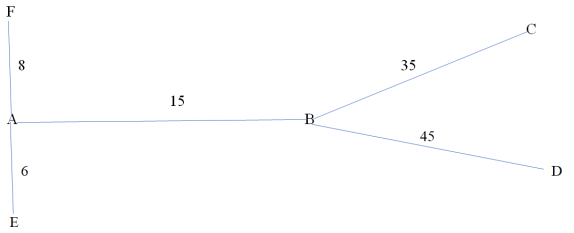

Figure 1: Asymetric Nearest neighbor

Problem 1

9.10. Nearest neighbor clusters are not symmetric, in the sense that if instance A is one of instance B's nearest neighbors, the reverse is not necessarily true. Explain how this can happen with a diagram.

Solution 1:

Figure 1 shows that nearest neighbors are not symmetric. It can be seen that although A is B's nearest neighbor because A has the closest distance to B compared to C and D, the reverse is not the case here because B is not the closest distance to A. E is A's nearest neighbor because E has the smallest distance to A. This explains how nearest neighbors are not symmetric.

Problem 2

9.11. The K nearest neighbors of a document could be represented by links to those documents. Describe two ways this representation could be used in a search application.

Solution 2:

K nearest neighbors of a document could be represented by links to those documents. This means that document will be linked to similar documents. These similar documents are considered the nearest neighbor of the document. K-NN can be used in search application where a user's task is to find items that are "similar to a particular item". This is referred to as K-NN search. Similarity is measured by creating a vector representation of the documents links, then we use a distance metric (Euclidean distance) to compare the vectors, or cosine similarity.

Examples of K-NN search:

Concept Search: In concept search, a user searches documents that contain similar topics. Concept search is found in several e-Discovery software packages, which are used to help organizations find all the e-mails, contracts, etc. that are relevanant to the organization.

Recommender Systems:Another use of K-NN search is recommender systems. If a user likes a particular item, a similar item can be recommended to them. K-NN search helps find similar items. If similar users liked two different items, it is possible that the items are similar. This is used in recommending products, recomemending advertisements to display to a user, etc.

Problem 3

9.8. Cluster the following set of two-dimensional instances into three clusters using each of the five agglomerative clustering methods:
(-4, -2), (-3, -2), (-2, -2), (-1, -2), (1, -1), (1, 1), (2, 3), (3, 2), (3, 4), (4, 3)
Discuss the differences in the clusters across methods. Which methods produce the same clusters? How do these clusters compare to how you would manually cluster the points?

Solution 3:

Listing 1: Agglomerative Clustering Code Snippet

```
# -*- coding: utf-8 -*-
from scipy import cluster
import scipy.cluster.hierarchy as hac
import matplotlib.pyplot as plt
import numpy as np

def clustering():
    #https://stackoverflow.com/a/21723473
    a = np.array([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1, 1],
        [2, 3], [3, 2], [3, 4], [4, 3]])

    fig, axes23 = plt.subplots(2, 3)

    for method, axes in zip(['ward'], axes23):
        z = hac.linkage(a, method=method)

        # Plotting
        axes[0].plot(range(1, len(z)+1), z[::-1, 2])
        knee = np.diff(z[::-1, 2], 2)
        axes[0].plot(range(2, len(z)), knee)

        num_clust1 = knee.argmax() + 2
        knee[knee.argmax()] = 0
        num_clust2 = knee.argmax() + 2


        #axes[0].text(num_clust1, z[::-1, 2][num_clust1-1], 'possible\n<- knee point')
        print 'num_clust1:', num_clust1
        print 'num_clust2:', num_clust2
        #num_clust1 = 3
        part1 = hac.fcluster(z, num_clust1, 'maxclust')
        part2 = hac.fcluster(z, num_clust2, 'maxclust')
        print('part1:', part1)
        print('part2:', part2)



        clr = ['#2200CC' ,'#D9007E' ,'#FF6600' ,'#FFCC00' ,'#ACE600' ,'#0099CC' ,
        '#8900CC' ,'#FF0000' ,'#FF9900' ,'#FFFF00' ,'#00CC01' ,'#0055CC']

        for part, ax in zip([part1, part2], axes[1:]):
```

```
                for cluster in set(part):
                    ax.scatter(a[part == cluster, 0], a[part == cluster, 1],
                               color=clr[cluster])

        m = '\n(method: {})'.format(method)
        plt.setp(axes[0], title='Screeplot{}'.format(m), xlabel='partition',
         ylabel='{}\ncluster distance'.format(m))

        c0 = len(set(part1.tolist()))
        c1 = len(set(part2.tolist()))

        plt.setp(axes[1], title='{} Clusters'.format(c0))
        plt.setp(axes[2], title='{} Clusters'.format(c1))

        plt.tight_layout()

        plt.savefig('clustering.png')

def plotOriginalPoints():

    fig = plt.figure()
    fig.suptitle('Original data points', fontsize=14, fontweight='bold')

    ax = fig.add_subplot(111)
    fig.subplots_adjust(top=0.85)
    #ax.set_title('axes title')
    ax.set_xlabel('x')
    ax.set_ylabel('y')

    #Original points
    sampMat = [[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1, 1],
     [2, 3], [3, 2], [3, 4], [4, 3]]
    X = np.array(sampMat)

    #add labels
    for i in range(len(sampMat)):
        ax.text(sampMat[i][0], sampMat[i][1], str(i+1))

    #plot points
    ax.plot(X[:,0], X[:,1], 'o', color='r')

    #set window size
    ax.axis([-5, 6, -3, 6])

    #save plot
    plt.savefig('originalPoints.png')

#plotOriginalPoints()

clustering()
```

1. Agglomerative clustering is a type of hierarchical clustering. In agglomerative clustering, each data point is defined to be a cluster, and existing clusters are combined at each step. There are for different

methods for achieving these steps and they are:

2. **Single Linkage**: In this method of clustering, the distance between two clusters is defined to be the minimum distance between any single data point in cluster A and any single data point in cluster B. At each stage in the process, the two clusters that have the smallest single linkage distance is combined.

3. **Complete Linkage**: In complete linkage, the distance between two clusters is defined to be the maximum distance between any single data point in the first cluster and any single data point in the second cluster. At each step, the two clusters that have the smallest complete linkage distance is combined.

4. **Average Linkage**: In this method of clustering, the distance between two clusters is defined to be the average distance between data points in the first cluster and the data points in the second cluster. At each step, the two clusters that have the smallest average link distance is combined.

5. **Centroid Approach**: In this approach, the distance between two clusters is defined as the distance between two mean vectors of the clusters. At each step, the two clusters that have the smallest centroid distance is combined.

6. **Ward's Approach**: This approach is not based on distances of two clusters like the previous agglomerative method. Wards is based on the statistical property of variation. The variance of a set of numbers measures how spread out the numbers are.

Figure 2 shows how the different agglomerative clustering methods are represented.

I used *clustering()* in Listing 1 to cluster the given set of two-dimensional instances into three clusters using the five agglomerative clustering method I have explained.

All linkages resulted in the same clustering when K=3. This gave me some concern. For some time, I thought I was doing something wrong or there was something wrong with the algorithm. When I changed the number of K, the clustering was different. For K= 3, all 5 methods gave the same clustering and the case is different when K is not 3. K= 2 gave the same clustering for average method, ward method, and complete method. The result of the clustering can be seen in Figures 3, 4, 5, 6, and 7 respectively.

For manual clustering, I used *plotOriginalPoints()* in Listing 1 to manually cluster the given set of two-dimensional instances. The result is in Figure 8. The result shows that the algorithm did well K= 3 and K= 2 clusterings showed similar clustersing pattern with the manual clustering.

My manual clustering of the given set of the two-dimensional instances into three clusters will be as follows:
Cluster 1: Point 1,2,3,4
Cluster 2: Point 5,6
Cluster 3: point 7,8,9,10

Problem 4

9.6. Compare the accuracy of a one versus all SVM classifier and a one versus one SVM classifier on a multiclass classification data set. Discuss any differences observed in terms of the efficiency and effectiveness of the two approaches.

Solution 4:

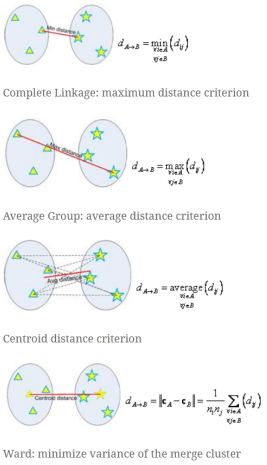

Figure 2: Agglomerative Clustering Methods

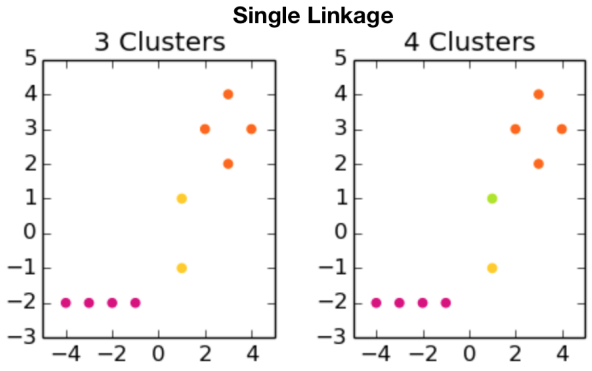

Figure 3: Single Linkage

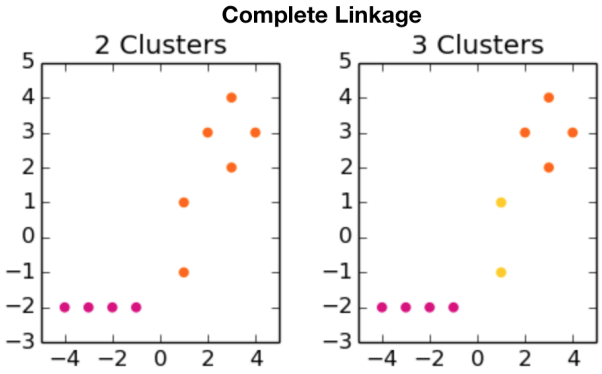

Figure 4: Complete Linkage

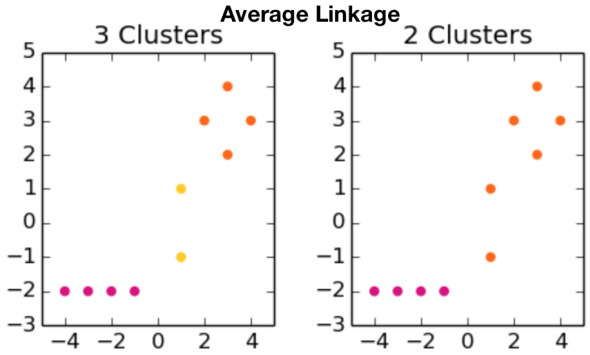

Figure 5: average Linkage

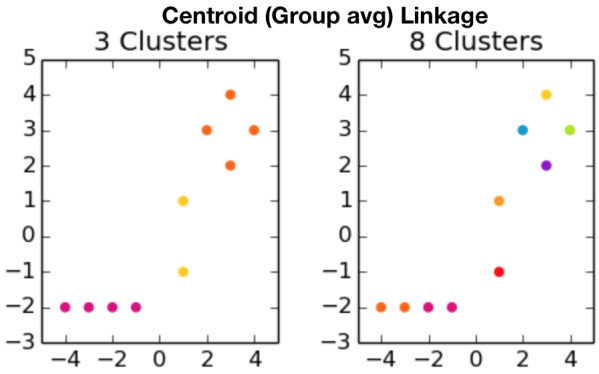

Figure 6: Centroid Linkage

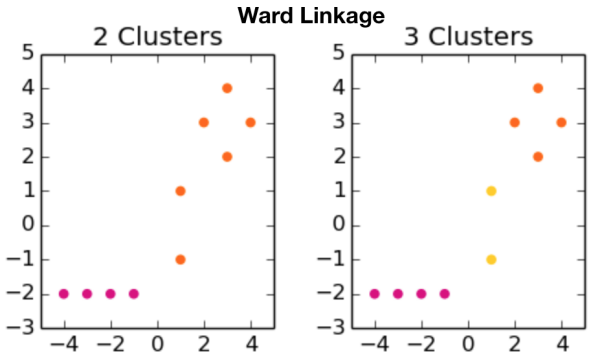

Figure 7: Ward Linkage

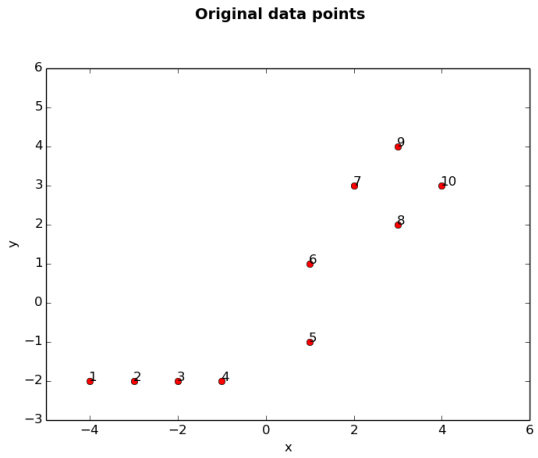

Figure 8: Manual Clustering

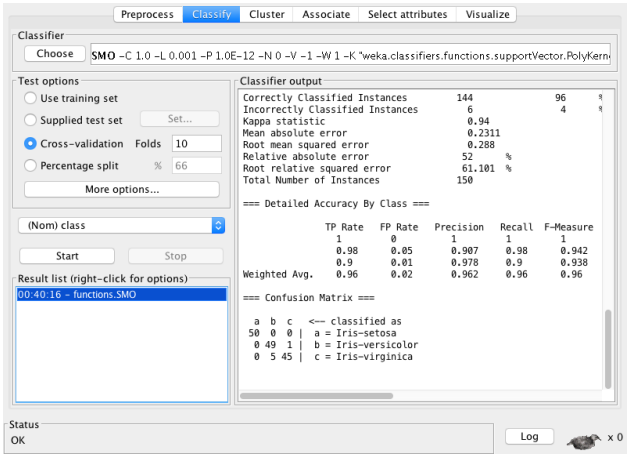

Figure 9: oneVsOne SVM Classifier result

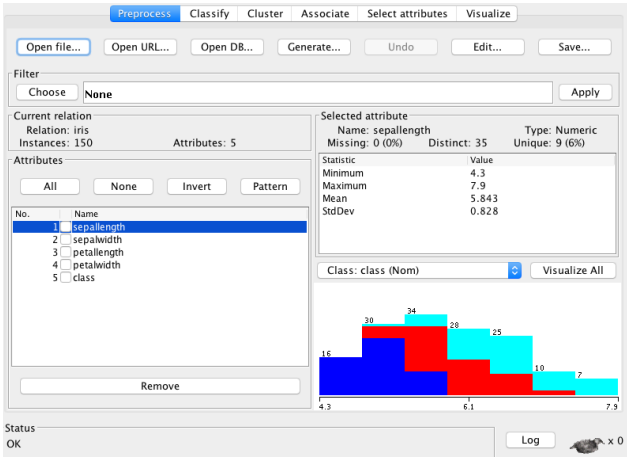

Figure 10: OneVsOne on a Multiclass Classification Data Set

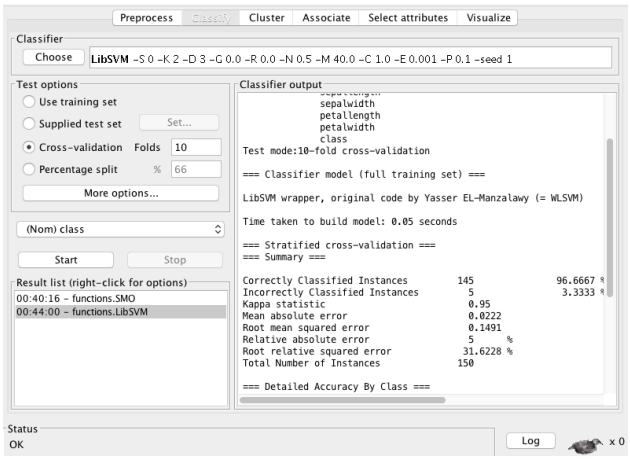

Figure 11: Result of OneVsAllResult

The Sequential Minimal Optimization (SMO) implements a one vs one multi class SVM classifier, while the LibSVM classifier implements a one vs all classifier.

I used Weka to test the effectiveness and efficiency of both methods on a multi-class dataset: Iris dataset (provided by weka). I used Weka because it is a popular application for classification tasks and I did not have to write the code to test both classifiers.

The Iris dataset (iris.arff) is as follows:

@RELATION iris

@ATTRIBUTE sepallength REAL
@ATTRIBUTE sepalwidth REAL
@ATTRIBUTE petallength REAL
@ATTRIBUTE petalwidth REAL
@ATTRIBUTE class Iris-setosa,Iris-versicolor,Iris-virginica

There are 3 classes for classifying Iris flower based on the attributes: sepallength, sepalwidth, petallength and petalwidth. This is seen in Figure 10

Figure 9 shows the result of the One vs One (SMO) classification result

Figure 11 shows the result of the One vs All (SMO) classification result

```
One vs One (SMO) classification result      One vs All (LibSVM) classification result

Time taken to build model: 0.12 seconds     Time taken to build model: 0.05 seconds


=== Stratified cross-validation ===        === Stratified cross-validation ===
=== Summary ===                            === Summary ===

Correctly Classified Instances  144  96 % Correctly Classified Instances  144 96 %
```

```
Incorrectly Classified Instances 6    4 %   Incorrectly Classified Instances  6   4 %
Kappa statistic                  0.94       Kappa statistic            0.94
Mean absolute error              0.2311     Mean absolute error        0.2311
Root mean squared error          0.288      Root mean squared error    0.288
Relative absolute error          52%        Relative absolute error    52  %
Root relative squared error      61.101%    Root relative squared error 61.101%
Total Number of Instances        150        Total Number of Instances   150
```

Both methods show the same accuracy (96%). This is because they implement the same underlying SVM algorithm. SVMs are inherently used for binary classification, but can be used for multi-class problems using:

1. One vs rest (one vs all)
2. One vs One (multiple classifiers; choose the class that is selected by the most classifiers)

However, the one-vs-one method is less efficient since it requires multiple sub-classification tasks: this is seen in the faster processing time of the LIBSVM method.

Problem 5

Use K-means and spherical K-means to cluster the data points in Exercise 9.9. How do the clusterings differ?

Solution 5:

Listing 2: K-means and spherical K-means Clustering Code Snippet

```
from sklearn.cluster import KMeans
import numpy as np
from sphere cluster import SphericalKMeans

X = np.array([[-4, -2], [-3, -2], [-2, -2], [-1, -2],
     [1, -1], [1, 1], [2, 3], [3, 2], [3, 4], [4, 3]])


def testKMeans():
     kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
     print( kmeans.labels_ )
     #print( kmeans.predict([[0, 0], [4, 4]]) )

def testSpericalKMeans():
     # Find K clusters from data matrix X (n_examples x n_features)
     # spherical k-means

     skm = SphericalKMeans(n_clusters=3)
     skm.fit(X)
     print( skm.labels_ )

testKMeans()
testSpericalKMeans()
```

The k-means clustering algorithm attempts to minimize a Euclidean distance between the center of a given cluster and the members of the cluster.

Problem 5 continued on next page...

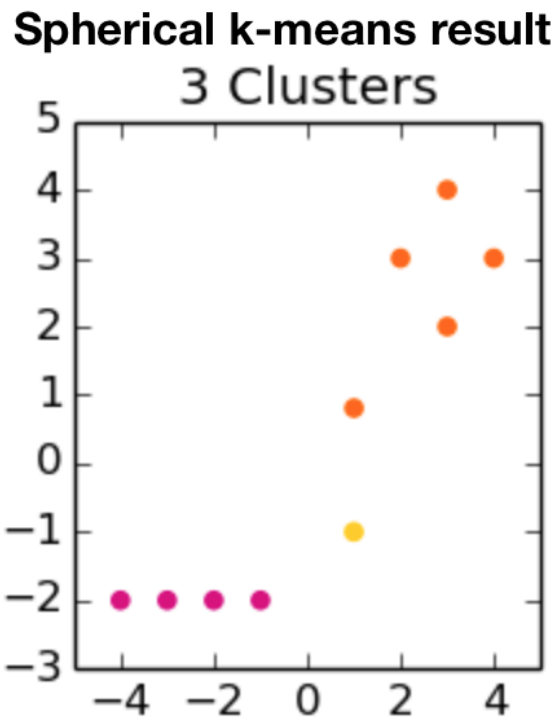

Figure 12: Spherical K-clustering

I used sklearn's k-means algorithm to cluster the data point. This is demonstrated with *testKMeans()* in listing 2

I specified 3 clusters for the algorithm. Figure **??** shows the clustering result. This is exactly the same as the hierarchical clustering result.

Spherical k-means

Spherical k-means attempts to minimize the angle between the center of a given cluster and the members of the cluster.

I used a python library (spherecluster) to test spherical k-means. The result was different from the previous k-means and hierarchical clustering. I believe this is because it minimizes angles and not just distances. Figure 13 shows the result of the spherical k-means clustering.

Problem 6

Listing 3: Multiple-Bernoulli and Multinomial Models Code Snippet

```
import numpy as np

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

```

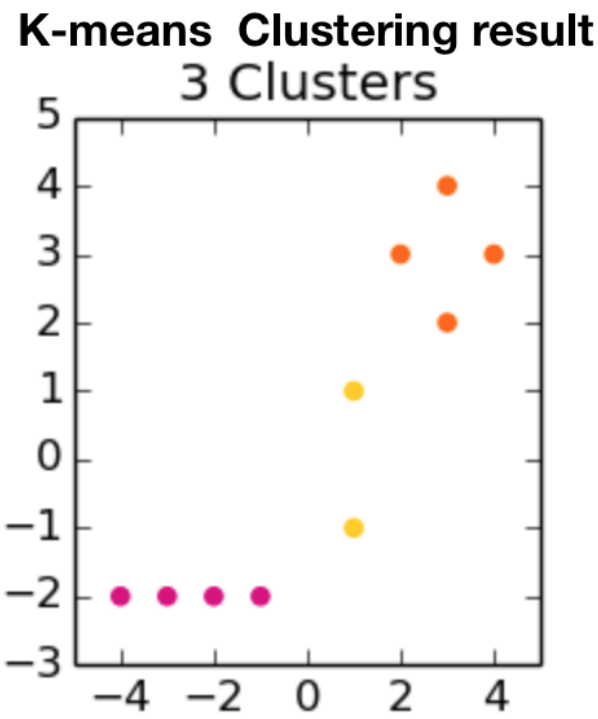

Figure 13: Kmeans Clustering

```
#spam dataset: https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection

def getInput(filename):

    infile = open(filename, 'r')
    lines = infile.readlines()
    infile.close()

    del lines[0]

    spamClassDict = {}
    spamClassDict['NOT'] = []
    spamClassDict['SPAM'] = []
    for line in lines:
        line = line.strip().split(' <> ')

        doc = line[0].strip()
        spamClass = line[1].strip()

        if( spamClass == '0' ):
            spamClassDict['NOT'].append(doc)
        else:
            spamClassDict['SPAM'].append(doc)

    return spamClassDict
```

```
def getTFMatrix(docList):
    np.set_printoptions(threshold=np.nan, linewidth=110)

    count_vectorizer = CountVectorizer(ngram_range=(1,1))
    term_freq_matrix = count_vectorizer.fit_transform(docList)

    sortedVocab = sorted(count_vectorizer.vocabulary_.items(), key=lambda x: x[1])

    #print( sortedVocab )
    #print( term_freq_matrix.todense() )

    return {'docMat': term_freq_matrix.todense().tolist(), 'vocab': sortedVocab}

def getTermIndexFromVocab(word, vocab):

    for term in vocab:
        term, index = term

        if( word == term ):
            return index

    return -1

def calcMultinomial(w, c, trainingSet):

    C = 0
    df_wc = 0

    wi = getTermIndexFromVocab(w, trainingSet['vocab'])

    if( wi != -1 ):
        start = 0
        end = 0

        if( c == 'NOT' ):
            #search non spam class
            start = 0
            end = 175
        else:
            #search spam class
            start = 175
            end = len(trainingSet['docMat'])

        for i in range(start, end):
            vec = trainingSet['docMat'][i]
            df_wc += vec[wi]
            C += sum(vec)
    else:
        #print('Not found in vocab')
        pass

    if( C != 0 ):
        return df_wc/C
```

```
85        else:
              return 0

def calcMultBernoulli(w, c, trainingSet):

90        N_c = 0
          df_wc = 0

          wi = getTermIndexFromVocab(w, trainingSet['vocab'])

95        if( wi != -1 ):
              start = 0
              end = 0

              if( c == 'NOT' ):
100               #search non spam class
                  start = 0
                  end = 175
              else:
                  #search spam class
105               start = 175
                  end = len(trainingSet['docMat'])

              for i in range(start, end):
                  N_c += 1
110               vec = trainingSet['docMat'][i]
                  if( vec[wi] != 0 ):
                      df_wc += 1

          else:
115           #print('Not found in vocab')
              pass

          p = df_wc/N_c
          #print('df_wc:', df_wc)
120       #print( 'P(w|c) = P(' + w + '|' + c + ') = ' + str(p) )
          return p

def testProbModels(trainingSet):

125       for term in trainingSet['vocab']:
              term = term[0]
              print('\n')
              print('*' * 50)
              print('Multiple-Bernoulli')
130           print('\t' + term)
              print('\t\tP(SPAM) = ', calcMultBernoulli(term, 'SPAM', trainingSet))
              print('\t\tP(NOT) = ', calcMultBernoulli(term, 'NOT', trainingSet))

              print('\nMultinomial')
135           print('\t\tP(SPAM) = ', calcMultinomial(term, 'SPAM', trainingSet))
              print('\t\tP(NOT) = ', calcMultinomial(term, 'NOT', trainingSet))
```

```
filename = 'Youtube02-KatyPerry.csv'
spamClassDict = getInput(filename)

dataset = spamClassDict['NOT'] + spamClassDict['SPAM']
trainingSet = getTFMatrix(dataset)

testProbModels(trainingSet)

#print( spamClassDict['0'] )
#print(spamClassDict)
```

9.4. For some classification data set, compute estimates for P(w|c) for all words using both the multiple-Bernoulli and multinomial models. Compare the multipleBernoulli estimates with the multinomial estimates. How do they differ? Do the estimates diverge more for certain types of terms?
Solution

Multiple-Bernoulli Model

The goal of multiple-Bernoulli model is to estimate the probability that a term is belongs to a class. In this event space, a document is represented as a binary metrix. The rows are the docoments and the columns are the vocabulary. The entries are binary, 1 if a term occurs in the document or 0 if it doesn't occur.

In order to compute estimates for p(w|c) for all words using multiple-bernoulli model, I used a youtube classification data set to learn if a word is spam and not spam. This data set is saved in *Youtube02-KatyPerry.csv* I calculated the probability of a random word belonging to this classifier using the maxumum likelihood estimate which is :
P(w|c) = $\frac{df_{w,c}}{Nc}$. This is demontrated with *calcMultBernoulli(w, c, trainingSet)* in Listing 3.

Youtube Data Snippet

```
Content                Class
I really like this song.ïż£ <> 0
my son love so muchïż£ <> 0
Y LOVE YOUïż£ <> 0
follow me on instagram bigboss286ïż£ <> 1
why the elephant have a broken hornïż£ <> 0
Free itunes $25 giftcard codes: http://shhort.com/a?r=OOCnjqU2bïż£ <> 1
Where did she find all that make up in a freakin jungle?!ïż£ <> 0
so cute that monkey *-*! ïż£ <> 0
Nice songïż£ <> 0
subscribe please ïż£ <> 1
```

Multinomial Model

Multinomial is a generalization of multiple-bernoulli. This model takes into consideration the importance term frequence feature has in retrieval and classification. The goal of this model is to estimate the probability that a term occurs in a document. This model helps to determine the weight of a document since it considers the number of times each word occurs in the document.

In order to compute the probability that a term belongs to a class, I used the maximum likelihood estimate for the multinomial model: $P(w|c) = \frac{tf_{w,c}}{|c|}$. This is demonstrated in *calcMultinomial(w, c, trainingSet)* in Listing 3. I used both distributions to compute estimate for words.

```
Multiple-Bernoulli
  song
    P(SPAM) =  0.05714285714285714
    P(NOT) =  0.21142857142857144

Multinomial
    P(SPAM) =  0.0034982508745627187
    P(NOT) =  0.019770114942528734


Multiple-Bernoulli
  buy
    P(SPAM) =  0.005714285714285714
    P(NOT) =  0.0

Multinomial
    P(SPAM) =  0.0002498750624687656
    P(NOT) =  0.0



Multiple-Bernoulli
  world
    P(SPAM) =  0.017142857142857144
    P(NOT) =  0.03428571428571429

Multinomial
    P(SPAM) =  0.0007496251874062968
    P(NOT) =  0.002758620689655172


 Multiple-Bernoulli
  video
    P(SPAM) =  0.10857142857142857
    P(NOT) =  0.14285714285714285

Multinomial
    P(SPAM) =  0.004747626186906547
    P(NOT) =  0.012413793103448275
```

According to Multiple-Bernoulli, the probability that the word "buy" is SPAM is 0.005714285714285714 and the probability that it is NOT SPAM is 0.0. Since the probability of SPAM is higher, the term "buy" is SPAM.

According to Multinomial the probability that the word "buy" is SPAM is 0.0002498750624687656 and the probability that it is NOT SPAM is 0.0. Since the probability of SPAM is higher, the term "buy" is SPAM

According to Multiple-Bernoulli, the probability that the word "song" is SPAM is 0.05714285714285714 and the probability that it is NOT SPAM is 0.21142857142857144. Since the probability of NOT SPAM is higher, the term "song" is NOT SPAM.

According to Multinomial the probability that the word "song" is SPAM is 0.0034982508745627187 and the probability that it is NOT SPAM is 0.019770114942528734. Since the probability of NOT SPAM is higher, the term "song" is NOT SPAM

From my observation according to my out in *proOutput.txt*, Multiple-Bernoulli gives the term more weight because it has a higher probabiity. Although Multinomial differentiates a term from SPAM and NOT SPAM correctly, the weight is not high. Multiple-bernoulli is more confident because of the weight. This applies to all output result. I think this is the reseaon why the text *Search Engines Information Retrieval* states that multiple-bernoulli is good at estimating short documents, and my data set consists of short documents

Problem 7

8.5. Generate the mean average precision, recall-precision graph, average NDCG at 5 and 10, and precision at 10 for the entire CACM query set.

Solution

Listing 4: Mean Average Precision Code Snippet

```
import matplotlib.pyplot as plt
import numpy as np

#https://gist.github.com/bwhite/3726239
def r_precision(r):
    """Score is precision after all relevant documents have been retrieved
    Relevance is binary (nonzero is relevant).
    >>> r = [0, 0, 1]
    >>> r_precision(r)
    0.33333333333333331
    >>> r = [0, 1, 0]
    >>> r_precision(r)
    0.5
    >>> r = [1, 0, 0]
    >>> r_precision(r)
    1.0
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
    Returns:
        R Precision
    """
    r = np.asarray(r) != 0
    z = r.nonzero()[0]
    if not z.size:
```

```
        return 0.
    return np.mean(r[:z[-1] + 1])

#https://gist.github.com/bwhite/3726239

#https://gist.github.com/bwhite/3726239
def ndcg_at_k(r, k, method=0):
    """Score is normalized discounted cumulative gain (ndcg)
    Relevance is positive real values.  Can use binary
    as the previous methods.
    Example from
    http://www.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf
    >>> r = [3, 2, 3, 0, 0, 1, 2, 2, 3, 0]
    >>> ndcg_at_k(r, 1)
    1.0
    >>> r = [2, 1, 2, 0]
    >>> ndcg_at_k(r, 4)
    0.9203032077642922
    >>> ndcg_at_k(r, 4, method=1)
    0.96519546960144276
    >>> ndcg_at_k([0], 1)
    0.0
    >>> ndcg_at_k([1], 2)
    1.0
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]
    Returns:
        Normalized discounted cumulative gain
    """
    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max

def plotOriginalPoints(masterPntCol):

    fig = plt.figure()
    fig.suptitle('Recall-Precision graph at 5. No. of relevant docs for recall is 16',
     fontsize=14, fontweight='bold')

    ax = fig.add_subplot(111)
    fig.subplots_adjust(top=0.85)

    ax.set_xlabel('Recall')
    ax.set_ylabel('Precision')

    for lstOfPnts in masterPntCol:
        #Original points
```

```
            X = np.array(lstOfPnts)

            #plot points
            ax.plot(X[:,1], X[:,0])

      #set window size
      ax.axis([0, 1.02, 0, 1.02])

      #save plot
      plt.savefig('samplePlot.png')

def genPrecRecallGraph(query, K, data):

      twoDpnt = []

      allPr = calcPrecRecallAtK(query, K, data, 'list')
      for pr in allPr:
            twoDpnt.append([pr['precision'], pr['recall']])

      return twoDpnt

def calcPrecRecallAtK(query, K, data, returnType='single'):

      #16 relevant documents/query in CACM:
      #http://www.cs.sfu.ca/CourseCentral/456/jpei/web%20slides/L19%20-%20Evaluation.pdf
      foundFlag = False

      relCount = 0
      count = 0
      precAtK = 0
      recallAtK = 0
      precisionRecallLst = []
      for q in data:

            if( q[0] == query ):
                  foundFlag = True
                  relCount += 1
                  #print('g:', relCount, q)
            else:
                  if( foundFlag ):
                        #print('b')
                        pass

            if( foundFlag ):
                  count += 1

                  precAtK = relCount/float(count)
                  recallAtK = relCount/float(16)

                  #print('precision@' + str(count) + ' =', precAtK)
                  #print('recall@' + str(count) + ' =', recallAtK)

                  precisionRecallLst.append({'precision': precAtK, 'recall': recallAtK})
```

```
            if( count == K ):
                break

    if( returnType == 'single' ):
        return {'precision': precAtK, 'recall': recallAtK}
    else:
        return precisionRecallLst

def calcAvgPrecForQuery(query, K, data):

    foundFlag = False

    relCount = 0
    count = 0
    avgPrecision = 0
    for q in data:

        if( q[0] == query ):
            foundFlag = True
            relCount += 1
            #print('g:', relCount, q)
        else:
            if( foundFlag ):
                #print('b')
                pass

        if( foundFlag ):
            count += 1
            #print('precision@' + str(count) + ' =', relCount/count)
            avgPrecision += relCount/float(count)

        if( count == K ):
            break

    return avgPrecision/float(K)



def getRankingForQuery(query, K, data):

    count = 0
    foundFlag = False
    ranking = []
    for q in data:

        if( q[0] == query ):
            foundFlag = True
            ranking.append(1)
        else:
            if( foundFlag ):
                #start populating from first find
                ranking.append(0)
```

```
185
            if( foundFlag ):
                count += 1

            if( count == K ):
                break

        return ranking


lines = getInput()
MAP = 0
rPrecision = 0
#for all queries get MAP (5 and 10), average NDCG at 5 and 10, precision at 10
for query in range(1, 65):
        print 'query: ' + str(query)

        #mean average precision (MAP)
        MAP += calcAvgPrecForQuery(query, 10, lines)

        #precision at 10
        print '\tPrecision at 10: ' + str(calcPrecRecallAtK(query, 10, lines)['precision'])

        ranking = getRankingForQuery(query, 5, lines)
        ranking5 = getRankingForQuery(query, 5, lines)
        ranking10 = getRankingForQuery(query, 10, lines)

        #averate NDCG for 5 rankings and 10 rankings
        avg = ndcg_at_k(ranking5, 5)
        avg += ndcg_at_k(ranking10, 10)
        avg = avg/float(2)
        print '\taverage NDCG: ' + str(avg)

        rPrecision += r_precision(ranking)


        print ''

MAP = MAP/float(64)
print 'MAP: ' + str(MAP)


rPrecision = rPrecision/float(64)
print 'R-Precision: ' + str(rPrecision)


'''
masterPntCol = []
for i in range(1, 65):
        res = genPrecRecallGraph(i, 5, lines)

        if( len(res) > 0 ):
            masterPntCol.append( res )
```

```
plotOriginalPoints(masterPntCol)
'''
```

In order to compute the mean average precision, recall-precision graph, average NDCG at 5 and 10, and precision at 10 for the entire CACM query set, I took the following steps:

For Mean Average Precision, I used ranking window of size 10 in the CACM query set. This is demonsrated with *calcAvgPrecForQuery(query, K, data):* in Listing 4.

Mean Average Precision: 0.719438244048

For recall-precision graph, I used *genPrecRecallGraph(query, K, data)* in Listing 4 to generate the recall-precision graph after computing recall-precision with *calcPrecRecallAtK(query, K, data, returnType='single')* Figure 14 is the recall- precision graph at 10 and Figure 15 is the recall- precision graph at 5.

For average NDCG at 5 and 10, I used *ndcg_ at_ k(ranking5, 5)* and *ndcg_ at_ k(ranking10, 10)* to compute NDCG at 5 and NDCG at 10 and calculated the average of NDCG at 5 and 10 respectively. The result is in *map-prec-at-10-avg-ndcg-r-prect.txt.* This file also contains result of precision at 10 for all CACM query set.

The result snippet for precion at 10 for all CACM query set and average NDCG is given below:

```
  query: 1
  Precision at 10: 0.5
  average NDCG: 1.0

query: 2
  Precision at 10: 0.3
  average NDCG: 1.0

query: 3
  Precision at 10: 0.6
  average NDCG: 1.0

query: 4
  Precision at 10: 1.0
  average NDCG: 1.0

query: 5
  Precision at 10: 0.8
  average NDCG: 1.0
```

Average NDCG for all CACM query set is 1.0.

For Precision at 10 for the entire CACM query set, I used *(calcPrecRecallAtK(query, 10, lines)['precision'])* in Listing 4 to calculate precicion at 10 for the entire CACM query set and the result is in *map-prec-at-10-avg-ndcg-r-prect.txt.*

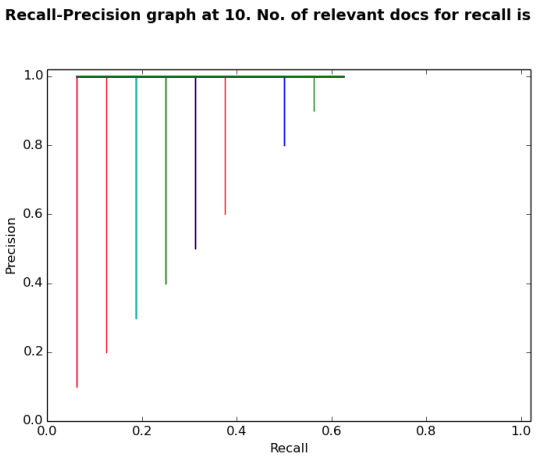

Figure 14: Recall-Precision at 10

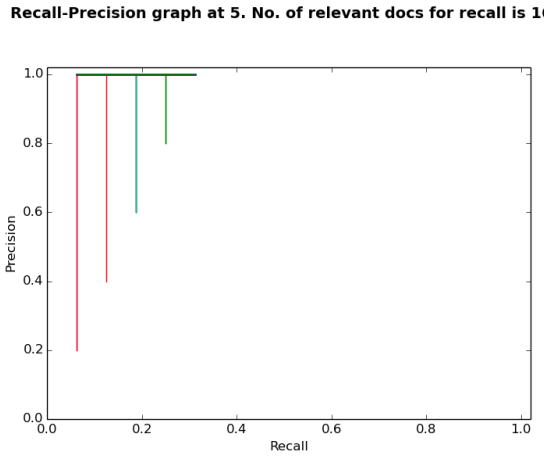

Figure 15: Recall-Precision at 5

Problem 8

8.7. Another measure that has been used in a number of evaluations is R-precision. This is defined as the precision at R documents, where R is the number of relevant documents for a query. It is used in situations where there is a large variation in the number of relevant documents per query. Calculate the average R-precision for the CACM query set and compare it to the other measures.

Solution
R-precision is the at R-documents, R is the number of relevant documents to a query. In order to compute R-precision for the CACM quert set, I used *r_precision(r)* in Listing 4

MAP: 0.719438244048
R-Precision: 0.8125
According to the result output, R-precision is highly correlated with the Mean Average Precision.

References

[1] Agglomerative Hierarchical Clustering. https://onlinecourses.science.psu.edu/stat505/node/143. Accessed: 2017-12-03.

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

[3] Githubgist. https://gist.github.com/bwhite/3726239. Accessed: 2017-12-03.

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

[5] Infornation Retrieval Evaluation. http://www.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf. Accessed: 2017-12-03.

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

[7] Multiclas SVMs. https://nlp.stanford.edu/IR-book/html/htmledition/multiclass-svms-1.html. Accessed: 2017-12-03.

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

[9] Quora. https://www.quora.com/What-are-industry-applications-of-the-K-nearest -neighbor-algorithm. Accessed: 2017-12-03.

[10] Youtube. https://www.youtube.com/watch?v=WVkD-jURBDg http://weka.sourceforge.net/doc.dev/weka/classifiers/fu Accessed: 2017-12-03.