**CS 7800 Information Retrieval**

**Text Mining**

**Abhishek Singh Thakur Nikhil Chavan**

**(U00933835) (U00969241)**

**Code Snippets**

**Feature Extraction:** *feature-extraction.py*

The code is implemented in feature-extraction.py where it takes 4 arguments which are filename of the dataset and the filenames which stores classlabels, features and its featured and training data for TF, IDF, and TFIDF each. The extractData() method extracts the data from all the 2000 documents contained in respective in 20 folders of mini\_newsgroup dataset. In featureProcessing() for each document it considers lines which have Subject and Lines as the words in the line after preprocessing the data i.e. after performing tokenization, stemming, lowercase and correction of data. The entire data is extracted and stored in document\_file for easy retrieval containing classlabel, documentName and extracted Data. The classlabels are for the 20 folders are divided into 6 groups which are stored in the class\_defination\_file and it is done in storeClassLabels(). In indexDoc()Using data in document\_file an Inverted matrix is generated and stored in inverted\_index file for easy calculation of TF, IDF and TFIDF values. The inverted index data is used to fetch all the terms and each term is allocated with the featured, and the featureId, term pair is stored in feature\_defination\_file which is implemented in featureFileCreation() method. Finally in createlibsvmFiles\_TF\_IDF\_TFIDF() method training data is generated in libsvm format i.e. < class label> <feature-id>:<feature-value> <feature-id>:<feature-value> ... in each of its line. However training\_data\_file.TF, training\_data\_file.IDF and training\_data\_file.TFIDFfiles contain <feature-value>  as the TF, IDF, and TFIDF respectively, values of the corresponding term in the document with its <feature-id> extracting from the feature\_defination\_file, the feature id and value pair is being sorted by its feature id before storing the information in the files

def tokenizer(text):

    text = re.sub("[^a-zA-Z]+", " ", text)

    tokens = nltk.tokenize.word\_tokenize(text)

    return tokens

def preprocessing\_txt(text):

    tokens = tokenizer(text)

    stemmer = EnglishStemmer()

    processedText = ""

    stopWords = nltk.corpus.stopwords.words('english')

    for token in tokens:

        token = token.lower()

        if token not in stopWords:

            processedText += stemmer.stem(token)

            processedText += " "

    return processedText

def test\_preprocessing\_txt(text):

    tokens = tokenizer(text)

    stemmer = EnglishStemmer()

    processedText = ""

    stopWordsList = []

    stopWords = nltk.corpus.stopwords.words('english')

    for token in tokens:

        token = token.lower()

        if token not in stopWords:

            processedText += stemmer.stem(token)

            processedText += " "

        else:

            stopWordsList.append(token)

    print('Removed Stopwords are: ' + str(stopWordsList))

    return processedText

# To find the normalized term location in the processed query

def wordPositions(word, processedText):

    pos = 1

    positions = []

    for data in processedText.split():

        if word == data:

            positions.append(pos)

        pos += 1

    return positions

def indexDoc(document):

      processedText = document['data']

      for word in processedText.split():

         position = wordPositions(word, processedText)

         if word in features.keys():

               if not int(document['documentId']) in list(features[word].keys()):

                  features[word][int(document['documentId'])] = position

         else:

               features[word] = {}

               features[word][int(document['documentId'])] = position

# Storing Class labels for the directories in the Dataset

def storeClassLabels():

   for dir in directory:

      labelClass.append(str(classLabelling[dir]) + ' ' + dir)

   with open(classFileName, 'w') as data:

      for item in labelClass:

         data.write("%s\n" % item)

   print('class\_defination\_file.TF file is generated')

# Finding and storing the inverted Index for the above all the datasets

def featureProcessing():

   print('Feature Id processing Started')

   for doc in document\_JSON:

      indexDoc(doc)

   with open('inverted\_index\_file','w') as invertedIndex:

      json.dump(features, invertedIndex)

# Storing the (FeatureId,Term) Pair in feature defination file

def featureFileCreation():

   count = 0

   featureIdTerm = ''

   print('Feature Id and Term pair processing started')

   for term in features.keys():

      count += 1

      featureIdTerm += str(count) + ' ' + str(term) + '\n'

   with open(featureDefinationFile,'w') as featureFile:

         featureFile.write(featureIdTerm)

   print('feature\_definition\_file file is generated')

# Finding TF, IDF, and TF-IDF values

def createlibsvmFiles\_TF\_IDF\_TFIDF():

   featureKeys = list(features.keys()) #===== Use featureKeys.index('term') to get the featureId

   totalDocs = len(document\_JSON)

   tf\_classLabel = ''

   idf\_classLabel = ''

   tf\_idf\_classLabel = ''

   training\_tf\_classLabel = ''

   training\_idf\_classLabel = ''

   training\_tf\_idf\_classLabel = ''

   print('Calculating and Storing of TF, IDF and TFIDF for each term has started')

   for doc in document\_JSON:

      totalData = doc['size']

      tf\_classLabel = ''

      idf\_classLabel = ''

      tf\_idf\_classLabel = ''

      orderFreq = {}

      for text in doc['data'].split(' '):

         if len(text) > 0:

            freq = len(features[text][int(doc['documentId'])])

            featureId = ' ' + str(featureKeys.index(text)  + 1) + ':'

            tf = float(freq/totalData)

            idf = float( 1 + math.log10(totalDocs/len(features[text].keys())))

            tfidf = float (tf \* idf)

            orderFreq[featureKeys.index(text)] = featureId + ',' + str(freq) + ',' + str(idf) + ',' + str(tfidf)

      for id in sorted(orderFreq.keys()):

         dataFreq = orderFreq[id].split(',')

         tf\_classLabel += dataFreq[0] + dataFreq[1]

         idf\_classLabel += dataFreq[0] + dataFreq[2]

         tf\_idf\_classLabel += dataFreq[0] + dataFreq[3]

      training\_tf\_classLabel += str(doc['classLabel']) + str(tf\_classLabel) + '\n'

      with open(tfFileName,'w') as tfdoc:

         tfdoc.write(training\_tf\_classLabel)

      training\_idf\_classLabel += str(doc['classLabel']) + str(idf\_classLabel) + '\n'

      with open(idfFileName,'w') as idfdoc:

         idfdoc.write(training\_idf\_classLabel)

      training\_tf\_idf\_classLabel += str(doc['classLabel']) + str(tf\_idf\_classLabel) + '\n'

      with open(tfidfFileName,'w') as tfidfdoc:

         tfidfdoc.write(training\_tf\_idf\_classLabel)

   print('training\_data\_file.TF file is generated')

   print('training\_data\_file.IDF file is generated')

   print('training\_data\_file.TFIDF file is generated')

def extractData():

   insertLines = False

   for dir in os.listdir(dataSetLocation):

      directory.append(dir)

      newdir = os.path.join(dataSetLocation, dir)

      for filename in os.listdir(newdir):

         with open(os.path.join(newdir, filename), 'r') as f:

            data = ''

            for line in f:

               if 'Subject:' in line:

                  data = data + line.replace('Subject: ','').replace('Re: ', '').replace('\n', ' ')

               elif 'Lines: ' in line:

                  insertLines = True

               elif '--' in line:

                  insertLines = False

                  break

               elif insertLines and (len(line) > 1):

                  data = data + line.split('\n')[0] + ' '

            data = preprocessing\_txt(data)

            if len(data) > 0:

               documentList.append(InvertedDocs(dir, filename, data))

      print('Number of Directories processed for feature Selection are  ' + str(len(directory)))

**Classification:** *classification.py*

In classification.py the 4 classifiers used in this lab are Multinomial Naive Bayes classifier, Bernoulli classifier, KNeighbors classifier, Support Vector-machine Classifiers. The dataset upon which these classifiers are tested is considered from training\_data\_file.TF, training\_data\_file.IDF and training\_data\_file.TFIDFfiles. TF data is used for Multinomial Naive Bayes classifier, IDF data is used for Bernoulli classifier and TFIDF data is used for KNeighbors and SVC classifiers. Using cross\_validation\_scores() method which calculates Accuracy of the classifiers upon the respective training data scores are generated for each classifier with 3 metrics i.e. f1\_macro, precision\_macro, recall\_macros. The scores contain Mean and Standard Deviation values which are plotted for the analysis

from sklearn.datasets import load\_svmlight\_file

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors  import KNeighborsClassifier

from sklearn.svm  import SVC

from sklearn.model\_selection import cross\_val\_score

import warnings

tf\_X, tf\_Y = load\_svmlight\_file('training\_data\_file.TF')

idf\_X, idf\_Y = load\_svmlight\_file('training\_data\_file.IDF')

tfidf\_X, tfidf\_Y = load\_svmlight\_file('training\_data\_file.TFIDF')

tf\_X\_train, tf\_X\_test, tf\_Y\_train, tf\_Y\_test = train\_test\_split(tf\_X, tf\_Y, test\_size=0.4, random\_state=0)

idf\_X\_train, idf\_X\_test, idf\_Y\_train, idf\_Y\_test = train\_test\_split(idf\_X, idf\_Y, test\_size=0.4, random\_state=0)

tfidf\_X\_train, tfidf\_X\_test, tfidf\_Y\_train, tfidf\_Y\_test = train\_test\_split(tfidf\_X, tfidf\_Y, test\_size=0.4, random\_state=0)

model\_MNB = MultinomialNB() # TF

model\_BNB = BernoulliNB(alpha=0.001) #IDF

model\_KNC = KNeighborsClassifier(weights='distance') #TFIDF

model\_SVC = SVC(gamma='scale') #TFIDF

modelClassifiers = [model\_MNB, model\_BNB, model\_KNC, model\_SVC]

scoringMethod = ['f1\_macro','precision\_macro', 'recall\_macro']

for model in modelClassifiers:

    warnings.filterwarnings("ignore")

    classfierName = ''

    if model == model\_MNB:

        classfierName = 'MultinomialNB'

        train\_data = tf\_X

        test\_data = tf\_Y

    elif model == model\_BNB:

        classfierName = 'BernoulliNB'

        train\_data = idf\_X

        test\_data = idf\_Y

    elif model == model\_KNC:

        classfierName = 'KNeighborsClassifier'

        train\_data = tfidf\_X

        test\_data = tfidf\_Y

    else:

        classfierName = 'SVC'

        train\_data = tfidf\_X

        test\_data = tfidf\_Y

    print('======= ' + classfierName + ' =======')

    for crossValidation in scoringMethod:

        scores = cross\_val\_score(model, train\_data, test\_data, cv=5, scoring=crossValidation)

        print(crossValidation + "accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

**Feature Selection:** *feature-selection.py*

In feature-selection.py the training data with reduced features are generated using SelectKBest algorithms which selects K best features and it is done using Chi2 and MI values. For some K sized feature data 2 training data are produced using Chi2 and MI metric with SelectKBest. The classifier models are generated for the 4 classifiers using the 2 generated K sized feature set data. Accordingly, in getScores() method cross-validation scores are generated for 4 classifiers, and aginst each metric i.e Chi2 and MI respective scores are plotted on graphs for 4 classifier score results and are analyzed to check the performance.

from sklearn.datasets import load\_svmlight\_file

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors  import KNeighborsClassifier

from sklearn.svm  import SVC

from sklearn.model\_selection import cross\_val\_score

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2, mutual\_info\_classif

import matplotlib.pyplot as plt

import warnings

import time

start = time.time()

training\_tf, test\_tf = load\_svmlight\_file('training\_data\_file.TF')

training\_idf, test\_idf = load\_svmlight\_file('training\_data\_file.IDF')

training\_tfidf, test\_tfidf = load\_svmlight\_file('training\_data\_file.TFIDF')

print(training\_tf.shape)

topFeatures = [100, 200, 800, 2400, 6400, 8000, 10000, 15000, 19999]

model\_MNB = MultinomialNB() # TF

model\_BNB = BernoulliNB(alpha=0.001) #IDF

model\_KNC = KNeighborsClassifier(weights='distance') #TFIDF

model\_SVC = SVC(gamma='scale') #TFIDF

modelClassifiers = [model\_MNB, model\_BNB, model\_KNC, model\_SVC]

tests = ['chi2', 'mic']

scoresValidation = {}

chi2\_MNB = []

chi2\_BNB = []

chi2\_KNC = []

chi2\_SVC = []

mic\_MNB = []

mic\_BNB = []

mic\_KNC = []

mic\_SVC = []

def getScores(featureSize):

    for i in tests:

        scoresValidation[top][i] = {}

        warnings.filterwarnings("ignore")

        for model in modelClassifiers:

            classfierName = ''

            if model == model\_MNB:

                classfierName = 'MultinomialNB'

                if i == 'chi2':

                    train\_data = training\_tf\_chi2

                else:

                    train\_data = training\_tf\_mic

                test\_data = test\_tf

            elif model == model\_BNB:

                classfierName = 'BernoulliNB'

                if i == 'chi2':

                    train\_data = training\_idf\_chi2

                else:

                    train\_data = training\_idf\_mic

                test\_data = test\_idf

            elif model == model\_KNC:

                classfierName = 'KNeighborsClassifier'

                if i == 'chi2':

                    train\_data = training\_tfidf\_chi2

                else:

                    train\_data = training\_tfidf\_mic

                test\_data = test\_tfidf

            else:

                classfierName = 'SVC'

                if i == 'chi2':

                    train\_data = training\_tfidf\_chi2

                else:

                    train\_data = training\_tfidf\_mic

                test\_data = test\_tfidf

            scores = cross\_val\_score(model, train\_data, test\_data, cv=5, scoring='f1\_macro')

            scoresValidation[top][i][classfierName] = scores.mean()

for top in topFeatures:

    scoresValidation[top] = {}

    training\_tf\_chi2 = SelectKBest(chi2, k=top).fit\_transform(training\_tf, test\_tf)

    training\_tf\_mic = SelectKBest(mutual\_info\_classif, k=top).fit\_transform(training\_tf, test\_tf)

    training\_idf\_chi2 = SelectKBest(chi2, k=top).fit\_transform(training\_idf, test\_idf)

    training\_idf\_mic = SelectKBest(mutual\_info\_classif, k=top).fit\_transform(training\_idf, test\_idf)

    training\_tfidf\_chi2 = SelectKBest(chi2, k=top).fit\_transform(training\_tfidf, test\_tfidf)

    training\_tfidf\_mic = SelectKBest(mutual\_info\_classif, k=top).fit\_transform(training\_tfidf, test\_tfidf)

    getScores(top)

    print(top)

for feature in scoresValidation.keys():

    for selection in scoresValidation[feature].keys():

        for model in scoresValidation[feature][selection].keys():

            value = scoresValidation[feature][selection][model]

            if selection == 'chi2':

                if model == 'MultinomialNB':

                    chi2\_MNB.append(value)

                elif model == 'BernoulliNB':

                    chi2\_BNB.append(value)

                elif model == 'KNeighborsClassifier':

                    chi2\_KNC.append(value)

                elif model == 'SVC':

                    chi2\_SVC.append(value)

            elif selection == 'mic':

                if model == 'MultinomialNB':

                    mic\_MNB.append(value)

                elif model == 'BernoulliNB':

                    mic\_BNB.append(value)

                elif model == 'KNeighborsClassifier':

                    mic\_KNC.append(value)

                elif model == 'SVC':

                    mic\_SVC.append(value)

print('Number of Features:  ' + str(topFeatures))

print('Chi2 Cross Validation Scores')

print('MultinomialNB:       ' + str(chi2\_MNB))

print('BernoulliNB:         ' + str(chi2\_BNB))

print('KNeighborsClassifier:' + str(chi2\_KNC))

print('SVC:                 ' + str(chi2\_SVC))

print()

print('Mutual Information Cross Validation Scores')

print('MultinomialNB:       ' + str(mic\_MNB))

print('BernoulliNB:         ' + str(mic\_BNB))

print('KNeighborsClassifier:' + str(mic\_KNC))

print('SVC:                 ' + str(mic\_SVC))

plt.xlabel('K feature Size')

plt.ylabel('Scores')

plt.subplot(2,1,1)

plt.title('Chi2 Cross Validation Scores')

plt.plot(topFeatures,chi2\_MNB, label='MultinomialNB')

plt.plot(topFeatures,chi2\_BNB, label='BernoulliNB')

plt.plot(topFeatures,chi2\_KNC, label='KNeighbors')

plt.plot(topFeatures,chi2\_SVC, label='SVC')

plt.legend(loc='best', shadow=True, fontsize=6)

plt.tight\_layout()

plt.subplot(2,1,2)

plt.title('Mic Cross Validation Scores')

plt.plot(topFeatures,mic\_MNB, label='MultinomialNB')

plt.plot(topFeatures,mic\_BNB, label='BernoulliNB')

plt.plot(topFeatures,mic\_KNC, label='KNeighbors')

plt.plot(topFeatures,mic\_SVC, label='SVC')

plt.legend(loc='best', shadow=True, fontsize=6)

plt.tight\_layout()

filename = 'chi2mic' + str(len(topFeatures))+'.png'

plt.savefig(filename)

print('Saved the plot in the' + filename)

elapsed\_time\_fl = (time.time() - start)

print(elapsed\_time\_fl)

**Clustering:** *clustering.py*

The clustering of the documents is performed with TFIDF files in clustering.py and it is done using KMeans and Agglomerative Clustering Methods. For K sized feature set extracted using SelectKBest using Chi2, the clustering methods are used upon with the range of 2 to 25. For each range of values, Silhouette and NMI scores are plotted against the KMeans and Agglomerative Clustering methods to check the performance of the clustering methods and its behaviour to validated which is optimal and why.

from sklearn.datasets import load\_svmlight\_file

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2, mutual\_info\_classif

import matplotlib.pyplot as plt

import warnings

import time

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn import metrics

start = time.time()

training\_tfidf, test\_tfidf = load\_svmlight\_file('training\_data\_file.TFIDF')

featureSize = 20000

training\_tfidf\_chi2 = SelectKBest(chi2, k=featureSize).fit\_transform(training\_tfidf, test\_tfidf)

clusterSize = []

kmeans\_model\_silhouette\_score = []

kmeans\_model\_normalized\_mutual\_info\_score = []

single\_linkage\_model\_silhouette\_score = []

single\_linkage\_model\_normalized\_mutual\_info\_score = []

for cluster in range(2,25):

    clusterSize.append(cluster)

    kmeans\_model = KMeans(n\_clusters=cluster).fit(training\_tfidf\_chi2)

    single\_linkage\_model = AgglomerativeClustering(n\_clusters=cluster, linkage='ward').fit(training\_tfidf\_chi2.toarray())

    warnings.filterwarnings("ignore")

    kmeans\_model\_silhouette\_score.append(metrics.silhouette\_score(training\_tfidf\_chi2, kmeans\_model.labels\_, metric='euclidean'))

    kmeans\_model\_normalized\_mutual\_info\_score.append(metrics.normalized\_mutual\_info\_score(test\_tfidf, kmeans\_model.labels\_, average\_method='arithmetic'))

    single\_linkage\_model\_silhouette\_score.append(metrics.silhouette\_score(training\_tfidf\_chi2, single\_linkage\_model.labels\_, metric='euclidean'))

    single\_linkage\_model\_normalized\_mutual\_info\_score.append(metrics.normalized\_mutual\_info\_score(test\_tfidf, single\_linkage\_model.labels\_, average\_method='arithmetic'))

print()

print('ClusterSize Range for Feature Size ' + str(featureSize) +' is:' + str(clusterSize))

print('Kmeans Model Silhouette Score:                    ' + str(kmeans\_model\_silhouette\_score))

print('Kmeans Model Normalized Mutual Info Score:        ' + str(kmeans\_model\_normalized\_mutual\_info\_score))

print('Single Linkage Model Silhouette Score:            ' + str(single\_linkage\_model\_silhouette\_score))

print('Single Linkage Model Normalized Mutual Info Score:' + str(single\_linkage\_model\_normalized\_mutual\_info\_score))

plt.xlabel('Cluster Size')

plt.ylabel('Method')

plt.subplot(2,1,1)

plt.title('Silhouette Measure')

plt.plot(clusterSize,kmeans\_model\_silhouette\_score, label='KMeans')

plt.plot(clusterSize,single\_linkage\_model\_silhouette\_score, label='AgglomerativeClustering')

plt.legend(loc='best', shadow=True)

plt.tight\_layout()

plt.subplot(2,1,2)

plt.title('Normalized Mutual Info')

plt.plot(clusterSize,kmeans\_model\_normalized\_mutual\_info\_score, label='KMeans')

plt.plot(clusterSize,single\_linkage\_model\_normalized\_mutual\_info\_score, label='AgglomerativeClustering')

plt.tight\_layout()

plt.legend(loc='best', shadow=True)

filename = 'clustering' + str(featureSize)+'.png'

plt.savefig(filename)

print('Saved the plot in the' + filename)

elapsed\_time\_fl = (time.time() - start)

print(elapsed\_time\_fl)