INFORMATION RETRIEVAL

PROJECT TITLE: Text Mining

Project Members:

Srivalli Kompella (U00965253)
Sai sri Malempati (U00973516)

SOURCE CODE

feature-extract.py

```
import math
import os
import sys

import news
import time
import datetime
import nltk
from collections import defaultdict
from nltk.stem.snowball import EnglishStemmer  # Assuming we're working with
English


class Index:
    """ Inverted index datastructure """

    def __init__(self, tokenizer, stemmer=None, stopwords=None):
        """
        tokenizer   -- NLTK compatible tokenizer function
        stemmer     -- NLTK compatible stemmer
        stopwords   -- list of ignored words
        """
        self.tokenizer = tokenizer
        self.stemmer = stemmer
        self.index = defaultdict(list)
        self.documents = {}
        self.unique_id = 0
        if not stopwords:
            self.stopwords = set()
        else:
            self.stopwords = set(stopwords)

    def lookup(self, word):
        """
        Lookup a word in the index
        """
        word = word.lower()
        if self.stemmer:
            word = self.stemmer.stem(word)

        return [self.documents.get(id, None) for id in self.index.get(word)]

    def add(self, document):
        """
        Add a document string to the index
        """
        content = document.title+ document.body
```

```
        for token in [t.lower() for t in nltk.word_tokenize(content)]:
            if token in self.stopwords:
                continue

            if self.stemmer:
                token = self.stemmer.stem(token)

            if self.unique_id not in self.index[token]:
                self.index[token].append(document.docID)

        self.documents[self.unique_id] = document
        self.unique_id += 1

class feature_extract():

    def __init__(self):
        self.featureLookup={}
        self.class_map_dic={}
        self.index = Index(nltk.word_tokenize,
                           EnglishStemmer(),
                           nltk.corpus.stopwords.words('english'))
    def find_class(self,x):
        x=x.split("_",1)[1]
        for keys,values in self.class_map_dic.items():
            if x in values:
                return keys

    def calculate_idf(self,termfreq):
        idf_value = self.index.unique_id / termfreq
        idf = abs(math.log(idf_value, 10))
        return idf
    def remove_dupes(self,orglist):
        duplist=[]
        for x in orglist:
            if x not in duplist:
                duplist.append(x)
        return duplist
    def load_training_data_file_TF(self,training_file):
        #f=open(training_file.type,"w")
        tfdoclist={}
        idfdoclist={}
        tfidfdoclist={}
        for term in self.index.index.keys():
          for doc in self.remove_dupes(self.index.index[term]):
              termfreq=self.index.index[term].count(doc)
              idfval=self.calculate_idf(termfreq)
              tfidf=termfreq*idfval
              term_id=self.featureLookup[term]
              if(doc in tfdoclist.keys()):
                  tfdoclist.get(doc)[term_id]=termfreq
              else:
                  tfdoclist[doc]={term_id:termfreq}
              if(doc in idfdoclist.keys()):
```

```
                idfdoclist.get(doc)[term_id]=idfval
            else:
                idfdoclist[doc]={term_id:idfval}
            if(doc in tfidfdoclist.keys()):
                tfidfdoclist.get(doc)[term_id]=tfidf
            else:
                tfidfdoclist[doc]={term_id:tfidf}
        #write termfrequency file
        print("Loading Term Frequency Training data file..")
        f=open(training_file+".TF","w")
        for key,value in tfdoclist.items():
            docstring=self.find_class(key)
            docstring+=' '
            docstring+=str(value).replace(',','').replace(':
',':').split('{')[1].split('}')[0]
            docstring+='\n'
            f.write(docstring)
        f.close()
        print("Succesfully Loaded Term Frequency Training data file")
        print("Loading IDF Training data file.....")
        #write IDF file
        f=open(training_file+".IDF","w")
        for key,value in idfdoclist.items():
            docstring=self.find_class(key)
            docstring+=' '
            docstring+=str(value).replace(',',' ').replace(':
',':').split('{')[1].split('}')[0]
            docstring+='\n'
            f.write(docstring)
        f.close()
        print("Succesfully IDF Training data file")
        print("Loading TF-IDF Training data file.....")
        #write TFIDF file
        f=open(training_file+".TFIDF","w")
        for key,value in tfidfdoclist.items():
            docstring=self.find_class(key)
            docstring+=' '
            docstring+=str(value).replace(',',' ').replace(':
',':').split('{')[1].split('}')[0]
            docstring+='\n'
            f.write(docstring)
        f.close()
        print("Succesfully TF-IDF Training data file")
    def load_feature_definition_file(self,feature_file):
        f=open(feature_file,'w')
        #define the feature_id & initiate to 1
        ftr_id=1
        for trm in self.index.index.keys():
            f.write('('+str(ftr_id)+','+trm+')\n')
            self.featureLookup[trm] = ftr_id
            ftr_id=ftr_id+1
        f.close()
```

```
    def load_class_definition_file(self,class_file):
        f = open(class_file, "w")
        #declare the class mapping
        self.class_map_dic = {
            '1': ['comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
                  'comp.windows.x'], '2': ['rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey'],
            '3': ['sci.crypt', 'sci.electronics', 'sci.med', 'sci.space'],
'4': ['misc.forsale'],
            '5': ['talk.politics.misc', 'talk.politics.guns',
'talk.politics.mideast'],
            '6': ['talk.religion.misc', 'alt.atheism',
'soc.religion.christian']}
        for key, value in self.class_map_dic.items():
            for x in value:
                f.write('(' + x + ',' + key + ')\n')
        f.close()

    def
feature_extraction(self,newsdir,feature_file,class_file,training_file):
        #Load newdirectory files
        inputdocument= news.read_news(newsdir)
        print("Generating Index for documents (it might take approximately 57
seconds)....")
        #Perform the Indexing
        for doc in inputdocument.docs:
          self.index.add(doc)
        #Load class definition,feature definition,training files
        print("Loading Class definition file..")
        self.load_class_definition_file(class_file)
        print("Loading feature definition file")
        self.load_feature_definition_file(feature_file)
        print("Loading Training File")
        self.load_training_data_file_TF(training_file)

def test():
    #check whether read all files from directory given
    newsdoc=news.read_news("mini_newsgroups")
    assert len(newsdoc.docs) == 2000
    print("Test Case :: Loading newsdirectory-2k Documents PASSED")
    #cehck whether Index created after stop words removed and done stemmed
for a document
    doc=newsdoc.docs[1]
    print("******* Document considered for Index testing::")
    print(doc.title+ doc.body)
    index = Index(nltk.word_tokenize,
                  EnglishStemmer(),
                  nltk.corpus.stopwords.words('english'))
    index.add(doc)
    indexstr=''
    for x in index.index.keys():
        indexstr+=x+' '
```

```
    print('***** Document after removal of stopwords and stemming *******')
    print(indexstr)
    print("Test Case :: Index created passed")
    #check whether
feature_definition_file,class_definition_file,training_data_file created.
    if(os.path.exists('feature_definition_file')):
        print('Test Case :: Loading feature_definition_file passed')
    if(os.path.exists('class_definition_file')):
        print('Test Case :: Loading class_definition_file passed')
    from sklearn.datasets import load_svmlight_file
    feature_vectors, targets = load_svmlight_file("training_data_file.TF")
    print("Test Case :: Loading training_data_file.TF passed")
    from sklearn.datasets import load_svmlight_file
    feature_vectors, targets = load_svmlight_file("training_data_file.IDF")
    print("Test Case :: Loading training_data_file.IDF passed")
    from sklearn.datasets import load_svmlight_file
    feature_vectors, targets =
load_svmlight_file("training_data_file.TFIDF")
    print("Test Case :: Loading training_data_file.TFIDF passed")

if __name__ == '__main__':
    feature_obj=feature_extract()

#feature_obj.feature_extraction("mini_newsgroups","feature_definition_file",
"class_definition_file","training_data_file")
    feature_obj.feature_extraction(str(sys.argv[1]),
str(sys.argv[2]),str(sys.argv[3]),str(sys.argv[4]))
```

test.py

```
feature_extarctobj=__import__('feature-extract')

print("******* Running Test Cases of feature extraction ***********")
feature_extarctobj.test()
```

news.py

```
import os
from os import walk
from doc import Document

class read_news():
    def __init__(self,newsdir):
        self.docs=[]
        filespaths = []
        #get path of newsdirectory
        pathfile=os.getcwd() + "\\" + newsdir #"mini_newsgroups"
```

```
        #loop in the subdirectoy and read the files
        for (dirpath, dirnames, filenames) in walk(pathfile):
             for x in dirnames:
               for (subdirpath, subdirnames, files) in
walk(pathfile+"\\"+x):
                   for f in files:
                       self.readfiles(subdirpath,f,x)
    def readfiles(self,dirname,filename,subdir):
        #read file subject and last xx lines
        filepath=dirname+"\\"+filename
        cf = open(filepath)
        docid = filename+"_"+subdir
        number_of_lines=0
        title = ''
        body = ''
        linemessage = ''
        startlines=False
        for line in cf:
            if 'Subject:' in line:
                title = line[9:].strip()  # got title
            elif 'Lines:' in line:
                try:
                 number_of_lines=int(line[6:])
                except Exception as e:
                    if 'dog' in str(e):
                        number_of_lines=24
                startlines = True
                line = ''
            if startlines:
                #last_line = cf.readlines()[-number_of_lines:]
                last_line=[i.replace('\n','') for i in cf.readlines()[-
number_of_lines:]]
                linemessage= ''.join(last_line)
        body = linemessage;
        #convert file to document format
        self.docs.append(Document(docid, title,body))
```

doc.py

```
'''

The document class, containing information from the raw document and
possibly other tasks

The collection class holds a set of docuemnts, indexed by docID

'''

class Document:
    def __init__(self, docid, title, body):
```

```
        self.docID = docid
        self.title = title
        self.body = body


    # add more methods if needed


class Collection:
    ''' a collection of documents'''

    def __init__(self):
        self.docs = {} # documents are indexed by docID

    def find(self, docID):
        ''' return a document object'''
        if self.docs.has_key(docID):
            return self.docs[docID]
        else:
            return None

    # more methods if needed
```

classification.py

```
import warnings
import datetime

from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

warnings.filterwarnings("ignore")
#run for multinomial navie Bayes classifier
print("*******multinomial Naive Bayes classifier********")
clf = MultinomialNB()
#load Term frquency file as features and targets for multinomial
feature_vectors, targets = load_svmlight_file("training_data_file.TF")
#run the cross_validation method  with f1_macro scoring and get the scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='f1_macro')
print("f1_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
#run the cross_validation method  with precision_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='precision_macro')
print("precision_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
2))
```

```
#run the cross_validation method  with recall_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='recall_macro')
print("recall_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
print("\n")
#run for Naive Bayes classifier
print("********Naive Bayes classifier***************")
#load Term frequency file as features and targets for navie bayes
clf = BernoulliNB()
feature_vectors, targets = load_svmlight_file("training_data_file.IDF")
#run the cross_validation method  with f1_macro scoring and get the scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='f1_macro')
print("f1_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
#run the cross_validation method  with precision_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='precision_macro')
print("precision_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
2))
#run the cross_validation method  with recall_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='recall_macro')
print("recall_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
print("\n")
#run for k-nearest neighbors classifier
print("***************k-nearest neighbors classifier
**********************")
clf = KNeighborsClassifier()
#load Term frequency file as features and targets for k-nn
feature_vectors, targets = load_svmlight_file("training_data_file.TFIDF")
#run the cross_validation method  with f1_macro scoring and get the scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='f1_macro')
print("f1_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
#run the cross_validation method  with precision_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='precision_macro')
print("precision_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
2))
#run the cross_validation method  with recall_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='recall_macro')
print("recall_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
print("\n")
#run for C-Support Vector Classifier
print("****************C-Support Vector Classifier*****************")
clf = SVC()
#load Term frequency file as features and targets for svm bayes
```

```
feature_vectors, targets = load_svmlight_file("training_data_file.TFIDF")
#run the cross_validation method  with f1_macro scoring and get the scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='f1_macro')
print("f1_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
#run the cross_validation method  with precision_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='precision_macro')
print("precision_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
2))
#run the cross_validation method  with recall_macro scoring and get the
scores
scores = cross_val_score(clf, feature_vectors, targets, cv=5,
scoring='recall_macro')
print("recall_macro: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

feature_selection.py

```
import datetime

from matplotlib import pyplot
from sklearn.datasets import load_svmlight_file
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

multinominal_chi=[]
multinominal_mutal=[]
bernouli_chi=[]
bernouli_mutual=[]
knn_chi=[]
knn_mutual=[]
c_support_chi=[]
c_support_mutal=[]
#print(datetime.datetime.now())
#kvals=[500, 1500, 2500, 3500, 4500, 5500, 6500, 8500,10000]
print("Note:: This Programme run more than 20 minutes...")
#considered K values ranging from 500 to 2k
kvals=[*range(500,2000,200)]
print(kvals)
#iterate for all the values of k
for k_value in kvals:
  print("Processing feature_Selection on all algorithms for k-
value:",k_value)
  clf = MultinomialNB()
  #load training file TF
```

```
  X, y = load_svmlight_file("training_data_file.TF")
  #select K best feature using ch-square method
  X_new1 = SelectKBest(chi2, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new1, y, cv=5, scoring='f1_macro')
  multinominal_chi.append(scores.mean())
  # select K best feature using mutualinfo method
  X_new2 = SelectKBest(mutual_info_classif, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new2, y, cv=5, scoring='f1_macro')
  multinominal_mutal.append(scores.mean())

  clf = BernoulliNB()
  X, y = load_svmlight_file("training_data_file.IDF")
  # select K best feature using ch-square method
  X_new1 = SelectKBest(chi2, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new1, y, cv=5, scoring='f1_macro')
  bernouli_chi.append(scores.mean())
  # select K best feature using mutualinfo method
  X_new2 = SelectKBest(mutual_info_classif, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new2, y, cv=5, scoring='f1_macro')
  bernouli_mutual.append(scores.mean())

  clf = KNeighborsClassifier()
  X, y = load_svmlight_file("training_data_file.TFIDF")
  # select K best feature using ch-square method
  X_new1 = SelectKBest(chi2, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new1, y, cv=5, scoring='f1_macro')
  knn_chi.append(scores.mean())
  # select K best feature using mutualinfo method
  X_new2 = SelectKBest(mutual_info_classif, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new2, y, cv=5, scoring='f1_macro')
  knn_mutual.append(scores.mean())

  clf = SVC()
  X, y = load_svmlight_file("training_data_file.TFIDF")
  # select K best feature using ch-square method
  X_new1 = SelectKBest(chi2, k=k_value).fit_transform(X, y)
  #apply cross val score method with scoring of f1_macro
  scores = cross_val_score(clf, X_new1, y, cv=5, scoring='f1_macro')
  c_support_chi.append(scores.mean())
  # select K best feature using mutualinfo method
  X_new2 = SelectKBest(mutual_info_classif, k=k_value).fit_transform(X, y)
  scores = cross_val_score(clf, X_new2, y, cv=5, scoring='f1_macro')
  c_support_mutal.append(scores.mean())


#plot figure for Chi-square
pyplot.figure(1)
#pyplot.subplot(211)
pyplot.plot(kvals, multinominal_chi,label = "Multinomial Naive Bayes")
pyplot.plot(kvals, bernouli_chi, label = "Bernoulli Naive Bayes")
pyplot.plot(kvals, knn_chi, label = "KNN")
pyplot.plot(kvals, c_support_chi, label = "SVM")
pyplot.xlabel("K")
```

```
pyplot.ylabel("f1_macro")
pyplot.title("CHI Square")
pyplot.legend(loc = 'best')
#plot figure for mutualinformation
pyplot.figure(2)
pyplot.plot(kvals, multinominal_mutal,label = "Multinomial Naive Bayes")
pyplot.plot(kvals, bernouli_mutual, label = "Bernoulli Naive Bayes")
pyplot.plot(kvals, knn_mutual, label = "KNN")
pyplot.plot(kvals, c_support_mutal, label = "SVM")
pyplot.xlabel("K")
pyplot.ylabel("f1_macro")
pyplot.title("Mutual Information")
pyplot.legend(loc = 'best')
#print(datetime.datetime.now())
pyplot.show()
```

clustering.py

```
from matplotlib import pyplot
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn import metrics
from sklearn.datasets import load_svmlight_file
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif

feature_vectors, targets = load_svmlight_file("training_data_file.TFIDF")
print("Note:: This Programme runs for approximatley 5 minutes...")
#K clsuters range from 2 to 25
clust_list=[*range(2,26)]
print(clust_list)
silhouette_score_kmeans=[]
normalized_score_kmeans=[]
silhouette_score_agglormative=[]
normalized_score_agglormative=[]

#consider 100 best features for clustering
print("Selecting 100 best features.....")
X=SelectKBest(mutual_info_classif, k=100).fit_transform(feature_vectors,
targets).toarray()
#run for each cluster number
for n_clust in clust_list:
  print("Running for clusters:",n_clust)
  #apply kmeans clustering algorithm
  kmeans_model = KMeans(n_clusters=n_clust).fit(X)
  clustering_labels = kmeans_model.labels_
  #calculate sc score
  silhoutescore=metrics.silhouette_score(X, clustering_labels,
metric='euclidean')
  silhouette_score_kmeans.append(silhoutescore)
  #calculate NMI score
```

```
  normalized_scores=metrics.normalized_mutual_info_score(targets,
clustering_labels)
  normalized_score_kmeans.append(normalized_scores)

for n_clust in clust_list:
    #apply hierarchial clustering algorithm
    single_linkage_model = AgglomerativeClustering(n_clusters=n_clust,
linkage='ward').fit(X)
    clustering_labels=single_linkage_model.labels_
    #calculate sc score
    silhoutescore = metrics.silhouette_score(X, clustering_labels,
metric='euclidean')
    silhouette_score_agglormative.append(silhoutescore)
    #calculate NMI score
    normalized_scores = metrics.normalized_mutual_info_score(targets,
clustering_labels)
    normalized_score_agglormative.append(normalized_scores)

#plot figue for sc
plot1=pyplot.figure(1)
pyplot.plot(clust_list, silhouette_score_kmeans,label="KMeans")
pyplot.plot(clust_list, silhouette_score_agglormative,label="Hierarchical
clustering")
pyplot.title("Sihouette Coefficient Scores")
pyplot.xlabel("Number of Clusters")
pyplot.ylabel("The Measures")
pyplot.legend(loc='best')
#plot figure for NMI
plot2=pyplot.figure(2)
pyplot.plot(clust_list, normalized_score_kmeans,label="KMeans")
pyplot.plot(clust_list, silhouette_score_agglormative,label="Hierarchical
clustering")
pyplot.title("Normalized Mutual Information Scores")
pyplot.xlabel("Number of Clusters")
pyplot.ylabel("The Measures")
pyplot.legend(loc='best')
pyplot.show()
```