April 21, 2019

Manas Gaur, Amanuel Alambo
Information Retrieval, CS 7800
Project-2
Text Mining

The following references have been used in the development of the codes that follow:
https://nlpforhackers.io/building-a-simple-inverted-index-using-nltk/
https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
https://scikit-learn.org/stable/modules/cross_validation.html

feature-extract.py

```
'''
Authors: Manas Gaur, Amanuel Alambo
Instructor: Dr. keke Chen
feature extractor

'''

import nltk
from collections import defaultdict
from nltk.stem.snowball import EnglishStemmer
import math
from collections import deque
import os
import json
import subprocess
import sys

nltk.download('stopwords')
nltk.download('punkt')


#class Index used for indexing the entire newsgroup datasets
#source: https://nlpforhackers.io/building-a-simple-inverted-index-using-nltk/
class Index:
    """ Inverted index data structure """

    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    #can be used as doc id
        if not stopwords:
            self.stopwords = list()
        else:
            self.stopwords = list(stopwords)

    def lookup(self, word):
        """
        Looks up 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
        """
        #words=[word.lower() for word in words if word.isalpha()]   #added
on 0415
        for token in [t.lower() for t in nltk.word_tokenize(document)]:
            if not token.isalpha():
                continue

            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(self.__unique_id)
        self.documents[self.__unique_id] = document
        self.__unique_id += 1

    #method to return inverted index of the entire newsgroup datasets(i.e.,
term to list of doc-ids mappings)
    def indexed_docs(self):
        return self.index
        #return self.documents

    #method to return documents index(i.e., mappings of doc-id(0-1999) and
entire document content(subject+body))
    def indexed_docs_I(self):
        #return self.index
        return self.documents

doc_subdir = {}    #dictionary of doc-id(0-1999) as key and the
directory(subdirectory) the doc belongs to

#method to take the root directory path as parameter and parse each doc in
each newsgroup subdirectory
def subject_body_index(rootdir):
    #rootdir = 'mini_newsgroups'
    nDocs = 0  #to keep track of number of indexed docs

    #doc_subdir = {}    #added on 0416----dictionary of doc-id(0-1999) as
key and the directory(subdirectory) the doc belongs to
    sub_dir_count = 0  #added 0416----to get over counting the root
directory itself----to be used for libsvm
    doc_count = 0   #increments with every doc read from all
subdirectories(there are 2000 docs in total)
    for subdir, dirs, files in os.walk(rootdir):  #iteration through
directory 'mini_newsgroup'
        if sub_dir_count > 0:   #In the first round of iteration, the
mini_newsgroup directory itself get read---this is to avoid that
            for file in files:
                nDocs += 1
                #cf = open('mini_newsgroups/rec.autos/103806')
                cf = open(os.path.join(subdir, file),
encoding='iso-8859-1')
                #print(os.path.join(subdir, file))
                subject = ''
```

April 21, 2019

```
                body = ''

                doc = ''
                for line in cf:
                    try:
                        if 'Lines' in line:
                            n = int(line.strip().split(': ')[1])
                            #print(type(n))
                            body = deque(cf, maxlen=n)
                            body = ''.join(body)
                            #print(body)
                            #index.add(body)

                        if 'Subject' in line:
                            subject = line.strip().split(': ', 1)[1]
                            #print(subject)
                            #index.add(doc_subject)
                    except:
                        continue


                doc = subject + ' @' + body   #a combination of 'subject'
and 'body' make up the document to index separated by character '@'
                index.add(doc.strip())
                doc_subdir[doc_count] = subdir.split('/')[1]   #added on
0416---doc_count is in effect docid
                doc_count += 1
        sub_dir_count += 1

    #print(doc_subdir)

#method to generate feature definition file
def feature_defn_gen(feature_defn_file):
    feature_id = 0
    with open(feature_defn_file, 'w') as f:
        for k,v in index.indexed_docs().items():
            feature_id_term_map = (feature_id,k)
            f.write(str(feature_id_term_map))
            f.write('\n')
            feature_id += 1

#hard-coded class labels
```

April 21, 2019

```
class_1 =  '(comp.graphics, comp.os.ms-windows.misc,
comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x)'
class_2 = '(rec.autos, rec.motorcycles, rec.sport.baseball,
rec.sport.hockey)'
class_3 = '(sci.crypt, sci.electronics, sci.med, sci.space)'
class_4 = '(misc.forsale)'
class_5 = '(talk.politics.misc, talk.politics.guns, talk.politics.mideast)'
class_6 = '(talk.religion.misc, alt.atheism, soc.religion.christian)'
class_definition = defaultdict(list)

#method to generate class definition file
def class_defn_gen(class_defn_file, rootdir):
    #####################################################################
    #class definition file generation snippet
    sub_dir_count = 0
    with open(class_defn_file, 'w') as f:
        for subdir, dirs, files in os.walk(rootdir):
            if sub_dir_count > 0:
                if 'comp' in subdir.split('/')[1]:
                        tuple_map = subdir.split('/')[1],class_1
                        f.write(str(tuple_map))
                        f.write('\n')
                elif 'rec' in subdir.split('/')[1]:
                        tuple_map = subdir.split('/')[1],class_2
                        f.write(str(tuple_map))
                        f.write('\n')
                elif 'sci' in subdir.split('/')[1]:
                        tuple_map = subdir.split('/')[1],class_3
                        f.write(str(tuple_map))
                        f.write('\n')
                elif 'misc.forsale' in subdir.split('/')[1]:
                        tuple_map = subdir.split('/')[1],class_4
                        f.write(str(tuple_map))
                        f.write('\n')
                elif 'talk.politics' in subdir.split('/')[1]:
                        tuple_map = subdir.split('/')[1],class_5
                        f.write(str(tuple_map))
                        f.write('\n')
                else:

                        tuple_map = subdir.split('/')[1],class_6
                        f.write(str(tuple_map))
                        f.write('\n')

            sub_dir_count += 1


featureids_terms = [line.rstrip('\n') for line in
open('feature_definition_file')]
class_definition_file = [line.rstrip('\n') for line in
open('class_definition_file.csv')]  #added on 0416
class_definition_pairs = {}

#method to compute normalized term frequency of a term in a document
def tf_compute(doc, term):
    tf = round((doc.count(term))/float(len(doc.split(' '))),2)
    return tf

#method to generate TF based training dataset
def training_data_tf_gen(training_data_tf_based):
    #training data file generation snippet---tf-based(term frequency based
training dataset)

    #iterates through each newgroup to class mapping
    for pair in class_definition_file:
        key = pair.lstrip('(').rstrip(')').split(', ',1)[0].replace("'",'')
        value = pair.lstrip('(').rstrip(')').split(',
',1)[1].replace("'",'')
        class_definition_pairs[key] = value

    #print(class_definition_pairs)
    count = 0
    doc_list = defaultdict(list)

    with open(training_data_tf_based, 'w') as f:
        for k,v in index.indexed_docs_I().items():
            #print(v)
            news_group = doc_subdir[k]
            class_label = class_definition_pairs[news_group]   #extracts
class label given newsgroup name

            if class_label == class_1:
                class_label = 0
            elif class_label == class_2:
                class_label = 1
            elif class_label == class_3:
                class_label = 2
            elif class_label == class_4:
                class_label = 3
            elif class_label == class_5:
                class_label = 4
            elif class_label == class_6:
                class_label = 5

            f.write(str(class_label))
            f.write(' ')
            for feat_term in featureids_terms:
                #print(feat_term)
                feat_term = feat_term.replace('(','').replace(')','')
                feat,term = feat_term.split(', ')[0],feat_term.split(',
')[1]
                term = term.replace("'", '')
                if term in v:
                    #tf = round((v.count(term))/float(len(v.split(' '))),4)
#normalized term frequency of a term in a document
                    tf = tf_compute(v,term)    #normalized term frequency
of a term in a document

                    feat_tf_map = str(feat)+':'+str(tf)

                    f.write(feat_tf_map)
                    f.write(' ')
                    doc_list[class_label].append(feat_tf_map)
                    #doc_list[k].append(feat_tf_map)
            f.write('\n')

#method to return the number of files(documents) in a directory---used in
function 'nDocs_in_subdir'
def filecount(dir_name):
    # return the number of files in directory dir_name
    dir_name = 'mini_newsgroups/'+str(dir_name)  #subdirectory pathname
from current directory
    try:
        return len([f for f in os.listdir(dir_name) if
os.path.isfile(os.path.join(dir_name, f))])
    except:
        return None

#method to return the total number of documents a given document belongs to
def nDocs_in_subdir(docid):
    sub_dir = doc_subdir[docid]
    file_count = filecount(sub_dir)
    return file_count

#method to compute idf score of a term in a document
def idf_compute(nDocs, nDocs_term):
        ''' computes the inverted document frequency for a given term'''
        try:
            idf_score = math.log(nDocs, nDocs_term)
        except:
            idf_score = 1.0
        return round(idf_score,2)


#method to generate IDF based training dataset
def training_data_idf_gen(training_data_idf_based,inv_index):
    for pair in class_definition_file:
        key = pair.lstrip('(').rstrip(')').split(', ',1)[0].replace("'",'')
        value = pair.lstrip('(').rstrip(')').split(',
',1)[1].replace("'",'')
        class_definition_pairs[key] = value

    #print(class_definition_pairs)
    count = 0
    doc_list = defaultdict(list)

    #open output file for IDF based training data
    with open(training_data_idf_based, 'w') as f:
        for k,v in index.indexed_docs_I().items():
            #print(v)
            news_group = doc_subdir[k]
            class_label = class_definition_pairs[news_group]

            if class_label == class_1:
                class_label = 0
            elif class_label == class_2:
                class_label = 1
            elif class_label == class_3:
                class_label = 2
            elif class_label == class_4:
                class_label = 3
            elif class_label == class_5:
                class_label = 4
            elif class_label == class_6:
                class_label = 5

            f.write(str(class_label))
            f.write(' ')
            for feat_term in featureids_terms:
                #print(feat_term)
                feat_term = feat_term.replace('(','').replace(')','')
                feat,term = feat_term.split(', ')[0],feat_term.split(',
')[1]
                term = term.replace("'", '')
                if term in v:
                    nDocs_term = len(inv_index[term])

                    nDocs = nDocs_in_subdir(k)   #number of docs(files) in
a subdirectory a document belongs to
                    idf_score = idf_compute(nDocs, nDocs_term)   #100 is
number of documents in a single newsgroup

                    feat_idf_map = str(feat)+':'+str(idf_score)

                    #feat_tf_map = str(feat)+':'+str(tf)

                    f.write(feat_idf_map)
                    f.write(' ')
                    doc_list[class_label].append(feat_idf_map)
                    #doc_list[k].append(feat_tf_map)
            f.write('\n')


#method to generate TF-IDF based training dataset
def training_data_tf_idf_gen(training_data_tf_idf_based,inv_index):
    for pair in class_definition_file:
        key = pair.lstrip('(').rstrip(')').split(', ',1)[0].replace("'",'')
        value = pair.lstrip('(').rstrip(')').split(',
',1)[1].replace("'",'')
        class_definition_pairs[key] = value
    #print(class_definition_pairs)
    count = 0
    doc_list = defaultdict(list)

    with open(training_data_tf_idf_based, 'w') as f:
        for k,v in index.indexed_docs_I().items():
            #print(v)
            news_group = doc_subdir[k]
            class_label = class_definition_pairs[news_group]

            if class_label == class_1:
                class_label = 0
            elif class_label == class_2:
                class_label = 1
            elif class_label == class_3:
                class_label = 2
            elif class_label == class_4:
                class_label = 3
            elif class_label == class_5:
                class_label = 4
            elif class_label == class_6:
                class_label = 5

            f.write(str(class_label))
            f.write(' ')
            for feat_term in featureids_terms:
                #print(feat_term)
                feat_term = feat_term.replace('(','').replace(')','')
                feat,term = feat_term.split(', ')[0],feat_term.split(',
')[1]
                term = term.replace("'", '')
                if term in v:
                    #tf = round((v.count(term))/float(len(v.split(' '))),4)
#normalized term frequency of a term in a document
                    tf_score = tf_compute(v,term)
                    try:
                        nDocs_term = len(inv_index[term])
                        #nDocs = os.system("ls subdir | wc -l")
                        nDocs = nDocs_in_subdir(k)   #number of docs(files)
in a subdirectory a document belongs to
                        idf_score = idf_compute(nDocs, nDocs_term)   #100
```

```
is number of documents in a single newsgroup
                        #idf_score = idf(100, nDocs_term)   #100 is number
of documents in a single newsgroup
                    except:
                        idf_score = 1.0

                    tf_idf_score = round(tf_score * idf_score, 2)
                    feat_tf_idf_map = str(feat)+':'+str(tf_idf_score)

                    #feat_tf_map = str(feat)+':'+str(tf)

                    f.write(feat_tf_idf_map)
                    f.write(' ')
                    doc_list[class_label].append(feat_tf_idf_map)
                    #doc_list[k].append(feat_tf_map)
            f.write('\n')


#main method
if __name__ == '__main__':

    #instantiate class 'Index'
    index = Index(nltk.word_tokenize,
              EnglishStemmer(),
              nltk.corpus.stopwords.words('english'))
    inv_index = index.indexed_docs()   #saves the inverted index into a
variable

    #reads arguments from command line
    dir_newsgroups_data = sys.argv[1]    #reads directory of newsgroups
data(which is the root directory mini_newsgroup)
    subject_body_index(dir_newsgroups_data)

    feature_defn_file = sys.argv[2]    #argument name to use to write
feature definition file
    feature_defn_gen(feature_defn_file)
    print('Produced feature definition file')

    class_defn_file = sys.argv[3]    #argument name to use to write class
definition file
    class_defn_gen(class_defn_file,dir_newsgroups_data)
    print('Produced class definition file')
    train_data_file = sys.argv[4]

    #generates either term-frequency based, inverse document frequency
based or TF-IDF based training dataset
    #file
    if '.TF' in train_data_file:
        training_data_tf_gen(train_data_file)
    elif '.IDF' in train_data_file:
        training_data_idf_gen(train_data_file,inv_index)
    elif '.TFIDF' in train_data_file:
        training_data_tf_idf_gen(train_data_file,inv_index)
    else:
        print("Use file names with extensions '.TF', '.IDF' or '.TFIDF' ")

    print('Produced training data file')
```

classification.py

```
'''
Authors: Manas Gaur, Amanuel Alambo
Instructor: Dr. keke Chen
classification

'''


import codecs
import random
from sklearn import model_selection
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
import warnings


#method to shuffle training data
def shuffle_train_data(file_path, output_file_path):
    with codecs.open(file_path, mode='r', encoding='utf-8') as f:
        data = f.read()
        # Split on \n
        blocks = data.split('\n')
        # Shuffle splits
        random.shuffle(blocks)

    with codecs.open(output_file_path,  mode='w', encoding='utf-8') as output:
        for block in blocks:
            output.write(block)
            # Add the line break
            output.write('\n')

#method to evaluate each classifier performance on f1-macro, recall-macro and precision-macro
def evaluate_classifier(clf, x, y):
    scores_f1_macro = cross_val_score(clf, x.toarray(), y, cv=5, scoring='f1_macro')
    scores_recall_macro = cross_val_score(clf, x.toarray(), y, cv=5, scoring='recall_macro')
    scores_precision_macro = cross_val_score(clf, x.toarray(), y, cv=5, scoring='precision_macro')

    mean_std_f1 = ("f1-macro : %0.2f (+/- %0.2f)" % (scores_f1_macro.mean(), scores_f1_macro.std() * 2))
    mean_std_precision = ("precision-macro : %0.2f (+/- %0.2f)" % (scores_precision_macro.mean(), scores_precision_macro.std() * 2))
    mean_std_recall = ("recall-macro : %0.2f (+/- %0.2f)" % (scores_recall_macro.mean(), scores_recall_macro.std() * 2))

    return mean_std_f1,mean_std_precision,mean_std_recall


#MultinomialNB classifer
def MultinomialNB_classifer(x,y,x_train,y_train,x_test,y_test):
    clf = MultinomialNB()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

#BernoulliNB classifer
def BernoulliNB_classifier(x,y,x_train,y_train,x_test,y_test):
    clf = BernoulliNB()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

#KNeighbors classifer
def KNeighbors_classifier(x,y,x_train,y_train,x_test,y_test):
    clf = KNeighborsClassifier()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

#svm_SVC classifier
def svm_SVC_classifer(x,y,x_train,y_train,x_test,y_test):
    clf = SVC()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)


if __name__ == '__main__':
    warnings.filterwarnings('ignore')

    shuffle_train_data('sample_training_data_file.TFIDF',
'shuffled_train_data.txt')


    feature_vectors, targets =
load_svmlight_file("shuffled_train_data.txt")
    print("Dimension of feature vectors:", feature_vectors.shape)
    print("Dimension of target vectors:", targets.shape)

    #x_test and y_test are the held-out dataset
    ## Splitting the whole dataset for training and testing
    x_train, x_test, y_train, y_test =
model_selection.train_test_split(feature_vectors, targets, test_size = 0.2,
random_state = 41) #75-25 split


    #MultinomialNB_classifer
    print("MultinomialNB Results:")
    print(MultinomialNB_classifer(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #Bernoulli NB classifier
    print("BernoulliNB Results:")
    print(BernoulliNB_classifier(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #K Neighbors classifier
    print("KNeighbors Results:")
    print(KNeighbors_classifier(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #SVM classifier
    print("svm_SVC Results:")
    print(svm_SVC_classifer(feature_vectors,targets,x_train,y_train,
x_test,y_test))
```

April 21, 2019

feature_selection.py

```
'''
Authors: Manas Gaur, Amanuel Alambo
Instructor: Dr. keke Chen
feature selection

'''

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif
from sklearn.datasets import load_svmlight_file
import warnings
from sklearn import model_selection
import pandas as pd
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
import sys
#from sklearn.naive_bayes import MultinomialNB
#from sklearn.naive_bayes import BernoulliNB


import scipy.sparse

import classification   #importing program 'classification'---for testing
the 4 classifiers

#method to call MultinomialNB classifier in program 'classification' and
def call_to_MultinomialNB(X_new1, X_new2, y):
      print('Testing classifiers(MultinomialNB) with chi2 selected
features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new1, targets, test_size = 0.2,
random_state = 15435)
      f_chi2,_,_ =
classification.MultinomialNB_classifer(X_new1,targets,x_train,y_train,
x_test,y_test)
```

```
      print('Testing classifiers(MultinomialNB) with mutual information
selected features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new2, targets, test_size = 0.2,
random_state = 15435)
      f_mi,_,_ =
classification.MultinomialNB_classifer(X_new2,targets,x_train,y_train,
x_test,y_test)

      return f_chi2,f_mi

def call_to_BernoulliNB(X_new1, X_new2, y):
      print('Testing classifiers(BernoulliNB_classifier) with chi2 selected
features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new1, targets, test_size = 0.2,
random_state = 15435)
      f_chi2,_,_ =
classification.BernoulliNB_classifier(X_new1,targets,x_train,y_train,
x_test,y_test)

      print('Testing classifiers(BernoulliNB_classifier) with mutual
information selected features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new2, targets, test_size = 0.2,
random_state = 15435)
      f_mi,_,_ =
classification.BernoulliNB_classifier(X_new2,targets,x_train,y_train,
x_test,y_test)

      return f_chi2,f_mi

def call_to_KNeighbors(X_new1, X_new2, y):
      print('Testing classifiers(KNeighbors_classifier) with chi2 selected
features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new1, targets, test_size = 0.2,
random_state = 15435)
      f_chi2,_,_ =
classification.KNeighbors_classifier(X_new1,targets,x_train,y_train,
x_test,y_test)
```

```
      print('Testing classifiers(KNeighbors_classifier) with mutual
information selected features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new2, targets, test_size = 0.2,
random_state = 15435)
      f_mi,_,_ =
classification.KNeighbors_classifier(X_new2,targets,x_train,y_train,
x_test,y_test)

      return f_chi2,f_mi

def call_to_svm_SVC_classifer(X_new1, X_new2, y):
      print('Testing classifiers(svm_SVC_classifer) with chi2 selected
features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new1, targets, test_size = 0.2,
random_state = 15435)
      f_chi2,_,_ =
classification.svm_SVC_classifer(X_new1,targets,x_train,y_train,
x_test,y_test)

      print('Testing classifiers(svm_SVC_classifer) with mutual information
selected features')
      x_train, x_test, y_train, y_test =
model_selection.train_test_split(X_new2, targets, test_size = 0.2,
random_state = 15435)
      f_mi,_,_ =
classification.svm_SVC_classifer(X_new2,targets,x_train,y_train,
x_test,y_test)

      return f_chi2,f_mi

#method to plot learning curve given a dataframe and model name
def plot_learning_curve(title, df):

          plt.figure()
          plt.title(title)
          plt.xlabel("K(top features)")
          plt.ylabel("Score")

          chi2_f1_mean, chi2_f1_std, mi_f1_mean, mi_f1_std = list(),
list(), list(), list()
```

```
     for i,row in df.iterrows():
      chi2_f1_mean.append(float(row['chi2_f1'].split(': ')[1].split('
(+/- ')[0]))
      chi2_f1_std.append(float(row['chi2_f1'].split(': ')[1].split('
(+/- ')[1].rstrip(')')))

      mi_f1_mean.append(float(row['mi_f1'].split(': ')[1].split('
(+/- ')[0]))
      mi_f1_std.append(float(row['mi_f1'].split(': ')[1].split(' (+/-
')[1].rstrip(')')))


     train_sizes = np.array(range(100, (df.shape[0]+1)*100, 100))

     plt.grid()
     plt.fill_between(train_sizes, np.array(chi2_f1_mean) -
np.array(chi2_f1_std),
                     np.array(chi2_f1_mean) + np.array(chi2_f1_std),
alpha=0.1,
                     color="r")

     plt.fill_between(train_sizes, np.array(mi_f1_mean) -
np.array(mi_f1_std),
                     np.array(mi_f1_mean) + np.array(mi_f1_std),
alpha=0.1,
                     color="g")


     plt.plot(train_sizes, np.array(chi2_f1_mean), 'o-', color="r",
         label="chi2 f1-score")
     plt.plot(train_sizes, np.array(mi_f1_mean), 'o-', color="g",
         label="MI f1-score")

     plt.legend(loc="best")
     return plt

if __name__ == '__main__':
   warnings.filterwarnings('ignore')
   feature_vectors, targets =
load_svmlight_file("shuffled_train_data.txt")
   X = feature_vectors
```

```
    y = targets

    d=list()
    title = ''
    model_name = sys.argv[1]
    if model_name.lower() == 'bnb':
        title = "Learning Curves(BernoulliNB)"
    elif model_name.lower() == 'mnb':
        title = "Learning Curves(MultinomialNB)"
    elif model_name.lower() == 'knb':
        title = "Learning Curves(KNeighbors)"
    elif model_name.lower() == 'svm':
        title = "Learning Curves(SVM)"
    #title = "Learning Curves",(model_name)

    for i in range(1, 11):
        X_new1 = SelectKBest(chi2, k=i*100).fit_transform(X, y)
        X_new2 = SelectKBest(mutual_info_classif,
k=i*100).fit_transform(X, y)

        #f_chi2,f_mi = call_to_BernoulliNB(X_new1,X_new2,y)
        #f_chi2,f_mi = call_to_MultinomialNB(X_new1,X_new2,y)
        #f_chi2,f_mi = call_to_KNeighbors(X_new1,X_new2,y)
        if model_name.lower() == 'bnb':
            f_chi2,f_mi = call_to_BernoulliNB(X_new1,X_new2,y)
        elif model_name.lower() == 'mnb':
            f_chi2,f_mi = call_to_MultinomialNB(X_new1,X_new2,y)
        elif model_name.lower() == 'knb':
            f_chi2,f_mi = call_to_KNeighbors(X_new1,X_new2,y)
        elif model_name.lower() == 'svm':
            f_chi2,f_mi = call_to_svm_SVC_classifer(X_new1,X_new2,y)
        else:
            print("Use either MNB(for MultinomialNB), BNB(for
BernoulliNB, KNB(for KNeighbors), or SVM")

        d.append({'k':i*100, 'chi2_f1': f_chi2, 'mi_f1': f_mi})
    df = pd.DataFrame(d)


    #title = "Learning Curves (BernoulliNB)"
    #title = "Learning Curves (MultinomialNB)"
    #title = "Learning Curves (KNeighbors)"
```

```
      plot_learning_curve(title, df)   #call to plotting learning curve
method

      plt.show()
```

clustering.py

```
'''
Authors: Manas Gaur, Amanuel Alambo
Instructor: Dr. keke Chen
clustering

'''

from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import load_svmlight_file
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
import sys

def plot_learning_curve(title, df):

          plt.figure()
          plt.title(title)
          #if ylim is not None:
           #   plt.ylim(*ylim)
          plt.xlabel("K(number of clusters)")
          plt.ylabel("Score")
          #train_sizes, train_scores, test_scores =
learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
train_sizes=train_sizes)
```

```
          #scores_f1_macro = cross_val_score(clf, x.toarray(), y, cv=5,
scoring='f1_macro')
          #train_scores = f1-macro : 0.07 (+/- 0.00)



          #train_sizes = np.array(range((2, df.shape[0]+2)))
          train_sizes = np.array(range(1, (df.shape[0]+1)))
          #print(train_sizes.shape)
          #print(len(chi2_f1_mean))
          #raise KeyboardInterrupt

          plt.grid()

          plt.plot(train_sizes, df.silhoutte_score.values, 'o-', color="r",
                   label="silhouette Score")
          plt.plot(train_sizes, df.nmi_score.values, 'o-', color="g",
                   label="NMI Score")

          plt.legend(loc="best")
          return plt

feature_vectors, targets = load_svmlight_file("shuffled_train_data.txt")
#kmeans_model = KMeans(n_clusters=20).fit(feature_vectors)
#single_linkage_model = AgglomerativeClustering(n_clusters=20,
linkage='ward').fit(feature_vectors.toarray())

X = feature_vectors
classification_labels = targets

model_name = sys.argv[1]

d=list()
for i in range(2, 25):
      print('number of clusters: ',i)
      if model_name.lower() == 'kmeans':
            kmeans_model = KMeans(n_clusters=i).fit(feature_vectors)
            clustering_labels = kmeans_model.labels_
      elif model_name.lower() == 'agglomerative':
            single_linkage_model = AgglomerativeClustering(n_clusters=i,
linkage='ward').fit(feature_vectors.toarray())
            clustering_labels = single_linkage_model.labels_
      else:
            print('Use KMeans or Agglomerative as your argument')

      X = feature_vectors
      classification_labels = targets

      silhoutte_score = metrics.silhouette_score(X, clustering_labels,
metric='euclidean')
      nmi_score =
metrics.normalized_mutual_info_score(classification_labels,
clustering_labels)
      print('silhoutte_score: ',silhoutte_score)
      print('nmi_score: ',nmi_score)
      #X_new1 = SelectKBest(chi2, k=i+1).fit_transform(X, y)
      #X_new2 = SelectKBest(mutual_info_classif, k=i*100).fit_transform(X,
y)
      #calls to each classifier for the new features
      #f_chi2,f_mi = call_to_MultinomialNB(X_new1,X_new2,y)
      #f_chi2,f_mi = call_to_BernoulliNB(X_new1,X_new2,y)
      d.append({'k':i, 'silhoutte_score': silhoutte_score, 'nmi_score':
nmi_score})
      #call_to_KNeighbors(X_new1,X_new2,y)
      #call_to_svm_SVC_classifer(X_new1,X_new2,y)
df = pd.DataFrame(d)

title = "Document Clustering Evaluation"
plot_learning_curve(title, df)

plt.show()
```

```
Testing classifiers(MultinomialNB) with chi2 selected features
Train accuracy: 0.361
Test accuracy: 0.362
('f1-macro : 0.26 (+/- 0.03)', 'precision-macro : 0.52 (+/- 0.05)',
'recall-macro : 0.30 (+/- 0.03)')
Testing classifiers(MultinomialNB) with mutual information selected
features
Train accuracy: 0.296
```

```
Test accuracy: 0.296
('f1-macro : 0.16 (+/- 0.03)', 'precision-macro : 0.36 (+/- 0.08)',
'recall-macro : 0.22 (+/- 0.03)')
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.604
Test accuracy: 0.586
('f1-macro : 0.56 (+/- 0.06)', 'precision-macro : 0.55 (+/- 0.05)',
'recall-macro : 0.57 (+/- 0.07)')
Testing classifiers(BernoulliNB_classifier) with mutual information
selected features
Train accuracy: 0.515
Test accuracy: 0.488
('f1-macro : 0.46 (+/- 0.03)', 'precision-macro : 0.48 (+/- 0.04)',
'recall-macro : 0.50 (+/- 0.04)')
Testing classifiers(KNeighbors_classifier) with chi2 selected features
Train accuracy: 0.586
Test accuracy: 0.353
('f1-macro : 0.34 (+/- 0.05)', 'precision-macro : 0.42 (+/- 0.11)',
'recall-macro : 0.34 (+/- 0.05)')
Testing classifiers(KNeighbors_classifier) with mutual information selected
features
Train accuracy: 0.586
Test accuracy: 0.372
('f1-macro : 0.35 (+/- 0.02)', 'precision-macro : 0.40 (+/- 0.05)',
'recall-macro : 0.34 (+/- 0.02)')
Testing classifiers(svm_SVC_classifer) with chi2 selected features
Train accuracy: 0.255
Test accuracy: 0.245
('f1-macro : 0.07 (+/- 0.00)', 'precision-macro : 0.04 (+/- 0.00)',
'recall-macro : 0.17 (+/- 0.00)')
Testing classifiers(svm_SVC_classifer) with mutual information selected
features
Train accuracy: 0.255
Test accuracy: 0.245
('f1-macro : 0.07 (+/- 0.00)', 'precision-macro : 0.04 (+/- 0.00)',
'recall-macro : 0.17 (+/- 0.00)')
```

Classification.py

```
import codecs
import random
from sklearn import model_selection
```

April 21, 2019

```
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
import warnings


def shuffle_train_data(file_path, output_file_path):
    with codecs.open(file_path, mode='r', encoding='utf-8') as f:
        data = f.read()
        # Split on \n
        blocks = data.split('\n')
        # Shuffle splits
        random.shuffle(blocks)

    with codecs.open(output_file_path,  mode='w', encoding='utf-8') as
output:
        for block in blocks:
            output.write(block)
            # Add the line break
            output.write('\n')

def evaluate_classifier(clf, x, y):
    scores_f1_macro = cross_val_score(clf, x.toarray(), y, cv=5,
scoring='f1_macro')
    scores_recall_macro = cross_val_score(clf, x.toarray(), y, cv=5,
scoring='recall_macro')
    scores_precision_macro = cross_val_score(clf, x.toarray(), y, cv=5,
scoring='precision_macro')

    mean_std_f1 = ("f1-macro : %0.2f (+/- %0.2f)" %
(scores_f1_macro.mean(), scores_f1_macro.std() * 2))
    mean_std_precision = ("precision-macro : %0.2f (+/- %0.2f)" %
(scores_precision_macro.mean(), scores_precision_macro.std() * 2))
    mean_std_recall = ("recall-macro : %0.2f (+/- %0.2f)" %
(scores_recall_macro.mean(), scores_recall_macro.std() * 2))

    return mean_std_f1,mean_std_precision,mean_std_recall
```

```
def MultinomialNB_classifer(x,y,x_train,y_train,x_test,y_test):
    clf = MultinomialNB()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

def BernoulliNB_classifier(x,y,x_train,y_train,x_test,y_test):
    clf = BernoulliNB()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

def KNeighbors_classifier(x,y,x_train,y_train,x_test,y_test):
    clf = KNeighborsClassifier()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)

def svm_SVC_classifer(x,y,x_train,y_train,x_test,y_test):
    clf = SVC()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    train_score = clf.score(x_train, y_train)
    test_score = clf.score(x_test, y_test)
    print("Train accuracy:",train_score)
    print("Test accuracy:",test_score)
    #print(evaluate_classifier(clf,x,y))
    return evaluate_classifier(clf,x,y)


if __name__ == '__main__':
    warnings.filterwarnings('ignore')

    shuffle_train_data('sample_training_data_file.TFIDF',
'shuffled_train_data.txt')

#shuffle_train_data('/home/amanuel/WSU-II/Courses/Information_Retrieval/Pro
ject_2/sample_training_data_file_tf_idf.libsvm', 'output.txt')
    ## Splitting the whole dataset for training and testing

    feature_vectors, targets =
load_svmlight_file("shuffled_train_data.txt")
    print("Dimension of feature vectors:", feature_vectors.shape)
    print("Dimension of target vectors:", targets.shape)

    #x_test and y_test are the held-out dataset
    x_train, x_test, y_train, y_test =
model_selection.train_test_split(feature_vectors, targets, test_size = 0.2,
random_state = 41) #75-25 split


    #MultinomialNB_classifer
    print("MultinomialNB Results:")
    print(MultinomialNB_classifer(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #Bernoulli NB classifier
    print("BernoulliNB Results:")
    print(BernoulliNB_classifier(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #K Neighbors classifier
    print("KNeighbors Results:")
    print(KNeighbors_classifier(feature_vectors,targets,x_train,y_train,
x_test,y_test))

    #SVM classifier
    print("svm_SVC Results:")
    print(svm_SVC_classifer(feature_vectors,targets,x_train,y_train,
x_test,y_test))
```