NLP Custom Algorithm

August 18, 2020

[ ]:
```
""" Natural Language Processing - custom algorithm

    Brian Hogan, bphogan@syracuse.edu Natural Language Processing project
    For computational linguist Dr. Nancy McCracken, Syracuse 2020
    Acquire Tweet Data and apply a custom NLP algorithm to parse data
    by POS, bigrams, collocations, and negation features.
"""
import os   #os.getcwd()
import sys
import nltk
from nltk.tokenize import TweetTokenizer
twtokenizer = TweetTokenizer()    # initialize NLTK built-in tweet tokenizer
import random
from nltk.corpus import stopwords
from nltk.collocations import * #for bigram feature sets; line 2173 my code
bigram_measures = nltk.collocations.BigramAssocMeasures()
from nltk.corpus import sentence_polarity #moview review week 8 lab used in␣
 ↪combining features
import re
from nltk.corpus import words as nltk_words
import pandas as pd

""" Manual Import Features """ #set these for path location, import settings
#def processtweets(dirPath,limitStr):
os.chdir('C:
 ↪\\Users\\17574\\Desktop\\ist664+NLP\\WK+x+Final+Project\\finalproject_DATA\\SemEval2014Twee
limitStr = 26928  # start tweets to import
limitStr = 3200
f = open('./downloaded-tweeti-a.dist.tsv', 'r') # train
#f = open('./train_raw_data.tsv', 'r') # train   [0:4]

""" Step-0: Import Data  : from semEval.py in final folder"""
limit = int(limitStr) #ensure check total tweets and if randomized
tweetdata = []                    #first get source file two 2 columns...
for line in f:
    if (len(tweetdata) < limit):
        line = line.strip()   #remove final end of line character
```

```
        tweetdata.append(line.split('\t')[0:4])       #[2:4])
len(tweetdata)
tweetdata[:10]
        #['"neutral"',
        # 'Some areas of New England could see the first flakes of the season␣
 ↪Tuesday.']
type(tweetdata)
tweetdata[1:10]
f.close()

#export as using Gates routine until I am able to write my own!
df_output = pd.DataFrame(tweetdata)
df_output.to_csv("rawData_mytrain.csv", index=True)
```

[ ]:
```
""" Data Cleaning"""
FILE = open(rawfilename, "r")
filename = "a_CLEAN.txt" #csv    #clean then write it back to csv!need empty␣
 ↪csv file
NEWFILE = open(filename, "w") ##in 1st row create lable & text columns
towrite = "Label, Text\n"      ##"""write this to new empty csv file"""
NEWFILE.write(towrite)
NEWFILE.close()
NEWFILE = open(filename, "a")
myfinaldf = pd.DataFrame()
outputfile = "a_audit.txt"
OUTFILE = open(outputfile, "w")
OUTFILE.close()
OUTFILE = open(outputfile,"a")  ###remember to close this below!
bbe_remove_non_english_words=0  #0 = off: CANT USE BECASE OF TEXTING
bbe_remove_stopwords = 1
tweetdocs = []
mylabels=[]

for row in FILE:   ##going line by line
    #os.chdir('c:\\Users\BBE\BBE\DATA')
    rawrow="\n\nThe row is: " + row + "\n"
    OUTFILE.write(rawrow) ##i am going to write this again later for comp
    row = row.lstrip()  #strip all space from the left
    row = row.rstrip()  ##strip all the space fro the right
    row = row.strip() #strip all extra spaces in general
    #print(row)  #split up the row of text by space - tokenenize it into list
    mylist = row.split(" ") #is this the issue here
    label = mylist[0] #ok this seperates the label
    del mylist[0]  #remove the label from the tweet
    #mylist = mylist.pop()
    #print(mylist)
    newlist = []
```

```
    for word in mylist:
        #print("the new word is: ",word)
        placeinoutputfile = "The next word before is: " + word + "\n"
        OUTFILE.write(placeinoutputfile)
        word = word.lower()
        word = word.lstrip()
        word = word.strip("\n")
        word = word.strip("\\n")
        word = word.replace(",","")
        word = word.replace(" ","")
        word = word.replace("_","")
        word = re.sub('\+', '',word)
        word = re.sub('.*\+\n','',word)   ##LOOKS FUNNY! single quotes!
        word = re.sub('zz+','',word)
        word = word.replace("\t","")
        word = word.replace(".","")
        word = word.replace("\'s","")  #was comment3d out
        word = re.sub('[!@#?$-.]','',word) #get the ques mark out
        word = word.replace("?","")
        word = word.strip()
        ##word.replace("\","")  #was commented out
        if word not in["","\\","'","*",":",";"]:
            if len(word) >=1:
                if not re.search(r'\d',word): ##remove the digits
                    # HW2 ===non english words
                    if bbe_remove_non_english_words==1:  #code to remove␣
↪nonenglish words
                        if word in nltk_words.words():
                            word= word
                        else:
                            word = ""
                    if bbe_remove_stopwords==1:
                        stop_words=set(stopwords.words("english"))
                        if word not in stop_words:
                            word = word
                    newlist.append(word)
                    placeinoutputfile = "The next word AFTER is: " + word + "\n"
                    OUTFILE.write(placeinoutputfile)
                    ##NOW WE HAVE ALL THE WORDS
    """thisis where I was having trouble"""
    #label = newlist[-1]  #LAVEL NOW IN FIRST POSITION  -1 is the last cell
    #print(newlist[0])
    mylabels.append(label)
    if "pos" in label:   # change to "pos"   or "neg" depend on file
        label = "pos"
    if "neg" in label:
        label = "neg"
```

```
    if "neu" in label:
        label = "neu"
#    else:
#        label = "neu"
    placeinoutputfile = "\n The label is: " + label + "\n"
    OUTFILE.write(placeinoutputfile)
    text = " ".join(newlist)
#    text = text.replace("\\n","")
#    text = text.strip("\\n")
#    text = text.replace("\\'","")
#    text = text.replace("\\","")
#    text = text.replace("''","")
#    text = text.replace("'","")
#    text = text.replace("s'","")
#    text = text.lstrip()
    tokens = twtokenizer.tokenize(text)
    tweetdocs.append((tokens, label))

    OUTFILE.write(rawrow)
    towrite = label+","+text+"\n"
    NEWFILE.write(towrite)
    OUTFILE.write(towrite)
FILE.close()       ##alwasy  the files!
NEWFILE.close()
OUTFILE.close()

test_clean_data = tweetdocs
train_clean_data = tweetdocs
  #double checking on label generation

tweetdocs[:10]
len(tweetdocs)
```

[ ]:
```
"""DAta Labeling """
tweetdocs = []    # create list of tweet documents as (list of words, label)
  # add all the tweets except the ones whose text is Not Available
  for tweet in tweetdata:
    if (tweet[1] != 'Not Available'):
      # run the tweet tokenizer on the text string - returns unicode tokens, so␣
 ↪convert to utf8
      tokens = twtokenizer.tokenize(tweet[1])
      if tweet[0] == '"positive"':
        label = 'pos'
      else:
        if tweet[0] == '"negative"':
          label = 'neg'
        else:          # labels are condensed to just 3:  'pos', 'neg', 'neu'
```

```
        if (tweet[0] == '"neutral"') or (tweet[0] == '"objective"') or␣
↪(tweet[0] == '"objective-OR-neutral"'):
          label = 'neu'
        else:
          label = ''
      tweetdocs.append((tokens, label))

for tweet in tweetdocs[:2]: #this has grabbed the data file+tokens + rating
    print (tweet)
for tweet in tweetdocs: #this has grabbed the data file+tokens + rating
    print (tweet)
```

[ ]:
```
"""Model evaluation functions"""
ef cross_validation_PRF(num_folds, featuresets, labels):
    subset_size = int(len(featuresets)/num_folds)
    print('Each fold size:', subset_size)
    # for the number of labels - start the totals lists with zeroes
    num_labels = len(labels)
    total_precision_list = [0] * num_labels
    total_recall_list = [0] * num_labels
    total_F1_list = [0] * num_labels

    # iterate over the folds
    for i in range(num_folds):
        test_this_round = featuresets[(i*subset_size):][:subset_size]
        train_this_round = featuresets[:(i*subset_size)] +␣
↪featuresets[((i+1)*subset_size):]
        # train using train_this_round
        classifier = nltk.NaiveBayesClassifier.train(train_this_round)
        # evaluate against test_this_round to produce the gold and predicted␣
↪labels
        goldlist = []
        predictedlist = []
        for (features, label) in test_this_round:
            goldlist.append(label)
            predictedlist.append(classifier.classify(features))

        # computes evaluation measures for this fold and
        #   returns list of measures for each label
        print('Fold', i)
        (precision_list, recall_list, F1_list) \
                  = eval_measures(goldlist, predictedlist, labels)
        # take off triple string to print precision, recall and F1 for each fold
        '''
        print('\tPrecision\tRecall\t\tF1')
        # print measures for each label
        for i, lab in enumerate(labels):
```

```
            print(lab, '\t', "{:10.3f}".format(precision_list[i]), \
              "{:10.3f}".format(recall_list[i]), "{:10.3f}".format(F1_list[i]))
        '''
        # for each label add to the sums in the total lists
        for i in range(num_labels):
            # for each label, add the 3 measures to the 3 lists of totals
            total_precision_list[i] += precision_list[i]
            total_recall_list[i] += recall_list[i]
            total_F1_list[i] += F1_list[i]

    # find precision, recall and F measure averaged over all rounds for all␣
↪labels
    # compute averages from the totals lists
    precision_list = [tot/num_folds for tot in total_precision_list]
    recall_list = [tot/num_folds for tot in total_recall_list]
    F1_list = [tot/num_folds for tot in total_F1_list]
    # the evaluation measures in a table with one row per label
    print('\nAverage Precision\tRecall\t\tF1 \tPer Label')
    # print measures for each label
    for i, lab in enumerate(labels):
        print(lab, '\t', "{:10.3f}".format(precision_list[i]), \
          "{:10.3f}".format(recall_list[i]), "{:10.3f}".format(F1_list[i]))

    # print macro average over all labels - treats each label equally
    print('\nMacro Average Precision\tRecall\t\tF1 \tOver All Labels')
    print('\t', "{:10.3f}".format(sum(precision_list)/num_labels), \
          "{:10.3f}".format(sum(recall_list)/num_labels), \
          "{:10.3f}".format(sum(F1_list)/num_labels))

    # for micro averaging, weight the scores for each label by the number of␣
↪items
    #    this is better for labels with imbalance
    # first intialize a dictionary for label counts and then count them
    label_counts = {}
    for lab in labels:
      label_counts[lab] = 0
    # count the labels
    for (doc, lab) in featuresets:
      label_counts[lab] += 1
    # make weights compared to the number of documents in featuresets
    num_docs = len(featuresets)
    label_weights = [(label_counts[lab] / num_docs) for lab in labels]
    print('\nLabel Counts', label_counts)
    #print('Label weights', label_weights)
    # print macro average over all labels
    print('Micro Average Precision\tRecall\t\tF1 \tOver All Labels')
    precision = sum([a * b for a,b in zip(precision_list, label_weights)])
```

```
    recall = sum([a * b for a,b in zip(recall_list, label_weights)])
    F1 = sum([a * b for a,b in zip(F1_list, label_weights)])
    print( '\t', "{:10.3f}".format(precision), \
      "{:10.3f}".format(recall), "{:10.3f}".format(F1))

def eval_measures(gold, predicted, labels):
    # these lists have values for each label
    recall_list = []
    precision_list = []
    F1_list = []
    for lab in labels:
        # for each label, compare gold and predicted lists and compute values
        TP = FP = FN = TN = 0
        for i, val in enumerate(gold):
            if val == lab and predicted[i] == lab:  TP += 1
            if val == lab and predicted[i] != lab:  FN += 1
            if val != lab and predicted[i] == lab:  FP += 1
            if val != lab and predicted[i] != lab:  TN += 1
        # use these to compute recall, precision, F1
        # for small numbers, guard against dividing by zero in computing␣
 ↪measures
        if (TP == 0) or (FP == 0) or (FN == 0):
          recall_list.append (0)
          precision_list.append (0)
          F1_list.append(0)
        else:
          recall = TP / (TP + FP)
          precision = TP / (TP + FN)
          recall_list.append(recall)
          precision_list.append(precision)
          F1_list.append( 2 * (recall * precision) / (recall + precision))
    # the evaluation measures in a table with one row per label
    return (precision_list, recall_list, F1_list)
```

[ ]:
```
"""word feature functions"""
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features

"""this is part were getting data from teh fucntions"""
                #from line 216 in classifykaggle-crossval....
  # continue as usual to get all words and create word features
all_words_list = [word for (sent,cat) in tweetdocs for word in sent]
all_words = nltk.FreqDist(all_words_list)
```

```
print("number of words",len(all_words))
word_items = all_words.most_common(3000) #vocabulary of the most occuring
word_features = [word for (word,count) in word_items] # (below)feature sets
 ↪from a feature definition function
featuresets = [(document_features(d, word_features), c) for (d, c) in tweetdocs]
"""so here what we have for a feature set is whether word is noun, verb, etc
 and whtehr the tweet contains one of the most common words"""

#line 228  classifykaggle-crossval....
  # train classifier and show performance in cross-validation; get list labels
  """still confused on how this works hwere---THE CODE UP below"""
#tweetdocs[1]
label_list = [c for (d,c) in tweetdocs]
labels = list(set(label_list))    # gets only unique labels
labels                  # ['neu', 'neg', 'pos']

num_folds = 3
cross_validation_PRF(num_folds, featuresets, labels)
```

[ ]:
```
"""bigram features  """
def bigram_document_features(document, word_features, bigram_features):
 ↪#documet and wordfeatures are the vocabaulary
    document_words = set(document)
    document_bigrams = nltk.bigrams(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    for bigram in bigram_features: #b for any bigram
        features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in
 ↪document_bigrams)
    return features

from nltk.collocations import * #for bigram feature sets
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(all_words_list)
bigram_features = finder.nbest(bigram_measures.chi_sq, 500)
# use this function to create feature sets for all sentences #something about
 ↪keep stopwords as this feature does its own
bigram_featuresets = [(bigram_document_features(d, word_features,
 ↪bigram_features), c) for (d, c) in tweetdocs]
# number of features for document 0
print(len(bigram_featuresets[0][0].keys()))#1500 words, 2100 bigrams
print(bigram_featuresets[0][0]) #first document; first of the pairs # features
 ↪in document 0
#########################################################################
label_list = [c for (d,c) in tweetdocs]
```

```
labels = list(set(label_list))    # gets only unique labels
num_folds = 5
cross_validation_PRF(num_folds, featuresets, labels)
```

[ ]:
```
"""--POS  FEATURES---------------------------------"""
def POS_features(document, word_features):
    document_words = set(document)
    tagged_words = nltk.pos_tag(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features

# define feature sets using this function
POS_featuresets = [(POS_features(d, word_features), c) for (d, c) in tweetdocs]

print(len(POS_featuresets[0][0].keys()))  # number of features for document 0
print(tweetdocs[0])   # the first sentence
# the pos tag features for this sentence
print('num nouns', POS_featuresets[0][0]['nouns'])
print('num verbs', POS_featuresets[0][0]['verbs'])
print('num adjectives', POS_featuresets[0][0]['adjectives'])
print('num adverbs', POS_featuresets[0][0]['adverbs'])
#                                                    POS part of speech
label_list = [c for (d,c) in tweetdocs]
labels = list(set(label_list))    # gets only unique labels
num_folds = 5
cross_validation_PRF(num_folds, POS_featuresets, labels)
```

[ ]:
```
"""feature combination functions"""
def My_COMBINED_features(document, word_features, bigram_features):
    document_words = set(document)
    document_bigrams = nltk.bigrams(document) #from bigram fearure function
    tagged_words = nltk.pos_tag(document) #from POS tagger
```

```
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    for bigram in bigram_features: #b for any bigram
        features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in␣
 ↪document_bigrams)
    numNoun = 0
    numVerb = 0   #POS features
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features

bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(all_words_list)
                                                    #can experiment␣
 ↪with 500 below
bigram_features = finder.nbest(bigram_measures.chi_sq, 600)

Example_combined = [(PROFESSOR_COMBINED_features(d, word_features,␣
 ↪bigram_features), c) for (d, c) in tweetdocs]
print(len(Example_combined[0][0].keys()))  # number of features for document 0
print(tweetdocs[0])   # the first sentence
print('num nouns', Example_combined[0][0]['nouns'])  # the pos tag features for␣
 ↪this sentence
print('num verbs', Example_combined[0][0]['verbs'])
print('num adjectives', Example_combined[0][0]['adjectives'])
print('num adverbs', Example_combined[0][0]['adverbs'])
#Example_combined[1:]
label_list = [c for (d,c) in tweetdocs]
labels = list(set(label_list))    # gets only unique labels
num_folds = 5
cross_validation_PRF(num_folds, Example_combined, labels)

negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone',␣
 ↪'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']

def BBE_features(document, word_features, bigram_features,negationwords):
    document_words = set(document)
```

```
    document_bigrams = nltk.bigrams(document) #from bigram fearure function
    tagged_words = nltk.pos_tag(document) #from POS tagger
    features = {}
    #the V and B are subset type features
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    for bigram in bigram_features: #b for any bigram
        features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in␣
 ↪document_bigrams)
    numNoun = 0
    numVerb = 0   #POS features
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    for word in word_features:                    ###BBE adding negation feature
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or (word.
 ↪endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in␣
 ↪word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    return features

len(tweetdocs)
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(all_words_list)  #can experiment␣
 ↪with 500 bel
bigram_features = finder.nbest(bigram_measures.chi_sq, 600)
#this is the baseline code
BBE_featuresets = [(BBE_features(d, word_features,␣
 ↪bigram_features,negationwords), c) for (d, c) in tweetdocs]
```

[ ]:
```
"""EXPERIMENTATION"""
len(tweetdocs)
test_clean_data = tweetdocs
test2_clean_data = tweetdocs
train2_clean_data = tweetdocs
len(train2_clean_data)
len(test2_clean_data)
train_clean_data #PROCESS FIRST

"""-----------AT LAST THE CORRECT COMPARISON OUTCOME METRIC---------------"""
BBE_featuresets = [(BBE_features(d, word_features,␣
 ↪bigram_features,negationwords), c) for (d, c) in train_clean_data]
#above is train, below is test
test_featuresets = [(BBE_features(d, word_features,␣
 ↪bigram_features,negationwords), c) for (d, c) in test_clean_data]
train_set, test_set = BBE_featuresets, test_featuresets
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
classifier.show_most_informative_features(30)

#experiment running more words
test2_featuresets = [(BBE_features(d, word_features,␣
 ↪bigram_features,negationwords), c) for (d, c) in test2_clean_data]
train2_featuresets = [(BBE_features(d, word_features,␣
 ↪bigram_features,negationwords), c) for (d, c) in train2_clean_data]

train_set, test_set = train2_featuresets, test2_featuresets
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
classifier.show_most_informative_features(30)

BBE_featuresets[0:1]
print(len(BBE_featuresets[0][0].keys()))  # number of features for document 0
print(tweetdocs[0])   # the first sentence
print('num nouns', BBE_featuresets[0][0]['nouns'])  # the pos tag features for␣
 ↪this sentence
print('num verbs', BBE_featuresets[0][0]['verbs'])
print('num adjectives', BBE_featuresets[0][0]['adjectives'])
print('num adverbs', BBE_featuresets[0][0]['adverbs'])

label_list = [c for (d,c) in tweetdocs]
labels = list(set(label_list))    # gets only unique labels
num_folds = 5
cross_validation_PRF(num_folds, BBE_featuresets, labels)

#negation cross validation
len(BBE_featuresets)
```

```
train_set, test_set = BBE_featuresets[1000:], BBE_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
classifier.show_most_informative_features(30)
```

[ ]:
```
"""Write out the feature sets for machien learning evaluation"""
def writeFeatureSets(featuresets, outpath):
    f = open(outpath, 'w')  # open outpath for writing
    featurenames = featuresets[0][0].keys()  # get the feature names from the␣
 ↪feature dictionary in the first featureset
          # create the first line of the file as comma separated feature names
    #    with the word class as the last feature name
    featurenameline = ''
    for featurename in featurenames:
        # replace forbidden characters with text abbreviations
        featurename = featurename.replace(',','CM')
        featurename = featurename.replace("'","DQ")
        featurename = featurename.replace('"','QU')
        #featurename = featurename.replace('?','Qu') #BBE addings
        #featurename = featurename.replace('','XX') #BBE addings
    featurenameline += featurename + ','
    featurenameline += 'class'
    # write this as the first line in the csv file
    f.write(featurenameline)
    f.write('\n')
    # convert each feature set to a line in the file with comma separated␣
 ↪feature values,
    # each feature value is converted to a string
    #   for booleans this is the words true and false for numbers, this is the␣
 ↪string with the number
    for featureset in featuresets:
        print(featureset)
        featureline = ''
        for key in featurenames:
            if featurenames != "":
                featureline += str(featureset[0][key]) + ','
#             if key == FALSE:
#                 featureline += str("missing") + ','
        featureline += featureset[1]
        # write each feature set values to the file
        f.write(featureline)
        f.write('\n')
    f.close()
"""writing the featuresets to a file"""
#writeFeatureSets(featuresets, outpath)
```

```
outpath = 'C:
 ↪\\Users\\17574\\Desktop\\ist664+NLP\\WK+x+Final+Project\\finalproject_DATA\\SemEval2014Twee
 ↪tsv'
writeFeatureSets(BBE_featuresets, outpath)

# define features (keywords) of a document for a BOW/unigram baseline
# each feature is 'contains(keyword)' and is true or false depending
# on whether that keyword is in the document
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features

"""was not using the main program"""
# Main program to produce movie review feature sets in order to show how to use
#   the writeFeatureSets function
if __name__ == '__main__':
    # Make a list of command line arguments, omitting the [0] element
    # which is the script itself.
    args = sys.argv[1:]
    if not args:
        print ('usage: python save_features.py [file]')
        sys.exit(1)
    outpath = args[0]
    # for each document in movie_reviews, get its words and category (positive/
 ↪negative)
    documents = [(list(movie_reviews.words(fileid)), category)
                 for category in movie_reviews.categories()
                 for fileid in movie_reviews.fileids(category)]
    random.shuffle(documents)
    # get all words from all movie_reviews and put into a frequency distribution
    #   note lowercase, but no stemming or stopwords
    all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
    # get the most frequently appearing keywords in the corpus
    word_items = all_words.most_common(vocab_size)
    word_features = [word for (word, freq) in word_items]
    # get features sets for a document, including keyword features and category␣
 ↪feature
    featuresets = [(document_features(d, word_features), c) for (d, c) in␣
 ↪documents]
    # write the feature sets to the csv file
    writeFeatureSets(featuresets, outpath)
    print ('Wrote movie review features to:', outpath)
```

```
[ ]: """------FINAL RUNNING OF THE TRAIN AND TEST ENVIRONMENT------"""
# function to read features, perform cross-validation with (several)␣
 ↪classifiers and report results
import sys;import pandas;import numpy
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_predict
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression

def process(filepath):
  # number of folds for cross-validation
  kFolds = 5
  # read in the file with the pandas package
  train_set = pandas.read_csv(filepath)
  # this is a data frame for the data
  print ('Shape of feature data - num instances with num features + class␣
 ↪label')
  print (train_set.shape)
  # convert to a numpy array for sklearn
  train_array = train_set.values
  # get the last column with the class labels into a vector y
  train_y = train_array[:,-1]
  # get the remaining rows and columns into the feature matrix X
  train_X = train_array[:,:-1]
  #  ** choose one of these classifiers **
  #print '** Results from Linear SVM'
  # now call sklearn with SVC to get a model
  #classifier = LinearSVC(C=1, penalty='l1', dual=False, class_weight='auto')
  #print '** Results from Naive Bayes'
  #classifier = MultinomialNB()
  print ('** Results from Logistic Regression with liblinear')
  #print '** Results from Logistic Regression with newton-cg'
  #print '** Results from Logistic Regression with lbfgs'
  ## solver options:  solver : {'newton-cg', 'lbfgs', 'liblinear'}
  ## multi-class options: multi_class : str, {'ovr', 'multinomial'}
  #but multinomial only for lbfgs
  classifier =␣
 ↪LogisticRegression(class_weight='balanced',solver='lbfgs',multi_class='multinomial')
  y_pred = cross_val_predict(classifier, train_X, train_y, cv=kFolds)
  # classification report compares predictions from the k fold test sets with␣
 ↪the gold
  print(classification_report(train_y, y_pred))
  # confusion matrix from same
```

```
  cm = confusion_matrix(train_y, y_pred)
  #print_cm(cm, labels)
  print('\n')
  print(pandas.crosstab(train_y, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'], margins=True))

  """this is shte code if running on Prompt"""
#             # use a main so can get feature file as a command line argument
#             if __name__ == '__main__':
#                 # Make a list of command line arguments, omitting the [0]␣
 ↪element
#                 # which is the script itself.
#                 args = sys.argv[1:]
#                 if not args:
#                     print ('usage: python run_sklearn_model_performance.py␣
 ↪[featurefile]')
#                     sys.exit(1)
#                 infile = args[0]
#                 process(infile)

"""Final running of the model and classifier """
outpath = 'C:
 ↪\\Users\\17574\\Desktop\\ist664+NLP\\WK+x+Final+Project\\finalproject_DATA\\SemEval2014Twee
 ↪tsv'
process(outpath)

"""-----------------------------------------------------------------------
  MANUAL RUNNING OF THE TRAIN-TEST WITH LOGISTIC REGRESSION
-----------------------------------------------------------------"""
train = pd.read_csv('mytrain.tsv')  #dont mess with names please
test = pd.read_csv('mytest.tsv')

# number of folds for cross-validation
kFolds = 5
# read in the file with the pandas package
#           train_set = pandas.read_csv(outpath)
# this is a data frame for the data
print ('Shape of feature data - num instances with num features + class label')
print (train.shape)
print (test.shape)
# convert to a numpy array for sklearn
train_array = train.values
# get the last column with the class labels into a vector y
train_y = train_array[:,-1]
# get the remaining rows and columns into the feature matrix X
train_X = train_array[:,:-1]
train_y.shape #labels
```

```
train_X.shape #tweets
#             y=train['Sentiment'].values  #LABLES
#             X=train['Phrase'].values
#             from sklearn.model_selection import train_test_split
#             X_train, X_test, y_train, y_test = train_test_split(X,y,␣
 ↪test_size=.01)
#             print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
#             X_train
#             y_train
  #  ** choose one of these classifiers **
  #print '** Results from Linear SVM'
  # now call sklearn with SVC to get a model
  #classifier = LinearSVC(C=1, penalty='l1', dual=False, class_weight='auto')
  #print '** Results from Naive Bayes'
  #classifier = MultinomialNB()
print ('** Results from Logistic Regression with liblinear')
  #print '** Results from Logistic Regression with newton-cg'
  #print '** Results from Logistic Regression with lbfgs'
  ## solver options:  solver : {'newton-cg', 'lbfgs', 'liblinear'}
  ## multi-class options: multi_class : str, {'ovr', 'multinomial'}
  #but multinomial only for lbfgs
classifier =␣
 ↪LogisticRegression(class_weight='balanced',solver='lbfgs',multi_class='multinomial')
y_pred = cross_val_predict(classifier, train_X, train_y, cv=kFolds)
  # classification report compares predictions from the k fold test sets with␣
 ↪the gold
print(classification_report(train_y, y_pred))
 # confusion matrix from same
cm = confusion_matrix(train_y, y_pred)
  #print_cm(cm, labels)
print('\n')
print(pandas.crosstab(train_y, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'], margins=True))


"""this is another approach"""
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
 #Out[48]: 0.753  -> bigram accuraccy one train/test split
num_folds = 5
cross_validation_accuracy(num_folds, bigram_featuresets)
num_folds = 10
cross_validation_accuracy(num_folds, bigram_featuresets)

##############################################################################
#####################################################################
##########   CREATING TRAINING AND TEST DATA SETS
```

```
""" using dr gates code to make up the environment """
######################################################################
import os
import re
import nltk
import pandas as pd
import numpy as np
import sklearn
import string
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
import matplotlib.pyplot as plt  #learnign dataframe
#############################################################
os.chdir('C:
 ↪\\Users\\17574\\Desktop\\ist664+NLP\\WK+x+Final+Project\\finalproject_DATA\\SemEval2014Twee

######################################################################
##########   NAIVE BAYNES
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.
 ↪MultinomialNB.html#sklearn.naive_bayes.MultinomialNB.fit
##look up this model you learn that it wasnt the df seperate from labels
######################################################################
## SAVE LABELS  ##IMPORTANT - CAN NOT LEAVE LABELS ON THE TEST SET
"""----------------------------gates EXCAMPLE 1----------------------------"""
train = pd.read_csv('mytrain.tsv')  #dont mess with names please
test = pd.read_csv('mytest.tsv')

#            trainDF_nolabels = trainDF.drop(['Label'], axis=1)
#            testDF = testDF.drop(["Label"], axis=1)
#            testDF.head()
#            trainDF_nolabels.head()
#            trainDF_nolabels.shape

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix

train_array = train.values
test_array = test.values
# get the last column with the class labels into a vector y
train_y_labels = train_array[:,-1] #labels
```

```
train_y_labels.shape #Out[99]: (26928,)
# get the remaining rows and columns into the feature matrix X
train_X = train_array[:,:-1] #tweet
#remove the labels
test_labels = test_array[:,-1] #labels
len(test_labels)
testDF_nolabels = test_array[:,:-1] #tweet
mymodelNB = MultinomialNB()
    #mymodelNB.fit(trainDF_nolabels, trainlabels)  #all labels need to be same
mymodelNB.fit(train_X, train_y_labels)  #all labels need to be same
prediction = mymodelNB.predict(testDF_nolabels)
print("Naive Bayes Prediction is :")
print(prediction)
prediction.shape
type(prediction)
cnf_matrix = confusion_matrix(test_labels, prediction)
print("the confusion matrix is: ")
print(cnf_matrix)
print(np.round(mymodelNB.predict_proba(testDF_nolabels),2))


"""this is another approach"""
classifier = nltk.NaiveBayesClassifier.train(train_array)
nltk.classify.accuracy(classifier, testDF_nolabels)
 #Out[48]: 0.753  -> bigram accuraccy one train/test split
num_folds = 5
cross_validation_accuracy(num_folds, bigram_featuresets)
num_folds = 10
cross_validation_accuracy(num_folds, bigram_featuresets)
```