The following is V3 with Section 3 (Experimentation) Completed After Deadline

Background:

This effort will build an understanding of natural language processing features working with a provider Twitter file. The data is from the Semantic Evaluation conference, a.k.a. SemEval, and has labeled data to help assess message labels to assist with understanding sentiment. Natural language assessment features, such as bigrams, will help build an understanding of lessons learned over the course of this semester. Some coding will include the characters "BBE" as this the author's short name when coding custom features, functions, or the like.

Step 1: Text Processing

Figure 0 illustrates the tokenizing of the Twitter text data starting with the source file "downloaded-tweeti-a.dev.dist.tsv" of which contained 3,280 labeled "tweets." This file was used for testing, building and validating programing and methods for this efforts Step 1 & 2. There was no preprocessing or filtering performed based on author's prior experience with this type of data.

Text Processing: Tokenizing and Filtering

```
""" Step-0: Import Data  | From semEval.py in final folder"""
limit = int(limitStr) #ensure check total tweets and if randomized
tweetdata = []

for line in f:
    if (len(tweetdata) < limit):
        #remove final end of line character
        line = line.strip()
          # each line has 4 items separated by tabs
          # ignore the tweet and user ids, and keep the sentiment and tweet text
        tweetdata.append(line.split('\t')[2:4])
len(tweetdata)

for tweet in tweetdata[1000:1010]:
    print (tweet)

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 conver
      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] ==
            label = 'neu'
          else:
            label = ''
      tweetdocs.append((tokens, label))

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

""" Step-2: Bag of Words   #from: classifyKaggle.crossval.py  """
```

Evidence of initial tweets with standard Cross Validation PRF code provided…

```
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(1000) #vocabulary of the most occuring
word_features = [word for (word,count) in word_items] # (below)feature sets fr
featuresets = [(document_features(d, word_features), c) for (d, c) in tweetdocs

#Line 228  classifyKaggle-crossval....
  # train classifier and show performance in cross-validation; get list labels
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)
```

```
In [956]: cross_validation_PRF(num_folds, featuresets, labels)
Each fold size: 2132
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4

Average Precision       Recall        F1      Per Label
neg          0.755       0.740     0.747
neu          0.000       0.000     0.000
pos          0.735       0.750     0.743

Macro Average Precision Recall        F1      Over All Labels
             0.497       0.497     0.497

Label Counts {'neg': 5331, 'neu': 0, 'pos': 5331}
Micro Average Precision Recall        F1      Over All Labels
             0.745       0.745     0.745
```

Figure 0

Step 2: Feature Engineering

Referring to Figure 1 the author was able to successfully engineer a feature combination including word features, bigrams, and negation words. Negation words included: ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']. The list likely could have been expanded but the author used words provided from class lectures.

Author Feature Function Engineering

Author's features included bag of words, bigram, and negation words

```
def BBE_features(document, word_features, bigram_features,negationwords):
    document_words = set(document)
    document_bigrams = nltk.bigrams(document) #from bigram feature 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
    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
```

Evidence working in production…

```
In [337]: BBE_featuresets = [(BBE_features(d, word_features, bigram_features,negationwords),
c) for (d, c) in tweetdocs]

In [338]: print(len(BBE_featuresets[0][0].keys()))  # number of features for document 0
4410

In [339]: print(tweetdocs[0])   # the first sentence
(['Theo', 'Walcott', 'is', 'still', 'shit', ',', 'watch', 'Rafa', 'and', 'Johnny', 'deal',
'with', 'him', 'on', 'Saturday', '.'], 'neg')

In [340]: print('num nouns', BBE_featuresets[0][0]['nouns'])  # the pos tag features for this
sentence
num nouns 6

In [341]: print('num verbs', BBE_featuresets[0][0]['verbs'])
num verbs 2

In [342]: print('num adjectives', BBE_featuresets[0][0]['adjectives'])
num adjectives 1

In [343]: print('num adverbs', BBE_featuresets[0][0]['adverbs'])
num adverbs 1

In [344]: print(BBE_featuresets[0][0]['V_NOTcare'])
False

In [345]: print(BBE_featuresets[0][0]['V_always'])
False
```

Figure 1

Referring to Figure 2 illustrates NLTK Naïve Bayes classifier train and test on author's BBE_featuresets and associated cross validation metrics for the initial data set.

This effort provided the necessary baseline for "Step 3 Experimentation" providing the structure to obtain vocabulary from the training environment for working on ***additional data sets*** to assess effectiveness of these features with label prediction and associated measurement outcomes.

Evidence of Algorithms Generating Result Scoring

Naïve Bayes

```
In [407]: len(BBE_featuresets)
Out[407]: 2742

In [408]: train_set, test_set = BBE_featuresets[1000:], BBE_featuresets[:1000]

In [409]: classifier = nltk.NaiveBayesClassifier.train(train_set)

Out[410]: 0.554
In [410]: nltk.classify.accuracy(classifier, test_set)

In [411]: classifier.show_most_informative_features(30) # V_engrossing = True
  neg    :    pos    =     20.7 : 1.0
Most Informative Features
                V_good = True              pos : neu    =     33.1 : 1.0
               V_great = True              pos : neu    =     21.9 : 1.0
                V_dont = True              neg : neu    =     21.9 : 1.0
                V_best = True              pos : neu    =     17.4 : 1.0
                 V_smh = True              neg : neu    =     15.0 : 1.0
                V_fuck = True              neg : pos    =     12.8 : 1.0
              V_always = True              neg : neu    =     12.7 : 1.0
                 V_bad = True              neg : neu    =     11.7 : 1.0
               V_hope = True               pos : neu    =     11.7 : 1.0
                 V_Why = True              neg : pos    =     11.3 : 1.0
               V_C'mon = True              neg : pos    =     11.3 : 1.0
                  V_:( = True              neg : neu    =     10.4 : 1.0
           V_Amsterdam = True              neg : neu    =     10.4 : 1.0
            V_NOThave = True               neg : neu    =     10.4 : 1.0
                V_lost = True              neg : neu    =     10.4 : 1.0
             V_excited = True              pos : neu    =     10.1 : 1.0
              V_better = True              pos : neu    =      9.8 : 1.0
               V_happy = True              pos : neu    =      9.5 : 1.0
                V_wait = True              pos : neu    =      9.0 : 1.0
             V_missed = True               neg : neu    =      9.0 : 1.0
               V_love = True               pos : neg    =      8.4 : 1.0
               V_Fuck = True               neg : pos    =      8.3 : 1.0
             V_injury = True               neg : pos    =      8.3 : 1.0
              V_Rider = True               neg : pos    =      8.3 : 1.0
             V_NOTone = True               neg : pos    =      8.3 : 1.0
              V_games = True               neg : pos    =      8.3 : 1.0
              V_Ghost = True               neg : pos    =      8.3 : 1.0
            V_awesome = True               pos : neu    =      8.3 : 1.0
                 V_tv = True               neg : neu    =      8.1 : 1.0
              V_Kinda = True               neg : neu    =      8.1 : 1.0
```

Cross Validation

```
In [352]: cross_validation_PRF(num_folds, BBE_featuresets, labels)
Each fold size: 170
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4

Average Precision       Recall          F1      Per Label
neg             0.416       0.330       0.367
neu             0.551       0.616       0.581
pos             0.546       0.546       0.546

Macro Average Precision Recall          F1      Over All Labels
                0.504       0.497       0.498

Label Counts {'neg': 154, 'neu': 389, 'pos': 307}
Micro Average Precision Recall          F1      Over All Labels
                0.525       0.539       0.530
```

Figure 2

~ ~ ~ Author Work After the Deadline (v2) Starts Here ~ ~ ~

Step 3: Part A: Experiment Base Level

When using the function "writeFeatureSets" the author experienced significant errors with features ***not*** being generated making train & test processing not possible. Different workarounds were tried including, but not limited to, modifying the "writeFeatureSets" code to skip the error, incorporating text cleaning to isolate "blank" features so remaining would generate, and finally re-writing the import of the raw tweet data with different types of cleaning for special characters and text "noise" with Regular Expressions. New Python learnings were incorporated until finally a data set would successfully execute the "writeFeatureSets" macro provided by the class professor (McCracken, 2019).

Referring to Figure 3, illustrates the raw Twitter text word cleaning that helped clean the text data for Feature function creation. The figure also illustrates a sample of the proper format for 3200 total tweets for experimentation and training. The author opted to initially "exclude" NLTK stopwords as part of the baseline analysis. A total of 18,018 words were in the training data set.

Tweet Data Text Cleaning High-level Overview

```
mylist = row.split(" ") #is this the issue here
label = mylist[-1] #ok this seperates the label
del mylist[-1]  #remove the label from the tweet
#mylist = mylist.pop()
#print(mylist)
newlist = []
for word in mylist:
    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 qu
    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
                    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)
```

3200 Tweets Properly Formatted, Cleaned, and Labeled

```
In [206]: tweetdocs[:1]
Out[206]:
[(['posgas',
   'by',
   'my',
   'house',
   'hit',
   'im',
   'going',
   'to',
   'chapel',
   'hill',
   'o',
   'sat'],
  'neu')]

In [207]: len(tweetdocs)
Out[207]: 3200
```

Figure 3

Referring to Figure 4, the training set included 1800 of the most common words, a bigram chi-square value of 400, and negation words compose the baseline. It was disappointing to have a baseline Precision, Recall, and F1 generate about 0.56 +/- 0.01 with a Naïve Bayes classifier. This suggests some fault with the text cleaning and/or other feature implementation as scores around 70% are desired. Perhaps use TF-IDF would improve on the performance measures.

Training Data Set Baseline with Bayesian Model Cross Validation Performance

```
In [151]: cross_validation_PRF(num_folds, BBE_featuresets, labels)
Each fold size: 640
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4

Average Precision       Recall          F1      Per Label
neg             0.454       0.374       0.409
pos             0.583       0.548       0.559
neu             0.574       0.662       0.612

Macro Average Precision Recall          F1      Over All Labels
                0.537       0.528       0.527

Label Counts {'neg': 521, 'pos': 1149, 'neu': 1530}
Micro Average Precision Recall          F1      Over All Labels
                0.557       0.574       0.560
```

Negation Words

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

Figure 4

Referring to Figure 5, based on the Twitter data cleaning modifications the author was able to successfully extract all the FeatureSets to a tab delimited file with writeFeatureSets code provided.

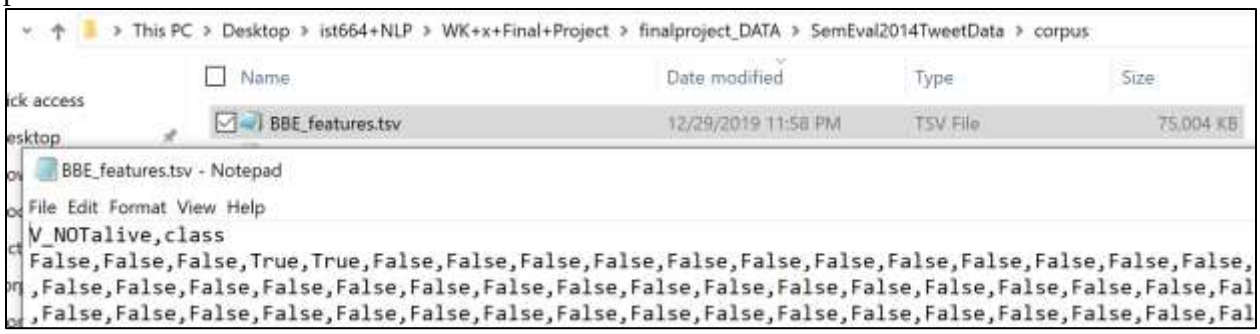

Figure 5

At this stage the author began work creating the feature data sets for the "test" data set. Initially the author was confused on how to execute the instructions provided for this project in asynchronous material "Final Project 10.4.3" but then "dawn lights on Marblehead" as it were and the author repeated the entire process above to the SemEval secondary data set provided. This resulted in train and test POS feature labeling for assessing label prediction. These files also build a bridge to layering additional tasks such as TF-IDF and\or exploring lexicon outcomes with LIWC.

Referring to Figure 6, Sci Kit Learn (SKLearn) was used building off of code provided in "run_sklearn_model_performance" generating an unsatisfactory outcome. All of the cleaned 26,928 Tweets were used but the author was having difficultly using this code to then predict on the test data set. As such the author researched and used another approach with SKLearn. This outcome also suggested that perhaps the POS features, use of negation words, and the like was insufficient approach for classifying the Tweet data.

Confusion Matrix & Evaluation Measure Comparison of Train & Test

Logistic Regression Outcome

```
In [151]: y_pred = cross_val_predict(classifier, train_X, train_y, cv=kFolds)

In [152]: train_array = train.values

In [153]: train_y = train_array[:,-1]

In [154]: train_X = train_array[:,:-1]

In [155]: train_y.shape #labels
Out[155]: (26928,)

In [156]: train_X.shape #tweets
Out[156]: (26928, 1)

In [157]: print ('** Results from Logistic Regression with liblinear')
** Results from Logistic Regression with liblinear

In [158]: classifier =
LogisticRegression(class_weight='balanced',solver='lbfgs',multi_class='multinomial')

In [159]: y_pred = cross_val_predict(classifier, train_X, train_y, cv=kFolds)

In [160]: print(classification_report(train_y, y_pred))
C:\Users\17574\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with n
predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

         neg       0.12      1.00      0.21      3122
         neu       0.00      0.00      0.00     17938
         pos       0.00      0.00      0.00      5868

    accuracy                           0.12     26928
   macro avg       0.04      0.33      0.07     26928
weighted avg       0.01      0.12      0.02     26928


In [161]: cm = confusion_matrix(train_y, y_pred)

In [162]: print(pandas.crosstab(train_y, y_pred, rownames=['Actual'], colnames=['Predicted'],
margins=True))
Predicted    neg    All
Actual
neg         3122   3122
neu        17938  17938
pos         5868   5868
```

Figure 6

Referring to Figure 7, a multinomial Naïve Bayes (MNB) approach was run SKLearn with a model being built on the 26,928 Tweet training data set with a prediction applied to the test data set. Once again, issues with the algorithm or POS tagging approaches are insufficient. The author suspects the following didn't work correctly as the source data is in Boolean format rather than converted to integer.

This resulted in the author stepping back and generating the train and test feature sets but instead of extracting the features to Excel leaving the resulting feature arrays in Spyder and running the analysis.

```
In [170]: train_array = train.values

In [171]: test_array = test.values

In [172]: train_y_labels = train_array[:,-1] #Labels

In [173]: train_y_labels.shape #Out[99]: (26928,)
Out[173]: (26928,)

In [174]: train_X = train_array[:,:-1] #tweet

In [175]: test_labels = test_array[:,-1] #labels

In [176]: len(test_labels)
Out[176]: 3200

In [177]: testDF_nolabels = test_array[:,:-1] #tweet

In [178]: mymodelNB = MultinomialNB()

In [179]: mymodelNB.fit(train_X, train_y_labels)  #all labels need to be same
Out[179]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [180]: prediction = mymodelNB.predict(testDF_nolabels)

In [181]: print("Naive Bayes Prediction is :")
Naive Bayes Prediction is :

In [182]: print(prediction)
['neu' 'neu' 'neu' ... 'neu' 'neu' 'neu']

In [183]: cnf_matrix = confusion_matrix(test_labels, prediction)

In [184]: print("the confusion matrix is: ")
the confusion matrix is:

In [185]: print(cnf_matrix)
[[   0  521    0]
 [   0 1530    0]
 [   0 1149    0]]

In [186]: print(np.round(mymodelNB.predict_proba(testDF_nolabels),2))
[[0.12 0.67 0.22]
 [0.12 0.67 0.22]
 [0.12 0.67 0.22]
 ...
 [0.12 0.67 0.22]
 [0.12 0.67 0.22]
 [0.12 0.67 0.22]]
```

Figure 7

Referring to Figure 8, feature sets data frames were created in Spyder and features were no longer extracted to Excel for running a classification experiment. This resulted in a desired outcome from which to draw conclusions and construct further efforts. NLTK was used to generate the classification accuracy. The effectiveness of the author's algorithm resulted in a low 0.478 classification accuracy. This baseline metric is was the necessary statistic necessary from which compare the implementation of other additional features.

Evaluation Measure Comparison of Train & Test

Feature Set Approach Inside Spyder

```
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)
```

```
In [265]: test_featuresets = [(BBE_features(d, word_features, bigram_features,negationwor
c) in test_clean_data]

In [266]: train_set, test_set = BBE_featuresets, test_featuresets

In [267]: classifier = nltk.NaiveBayesClassifier.train(train_set)

In [268]: nltk.classify.accuracy(classifier, test_set)
Out[268]: 0.478125
```

Figure 8

It was encouraging to have a healthy information features list from the train & test outcome with quite a varied list of pronouns, verbs, a proper noun and nonsensical word "oooooo" suggesting some additional data cleaning could be performed.

```
In [269]: classifier.show_most_informative_features(30)
Most Informative Features
                 V_alone = True              neg : pos    =     16.9 : 1.0
              V_favorite = True              pos : neg    =     14.4 : 1.0
                 V_delay = True              neg : pos    =     13.2 : 1.0
                 V_picks = True              neg : pos    =     13.2 : 1.0
               V_anymore = True              neg : pos    =     12.4 : 1.0
             V_beautiful = True              pos : neg    =     11.9 : 1.0
                 V_blame = True              neg : pos    =     10.6 : 1.0
                 V_todd = True               neg : pos    =     10.6 : 1.0
                 V_piss = True               neg : pos    =     10.6 : 1.0
                V_forgot = True              neg : pos    =     10.1 : 1.0
              V_breakout = True              neg : pos    =     10.1 : 1.0
             V_oooooooo = True               neg : pos    =     10.1 : 1.0
                 V_adria = False             neg : neu    =      9.6 : 1.0
                V_fuckin = False             neg : neu    =      9.6 : 1.0
              V_badlands = False             neg : neu    =      9.6 : 1.0
              V_omelette = False             neg : neu    =      9.6 : 1.0
                 V_cali = False              neg : neu    =      9.6 : 1.0
           V_politicians = False             neg : neu    =      9.6 : 1.0
              V_tenerife = False             neg : neu    =      9.6 : 1.0
               V_notfair = False             neg : neu    =      9.6 : 1.0
               V_adopts = False              neg : neu    =      9.6 : 1.0
                  V_swa = False              neg : neu    =      9.6 : 1.0
               V_coopts = False              neg : neu    =      9.6 : 1.0
              V_parlier = False              neg : neu    =      9.6 : 1.0
               V_clovis = False              neg : neu    =      9.6 : 1.0
            V_pointless = False              neg : neu    =      9.6 : 1.0
              V_greatest = True              pos : neg    =      9.4 : 1.0
                 V_vegas = True              pos : neg    =      9.4 : 1.0
                V_lovely = True              pos : neg    =      9.4 : 1.0
                 V_lewis = True              neg : pos    =      9.4 : 1.0
```

Figure 9

Step 3: Part B: Experiments Advanced Level

Referring to Figure 10, the author was curious if increasing the most common words from 2500 to 3000 would help improve classification. This was performed along with bigram chi-square value setting to 600. The chi-square value was set so high simply to ensure the most significant bigram scores were used in the data set given 18,018 words in the training and 8,298 words in the test data sets.

This approach resulted in a reduction in accuracy instead of an improvement. Furthermore the "most informative features" was unchanged. This suggests other types of experiments incorporating techniques such as subjectivity, LIWC, or trigrams will be necessary to assist with the classification performance.

Confusion Matrix & Evaluation Measure Comparison of Train & Test

Logistic Regression Outcome

```
In [300]: nltk.classify.accuracy(classifier, test_set)
Out[300]: 0.3575
```

```
Most Informative Features
                V_alone = True              neg : pos    =     16.9 : 1.0
             V_favorite = True              pos : neg    =     14.4 : 1.0
                V_delay = True              neg : pos    =     13.2 : 1.0
                V_picks = True              neg : pos    =     13.2 : 1.0
              V_anymore = True              neg : pos    =     12.4 : 1.0
            V_beautiful = True              pos : neg    =     11.9 : 1.0
                V_blame = True              neg : pos    =     10.6 : 1.0
                 V_todd = True              neg : pos    =     10.6 : 1.0
                 V_piss = True              neg : pos    =     10.6 : 1.0
               V_forgot = True              neg : pos    =     10.1 : 1.0
             V_breakout = True              neg : pos    =     10.1 : 1.0
           V_ooooooooo = True               neg : pos    =     10.1 : 1.0
                V_adria = False             neg : neu    =      9.6 : 1.0
               V_fuckin = False             neg : neu    =      9.6 : 1.0
             V_badlands = False             neg : neu    =      9.6 : 1.0
             V_omelette = False             neg : neu    =      9.6 : 1.0
                 V_cali = False             neg : neu    =      9.6 : 1.0
          V_politicians = False             neg : neu    =      9.6 : 1.0
             V_tenerife = False             neg : neu    =      9.6 : 1.0
              V_notfair = False             neg : neu    =      9.6 : 1.0
              V_adopts = False              neg : neu    =      9.6 : 1.0
                 V_swa = False              neg : neu    =      9.6 : 1.0
              V_coopts = False              neg : neu    =      9.6 : 1.0
             V_parlier = False              neg : neu    =      9.6 : 1.0
              V_clovis = False              neg : neu    =      9.6 : 1.0
            V_pointless = False             neg : neu    =      9.6 : 1.0
             V_greatest = True              pos : neg    =      9.4 : 1.0
                V_vegas = True              pos : neg    =      9.4 : 1.0
              V_lovely = True               pos : neg    =      9.4 : 1.0
                V_lewis = True              neg : pos    =      9.4 : 1.0
```

Figure 10

Conclusion:

The author was successful in cleaning, transforming, and building a train & test environment of semEval Twitter tweet data. Some confusion was initially experienced in piecing together class asynchronous lab material for POS feature extraction and model prediction exploration. The author's new feature function with POS tagging and negation was insufficient to assist an acceptable accuracy score with Tweet classification. However, the author was please to be able to complete the work even though it consumed 98% of the available 16 gigabytes of memory.

References:

McCracken, N, 2019. IST664 Natural Language Processing. Syracuse University

Addendum: Full Python Program

The entire program was built in Spyder v3. There are some settings at the top of the file to make the program figure out number of tweets to include, operating directories, and so forth. The author did not have the time to make this program run from command prompt and apologies for the manual settings. It is provided here as reference with all sections and has a history of the coding learned to make this effort successful.

```
"""
Created on Mon Dec 16 14:05:51 2019
@author: BBE
Class: ist664 Syracuse University Professor McCracken
DataSet: Class Final Project Twitter datasets
PUrpose:
   The program is a combination of several class programs that will have use
   Natural language processing sentiment evaluation techniques to assess a
   Twitter data file, use NLTK Naive Bayes to classify and train feature sets.
   Results also will incorporate cross-validation and run a range of
   experiments with different feature functions.
"""
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\\SemEval2014TweetData\\corpus')
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]

f = open('./downloaded-tweeti-b-dist.tsv', 'r') #  test  #[2:4])
```

```
    #'C:\\Users\\17574\\Desktop\\ist664+NLP\\WK+x+Final+Project\\finalproject_DATA\\SemEval2014TweetData\\corpus
    #downloaded-tweeti-a.dev.dist.tsv; (dont think this one!)
    #downloaded-tweeti-a.dist.tsv----------this one is for test
    #downloaded-tweeti-b.dev.dist.tsv;
    #downloaded-tweeti-b-dist.tsv

""" 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)

df_output = pd.DataFrame(tweetdata)
df_output.to_csv("rawData_mytest.csv", index=True)


#[['"neutral"',
#  'Some areas of New England could see the first flakes of the season Tuesday.'],
# ['"negative"', 'Not Available'],
# ['"neutral"', 'Not Available'],
# ['"objective-OR-neutral"', 'Not Available'],
# ['"positive"', 'Not Available'],
# ['"positive"', 'Not Available'],
# ['"objective-OR-neutral"',
#  'Tina Fey &amp;amp; Amy Poehler are hosting the Golden Globe awards on January 13. What do you think?'],
# ['"positive"',
#  'Lunch from my new Lil spot ...THE COTTON BOWL ....pretty good#1st#time#will be going back# http://t.co/Dbbj8xLZ'],
# ['"positive"',
#  "SNC Halloween Pr. Pumped. Let's work it for Sunday....Packers vs....who knows or caresn. #SNC #cheerpracticeonhalloween"]]
"""--------------------------------------------------------------------------
THIS IS WHERE WANT TO CLEAN ALL THIS STUFF
--------------------------------------------------------------------------"""
rawfilename ="rawData_mytrain.csv" #data sets: note had to manually clean
rawfilename ="rawData_mytest.csv"  #panda column ID in Excel as making faster

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()

row
mylist
newlist
label
mylabels

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

tweetdocs[:10]
len(tweetdocs)
 """-------------------------------------------------------------------------
   BBE EXTRA WORK CLEANING UP ABOVE
-------------------------------------------------------------------------"""
"""writing out to see what is wrong with the data"""
import pandas as pd
df_output = pd.DataFrame(tweetdata)
df_output.to_csv("tweet_0.csv", index=True)

for tweet in tweetdata[1000:1005]:
    print (tweet)

 """-------------------------------------------------------------------------
this IS THE PROFESSORS WAY
-------------------------------------------------------------------------"""
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)
    #(['Gas', 'by', 'my', 'house', 'hit', '$', '3.39', '!', '!', '!', "I'm",
        #'going', 'to', 'Chapel', 'Hill', 'on', 'Sat', '.', ':)'], 'pos')
    #(['Theo', 'Walcott', 'is', 'still', 'shit', ',', 'watch', 'Rafa', 'and',
        #'Johnny', 'deal', 'with', 'him', 'on', 'Saturday', '.'], 'neg')
```

```
""" Step-2: Bag of Words   #from: classifyKaggle.crossval.py  """
      """ possibly filter tokens
        # continue as usual to get all words and create word features
        # feature sets from a feature definition function
        # train and test a classifier
        # show most informative features  """

def 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))

"""----------------------------------------------------------------------------"""
# Function to compute precision, recall and F1 for each label
#  and for any number of labels
# Input: list of gold labels, list of predicted labels (in same order)
# Output: returns lists of precision, recall and F1 for each label
#      (for computing averages across folds and labels)
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)

#section 2 - adding different features
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)
       #Each fold size: 2132
       #Fold 0
       #Fold 1
       #Fold 2
       #Fold 3
       #Fold 4
       #
       #Average Precision       Recall         F1      Per Label
       #neg          0.755      0.740      0.747
       #neu          0.000      0.000      0.000
       #pos          0.736      0.750      0.743
       #
       #Macro Average Precision Recall         F1      Over All Labels
       #             0.497      0.497      0.497
       #
       #Label Counts {'neg': 5331, 'neu': 0, 'pos': 5331}
       #Micro Average Precision Recall         F1      Over All Labels
       #             0.745      0.745      0.745
"""-------------------------------------------------------------------------------------------"""
"""bigram features - from week 9   """
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)

"""----------------------------------------working NB----------"""
from nltk.corpus import sentence_polarity
# training using naive Baysian classifier, training set is approximately 90% of data
train_set, test_set = featuresets[1000:], featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
# evaluate the accuracy of the classifier
nltk.classify.accuracy(classifier, test_set)
# the accuracy result may vary since we randomized the documents
# show which features of classifier are most informative
classifier.show_most_informative_features(30)
#Most Informative Features
#           V_engrossing = True              pos : neg    =     19.0 : 1.0

"""----------------------------------------------------combined-----------COMBIEND----------"""
def PROFESSOR_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 epxeriment 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)
"""----------------------------------------------------------------------"""
    """----------------------------BBE LAND----------------------------------"""
        """----------------------------------------------------------------------"""
"""originally tried to add in week 8 but couldnt figure out getting labels for the functions from tweedocs"""

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 epxeriment 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]

"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
""" ---BBE HACKING TO MAKE SOMETHING WORK"""
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)
====================>>>>>>>>>>>>>>>>>>>>      Out[268]: 0.478125

#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)
====================>>>>>>>>>>>>>>>>>>>>      Out[268]:

"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""
"""----------------------------------------------------------------------"""

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'])
#print(BBE_featuresets[0][0]['V_NOTcare'])
#print(BBE_featuresets[0][0]['V_always'])

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) # V_engrossing = True            pos : neg    =     20.7 : 1.0

"""----------------------------step three: EXPERIMENTATION
Have further broken down into 3 sections
==> A: all the work done above; this gets exported
==> B: exporting of features
==> C: Train\Test evaluation work
being sick was only able to complete "A" and hand in as really no other recourse
Finishing for end of year B & C so can include in my learning portfolio.
Even though I feel like I am going to throw up with all that is going on I am
still very grateful for all this learning and learning how to learn again.
I am sorry I criticized and brought myself down into the dumpster and felt like
nothing was going to turn into anything given all the legal grief experienced
#when you spiral down it continues for long periods of time and can't live that way
--------------------------------------------------------------------------------
-----------PART A IS ALL MY WORK UP ABOVE--------------------------------------
-----------------------------------------OUTCOME IS:::: BBE_featuresets--------
--------------------------------------------------------------------------------
------the use of bbe_featuresets is exported for then using with skLearn------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------"""


"""----------------------------B-------------WRITE FEATURES TO CSV FILE-------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------"""
'''
  The main goal of the program is to illustrate the use of the writeFeatureSets function,
    which can be used for other data represented as feature sets to write a file
    for weka or sklearn to read for classification
  this program uses the movie reviews from the original Pang and Lee data,
    including 1000 positive and 1000 negative reviews
  for each review, we define what the NLTK calls a feature set,
    which consists of a feature dictionary, a python dictionary mapping feature names to values
    paired together with the gold label of that review
  the function writeFeatureSets will write that data structure to a csv file that
    either weka or sklearn can read

  Usage:  python save_features.py <filename or path>
'''
import sys
from nltk.corpus import movie_reviews
import random
import pandas as pd
import re
"""writing out to see what is wrong with the data"""
#df_output = pd.DataFrame(featuresets)
#df_output.to_csv("debug_today.csv", index=True)

# for testing, allow different sizes for word features
vocab_size = 100

# Function writeFeatureSets:
# takes featuresets defined in the nltk and convert them to weka input csv file
#    any feature value in the featuresets should not contain "," """ or " itself
#    and write the file to the outpath location
#    outpath should include the name of the csv file
type(featuresets)
featuresets[1]
featuresets[0][0].keys()

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\\SemEval2014TweetData\\corpus\\BBE_features.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)


tweetdocs[:1]

"""----------------------------C-----------------------------------------
------------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\\SemEval2014TweetData\\corpus\\BBE_features.tsv'
process(outpath)

"""----------------------------------------------------------------------
  MANUAL RUNNING OF THE TRAIN-TEST WITH LOGISTIC REGRESSION
  below is what i am supposed to be using but i am not getting anywhere
----------------------------------------------------------------------"""
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
#################################################################
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\\SemEval2014TweetData\\corpus')

#####################################################################
#########   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)

"""this is from the last week of asynch to see if can make work"""
train_set, test_set = train_array[5000:], train_array[:5000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
#Out[55]: 0.75
num_folds = 5
cross_validation_accuracy(num_folds, POS_featuresets)
train_set, test_set = POS_featuresets[1000:], POS_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set
```