School of Information Studies
SYRACUSE UNIVERSITY

Course:2022-0706 IST 664 Natural Language Processing

Name: Jacob N. Conard

Assignment: Final Project

Date due: 15 September 2022

Date submitted:12 September 2022

Number of Pages: 13

Final Project: NLP Final Project

Background:

The final project will be a classification task, where you will develop features for the task and demonstrate that you can carry out experiments that show which sets of features are the best for that data.

Data Description:

This dataset was produced for the Kaggle competition, described here, and which uses data from the sentiment analysis by Socher et al, detailed at this web site. The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site. Socher's group used crowd-sourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: "negative", "somewhat negative", "neutral", "somewhat positive", "positive".

Although the actual Kaggle competition is over, the data is still available here. We are going to use the training data "train.tsv", and some test data is also available "test.tsv". There appear to be 156,060 phrases in the training data file, and one of the challenges will be to choose an appropriate subset for processing and training.

Questions:

Step:1 Data Cleaning and Data Preparation

For the final project, I have two different corpora one is a training dataset related to the Rotten Tomatoes reviews and the other corpus is a testing dataset that is also related to the Rotten Tomatoes reviews. The first step that needs to be accomplished is to read the datasets in to memory for processing by the Jupyter Notebook.

Read in the Train and Test .tsv from the directory

```
In [69]: train = open("final_train.tsv", 'r')

In [70]: test = open("final_test.tsv", 'r')
```

The data is tab delineated and needs some more cleaning to only process the data that we need and make the processing faster. The label data starts with the word "phrase" so we need to skip over that, and the skip the first couple columns which are the phrase and the sentence identifiers which are not needed for processing.

Start cleaning the data

In [71]:
```
# loop over lines in the file and use the first limit of them
phrasedata = []
for x in train:
    # ignore the first line starting with Phrase and read all lines
    if (not x.startswith('Phrase')):
        x = x.strip()
        phrasedata.append(x.split('\t')[2:4])
```

In [72]:
```
phrasedata[:15]
```

Out[72]:
```
[['A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which oc
casionally amuses but none of which amounts to much of a story .',
  '1'],
 ['A series of escapades demonstrating the adage that what is good for the goose',
  '2'],
 ['A series', '2'],
 ['A', '2'],
 ['series', '2'],
 ['of escapades demonstrating the adage that what is good for the goose', '2'],
 ['of', '2'],
 ['escapades demonstrating the adage that what is good for the goose', '2'],
 ['escapades', '2'],
 ['demonstrating the adage that what is good for the goose', '2'],
 ['demonstrating the adage', '2'],
 ['demonstrating', '2'],
 ['the adage', '2'],
 ['the', '2'],
 ['adage', '2']]
```

The next step is to take a random sample of a defined number because of phrase overlapping sentences. For this example, we selected 5,000.

In [125]:
```
# pick a random sample of length limit because of phrase overlapping sequences
random.shuffle(phrasedata)
phraselist = phrasedata[:5000]
```

In [126]:
```
print('Read', len(phrasedata), 'phrases, using', len(phraselist), 'random phrases')
```

```
Read 156060 phrases, using 5000 random phrases
```

The next data cleaning step that we need to accomplish is to tokenize the sentences and convert all of the text to lowercase. This is accomplished with just a few lines of code.

Tokenize and Lower

In [98]:
```
# create list of phrase documents as (list of words, label)
phrasedocs = []

# add all the phrases

# each phrase has a list of tokens and the sentiment label (from 0 to 4)
# bin to only 3 categories for better performance
for phrase in phraselist:
    tokens = nltk.word_tokenize(phrase[0])
    phrasedocs.append((tokens, int(phrase[1])))
```

In [99]:
```
# lowercase - each phrase is a pair consisting of a token list and a label
docs = []
for phrase in phrasedocs:
    lowerphrase = ([w.lower() for w in phrase[0]], phrase[1])
    docs.append (lowerphrase)
# print a few
for phrase in docs[:10]:
    print (phrase)
```

```
(['though', 'it', "'s", 'told', 'with', 'sharp', 'ears', 'and', 'eyes', 'for', 'the', 'tenor', 'of', 'the', 'times'], 3)
(['evil', 'dead'], 2)
(['those', 'who', 'are', 'only', 'mildly', 'curious'], 2)
(['takes', 'its', 'doe-eyed', 'crudup', 'out', 'of', 'pre-9', '\\/', '11', 'new', 'york', 'and', 'onto', 'a', 'cross-country',
'road', 'trip', 'of', 'the', 'homeric', 'kind'], 2)
(['it', "'s", 'difficult', 'to', 'tell', 'who', 'the', 'other', 'actors', 'in', 'the', 'movie', 'are'], 2)
(['study', 'them'], 2)
(['play', 'off', 'each', 'other', 'virtually', 'to', 'a', 'stand-off', ','], 1)
(['in', 'their', 'own', 'idiosyncratic', 'way'], 2)
(['the', 'movie', "'s", 'heavy-handed', 'screenplay', 'navigates', 'a', 'fast', 'fade', 'into', 'pomposity', 'and', 'pretentiou
sness', '.'], 2)
(['snoozer', '.'], 0)
```

Step:2 Feature Functions and NLTK Naïve Bayes

Now that the training corpus is "clean" the next step is to produce the features in the notation of the NLTK. For this we need to write feature functions in Python as we have been doing in the lab. We need to start with the "bag-of-words" features where we collect all the words in the corpus and select some number of most frequent words to be the word features. For this example, we are going to use 1,500 of the most frequent words in the training corpus. We will print out the first 25 most frequent words from the "bag-of-words" features to get an idea of processing that may need to be completed.

Create a list of words and word features

```
In [10]: # continue as usual to get all words and create word features
         all_words_list = [word for (sent,cat) in docs for word in sent]
         all_words = nltk.FreqDist(all_words_list)
         print(len(all_words))

         7379

In [13]: # get the 1500 most frequently appearing keywords in the corpus
         word_items = all_words.most_common(1500)
         word_features = [word for (word,count) in word_items]

In [16]: for x in word_items[:25]:
             print(x[0], "\t", x[1])

         the      1573
         ,        1302
         a        1113
         and      1049
         of       1029
         to       690
         .        551
         's       504
         is       441
         in       417
         that     391
         it       369
         as       272
         with     253
         for      241
         its      233
         an       206
         film     190
         movie    190
         you      172
         but      170
         this     167
         on       155
         be       145
         his      125
```

It looks like we will need to do some stop word processing in this corpus to get rid of some of the stop words, but you can already see towards the end of the list the words "file" and "movie" are fairly common in this corpus. That is what we would expect to see considering that this corpus is related to Rotten Tomatoes reviews. The next step in the processing is to create the stop-word list and remove those stop-words from the corpus.

Create stopword list and remove stopwords

```
In [41]: stopwords = nltk.corpus.stopwords.words('english')

In [42]: stopwords.extend([',', '.',"'s", "'", '--', '```', '...', '`', '-rrb-', '-lrb-', ':'])

In [43]: negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom',

In [44]: negationwords.extend(['ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn',

In [45]: newstopwords = [word for word in stopwords if word not in negationwords]
         new_all_words_list = [word for (sent,cat) in docs for word in sent if word not in newstopwords]
```

We needed to extend the stop-words list to include some odd examples that were present in the Rotten Tomatoes corpus. We also created a list of negation words and extended that list. We are going to use this negation list later on to conduct experiments and test out what Naïve Bayes model works the best. Now we will check out the "bag-of-words" with the stop-words removed.

Create new wordlist and features with the stopwords removed

```
In [46]: # continue to define a new all words dictionary, get the 1500 most common as new_word_features
         new_all_words = nltk.FreqDist(new_all_words_list)
         new_word_items = new_all_words.most_common(1500)

In [47]: new_word_features = [word for (word,count) in new_word_items]

In [49]: for x in new_word_items[:25]:
             print(x[0], "\t\t", x[1])
```

```
film            190
movie           190
not             113
n't             106
like            103
one             102
story           73
comedy          62
much            62
good            61
characters              60
time            54
funny           54
make            53
love            49
even            49
never           48
us              48
way             46
work            46
no              44
little          43
audience                40
enough          39
movies          39
```

The next step is to create the bigram features.

Create Bigram Features

```
In [50]: finder = BigramCollocationFinder.from_words(all_words_list)
         bigram_features = finder.nbest(bigram_measures.chi_sq, 500)
```

The next thing that we need to do is create a list of functions to conduct the experiments in Step 3. These functions will be for Negation Features, Part-of-Speech Features, Original Document Features, and Bigram Document Features. We will provide screenshots of the code snippets for the functions.

Document Features Function

This function is going to take two parameters to include the document and the word features that we created previously. It is going to iterate through the all the words and make a determination if the words are present in the word features list (top 1,500).

Document Features Function

```
In [51]: 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
```

Negation Features Function

This function is very similar to the Document Features Function mentioned above; however, it is also going to include the negation words.

Negation Features Function

```
def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        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
```

Part-of-Speech Features Function

This function is very similar to the Document Features Function mentioned above; however, it is also going to do POS tagging for the words as well.

Part of Speech Features Function

```
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
```

Bigram Features Function

This function is very similar to the Document Features Function mentioned above; however, it is also going to also utilize the bigram features list that we created above.

Bigram Document Features Function

```
def bigram_document_features(document, word_features, bigram_features):
    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:
        features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
    return features
```

Cross Validation Function

This function is going to create the NLTK Naïve Bayes classifiers to be utilized to test the performance of the different models in Step 3 during the experiments. We are going to leverage this function to test how well each of the different document features performs, to include, Bigram, Negation, POS, and the Original feature set. The function sets the number of folds and utilizes a specific feature set and labels. The function will also be utilized to calculate and display the precision metrics, recall metrics, and F1 scores of each of the classifiers.

Cross Validation Function

```
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)
        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))
```

Evaluation Measures Function

This function is going to allow for the creation and presentation of the Naïve Bayes classifier metrics to include the recall metrics, precision metrics, and F1 scores for each model.

Evaluation Measures Function

```
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)
```

Step:3 Experiments

In this portion we are going to conduct experiments where we use different sets of features and compare the results of each of the different Naïve Bayes classification models to determine which one performs the best. We are going to then compare the different features we designed to determine which model performs the best in the classification tasks. We will do this with Negation, POS, Bigrams, and the Original feature set.

```
  # feature sets from feature definition functions above
featuresets = [(document_features(d, word_features), c) for (d, c) in docs]
negfeaturesets = [(NOT_features(d, word_features, negationwords), c) for (d, c) in docs]
bigramfeaturesets = [(bigram_document_features(d, word_features, bigram_features), c) for (d, c) in docs]
POSfeaturesets = [(POS_features(d, new_word_features), c) for (d, c) in docs]
# train classifier and show performance in cross-validation
# make a list of labels
label_list = [c for (d,c) in docs]
labels = list(set(label_list))    # gets only unique labels
num_folds = 10
```

Original Feature Set

```
featuresets[:1]
```

```
[({'V_the': True,
   'V_,': False,
   'V_a': False,
   'V_and': False,
   'V_of': False,
   'V_to': False,
   'V_.': False,
   "V_'s": False,
   'V_is': False,
   'V_in': False,
   'V_that': False,
   'V_it': False,
   'V_as': False,
   'V_with': False,
   'V_for': False,
   'V_its': False,
   'V_an': False,
   'V_film': False,
   'V_movie': False,
```

Negation Feature Set

```
negfeaturesets[:1]
```

```
   'V_NOTit': False,
   'V_as': False,
   'V_NOTas': False,
   'V_with': False,
   'V_NOTwith': False,
   'V_for': False,
   'V_NOTfor': False,
   'V_its': False,
   'V_NOTits': False,
   'V_an': False,
   'V_NOTan': False,
   'V_film': False,
   'V_NOTfilm': False,
   'V_movie': False,
   'V_NOTmovie': False,
   'V_you': False,
   'V_NOTyou': False,
   'V_but': False,
   'V_NOTbut': False,
   'V_this': False,
```

Bigram Feature Set

```
bigramfeaturesets[:1]
```

```
    'V_.': False,
    "V_'s": False,
    'V_is': False,
    'V_in': False,
    'V_that': False,
    'V_it': False,
    'V_as': False,
    'V_with': False,
    'V_for': False,
    'V_its': False,
    'V_an': False,
    'V_film': False,
    'V_movie': False,
    'V_you': False,
    'V_but': False,
    'V_this': False,
    'V_on': False,
    'V_be': False,
    'V_his': False,
```

Part-of-Speech Feature Set

```
POSfeaturesets[:1]
```

```
[({'contains(film)': False,
   'contains(movie)': False,
   'contains(not)': False,
   "contains(n't)": False,
   'contains(like)': False,
   'contains(one)': False,
   'contains(story)': False,
   'contains(comedy)': False,
   'contains(much)': False,
   'contains(good)': False,
   'contains(characters)': False,
   'contains(time)': False,
   'contains(funny)': False,
   'contains(make)': False,
   'contains(love)': False,
   'contains(even)': False,
   'contains(never)': False,
   'contains(us)': False,
   'contains(way)': False,
```

Cross Validation Results for Original Feature Set

```
Original Featureset
Each fold size: 500
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9

Average Precision       Recall          F1      Per Label
0             0.137      0.145      0.139
1             0.218      0.333      0.262
2             0.825      0.612      0.702
3             0.257      0.430      0.321
4             0.197      0.308      0.236

Macro Average Precision Recall          F1      Over All Labels
              0.327      0.366      0.332

Label Counts {0: 226, 1: 895, 2: 2491, 3: 1102, 4: 286}
Micro Average Precision Recall          F1      Over All Labels
              0.524      0.483      0.487
```

Cross Validation Results for Bigram Feature Set

```
Bigrams Featureset
Each fold size: 500
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9

Average Precision       Recall          F1      Per Label
0             0.137      0.145      0.139
1             0.218      0.333      0.262
2             0.825      0.612      0.702
3             0.257      0.430      0.321
4             0.197      0.308      0.236

Macro Average Precision Recall          F1      Over All Labels
              0.327      0.366      0.332

Label Counts {0: 226, 1: 895, 2: 2491, 3: 1102, 4: 286}
Micro Average Precision Recall          F1      Over All Labels
              0.524      0.483      0.487
```

Cross Validation Results for Negation Feature Set

```
Negated Featureset
Each fold size: 500
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9

Average Precision       Recall          F1      Per Label
0             0.221      0.133      0.164
1             0.218      0.341      0.265
2             0.787      0.634      0.702
3             0.268      0.436      0.331
4             0.237      0.240      0.232

Macro Average Precision Recall          F1      Over All Labels
              0.346      0.357      0.339

Label Counts {0: 226, 1: 895, 2: 2491, 3: 1102, 4: 286}
Micro Average Precision Recall          F1      Over All Labels
              0.514      0.493      0.491
```

Cross Validation Results for Part-of-Speech Feature Set

```
POS Featureset
Each fold size: 500
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9

Average Precision   Recall      F1  Per Label
0         0.124      0.141      0.131
1         0.241      0.337      0.280
2         0.824      0.623      0.709
3         0.267      0.430      0.328
4         0.172      0.286      0.210

Macro Average Precision Recall      F1  Over All Labels
          0.326      0.363      0.332

Label Counts {0: 226, 1: 895, 2: 2491, 3: 1102, 4: 286}
Micro Average Precision Recall      F1  Over All Labels
          0.528      0.488      0.494
```

Step:4 Interpretation of Results

The final step of the assignment is to take the analysis of the four Naïve Bayes classification models and interpret the results. The first step in doing an interpretation of the results is to establish a baseline of the precision, recall, and F1 measurements and what they actually are.

Precision Scores

The model precision score measures the proportion of positively predicted labels that are actually correct. Precision is also known as the positive predictive value. Precision is used in conjunction with the recall to trade-off false positives and false negatives. Precision is affected by the class distribution. If there are more samples in the minority class, then precision will be lower. Precision can be thought of as a measure of exactness or quality. (https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/)

Recall Scores

Model recall score represents the model's ability to correctly predict the positives out of actual positives. This is unlike precision which measures how many predictions made by models are actually positive out of all positive predictions made. For example: If your machine learning model is trying to identify positive reviews, the recall score would be what percent of those positive reviews did your machine learning model correctly predict as a positive. (https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/)

F1 Scores

Model F1 score represents the model score as a function of precision and recall score. F-score is a machine learning model performance metric that gives equal weight to both the Precision and Recall for measuring its performance in terms of accuracy, making it an alternative to Accuracy metrics (it doesn't require us to know the total number of observations). It's often used as a single value that provides high-level information about the model's output quality. (https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/)

Interpretation of Results

Since the F1 score is often used as a single value that provides high-level information about the model's output quality this is the score that we will utilize to determine the model that performed the best during our experiments. The F1 score is actually a score of the precision and the recall scores, so this metric will be a great metric to test the performance of the models. We will start with the performance of the Original Feature Set, this model had a precision average of 52%, recall of 48%, and an overall F1 score of 48%. The Bigram Feature Set had a precision average of 52%, recall of 48%, and an overall F1 score of 48%. The Negation Feature Set had a precision of 51%, recall of 49%, and an overall F1 score of 49%. The Part-of-Speech Feature Set had a precision of 52.8%, recall of 48.8%, and an overall F1 score of 49.4%. The best preforming model was the Part-of-Speech Feature Set.