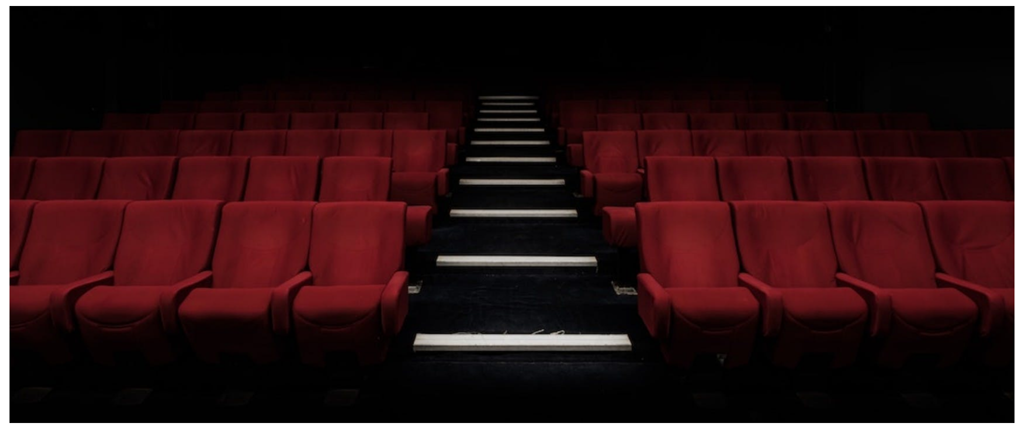
John Fields
Dr. Norma Palomino
IST664 - Final Project
Classification of Movie Reviews
December 13, 2019



Introduction

Prior to the introduction of the internet, finding reviews of movies was more challenging and time consuming than it is today. The available options were limited to talking to family, friends or reading movie reviews in newspapers or magazines. Today, prior to going to the cinema or watching a movie at home, there are a variety of sources for feedback on movies such as IMdb.com. As this data became more widely available, scholars such as Bo Pang and Lillian Lee became interested in using Natural Language Processing (NLP) to classify the sentiment in these reviews.

One of the more interesting datasets with movie reviews was collected in 2005 by Pang and Lee for their academic paper, *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.*[i] Pang and Lee had human annotators review the text of the movie reviews and assign them to the following sentiment categories:

0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 - positive

The goal for this assignment is to utilize various NLP techniques to predict the labels by training models to achieve the maximum accuracy as measured by precision, recall and F1 scores (see Section 3 for definitions of these terms).

Data and Programs

The data utilized for this analysis was from Kaggle.com and contains movie review phrases and sentiment for the following:

- Training Data (train.tsv) = 156,000 examples
- Testing Data (test.tsv) = 66,300 examples

Several Python programs were provided for students to begin the assignment. Below is a summary of the programs provided:

- classifykaggle.py - reads in the movie review phrases/labels and prints sample phrases
- classifykaggle.crossval.py - has similar code to the program above but also includes classification code using cross validation
- sentiment_read_liwc_pos_neg_words.py - contains a function to get positive and negative emotion words from the Linguistic Inquiry and Word Count (LIWC) resource from James Pennebaker
- sentiment_read_subjectivity.py - contains a function to read the subjectivity lexicon file from Wiebe et al at http://www.cs.pitt.edu/mpqa (part of the Multiple Perspective QA project)
- savefeatures.py - creates a features.csv file from the features generated by the classifykaggle.crossval.py file.
- run_sklearn_model_performance - utilizes external classifiers in the Sci Kit Learn module of Python by importing data created by the savefeatures.py file. To run this file, use the following command:
    - python run_sklearn_model_performance.py <insert path to features.csv here>

In addition to the programs provided above, one additional Jupyter notebook file was developed by the author and another notebook from fastai was utilized for the advanced task.

- IST664_FinalProject_EDA.ipynb - this notebook was developed to do Exploratory Data Analysis (EDA) for the dataset
- fastai_with_transformers_bert_roberta.ipynb - this tutorial was developed by fastai for NLP processing with deep learning (BERT). This program will be utilized for the advanced task in section 9.

Analysis

<u>Methodology</u>

1. Exploration of the data
2. Read in the phrase data
    a. Randomize
    b. Tokenize
    c. Filter tokens
    d. Create the "phrasedocs" variable
    e. Print examples
3. Set a baseline for classification
4. Explore pre-processing options
5. Explore other tokenization options
6. Generate feature sets – bi-grams, part-of-speech (POS), sentiment, subjectivity and negation
7. Export feature sets to CSV
8. Experimentation and analysis
9. Analyze with BERT neural network
10. Create test submission file and submit to Kaggle

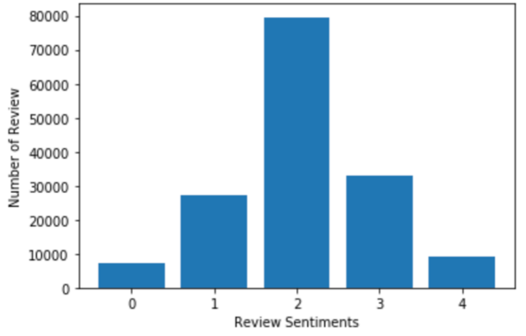
1. Exploration of the Data

Prior to running the Python programs, the train.tsv data was explored using Jupyter notebook for Exploratory Data Analysis (EDA). Figure 1 below shows the first five rows of information in the training data set.

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

Figure 1 - Movie review dataset sample - train.tsv

Next, the distribution of sentiment scores was analyzed to determine if the dataset was balanced. As shown in Figure 2, there is not an equal number of reviews and there is an imbalance with many more "Neutral" reviews in Category 2. With fewer negative (0) and positive (4) reviews, it may be more difficult for the models to classify these review sentiments.



0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 - positive

Figure 2 - Graph of sentiment reviews by rating

The last step in the EDA is to look at the typical number of word in each phrase. Figure 3 shows that 95% of the phrases have less than 22 words.

```
Minimum word count required to include all words in 95.0% of the reviews: 22.0
```

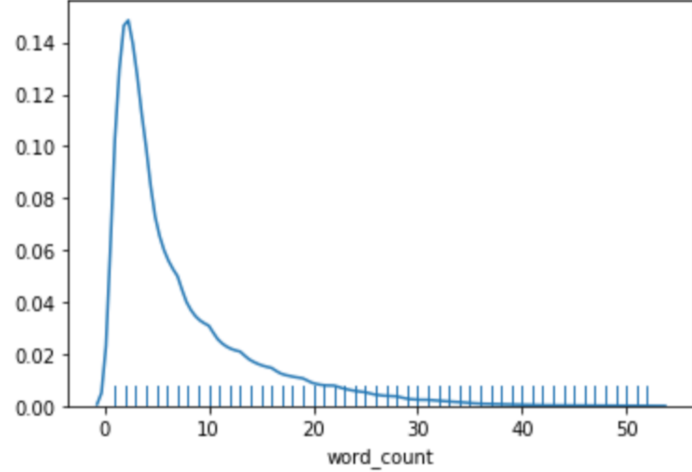


Figure 3 - Word count in the phrases

2. Read in the Phrase Data

Running the file classifykaggle.py with 1000 random phrases generated the example in Figure 4 showing the first five phrases.

```
Read 156060 phrases, using 1000 random phrases
```

```
['the like', '2']
['partly satisfying', '3']
['unholy mess', '1']
["'re in All of Me territory again", '2']
['bathroom breaks', '2']
```
Figure 4 - Sample of phrases with sentiment score

3. Set a Baseline for Classification

Baselines for classification will be set using the classifykaggle.crossval.py file to determine the Precision, Recall and F1 values. Cross-validation was chosen as the preferred training and testing procedure since it reduces the chances of overfitting the data. A separate test.tsv dataset was also provided, and this will be submitted to Kaggle.com using BERT in Section 10. Accuracy will be also be assessed using an external classifier and this will be discussed in Section 8 -Experimentation and Analysis.

Scoring definitions from Jurafsky and Martin (quotations from their book are *italicized*)[ii]:

***Precision** measures the percentage of the items that the system detected (i.e., the system labeled as positive) that are in fact positive (i.e., are positive according to the human gold lables).*

Precision = true positives / (true positives + false positives)

***Recall** measures the percentage of items actually present in the input that were correctly identified by the system.*

Recall = true positivies / (true positives + false negatives)

***F-measure** comes from a weighted harmonic mean of precision and recall.*

F1 = 2PR / (P+R)

Note: When you have more than two classes, the macroaveraging and microaveraging techniques can be used to calculate additional F1 scores.

- *In **microaveraging**, we collect the decisions for all classes into a single contingency table, and then compute precision and recall from that table.*
- *In **macroaveraging**, we compute the performance for each class, and then average over classes.*

Macroaveraging will be utilized for scoring comparisons since the cross-validation scoring utilized this metric and this will provide easier comparisons between the NLTK classifier and external Sci Kit classifier.

***Accuracy** is the percentage of all the observations our system labeled correctly. Although accuracy might seem a natural metric, we generally don't use it.* *<u>That's because accuracy doesn't work well when the classes are unbalanced.</u>*

> *Accuracy = (true positives + true negatives) / (true positives + false positives + true negatives + false negatives)*

Next, the file classifykaggle.crossval.py was run with 156,000 phrases to determine the Precision, Recall and F1 baseline measures with the default features and no external sentiment data.

```
johnfields@Johns-iMac kagglemoviereviews % python3
classifyKaggle.crossval.py corpus 156000
Read 156060 phrases, using 156000 random phrases
(['your', 'seat', 'a', 'couple', 'of', 'times'], 3)
(['like', 'a', 'travel-agency', 'video'], 1)
(['tutorial', 'service'], 2)
(['year'], 2)
(['brian', 'tufano', "'s"], 2)
(['for', 'all', 'the', 'time'], 2)
(['but', 'rather', ',', '`', 'how', 'can', 'you', 'charge', 'money',
'for', 'this', '?', "'"], 0)
(['mikes'], 2)
([',', 'amusing', 'and', 'unsettling'], 3)
(['stark', ',', 'straightforward', 'and', 'deadly'], 3)
16539
Each fold size: 31200
Fold 0
 Fold 1
Fold 2
Fold 3
Fold 4

Average  Precision    Recall       F1   Per Label
0        0.318      0.250      0.280
1        0.267      0.405      0.322
2        0.824      0.638      0.719
3        0.280      0.474      0.352
4        0.296      0.394      0.338

Macro Average Precision   Recall       F1   Over All Labels
          0.397      0.432      0.402

Label Counts {0: 7071, 1: 27262, 2: 79548, 3: 32916, 4: 9203}
Micro Average Precision   Recall       F1   Over All Labels
                0.558      0.531      0.530
```

PRECISION	RECALL	F1
.558	.531	.530

Table 1 - Base cross-validation scores with all training data

The NLTK Naive Bayes classifier was run again with 10% of the training data (15,600 examples) and resulted in slightly lower accuracy but runs must faster compared to running the entire training data set.

PRECISION	RECALL	F1
.535	.504	.505

Table 2 - Base cross-validation scores with 15,600 examples from training data

4. Explore pre-processing options

Below is a list of the most frequent word items:

```
Ten Most Common Word Items
[("'", 153), ('s', 97), ('film', 37), ('movie', 34), ('n', 27), ('t',
27), ('like', 24), ('not', 22), ('one', 22), ('story', 16)]
```

The following code was added for pre-processing to remove stopwords, punctuation, numbers and convert to lower case:

```
stopwords = nltk.corpus.stopwords.words('english')
  newstopwords = [word for word in stopwords if word not in
['not','no','can','don','t']]

  def pre_processing_documents(document):
    # "Pre_processing_documents"
    # "create list of lower case words"
    word_list = re.split('\s+', document.lower())
    # punctuation and numbers to be removed
    punctuation = re.compile(r'[-.?!/\%@,":;()|0-9]')
    word_list = [punctuation.sub("", word) for word in word_list]
    final_word_list = []
    for word in word_list:
      if word not in newstopwords:
        final_word_list.append(word)
```

```
    line = " ".join(final_word_list)
    return line
```

The following movie review examples show the types of text that will be affected by the pre-processing:

- "But it has an ambition to say something about its subjects , but **not** a willingness ."
- "The narrative is so consistently unimaginative that probably the only way to have saved the film is with the aid of those wisecracking Mystery Science Theater **3000** guys ."

This change resulted in an improvement of the F1 score from .505 to .514 and pre-processing will be used for future tests.

PRECISION	RECALL	F1
.551	.511	.514

Table 3 - Cross-validation scores after pre-processing

5. Explore other tokenization options

In addition to the nltk.word_tokenize and RegexpTokenizer commands, the WordPunctTokenizer with the pre_processing_documents was also tested with the results shown in Table 4.

PRECISION	RECALL	F1
.557	.516	.521

Table 4 - Cross-validation scores with Wordpunct tokenizer and pre-processing

Based on the test results, the WordPunctTokenizer will be used since it generated the best F1 score.

6. Generate feature sets – bi-grams, part-of-speech, sentiment, subjectivity, and negation

Functions were written to add features such as bi-grams, part-of-speech (POS), sentiment, subjectivity and negation. For this section, the added code is shown and then a table with the results of testing the new feature.

BI-GRAMS

```
## BI-GRAM FEATURES (added 11/28)
# define features that include words as before
# add the most frequent significant bigrams
# this function takes the list of words in a document as an argument
and returns a feature dictionary
# it depends on the variables word_features and bigram_features
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

#Bigrams (added 11/28)
  bigram_measures = nltk.collocations.BigramAssocMeasures()

  # create the bigram finder on all the words in sequence
  finder = BigramCollocationFinder.from_words(all_words_list)

  # define the top 500 bigrams using the chi squared measure
  # can also use raw count or PMI
  bigram_features = finder.nbest(bigram_measures.chi_sq, 500)
  print(bigram_features[:50])

  # use this function to create feature sets for all sentences
  bigram_featuresets = [(bigram_document_features(d, word_features,
bigram_features), c) for (d, c) in docs]
```

```
## TRAIN CLASSIFIER AND SHOW PERFORMANCE IN CROSS-VALIDATION - BIGRAMS
#Added 11/28
  # make a list of labels
  label_list = [c for (d,c) in docs]
  labels = list(set(label_list))    # gets only unique labels
  num_folds = 5
  cross_validation_PRF(num_folds, bigram_featuresets, labels)
```

PRECISION	RECALL	F1
.557	.518	.521

Table 5 - Cross-validation scores with bi-gram feature

PART OF SPEECH

```
## PART OF SPEECH (POS) FEATURES (added 11/28)
# this function takes a document list of words and returns a
feature dictionary
# it runs the default pos tagger (the Stanford tagger) on the
document
#   and counts 4 types of pos tags to use as 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


## POS (added 11/28)

  # define feature sets using this function
  POS_featuresets = [(POS_features(d, word_features), c) for (d, c) in
docs]
  # number of features for document 0
  print(len(POS_featuresets[0][0].keys()))
```

PRECISION	RECALL	F1
.492	.435	.424

Table 6 - Cross-validation scores with Part of Speech feature

SENTIMENT LEXICON: SUBJECTIVITY

```
## SENTIMENT LEXICON: SUBJECTIVITY COUNT
#initialize the positive, neutral and negative word lists
SLpath =
"/Users/johnfields/Desktop/kagglemoviereviews/SentimentLexicons/subjcl
ueslen1-HLTEMNLP05.tff"
#(positivelist, neutrallist, negativelist)=
sentiment_read_subjectivity.read_three_types(SLpath)


def readSubjectivity(path): #added 11/30
    flexicon = open(path, 'r')
    # initialize an empty dictionary
    sldict = { }
    for line in flexicon:
        fields = line.split()   # default is to split on whitespace
        # split each field on the '=' and keep the second part as the
value
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        # put a dictionary entry with the word as the keyword
        #     and a list of the other values
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict
```

```
# define features that include word counts of subjectivity words
# negative feature will have number of weakly negative words +
#    2 * number of strongly negative words
# positive feature has similar definition
#    not counting neutral words
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    # count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
##SUBJECTIVITY (added 11/30)
  SL = readSubjectivity(SLpath)
  SL_featuresets = [(SL_features(d, word_features, SL), c) for (d, c)
in docs]
```

PRECISION	RECALL	F1
.548	.535	.540

Table 7 - Cross-validation scores with subjectivity sentiment lexicon feature

NEGATION WORDS

```
# this list of negation words includes some "approximate negators"
like hardly and rarely
```

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

# One strategy with negation words is to negate the word following the
negation word
#   other strategies negate all words up to the next punctuation
# Strategy is to go through the document words in order adding the
word features,
#   but if the word follows a negation words, change the feature to
negated word
# Start the feature set with all 2000 word features and 2000 Not word
features set to false
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

## TRAIN CLASSIFIER AND SHOW PERFORMANCE IN NEGATION WORDS #Added
11/30
  # define the feature sets
  NOT_featuresets = [(NOT_features(d, word_features, negationwords),
c) for (d, c) in docs]\
  # make a list of labels
  label_list = [c for (d,c) in docs]
  labels = list(set(label_list))    # gets only unique labels
  num_folds = 5
  cross_validation_PRF(num_folds, NOT_featuresets, labels)
```

PRECISION	RECALL	F1
.519	.533	.522

Table 8 - Cross-validation scores with negation word feature

LIWC SENTIMENT

```
def liwc_features(doc, word_features,poslist,neglist):
      doc_words = set(doc)
      features = {}
      for word in word_features:
        features['contains({})'.format(word)] = (word in doc_words)
      pos = 0
      neg = 0
      for word in doc_words:
        if sentiment_read_LIWC_pos_neg_words.isPresent(word,poslist):
          pos += 1
        if sentiment_read_LIWC_pos_neg_words.isPresent(word,neglist):
          neg += 1
        features['positivecount'] = pos
        features['negativecount'] = neg
      if 'positivecount' not in features:
        features['positivecount']=0
      if 'negativecount' not in features:
        features['negativecount']=0
      return features

##LIWC SENTIMENT
  poslist,neglist = sentiment_read_LIWC_pos_neg_words.read_words()
  #poslist = poslist+negativewordproc(poslist)
  #neglist = neglist+negativewordproc(neglist)
  featuresets = [(document_features(d, word_features), c) for (d, c)
in docs]

## TRAIN CLASSIFIER AND SHOW PERFORMANCE IN CROSS-VALIDATION - LIWC
FEATURES #Added 12/1
  # make a list of labels
  label_list = [c for (d,c) in docs]
  labels = list(set(label_list))    # gets only unique labels
  num_folds = 5
  cross_validation_PRF(num_folds, featuresets, labels)
```

The path in sentiment_read_LIWC_pos_neg_words.py was also updated to

```
flexicon =
open('/Users/johnfields/Desktop/kagglemoviereviews/SentimentLexicons/l
iwcdic2007.dic', encoding='latin1')
```

Below are the results with the LIWC features.

PRECISION	RECALL	F1
.549	.510	.513

Table 9- Cross-validation scores with LIWC sentiment feature

Testing features for Bi-Grams, Part of Speech, Subjectivity, Negation and LIWC generated the best F1 results from the Subjectivity Sentiment Lexicon with a score of .540.

7. Export feature sets to CSV

Since the Sentiment Lexicon Subjectivity feature with pre-processing and WordPunct tokenization provided the best cross-validation scores of Precision = .548, Recall = .535 and F1 = .540 (see Table 7), these features will be exported to a CSV for analysis with external classifiers in SK Learn.

Code added to save_features.py program for the different levels in the featureset:

```
## SAVE FEATURES TO CSV FILE
for key in featurenames:
          try:
            featureline +=str(featureset[0].get(key,[]))+','
            #featureline += str(featureset[0][key]) + ','
          except KeyError:
            continue
if featureset[1] == 0:
          featureline += str("neg")
        elif featureset[1] == 1:
          featureline += str("sneg")
        elif featureset[1] == 2:
          featureline += str("neu")
        elif featureset[1] == 3:
          featureline += str("spos")
        elif featureset[1] == 4:
          featureline += str("pos")
        #featureline += featureset[1]
        # write each feature set values to the file

save_features.writeFeatureSets(featuresets,"features.csv")
```

7. Baseline analysis with Sci Kit Learn

```
#Command to run sklearn
python3 run_sklearn_model_performance.py
/Users/johnfields/Desktop/kagglemoviereviews/corpus/features.csv
```

Results for the Sci Kit Learn experimentation are shown in Table 11 in the next section.

8. Experimentation and analysis

Comparison of Precision/Recall/F1 with NLTK Naive Bayes Classifier (feature tests used Pre-Processing and WordPunct Tokenization). Note: variables and scores are highlighted to show top results.

Features	# of Examples	Folds	Classifier	Precision	Recall	F1
Baseline	156,000	5	Naive Bayes	.558	.531	.530
Baseline	15,600	5	Naive Bayes	.535	.504	.505
+Pre-processing	15,600	5	Naive Bayes	.551	.511	.514
+WordPunct Tokenization	15,600	5	Naive Bayes	.557	.516	.521
Bi-Grams	15,600	5	Naive Bayes	.557	.518	.521
Part of Speech	15,600	5	Naive Bayes	.492	.435	.424
SL - Subjectivity	15,600	5	Naive Bayes	.548	.535	.540
Negation Words	15,600	5	Naive Bayes	.519	.533	.522
SL - LIWC	15,600	5	Naive Bayes	.549	.510	.513
SL - Subjectivity	156,000	5	Naive Bayes	.572	.565	.567
SL - Subjectivity	156,000	10	Naive Bayes	.572	.565	.568

Table 10 - Comparison of results from NLTK classifier

Comparison of Precision/Recall/F1/Accuracy with Sci Kit Learn Classifiers (all tests used Pre-Processing and WordPunct tokenization). Note: Top results/attributes are highlighted. *See appendix for detailed output with confusion matrices for each test.*

Features	# of Examples	Folds	Classifier	Precision	Recall	F1	Accuracy
SL-Subjectivity	15,600	10	Logistic Regression with newton-cg	.53	.51	.51	.51
SL-Subjectivity	15,600	10	Logistic Regression with liblinear	.53	.54	.53	.54

SL-Subjectivity	15,600	10	Logistic Regression with lbfgs	.52	.49	.50	.49
SL-Subjectivity	15,600	10	Linear SVM	.52	.52	.52	.52
SL-Subjectivity	15,600	10	Multinomial Naive Bayes	.52	.56	.50	.56
SL-Subjectivity	15,600	10	Random Forest	.50	.54	.51	.54
SL-Subjectivity	39,000	10	Multinomial Naive Bayes	.54	.57	.52	.57
SL-Subjectivity	156,000	10	Multinomial Naive Bayes	n/a	n/a	n/a	n/a

Table 11 - Comparison of results from Sci Kit Learn classifiers

Note: A test with all training examples (156,000) was not possible with the external classifier. This produced an error when processing the CSV file in run_sklearn_model_performance.py. The most examples used in the external classify without an error was 39,000 (25% of training data).

The best result of the experimentation was achieved with Pre-Processing, WordPunct Tokenization, and the Subjectivity Sentiment Lexicon with the NLTK Naive Bayes classifier on all training examples with 10 folds. This resulted in an F1 score of .568.

9. Analyze with BERT neural network

In addition to the traditional NLP methods described above, the advanced task for this assignment utilizes a new deep learning technique to classify the movie reviews and then export the results from test.tsv for scoring on Kaggle.com. The deep learning technique that will be used is called BERT (Bidirectional Encoder Representation from Transformers). Google announced this new NLP innovation in October 2018 in the paper, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*[iii]. This new neural network-based technique has broken many records for classification, sentiment analysis and question-answer benchmarks during 2019. Several additional enhancements to BERT such as XLNet, Roberta, XLM and DistilBERT have continued to push the "state of the art" for several NLP tasks.

For this assignment, the Fastai tutorial was implemented using the same Kaggle movie review dataset. This code was run on Google Colab using Graphical Processing Units (GPUs) since BERT is a very memory intensive application. The results of this code are provided as an accuracy score for each "epoch" or cycle of the training:

Training	Accuracy	Time (minutes)
Learner Fit 1	.598744	06:57
Learner Fit 2	.633282	08:54
Learner Fit 3	.646034	10:30
Learner Unfreeze 1	.699795	30:29
Learner Unfreeze 2	.706908	29:07

The result of the BERT classification is a 26% increase in the accuracy score for the NLTK Naive Bayes classifier (.56) vs. BERT (.71). Accuracy scores were used for this evaluation since the BERT tutorial did not have Precision, Recall and F1 measures. In the next section, the BERT scores on the test.tsv file will be exported for evaluation on Kaggle.com.

10. Create test submission file and submit to Kaggle

The Fastai code generated the predictions.csv file that was uploaded to Kaggle.com for scoring on the test.tsv file. The resulting score was .69860 which would have placed 5th in the Kaggle competition in 2014.

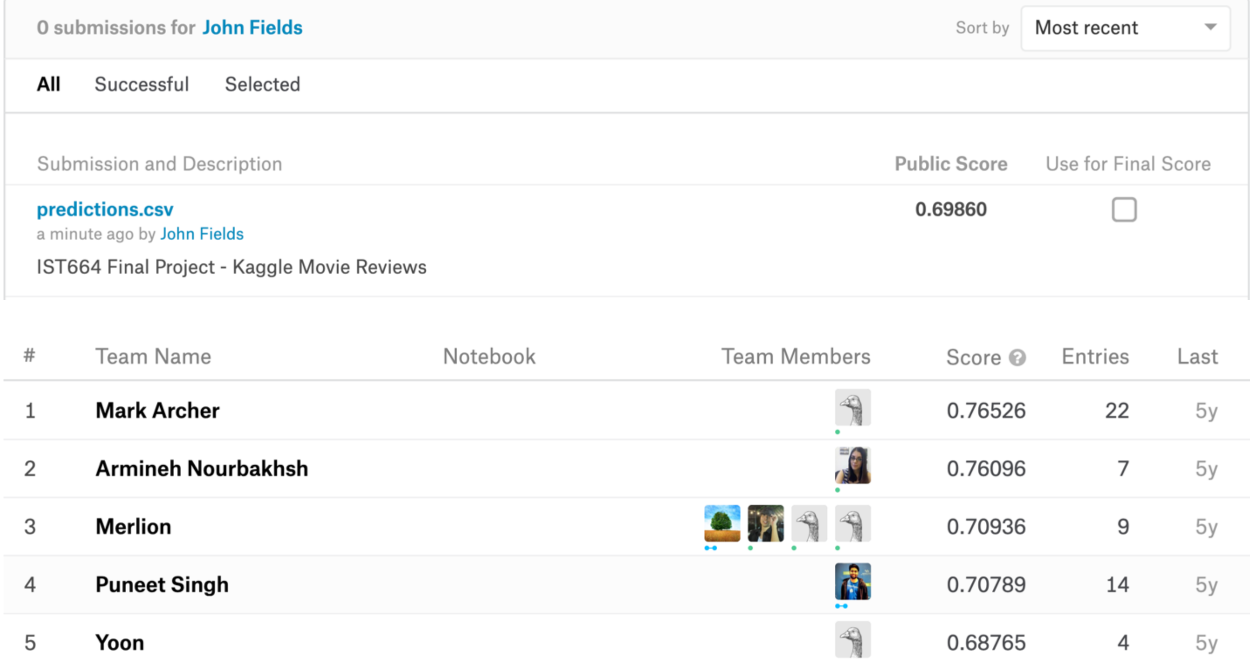


Figure 5 - Kaggle results for test.tsv predictions

Conclusion

Natural Language Processing has dramatically improved the ability of computers to be trained on large text datasets to perform tasks like classification. Understanding the skills required to pre-process a corpus of text, tokenize it and select the best features to use is a basic skill required for data scientists working in NLP. Utilizing these techniques for this assignment resulted in the top F1 score of .567 and accuracy of .56.

The advanced task using BERT deep learning provided a dramatic improvement and achieved .706908 accuracy on training data and .69860 accuracy on test data. The 26% improvement using BERT over traditional NLP methods is impressive but shouldn't preclude data scientists from learning the traditional NLP methods. Using unsupervised methods like BERT have advantages in some applications but can also introduce hidden bias that may be hidden in the pre-training corpus (e.g. Google Books and Wikipedia). There are also instances where BERT doesn't work well unsupervised and the techniques used in this paper such as pre-processing, tokenization and feature selection can be utilized in a semi-supervised manner to produce state-of-the-art results for NLP tasks.

Appendix

Details of Sci Kit Learn Classifiers

Logistic Regression with newton-cg

```
Shape of feature data - num instances with num features + class label
(15600, 1501)
** Results from Logistic Regression with newton-cg
              precision    recall  f1-score   support

         neg       0.22      0.37      0.28       730
         neu       0.69      0.66      0.68      7942
         pos       0.27      0.42      0.33       919
        sneg       0.35      0.33      0.34      2664
        spos       0.42      0.34      0.38      3345

    accuracy                           0.51     15600
   macro avg       0.39      0.43      0.40     15600
weighted avg       0.53      0.51      0.51     15600

Predicted   neg   neu   pos  sneg  spos    All
Actual
neg         271   139    27   251    42    730
neu         382  5276   319  1004   961   7942
pos          22   150   387    56   304    919
sneg        425   979   111   889   260   2664
spos        131  1146   616   318  1134   3345
All        1231  7690  1460  2518  2701  15600
```

Logistic Regression with liblinear

```
Shape of feature data - num instances with num features + class label
(15600, 1501)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.25      0.37      0.30       730
         neu       0.66      0.77      0.71      7942
         pos       0.30      0.42      0.35       919
        sneg       0.39      0.28      0.33      2664
        spos       0.45      0.30      0.36      3345

    accuracy                           0.54     15600
   macro avg       0.41      0.43      0.41     15600
weighted avg       0.53      0.54      0.53     15600

Predicted   neg   neu   pos  sneg  spos    All
Actual
```

```
neg          268   200    26   199    37    730
neu          267  6093   240   673   669   7942
pos           24   192   384    42   277    919
sneg         392  1230    90   741   211   2664
spos         110  1462   555   224   994   3345
All         1061  9177  1295  1879  2188  15600
```

<u>Logistic Regression with lbfgs</u>

```
Shape of feature data - num instances with num features + class label
(15600, 1501)
** Results from Logistic Regression with lbfgs

precision    recall  f1-score   support

         neg       0.21      0.43      0.28       730
         neu       0.70      0.62      0.66      7942
         pos       0.26      0.48      0.34       919
        sneg       0.34      0.33      0.33      2664
        spos       0.41      0.33      0.36      3345

    accuracy                           0.49     15600
   macro avg       0.38      0.44      0.39     15600
weighted avg       0.52      0.49      0.50     15600

Predicted   neg   neu   pos  sneg  spos    All
Actual
neg         313   129    33   216    39    730
neu         488  4926   392  1089  1047   7942
pos          28   129   441    52   269    919
sneg        514   883   133   873   261   2664
spos        163  1015   711   354  1102   3345
All        1506  7082  1710  2584  2718  15600
```

<u>Linear SVM</u>

```
Shape of feature data - num instances with num features + class label
(15600, 1501)
** Results from Linear SVM
              precision    recall  f1-score   support

         neg       0.21      0.37      0.27       730
         neu       0.67      0.73      0.70      7942
         pos       0.26      0.41      0.32       919
        sneg       0.37      0.27      0.31      2664
        spos       0.44      0.30      0.36      3345

    accuracy                           0.52     15600
   macro avg       0.39      0.42      0.39     15600
weighted avg       0.52      0.52      0.52     15600
```

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	269	181	27	217	36	730
neu	389	5785	318	734	716	7942
pos	35	190	379	39	276	919
sneg	433	1165	123	721	222	2664
spos	141	1365	598	244	997	3345
All	1267	8686	1445	1955	2247	15600

Multinomial Naive Bayes

Shape of feature data - num instances with num features + class label
(15600, 1501)
** Results from Naive Bayes

	precision	recall	f1-score	support
neg	0.46	0.12	0.19	730
neu	0.59	0.89	0.71	7942
pos	0.39	0.11	0.17	919
sneg	0.43	0.20	0.28	2664
spos	0.46	0.27	0.34	3345
accuracy			0.56	15600
macro avg	0.47	0.32	0.34	15600
weighted avg	0.52	0.56	0.50	15600

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	87	412	1	199	31	730
neu	17	7029	42	355	499	7942
pos	2	413	102	32	370	919
sneg	76	1889	8	546	145	2664
spos	6	2191	106	142	900	3345
All	188	11934	259	1274	1945	15600

Random Forest

	precision	recall	f1-score	support
neg	0.35	0.18	0.24	730
neu	0.61	0.79	0.69	7942
pos	0.31	0.20	0.25	919
sneg	0.39	0.25	0.31	2664
spos	0.42	0.33	0.37	3345
accuracy			0.54	15600
macro avg	0.42	0.35	0.37	15600

```
weighted avg       0.50      0.54      0.51     15600

Predicted  neg    neu  pos  sneg  spos    All
Actual
neg        135    304   12   227    52    730
neu         83   6292  108   615   844   7942
pos          6    317  186    29   381    919
sneg       143   1553   23   676   269   2664
spos        19   1772  267   174  1113   3345
All        386  10238  596  1721  2659  15600
```

Additional Resources for BERT:
https://peltarion.com/knowledge-center/tutorials/bert-movie-review-sentiment-analysis

https://www.kaggle.com/maroberti/fastai-with-transformers-bert-roberta

https://www.kaggle.com/ymcdull/bert-experiment

---

[i] Pang, B., & Lee, L. (2005). *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales*. Ithaca: Cornell University Library, arXiv.org. Retrieved from https://search-proquest-com.libezproxy2.syr.edu/docview/2091230524?accountid=14214

[ii] Jurafsky, Dan, and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*, 2018.

[iii] Devlin, J., Ming-Wei, C., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of deep bidirectional transformers for language understanding*. Ithaca: Cornell University Library, arXiv.org. Retrieved from https://search-proquest-com.libezproxy2.syr.edu/docview/2118630252?accountid=14214