Project 4 Report: Movie Review Sentiment Analysis

CS 598 Practical Statistical Learning

Graham D Chester, 5-Dec-2018

Problem Statement / Objective

The problem assigned was to predict the sentiment of three splits of IMDB movie reviews provided to an AUC level of more than 0.96 for the lowest split, while not exceeding a vocabulary count of 3000.

The problem of predicting the sentiment was split into the following steps in a Jupyter notebook (Appendix B). The movie reviews dataset and the splits dataset were read as input and the text was pre-processed to remove spurious data, this clean text was then converted to word count vectors with some transformation, and finally a vocabulary was selected from these results and saved to a text file. A logistic regression model was then applied to the three data splits provided which were limited by the vocabulary selected, and a result file was generated for each split as output, and the AUC was calculated.

The Python code submitted is effectively a subset of the notebook just described. It reads the movie reviews dataset, the splits dataset, and the vocabulary text file as input, applies the same preprocessing, tokenization, transformations and logistic regression model on the split specified by the ‘s’ variable in the code (as required), and a submission text file is created as the output.

The technology used for this project was Jupyter Notebooks, Python, with Pandas, Numpy, Scipy, and Scikit-learn libraries. The modeling was run on an iMac quad-core i7 4.2GHz, 40GB RAM, and the runtime of the Python code is approximately 20 seconds. The metric used for this report is ROC AUC as required.

Method

The method is described in sequence in the following sections, although the actual process was highly iterative with changes in one area having impacts in other areas. For example initially the stop words list had a significant impact, but once the vocabulary limit was in place, stop words made only a minor improvement and the list had to be refined.

Text Pre-processing

A visual inspection of the movie reviews dataset revealed some <br> html tags that needed to be removed, and identified options to remove numbers, punctuation etc. However a baseline was set by initially just removing the html tags. Later attempts to improve performance by removing numbers, stemming, lemmatization etc. actually reduced model accuracy so were dropped, leaving only the very simple html tag removal preprocessing.

Tokenization, Transformation and Filtering

Initially words were simply tokenized as-is, with unigrams and simple word count vectors used to establish a baseline. TFIDF transformation was found to significantly improve performance, and using a basic set of stop words also provided a minor improvement so both were included in the model. Further experimentation found that using up to bigrams improved performance, as did establishing a minimum document frequency of 20 documents, and a maximum document frequency of 30%. However filtering tokens by length actually reduced performance so this was dropped. Also using n-grams greater than 2 did have good performance but at the expense of much longer run-times and memory usage so the benefit was not worthwhile.

Vocabulary Selection

The method used to select the optimal vocabulary was to take the whole movie review dataset and apply the pre-processing, filtering and transformations as described above, and then to apply a statistical method (t-test) to select the most ‘influential’ words. The t-test applied (as recommended) compares a dataset of positive sentiment reviews with a dataset of negative sentiment reviews and generates t-statistics measuring how significantly different the sentiment is for each token. Tokens were then sorted by t-statistic magnitude and the top 2900 most significant tokens were used to create the vocabulary.

Scaled t-statistics were written to a CSV file for the word importance data visualization discussed below.

Prior using the t-test approach, L1 regularized logistic regression was used and tokens with non-zero coefficients were used to determine the vocabulary, however while the results met the 0.96 AUC, they were not as good as the t-test method results, and this method was not as statistically justified as using the t-test.

Modelling

Logistic Regression was selected from a number of other classifiers (such as multinomial naïve Bayes, XGBoost, SVM RBF kernel, SVC) for its excellent performance, and very fast run times for the volume of data provided. All other models performed worse and their run-times were many times slower. A baseline model using L1 regularization with C=1 gave reasonably good results, however tuning the model to use L2 regularization and setting C = 20 made a substantial improvement in AUC.

The model configuration and pre-processing settings are listed in Appendix A1.

Results

The minimum AUC across all splits is greater than the 0.96 required, and while the vocabulary is 2900 words (unigrams and bigrams), slightly less than this is used for the different splits due to words not appearing in all splits.

Model Performance Summary

	AUC
Performance for Split_1	0.96685
Performance for Split_2	0.96580
Performance for Split_3	0.96612
Vocabulary Size	2900

Figure 2

Experiments with vocabulary size showed that the best minimum AUC is obtained using a vocabulary size of 2999 (given the constraint on this of 3000), however the baseline minimum AUC of 0.96 across all three splits can be achieved with as few as 1200 words. The 2900 limit was chosen as a balance between these (see Appendix A3 below).

Conclusion

The performance table above shows a minimum AUC over the 3 splits of 0.96580, well over the 0.96 required. The model and pre-processing required to achieve this are both fairly simple, but it was a challenge to both select the right model/methods, and to tune the various parameters to achieve the target.

It was possible to achieve higher AUC values than shown, but only by sacrificing performance on one split to benefits the others. For example a different model had a performance of 0.961 on split one, and around 0.97 on splits 2 and 3. However the final model chosen had the highest minimum AUC based on the vocabulary size. It was also possible to increase the performance by using a vocabulary size of 2999 instead of 2900, but the improvement was very small, and so it seemed better to go with the lower vocabulary size which is well clear of the maximum allowed.

Future Steps: a broad range of models, methods and settings were evaluated to get to the current setup, so further adjustments on this setup could easily result in over-tuning. For example the stop words list could adjusted to contain only those words which improve model performance, but that would be adjusting the model to be highly specific to the current dataset. Potentially evaluating more complex models such as neural nets could result in improvements, also if additional training data was available that may also allow the model to be tuned to be a little more general.

In summary, the simplicity of the overall approach and model seems to indicate that is has not been too over-tuned to the dataset provided, but still provides a high AUC score.

Acknowledgements

All code and models are original however the idea for using t-test to determine the vocabulary was derived from the course Piazza posting by Dr. Feng at https://piazza.com/class/jky28ddlhmu2r8?cid=663

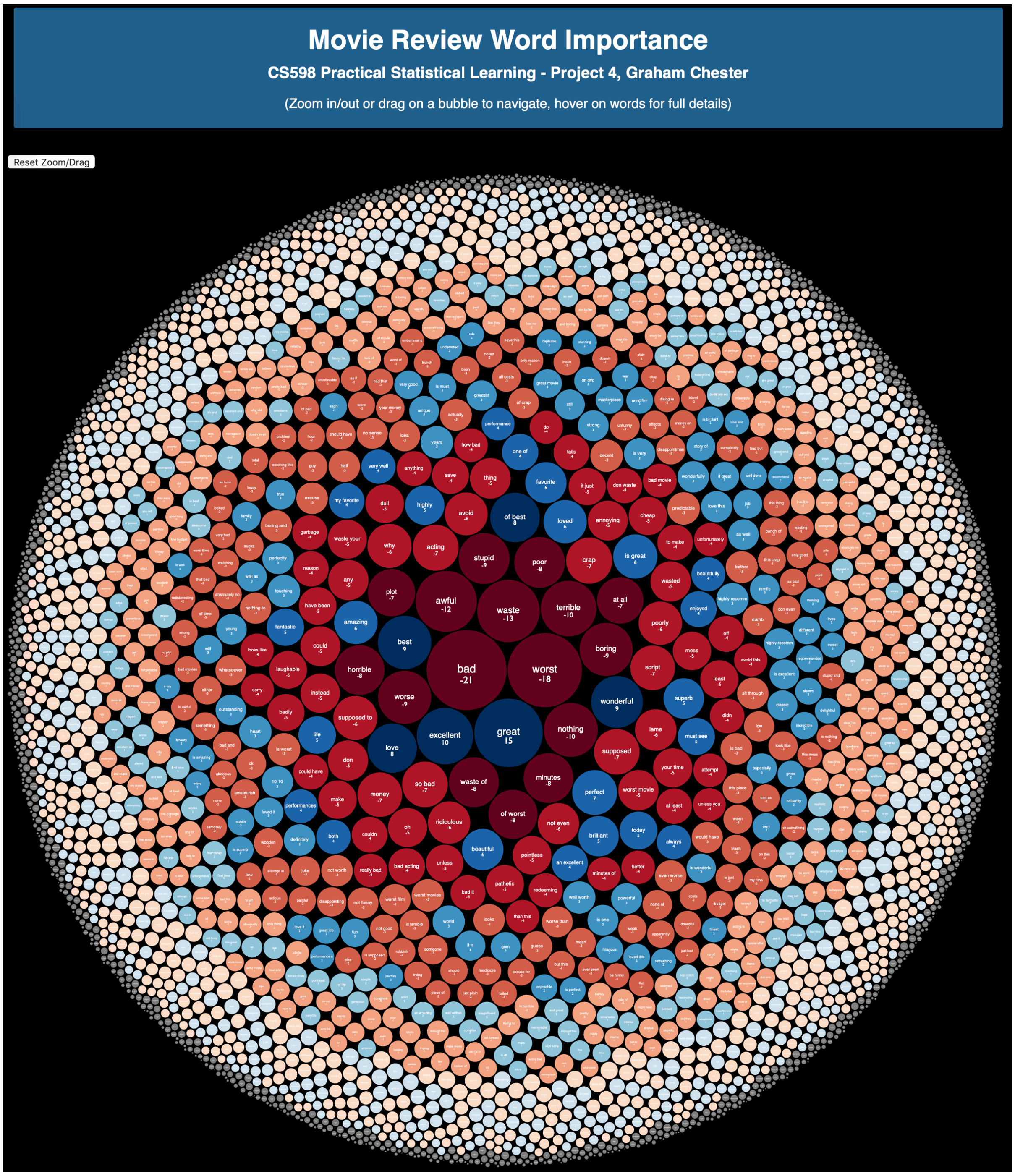
Bonus: Visualization

One of the most critical aspect of the modelling is establishing the vocabulary using word importance as described in the vocabulary selection section above. Since there are 2900 words in the vocabulary, it is a challenge to present this in an interpretable/interesting manner, however bubble charts are a good approach.

The bubble chart developed using D3 and JavaScript, uses a circle for each word, where the size of the circle represents the word importance, and the color of the circle represents whether the word is associated with negative or positive sentiment reviews. A red through white/grey to blue color spectrum is used to show words associated with negative through (almost) neutral to positive sentiment.

Since many of the bubbles are quite small, zoom in/out and drag functionality is provided (place mouse on a bubble and use the scroll wheel to zoom or left mouse button to grasp and drag). Clicking the Reset button will restore the original view. Also hovering over any text will cause a tooltip to appear after a short delay.

The visualization is available at http://gchester.com/psl4/ and a snapshot is included here for completeness.



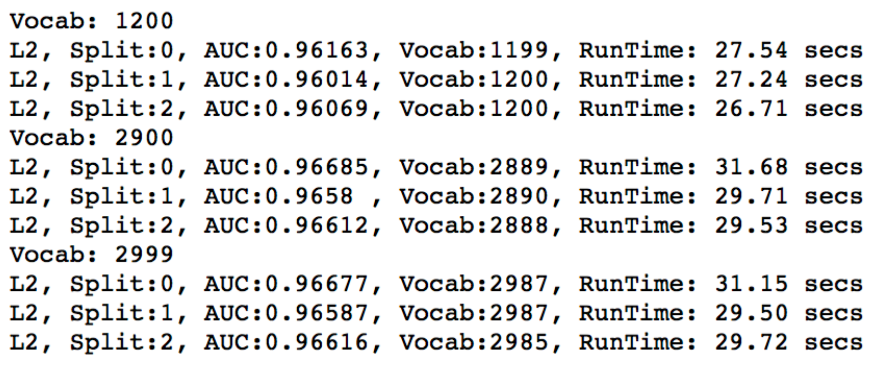
Appendix A

1) Modelling settings
Text Preprocessing: Remove <br> html tags
Stop Words: 'the', 'with', 'he', 'she', 'also', 'made', 'had', 'out', 'in', 'his', 'hers', 'there', 'was', 'then'
Vectorizer: unigrams and bigrams, minimum document frequency=20, maximum document frequency=0.3
Vocabulary size: 2900, filtering by t-test
Model: Logistic Regression, L1 regularization, C=20

2) Files Submitted
Core requirement:
Project4 Report.pdf: Project Report (this file)
mymain.py: Python source code to read data.tsv, splits.csv, vocab.txt and generate mysubmission.txt
myVocab.txt: Vocabulary file generated by Jupyter notebook (not submitted) and used by mymain.py

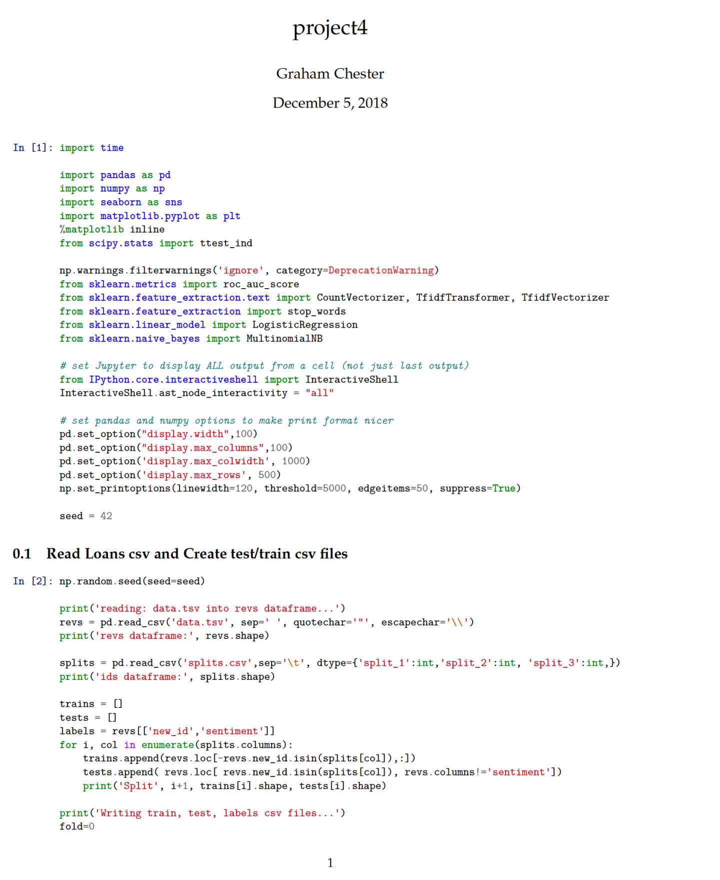
3) Visualization Bonus:
index.html: html and JavaScript using D3 library to generate bubble chart visualization
word_weights.csv: generated by Jupyter notebook, scaled version of the t-stats used for vocabulary filtering.
BubbleMapScreenShot.jpg: Screen shot of http://gchester.com/psl4/

4) Performance by Vocabulary Size
The results below show that 1200 is the smallest vocabulary size to achieve the minimum AUC, 2999 is the vocabulary size which achieves the best AUC, and 2900 is the vocabulary size (submitted) that balances very good AUC with vocabulary size.



```
Vocab: 1200
L2, Split:0, AUC:0.96163, Vocab:1199, RunTime: 27.54 secs
L2, Split:1, AUC:0.96014, Vocab:1200, RunTime: 27.24 secs
L2, Split:2, AUC:0.96069, Vocab:1200, RunTime: 26.71 secs
Vocab: 2900
L2, Split:0, AUC:0.96685, Vocab:2889, RunTime: 31.68 secs
L2, Split:1, AUC:0.9658 , Vocab:2890, RunTime: 29.71 secs
L2, Split:2, AUC:0.96612, Vocab:2888, RunTime: 29.53 secs
Vocab: 2999
L2, Split:0, AUC:0.96677, Vocab:2987, RunTime: 31.15 secs
L2, Split:1, AUC:0.96587, Vocab:2987, RunTime: 29.50 secs
L2, Split:2, AUC:0.96616, Vocab:2985, RunTime: 29.72 secs
```

Appendix B – Vocabulary Generation Notebook



project4

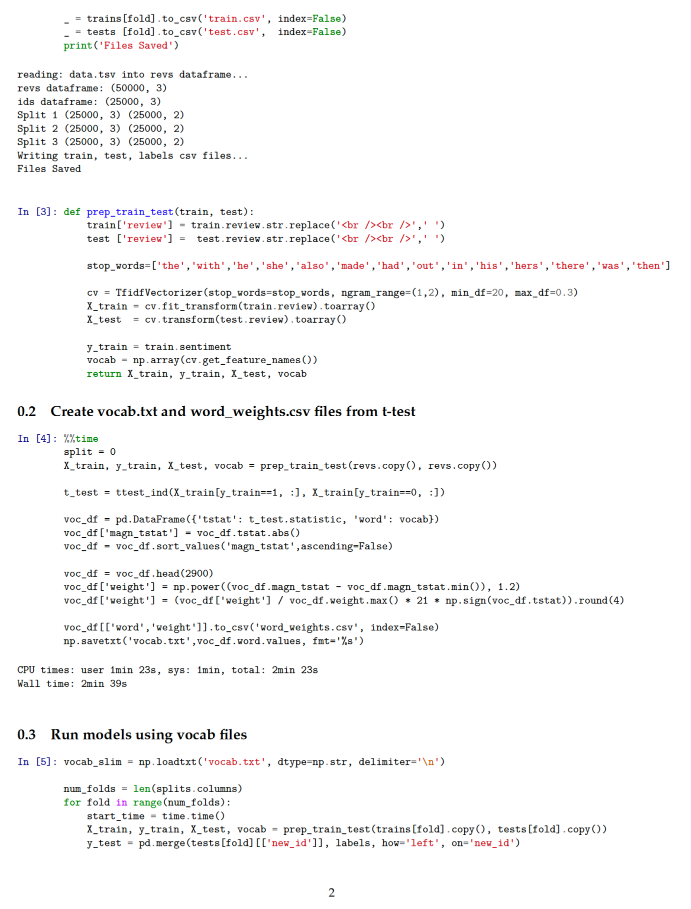
Graham Chester

December 5, 2018

```
In [1]: import time

        import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        from scipy.stats import ttest_ind

        np.warnings.filterwarnings('ignore', category=DeprecationWarning)
        from sklearn.metrics import roc_auc_score
        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
        from sklearn.feature_extraction import stop_words
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import MultinomialNB

        # set Jupyter to display ALL output from a cell (not just last output)
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"

        # set pandas and numpy options to make print format nicer
        pd.set_option("display.width",100)
        pd.set_option("display.max_columns",100)
        pd.set_option('display.max_colwidth', 1000)
        pd.set_option('display.max_rows', 500)
        np.set_printoptions(linewidth=120, threshold=5000, edgeitems=50, suppress=True)

        seed = 42
```

0.1 Read Loans csv and Create test/train csv files

```
In [2]: np.random.seed(seed=seed)

        print('reading: data.tsv into revs dataframe...')
        revs = pd.read_csv('data.tsv', sep=' ', quotechar='"', escapechar='\\')
        print('revs dataframe:', revs.shape)

        splits = pd.read_csv('splits.csv',sep='\t', dtype={'split_1':int,'split_2':int, 'split_3':int,})
        print('ids dataframe:', splits.shape)

        trains = []
        tests = []
        labels = revs[['new_id','sentiment']]
        for i, col in enumerate(splits.columns):
            trains.append(revs.loc[~revs.new_id.isin(splits[col]),:])
            tests.append( revs.loc[ revs.new_id.isin(splits[col]), revs.columns!='sentiment'])
            print('Split', i+1, trains[i].shape, tests[i].shape)

        print('Writing train, test, labels csv files...')
        fold=0
```



```
        _ = trains[fold].to_csv('train.csv', index=False)
        _ = tests [fold].to_csv('test.csv',  index=False)
        print('Files Saved')

reading: data.tsv into revs dataframe...
revs dataframe: (50000, 3)
ids dataframe: (25000, 3)
Split 1 (25000, 3) (25000, 2)
Split 2 (25000, 3) (25000, 2)
Split 3 (25000, 3) (25000, 2)
Writing train, test, labels csv files...
Files Saved
```

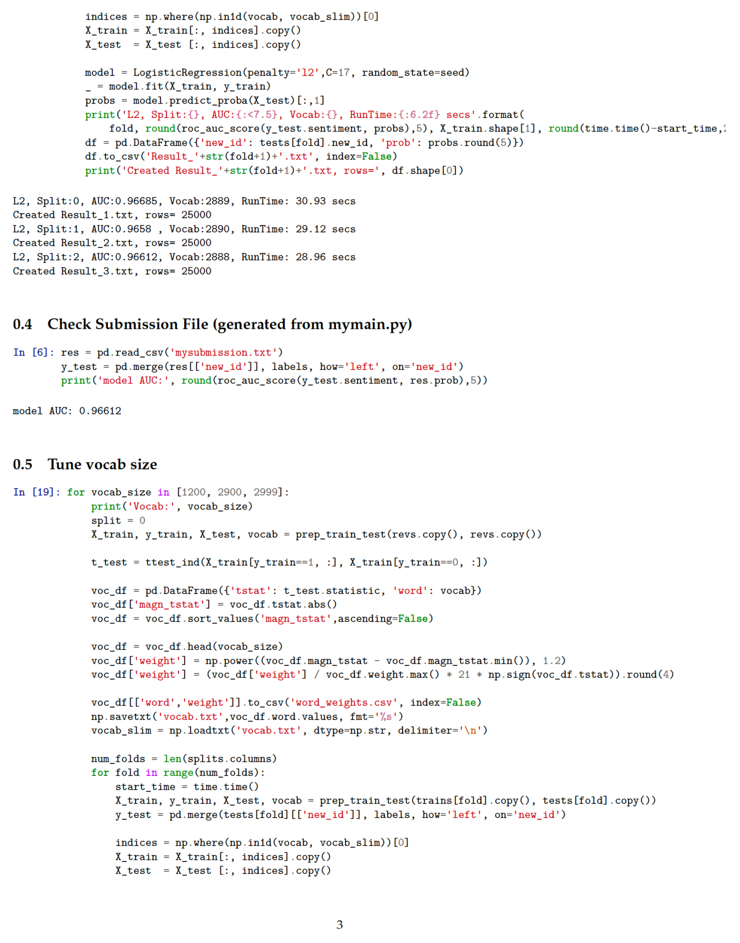
```
In [3]: def prep_train_test(train, test):
            train['review'] = train.review.str.replace('<br /><br />',' ')
            test ['review'] =  test.review.str.replace('<br /><br />',' ')

            stop_words=['the','with','he','she','also','made','had','out','in','his','hers','there','was','then']

            cv = TfidfVectorizer(stop_words=stop_words, ngram_range=(1,2), min_df=20, max_df=0.3)
            X_train = cv.fit_transform(train.review).toarray()
            X_test  = cv.transform(test.review).toarray()

            y_train = train.sentiment
            vocab = np.array(cv.get_feature_names())
            return X_train, y_train, X_test, vocab
```

0.2 Create vocab.txt and word_weights.csv files from t-test

```
In [4]: %%time
        split = 0
        X_train, y_train, X_test, vocab = prep_train_test(revs.copy(), revs.copy())

        t_test = ttest_ind(X_train[y_train==1, :], X_train[y_train==0, :])

        voc_df = pd.DataFrame({'tstat': t_test.statistic, 'word': vocab})
        voc_df['magn_tstat'] = voc_df.tstat.abs()
        voc_df = voc_df.sort_values('magn_tstat',ascending=False)

        voc_df = voc_df.head(2900)
        voc_df['weight'] = np.power((voc_df.magn_tstat - voc_df.magn_tstat.min()), 1.2)
        voc_df['weight'] = (voc_df['weight'] / voc_df.weight.max() * 21 * np.sign(voc_df.tstat)).round(4)

        voc_df[['word','weight']].to_csv('word_weights.csv', index=False)
        np.savetxt('vocab.txt',voc_df.word.values, fmt='%s')

CPU times: user 1min 23s, sys: 1min, total: 2min 23s
Wall time: 2min 39s
```

0.3 Run models using vocab files

```
In [5]: vocab_slim = np.loadtxt('vocab.txt', dtype=np.str, delimiter='\n')

        num_folds = len(splits.columns)
        for fold in range(num_folds):
            start_time = time.time()
            X_train, y_train, X_test, vocab = prep_train_test(trains[fold].copy(), tests[fold].copy())
            y_test = pd.merge(tests[fold][['new_id']], labels, how='left', on='new_id')
```



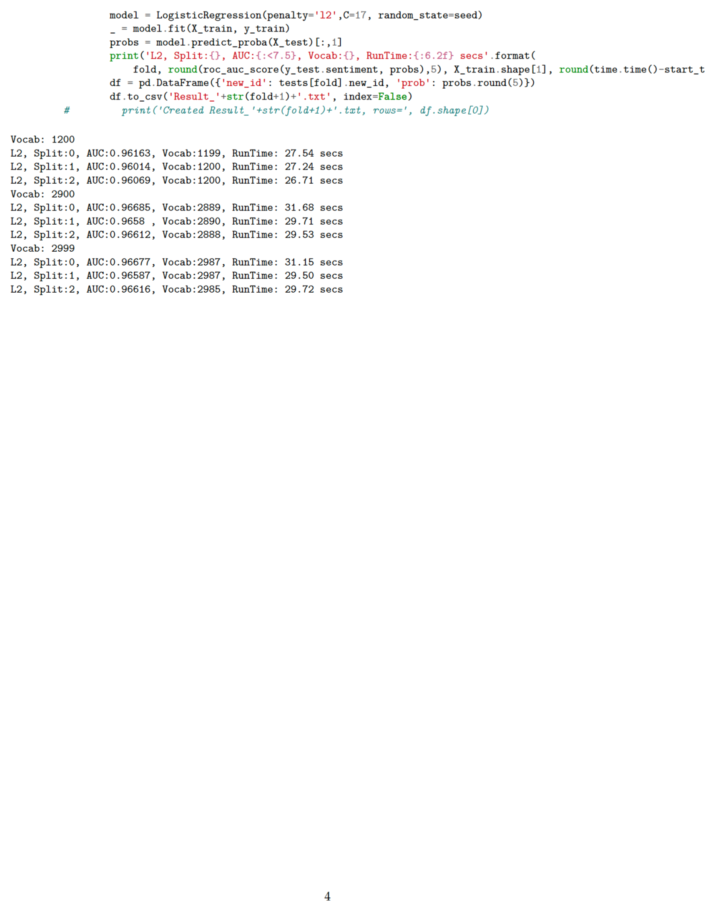
```
            indices = np.where(np.in1d(vocab, vocab_slim))[0]
            X_train = X_train[:, indices].copy()
            X_test  = X_test [:, indices].copy()

            model = LogisticRegression(penalty='l2',C=17, random_state=seed)
            _ = model.fit(X_train, y_train)
            probs = model.predict_proba(X_test)[:,1]
            print('L2, Split:{}, AUC:{:<7.5}, Vocab:{}, RunTime:{:6.2f} secs'.format(
                fold, round(roc_auc_score(y_test.sentiment, probs),5), X_train.shape[1], round(time.time()-start_time,
            df = pd.DataFrame({'new_id': tests[fold].new_id, 'prob': probs.round(5)})
            df.to_csv('Result_'+str(fold+1)+'.txt', index=False)
            print('Created Result_'+str(fold+1)+'.txt, rows=', df.shape[0])

L2, Split:0, AUC:0.96685, Vocab:2889, RunTime: 30.93 secs
Created Result_1.txt, rows= 25000
L2, Split:1, AUC:0.9658 , Vocab:2890, RunTime: 29.12 secs
Created Result_2.txt, rows= 25000
L2, Split:2, AUC:0.96612, Vocab:2888, RunTime: 28.96 secs
Created Result_3.txt, rows= 25000
```

0.4 Check Submission File (generated from mymain.py)

```
In [6]: res = pd.read_csv('mysubmission.txt')
        y_test = pd.merge(res[['new_id']], labels, how='left', on='new_id')
        print('model AUC:', round(roc_auc_score(y_test.sentiment, res.prob),5))

model AUC: 0.96612
```

0.5 Tune vocab size

```
In [19]: for vocab_size in [1200, 2900, 2999]:
             print('Vocab:', vocab_size)
             split = 0
             X_train, y_train, X_test, vocab = prep_train_test(revs.copy(), revs.copy())

             t_test = ttest_ind(X_train[y_train==1, :], X_train[y_train==0, :])

             voc_df = pd.DataFrame({'tstat': t_test.statistic, 'word': vocab})
             voc_df['magn_tstat'] = voc_df.tstat.abs()
             voc_df = voc_df.sort_values('magn_tstat',ascending=False)

             voc_df = voc_df.head(vocab_size)
             voc_df['weight'] = np.power((voc_df.magn_tstat - voc_df.magn_tstat.min()), 1.2)
             voc_df['weight'] = (voc_df['weight'] / voc_df.weight.max() * 21 * np.sign(voc_df.tstat)).round(4)

             voc_df[['word','weight']].to_csv('word_weights.csv', index=False)
             np.savetxt('vocab.txt',voc_df.word.values, fmt='%s')
             vocab_slim = np.loadtxt('vocab.txt', dtype=np.str, delimiter='\n')

             num_folds = len(splits.columns)
             for fold in range(num_folds):
                 start_time = time.time()
                 X_train, y_train, X_test, vocab = prep_train_test(trains[fold].copy(), tests[fold].copy())
                 y_test = pd.merge(tests[fold][['new_id']], labels, how='left', on='new_id')

                 indices = np.where(np.in1d(vocab, vocab_slim))[0]
                 X_train = X_train[:, indices].copy()
                 X_test  = X_test [:, indices].copy()
```



```
                 model = LogisticRegression(penalty='l2',C=17, random_state=seed)
                 _ = model.fit(X_train, y_train)
                 probs = model.predict_proba(X_test)[:,1]
                 print('L2, Split:{}, AUC:{:<7.5}, Vocab:{}, RunTime:{:6.2f} secs'.format(
                     fold, round(roc_auc_score(y_test.sentiment, probs),5), X_train.shape[1], round(time.time()-start_t
                 df = pd.DataFrame({'new_id': tests[fold].new_id, 'prob': probs.round(5)})
                 df.to_csv('Result_'+str(fold+1)+'.txt', index=False)
 #                 print('Created Result_'+str(fold+1)+'.txt, rows=', df.shape[0])

Vocab: 1200
L2, Split:0, AUC:0.96163, Vocab:1199, RunTime: 27.54 secs
L2, Split:1, AUC:0.96014, Vocab:1200, RunTime: 27.24 secs
L2, Split:2, AUC:0.96069, Vocab:1200, RunTime: 26.71 secs
Vocab: 2900
L2, Split:0, AUC:0.96685, Vocab:2889, RunTime: 31.68 secs
L2, Split:1, AUC:0.9658 , Vocab:2890, RunTime: 29.71 secs
L2, Split:2, AUC:0.96612, Vocab:2888, RunTime: 29.53 secs
Vocab: 2999
L2, Split:0, AUC:0.96677, Vocab:2987, RunTime: 31.15 secs
L2, Split:1, AUC:0.96587, Vocab:2987, RunTime: 29.50 secs
L2, Split:2, AUC:0.96616, Vocab:2985, RunTime: 29.72 secs
```