Programming Assignment 2

A significant amount of time spent on this assignment was focused on determining the *best* possible features to use for the classifier. The majority of this document describes the different feature selection methods and preprocessing of the data. Results and observations as it relates to the experiments with the different methods and hyperparameters are then discussed.

Select Feature Methods

Feature selection defines specific words or groups of words to consider when building the feature vector associated with each document. The following methods are found using the entire training set of documents (with the exception of the lexicon method).

- **All:** use the entire unique vocabulary of the training set as feature.

- **Ngram:** inspired by the lecture on collocation and the frequency detection method, the use of ngrams and their frequencies are found to determine candidate features. Specifically, bigrams and trigrams are extracted from every document and then added to a unique set iff the ngram is not composed of *only* stop words (i.e., a predetermined set of potential adpositions, determiners, and pronouns). I view this as a simple method to reproduce part-of-speech filtering. For example, the bigrams ("would", "seemingly") is kept as valid, whereas ("and", "the") is discarded. The count of every valid ngram in the training set is maintained and used to filter out those that rarely or too frequently occur. I justify this design by speculating that the rare ngrams will not generalize well as features and the common ngrams will hold little contextual value or contain words that are not in my set of stop words. By trial and error, the min/max thresholds are set to be greater than two and less than 200 for bigrams and greater than three and less than 300 for trigrams. Furthermore, by reducing the number of candidate features, we effectively reduce the number of total features and training parameters.

- **Length:** use as features the words in the vocabulary that include more than or fewer characters than specified lengths. By trial and error, a length greater than or equal to three and less than or equal to six performs best.

- **Lexicon:** a total of 70 features were predetermined and used as features. This includes the words "friend", "never", "warm", "no", "yes", "sad", "happy", and others that came to mind. These were personally chosen as words that I believe could potentially have influence over a positive or negative review.

Featurize Methods

The vector of features assigned with each document are found using the featurize function. The count and binary methods rely on the features found by selecting features (as described above), but the statistics method evaluates the properties of each document to make a discrete feature vector. These methods are as follows:

- **Count:** sum for each word that equals a candidate feature. The minimum count for a feature is zero and the maximum is N iff there all N words in the document that matches one of the candidate features.

- **Binary:** an identifiable marker for features that represents the occurrence of the word in the document. Specifically, a word matching a candidate feature is set to one if it exists, and otherwise set to zero.

- **Statistics:** eight features that are derived from the document and do not rely on the pre-selection of features from the training data. These statistics include:
  - Total count of stop words, e.g. "the", "a", etc. There are 35 *stop* words that are specified and are the same set of words used when filtering ngrams.
  - Total count of sentences as determined by the occurrences of the count of punctuation marks, i.e. ".", "?", "!", and ";".
  - Total count of predetermined *positive* words that appear in the document, e.g. "soundtrack", "home", "friend's", etc. *could* be positive.
  - Total count of predetermined *negative* words that appear in the document, e.g. "slow", "waiting", "loud", etc. *could* be negative.
  - Ratio of unique words and total count of words in the document.
  - Mean word length.
  - Standard deviation of the word length.
  - Maximum word length.

Work Stemming

A modified version of the Porter stemming algorithm, a rule-based stemmer, is employed when documents are first read in and before selecting features and/or featurizing the samples. This is done to try and lower the complexity of the document by reducing inflected words to their root form. As such, the total vocabulary decreases as multiple words can have the same stem. For example, {"waiting", "waited", "waits", "wait"} → {"wait"}. The Porter algorithm stems correctly *most* of the time, but it is not always correct. Therefore, a list of exceptions are first checked so that words like "skies" → "sky" instead of "ski". The "Exception.txt" file is not included in the submission, and coincidently will not be used if attempting to run with exception checks. In addition to stemming the documents, the predetermined lists of stop words and positive/negative words are also stemmed. The `Stemmer()` class implements the Porter algorithm from scratch using as reference a C++ version[1] that I previously wrote.

Standardizing

It is possible for the range of features to vary significantly among samples, which can negatively influence the object function with disproportionate contributions from different features. For this assignment, the features (except for the bias in the input feature vector) are standardized using z-score normalization with the mean and standard deviation calculated from the training set. Thereafter, any new test document can be standardized using the statistics from the training set.

Hyperparameter Tuning

A grid-search is preformed over a set of different epochs, batch sizes, and learning rates for each combination of feature selection and featurize methods as well as word stemming and standardization. While some manual tuning was done to select thresholds for feature methods, the parameters that were ultimately experimented with are as follows:

[1] https://github.com/stockeh/search-engine/blob/master/src/stemming.cpp

```
n_epochs    = [2, 8, 16, 32, 64, 128]
batch_sizes = [2, 4, 8, 16, 32]
etas        = [0.0001, 0.001, 0.01, 0.1]
featurize_methods       = ['count', 'binary', 'stat']
select_features_methods = ['all', 'lexicon', 'length',
                           '2gram_2_200', '3gram_2_300']
standardizes = [True, False]
stems        = [True, False]
```

The cartesian product over all parameters outline every combination of specified values to run, with the exception of the statistics method not being re-run for each select feature method. In total there are 5280 models being trained with evaluation results on the test set being written to a csv file. The number of experiments is kind of excessive, but it allows for a more thorough post-hoc analysis of hyperparameters.

Results

Bigrams with the count featurize method yields the best overall model found during my search. Using standardization and/or word stemming lowers the overall accuracy by 3-7.5% depending on the combination. There also seems to be a bias toward negative reviews (as seen the in confusion matrix and metrics). I speculate there to be a lot of zero value features and more common bigrams pertaining to one class that is skewing the prediction - especially with there being so many features.

featurize_method='count', **select_features_method**='2gram_2_200', **standardize**=False, **stem**=False, **n_features**=61103, **training_time**=2.43, **n_epochs**=8, **batch_size**=4, **eta**=0.01

```
Classes: neg: 0, pos: 1                            0     1     mean
                                                 -------------------
Confusion Matrix:                      Precision | 0.877 0.915 0.896
    0  1                               Recall    | 0.921 0.869 0.895
  -------                              F1        | 0.899 0.891 0.895
0 | 93 8
1 | 13 86                              Overall Accuracy: 89.500 %
```

Trigrams features did not perform as well as bigrams, but interestingly they performed best when first stemming words and then standardizing the feature vectors. Similar to bigrams, there is a bias toward negative reviews with a relatively low class precision.

featurize_method='binary', **select_features_method**='3gram_2_300', **standardize**=True, **stem**=True, **n_features**=49597, **training_time**=10.79, **n_epochs**=2, **batch_size**=4, **eta**=0.0001

```
Classes: neg: 0, pos: 1                            0     1     mean
                                                 -------------------
Confusion Matrix:                      Precision | 0.793 0.893 0.843
    0  1                               Recall    | 0.911 0.758 0.834
  -------                              F1        | 0.848 0.820 0.839
0 | 92 9
1 | 24 75                              Overall Accuracy: 83.500 %
```

All unique word counts performs better than the best trigram model, and even has similar precision and recall between classes, but does not preform as well as bigrams.

featurize_method=‘count’, **select_features_method**=‘all’, **standardize**=False, **stem**=False, **n_features**=45674, **training_time**=17.46, **n_epochs**=128, **batch_size**=2, **eta**=0.001

```
Classes: neg: 0, pos: 1                          0     1     mean
                                               -------------------
Confusion Matrix:                      Precision | 0.846 0.865 0.855
    0  1                               Recall    | 0.871 0.838 0.855
  -------                              F1        | 0.859 0.851 0.855
0 | 88 13
1 | 16 83                              Overall Accuracy: 85.500 %
```

Length feature candidates benefit from word stemming when counting features. This is as a result of more words being reduced to their root form and reducing the total number of model parameters and features.

featurize_method=‘count’, **select_features_method**=‘length’, **standardize**=False, **stem**=True, **n_features**=13362, **training_time**=13, **n_epochs**=128, **batch_size**=4, **eta**=0.0001

```
Classes: neg: 0, pos: 1                          0     1     mean
                                               -------------------
Confusion Matrix:                      Precision | 0.856 0.825 0.840
    0  1                               Recall    | 0.822 0.859 0.840
  -------                              F1        | 0.838 0.842 0.840
0 | 83 18
1 | 14 85                              Overall Accuracy: 84.000 %
```

Lexicon models have a very small number of parameters relative to the aforementioned methods with 70 features without stemming and 66 features when stemming. The best lexicon models all benefit from word stemming and standardizing features. There is a slight bias toward positive classes with a higher class F1 score. This suggests that maybe the predetermined features capture the positive sentiment more than negative.

featurize_method=‘count’, **select_features_method**=‘lexicon’, **standardize**=True, **stem**=True, **n_features**=66, **training_time**=9.47, **n_epochs**=64, **batch_size**=4, **eta**=0.01

```
Classes: neg: 0, pos: 1                          0     1     mean
                                               -------------------
Confusion Matrix:                      Precision | 0.777 0.736 0.756
    0  1                               Recall    | 0.723 0.788 0.755
  -------                              F1        | 0.749 0.761 0.756
0 | 73 28
1 | 21 78                              Overall Accuracy: 75.500 %
```

Statistics features as determined by each document performs better than anticipated with nearly equal class metrics. This method finds good parameters very quickly with few epochs and benefits by 3% with overall accuracy when word stemming and standardizing features relative to models using the same hyperparameters.

```
featurize_method='stats', select_features_method=None, standardize=True, stem=True,
n_features=8, training_time=9.49, n_epochs=8, batch_size=2, eta=0.01
```

```
Classes: neg: 0, pos: 1                                 0     1     mean
                                                      -------------------
Confusion Matrix:                           Precision | 0.720 0.710 0.715
    0  1                                    Recall    | 0.713 0.717 0.715
  -------                                   F1        | 0.716 0.714 0.715
0 | 72 29
1 | 28 71                                   Overall Accuracy: 71.500 %
```

Summary

The search space of possible hyperparameters, including the predetermined features and thresholds, is significantly large, and only so many combinations were explored in my experiments. Featurizing documents via the statistics method with an accompanying low learning rate, large batch size, and no stemming or normalization has an accuracy of 49%, which is worse than a random coin toss. With that said, the choice of appropriate hyperparameters can make a big difference in the performance of the model. It may be possible that there exists some hyperparameters that yield relatively better results and were not found in my experiments. Not every combination of hyperparameters are shown/discussed within, and for brevity, only the best hyperparameters are reported for each method.

In general the results show that counting existing features outperforms binary labeling, the standardizing and word stemming methods are more effective when there are fewer features/parameters, and the ngram method of selecting feature candidates performs best. However, there is some bias with the ngram method toward the negative class that is potentially a result of biased filtering. Additionally, while the best performing methods have more trainable parameters, I am still amazed by the statistics method in being able to learn to classify documents with only eight features. Lastly, with more time, it would be interesting to include more information about the class as a feature. For example, looking at class specific occurrence of words and taking the set difference of these classes and then defining new features as the count of words that belong to each class set.