Machine Learning Engineer Nanodegree

Capstone Project

I. Definition

Project Overview

The project we will work on is the sentiment classification for a large movie review dataset from IMDB. Each movie review is a variable length of words and the sentiment of each movie review must be classified.

The dataset used in this project is from the link: http://ai.stanford.edu/~amaas/data/sentiment/. This is a publicly available dataset and is used in one of the Kaggle competition. The data (IMDB movie reviews) are written in 50,000 text files. Among these, 50% are positive reviews and 50% are negative reviews - 12,500 positive reviews, and 12,500 negative reviews in each of training and testing datasets.

This kind of task belong to the domain of Nature Language Processing (NLP). What makes this problem difficult is that the sequences can vary in length, be comprised of a very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols in the input sequence. In this project, we will develop several LSTM recurrent neural network models for sequence classification problem specified, compare the performance of each model, and conclude the most suitable model for this problem.

Problem Statement

The dataset is the Large Movie Review Dataset often referred to as the IMDB dataset.

The Large Movie Review Dataset contains 25,000 highly polar moving reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given moving review has a positive or negative sentiment.

The data was also used as the basis for a Kaggle competition titled "Bag of Words Meets Bags of Popcorn" in late 2014 to early 2015.

In this project, we will develop multiple potential solutions and pick the most suitable one for this problem. NLP problems can be solved by using simple "Bag of Words" method plus simple Machine Learning classifier like Naïve Bayes. However, the subtler way to solve the NLP problem would be using word embedding plus Deep Neural Networks such as LSTM (with optional CNN plus Maxpooling layer). This is because

1) "Bag of Words" method can only utilize the statistic information of the corpus (occurrence counts of words), while word embedding may capture the characteristics of each word in higher dimension vector space and better utilize word analogies;

2) LSTM can learn either the short-term or long-term context/dependencies between symbols in the input sequences while simple Machine Learning classifiers won't able to do these.

Based on the above reasons, our solution will be a model using word embedding plus RNN (LSTM).

The major tasks in this project would be:

1) Perform data preprocessing – read in plain text from each review files, clean data – remove punctuations, remove non-alphabet tokens, remove stop words, remove short words, then save each review in a row of pandas Dataframe.
2) Perform data analysis and visualization – determine various number needed for each model: size of vocabulary, max length for each review etc.
3) Convert text to vectors – count vector for Naïve Bayes classifier for benchmark model, or word vector for Deep Neural Network models (RNN) using LSTM.
4) Create models – Bag of Words plus Naïve Bayes, word embedding (generated by Keras using vocabulary/tokens created from training set, or pre-trained Glove word representation) plus RNN.
5) Train model on training set and making predictions on testing set.
6) Fine tuning the model.
7) Compare model performance and select best model.

Text classification/sentiment analysis problem can be easily found in our daily life – Yelp review, movie review, survey for something, etc. The techniques suggested in this project can be easily applied to these applications.

Metrics

The predicting result/performance on test set will be measured by the model prediction metrics "accuracy" in Keras. This is widely used by solutions using Deep Neural Network on Keras programming framework. We just need to specify the evaluation metrics in model.compile API call and collecting the output from model.evaluate API call.

II. Analysis

Data Exploration

The dataset used in this project is from the link:
http://ai.stanford.edu/~amaas/data/sentiment/

This is a publicly available dataset. The data (IMDB movie reviews) are written in 50,000 text files, 25,000 files for training data, 25,000 files for testing data – among these, 50% are positive reviews and 50% are negative reviews – 12,500 positive reviews, and 12,500 negative reviews in each of training and testing dataset. The contents of the text files contain the review (text) part. The directory/file structure of the dataset (after untar and unzip) looks like below: **aclImdb** (root)/**train**(or **test**)/**pos**(or **neg**)/**file**(**x_y**). File is named in the format: **file_id** (**x**)– unique id for each review entry, **number**(**y**) (integer 1~4 means negative review, integer 7~10 means positive review). (Note: directory aclImdb has been renamed in my local environment – it's "input" directory now).

There will be one input file: "glove.6B.50d.txt". This is the Global Vectors for Word Representation (6B tokens, 400K vocab, uncased, 50-dimensions vector) file we will use for word embedding layers in Keras. Each line of the file is in this format: "word word_vec" – a word followed by 50 float point numbers (which are the 50-dimensions word vector).

We will visit the directories of the positive and negative reviews for the training set, then print out a few samples. There are no empty files. All the files have review text. We will have some statistic data after cleaning the text.

Observations:

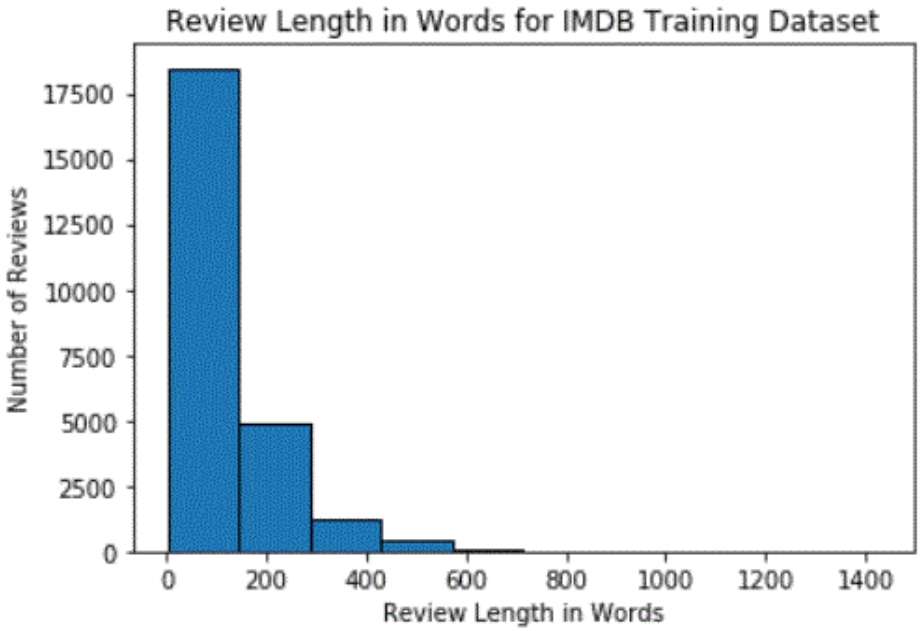
1. The review contents (text) may include punctuations, upper-case or lower-case words, non-alphabet symbols, stop words, or very short words.

2. There are not many long reviews, most of them are short reviews. We will have a histogram plot after we clean the text (after removing unwanted tokens).

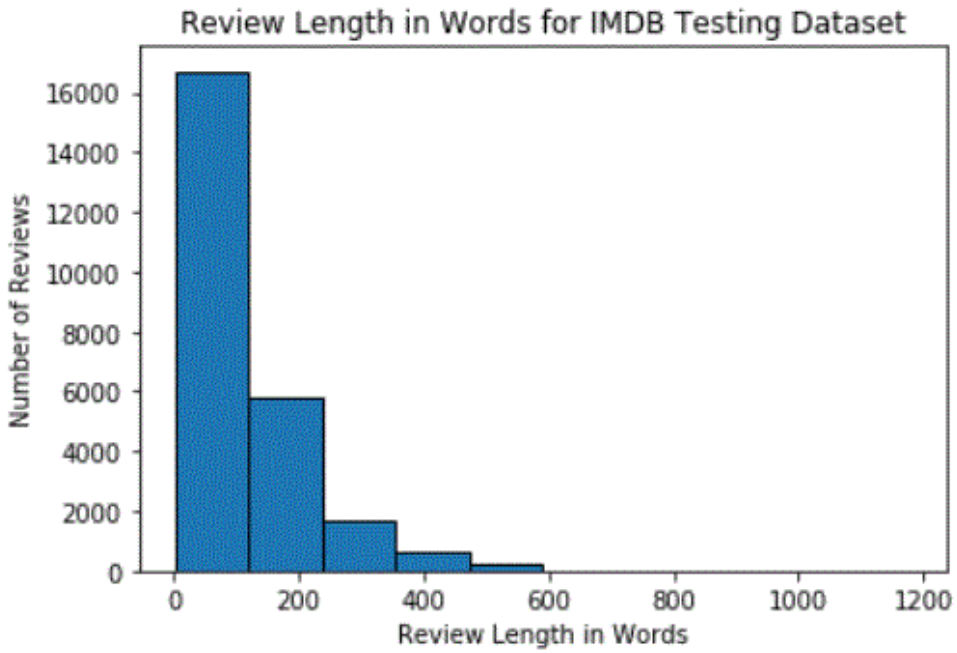
Exploratory Visualization

Below are a few plots based on the review length in the training and testing datasets for our problem. We need to determine a maximum review length for each review – maximum length of the input sequences is one of the parameters we need to pass to the Embedding API in Keras.

```
Text(0.5, 1.0, 'Review Length in Words for IMDB Training Dataset')
```



```
Text(0.5, 1.0, 'Review Length in Words for IMDB Testing Dataset')
```



Looks like most review are less than 500 words. Set input length to 500 when creating word embedding should be fine.

Algorithms and Techniques

The algorithms and techniques used in our benchmark model are: Naïve Bayes with Bag of Words.

The algorithms and techniques used in our advanced Deep Neural Network models are: word embedding, RNN, LSTM and CNN.

The algorithms and techniques used in our advanced Deep Neural Network models are: word embedding, RNN, LSTM, and CNN. We will use two different approaches to construct the word embedding layer – there will be at least one model created for each approach. One approach uses trainable word embedding layer based on vocabulary/tokens generated from the training set (after text cleaning by us, and tokenizing by Keras tools) and the other approach uses pre-trained Glove (Global Vectors for Word Representation) glove.6B.50d (6B tokens, 400K vocab, uncased, 50-dimension vector) as word embedding layer. The models for potential solutions will include the following combinations of neural network layers – embedding layer, (optional convolutional layer and Maxpooling layer), LSTM layer(s), Dropout layer, and Dense layer. We

will also create a reference models, which uses vocabulary generated/loaded by imdb.load_data API provided by Keras, to compare with our results. The training/testing data returned by imdb.load_load API are real-valued vectors that are ready to be used by Deep Neural Networks.

The Bag of Words Representation

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

To address this, scikit-learn provides utilities for the most common ways to extract numerical features from text content, namely:

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **counting** the occurrences of tokens in each document.
- **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

In this scheme, features and samples are defined as follows:

- each **individual token occurrence frequency** (normalized or not) is treated as a **feature**.
- the vector of all the token frequencies for a given **document** is considered a multivariate **sample**.

A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the **Bag of Words** or “Bag of n-grams” representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

Reference for above explanations:

http://scikit-learn.org/stable/modules/feature_extraction.html

Word Embedding

To use Deep Neural Network for text classification, we need to convert the words in reviews into real valued vectors – this is a popular technique when working with text. This is a technique where words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.

Using Pre-Trained GloVe Embedding

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. More descriptions about Glove can be found in the following 2 links:

1. https://nlp.stanford.edu/pubs/glove.pdf
2. https://nlp.stanford.edu/projects/glove/

Citing some descriptions from the first link: A few years ago, Mikolov et al. introduced an evaluation scheme based on word analogies that examining not the scalar distance between word vectors (which was the primary method to evaluate the intrinsic quality of word representations), but rather their various dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation king − queen = man − woman. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations. The two main model families for learning word vectors are:

3. global matrix factorization methods, such as latent semantic analysis (LSA)
4. local context window methods, such as the skip-gram model.

Currently, both families suffer significant drawbacks. While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts. Glove is a weighted least squares model that trains on global word-word co-occurrence counts and thus makes efficient use of statistics. Besides that, it is proved to have very good performance (high accuracy) on the word analogy dataset, which makes it an attractive choice of word embedding layer.

Recurrent Neural Networks

The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. Because of the nature that the output of current time step is depending on the output from previous time step, RNN is a good choice in predicting trend in sequence (time-series) data like stock marketing price, and many NLP tasks such as text classification (sentiment analysis), machine translation, speech recognition, etc. Our model will be based on RNN networks.

Reference for RNN:

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

LSTM Networks

Long short-term memory (LSTM) units are units of a recurrent neural network (RNN). A common LSTM unit is composed of a cell, an input (or update) gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying and making predictions based on time-series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problems that can be encountered when training traditional RNNs.

Reference for LSTM network:

https://en.wikipedia.org/wiki/Long_short-term_memory.

Convolutional Neural Network and LSTM for Sequence Classification

Convolutional neural networks excel at learning the spatial structure in input data. The IMDB review data does have a one-dimensional spatial structure in the sequence of words in reviews and the CNN may be able to pick out invariant features for good and bad sentiment. This learned spatial features may then be learned as sequences by an LSTM layer. Another

advantage of using CNN is: it will greatly reduce the training time – especially when we have a large vocabulary and large dataset.

Reference for using CNN in sentiment analysis:

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Benchmark

The benchmark result we will use in this project is the result from a Naïve Bayes classifier with "Bag of Words" algorithm. We will use CountVectorizer class in scikit-learn to fit (build a vocabulary) and transform (vectorize) the training text. We then use MultinomialNB classifier (a Naive Bayes classifier for multinomial models) to fit the training set and predict on the testing set. Naïve Bayes plus "Bag of Words" is the simplest but almost the most common model for NLP tasks. This combination should be suitable for our benchmark model.

III. Methodology

Data Preprocessing

The data preprocessing tasks include the following:

1. Read the contents from each review text file, perform text cleaning: replace non-alphabet symbols (punctuations, numbers) with white space; convert upper case to lower case; remove stop words ("the", "a", "to", "about"…), filter out short words (len = 1).
2. Update our vocabulary with the processed words/counts from step 1.
3. Create 2 pandas Dataframes – one for training set, one for testing set. Save each cleaned review text in training/testing sets in a row of one of the above Dataframes.
4. Determine the size of the vocabulary we want to use in the BOW algorithm/word embedding layer.
5. Determine the maximum review length – this number is need when preparing word embedding layer.

Implementation

For benchmark model using BOW and Naïve Bayes:

1. Use CountVectorizer class from scikit-learn to fit/transform training set, and transform testing set into vector of word counts.
2. Use MultinomialNB classifier from Naïve Bayes to perform training/predicting tasks.

We may need to repeat step 1 and 2 for different vocabulary size and determine the vocabulary size according the best test accuracy from different models.

For initial model using word embedding and Deep Neural Networks:

1. Use Tokenizer class in Keras to fit/transform text in training set and transform text in testing set into sequences of tokens (numbers or indexes), zero-pad the sequences if the input length is smaller than max review length specified (determined in step 5 in previous section).
2. Create a model with word embedding (vocabulary_size, embedding_vector_length, max_review_length), LSTM (hidden unit = 100), and Dense layer.

   Note: Embedding API turns the positive integers (indexes) into dense vectors of fixed size. The model will take as input an integer matrix of size (batch_size, max_review _length). The largest integer (i.e. word index) in the input should be no larger than the vocabulary_size. The model.output_shape == (batch_size, max_review_length, embedding_vector_length).

   Hidden unit = 100 or 128 are common choices for LSTM.

Refinement

Based on the initial solution implemented in the previous section, we implement 3 more models which uses the same trainable word embedding prepared by us:

1. Model II: word embedding, Dropout (rate=0.2), LSTM (hidden units = 100), Dropout (rate = 0.3) , Dense.

   Note: adding Dropout layer between layers is a common technique to prevent overfitting. Dropout rate 0.2 ~ 0.5 are common choices. After some experiment, I found that using rate 0.2 and 0.3 in first/second Dropout layer perform the best. Hidden unit 100, 128 are common choices for LSTM in many articles. I use 100 here.

2. Model III: word embedding, CNN (filter=32, kernel_size = 3, padding = 'same', activation='relu'), Maxpooling (pool_size=2), LSTM (hidden unit = 100), Dropout (rate = 0.3), Dense.

   Note: for CNN, filter size 32, kernel_size 2,3,5, padding = 'same', activation = 'relu' are common choices. I just stay with these common choices (set kernel_size = 3). Pool size = 2 is also a common choice for Maxpooling layer. I don't think we should use a big pool size so just use the small common choice 2.

3. Model IV: word embedding, LSTM (hidden unit = 100), Dropout (rate = 0.2), LSTM (hidden unit = 100), Dropout (rate = 0.2), Dense. This multi-LSTM layers architecture is inspired by the course "Sequence Models" by deeplearning.ai

To compare with the above 3 models, we built another model which uses the pre-trained Glove word representation. The implementation of this model includes:

1. Perform 2 preprocessing steps:
   * Read out the word vectors from the Glove file, built 3 data structures (directories) word_to_index, index_to_word, word_to_vec (word vectors)
   * Use the same Tokenizer object tokenizer created for the initial model to prepare the embedding matrix: we've got the training/testing set sequences loaded with the indexes to the same vocabulary used by other models – these indexes are ordered by the word frequency. Initialize the embedding matrix with size (vocabulary_size, embedding_vector_length)(50). For every (word, index) in the word_index dictionary of the tokenizer, if the index is less than the vocabulary_size (note: the maximum index is one less than the vocabulary_size), copy the word vector for the correspondence word from Glove into the embedding matrix. Note: the vocabulary_size may be less than the actual vocabulary size made up from the whole training set. Two good things for this model:
      - We take the advantage of using the tokenizer created earlier which already has the vocabulary, word occurrence count, indexes to the word (according to the frequency of the word) set up for us.
      - The vocabulary in our Glove file is very large. If we want to use the whole vocabulary in that file, it will take a long time to train the model.

2. Create the model:

   * The architecture of the model is: same as model II.

IV. Results

Model Evaluation and Validation

By looking at the accuracy score and training time took (reported after model training and evaluating phase), we may conclude that model 2 (LSTM plus Dropout) is the best among all the models. The test accuracy achieved 0.87336 – it's the highest among all the models. Here are more comparison detail:

1) (LSTM, Dropout) is more robust than single LSTM layer since the former does better job in preventing overfitting.
2) (LSTM, Dropout) performs better than (CNN, Maxpooling, LSTM, Dropout) in terms of accuracy, speed and overfit prevention. Although one-dimensional CNN may excel at learning the spatial structure and pick out invariant features for good and bad sentiment, sometimes it may lead to overfitting. We may find out this by comparing the differences in training and testing set accuracy between these 2 models.
3) (LSTM, Dropout) performs better than multiple (LSTM, Dropout) in terms of accuracy, and speed. The latter takes much longer besides having lower accuracy score.
4) (Trainable word embedding, LSTM, Dropout) performs better than (pre-trained Glove, LSTM, Dropout) in terms of accuracy and speed. It took the latter much longer to train, although the latter model still achieved pretty good test accuracy. Model using Glove may outperform other models if the training/testing set need to use word analogies or the training set is not too large (this means the testing set may contain data not seen in the training set) - this does not seem to apply here.

Regarding the parameters used in model 2: LSTM with 100/128 hidden units are common choices in many articles, dropout rate 0.2~0.5 are also common choices. Our parameters seem to be appropriate.

We've also applied different training set (loaded by using imdb.load_data – vocabulary and tokens are generated by Keras from original training set) and different testing set (run time input) on our model, the results are still good. These prove our final model is robust, generalized well to unseen data and hence trustable.

Justification

By comparing the accuracy score from our model with the same score from the benchmark model, we may conclude that our final model has stronger performance and is superior than the benchmark model.

In the previous sections, we've thoroughly analyzed and discussed the final solution –

1) We ensure that using word embedding layer plus LSTM is superior than traditional BOW: word embedding may capture the characteristics of each word in higher dimension vector space besides considering the word frequency and LSTM may learn from short-term or long-term context while BOW only consider the word occurrences.
2) We've compared results using different word embedding layers (trainable vs pre-trained word embedding).
3) We've compared models using different combinations of layers: single LSTM, (LSTM, Dropout), (CNN, Maxpooling, LSTM, Dropout), and multiple LSTMs.
4) We ensure that the parameters used in each layer are appropriate: LSTM with 100 hidden units/128 hidden units are common choices in many articles, dropout rate 0.2 ~ 0.5 are common in many articles.
5) We've applied different training set (input) and different testing set to our final model: the results are very good. That proves our model is very robust.

Based on the above observation and summary, our final solution is significant enough to have solve the problem.

V. Conclusion

Free-Form Visualization

One important number we need to determine for either our benchmark model or our Deep Neural Network models is the number of words we want to use – the models will only take top number_of_words for training and predicting.

The steps we take to determine this number is:

1. Estimate the total occurrences of all the words in the whole training set. The equation is: (number of reviews) * (mean of the review length).
   The supporting output are:
   ```
   Mean of the review length in training set is: 123.28
   …
   Mean of the review length in testing set is: 120.44
   ```

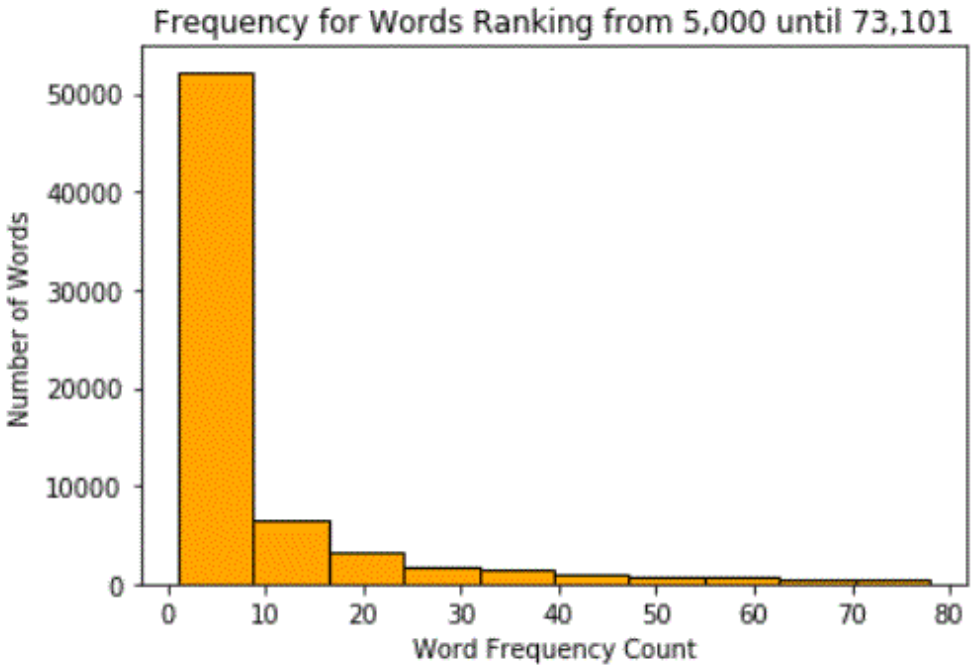
   We got the rough number for total occurrences is: 3,000,000 (25,000 * 120) or 1,500,000 (12,500 * 120) word occurrences for positive or negative reviews.

2. Determine the low watermark of word frequency and find out how many words have low frequency. A word with frequency count less than 150 (0.01 % of 1,500,000) might be a low frequency word. A word with frequency count less than 12 (~0.1% of 12,500)

shows up only once in every 1000 reviews. That should be a low frequency word too. Print out some numbers to find how many words have low frequency.
The supporting output are:

```
There are 1612 words with frequency count >= 300.
There are 2943 words with frequency count >= 150.
There are 4936 words with frequency count >= 80.
There are 9998 words with frequency count >= 30.
There are 19706 words with frequency count >= 10.
There are 28600 words with frequency count >= 5.
There are 32124 words with frequency count >= 4.
There are 37233 words with frequency count >= 3.
```

Note: we have 73,101 words in our vocabulary.



Observations: there are over 50,000 words with occurrence less than 10. The highest occurrence here is less than 80. If the word with that occurrence reflect a positive (or negative) sentiment, it only shows up 6 times in every 1,000 reviews.
(80/12500 = 0.0064) - is that frequency high enough for model to learn? We will keep this information in mind.

3. Get test accuracy from benchmark model by pass different numbers for max_features parameter in CountVectorizer API – this API will build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.
   The supporting output are:

Max_features: 5,000 Test accuracy: 83.96
Max_features: 10,000 Test accuracy: 83.62
Max_features: 20,000 Test accuracy: 83.08
Max_features: 30,000 Test accuracy: 82.65
Max_features: 40,000 Test accuracy: 82.56

4. Pick the max_features from the model which has the highest test accuracy. Hence 5,000 is the vocabulary size (top word to keep) we use in this project.

Reflection

Summary for the entire end-to-end solution for this problem (sentiment classification for large movie review):

1. Do a good job in cleaning the training/testing data.
2. Create a simple but work benchmark model: like Naive Bayes with Bag or Words.
3. Create model(s) with trainable word embedding generated by Keras based on vocabulary/tokens created from the cleaned training data prepared by us. Use RNN /LSTM with proper dropout. May add optional CNN plus Maxpooling layer to comp are the results.
4. Create a reference model which uses pre-trained word embedding (like Glove). Compare the results from step 3). Note: model with Glove usually takes some time to train to get good result.
5. Compare step 3) and step 4) and pick a best model. Glove is a good choice when we have a smaller training set. However, if we have a large dataset, it will take us some time to train the model. But it's good to use Glove on the same architecture we created (which uses trainable word embedding prepared by us) to verify if all the parameters are appropriate.

The difficulties I found in the project:

1. Research/get enough information regarding how to design the layers in the RNN.
2. I was not familiar with how to a) create a trainable word embedding use vocabulary built by us b) how to use a pre-trained word embedding layer (like Glove) in Keras. There are too many examples which just use imdb.load_data to create vocabulary plus load training/testing data which are already coded with tokens that are ready f or the Deep Neural networks. But there are not many good examples about creatin g trainable word embedding or use pre-trained word embedding. It took me a lot of time to find such examples and try myself.

In general, the final model and solution fit my expectation for the problem, and it can be used in a general setting to solve these types of problems.

Improvement

Anything we can do to improve our model? I can think of a few:

1) Text cleaning: replacing sentences like “I don’t like” to “I do not like”, “I’ll” to “I will” etc. These might have some impact on sentiment classification.
2) Text cleaning: applying Porter Stemming Algorithm – The Porter stemming algorithm (or ‘Porter stemmer’) is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval system.
3) Using word2vec for word embedding layer.