Machine Learning Engineer Nanodegree

Capstone Project

Siddheshwar Kumar
April 28th, 2018

I. Definition

Project Overview

Sentiment analysis is one of the most interesting applications of Natual Language Processing. It's used to figure out if a text, sentence or paragraph expresses negative, positive or neutral emotional feeling.

Social platforms like Twitter, Facebook etc have opinions expressed about different products and services. These opinions are a trove of information for companies which want to improve their product, services, and marketing strategies. This is where sentiment analysis can help. It can tell in general what people think about your product i.e. they like it or dislike it.

In the past, I have explored this topic a bit but could not make much progress. I plan to use this opportunity to understand this interesting problem and apply deep learning to solve sentiment analysis on IMDB movie reviews. Other related datasets for performing sentiment analysis are Tweeter Feeds (https://www.kaggle.com/c/twitter-sentiment-analysis2), and rotten tomatoes(https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews) datasets.

Problem Statement

The goal is to perform sentiment analysis on IMDB reviews. This dataset is from an old Kaggle competition, **Bag of Words Meets Bags of Popcorn (https://www.kaggle.com/c/word2vec-nlp-tutorial)**.

The trained model should be able to predict whether a movie review is negative or positive given only the text.

For this problem, I will apply Naive Bayes first to create a benchmark. Naive Bayes works quite well in text classification problems. And, then plan to use the accuracy obtained in the Naive Bayes as a baseline and improve it further using Deep Learning Technique. Recurrent Neural Networks are often used to deal with texts; so I will be using RNN as the final approach.

Metrics

For Naive Bayes approach, I have used Compute Receiver operating characteristic (ROC) method. This method is suitable for the binary classification problem which is the case here. The ROC curve is a graphical plot that illustrates the performance of any binary classifier system as its discrimination threshold is varied. To understand ROC curve, please refer this(https://www.quora.com/Whats-ROC-curve).

And for RNN technique, I have used *Sigmoid* activation function. RNN/LSTM networks are bit different than the normal feedforward network. Only the last output gets considered, rest all get discarded. Cost is calculated using Mean Squared Error and uses AdamOptimizer.

II. Analysis

Data Exploration

IMDB dataset contains 25,000 labeled training reviews, 50,000 unlabeled training reviews, and 25,000 testing reviews. Unlabeled training data and testing data doesn't have a sentiment or output field; only labeled training data has sentiment field. So, to solve the problem, I have used only labeled training data. This way the implementation can get trained as well as tested.

The file *(labeledTrainData.tsv)* is tab-delimited and has a header row followed by 25,000 rows and contains three columns/fields:

```
['id', 'sentiment', 'review']
```

Fields in the file are:

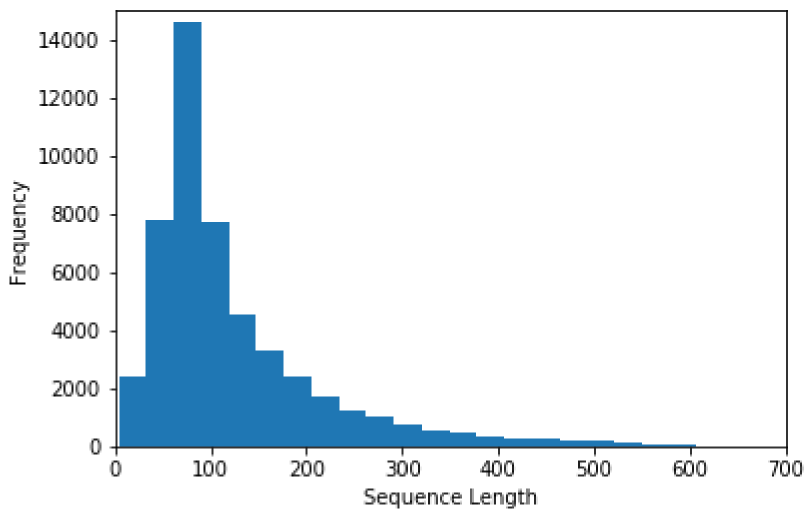
- *id*: Unique identifier for each entry in the dataset; we don't need this field for modeling.
- *sentiment*: Contains binary values (1 and 0). 1 for positive and 0 for negative. This is the label of the model.
- *review*: a Detailed review of movies. This is the text or feature on which machine learning models will get trained.

I have used 20% of data for testing and will report the accuracy of the model on this randomly selected dataset. The remaining 80% will be used for training the model. Distribution is Dataset is balanced (i.e. both positive and negative are 50% each). 12,500 are positive reviews and an equal number of negative reviews as well.

Exploratory Visualization

One of the major aspects of neural networks is that they take inputs of fixed length. Reviews are not of fixed length, this means we can't directly feed them to the network.

Let's study the distribution of word count across the reviews and then decided upper limit on the number of words to be considered for each review.



On the X-axis, it defines the distribution of the number of words. And on Y-axis it captures how many reviews fall into that category.

From the chart, it's evident that most of the reviews fall under 250 words. **Also, please note that this chart was prepared after cleaning the reviews of punctuations, special chars, and stop words.**

Algorithms and Techniques

For this text classification problem, I have used two approaches:

Naive Bayes Classifier:

One particular feature of Naive Bayes is that it's a good algorithm for working with text classification. The relative simplicity of the algorithm and the independent features assumption of Naive Bayes make it a strong performer for classifying texts. The Naive Bayes classifier uses the **Bayes Theorem** to select the outcome with the highest probability. This classifier assumes the features(in this case- words) are independent and hence the word naive.

The Naive Bayes classifier for this problem says that the probability of the label (positive or negative) for the given review text is equal to the probability of the text given the label, times the probability a label occurs, everything divided by the probability that this text is found.

$$P(label|text) = \frac{P(text|label) * P(label)}{P(text)}$$

Text in our case is a collection of words. So above equation can be expressed as:

$$P(label|word1, word2, word3...) = \frac{P(word1, word2, word3...|label) * P(label)}{P(word1, word2, word3...)}$$

We want to compare the probabilities of the labels and choose the one with higher probability. The denominator, i.e. the term P(word1, word2, word3…) is equal for everything, so we can ignore it. Also, as discussed above there is no dependence between words in the text (not possible always as few words mostly appear together but we can ignore such aberrations); so equation can be re-written as:

$$P'(label|word1, word2...) = P(label) * P(word1|label) * P(word2|label)\ldots$$

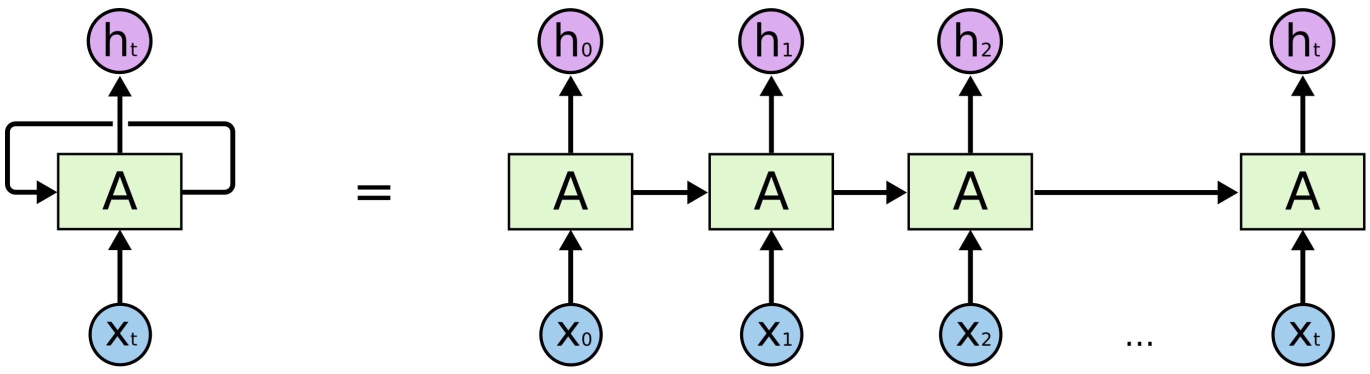
P(label=positive) is the fraction of the training set that is a positive text; P(word1|label=negative) is the number of times the word1 appears in a negative text divided by the number of times the word1 appears in every text. I have used sklearn library for implementing Naive-Bayes.

Recurrent Neural Network/Long Short-Term Memory (RNN/LSTM):

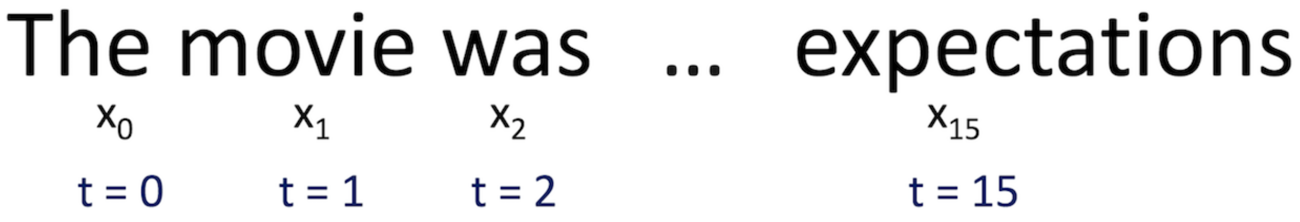
Recurrent Neural Network, is basically a neural network that can be used when data is treated as a sequence, where the particular order of the data-points matter. More importantly, this sequence can be of arbitrary *length.*

They are networks with loops in them, allowing information to persist. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop.

Below is an image from famous colah blog (http://colah.github.io/posts/2015-08-Understanding-LSTMs/).



Recursive neural network proved to be efficient in constructing sentence representations. The model has tree structure, which is able to capture semantic of the sentence.



RNN suffer from vanishing gradients problem and makes it difficult to learn long-distance correlation in sequence. LSTM is a type of RNN and now mostly the de-facto implementation of RNN.

Benchmark

Naive-Bayes approach discussed above acts as a benchmark. The final approach should be provided accuracy more than the benchmark value.

naive-bayes.ipynb gives the benchmark values as 85%. So the RNN/LSTM network should achieve more than 85% accuracy on test data.

III. Methodology

Data Preprocessing

Text contains a lot of noise or un-important content. In current problem as well, not all features will be equally important. I have used techniques to preprocess the dataset.

Remove special chars

Punctuation, numbers and special characters are not going to add any value in sentiment analysis, so will remove all such characters. Also, the data is generated from an online platform so it might have some HTML tags as well;

will get rid of them as well.

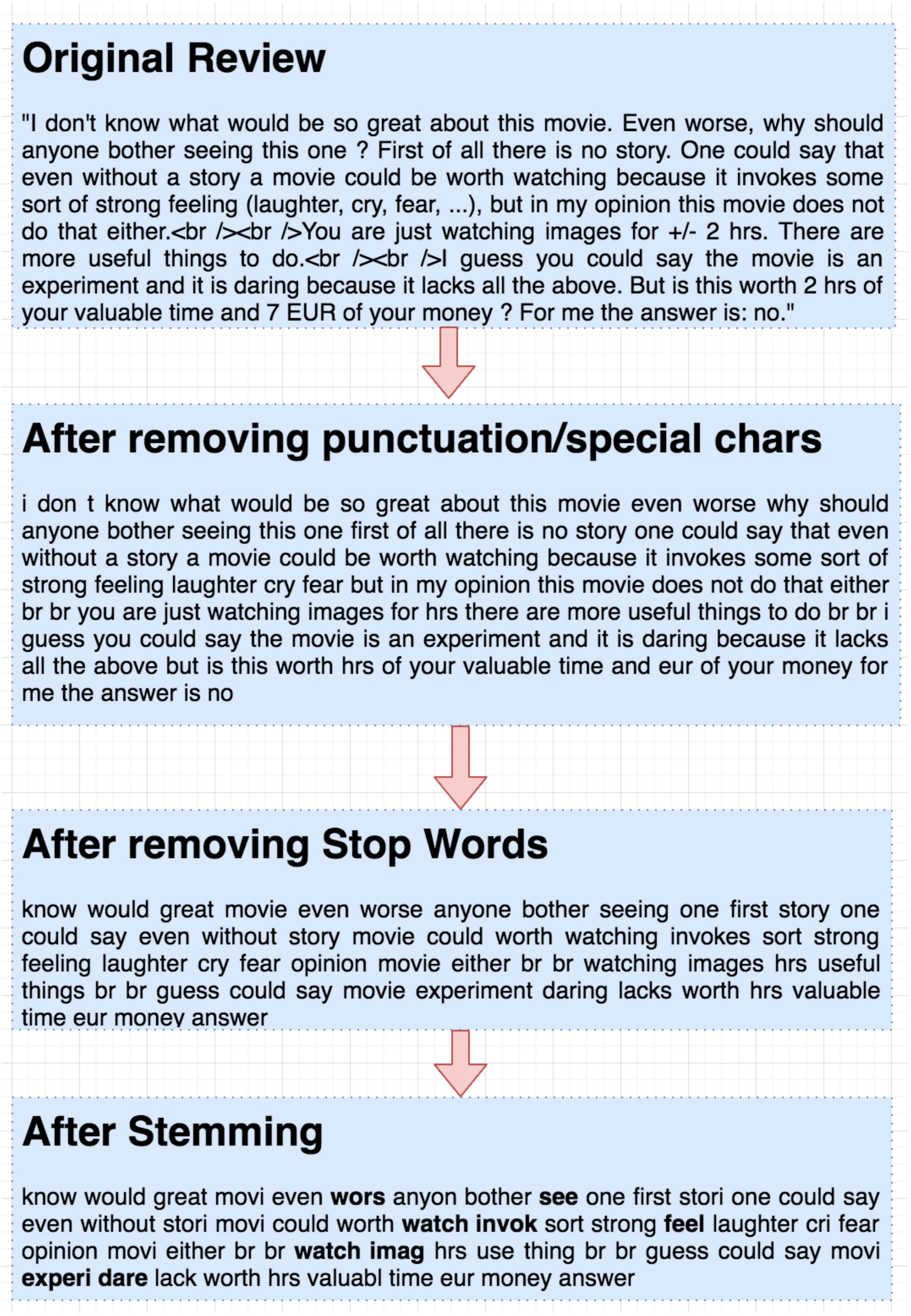
Remove stop words

Words like **and, the, it** etc don't carry any meaningful information for sentiment classification. These words are known as Stop Words. I will remove all stop words from the reviews. Plan to use Python's **NLTK** library to get list of English Stop words.

Stemmerize

Not all unique words are different. Take the example of *love* and *loves*; both are same but if treated differently it will unnecessarily increase the vocab (size).

Below diagram shows how above techniques change review:



Original Review

"I don't know what would be so great about this movie. Even worse, why should anyone bother seeing this one ? First of all there is no story. One could say that even without a story a movie could be worth watching because it invokes some sort of strong feeling (laughter, cry, fear, ...), but in my opinion this movie does not do that either.<br /><br />You are just watching images for +/- 2 hrs. There are more useful things to do.<br /><br />I guess you could say the movie is an experiment and it is daring because it lacks all the above. But is this worth 2 hrs of your valuable time and 7 EUR of your money ? For me the answer is: no."

After removing punctuation/special chars

i don t know what would be so great about this movie even worse why should anyone bother seeing this one first of all there is no story one could say that even without a story a movie could be worth watching because it invokes some sort of strong feeling laughter cry fear but in my opinion this movie does not do that either br br you are just watching images for hrs there are more useful things to do br br i guess you could say the movie is an experiment and it is daring because it lacks all the above but is this worth hrs of your valuable time and eur of your money for me the answer is no

After removing Stop Words

know would great movie even worse anyone bother seeing one first story one could say even without story movie could worth watching invokes sort strong feeling laughter cry fear opinion movie either br br watching images hrs useful things br br guess could say movie experiment daring lacks worth hrs valuable time eur money answer

After Stemming

know would great movi even **wors** anyon bother **see** one first stori one could say even without stori movi could worth **watch invok** sort strong **feel** laughter cri fear opinion movi either br br **watch imag** hrs use thing br br guess could say movi **experi dare** lack worth hrs valuabl time eur money answer

Implementation

The implementation can be divided into two major stages:

1. Pre-processing component which deals with loading dataset in the memory and performing different operations
2. Model Implementation (Naive Bayes and RNN/LSTM)

Pre-Processing Layer

There are two implementations; so I have abstracted common operations in a separate python file, Imdb.py. Some of the operations are:

- clean_text(review): Takes a particular review and removes punctuation, numbers and special characters. Also, convert all chars to lower case.
- remove_stop_words(review): Takes a particular review and removes all stop words by using Python's NLTK library. It removes only English stop words as we are dealing with English text.
- stemmerize(review): STEMMER helps to reduce the vocabulary size drastically by treating similar words like love, lovely, loves as same.

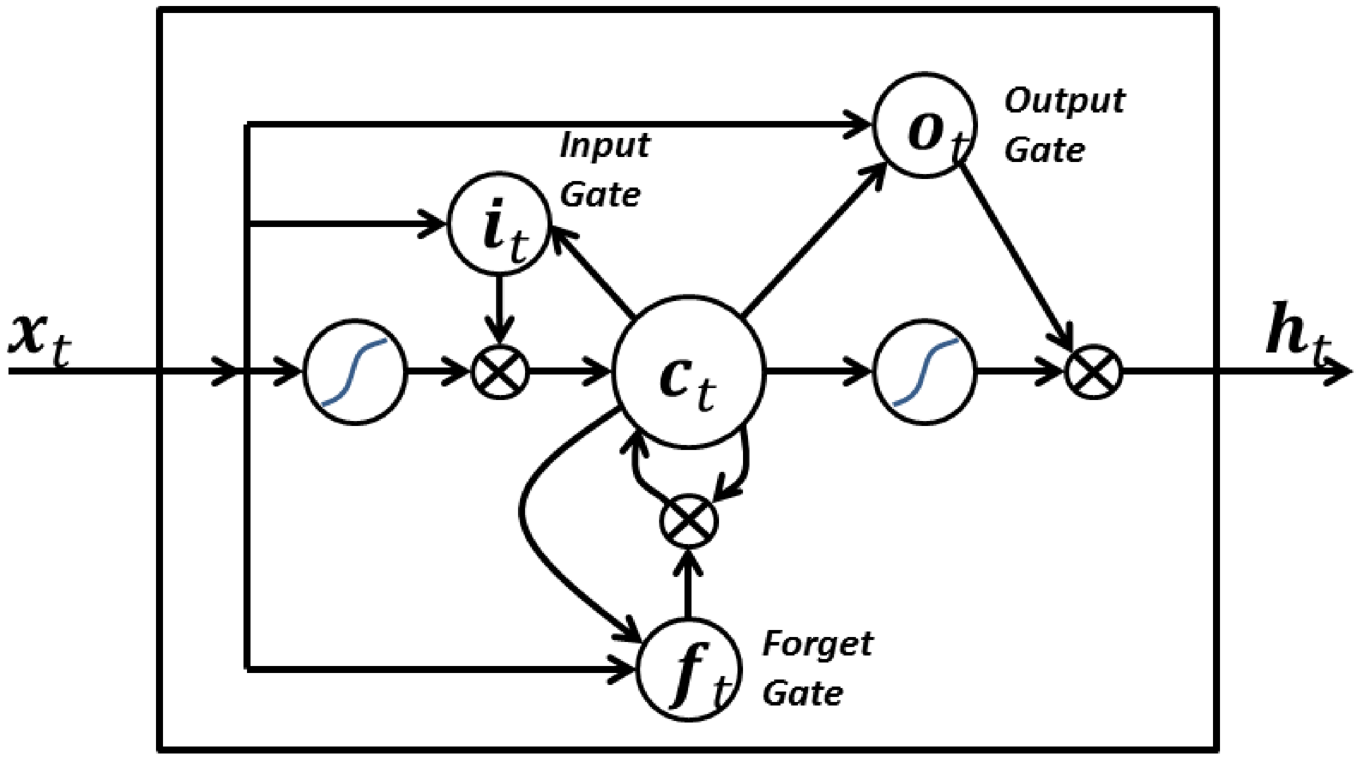
Apart from above it also provides some reusable methods. Imdb.py is documented properly.

Model Implementation

The project contains two notebooks files one for each approach. For the first benchmarking approach, I have used Naive Bayes and it's implemented using SKLearn library (check naive-bayes.ipynb). For the final approach, I have used LSTM and have used Tensorflow for this. The implementation can be found in rnn-lstm.ipynb. It's a quite simple implementation with one Input layer, one LSTM layer and the final output layer.

In RNN/LSTM model, words are passed to an embedding layer. Embedding layer is required because we have tens of thousands of words, and we need a more efficient representation for our input data than one-hot encoded vectors. Word2Vec or similar implementation can also be used for embedding. But it's good enough to just have an embedding layer and let the network learn the embedding table on it's own.

From the embedding layer, the new representations will be passed to LSTM cells.



These will add recurrent connections to the network so we can include information about the sequence of words in the data. Finally, the LSTM cells will go to a sigmoid output layer here. The output layer will just be a single unit then, with a sigmoid activation function. We don't care about the sigmoid outputs except for the very last one, we can ignore the rest. We'll calculate the cost from the output of the last step and the training label.

Refinement

Below are parameters which I tweaked to improve the performance:

- dropout_rate: started with .8 and then increased it to .9.
- num_units: num_units, the number of units in the cell, called lstm_size in this code. Usually, larger is better performance wise. Started with the value of 128 and then finally trained model with the value of 512.
- num_hidden_layer: I tested model on my local laptop so kept it at 1. But it can be increased further in a more powerful CPU/GPU machine.
- dropout_rate: used 0.9
- learning_rate: started with 0.001 and then finally used 0.01

IV. Results

Model Evaluation and Validation

During model implementation, tested the performance by varying different hyperparameters:

- Removed all characters other than alphabets i.e. only [a-z]
- Removed all stop words and stemmerized to keep the vocab limited
- For word embedding; used embedding layer and let the network learn the appropriate representation of encoding of the words on its own in the course of training.
- Reviews length are variable. Fixed the sequence length to 250 words. Reviews which are more than 250 words will have only last 250 words. And reviews which are less than 250 words will get padded with 0 in

the beginning.
- From the embedding layer, the new representations will be passed to LSTM cells. These will add recurrent connections to the network so we can include information about the sequence of words in the data.
- Wrap that LSTM cell in a dropout layer to help prevent the network from overfitting.
- LSTM cells go to sigmoid output.
- We don't care about the sigmoid outputs except for the very last one, we can ignore the rest.

The model is generic for sentiment analysis problems. I have tweaked many parameters and saw that results were almost consistent. Minor changes in parameters didn't result in major deviation in results. So, can say that the results can be trusted.

Tested this model on another Kaggle problem(https://www.kaggle.com/c/si650winter11), Michigan University Sentiment Classification Contest. The data set has sentences extracted from social media and then classified into positive or negative. Model without much changes gave good results.

Justification

RNN/LSTM models have proved efficient in natural language problems. LSTM cells help the network remember the inputs which are important. So certain words which can play a major role in deciding if the sentiment is positive or negative will get captured.

I have implemented a quite basic LSTM network but, still, it outperformed the benchmark model (through Naive-Bayes) accuracy. The fundamental problem with Naive Bayes approach is that it doesn't take into order the the relative positive of words in the review or sentences. It barely goes by the presence, absence of a word and how many times it appeared. But, LSTM model captures the order part as well. It can also capture the influence of a word or words which appeared to say 100 words before.

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

Reflection

I thoroughly enjoyed exploring sentiment analysis. I was aware of the subject earlier but had never explored it so deeper. Here are major steps:

- *There are quite a few datasets available publicly for sentiment analysis. I wanted to use a dataset which is not very trivial (like just a few lines of tweets), and that's where I feel IMDB dataset is perfect. Reviews size go all the way up to 1400 words.*
- *The dataset is derived from an online platform, this means that there going to be enough noise which needs to be effectively filtered out*
- *One of the interesting part of the Capstone project is to decide your own success/evaluation criteria. That made me to think through the problem and recommend using Naive Bayes to benchmark the*

accuracy.

- *Naive Bayes fits quite well in this problem, it's simple to implement and found that it gave surprisingly good accuracy in just a few minutes. In fact, I was quite surprised to see the accuracy of 85% for Naive Bayes algorithm. It shows how powerful it is. And, unlike Neural Networks which take hours to get decent accuracy, Naive Bayes produced this accuracy in just matter of 5-10 mins.*
- *Once I decided to use one model for benchmarking and then again a different model as final solution. I thought, to abstract some repetitive steps in a separate python file. This is where, I ended up writing Imdb.py. This python 3 class file has methods to load dataset, clean dataset and perform some other reusable methods on dataset.*
- *My next dilemma was to whether let network learn the word embedding or use some existing word embedding like Word2Vec. Implementation would vary depending on which path is chosen. I decided to go ahead with network learning word representation on its own, keeping in mind that in this case performance might not be as good as using a pre-trained model.*
- *Deciding on sequence length was interesting. Drew a chart on distribution of sequence length and frequency gave idea that 250 looks a good choice _ RNN/LSTM networks are bit different that normal feed forward networks so it took a while to appreciate LSTM networks and why they are apt fit for the current problem. I did few POCs around LSTM to understand them. _ Once the network was implemented, I explored on fine tuning it. One surprising aspect was, network performance on training and test data was not consistent. On training dataset it gave good performance and reached as high as 99% but on test dataset it hovered around 85%. I tweaked parameters to improve the performance.*
- *I think there is still scope to improve the performance. The network can still be made deeper and can be trained on GPU. Currently, it takes 4-5 hours on my laptop to complete one cycle of modelling.*

Improvement

In the model, I let the network learn the word encoding. I think, using pre-trained models like Word2Vec can help improve the performance of the model. The network can be made deeper to improve performance.

As discussed above, there are possibilities to improve the performance further.