Project 2 proposal: Sentiment Analysis on Movie Reviews

By Noel Mathew

What is the problem I want to solve?

Every time a movie comes out, there is always somebody reviewing it. Sometimes the movie could get great reviews. In other cases, the movie could be a flop and get bad reviews. This project goal is to evaluate what type of model can predict a good review or a bad review the best. In addition, this project will check if any of these words can show if a review is good or bad. This means does certain words show that the movie is a classic or a flop.

Who is my client and why do they care about this problem?

One of the clients in this case will be the movie critics. The movie critics can use this project to learn which phrase of words strongly highlight if a movie is bad or good. Furthermore, they can use the project to figure out what phrase to use to get to the point of their review. Another client could be the general audience who can use the solution of the problem to write user reviews on sites like Rotten Tomatoes. Another client could be people who work in the movie industry. For example, a movie producer or executive like Disney and can run the model on multiple reviews online to see which movie was good or bad and decide which are good movies to make sequels from. This will help companies save time and resources in manually looking at each review and determining that the movie is good or bad.

What data am I using? How will I acquire the data?

The data I will be using is a movie phrase train set which has phases from reviews and their sentiment analysis score. I will use this training set to help fit a model to the test set. I will use the test-set to assign a sentiment label score to the phrase. All the data will come from https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data and https://www.kaggle.com/c/word2vec-nlp-tutorial/data.

Briefly outline how I'll solve this problem

I believe that various machine learning methods will be used to see relationships between phrases and words and their sentiment towards a movie. I will run many different performance scores to see which model can show the relationship between words and sentiment the best.

Here's my outline as I envision the project right now:

1. Gather, wrangle, and tidy up data for movies reviews I can get data for
2. Train various natural language processing/machine models I can on the data
3. Use cross-validation techniques to tune hyper-parameters for each of the model
4. Use the results of the model to see what model was best in predicting if the movie was good or bad.

What are my deliverables?

At the end of the project, I will have a Jupyter Notebook with both the project code and my report comments. I will also have a slide deck for presenting the project. I will have a list of models and their performance scores. In addition, the slides will show how certain words lead to positive or negative words.

Data Cleaning

I got the data to do the sentintement analysis for movie reviews from https://www.kaggle.com/c/word2vec-nlp-tutorial/overview/part-2-word-vectors. The data set was used part of a Kaggle Word2Vec and natural language processing tutorial and competition. The file I wanted to use to do the modeling was called labeledTrainData.tsv. The file contains 25000 reviews from IMDB. It contained three fields, the id for the movie, the review of the movie, and the sentiment score. The sentiment score is 0 for a negative review and 1 for a positive review. Therefore, the problem I want to solve is a binary classification one. The file was converted to a dataframe using pandas package and its DataFrame method. The reviews did not only have words in them. It had html tags like <br> and "/". They also had punctuation which needed to be removed as it will mess up building the vectors and thus mess up the machine learning. There were numbers in reviews which could have the same effect as punctuation. The non-words had to be removed from the review and the next paragraphs will explain that process.

In order to get rid of the html tags, the Beautiful Soup library was used. The Beautiful Soup library is a library that is used for pulling data out of HTML and XML files. This library is needed as the reviews are html encoded or have html tags. So I got each raw review from the dataframe and got rid of the HTML using BeautifulSoup().get_text() method. Then I got rid of the numbers and punctuation by using regex expression to replace nonwords with spaces. Then the words in the review was converted to lowercase because uppercase and lowercase of the same word has different values. So if one word has both an uppercase and lowercase form then those would different values even though they are the same word. Therefore, by converting the words to lowercase it keeps consistent values among the words. The review was split into a list with each item being a words, so the review can be iterated through. Then set of stopwords was brought in. A stopword is a word that has no or neutral meaning. For example, the word “the” is a stopword because it provides no meaning on context to the content. This in turn can make a review more neutral than it is. So, a set was made with those stopwords like ‘the’. Then the stopwords was removed in the list of words in the review by making a list comprehension that excluded the stopwords. Then new list of words was join back to one string. This was done in one function called review_to_words.

The review_to_words function was run on each review on the dataset. Then each review got appended to a list called clean_train_reviews. Then a new column in dataframe called cleanReview was made and the values were the items in clean_train_reviews. Then dataframe was converted to a csv file using pandas; therefore the csv file had the clean reviews in it.

Exploratory Analysis

The new csv file with the clean reviews was brought over using the dataframe feature from pandas. The dataframe had 25000 entries and 4 columns. The 4 columns are ‘id', 'sentiment', 'review', 'cleanReview'. The first three columns were like before and ‘cleanReview’ column is the new columns with reviews with just the words. There are no null values in the datalist, so did not have to worry about imputing null values. The clean reviews are all strings, so data cleaning process was done correctly. Then prebuild word2vec model gensim was brought in to find words with similar meanings in reviews and see if those reviews containing those words were positive to negative. A word2vec model translates words into vectors that are plotted in a coordinate space. Therefore, one could find similar words by distance of the vectors between each word. So, to see

how gensim finds similar words, it was tested on some words. For example, it was tested on the words sad. By using most_similar positive method of the gensim model it was able to find similar words to sad which were ('saddening', 0.7273085713386536), ('Sad', 0.6610826253890991), ('saddened', 0.6604382991790771), ('heartbreaking', 0.6573507785797119), ('disheartening', 0.6507317423820496), ('saddens_me', 0.6407118439674377), ('distressing', 0.6399092674255371), ('saddest', 0.6345508694648743), ('unfortunate', 0.6272086501121521), ('sorry', 0.6194046139717102). This shows the words that are closest to the word sad and their vector scores. Likewise this test was run on other words like good, bad, horrible, and fantastic. In order to see the distribution of the review scores, a histogram was used on sentiment score to see the frequency of positive and negative reviews. The distribution was even between positive and negative reviews which 12500 each. This could have been done on purpose as this data set came from a tutorial. They probably made the data set even to run accuracy score to check the performance of the tutorial models as it quick and easy performance test. The next analysis that I wanted to do was to see how a word in a review affects if a review as positive or negative. Therefore, I used the most similar method from the gensim model again. So, I started with the word fantastic because the word is very positive in nature. So, I got the similar words to fantastic which were 'terrific', 'wonderful', 'great', 'amazing', 'marvelous', 'fabulous', 'awesome', 'phenomenal', 'incredible', 'unbelievable'. Then a loop was ran and the reviews which contained those words were appended to the list. The list had a length of 9968 scores and those scores were converted to a dataframe. Then the scores were graphed using a histogram plot. The distribution was the reviews were mostly positive with some negatives. The negatives could come from the word like 'unbelievable' For example a review could have been written as 'the movie was 'unbelievable' in its premise'. Therefore, that is a negative review. This was done on the word horrible as well because horrible is a negative word.
The most similar words to horrible were 'terrible', 'horrendous', 'dreadful', 'horrid', 'awful', 'atrocious', 'horrific', 'horrible_horrible', 'hideous', 'appalling'. The numbers of the reviews that contained the list of words above were 4618. The distribution of the score was mostly negative. The positive scores could come from reviews like "The dreadful nature of the villain was greatly fleshed out". The next word that was tried was bad because it should be a common word used in negative reviews. The words similar to bad were 'good', 'terrible', 'horrible', 'Bad', 'lousy', 'crummy', 'horrid', 'awful', 'dreadful', 'horrendous'. It is weird to see good there, but this might mean that good is words that does not have a lot of positivity. The number of reviews that the words above were 14333 reviews and the distribution of the sentiment was mostly negative, but there is a good number of positive reviews. This could be that the word good appears in more reviews compared to the word bad as good is the only positive word in the list. The next word that was tried

was good because it should be a common word used in positive reviews. The words similar to good were 'great', 'bad', 'terrific', 'decent', 'nice', 'excellent', 'fantastic', 'better', 'solid', 'lousy''. It is weird to see bad and lousy there, but this might mean that good is words that does not have a lot of positivity or bad does not have a lot of negativity. Furthermore, good could be used negative manner like the phrase it’s just good.   The number of reviews that the words above were 18757 reviews and the distribution of the sentiment was even. This could be that the word good has a double meaning and the word bad appears more often than the other words in the list. I did the same for best and worst and found their distributions to be even respectively. This could be due to the nature of the review. This could be done with more words, phrases, and combinations. but it will take a lot of time doing this.

Since these are unique reviews with words or text, it is not possible to do statistical analysis on the features which are the clean reviews. The labels have a mean of 0.5 as the scores are evenly disburtubed between negative (0) and positive (1).

Machine Learning

Feature Selection

The problem with modeling text is that is messy and machine learning algorithms prefer well defined fixed lengths inputs and outputs. In addition, machine learning can not work raw text directly. The text has to has to be converted to a vector of numbers. This method is called feature encoding or feature extraction. In this project, two models for feature extraction was used. One model was a bag of words model and the other model was a Word2Vec model. Those models are going to be explained in the next paragraphs.

Bag of Words

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling. It is a representation of text that counts the occurrence of words in text body which are movie reviews for the project. A BoW model has two features, a vocabulary of the known words and the count of the words. However, how the words are ordered are completely disregarded. In this case the vocab comes from the words in the clean reviews and the model counts how many times a word appears in the reviews.

In order to convert a review into a bag of words model, scikit-learn has a CountVectorizer Object which does that. By using its fit transform function, it does the following two steps. First, it fits the model and learns the vocabulary; second, it transforms our training data into feature vectors which counts the words in the review. There is no need to worry about adding a penalty words that have no meaning because those words were removed when the data cleaning happened. Some problems with the BoW model are the sparse vectors where a review could translate to a vector with a bunch of zeros which could be the case as no reviews would contain every word in the vocab and the order of the words are gone. Therefore, the reviews could lose their meaning. The machine learning algorithms that used the Bag of Words for this project was Random Forest, LinearSVC, and Naive Bayes.

Word2Vec

The next method that was used in this project to transform text to vectors was Word2Vec. Word2vec is a two-layer neural net made by Google that processes text by “vectorizing” words. Its input is a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus. The purpose and usefulness of Word2vec is to group the vectors of similar words together in a vectorspace. Word2vec can make highly accurate guesses about a word’s meaning based on past appearances. Those guesses can be used to establish a word’s association with other words or cluster documents and classify them by topic which is useful for sentiment analysis. The vectors that are used to represent words are called neural word embeddings. Since making and running a Word2Vec model takes a long time, in this project a pre made Word2Vec model, gensim, was used with a limit of 500000 words. The model as get_vector function to get the vector for each word in the review and that vector was appended to a list. Since reviews come in different lengths which might adversely affect the number of features the average word vector was taken for each review. Therefore, it cut the number of features, and it could possibly lead to reviews being more neutral/negative/positive then it should be. The machine learning algorithms that used the Bag of Words for this project was Random Forest, LinearSVC, and K-Nearest-Neighbors.

Machine Learning Algorithms

Bag of Words Algorithms

Random Forest

A Random Forest Algorithm is used because it is a good way to split and classify inputs into certain groups. In addition, it is an ensemble method it run many decision trees, so it has the benefit of running many tests in parallel. Since the problem this project is solving is binary classification problem where a movie is positive or negative, based on the research a random forest algorithm should be good performing model with the bag of word features. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to vectors the random forest can train and predict on them. After training and predicting the model the accuracy score came out as 1.0 on the training set and 0.844 on the test set. This clearly shows the model is overfitting. Accuracy score can be used for the models here as the labels are evenly distributed. The F1 score was 0.85 on the negative score and 0.84 on the positive score and AUC of ROC was 0.92. Hyperparameter tuning was attempted to be run on this model. However, due to the fact that the data was sparse and there was too many features the machine froze.

Linear SVC

For this project, a linear SVC model can be a good model since it is a binary classification problem. Furthermore, it is a fast model to run. Based on research, Linear SVC models do well in text classification and can handle sparse vectors and increased features better than a random forest model. Since it is a binary classification problem, Linear SVC model should have no problem making hyperplane dividing positive and negative reviews. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to vectors the SVC can train and predict on them. After training and predicting the model the accuracy score came out as 0.998 on the training set and 0.833 on the test set. This clearly shows the model is overfitting. The F1 score was 0.83 on the negative score and 0.84 on the positive score. Hyperparameter tuning was done on this model by running the model using different penalty (C’s) values (0.1, 1, 10, 100, 1000) and the loss ('squared_hinge', 'hinge') on a GridSearchCV. This improved the model performance and reduce the overfitting issue. After training and predicting the model with the best parameters (C = 0.1, loss = ‘squared_hinge’) the accuracy score came out as 0.96 on the training set and 0.86 on the test set. This clearly shows the model is still overfitting, but less than before. The F1 score was 0.86 on the negative score and 0.86 on the positive score.

Naive Bayes:

For this project, a Naive Bayes model can be a good model since it does well in text classification according to research and other experience. Since the bag of words model in this project has no order the words are independent from each other which is needed to run a Naive Bayes model. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to vectors the Naive Bayes can train and predict on them. After training and predicting the model the accuracy score came out as 0.992 on the training set and 0.864 on the test set. This clearly shows the model is overfitting. The F1 score was 0.83 on the negative score and 0.84 on the positive score. Hyperparameter tuning was done on this model by running the model using different alpha values (.1, 1, 5, 10, 50). This improved the model performance and solved overfitting issue. After training and predicting the model with the best alpha (50) the accuracy score came out as 0.876 on the training set and 0.853 on the test set. The F1 score was 0.86 on the negative score and 0.85 on the positive score, and AUC on the ROC was 0.92

Overall, the bag of words had an overfitting issue. That could be due to the sparseness of the data or vocab that existed in the training set and that is not there in test set. The best performing model with Bag of Words was Naive Bayes because it got rid of its overfitting issue after doing hyperparameter tuning.

Word2Vec Algorithms

Random Forest

A Random Forest Algorithm is a good model to use for Word2Vec because the reasons listed above. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to the average word vector using gensim so the random forest can train and predict on them. After training and predicting the model the accuracy score came out as 0.992 on the training set and 0.756 on the test set. This clearly shows the model is overfitting. The F1 score was 0.77 on the negative score and 0.74 on the positive score and AUC of ROC was 0.83. Hyperparameter tuning was done on this model using GridsearchCV with different estimators, depths, and samples_split. After fitting three fold the best features were (max_depth=60, min_samples_split=5, n_estimators=300) After training and predicting the model with the best parameters the accuracy score came out as 1.000 on the training set and 0.824 on the test set. The F1 score was 0.82 on the negative score and 0.83 on the positive score. Both random forest models had an overfitting issues. This shows that on this dataset random forest is not a good model to use due to overfitting especially compared to Naive Bayes model which does not have the issue, but has a similar performance score.

Linear SVC

A Linear SVC is a good model to use for Word2Vec because the reasons listed above. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to the average word vector using gensim so the SVC can train and predict on them. After training and predicting the model the accuracy score came out as 0.865 on the training set and 0.857 on the test set.. The F1 score was 0.86 on the negative score and 0.86 on the Hyperparameter tuning was done on this model using GridsearchCV with different C values and loss. After fitting three fold the best features were (C=1, loss=squared_hinge) which were the same as the default model. This shows Linear SVC with the Word2Vec is best performing model so far as it has the best accuracy score and has no overfitting/underfitting problems.

K-Nearest-Neighbors.

A KNN model was used to see it can group positive and negative reviews better than a Linear SVC, due to the fact an observation gets labeled by its closest points. However that is not to be the case. Even with hyperparameter tuning the Linear SVC model still outperform the KNN model.

Auto ML

Tpot, a Auto ML model, was used to find the best model to use with Word2Vec. A train test split was used with the clean reviews as the features and sentiment as the analysis with a test size of 0.33. The features were converted to the average word vector using gensim so the tpot can train and predict on them. After running tpot with 5 fold validation with a population size 10 and 5 generations, the best model it came up with during that time period was ExtraTreesClassifier on top of an LinearSVC   After training and predicting the model the accuracy score came out as 0.878 on the training set and 0.856 on the test set.. The F1 score was 0.86 on the negative score and 0.86 and the ROC curve was 0.92.

Overall, the best model to use is the Linear SVC model with Word2Vec as it had the best performance without overfitting/underfitting. This could be the best model as the reviews are linearly related to the sentiment. Also, Linear SVC could probably handle a multitude of different features the best as well as sparse features.

The outcome of the project can help movie review sites like Rotten Tomatoes run different Linear SVC models to find reviews online on forums like reddit and label them positive or negative with roughly 87% accuracy. Likewise movie executives and producers can find reviews online about their movies and be 87% sure which movie was a hit. For example, Disney can be 87% sure that a movie was hit then it can make a sequel of that movie. By running these models, this can help movie related companies save time, money, and resources by not having people manually reading each review and labelling them as positive or negative. Therefore, the companies have the money to do more worthwhile projects.

Some improvements that could have been done on this project was that a Keras model could have been used. Another improvement could be having different reviews from different sites to go along with IMDB reviews. More hyperparameter tuning could have been done as well to see that improves the model performance or not. Tpot could have been run longer to possibly find a better model.