Sentiment Labeling on Short Movie Reviews

Zihan Shao
School of Arts and Science
New York University Shanghai
Shanghai, China
Email: zs1542@nyu.edu

Yuqian Sun
School of Arts and Science
New York University Shanghai
Shanghai, China
Email: ys3982@nyu.edu

***Abstract*—Sentiment has long been seen as a unique characteristic that only humans hold. But with the development of technology, it is becoming possible for computers to predict the sentiment contained in human language. In this project, we build several binary classification models that analyse short movie comments to predict whether it delivers a positive or negative sentiment, which can make qualitative judgments on audience' reflection of the movie. The procedure of building the model involves a data preprocessing part with many, then with word embedding methods like Word2Vec, TFIDF, and classifier SVM, SVC, Random Forest, Naive Bayes, FCNN. The predicted sentiments were evaluated by accuracy. The results show that word embedding method TF-IDF followed by a simple FCNN outperforms all other models with 91% accuracy. In future, more effort should be devoted to a better word embedding method or a trainable word embedding method.**

I. Introduction

With the rapid development of movie industry and social media, it is very likely that people may leave a short comment on social media after watching a movie. The sentiment involved in the comment may provide important feedback to the movie producers and other people in the industry. In particular, the proportion of positive(negative) comments is an important reference for them, while labeling each comment one by one manually will be extremely inefficient and troublesome.

Given this background, in this project, we are going to build classification machine learning models to predict the sentiments (either positive or negative) delivered in short movie comments. With this model, people may aggregate the number of "like"s(positive) and 'dislike"s(negative) in numerous movie reviews.

In this paper, we will first deliver a preview of the data set, including our methods of text preprocessing. Then we will introduce our general methodology and some unsuccessful early trials, which finally leads to our final model. After that, we will look into the operations we have done to optimize the model. In these parts, We will explain the reason why we pick or not pick a certain method. In the end, we will offer a detailed view of the result of the project, some interesting facts and for future improvement.

II. Data Set

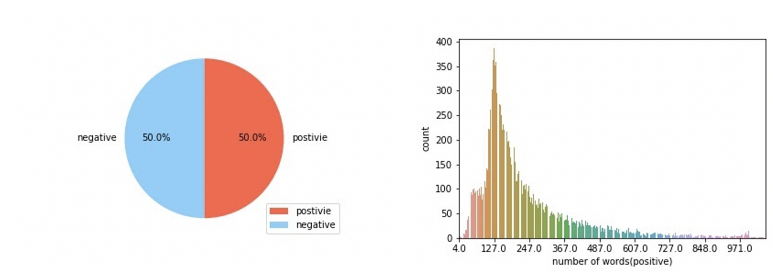
A. Overview of the data

The data set used in the project is **IMDB Dataset of 50K Movie Reviews**[1] from **Kaggle**. This data set contains 2 type of the data, the comment and their corresponding sentiment label("positive" or "negative"). As the name suggests, there are in total 50,000 instances and it is a balanced data set with equal number of positive instance and negative instance.

The information for words in each review is listed as followed.

Attributes	Value
Max Length	4(words)
Min Length	2470(words)
Mean	231(words)
Variance	29358.18
Standard Deviation	171.34

Here's the visualization of some attributes of our data. (see figure1)



Distribution of positive and negative comments

Distribution of total word count

Fig. 1. Some attributes of the data set

B. Data Preprocessing

To enhance the performance of models, we usually preprocess the texture data before training. The left hand side of figure(see fig 2) shows the regular text preprocessing routine. However, due to performance concern, several adjustments are made to make a better word list for vectorization and later training. Here's the step by step explanation. Note that the steps that are distinct from the regular routine are marked $*$.

1) Removal of numbers
2) Lowercase*

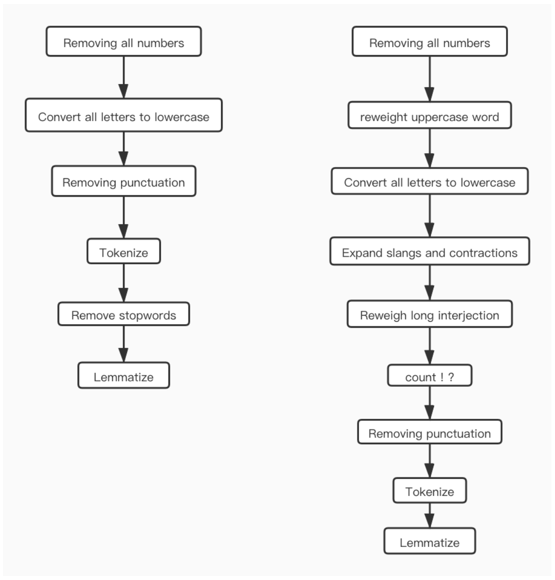


Fig. 2. pipeline for text preprocessing

In order to better classify words and clear format, the step of unifying all the letter cases is necessary. However, as we are dealing with the sentiment involved in the text, one should notice that words the words with all uppercase letters indicates a stronger emotion than the same word in lowercase. For example, "SURPRISE" delivers a more intense mood than "surprise". The improved method is that the weight of those uppercase words is multiplied by the number of uppercase letters(e.g. "AMAZING" is equivalent to amazing for 7 times). After that, we convert all the letters into lowercase.

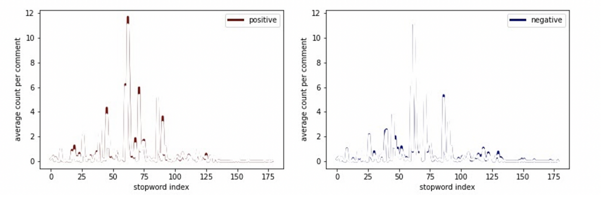
3) Slangs and Extractions*
The step of expanding contractions and slang is also crucial for the model since term like ''ll" create a lot of noise. We find some corpus that enable us to do replacement between . This step should be done between lowercase and removal of punctuation since the words in the dictionary are all lowercase words and the punctuation matters in the expanding process.

4) Reweighing interjection*
There are many similar long interjections in our dataset, like "haha", 'hahaha", 'hahhhhaaaa". Those words express the same feeling but are counted as different word, which causes inefficiency. Therefore, we simplify some long expressions of interjection and reweigh them based on their length. For words whose lengths excess a certain threshold(set to be 8), we find the simplest version of the word and duplicate it with its length divided by its simplified length.

5) Removal of punctuation*
Most punctuation characters play a role in separating sentences. While some punctuation characters like question marks(?) and exclamation mark(!) conveys strong emotions. For example, the sentence ending with a "?" adds to the doubted feeling into the meaning and the exclamation undoubtedly strengthens the positive feeling in that sentence. Therefore, we count the occurrences of questions mark and exclamation mark before removing all the punctuation characters and treat them as words.

6) Tokenization
Tokenization helps split the sentence with regard to space and other characters predefined. This step makes a fair division of the sentence, without which we cannot manipulate the text. In our project, we use `nltk.tokenize` to do tokeinization.

7) Stopwords*
Stopwords are words that don't add meaning to a sentence. Words like propositions, conjunctions etc all belong to the range of stopwords. In most of the NLP project, removal of stopwords is always applied during data preprocessing. However, some of early trials indicates that removal of stopwords even reduces the accuracy.
In order to analyse the function of stopwords in the two classes, we plot two figures to look into the distribution of stopwords in both classes.(see figure3)



Distribution of stopwords in Positive class and Negative class

Fig. 3. Distribution of stopwords

In these two figures, the x-axis means the index of stopwords list (from `nltk`), the y-axis represents the average occurrences of that particular stopwords in one movie review. It is shown that the positive class(the dark red line) and the negative class(the dark blue line) have superiority over one another at some stopwords, which indicates stopwords may help us to do the classification. Therefore, removal of stopwords is not executed in our project.

8) Lemmatization
Lemmatization is to extract the lemma of each word. For example, best, well$\rightarrow$ good, cried$\rightarrow$cry, movies$\rightarrow$movie. Lemmatization can aggregate words of different forms but essentially the same, making future operations more efficient by reducing the size of the word list.

C. Train-Validation-Test Split

we have 50000 reviews in total. We are going to use 35000 for training, 5000 for validation, and 10000 for testing.

III. Methods

In this section, we are going to discuss how we tackle the problem. We will first introduce the very general methodology to solve the problem. Then we will go over some early trials with word2vec, which finally get us to our final model with TFIDF. In the end, we show you how we optimized our model.

A. General Methodology

Before going to any actual model, we need to nail down some basic

1) General Pipeline
   We takes the usual routine of Natural Language Processing. We first convert the text, which cannot be directly manipulated by computer, into feature vectors(Word Embedding). Then we build a classification to model to output the label.
   In our project, we will build the model based on the word embedding in the first place.
2) Loss function
   We mainly use cross entropy loss since it is a binary classification.
3) Evaluation
   Since positive sentiment and negative sentiment are equally important in real life application, we use the accuracy as the metric to evaluate the performance of our models.

B. Early trials with Word2Vec

We first tried Word2Vec for word embedding.

Word2Vec adopts a neural network that converts each distinct word into a unique vector based on its context. Since it takes the context into consideration, the feature vector generated will well depict the relationship between words. For example, vectors of similar words(such as subway and metro) should have smaller minkowski distance in the vector space. Anotehr example is, if everything goes on smoothly, the difference between the feature vectors for king and queen, should be equivalent to the difference between man and woman.

The reason why we pick up the Word2Vec in the first place are as followed.

- It takes the context of each word into consideration.
- Since the feature vector we generates is essentially the nodes in the hidden layer, the size of the feature vector can be arbitrary number. This can be really helpful, since we have a quite large corpus(data set) we may need to reduce the dimension of the feature vector.

For implementation, We have tried both training Word2Vec model on our data set and adopting the pretrained Word2Vec model. For the former, We use the class `gensim.Word2Vec` to generate word vectors based on our Data set. For the latter, we adopt the `Wiki-words-500-with-normalization` from TensorHub.

Followed is our result obtained by using Word2Vec for word embedding process, followed by some usual classifiers. For comparison, we also show the result obtained by Onehot encoding with a Naive Bayes classifier, which might be the most elementary method.

Word Embedding Method	**Classifier**	**Accuracy**
Word2Vec(Gensim)	SVM	84.6%
	Random Forest	84.2%
	FCNN	84.6%
Word2Vec(pretrained)	SVM	84.8%
Onehot Encoding	Naive Bayes	84.5%

As is listed in the table, the performances of the models with Word2Vec are much the same as the performance of the most elementary algorithm. This actually means the trial is far from satisfactory since the essence of Word2Vec is a fully connected neural network, which is much more complex than Onehot encoding with Naive Bayes.

C. Final model

After many rounds of trials, we finally find to our best algorithm, which adopts TFIDF as the word embedding method followed by a light weight neural network.

TF-IDF is the abbreviation for Term Frequency - Inverse Document Frequency. It is in some way an enhanced version of Onehot encoding. Like Onehot encoding, the dimension of the feature vector generated by TF-IDF depends on the size of the dictionary as well. However, instead of using 1 or 0 to represent in or not in, TF-IDF to give every word a weight based on its importance. As the name suggests, it is basically the product of term frequency and the inverse document frequency. Let d be a document (one movie review in our case) in the corpus D (all movie reviews in our case), and let t be a term(word). The rigorous mathematical definition is as followed.

$$\text{TF}(t,d) = \frac{\text{numbers of occurrences of } t \text{ in } d}{|d|}$$

$$\text{IDF}(t,d,D) = \lg(1 + \frac{|D|}{|\{d' \in D : t \in d'\}|})$$

$$\text{TFIDF}(t,d,D) = \text{TF}(t,d) \cdot \text{IDF}(t,d,D)$$

The essence behind the mathematical formula is quite intuitive. If a word is frequently occurring in one text piece (movie review) but not in others, it should be a important word in the text(higher TF-IDF value), vice versa.

For implementation, we use `Tfidfvectorizer` from `sklearn`.

The results are as followed. Note that the fully connected neural network applied here consists of two hidden layer with relu activation(sigmoid for output) which have 16 and 32 hidden node respectively. Here we keep the results obtained by Word2Vec and Onehot encoding for comparison.

Word Embedding Method	Classifier	Accuracy
TFIDF	SVM	87.6%
	Random Forest	83.2%
	FCNN	88.3%
Word2Vec	SVM	84.8%
Onehot Encoding	Naive Bayes	84.5%

As is shown in the table, by using TFIDF for word embedding, we reached around 88%, which behaves much better than Onehot encoding and Word2Vec.

We also tried to combine these two word-embedding method together. We tried to re-weight the vector obtained by Word2Vec by the vector generated. Let $w = [w_1, \ldots, w_n]$ be the vector generated by TFIDF, $V = [v_1, \ldots, v_n] \in \mathbf{Mat}(n \times 500)$ where each row vector v_i is the Word2Vec vector for i-th word in the text. The new feature vector after re-weighing is given by

$$v' = \sum_{i=0}^{n} w_i \cdot v_i$$

or in matrix form

$$v' = w \cdot V$$

It is worth mentioning that, in practice, implementation with matrix multiplication by `numpy` is much more efficient than adding the re-weighted vector one by one. In our model, the latter method takes nearly 2 hours to process 35000 reviews, while the matrix multiplication reduce the time to only 5 minutes.

After manipulating the feature vector, We applied the same fully connected neural network as in the previous experiment here. The result obtained is as followed.

Word Embedding Method	Classifier	Accuracy
TFIDF+Word2Vec	FCNN	87.6%

As we can see, though the performance of combination of TFIDF and Word2Vec is better than Word2Vec only, it is no better than using pure TFIDF.

Therefore, we use TFIDF for the Word Embedding for our sentiment analysis tool. As for the further classifier, Though SVM and FCNN has approximately the same test accuracy, we eventually select FCNN. The reasons are listed as followed.

- It is very possible that the feature vectors are not linear separable. Also, due to the high dimensionality, it may be hard to implement the correct kernel. However, these problems can be easily handled by neural network. In other words, we think neural network has larger space for improvement when tuning parameters.
- Some readers may worry about the cost of neural network may be much higher than SVM. However, because of the high dimensionality, training time for SVM(around 15 minutes) is much longer than fully connected neural network with mini-batch gradient descent. Though we can also apply stochastic gradient descent to SVM to reduce the run time significantly as well, yet SVM does not outperform FCNN neural network largely.

Up to now, we have nailed our model, which is TFIDF followed by a fully connected neural network. The rest is optimize the hyperparameter.

D. Optimization

In this part, we are going to show you how we optimize our model.

1) The structure of the fully connected neural network
Since there's no closed form for designing a good neural network, we tried a lot of well-known structures, including but not limited to $[1-16-32-1]$, $[1-32-64-1]$, $[1-64-128-1]$, $[1-32-64-128-1]$, $[1-32-64-32-1]$, $[1-128-1, 1-64-1]$(The number represents the number of neurons in each layers).
Based on the accuracy, we finally use the neural network with only one hidden layer with 32 node,and Adam for Gradient descent.
2) Avoid Overfitting Since we are using a quite shallow neural network for accuracy concern, there's a great chance for overfitting (see figure). To avoid this, we do the following:
   - Apply early stop. For implementation, we use `Earlystopping` class from `tensorflow.keras.callback`.
   - Dropout Layer. We implement an extra Dropout layer to avoid input layer
   - We tried to apply L2(L1)-regularization. However, though it did solve the problem of overfitting by a little bit, it also reduced the validation accuracy. Therefore, we are not using these two regularization methods in our final model.
3) Max document frequency
Document frequency is a hyperparameters in the part of TFIDF vectorizing process. The max document frequency means, if the document frequency is higher then some threshold, it will be ignored during the word embedding. The intuition behind it is straight forward, since some recurring word (such as "movie", "one", etc.) does not help with further sentiment labeling. This can help our model in two ways. On the one hand, it can reduce the dimension of the feature vector. On the other hand, since in text pre-processing part we didn't remove stop words in case some of them do help us to tell the sentiment. However, some of the stop words are trivial(such as "I", "the", "you", etc.).
By doing gird search on interval $[0.4, 0.7]$ with a step size of 0.05, we set the max document frequency to be 0.6, which means the word that appears in more than 60% of the reviews will be ignored.(see fig 4)
4) Threshold for output
Since we use sigmoid function for the output layer

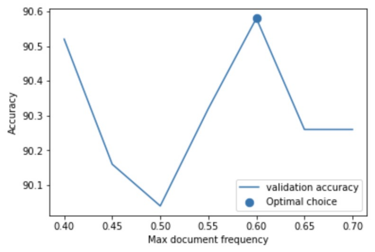


Fig. 4. Tuning max document frequency

in the FCNN, we are actually output a continuous numerical value between 0 and 1. We usually use a threshold to round it into 1 and 0 as the categorical value representing "positive" and "negative". By default, we set the threshold 0.5.
To increase the accuracy, we do the grid search on the validation set (5000 reviews) with in the interval $[0.4, 0.6]$, with a step of 0.03. It turns out that 0.55 is the best for accuracy.(see fig 5)

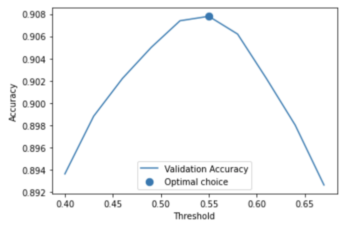


Fig. 5. Tuning threshold

Now we have our final model, consists of a TFIDFvectorizer and a fully connected neural network. See pipeline in fig 6.

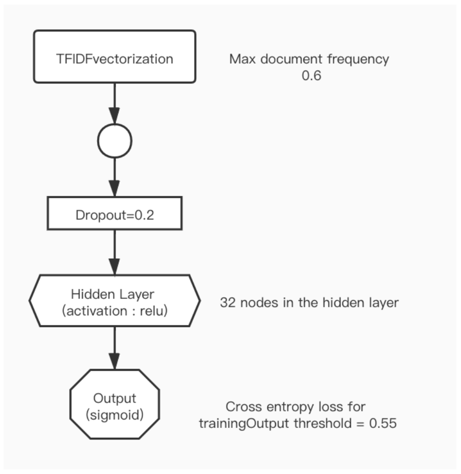


Fig. 6. pipeline

IV. Results, Discussion & Future improvements

As is shown in previous sections, the combination of TFIDF(as the Word Embedding) and a fully connected neural network(as the following classifier) has the best performance. After optimization, the model reaches the testing accuracy of around 90.7%. According to the confusion matrix(see fig 7), the model performs almost equally in labeling "positive" and "negative" review.

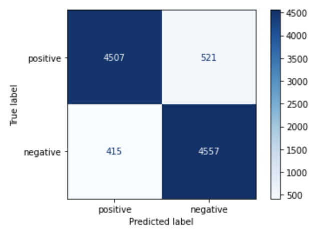


Fig. 7. confusion matrix

In the rest of this section, we are going to discuss

- Strengths and shortcomings of the model
- Potential reason for good performance of TFIDF
- Future Improvement

A. *Strengths and Shortcomings of the model*

Our model did a good job in the following aspects.

- The accuracy, which is around 90.7%, is quite satisfactory. This result is better than most of the model published by other users on **Kaggle** using the same data set.
- Efficiency. The cost of our model is really low since it only consists of a TFIDFvectorizer and a light-weight neural network. Training, validation, and testing process take no more than 8 minutes. However, nearly all the well-trained(**i.e.** accuracy ≥ 89)model published on **Kaggle** adopts some complex neural network (`LSTM`, `GRU`, or even `Bert`).

Our model needs to improve in the following aspects.

- Overfitting. In this project we have tried to use Dropout and early stopping to avoid overfitting. However, the model is still very likely to be overfitted.
- Limitation on topic. Since we use TFIDF for word embedding, which calculates "relative" frequency of the word. Therefore, once our data set consists not only movie reviews but text with diverse topics, the performance of our model is not granted since the diverse corpus may add a lot of noise. For example, one word may carry a strong negative sentiment under topic A but not in topic B. This will cause some misclassifications to some extent.

B. *Analysis on some interesting facts*

As is shown in previous sections, the performance of Word2Vec is far from satisfactory while TFIDF performs fairly well. However, TFIDF is a relatively simple model while

Word2Vec use a neural network which even takes the context into consideration. The reason may be as followed.

Word2Vec and TFIDF have very different ideas. TFIDF, in some way, is an enhanced version of Onehot encoding. It calculates the "relative frequency" of each word. Intuitively, the classification is based on the cumulative weight of "negative" words and "positive" words. The Word2Vec, however, attributes each word a vector which is based on the relationship between the words. Every word in the sentence moves a little bit in a certain direction, and we make the classification on the final state when the sentence finished. (see figure 8)

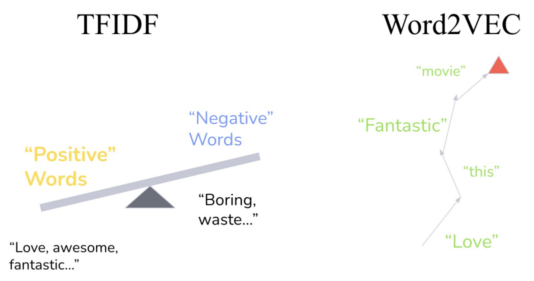


Fig. 8. visualization of the idea in w2v and tfidf

It is hard to say which idea is better. However, it is evident that TFIDF performs much better in this case. One possible reason is the average length of our corpus is too big, which is not good for Word2Vec since there's too much noise.

C. *Future Improvement*

If we have infinite time, we may keep working on

- Looking for a better fully connected neural network as the classifier
- Solving the problem of easy overfitting
- TFIDF discards some properties of text, including tune and context. We may need find a better word embedding tool to take those properties into consideration. Furthermore, we may try to make the word embedding process trainable to further enhance the performance.

V. Conclusion

In conclusion, we tried various way to label the sentiment of movie reviews. We first tried Word2Vec for Word Embedding, which gives us some unsatisfactory results comparable to Onehot encoding & naive bayes regardless of which followed classifier we use. It motivates us to change the method for word embedding and we finally get to TFIDF. In the end, TFIDF followed by a fully connected neural network gives us the best accuracy of 90.7%, which is quite satisfactory. In the future, more effort should be spent on a better word embedding which takes various properties into consideration. Also, we can try to make the word embedding process trainable to further enhance the performance.

Acknowledgment

We acknowledge Prof. Li Guo and teaching assistant Yunjie Song for conducting such a miraculously rewarding Machine learning course this semester, without which we would not be able to complete the above paper.

References

[1] IMDB Dataset of 50K Movie Reviews. (2019, March 9). Kaggle. https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews