CS230

Sentiment Analysis behind Text with Different Length and Formality

Jiwen Chen (jiwchen@stanford.edu)
Department of Mechanical Engineering, Stanford University

Abstract

The purpose of the project is to evaluate the capability of sentiment analysis for different models on dataset with various text length and formality. Three models, logistic regression with TF-IDF, RNN with LSTM layer and word embedding, RNN with LSTM layer, word embedding and an average pooling, are tested on each of the three different datasets. The result shows that for twitter tweets, RNN with LSTM and pre-trained embedding has the highest accuracy, for IMDB reviews, logistic regression with TF-IDF has the highest accuracy, and for Yelp reviews, RNN with LSTM and trainable embedding has the highest accuracy.

1 Introduction

Sentiments are hidden behind online comments on social media of all kinds. For a newly issued proposal, a newly launched movie, or a coming festival, there will be tons of comments showing public opinions. The extracted sentiments are important for prediction purpose, for example the approval of a proposal, the success of a film, or the degree of recognition of a festival. Therefore, instead of navigating the comments thoroughly, a more efficient method to extract the sentiments behind texts is needed. We proposed a logistic regression with TF-IDF, RNN with LSTM layer and word embedding, RNN with LSTM layer, word embedding and an average pooling to evaluate their classification capability for online comments varying in length and formality. With the English text of an online comment as the input, the model should predict the sentiment as either negative or positive with high accuracy.

2 Related work

Techniques for sentiment analysis have been developing in recent years. In general, sentiment analysis include both the processes of feature extraction and sentiment classification [1]. A typical feature extraction process includes data preprocessing, TF-IDF, and selection methods such as Odds Ratio and Chi-Square [2]. The sentiment classification methods mainly include two approaches: lexion-based, and machine-learning based, where support vector machines, neural networks and trainable bayesian networks are involved [1].

In 2016, Li has proposed an RNN with LSTM layer model. Theoretically, RNN covers the time-order structure of the whole text and the LSTM layers solve the problem from the long interval between the related previous texts and the current location [3]. Li's model proved better performance than conventional RNN [3]. However, Li's model only ran on the English movie reviews, and whether such model performs well on text with different length and formality is unknown. This project aims at comparing three models (logistic regression with TF-IDF, RNN with LSTM layer and word embedding, RNN with LSTM layer, word embedding and an average pooling) on three datasets with text of different length and formality to see which model outperforms the others.

CS230: Deep Learning, Fall 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

Three datasets with text of different length and formality have been applied in the project. The first dataset is the twitter tweets, with 18467 tweets and an average text length of 12.9 words. The first dataset is split into train, valid, test set with 80%, 10%, and 10% ratio. The second dataset is the IMDB review, with the training set having 40000 reviews and an average text length of 231.3 words, the validation set having 5000 reviews and an average text length of 228.9 words, and the test set having 5000 reviews and an average text length of 231.9 words. The third dataset is the Yelp review, with 560000 reviews and an average text length of 133 words. The third dataset is split into train, valid, test set with 98%, 1% and 1% ratio. All the three datasets are from Kaggle [4][5][6]. All three datasets have a balanced distribution between negative sentiment and positive sentiment. Therefore, there is no need for data augmentation.

For data preprocessing, we applied punctuation removal, stress-mark removal with python unicode, word lemmatization with python nltk package, and stop-word removal. The stop-word vocabulary here excludes the negation words, such as "not" and "won't". The preprocessing procedures are shown in Figure 1 below with an example sentence.

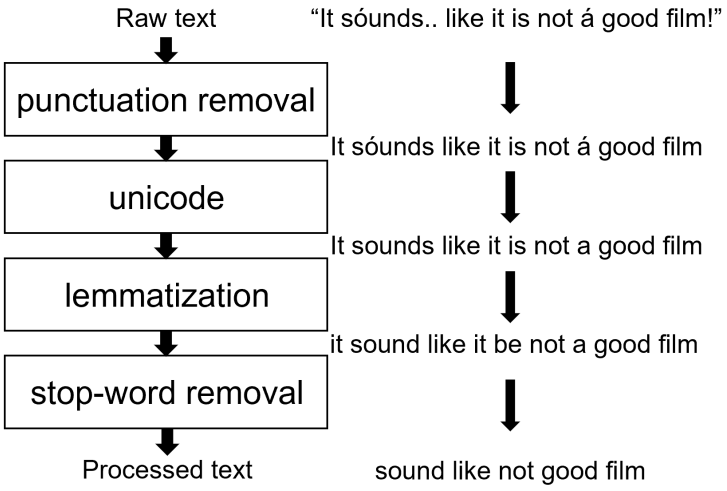

Figure 1: Dataset Preprocessing

After the preprocessing, the average text length of twitter tweets reduces to 7.9 words, the average text length of IMDB reviews reduces to 125.0 words, and the average text length of Yelp review reduces to 70.3 words. The text length distribution among samples show a right-skewed pattern. An example of Yelp review processed text length distribution is shown in Figure 2 below.

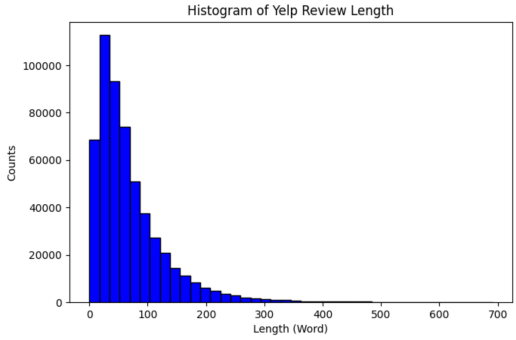

Figure 2: Processed Yelp Review Text Length Histogram

4 Methods

We proposed three models to test on different data sets: logistic regression with TF-IDF, RNN with LSTM layer and word embedding, RNN with LSTM layer, word embedding and an average pooling.

4.1 Logistic Regression with TF-IDF

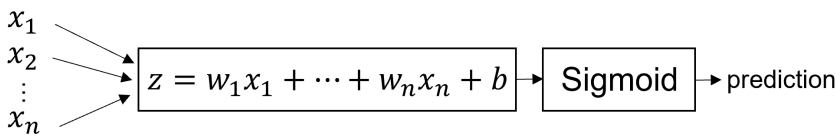

Figure 3: Logistic Regression

Figure 3 above shows the logistic regression model. The input of the model x_i is the value of the TF-IDF processed bag-of-word representation of the text sample for the i^{th} word in the vocabulary. The bag-of-word representation keeps track of the word frequency for each word in the vocabulary regardless of the sequence of the word's appearance. For example, the bag-of-word representation of the sentence "cat is cat" is [0 2 1] for a given vocabulary ["dog", "cat", "is"]. TF-IDF is an unsupervised feature extraction algorithm dealing data at the lexical level regardless of the sequence [2]. TF-IDF takes into account the words that are frequently appeared but not important for the classification purpose.

TF-IDF: TF-IDF score is the multiplication of both the term frequency (TF) score and the inverse document frequency (IDF) score, as shown in Eq. 1 [2]. The TF score is calculated in Eq. 2 and the smoothed IDF score is calculated in Eq. 3 [2].

$$TF - IDF(w) = TF(w) \times IDF(x) \tag{1}$$

$$TF(w) = \frac{\text{Number of times word w appears in a document d}}{\text{Total number words in a document d}} \tag{2}$$

$$IDF(w) = ln(\frac{\text{Total number of documents} + 1}{\text{Total number of documents with word w in them} + 1}) \tag{3}$$

The logistic regression model is implemented in python with l2 regularization and liblinear solver, which applies coordinate descent optimizer, an iterative algorithm that fixes most components of the variable vector x at rest for the current iteration, and minimizes the objective with respect to the remaining components [7].

4.2 RNN with LSTM layer and word embedding

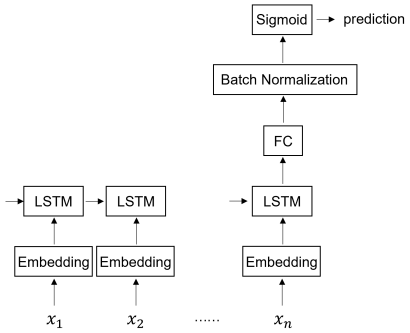

Figure 4: RNN with LSTM Layer and Word Embedding

Figure 4 above shows the RNN model with LSTM layer and word embedding. We applied two kinds of word embedding in this model: one of the word embedding is completely trainable according to

the vocabulary size of the training set, and the other word embedding is an open-sourced pre-trained GloVe word embedding from Stanford NLP lab [8]. The input of the model x_i is the tokenized representation of the i^{th} word from the text. The vocabulary for tokenization depends on the embedding matrix (vocabulary generated from the training set for using trainable embedding layer, while the GloVe vocabulary for using GloVe word embedding as the embedding layer). The loss function is the binary cross entropy. The optimizer of the model is Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $decay = 0.01$. Early stop keeping track of the validation accuracy is applied to avoid overfitting.

4.3 RNN with LSTM layer, word embedding and an average pooling

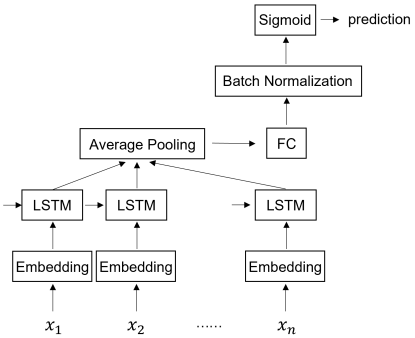

Figure 5: RNN with LSTM Layer, word embedding and an average pooling

Figure 5 above shows the RNN model with LSTM layer, word embedding and an average pooling. The average pooling takes the idea of averaging the word vectors while still retaining the consideration of the sequence of the word. The embedding layer is trainable. The loss function, optimizer and early stop method is the same with the RNN with LSTM layer and word embedding model.

For the inputs of RNN model (both model 4.2 and 4.3), since the sequence length should be kept same for training, a hyperparameter max_len is set. Zero-padding is applied to the input text if the text length is smaller than max_len and truncating is applied if the text length is larger than max_len. After observing the input text, the opinion sentences that have a high tendency to show the sentiments often take place in either the beginning or the end of the text, or both. As a result, instead of truncating in the front or in the back, we extracted the max_len/2 words from the beginning and max_len/2 words from the end for the sentences with word length larger than max_len.

5 Experiments/Results/Discussion

The hyperparameters for tuning include:

1. Logistic regression with TF-IDF: inverse of regularization strength (C), and intercept_scaling
2. RNN with LSTM layer and word embedding with and without an average pooling: learning_rate, drop_out_rate, trainable embedding layer dimension, LSTM hidden layer size, fully connected layer size, mini_batch_size, and maximum length of the input text (max_len).

The primary matrices we are considering in the project is the accuracy since we want the model to predict the sentiments as correct as possible. The accuracy is calculated in Eq. 4.

$$\text{Accuracy} = \frac{\text{True positive} + \text{True negative}}{\text{Total number of predictions}} \tag{4}$$

The hyperparameters are chosen to maximize the accuracy. Early stop is applied to all the RNN models in order to mitigate the overfitting. The accuracy comparison between the three models testing on three different datasets are shown in Table 1 in the next page.

Table 1: Comparison between accuracy of three models for different datasets

	Twitter tweet	IMDB review	Yelp review
Logistic regression with TF-IDF	87.71%	89.62%	93.63%
RNN with LSTM and trainable embed	88.52%	89.40%	94.21%
RNN with LSTM and pre-trained embed	88.96%	89.06%	93.04%
RNN with LSTM, trainable embed and avg pool	87.44%	88.92%	94.09%

For twitter tweets, RNN with LSTM and pre-trained embedding has the highest accuracy. For IMDB reviews, logistic regression with TF-IDF has the highest accuracy. For Yelp reviews, RNN with LSTM and trainable embedding has the highest accuracy. The other matrices (precision, recall and f1-score) and the confusion matrix for the most accurate models for each dataset are shown in Table 2 and Table 3 below.

Table 2: Precision, recall and f1-score for the best model for each dataset

		Precision	Recall	F1-score
RNN+LSTM+Pre-trained Embed for twitter tweets	Neg	85.88%	89.87%	87.83%
	Pos	91.62%	88.23%	89.89%
Logistic regression+TF-IDF for IMDB reviews	Neg	89.06%	90.03%	89.54%
	Pos	90.18%	89.22%	89.70%
RNN+LSTM+Trainable Embed for Yelp reviews	Neg	94.18%	94.31%	94.25%
	Pos	94.25%	94.11%	94.18%

Table 3: Confusion matrix for the best model for each dataset

		Neg	Pos
RNN+LSTM+Pre-trained Embed for twitter tweets	Neg	736	83
	Pos	121	907
Logistic regression+TF-IDF for IMDB reviews	Neg	2222	246
	Pos	273	2259
RNN+LSTM+Trainable Embed for Yelp reviews	Neg	2654	160
	Pos	164	2622

6 Conclusion/Future Work

In the project, we have tested three models for three different datasets. Accuracy is selected as the primary matrices to evaluate the performance. It is shown that for twitter tweets, RNN with LSTM and pre-trained embedding has the highest accuracy, for IMDB reviews, logistic regression with TF-IDF has the highest accuracy and for Yelp reviews, RNN with LSTM and trainable embedding has the highest accuracy. Comparing the three datasets in parallel, the difference between models are not huge but the size of the training set plays a pivotal role in the prediction accuracy. However, due to the small difference between models, and limitation on different size of the training sets, no deterministic conclusion could be drawn showing which model outperforms the others for various text length and formality. To further improve the result and discover into the problem, the future steps involve:

1. Test on more datasets with text in different length and formality with large training examples.
2. Add a misspelling correction preprocessing for non-word input, for example, correcting "nwe" to "new"
3. Finer tune the hyperparameters.
4. Explore more models, for example, RNN with attention.

References

[1] Medhat, Walaa, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey." Ain Shams engineering journal 5.4 (2014): 1093-1113.

[2] Madasu, Avinash, and Sivasankar Elango. "Efficient feature selection techniques for sentiment analysis." Multimedia Tools and Applications 79.9 (2020): 6313-6335.

[3] Li, Dan, and Jiang Qian. "Text sentiment analysis based on long short-term memory." 2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI). IEEE, 2016.

[4] Tweet Sentiment Extraction https://www.kaggle.com/c/tweet-sentiment-extraction/

[5] IMDB dataset (Sentiment analysis) in CSV format https://www.kaggle.com/columbine/imdb-dataset-sentiment-analysis-in-csv-format

[6] Yelp Review Sentiment Dataset https://www.kaggle.com/ilhamfp31/yelp-review-dataset

[7] Wright, Stephen J. "Coordinate descent algorithms." Mathematical Programming 151.1 (2015): 3-34.

[8] GloVe: Global Vectors for Word Representation https://nlp.stanford.edu/projects/glove/