Covid Tweet Sentiment Classification with Machine Learning and Deep Learning Models

Hao Xu
haoxu2020@u.northwestern.edu
https://github.com/MSIA/hxq9433_msia_text_analytics_project_2021

Abstract

Emotion detection is always subjective, and language based. Yet with the advancement of deep learning models, sentiment classification has become much doable in recent years. In this project, I trained and fine-tuned the parameters for several traditional machine learning models and deep learning models to provide a quantitative analysis of their performance based on Covid related tweets.

1 Introduction

Text classification is one of the fundamental and most typical tasks in natural language processing (NLP) and has a wide range of applications in the business world. With the large improvement in the social media platforms, there are now plenty of public text-based resources to build applications on. Recently, as covid uncertainty goes on, plenty of tweets were posted regarding people's opinion on covid situation. It would be interesting to have a model to perform social listening and track public opinions. This project trained and fine-tuned different text classification models (logistic regression, support vector machine, LSTM and BLSTM) and evaluated and compared their performances.

2 Related Work

2.1 Traditional Machine Learning Models

Models such as logistic regression, Naïve Bayes, k-nearest neighbor, support vector machine are still valid options for text classification. Used with different word embedding methods like TF-IDF endocding and Word2Vec feature representations (Mikolov, et al., 2013), traditional machine learning models can still yield decent results.

2.2 Deep Learning Models

Common deep learning models include recurrent neural networks (RNNs), convolutional neural networks (CNNS), Long short-term memory (LSTM), and transformer-based language models such as bidirectional encoder representations from transformers (BERT).

Deep learning models takes context into consideration and languages models like Bert and ELMO also has the advantage of polysemy disambiguation. Thus, deep learning models usually yields superior results. For example, ELMO looks at the entire sentence and then assigns an embedding to each word using bidirectional LSTM (Peters, et al., 2018). Whereas Google's BERT (2018) made further improvement with several transformers.

3 Dataset

The dataset used for the project is covid19 related tweets fetched from Twitter's API. The raw dataset contains information of 44955 tweets with their tweet time, location as well as manual labeled sentiment class tags. The data is already spitted into a training set of 41157 records and a testing data set of 3798 records.

Data statistics:

Summary Statistics	
Number of Reviews	44955
Average Length of Reviews	202.89
Standard Deviation of Length of Reviews	68.24
Minimum Length of Reviews	3
Length of Reviews (25 percentiles)	150
Length of Reviews (50 percentiles)	214
Length of Reviews (75 percentiles)	258
Maximum Length of Reviews	343

Table 1: Dataset Statistics

There are 5 sentiment classes and the distribution of them are as follows:

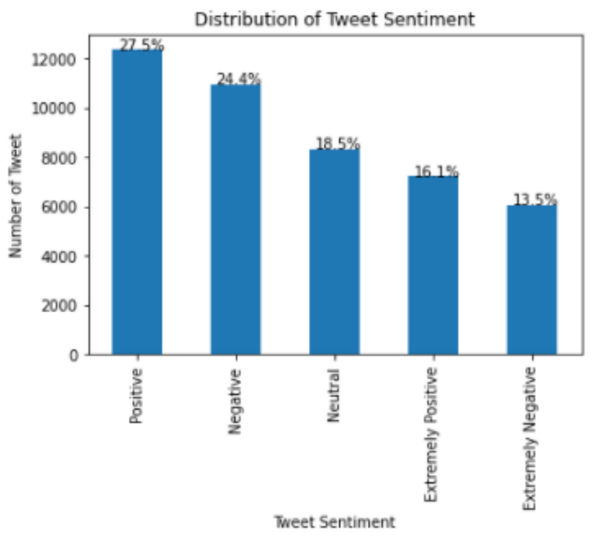

Figure 1: Distribution of Review Scores

4 Methods

4.1 Text Preprocessing

To clean up the raw tweets to a more digestible and usable format, all the raw tweets were preprocessed by removing special characters, numbers, removing common English stop-words, as well as converting texts to lower cases.

4.2 Models

I used 5 different models in this project for comparison. For each model, I tuned different model parameter combinations to get the best results.

Models implemented:

1. Logistic regression model
2. Support vector machine model
3. LSTM model
4. BiLSTM model
5. Bert (implemented but took too long to train so didn't include the performance scores in this paper.)

4.3 Model Training and Evaluation

TF-IDF score was used for word representation for the traditional machine learning algorithms (logistic regression and SVM). For LSTM models, texts were first tokenized and turned into vectors and then fed into an embedding layer to generate word embeddings. For Bert, pre-trained BERT model has its own word representations.

In terms of evaluation, I used metrics including accuracy, F1 score, precision and recall to evaluate the performances of different models on the testing dataset.

5 Results

5.1 Logistic Regression

To improve the performance, I tried different hyperparameters including penalty type 155 (penalty), the regularization parameter (C), and 156 the gram range of tf-idf vectorizer. The models tested and results are:

1. model_p=l1 c=1 ngram=(1, 1)
2. model_p=l2 c=1 ngram=(1, 1)
3. model_p=l1 c=2 ngram=(1, 1)
4. model_p=l2 c=2 ngram=(1, 1)
5. model_p=l1 c=1 ngram=(1, 2)
6. model_p=l2 c=1 ngram=(1, 2)
7. model_p=l1 c=2 ngram=(1, 2)
8. model_p=l2 c=2 ngram=(1, 2)

	Accuracy	Precision	Recall	F1
1	0.577	0.576	0.577	0.573
2	0.538	0.548	0.538	0.539
3	0.579	0.576	0.579	0.575
4	0.541	0.546	0.541	0.541
5	0.560	0.559	0.560	0.551
6	0.506	0.518	0.506	0.506
7	0.582	0.578	0.582	0.574
8	0.522	0.528	0.522	0.522

Table 2: Logistic Regression Model Results

5.2 Support Vector Machine

To improve the performance, I tried different hyperparameters including penalty type 155 (penalty), the regularization parameter (C), and 156 the gram range of tf-idf vectorizer. The models tested and results are:

1. model_p=l1 c=1 ngram=(1, 1)
2. model_p=l2 c=1 ngram=(1, 1)
3. model_p=l1 c=2 ngram=(1, 1)
4. model_p=l2 c=2 ngram=(1, 1)
5. model_p=l1 c=1 ngram=(1, 2)
6. model_p=l2 c=1 ngram=(1, 2)
7. model_p=l1 c=2 ngram=(1, 2)
8. model_p=l2 c=2 ngram=(1, 2)

	Accuracy	Precision	Recall	F1
1	0.555	0.550	0.555	0.551
2	0.525	0.525	0.525	0.524
3	0.526	0.522	0.526	0.523
4	0.516	0.516	0.516	0.515
5	0.560	0.555	0.560	0.553
6	0.527	0.527	0.527	0.525
7	0.545	0.543	0.545	0.540
8	0.524	0.525	0.524	0.522

Table 3: SVM Model Results

5.3 LSTM

To improve the performance, I tried different lstm model architectures tuning dropout layer and LSTM node number:

1. 20 dimensions embedding layer followed by 0.2 spatial dropout layer followed by 64 nodes LSTM layer. Trained on 10 epochs with 256 batch size.
2. 20 dimensions embedding layer followed by 0.1 spatial dropout layer followed by 64 nodes LSTM layer. Trained on 10 epochs with 256 batch size.
3. 20 dimensions embedding layer followed by 0.2 spatial dropout layer followed by 32 nodes LSTM layer. Trained on 10 epochs with 256 batch size.

	Accuracy	Precision	Recall	F1
1	0.692	0.707	0.692	0.695
2	0.700	0.715	0.700	0.702
3	0.665	0.671	0.665	0.664

Table 4: LSTM Model Results

5.4 BiLSTM

To improve the performance, I tried different lstm model architectures tuning dropout layer and LSTM node number:

1. 20 dimensions embedding layer followed by 0.2 spatial dropout layer followed by 64 nodes BiLSTM layer. Trained on 10 epochs with 256 batch size.
2. 20 dimensions embedding layer followed by 0.1 spatial dropout layer followed by 64 nodes BiLSTM layer. Trained on 10 epochs with 256 batch size.
3. 20 dimensions embedding layer followed by 0.2 spatial dropout layer followed by 32 nodes BiLSTM layer followed by self-attention and flatten layer. Trained on 10 epochs with 256 batch size.

	Accuracy	Precision	Recall	F1
1	0.700	0.709	0.700	0.702
2	0.699	0.705	0.699	0.701
3	0.660	0.671	0.660	0.661

Table 5: LSTM Model Results

6 Discussions

For the traditional machine learning models, both logistic regression and SVM yield around 50% F1 score. This shows that the limitation of the approach where we combine static word representation plus classification algorithm. LSTM models generally performs better than traditional models which reflects LSTM's ability to remember or forget sequential information in an efficient manner.

In theory, BiLSTM is expected to outperform normal LSTM for the ability to take both side of the context into considerations. But we didn't see a big difference in this experiment. This is probably due to the lack of hyperparameter tuning and small data size. We can also see that including a self-attention layer didn't improve the performance probably due to the same reason just mentioned. Overall speaking, the second LSTM model is the best in terms of both performance and training cost. As a result, I implemented this model to the API.

7 Conclusions

In conclusion, deep learning based model had the much better performance in compare with traditional ML methods. The tradeoff is the much longer training time and computing resources requirements. Besides, with enough computational power, transformer based models Bert might yield more promising results.

References

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2020). Deep learning based text classification: A comprehensive review. *arXiv preprint arXiv:2004.03705*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019, October). How to fine-tune bert for text classification?. In *China National Conference on Chinese Computational Linguistics* (pp. 194-206). Springer, Cham.

Wang, S. I., & Manning, C. D. (2012, July). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 90-94).