Project 2: Sentiment Analysis on Tweets

Li Zhan, Li Xinwei, Zhang Xingyue
EPFL, Switzerland

***Abstract*—The objective of this project is to predict whether a tweet message is positive or negative with a text classifier system, which is a sentiment analysis task. Sentiment analysis is one of the most fundamental natural language processing (NLP) tasks. We extensively experimented with preprocessing techniques and word representation methods. We also built several text classification models, among which the *BERT* model achieved the highest accuracy 86.9% and F1 score 86.8%.**

I. Introduction

Much modern data is unstructured text, and sentiment analysis is one of the typical tasks on textual data. Natural language processing (NLP) is a part of machine learning that deals with understanding, analyzing, and generating the languages that humans use naturally or communicate in order to interface with computers instead of machine language [1]. In the last decade, representational learning and deep neural network-style machine learning approaches have become commonplace in natural language processing. In this project we aim to predict if a tweet message used to contain a positive :) or negative :(smiley, by considering only the remaining text using Machine Learning and Deep Learning approaches.

In the following, we start with introducing data analysis and data pre-processing in Section II. In Section III, we describe the word representation techniques and turn them into tweet representation in Section IV. Classification models are given in Section V and Section VI concludes the performances of different models we used.

II. Data Analysis and Pre-processing

We initially conducted some analysis and visualization on the given dataset to gain some first insight of the data. The whole train dataset includes 1250,000 positive tweets and 1250,000 negative tweets. The test set includes 10000 unlabeled tweets. The number of words per tweet is concentrated between 10 and 25 as showed in Figure 1.

The following section describes our data pre-processing methods. Some of the chosen pre-processing techniques are based on [2].

A. Lower case

People often mix upper and lower case letters when tweeting. In such a context, the sensitive-case of letters does not affect the semantics, but makes computation more difficult. Thus, we treated all upper case letters as lower case ones.

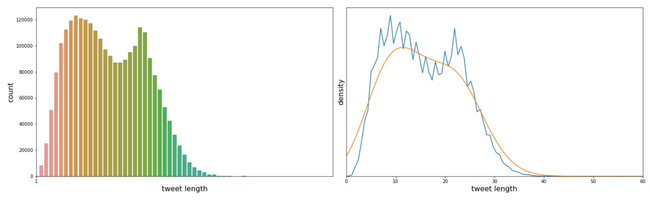

Figure 1. Density of tweet length in train set

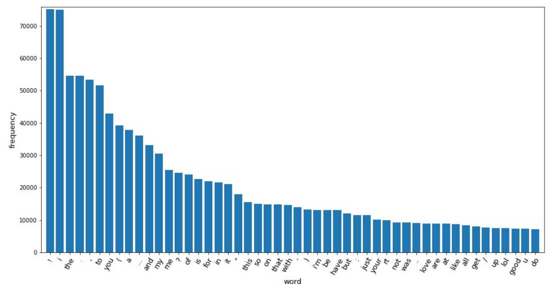

Figure 2. Word frequency in train set

B. Drop Duplicates and conflicts

We found a large number of repeated tweets, as well as tweets with conflicting sentiment labels. It will confuse the models, so we dropped them before feeding into the models.

C. Handle Punctuation and Emoji

After the de-duplication process, we used spaces to initially split the tweets and plot the word frequency in Figure 2. Punctuation marks such as ":", " (", "?", ")" and "!" appear in tweets with very high frequency. By combining punctuation marks, users can create an emotion icon, like ":)" and ":(", and they matter a lot in sentiment analysis. To imply the positive or negative emotions, we replaced the most frequently used emoticons with "happy" and "sad" respectively. The remaining punctuation was deleted.

D. Rewrite abbreviations

Since abbreviations enlarge the word embedding size and have no impact on the meaning of sentences, we rewrite them with regular phrases. Some users wrote tweets at will, so we saw expressions such as "isnt" and "youve" instead of "is not" and "you have". These words were also taken into consideration.

E. Remove hashtags and unnecessary words

People use hashtags to be linked under a given topic or tag someone to share a tweet. The topics are usually a sentence with no space in between and user names are not of our interest. This creates many random words if we simply remove the "#" and "@" sign, so we deleted the linked tags and user names. We then removed the unnecessary words such as "<user>" and "<url>". Some tweets only consist of tags, which then became an invalid tweet after previous steps. We replaced these tweets with an unmeaning word to ensure the completeness of prediction of the test set. Invalid tweets in the training data were deleted.

F. Process word stemming

By applying word stemming, every word will be reduced to its base form. We need to convert all words to their base form in order to reduce the number of unique words, and thus reduce the scale of word embeddings. In this work, we applied a powerful algorithm PorterStemmer [3], which is widely used in text classification tasks. Word stemming is not used in deep learning models because they follow a different approach. N-dimensional vectors are created instead of sparse matrices, so words from the same stem are close in the vectors and stemming becomes unnecessary.

III. Word Representation

In natural language processing tasks, it is necessary to consider how words are represented as mathematical embedding of continuous vector space. Usually, there are two types of representation: one-hot representation and distribution representation. The former requires significant computational resources to support and is less capable of feature extraction. We adopted the latter. This section describes the word embedding approaches we used.

A. Doc2Vec

The Doc2Vec model, in contrast to the Word2Vec model, allows the creation of word embedding set while directly obtaining a vector of the entire document. Since it is a direct segment vector, it takes into account the order of words and has better semantic information. The traditional Word2Vec model, on the other hand, uses averaged word vectors or clustering to obtain segment vectors without considering the order of words. There are two versions of Doc2Vec [4].

PV-DM model: The PV-DM model slide-samples a fixed length of words from a sentence at a time, taking one of the words as the predicted word and the others as the input words. The word vector corresponding to the input word and the paragraph vector corresponding to the current sentence are used as inputs to the input layer. The vector of the current sentence and the vector of words sampled this time are averaged or summed to form a new vector X, which is then used to predict the predicted words in this window.

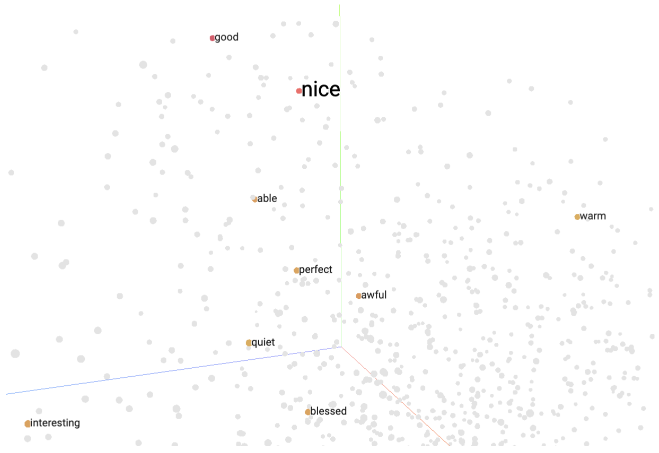

Figure 3. Search the 10 most similar words to "nice"

PV-DBOW model: The model is based on the same principle as the PV-DM model, but it ignores the context of the input and allows the model to predict a random word in the passage.

For most tasks, the PV-DM approach performs well, so we choose the PV-DM model. We tuned other parameters to be: min_count 10, window 10, vector_size 100, epoch_num 20.

To visualise the effect of embedding vectors, we use TensorBoard to visualise the relationships between high-dimensional word vectors. As shown in Figure 3, we searched the 10 most similar words to "nice" on TensorBoard.

B. GloVe

One observation that motivates the *GloVe* model is that the ratio of words better distinguishes relevant words from irrelevant words [5]. *GloVe* uses the global co-occurrence counts of words, which is different from skipgram or CBOW. This model is an unsupervised algorithm.

IV. Tweet Representation

A. Doc2Vec

We can simply get a tweet representation vector of the whole text by Doc2Vec. By feeding the cut word list of the tweet to a trained Doc2Vec model, we can obtain the tweet representation by *model.infer_vector()*.

B. Mean Aggregation

For some basic machine learning models in contrast to neural network models, it's easier to classify vectors rather than matrices. We can construct a feature representation of each training tweet by averaging the word vectors over all words of the tweet based on the fact that all word vectors are of the same length.

C. *Listing of Word Vectors*

Considering the ordering of words in the text, we can represent each tweet word by word instead of aggregation. Initially we need to cut all the tweets to the same length. We took the first 25 words after analysis of all tweets' lengths. We constructed a $25 \times D$ matrix as the representation of one tweet, where D is the dimension of one word vector.

V. Classification

A. *Basic Machine Learning Classifiers*

Before applying the deep learning approaches, we applied some basic machine learning algorithms such as Logistic Classifier, Random Forest, and Stochastic Gradient Descent Classifier as classifiers in conjunction to obtain the evaluation baseline. Among these models, the random forest model was the best performer, but it only had an accuracy of 65.2% with GloVe embedding and 69.3% with the Doc2Vec embedding.

B. *Bi-LSTM with Attention*

We used the Bi-LSTM classification model with Attention and achieved good results, with an accuracy of 83.3% (using Doc2Vec embedding) in the test set. The architecture of the model is shown in Figure 4. We applied hidden_size 128, and 3 layers of LSTM network.

The combination of the representations of words into the representation of a sentence can be done by adding them together, or by aggregation methods such as averaging, but these methods do not take into account the order of the words in the sentence. For example, in the sentence "I don't think this is good", the word "don't" is a negation of the word "good" that follows, i.e. the sentiment of the sentence is negative. Long distance dependencies can be better captured using LSTM models. This is because the LSTM learns through the training process what information should be remembered and what information should be forgotten.

However, there is a problem with modelling sentences using LSTM: it is not possible to encode back-to-front information, whereas with Bi-LSTM the semantic dependencies can be better captured in both directions.

Moreover, we applied the attention mechanism to our model. Attention model is about learning the importance of each element from the sequence and then combining the elements in order of importance. It allows for flexibility in capturing global and local connections in a single step. Each step of the attention model does not depend on the results of the previous step. Parallel computing reduces model training time[6].

C. *Transformer*

Transformer uses the attention mechanism as a building block, differentially weighting the significance of each part of the input data [7]. This allows the model to directly

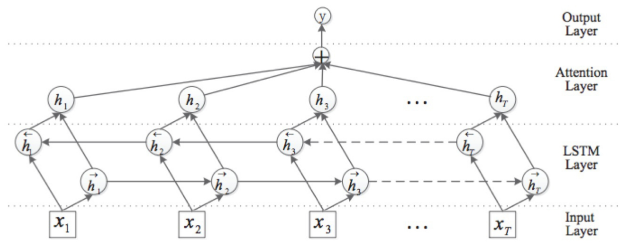

Figure 4. Architecture of Bi-LSTM with Attention

consider the relationships between different input words, rather than relying solely on the order in which they appear. We used GloVe pre-trained representations as word embedding. This would serve as the input in our model along with the positional embedding as presented in Figure 5. The Transformer model typically uses an encoder-decoder architecture, where the encoder processes the input sequence and the decoder generates the output sequence. In our task, the output sequence is typically a single class label, so the decoder consists of a single layer that maps the output of the encoder to the final class label.

We experimented with different parameters and chose embedding dimension = 100, number of epochs = 50 and batch size = 512. The Transformer model achieved an highest accuracy rate of 83.1% and F1 score of 83.5% using the full data-set.

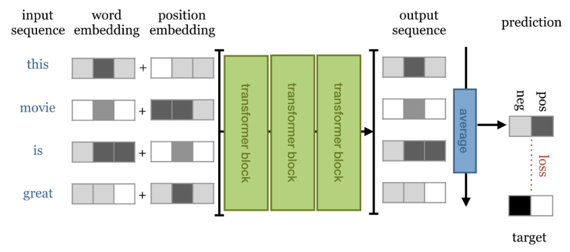

Figure 5. Building blocks of the Transformer model

D. *BERT*

Bidirectional Encoder Representations from Transformers (*BERT*) is a machine-learning approach for NLP. It is based on *transformer*. In the paper that proposed the *BERT* model [8], Devlin et al. mentioned that two current approaches for applying pre-trained language representations to downstream tasks, *feature-based* and *fine-tuning*, limit the power of the pre-trained representations. *BERT* is the improved model from this limitation. The significance of bidirectional pre-training for language representations is emphasized in this model. The pre-trained representations alleviate the requirement for heavily-engineered architectures [8]. Figure

6 demonstrates the overall pre-training and fine-tuning procedures for *BERT*.

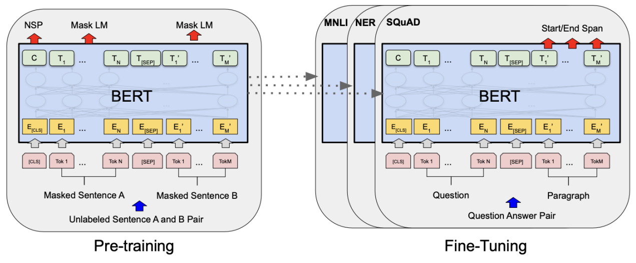

Figure 6. Overall pre-training and fine-tuning procedures for BERT

Only using 100,000 positive tweets and 100,000 negative tweets, we were able to obtain accuracy 86.9% and F1 score 86.8% using the *BERT* model. We used *Adam* as our optimizer, with learning rate $3e-5$, epsilon $1e-8$, and clipnorm 1.0. Other hyperparameter settings include batch size 64 and epochs 2.

E. *fastText*

fastText is a library for learning word embeddings [9] and text classifications [10]. It supports both supervised learning and unsupervised learning. The implementation of *fastText* is easy and efficient. We obtained accuracy of 85.3% and F1 score 85.6% on AIcrowd using the full dataset. We tuned the parameters to be: learning rate 0.1, epoch 2, and wordNgrams 3.

We also experimented with fastText's automatic hyperparameter optimization. Although convenient, the accuracy achieved was not as high as our previous model.

VI. Conclusion

The accuracy of our models is summarized in Table VI.

Model	Test Accuracy
GloVe + Random Forest (Baseline)	65.2%
GloVe + Transformer	83.1%
Doc2Vec + FCNN	72.6%
D2V.W2V + BiLSTM Attention	83.3%
BERT	**86.9%**
fastText + N-grams=3	85.3%

Among all our models, BERT shows the best performance. The experimental results fully demonstrate the superiority of deep learning on textual data. Neural networks generally adopt a multi-layer crossover network architecture for learning model capabilities. With the powerful multi-layer nonlinear capability, the model capabilities obtained from deep learning can more accurately match the realistic data distribution, learn about the sequential nature of data, polysemy, etc. and thus have a more powerful and accurate generalization capability.

References

[1] What is natural language processing? intro to nlp in machine learning. 2022, July. [Online]. Available: https://www.gyansetu.in/blogs/what-is-natural-language-processing/

[2] Updated text preprocessing techniques for sentiment analysis. 2021, August. [Online]. Available: https://towardsdatascience.com/updated-text-preprocessing-techniques-for-sentiment-analysis-549af7fe412a

[3] The porter stemming algorithm. [Online]. Available: https://tartarus.org/martin/PorterStemmer/

[4] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, 05 2014.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[6] Y. Wang, M. Huang, X. Zhu, and L. Zhao, "Attention-based LSTM for aspect-level sentiment classification," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 606–615, Nov. 2016. [Online]. Available: https://aclanthology.org/D16-1058

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[10] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.