ML Project 2: Twitter Text Classification

Po-Jui Chang
Department of Computer Science
EPFL, Switzerland

Yi-Hsin Jen
Department of Computer Science
EPFL, Switzerland

Si-Ying Lai
Department of Life Science Engineering
EPFL, Switzerland

***Abstract*—In this project, our goal is to classify the sentiments of Twitter text, either positive or negative. We preprocessed the raw data with hashtags detection, emotions replacement, contraction resolving, and time replacement for better data analysis. After that, we implemented several machine learning approaches such as Naive Bayes, Support Vector Machine, and Random Forests to train the classifier. Besides, we also apply two deep learning models, BERT and XLNet. Our experimental results show that BERT-Base-Uncased with 2 epochs has the best accuracy in both validation and online test.**

I. Introduction

Twitter is a micro blogging service that has a large and rapidly growing user base. Users on Twitter can create short status messages called tweets. In recent years, sentiment analysis of a tweet has become a trendy research topic.

To address this problem, in this paper, we implemented a data preprocessing pipeline to handle the special format of twitter messages. The whole process includes hashtags detection, emoticons replacement, contraction resolving, and time replacement. Based on the refined training data, we then applied different machine learning methods to train a sentimental classification model. Experimental results show that our best model, Bert, can reached 0.908 accuracy on the online test.

The rest of this paper is organized as follows: Section II presents the problem formulation. Section III describes the data processing pipeline. Section IV describes the model and methods we used. Section V provides experimental results, Section VI discusses the strengths and weakness of the results, and Section VII concludes this paper.

II. Problem Formulation

Given training data sets which contain tweets and the sentiment, train a machine learning model to classify whether a tweet is in positive sentiment or negative sentiment. The performance is evaluated by the testing data set provided by AICrowd.

III. Data Preprocessing

In this section, we first introduce our text preprocessing approach which is simplified by the approach provided in [1]. Hashtags detection, emoticons replacement, contraction resolving, and time replacement are described in III-A, III-B, III-C, and III-D, respectively. Then, we applied word embedding and tokenization to generate features for the machine learning model. These two approaches are described in III-G and III-F. In the end, we explain why we skip some common steps of natural language processing in III-H. The whole process flow is shown in fig. 1.

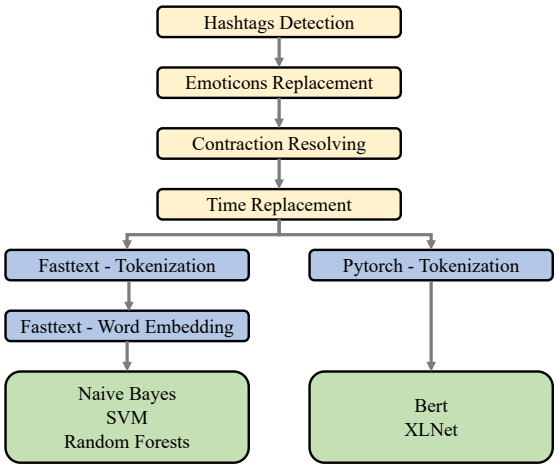

Figure 1. Data Preprocessing Pipeline.

A. Hashtags Detection

In Twitter, users are able to add a # token to tag tweets pertaining to a category. However, adding any characters in front of a regular vocabulary will increase the number of features. To avoid feature explosion, we then replace the original tag with a fix tag and a regular vocabulary separately. (e.g. replace "$\#Good$" by "$< hashtag > +Good$")

B. Emoticons Replacement

Nowadays, people like to use emoticons as a easy way to express their emotion on the internet. Therefore, emoticons are important features for us to classify the sentiment of a tweet. However, most tokenizers could not recognize emoticons. To address this problem, we manually looked for the emoticons in the training data set and assigned each emoticons with a emotion tag. Then, we used regular expression to replace up to 15 emoticons by the corresponding assigned tags. (e.g. replace " $:')$" by "$< happy >$")

C. Contraction Resolving

Since a computer does not know that contractions are abbreviations for a sequence of words, it might consider "we're" and "we are" to be two completely different things and does not recognize that these two terms have the exact same meaning. Besides, contraction might increase the dimensions of features. Therefore, we resolve contractions to solve both two problems.

D. Time Replacement

Comparing to general numbers, time can sometimes imply the emotion of the user. To catch this feature. We then replace the time format string by a fix time tag. (e.g. replace "$00:01$" by "$<time>$")

E. Remove User

In social media like Twitter, people often use interesting words as usernames. Some of them are merely names with some numbers, yet some contain nouns or adjectives, which have nothing to do with the whole context, but still effect the training outcome. Therefore, we remove the username in our data set, for better training and testing.

F. Tokenization

In order to get the computer to understand any text, we need to separate the text to into smaller units called tokens.

Both [2] and [3] provides built-in tokenization functions. In [2], tokenization is combined with the word vector generation, so we applied it on the models required word embeddings. Otherwise, we applied tokenization tool provided in [3].

G. Word Embedding

To map a pure text to a bunch of features, we need to convert words to vectors, and then we could describe each pure texts as a vector by averaging the word vectors of its content.

In [4], the authors present a state of art word embedding models named Continuous Bags of Words (CBOW). The CBOW model is a deep learning model that takes surrounding words as input and the center word as output. The word embeddings can be extracted by embedding layer in the neural network.

In our experiments, we directly applied the CBOW tool provided in [2].

H. Difference With Common NLP

In many natural language processing tasks, lower casing and removing stop words are two important steps to reduce the number of features. However, capital letters sometimes imply the emotion of the writer, which is important in sentiment analysis. Besides, many stop words such as "not" contain strong emotion. Removing these words might reduce the accuracy while dealing with a sentiment analysis task. Therefore, we do not apply lower casing and stop word removing to the training data set in our experiments.

IV. Model and Methods

In this section, we introduce five different approaches we attempted. Among them, Naive Bayes, Support Vector Machines, and Random Forests are traditional machine learning methods based on word vectors. The remaining two method, Bert and XLNet are the state-of-art deep learning approaches presented by google.

A. Naive Bayes Classifier

Naive Bayes is a traditional probabilistic model. We briefly introduce the model below.

Given a input $x = (x_1, x_2, ..., x_n)$ where x_i represents features of input data, the probabilities that the input belongs to class C_k can be denoted as

$$P(C_k|x_1, x_2, ..., x_n). \tag{1}$$

If we assume all features are mutually independent, we can apply the Bayes Rule and the probabilities can be expressed as

$$P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{2}$$

Based on the equation, we could construct a classifier that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \arg\max_k P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{3}$$

In our work, we already generate features by word embedding in previous section. Therefore, we directly applied the Gaussian Naive Bayes tool provided by [5] on our training data.

B. Support Vector Machines (SVM)

Support Vector Machines are supervised learning models that used on both regression and classification problem. Support Vector Machines correspond to the following optimization problem:

$$\min_w \sum_{n=1}^{N} [1 - y_n x_n^T w]_+ + \frac{\lambda}{2}||w||^2 \tag{4}$$

where x_n and y_n denotes the predictor and the outcome respectively, λ parameter is the regularized term, and w is the model.

The detail derivation and the proof are presented in [6]. Similar to the Naive Bayes Classifier, in our experiment, we directly applied the SVM tool provided by [5] on our training data.

C. Random Forest

Random Forests are ensemble learning methods for classification. The classifier is constructed by a cluster of decision trees. The training algorithm randomly sample the training sets and fit these samples to decision trees. After training, the classifier can assign an unseen data x by the following method:

$$\hat{y} = Majority(f_1(x), f_2(x), ..., f_n(x)). \quad (5)$$

$\hat{y}$ denotes the classification result, and each f denotes one decision tree.

Similarly, in the experiment, we directly applied the Random Forests Classifier tool provided by [5] on our training data.

D. BERT

BERT (Bidirectional Encoder Representations from Transformers) is a language model which is bidirectionally trained based on the powerful Transformer model architecture that gives us a deeper sense of language context. The transformer is composed of multiple "Multihead Attention Layers". Each layer applies a linear transformation to the input and then uses attention to aggregate information across the sequence.

For text preprocessing before entering the model, some inputs are needed to distinguish two sentences: token embeddings, segment embeddings, and positional embeddings. [CLS] and [SEP] token is inserted at the beginning and the end of a sentence respectively, a segment embedding that indicates two different sentences is added to each token, and a positional embedding specifies the position of a token in the sentence.

BERT is pre-trained on two NLP tasks, and the breakthrough of BERT is done by applying Masked LM (MLM), who randomly masks words in the sentence (15%) and then recover them. Furthermore, Next Sentence Prediction (NSP) is also used to help understanding the relationship between two sentences with the tokens explained previously.

There are four types of BERT architectures with different scales to choose from, BERT-Base-Uncased, BERT-Base-cased, BERT-Large-Uncased, and BERT-Large-cased. In our project, we used BERT-Base-Uncased to train our model.

The detailed descriptions of this model are provided in [7].

E. XLNet

While BERT is already a strong model, but there are two major problems that XLNet tried to improve. In BERT, the [Mask] used in training isn't appearing when fine-tuning and that BERT generates predictions independently.

Under this framework, XLNet keeps the idea of bidirectional context while avoids using [Mask] and parallel predictions by implementing "Permutation Language Modeling" (PLM). PLM predicts tokens in some random order, which can thus learn to model the dependencies between all combinations of inputs.

As for replacing [Mask], XLNet uses Two-Stream Self-Attention to achieve the goal of predicting the position of words and the relationship of the context, where Conten Stream learns the context and Query Stream serves as the role of [Mask] that predicts the representation made by Conten Stream.

To sum up, XLNet is an auto-regressive language model which outputs the joint probability of a sequence of tokens based on the transformer architecture with recurrence.

The detailed descriptions of this model are provided in [8].

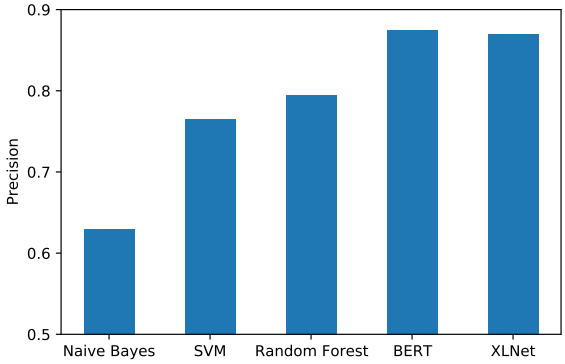

Figure 2. Performance Comparison.

V. Results

The whole experiments were ran on a Linux workstation with Geforce GTX 1080 GPU and 8 GB memory.

Here, we first implemented Naive Bayes, Random Forests, Support Vector Machines, Bert, and XLNet to compare their performance with 10% data. As the results shown in Figure 2, it is clear that Bert, and XLNet works far better than other traditional machine learning methods.

Thus, we then emphasis on examining how to improve BERT and XLNet by testing the result using different numbers of epoch; while did not spend time on fine-tuning the machine learning models by grid searching or trying other approaches.

A. Traditional ML Approaches

From Table I, we can compare the performance of Naive Bayes, Random Forests, and Support Vector Machines. The best performance here is the Support Vector Machines with a accuracy of 0.793, yet this is still incomparable to the deep learning models.

B. Deep Learning Approaches

As one way to improve the models, we tried to test the result using different number of epoch from 1 to 4 to both

Table I
COMPARISON OF ML METHODS TRAINED ON 10% DATA

Approach	Cross Validated Precision
Naive Bayes	0.648
Random Forests ($n = 300$)	0.761
Support Vector Machines	0.793

BERT and XLNet with 10% of data. In Table II, we found that in BERT, it worked best when we use 2 epoch and the accuracy decayed as the epoch increased; where as for XLNet, 3 epoch performed the best.

Table II
COMPARISON OF EPOCH

Approach	Validation Accuracy
1 epoch	
BERT	0.877
XLNet	0.877
2 epochs	
BERT	0.884
XLNet	0.885
3 epochs	
BERT	0.880
XLNet	0.887
4 epochs	
BERT	0.879
XLNet	0.885

After determined the parameters, we started our training with full data. The final result of our models are listed in Table III, which we also recorded the time required for training as a reference for comparing these two models. As a result of validation, BERT and XLNet reached a high accuracy at 0.909 and 0.903 respectively on AICrowd Test F1 Score.

VI. DISCUSSION

From our project, we again proved the strength of the state-of-art deep learning approaches comparing to the traditional machine learning models. This result can be easily imagined and understood due to the fact that both BERT and XLNet are bidirectionally trained which enables us learn language context deeper and better.

However, comparing the two deep learning models, BERT and XLNet, we did not observe a huge difference that XLNet serves as a better model as stated. As we read from paper research, we found that the XLNet models that out beat BERT were XLNet-Large, whereas we implement XLNet-Base here to run our test. Due to the fact that XLNet-Large is a far deeper model than XLNet-Base, it is therefore possible for us to get a similar performance.

Additionally, it is another constraint for us considering the large amount of time needed per one single training for XLNet. Therefore, it is hard for us to test different parameters to fine-tune our model. Considering the issues described above, we can not draw conclusion here whether BERT or XLNet works better for text classification generally. Yet, we could only proposed one best model under this circumstances within limited time.

Table III
COMPARISON OF DL MODELS TRAINED ON FULL DATA

Approach	Training Time	Validation Accuracy	AICrowd Accuracy	AICrowd F1 Score
BERT	19 : 21 : 35	0.904	0.908	0.909
XLNet	44 : 04 : 24	0.904	0.903	0.903

To compare the performance of the two deep learning models, we also take the required training time into account. Since we applied 3 epochs in XLNet, it was far more time consuming, almost 2.3 times longer, than BERT did. Hence, we concluded here that in the model we trained for this Twitter Text Classification task, BERT-Base-Uncased with 2 epochs serves as a better model concerning both desirable accuracy and required training time.

VII. SUMMARY

In this paper, we compare different models to classify twitter text sentiments. By preprocessing hashtags detection, emotions replacement, contraction resolving, and time replacement, we get a refined training data. We focused our training with deep learning models after a first brief testing on 10% of data. Then by testing the performance of different number of epoch, we determining the parameters of 2 and 3 epochs to BERT and XLNet respectively. Running the test with the fine-tuned models, both BERT and XLNet can achieve high precision on online testing data, whereas BERT serves as a better one considering its rather short training time.

REFERENCES

[1] B. Gokulakrishnan, P. Priyanthan, T. Ragavan, N. Prasath, and A. Perera, "Opinion mining and sentiment analysis on a twitter data stream," in *International Conference on Advances in ICT for Emerging Regions (ICTer2012)*, 2012, pp. 182–188.

[2] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

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

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] M. Jaggi and M. E. Khan, *Machine Learning Lecture Slides*. EPFL, 2020.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[8] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," 2020.