Machine Learning Project 2 Report

Mao Yisong, Sun Haoxin, Xu Sixiao
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—This report presents our results in machine learning for the classification of texts based on semantics. In this project, we exploit five machine learning techniques to predict the emotion of texts. To convert text data to a mathematical form, we first remove useless symbols and tokenize words. Then different models are trained respectively and used to make predictions on the test set. In order to avoid overfitting and select hyper-parameters, we implement algorithms like early stopping in the project. From our validation results and testing results, the Transformer-based models have the best performance on prediction accuracy, while SVM and RNN cost less time on training.**

***Keywords*-Machine learning, natural language processing, neural network**

I. Introduction

In recent years, machine learning methods, especially deep learning, have generated many start-of-the-art results in various domains, like natural language processing (NLP), computer vision (CV), and so on. For NLP, it is a theory-motivated computational technique used for analyzing human language automatically [1]. There are many different tasks in NLP, such as text classification, question answering, text representation generation, and this project is a binary text classification task. Recently, various approaches are developed to deal with NLP tasks, from distributed representations, like Word2vec [2], to basic convolutional neural networks and recurrent neural networks, doing works like sentence modeling [3] and language modeling [4], to state-of-the-art bidirectional transformers like BERT [5]. In this project, we use several machine learning methods to predict the emotion of texts, namely support vector machine (SVM), LSTM network, BERT, RoBERTa and XLNet.

This report is organized as the following. Section II displays how we process the raw text data, Section III presents the models and approaches we apply in this project. Section IV describes the tricks we use during the training process to do some optimization. Section V shows the validation and test results of different models. Finally, in Section VI, we conclude our project.

II. Data Analysis and Pre-Processing

We first do data pre-processing including removing useless symbols and tokenizing words, so that the plain texts can be converted into numerical vectors, which can be easier to deal with in machine learning.

A. Remove useless symbols

Since a tweet usually includes lots of punctuation and symbols, many of which may be useless for semantic identification. In SVM, after loading the data, we clean the text by removing meaningless symbols, like HTML markup, non-ascii digits, and so on. By doing this, the model can focus on the words in the texts and not be disturbed by these symbols. Some examples of texts before and after this process are shown in Table I.

Table I
TEXT BEFORE AND AFTER PROCESS

Before	After
<user> is the most adorable piece of sweetness i know * muahhh *	is the most adorable piece of sweetness i know muahhh
ibm network adapter (73p6112 ibm bladecenter js20 fibre channel exp card new 73p6112 fc with 49p2514 bracket assy <url>	ibm network adapter p ibm bladecenter js fibre channel exp card new p fc with p bracket assy
boot rom for 1255tx - & smc 1255tx / lp (smc 1255b - rom item # : 620124 . as a recognized industry leader , smc networ ... <url>	boot rom for tx smc tx lp smc b rom item as a recognized industry leader smc networ

B. Tokenize

Given the texts and words, we have to chop the long sentences into pieces, so the documents become tokens for later processing. There are many types of tokenize methods. We applied a few different tokenizers during our project.

1) tfidf: Term frequency–inverse document frequency counts the frequency of each word in the document, the value increases as the word appear more times in the document while occurring fewer times in all documents. It is easy to compute, though reflecting only a statistical result of single words. Specifically, we used tf.keras.preprocessing.text.Tokenizer [6] in the SVM and LSTM models.

2) Bert Tokenizer: The Tokenizers [7] library provides a pre-trained BERT tokenizer. It is a fast way to start the task. We will introduce it in more detail in the model section.

III. Machine Learning Models

We use three different categories of ML methods to do text classification in this project, which are support vector machine (SVM), traditional RNN, and Transformer-based approach. In each category, we select the most widely used techniques.

A. SVM

Support vector machine is a conventional machine learning method to do binary classification. In general, it formulates and solves an optimization problem. We use the LinearSVC [8] library to implement a Linear Support Vector Classifier in the project.

B. LSTM network

As texts are a kind of series data, it is intuitive to use some recurrent networks with feedback in the architecture. Long short-term memory (LSTM) network is a special recurrent neural network architecture with feedback, which is suitable for our project. By controlling the transmission state through the gated state, the LSTM network is able to remember what needs to be recalled, and forget the unimportant information. In our project, we use the PyTorch LSTM [9] library to implement an LSTM network with 2 LSTM layers, each one with the hidden size set to be 32. Then we have two linear layers to generate the classification results in the network.

To train this LSTM network, we use the Stochastic Gradient Descent (SGD) method, with the batch size set to be 32.

C. BERT

Bidirectional Encoder Representations from Transformers (BERT) is a deep bidirectional transformer, applying fine-tuning based approach for pre-trained language representations. The architecture is similar to Transformer [10], an important breakthrough in sequence transduction models. BERT is pre-trained using two unsupervised tasks, i.e., masked language model and next sentence prediction. The resulting language representation takes into account the content of each word. It achieved high performance on sentence-level and token-level tasks. As general language representations, it can model many downstream tasks as well, which is suitable for our task. In the code, we use the simpletransformers [11] package.

Then for the training, we also use SGD with batch size equals to 32, and this holds for the following RoBERTa and XLNet too.

D. RoBERTa

To further explore and improve performance, RoBERTa, a replication study of BERT, evaluated the effects of hyper-parameter and data when training BERT models. By adjusting the training procedure and controlling the training data, it further improves downstream task performance. Following its exploration of different parameter choices and adopting this model in our task, we can get a better understanding of the transformer models.

E. XLNet

During BERT, some words are masked so that the model has the ability to use the bidirectional information of sentences, however, the relationships between mask words are ignored. XLNet adopts PLM (Permutatoin Language Model), arranging sentences randomly and using the auto-regressive method to train data, which allows the network to obtain dual-direction information and learn the dependencies between tokens. Besides, XLNet uses Transformer-XL to gain broader contextual information. Therefore, we also implement XLNet in this project to do the classification.

IV. Optimization Tricks

Since the training time is quite long for large models, and the hyper-parameters are difficult to select, we deploy several optimization tricks during the training process of Transformer-based techniques.

A. Early stopping

Early stopping is a popular technique in deep learning to avoid overfitting. The key idea is to keep evaluating the model on the validation set during the training process. Once the performance on the validation set stops improving, the training will be terminated so that overfitting can be mitigated. In our project, we use this technique when training BERT, RoBERTa, and XLNet.

To be more specific, we evaluate the Matthews Correlation Coefficient (MCC) during training periodically, then if the values in continuous several evaluations fluctuate in a very small range, we stop the training and return the current model. This process can be described by Algorithm 1.

Algorithm 1 Early Stopping in Training

```
Input: $\delta$, $patience$, $eval\_step\_size$, $current\_model$
1: $counter \leftarrow 0$
2: while training do
3:     $step = step + 1$
4:     $TRAIN(current\_model)$
5:     if $step \,\%\, eval\_step\_size = 0$ then
6:         $eval\_res = EVAL(current\_model)$
7:         if $|eval\_res - last\_eval\_res| < \delta$ then
8:             $no\_impr\_counter = no\_impr\_counter + 1$
9:             if $no\_impr\_counter > patience$ then
10:                break
11:            end if
12:        else
13:            $no\_impr\_counter = 0$
14:        end if
15:        $last\_eval\_res = eval\_res$
16:    end if
17: end while
Output: $current\_model$
```

B. *Automatically select hyper-parameters*

The performance of deep learning models is usually sensitive to hyper-parameters. For example, the learning rate, batch size, and many other parameters can play a key role in performance. For Transformer-based techniques, this fact still holds [12]. Thus, in the project, we automatically select and tune these hyper-parameters, rather than setting them manually. A widely used way to select hyper-parameters, usually called ”sweep”, is to calculate the loss on the evaluation set after training with different hyper-parameters, and then choose the ones achieving minimum loss. However, using sweep means we need to train the same models several times with different hyper-parameters to compare the performance, which can be time-consuming. Thus in our project, we first implement this method on a small part of the training data, selecting the best hyper-parameters, and then use these chosen hyper-parameters to train models on the entire training set. And in this process, we implement the Bayesian hyper-parameter search method to do the sweep, which models the relationship between the parameters and the evaluation loss based on a Gaussian Process [13].

V. Results in Evaluation and Test

In this section, we display the performance of different methods. Specifically, these results are measured based on a single NVIDIA GeForce RTX 3090 GPU.

Table II shows the prediction results on the evaluation set and test set of the above methods. The hyper-parameters we use here is determined by doing sweep, which will be described later. It can be seen that the performance of methods in different categories varies a lot. Transformer-based techniques have higher prediction accuracy on both evaluation set and testing set compared with the traditional SVM and LSTM network, while RoBERTa has the best performance on both the evaluation set and the test set.

Table III shows the training time of these models, which also includes time consumed on data pre-processing. It can be seen from the table that SVM takes the least time since it just solves an optimization problem and doesn’t need to do iterations like Stochastic Gradient Descent, which are used in the training of the other four models. While the LSTM network and the three Transformer-based ways take a much longer time to do the training, especially for XLNet, its training takes more than 12 times as long as that of SVM. Thus, there is a trade-off between prediction accuracy and training time in applications.

Table II
Prediction performance of different methods

Model	SVM	LSTM	BERT	RoBERTa	XLNet
Eval. Acc.	0.8504	0.8137	0.9047	0.9085	0.9077
Eval. F1.	0.8504	0.8237	0.9049	0.9080	0.9067
Test Acc.	0.845	0.805	0.899	0.907	0.904
Test F1.	0.846	0.816	0.899	0.907	0.904

Table III
Time consuming of different methods

Model	Train Time	Eval. Time
SVM	0:35:15	0:0:25
LSTM	2:44:06	0:00:34
BERT	5:36:28	0:01:52
RoBERTa	5:45:35	0:01:53
XLNet	7:33:16	0:04:34

Then we display the effectiveness of early stopping and sweep on hyper-parameters. We use the tool Weights & Biases [14] to sweep different hyper-parameters and to record how the training loss and evaluation loss change during the training of BERT. We sweep the number of epochs from 2 to 4, and the learning rate in range $\left[5 \times 10^{-6}, 1 \times 10^{-4}\right]$.

Fig. 1 shows how the training loss varies with training steps in different learning rates and the number of epochs. It can be seen that when the hyper-parameter varies, the training loss will be different too, and sometimes even generates outlier results, whose F1 score jumps severely during the training. Fig. 2 draws how the evaluation loss changes in different hyper-parameter settings, which omits the outlier. Table IV shows the importance of hyper-parameters and correlation with the evaluation loss, indicating that the number of epoch plays a really important role in the performance. Finally, Fig.3 displays the evaluation loss under different hyper-parameters, and it can be seen that with $l.r. = 1.96 \times 10^{-5}$, $\#\ epoch = 2$, we can have the minimum evaluation loss in this project, and this couple is actually the parameters we use to gain results of BERT, RoBERTa and XLNet in Table II and Table III.

From Fig. 1 and Fig. 2, we can also observe the necessity of using early stopping, by the fact that after some steps, the evaluation loss will stop decreasing and becomes higher again. Without early stopping, we may miss the point with the smallest evaluation loss and get less prediction accuracy. Thus, using early stopping will not only save training time but also improve models’ performance.

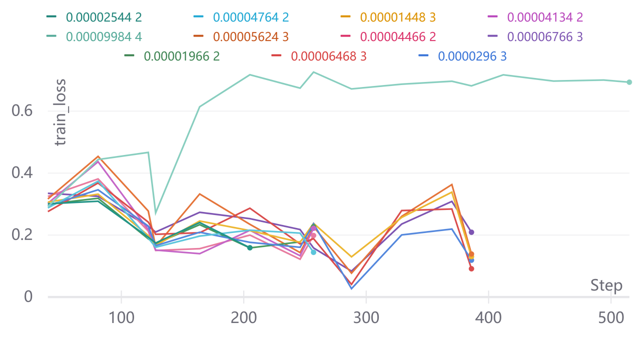

Figure 1. Training loss using various hyper-parameters.

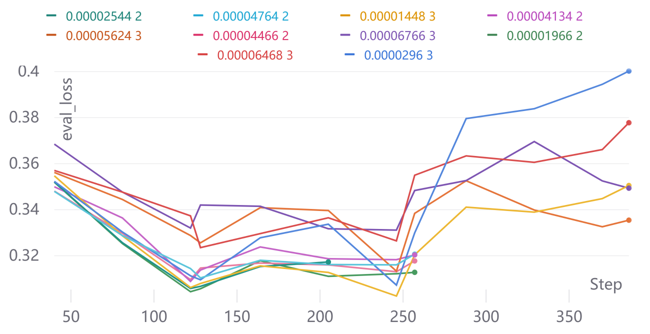

Figure 2. Evaluation loss using various hyper-parameters.

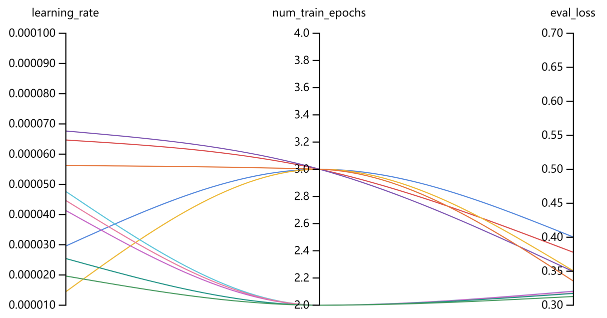

Figure 3. Evaluation loss vs l.r. and # epoch.

VI. Conclusion

In this report, we demonstrate our results on text classification, a typical NLP application in ML. We first show how to do data pre-processing to transform texts into numerical data. Then we introduce five different models we use in the project to do the classification, as well as some tricks we implement during the training process. Finally, the testing results show that Transformer-based models, which have more than 100 million parameters, have the best performance on prediction, while SVM costs the least time on training. Besides, apart from the models themselves, techniques used during training like early stopping and hyper-parameter sweep can have an effect on the result too.

References

[1] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[4] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[6] "tf.keras.preprocessing.text.tokenizer : Tensorflow core v2.7.0." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

[7] "Tokenizers." [Online]. Available: https://huggingface.co/docs/tokenizers/python/latest/

[8] "sklearn.svm.linearsvc." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[9] "Lstm." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[10] 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

[11] "simpletransformers." [Online]. Available: https://pypi.org/project/simpletransformers/

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[14] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

Table IV
Effect of hyper-parameter selection on evaluation loss

Hyper-Parameter	Importance	Correlation
# epoch	0.657	0.805
l.r.	0.353	0.169