Classification Competition: Twitter Sarcasm Detection
CS 410 Final Project Documentation

Yang Yang
yangy19@illinois.edu

Abstract

Sarcasm detection is a specific case of sentiment analysis where instead of detecting a sentiment in the whole spectrum, the focus is on sarcasm. In this Classification Competition, the task is to detect sarcasm in contextual Twitter text. In order to beat the baseline F1 score and improve the performance, the main model used in this project is one of the State-of-the-Art NLP models, BERT. I adapt the off-the-shelf BERT classifier model by Huggingface, modify and expand the use of fine-tuning for other BERT-based models. Further more, I investigate the BERT model performance when context information is used in different manners. The result interestingly shows that doing this specific task as a sentence pair classification outperforms it as a normal text classification.

1. Introduction

With the growing role of social media across the world, Sarcasm in tweets has raised more attention. Thus, how to use NLP models to efficiently detect Sarcasm in tweets also has been a hot topic in both academia and industry.

In this project, I first use a couple of BERT-based pre-trained models, such as BERT, ALBERT, DistilBERT and SqueezeBERT, to understand language and beat the baseline performance. Then I start to look at the methods of utilizing Context information in tweets. Sarcasm Detection, from the topic name itself, sounds like a very typical binary text classification. Since we have Context information together with Response, I use Context sentence(s) in three different methods to run the hyperparameter tuning under BERT pre-trained model.

2. Approach

Overall, I follow the procedures below to fine-tune and improve the model performance in this project:

(1) Adapt BERT classifier from Google Research and BERT example by Huggingface transformers [6];

(2) Modify BERT model code to make it applicable for other BERT-based models;

(3) Load the train and test dataset and split original train dataset in $80:20$ for train and validation;

(4) Run Compare model performances of BERT, ALBERT, DistilBERT, SqueezeBERT and XLNet with same hyperparameter setting;

(5) Fine-tune BERT hyperparameters when using three different methods for Context information;

(6) Compare performances with different Context methods.

2.1. BERT Classifier Adaption and Expansion

Inspired by Rajapakse [3] using one of the State-of-the-Art NLP model, BERT, I first look into Google's original BERT paper [7], and then notice the off-the-shelf BERT classifier from Google Research GitHub [1]. While some TensorFlow `bert` packages have revision issues and haven't been solved for a while. So I switch some functions to similar ones in PyTorch to fix incompatible issues in the code, luckily because NLP researchers from Huggingface have developed a PyTorch version of BERT.

The next step is to expand the BERT classifier for other BERT-based model fine-tuning for model performance comparison. Thanks to PyTorch `AutoClasses` model, it can automatically recognize the architecture from pre-trained model id to extract tokenizer and configuration file accordingly. In order to make the training portion compatible, I dig into other four models' `SequenceClassification` functions on Huggingface Transformers Documentation website [2]. It turns out that not all `SequenceClassification` functions are able to accept same inputs:

The `SequenceClassification` functions for BERT, ALBERT and SqueezeBERT all take `input_ids`, `attention_mask` and `token_type_ids` tokens. While the `SequenceClassification` functions for DistilBERT and XLNet both can only read `input_ids` and `attention_mask` tokens, no `token_type_ids` tokens. So I need to differentiate these two situations and supply different inputs.

2.2. BERT-based Models

After adaption and modification, the code can run to fine-tune other pre-trained models like BERT, ALBERT, SqueezeBERT, DistilBERT and XLNet. To begin with a basic comparison, I go through the pre-trained models hub and list hosted by huggingface [4]. Based on the model description and community results, I select following models from these five architectures:

1. BERT: A transformers model pre-trained on a large corpus of English data in a self-supervised fashion.
    - `bert-base-uncased` is the most popular BERT model and trained on lower cased English text.
2. ALBERT: A lite BERT for self-supervised learning of language representations. It uses repeating layers which results in a small memory footprint, however the computational cost remains similar to a BERT-like architecture with the same number of hidden layers as it has to iterate through the same number of (repeating) layers.
    - `albert-base-v2` is trained on ALBERT base model with no dropout, additional training data and longer training.
3. DistilBERT: A transformers model, smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher.
    - `distilbert-base-uncased` is distilled from `bert-base-uncased` checkpoint.
4. SqueezeBERT: A bidirectional transformer similar to the BERT model. The key difference between the BERT architecture and the SqueezeBERT architecture is that SqueezeBERT uses grouped convolutions instead of fully-connected layers for the Q, K, V and FFN layers.
    - `squeezebert-mnli-headless` is the `squeezebert-uncased` model finetuned on MNLI sentence pair classification task with distillation from `electra-base`. This pre-trained model is specifically recommended on Huggingface SqueezeBERT site [5] for best results when fine-tuning on sequence classification tasks.
5. XLNet: An extension of the Transformer-XL model pre-trained using an autoregressive method to learn bidirectional contexts by maximizing the expected likelihood over all permutations of the input sequence factorization order.
    - `xlnet-base-cased` is the XLNet base English model.

Architecture	Model ID	Model Size			
		Hidden Layer (L)	Hidden Size (H)	Attention Heads (A)	Parameters
BERT	`bert-base-uncased`	12	768	12	110M
ALBERT	`albert-base-v2`	12 (repeating)	768	12	11M
DistilBERT	`distilbert-base-uncased`	6	768	12	66M
SqueezeBERT	`squeezebert-mnli-headless`	12	768	12	51M
XLNet	`xlnet-base-cased`	12	768	12	110M

Table 1: Model Size Summary of Selected Pre-trained Models

These five pre-trained models are selected from each of the architecture because they are either the most efficient or best for the task among all in the architecture. Table 1 shows a model size summary of the pre-trained models I use in the project.

2.3. Methods to Use Context Info

When adapting BERT classifier, I notice that the `InputExample` class has two different string attributes `text_a` and `text_b` for sequence text. Apparently, Response information always goes into `text_a`. So I can implement Context information in three different methods:

- *Method 1 - Use No Context Info:* Only use Response information in `text_a` and ignore Context information.
- *Method 2 - Concatenate Context with Response (used in 2.2):* Concatenate Context string after Response string in `text_a` and run it as a normal text classification task.
- *Method 3 - Use Context as Separate Sentence Info:* Use Response information in `text_a` and Context information in `text_b`. Now when running `SequenceClassification`, it is actually a sequence pair classification.

2.4. BERT Model Hyperparameter Fine-tuning

For hyperparameter fine-tuning, I sweep the batch size in $\{8, 16, 32\}$, learning rate in $\{2{\cdot}10^{-5}, 5{\cdot}10^{-5}, 1{\cdot}10^{-4}\}$, gradient accumulation steps in $\{1, 2, 3\}$, and number of epochs in $\{1, 2, 3\}$ for all three methods.

3. Experiments and Results

3.1. Data

In this project, Instructors provide us the dataset from Twitter. The train dataset has 5000 sarcastic or non-sarcastic posts labeled with `SARCASM` or `NOT_SARCASM`. There are in total of 2010 `NOT_SARCASM` posts and 1990 `SARCASM` in train dataset, which constructs a balanced dataset for binary classification. The test dataset has 1800 posts without labels.

Both datasets include Response and Context information. Response is the Tweet to be classified and the Context is the conversation context of the Response, which I place in three methods.

3.2. Evaluation Metrics

For evaluation, I use F1 score as my primary metric since it is the criteria to beat the baseline. Since F1 is calculated based on Precision and Recall, I generate all three metrics in my code.

3.3. Performances from Different Pre-trained Models

To compare the five selected models, I sweep the number of epochs in $\{1, 2, 3, 4, 5\}$ for all models with same remaining hyperparameters as shown below:

- Maximum Sequence Length = 128
- Warmup Proportion = 0.1
- Batch Size = 16
- Learning Rate = $2 \cdot 10^{-5}$
- Gradient Accumulation Steps = 1
- Context Method: Concatenate after Response

Figure 1 shows the learning curves on validation dataset for the selected models from five architectures. In F1 plot, BERT shows a high and stable F1 score, even though the highest F1 score is not from BERT. Although the highest F1 score happens with XLNet pre-trained model, it only happens when epochs reaches at 5 with large fluctuation.

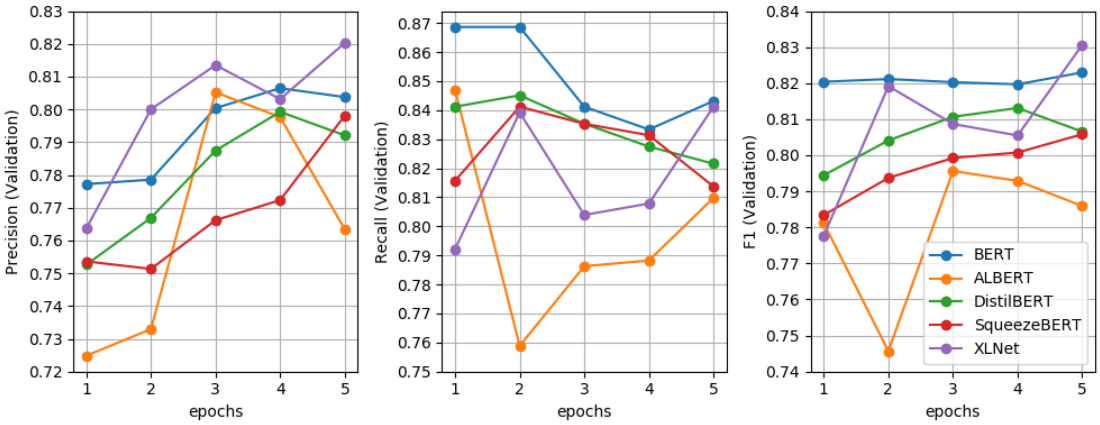

Figure 1: Learning Curves of Selected Models. Overall BERT model shows the best performance. Although the highest $F1$ score happens with XLNet pre-trained model, it only happens when epochs reaches at 5 with large fluctuation. Considering recall and the impact of random seeds on fine-tuning, the BERT model is better.

Since it is a sarcasm detection task, the goal is to detect more positive cases. So the cost of missing a positive case is more problematic than the cost of including a negative case. That means Recall is more important Precision. Looking at the Recall plot, it is clear that BERT model is the best one.

In Dodge's paper [8], he mentioned that even with the same hyperparameter values, distinct random seeds can lead to substantially different results due to weight initialization, training data order and other reasons. It tells us that users may not be able to generate such high F1 score with XLNet model every single time by looking at its unstable learning curve.

Considering both Recall learning curve and the impact of random seeds on fine-tuning, I choose the BERT model `bert-base-uncased` as the best model among all.

3.4. Performances from Different Context Methods

As mentioned in 2.3, I use three different methods for Context information, they are:

- `Response_only` denotes to Method 1: Use No Context Info
- `ResponseContext_Connect` denotes to Method 2: Concatenate Context with Response
- `ResponseContext_Separate` denotes to Method 3: Use Context as Separate Sentence Info

Figure 2 shows the performance metrics of three Context methods. In F1 score histogram, more than 80% of Method 3 iterations reach 0.8 of F1, which is the best method in terms of F1 score. While surprisingly Method 1, which does not use Context info is even better than Method 2 - Concatenate Context with Response.

Looking at the Precision vs. Recall plot, most of Method 3 dots have both high precision and recall, which result in high F1 score, whereas dots of the other two methods have either low precision or low recall.

Overall, the BERT model using Method 3 - Use Context as Separate Sentence Info, has the best performance on average.

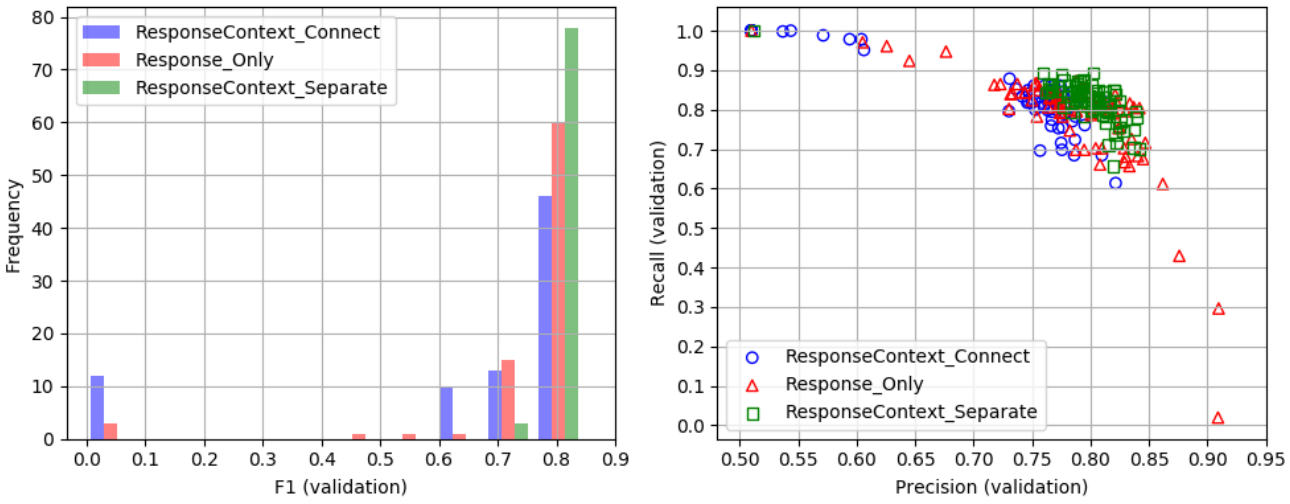

Figure 2: Performance Metrics for Three Context Methods.

3.5. BERT Model Hyperparameter Fine-tuning with Method 3

After selecting pre-trained model and Context method, I sweep hyperparameters of batch size, learning rate, number of epochs and gradient accumulation steps. Figure 3 shows the hyperparameter optimization sweep result.

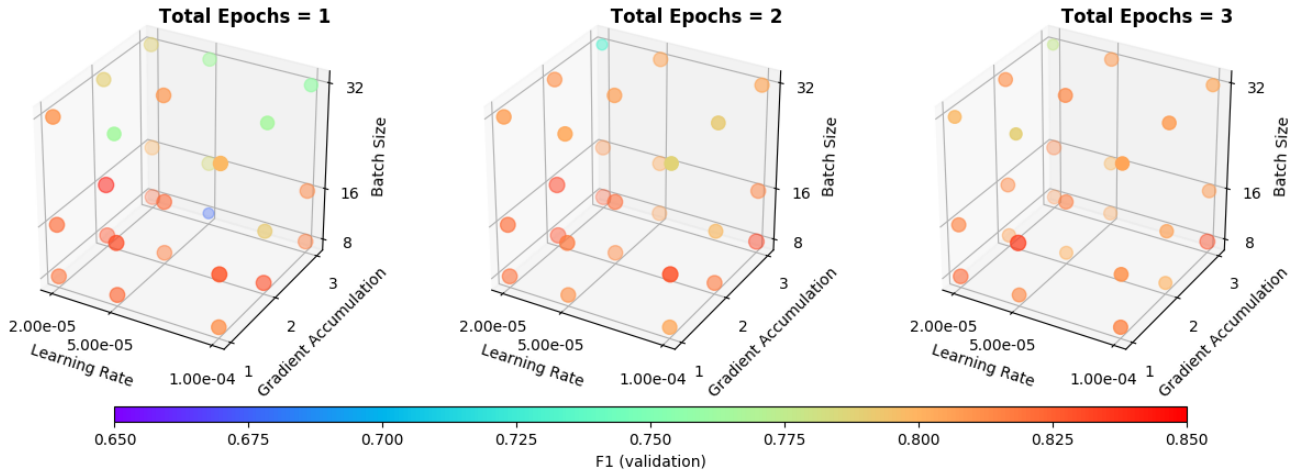

Figure 3: Hyperparameter Fine-tuning for BERT Model with Context Method 3. More red(ish) dots indicate higher $F1$ scores, whereas more blue(ish) dots indicate lower $F1$ scores.

	Precision	Recall	F1
Validation	0.783	0.892	0.834
Test	0.708	0.827	0.763

Table 2: Final Result

Based on the optimization result, I select the following hyperparameters as my final BERT model hyperparameters:

- Maximum Sequence Length = 128
- Warmup Proportion = 0.1
- Batch Size = 16
- Learning Rate = $2 \cdot 10^{-5}$
- Gradient Accumulation Steps = 2
- Number of Epochs = 2

By using this hyperparameter set, my model is able to reach F1 score at $\mathbf{0.763}$ for test dataset. Details are shown in Table 2.

4. Conclusion

In this Classification Competition, I use pre-trained base models of BERT, ALBERT, DistilBERT, SqueezeBERT and XLNet for contextual Twitter sarcasm Detection. Among those base models with same hyperparameters, BERT shows the best performance. In the further discussion of Context information use, I place Context sentences in three different methods. After hyperparameter tuning with all three, the BERT model using Context as separate sentence information shows the best performance on average.

The best result from the best BERT model reaches on the test dataset $\mathbf{0.763}$ as F1 score and beat the baseline F1 score of 0.723.

5. GitHub Repo

The whole project is developed with PyTorch framework in Google Colab environment. The source code, voiced presentation and this project documentation are all available in this CourseProject GitHub Repo. (https://github.com/yangyangsquare/CourseProject).

References

[1] google-research/bert/run_classifier.py. https://github.com/google-research/bert/blob/master/run_classifier.py. Accessed: 2020-12-10. 1

[2] Huggingface transformers documentations. https://huggingface.co/transformers/index.html. Accessed: 2020-12-10. 2

[3] A simple guide on using bert for binary text classification. https://medium.com/swlh/a-simple-guide-on-using-bert-for-text-classification-bbf041ac8d04. Accessed: 2020-12-10. 1

[4] Transformers pre-trained models. https://huggingface.co/transformers/pretrained_models.html. Accessed: 2020-12-10. 2

[5] Transformers squeezebert model. https://huggingface.co/transformers/model_doc/squeezebert.html. Accessed: 2020-12-10. 2

[6] transformers/examples/movement-pruning. https://github.com/huggingface/transformers/blob/67ff1c314a61a2d5949b3bb48fa3ec7e9b697d7e/examples/movement-pruning/masked_run_glue.py. Accessed: 2020-12-10. 1

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[8] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping, 2020. 4