Documentation: BERT fine-tuning for Twitter sarcasm detection

Name: Junzhe Sun **NetID**: junzhes2 **Team**: Junzhe Sun (individual)
Competition: text classification competition

Overview

This project aims to detect sarcasm from Twitter posts. This is a text classification task related to sentiment analysis. The training data set includes text-label pairs, therefore constituting a supervised machine learning problem. I experimented with two different methods, including FastText and BERT. I found BERT with fine-tuning to have the better performance, which achieved a higher f1 score than the baseline.

The code is implemented in Python under the Google Colab environment. Two Jupyter notebooks are available, one using fastText and the other using BERT in PyTorch. Both notebooks are self-contained, in that data loading, data cleaning/processing, training and prediction are available in each notebook. The input to both notebooks are a training data set with text-label pairs, and a test data set without labels for prediction. The notebook outputs predicted labels after training the respective neural network using the input data. They can be used for general text classification tasks beyond the binary classification task in this project.

Implementation details

Model

Two models were tested for the text classification competition. I first tried fastText, a shallow-learning library that can be used for both word embedding and text classification. FastText is a close sibling to Word2Vec, and was introduced by Bojanowski et al. (2017) from Facebook AI Research. FastText is a method that aims to extend word vectors to capture subword information using character n-grams, instead of representing each word in the vocabulary as distinct vectors. This concept can be extended to the bag of word (BoW) representation of texts using latent vectors of words and word n-grams, which leads to a simple yet efficient baseline method for text classification (Joulin et al., 2016). The second model is a transformer network known as BERT, which stands for Bidirectional Encoder Representations from Transformers and is introduced by Devlin et al. (2018) from Google AI Language. The transformers is a type of deep neural network that's based on attention mechanism (Vaswani et al., 2017), and has recently gained a lot of popularity in the NLP community. The BERT model is designed to pre-train deep bidirectional representations of language models using unlabeled text. Bidirectional training means that representations are trained by jointly conditioning on both left and right context in all layers of the deep network. Consequently, a pre-trained BERT model, available from a variety of sources, can be very conveniently obtained and fine-tuned with just one additional output layer for a wide range of tasks, such as text classification. The Transformers library from Hugging Face provides a wide selection of pre-trained, general-purpose architectures for NLP tasks, including BERT. In particular, they provide the

BertForSequenceClassificaiton model which attaches a pooling layer and a linear layer after the BERT network. It can be used for the Twitter sarcasm detection task.

Training

Data loading and data cleaning are performed using Pandas. Consecutive spaces were replaced by a single whitespace, and leading and trailing spaces were removed. Some special characters, such as “@USER”, were removed, while Emoji’s were kept as they capture emotions. Emoji needs to be converted to known tokens for the network, and the emoji library provides a convenient function to do so. For example, `emoji.demojize('🤔')` returns `':thinking_face:'`. Cases were also kept because in my tests they tend to improve the f1 score (for both models). Punctuations are treated differently depending on the model. They were removed in the case of fastText, while for BERT, they were kept since BERT has embedding for most of the punctuations. The data cleaning steps are the same for both the training and test data sets.

The training data (after data cleaning) is first shuffled randomly in order to mix SARCASM and NOT_SARCASM entries. Then a total of 5000 examples in the training data are split into a training set of 4500 examples and a validation set of 500 examples (10% split). The validation set is used for periodically evaluating the current model for tuning the model’s hyperparameters and preventing overfitting. A minor challenge is that, since the training data set is not very big, splitting it into training and validation sets can be tricky. My solution is to first use the validation to find the optimal hyper parameters, and then retrain the model using the entire training set. For fastText, the most important hyperparameters include learning rate, number of epochs and wordNgrams. For BERT, some of the hyperparameters include learning rate, number of epochs, drop out rate and maximum length of text that can be handled by the tokenizer.

Result and Deliverables

For fastText using the tri-gram model, it is unable to beat the baseline, having an f1 score of 0.6642. BERT with fine tuning is able to reach an f1 score of 0.7362 within a few rounds to hyperparameter tuning. All model parameters allowed to be updated during the training process. I found four epochs is enough for the fine turning, while running more epochs tends to overfit the training data set and leads to a lower f1 score on the test data set.

The code/deliverables include two Jupyter notebooks. More detailed comments are available in the notebooks about the specific functions of each cell.

- **01_fasttext4Twitter.ipynb**: This notebook contains implementation using the fastText library. The data loading and data cleaning steps use Pandas. The preprocessed training data set is written to disk in a format that’s compatible with fastText. Training and prediction follows the steps of the text classification example in fastText’s documentation ([5]), and only involves a few lines of code. To obtain additional metrics, the scikit-learn library is used to compute precision, recall and f1 score. Finally, both the trained model and the predicted labels are written to disk.
- **02_bert4Twitter_pytorch.ipynb**: This notebook contains implementation using PyTorch and the transformers library from Hugging Face. Pre-trained

BertForSequenceClassification and BertTokenizer is downloaded from Hugging Face, and additional fine tuning of Bert network weights is performed using the training data. Similar to fastText, data is preprocessed using Pandas. To convert from Pandas data frame to PyTorch compatible data format, the datasets library from Hugging Face is used. To obtain additional metrics, the scikit-learn library is used to compute precision, recall and f1 score. Hugging Face also provides a simple but feature-complete Trainer class for training/fine-tuning the network ([6]). Trained model is then set to evaluation mode, and test data is transferred to GPU for prediction. Finally, both the trained model and the predicted labels are written to disk.

Usage of software

This project is implemented using Google Colab. Therefore, the most convenient way to reproduce and verify the results is through Google Colab and using a GPU runtime environment. Google Colab provides a free Nvidia Tesla T4 GPU for 12 hours for continuous usage for free. Open the Jupyter notebooks in Google Colab, map the correct Google Drive that contains the data and trained model, then modify the "filepath" variable to reflect the correct path. The notebook can be executed block by block, or select run all under Runtime.

For 01_fasttext4Twitter.ipynb, since the training is so efficient, there is no need to load the trained model. On the other hand, for 02_bert4Twitter_pytorch.ipynb, the deep neural network can take a while to train. If one would like to avoid re-training the model, skip the training block and execute the Demo part instead (but don't skip the data loading blocks).

Team Contribution

This is a one-person team.

Tutorial

A software tutorial presentation is uploaded to a Box drive accessible with the link:
https://uofi.box.com/s/wedyjqpel0rw2uxprax10mh2rnwxhu7t

Note that although the tutorial is 15 mins long, the review can feel free to watch only the first 10 mins, which is about the BERT approach. The last 5 mins talks about fastText, which is optional to watch. Finally, to avoid re-training the BERT model, a trained model is available with the link:
https://uofi.box.com/s/t64lnt83ck2m3khtrk4s3r46fjhd5gsk

References

[1] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135-146.
[2] 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.
[3] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30, 5998-6008.
[5] https://fasttext.cc/docs/en/supervised-tutorial.html
[6] https://huggingface.co/transformers/training.html
[7] https://medium.com/atheros/text-classification-with-transformers-in-tensorflow-2-bert-2f4f16eff5ad
[8] https://curiousily.com/posts/sentiment-analysis-with-bert-and-hugging-face-using-pytorch-and-python/