TU Wien

Data Science

Applied Deep Learning

- Sentiment Analysis Exercise 2 -

Authors
Johannes Bogensperger -
01427678

December 19, 2019

1 Introduction

Due to the limitations of BERT when it come to classifying long texts, I needed to change my dataset. It turned out that classical methods such as a Bag-of-words approach in combination with an SVM outperform deep learning methods. In addition to the high resource efforts needed to apply BERT on long texts and high volumes of data.

1.1 Data Source

The substition dataset is a list 1.6 million tweets which are classified either as positive or negative. [1] The dataset is perfectly balanced with 800k positive and 800k negative tweets. All additional data apart from the target (0 for negative and 1 for positive sentiment) and the text of the tweet will be discarded at the preprocessing.

2 Target

Since the dataset is balanced and the benchmarks in the literature are provided with the accuracy metric I decided to use **accuracy** as well.

The current results with classical approaches with SVM deliver results from 70 to 75% accuracy. [4] [5] It should be mentioned, that those approaches are not trained on the full 1.6million tweets.

The most successful Deep Learning approaches with LSTMs on Kaggle reach up to 80% accuracy. The only academic implementation of DL on this dataset uses, since its compared against SVMs, also only a subset of the data and reaches up to 75%.[4]

There are classical approaches which reach a higher percentage (up to 83%), but they use a dataset which still includes the emoticons. Learning upon emoticon features is not desired, since we are interested in natural language processing and emoticons tend to have a clear positive or negative semantic. [3]

Since the most popular kernels on Kaggle often represent the state of the art, we aim for reaching **79% accuracy**, as the most popular current Kernel for this problem. [2]

2.1 Results

2.1.1 BERT with Kashgari Framework

The first architecture used are BERT word-embedding with various LSTM/BiLSTM/BiGRU architectures on top. Those models deliver an slightly better performance than the most popular Kaggle Kernel with **81,05% accuracy**.

We use various BERT Models:

- BERT Small uncased (12 Layers)
- BERT Large uncased (24 Layers)
- Multilingual Small (12 Layers)

It is interesting that the accuracy neither decreased upon using multilingual BERT or increased while using BERT Large. Furthermore, there were no increases in Accuracy when the amount of layers, the number of LSTMs per Layer or parameters like the Dropout rates etc. were increased. That is why we ended up with a simple 2 BiLSTM layers architecture where each layer had 64 BiLSTM (128 LSTMs).

2.1.2 LSTM with Word2Vec Embeddings

Furthermore I implemented a model with an Word2Vec embedding and an LSTM model which turned out to deliver **80,98% accuracy**. This is equal to the most popular Kaggle Kernel. I should be mentioned that this model needs way less training time and needs less parameters (depending embedding size etc). It is possible to sacrifice a little accuracy and drive down the number of parameters significantly.

2.1.3 BERT with Pytorch Framework

The last implementation uses BERT embeddings as well. The big advantage here is, that using the Pytorch implementation (Instead of Kashgari) of BERT, we can finetune the model with masking certain words in our corpus. On top of BERT we use a single linear layer for the classification task.

This process is highly resource intensive. Therefore I was only execute a 2 runs until the project deadline (one epoch for the finetuning with 1.440.000tweets takes about 8h). After a single epoch I reached an **accuracy** of **86,23%**.

3 Time Schedule

Description	Planned	Needed
Understanding of the Basics of BERT	8	12
Meetings with AAF to gather all necessary data and understand the domain	4	8
Parse and preprocess XML files	8	2
Investigate and set-up optimal model architecture and run initial training runs	10	50
Fine tune Model, consider possible improvement possibilites and evaluate performance	8	30
Deploy model and integrate it in a basic webfrontend	16	
Documenting the final solution + Readme	2	
Craft final report and presentation	10	
Sum	66	102

4 Usage of the application

4.1 Data Models

The data needs to be downloaded from the source below and placed in the "data" folder of the project with the filename "training.1600000.processed.noemoticon.csv". If another name/path for the inputfile is desired, it can be altered using "preprocessing.py" line 20 - filepath=enter_your_path.

The data can be downloaded from:
http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip

4.2 Training the models

The various models can be trained using the "main.py" file after downloading the with a proper parametrization. To reproduce the results of the Pytorch-BERT model, just run "main.py" without additional parameters. It will by default preprocess the data and train this kind of model with a single epoch.

For the basic functionality the following parameters are available:

- –doPreProcessing: If preprocessing should be done use "True". This is necessary after the initial run for "BERT/pytorchBert" and "W2V"
- –embedding: Choosing Either "BERT", "pytorchBert" or "W2V" for the embedding. The other parameters will work fine with default values and reproduce the mentioned results above.

The other parameters are relevant if you are planning to fine-tune certain models. Nevertheless it is necessary to checkout which parameters apply where. E.g. action/existingModelPath are only relevant for re-training existing Kashgari BERT Models.

Sidenote regarding the tests: Preprocessing should be executed for W2V and BERT so that all tests will pass. To do so without training any model/both models call doPreprocessing(True) and do PreProcessing(False) from the "preprocessing.py" file. The boolean indicates whether its for BERT or W2V.

References

[1] Dataset Sentiment140 twitter sentiment classification. http://help.sentiment140.com/for-students. Accessed: 2019-12-10.

[2] Most Popular Kaggle Kernel for twitter sentiment classification. https://www.kaggle.com/paoloripamonti/twitter-sentiment-analysis. Accessed: 2019-12-18.

[3] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1–6, 2009.

[4] A. K. Uysal and Y. L. Murphey. Sentiment classification: Feature selection based approaches versus deep learning. In *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pages 23–30, Aug 2017.

[5] Yili Wang, Le Sun, Jin Wang, Yuhui Zheng, and Hee Yong Youn. A novel feature-based text classification improving the accuracy of twitter sentiment analysis. In James J. Park, Vincenzo Loia, Gangman Yi, and Yunsick Sung, editors, *Advances in Computer Science and Ubiquitous Computing*, pages 440–445, Singapore, 2018. Springer Singapore.