TU Wien

Data Science

Applied Deep Learning

TwitterSherlock3000 - Project Report

Authors
Johannes Bogensperger -
01427678

January 22, 2020

1 Introduction

Due to the limitations of BERT when it come to classifying long texts, I needed to change my dataset. The original use-case was a multilabel text classification for texts with about 3k-10k words. 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 amount of computing resources needed to apply BERT on long texts and high volumes of data.

Therefore I came up with a new idea to classify the sentiment of tweets for opinion/sentiment mining. The heart of the project is a BERT Model for Sequence classification which categorizes a tweet if it has a positive or negative sentiment. In combination with a small web-application with access to the Twitter API, we can derive sentiment statistics about arbitrary topics.

2 Problem Domain

A vast amount of organizations are interested in how their products, actions or general reputation are/is seen by customers or the general public. Due to that, nearly every service provider in the field of machine learning provides various approaches for text mining, which usually include possibilities for sentiment analysis. The goal of such a tool is to derive insights about the opinion of people upon a certain topic.

Unfortunately, classifying sentiment precisely is still a problem in natural language processing due to the need for understanding and interpreting the surrounding context of each word. Therefore, traditional bag-of-words approaches with classical ML models are not applicable to solve such a problem. Potentially, every word in a sentence can alter the context or even flip the meaning of other words.

In opposition to older approaches, we are going to discard all emoticons and smileys from the tweets, since they usually contain a clearly defined positive or negative sentiment. We strive to categorize sentiment only based on textual interpretation.

2.1 Data Source

The dataset for training our model is a list of 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 features, apart from the target (0 for negative and 1 for positive sentiment) and the text of the tweet, will be discarded at the preprocessing.

Furthermore, the following items are removed during the data cleansing process from the text feature:

- Emoticons and smileys
- Redundant punctation (e.g. "..." to ".")
- Hyperlinks
- special characters
- numbers

3 Possible Target

Since the dataset is balanced and the benchmarks in the literature are provided with the accuracy metric as quality 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]

4 Evaluated models

We evaluated various implementations until we were satisfied with the achieved results. While doing so, we mainly focused on approaches which were able to interpret the context and extract the semantics of a sentence. Therefore, we chose the Word2Vec and BERT Model from Google Research.

Both approaches are based on deep neural networks. BERT (short for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers), which proves its abilities to predict the context information in its outstanding "Question-Answering" and "Next-Word-Prediction" results. BERT is based on the Transformers architecture using Encoder and Decoders with Selfattention layers whilst integrating the left- and right context jointly. This means the text left and right from a token (word) is included in the estimation of its semantic / mapping into a high dimenisonal space for word embeddings.

Word2Vec on the other hands uses a far simpler approach of skip-gram and continuous bag-of-words approach, which results in a worse performance. Therefore Word2Vec cannot take the context of a token into consideration for estimating semantic/word embedding. Resulting that the bank (financial institution) and bank (object to sit on) will cannot be distinguished. Therefore our desired solution will have to work with a Transformer architecture. Traditional machine learning approaches cannot compete with deep learning in this problem domain.

4.0.1 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**.

Since this is based on out of the Box BERT model and not on not fine-tuned BERT model, this is quite impressive. This highlights the generalization capabilities of the BERT models, to be able to interpret sentiment with any adaption to the specific domain language.

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 chose a simple 2 BiLSTM layers architecture where each layer had 64 BiLSTM (128 LSTMs).

4.0.2 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.

4.0.3 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 fine-

tune 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%**.

5 Deployment of the model - Final Application

To apply this model on a real world problem, we built a lightweight flask web-app. This app will analyze a mixture of recent and popular(not necessarily new) tweets which contain an arbitrary phrase provided by the user.

During this analysis the application with define the sentiment of the tweets and extract all used hashtag. The final product for the user will be a statistic of all contained hashtags and which of them were used in how many positive and negative tweets and how many accumulated likes the positive and negative tweets got.

Users could use this tool to analyze the sentiment about arbitrary topics in the twitter community. Therefore we assume it would enable users to research and derive insights about twitter sentiments with this application.

Unfortunately, without paying for the premium functionality of the Twitter API those results will always be restricted to a maximum of 100 tweets per query. For a usage in a real-world scenario, this will most probably be insufficient and a Twitter accoung upgrade will be needed.

6 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	18
Documenting the final solution + Readme	2	2
Craft final report and presentation	10	8
Sum	66	102

Due to the change of project from the original project with the Austrian Armed Forces to a project where deep learning actually makes sense The amount of effort increased a lot. But it was possible to meet all deadlines as expected.

7 Lessons learned

The biggest learning for me was that it is mandatory to inform yourself properly and gather preliminary experience with the usage of underlying frameworks such as pytorch, keras and tensorflow(-gpu) even when using a highly abstract implementation (such as Kashgari). Due to the vast amount of errors and problems we encountered while training our model, a preliminary tutorial of the implementation specific details would have saved us quite some time.

Furthermore a big takeaway would be to execute a more detailed feasibility study before submitting the project proposal. Due to the superficial nature of my feasibility check, I had to switch to a different product during the project phase.

When it comes to BERT and models specifics I have to state, that I am impressed how much positive influence the fine-tuning had on the results, while we switched to a very simple single classification layer. Therefore I conclude that the big key for natural language processing lies in the proper and meaningful word embeddings rather than in potent models. Appropriate embeddings are more important than the models separation capabilities.

It should be noted that sophisticated Transformer architecture still perform poorly on very long texts. Traditional approaches such as bag-of-words embeddings with an SVM/RF/MLP can outperform sophisticated architectures in computational efficiency an achieved classification performance.

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.