Text Classification Competition: Twitter Sarcasm Detection

Overview

The code classifies tweets as either "Sarcasm" or "Not Sarcasm". We are given 5000 tweets with half labelled as "SARCASM" and the other half labelled as "NOT_SARCASM". The code pre-processes this training set, trains a model and evaluates the model on a test set of 1800 unlabeled tweets.

We use a large-scale pre-trained model, called DistilBERT[1], which is then fine-tuned on our twitter sarcasm dataset. Our model achieves an F1 score of 0.735 with a precision of 0.626 and a recall of 0.889, outperforming the baseline.

Implementation

We start by pre-processing the training and test datasets. The datasets are given in a *jsonl* format, which we open and save with an appropriate library called *jsonlines*. The datasets are stored in a *pandas* dataframe and the "context" variable is dropped. In our initial training we did not find this variable to be significant and hence dropped it in our final implementation. We then encode the response variable: SARCASM = 1, NOT_SARCASM = 0. Finally, we convert the *pandas* columns to lists. Note that we are not given the labels for the test data.

Once preprocessing is complete, we use *scikit-learn* to split the training set into a train set (80%) and a cross-validation set (20%).

To use the pretrained DistilBERT model we need to prepare the train/cross-validation responses (the tweets). To accomplish this, we pass the response to the tokenizer using truncation and padding. This ensures that all of our sequences are padded to the same length and are truncated to be no longer than the model's maximum input length. This transformation will allow us to feed batches of sequences into the model at the same time. This tokenizer and the pre-trained model are provided by the *transformers* package[2]. This package has become very popular as it provides deep pre-trained NLP models and is easier and faster to implement than similar packages. After tokenization we convert the tokens to tensors in preparation for training.

The training code is fairly straightforward. We create the model from *TFDistilBertForSequenceClassification* and run it using an Adam optimization (with a 5e-5 learning rate) where the binary cross entropy loss is used to measure the model's accuracy. The provided dataset is not randomized with the first 2500 rows in the dataset being labelled as sarcasm and the remaining 2500 are labelled as not sarcasm. Therefore, during training we shuffle 100 random rows and use a batch size of 16. The literature recommends 3 epochs.

However, we found that using an early stopping callback limited the number of epochs to 2 on some occasions. Therefore, we run the model with early stopping, which we found adequate.

Once the training is complete, we save the model and use it to predict the labels for the test set. We do this by looping over the test set, tokenizing the sequence, predicting the label and using a softmax to get the probabilities for both classes. We then use a 0.5 threshold to classify each row as either sarcastic or not sarcastic. The answers are saved to a text file which is uploaded to github for evaluation.

Usage

To run the code, we recommend using Amazon SageMaker. We used a *ml.p3.2xlarge* instance, which has 8 CPUs and 61 GB of RAM with a cost of $3.825/hour. Running the training on AWS takes on average 18 seconds per epoch. It is possible to run the code on Google Colab, which provides 12 GB of free RAM. However, we did not run all epochs on Google Colab as it takes about 20 minutes to train per epoch. We have provided the IPYNB file, which can be opened with Amazon SageMaker or Google Colab. If running the file on Amazon SageMaker, the reviewer will need to uncomment the following imports:

```
import boto3, re, sys, math, json, os, sagemaker
from sagemaker import get_execution_role
```

And the following AWS IAM role definition:

```
role = get_execution_role()
my_region = boto3.session.Session().region_name
```

Running the code is straightforward. There are four cells to run and can be opened using Jupyter Notebook. The third cell is the training cell and will take some time to complete, especially if the reviewer's machine's memory is not sufficient. Therefore, we recommend using either Amazon SageMaker or Google Colab. Again, for Google Colab, reduce the number of epochs to 3 or less. The fourth cell will save the text file in csv format with each row representing a tweet and its label. This file is the final "product" of the code and can be evaluated against the true labels.

References

[1] https://arxiv.org/abs/1910.01108

[2] https://huggingface.co/transformers