

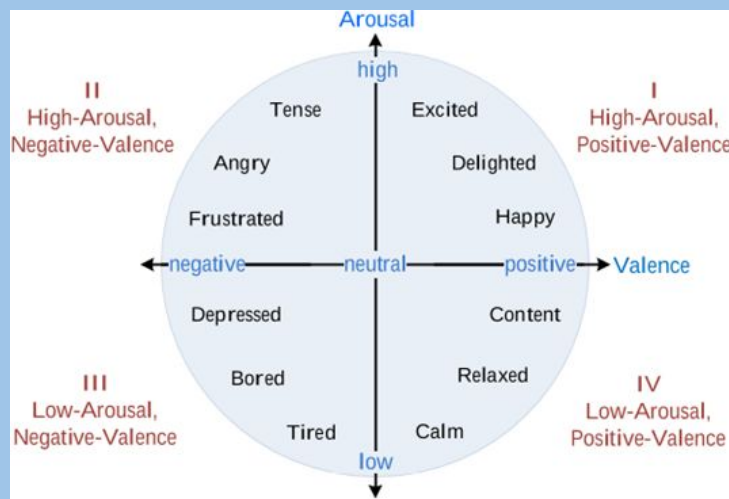
Fine-Tuning RoBERTa for Angry Tweets Prediction

Pod: Kibbeh - Group: Paragons



Sentiment Analysis and Emotion Detection

Sentiment analysis aims to classify the given input of text into positive, neutral, or negative classes. On the other hand, emotion detection aims to recognize types of feelings, which is more complicated because of the number of different classes.



Question

- "Is there a way to make the classification of angry tweets more accurate?"
- "Training language models from scratch is computationally expensive, so can we fine-tune a pre-trained sentiment analysis model on a subset of negative tweets?"

The Model and Dataset

- The Twitter-RoBERTa-base model trained on nearly 58M tweets and fine-tuned for sentiment analysis with the TweetEval benchmark.

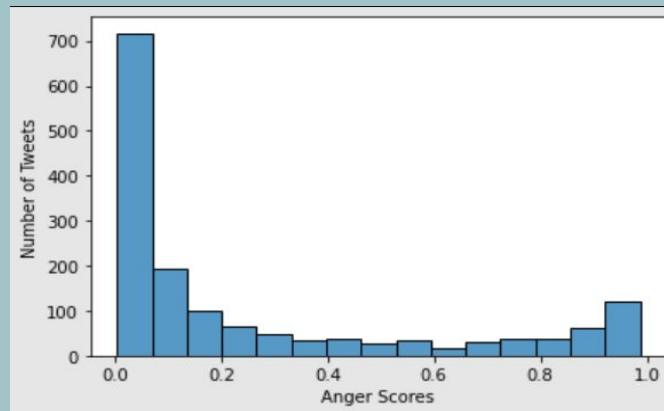
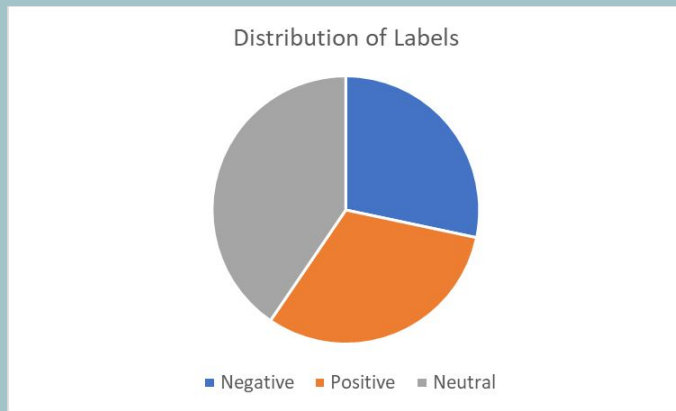
(<https://arxiv.org/pdf/2010.12421.pdf>)

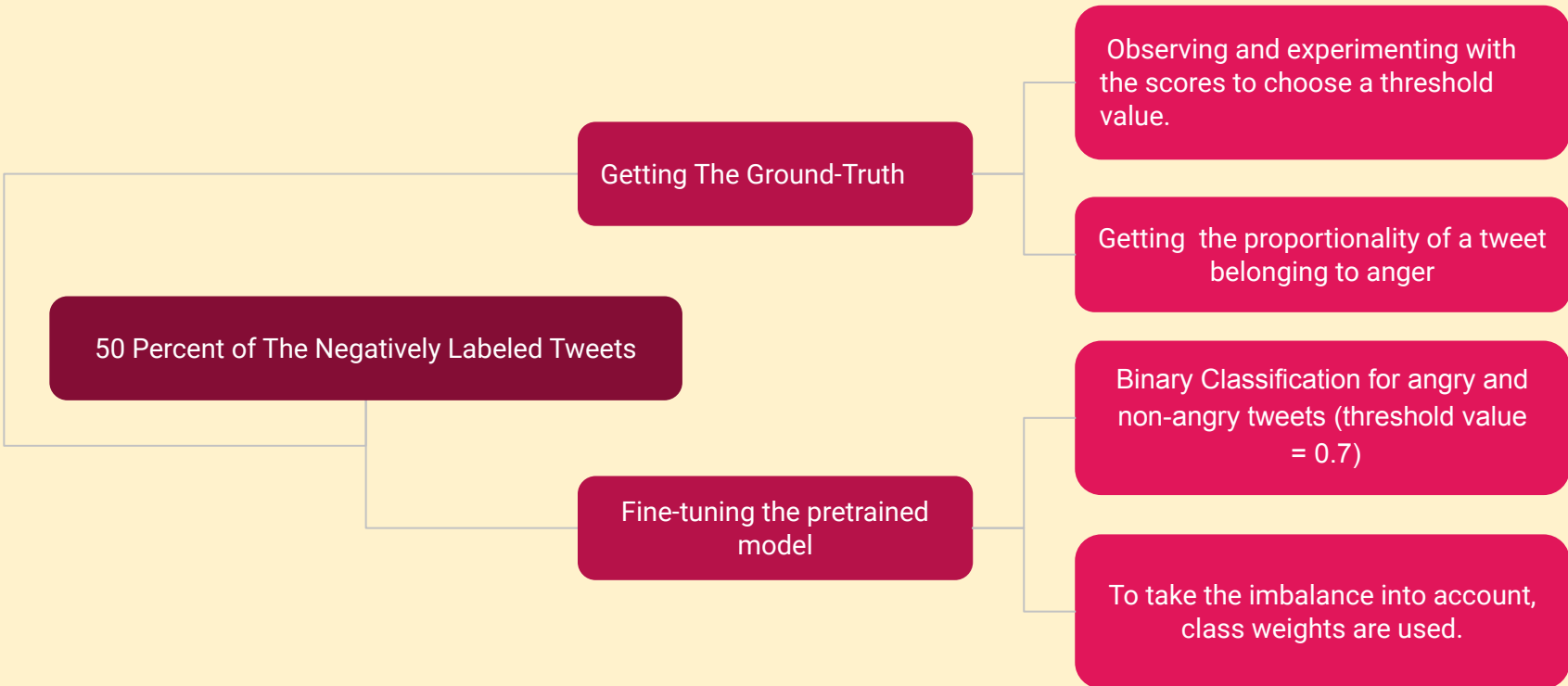
- The model had four classes of emotions, which are joy, optimism, anger and sadness.

Dataset

- The Twitter Tweets Sentiment Dataset is the dataset we used in our project.

- The dataset consists of 27481 tweets in total and the distribution of the sentiment labels are, 11118 neutral, 7781 negative and 8582 positive.
- Because we wanted to work only on the classification of angry tweets, we selected the 7781 negative labeled tweets. (28% of total dataset)





Method

We used a threshold equals to 0.7. (in order to eliminate randomness that might occur in classification, we can select only the tweets which have higher than 0.9 and lower than 0.1 probabilities of belonging to anger emotion class in the future.)


A subset of negative tweets used for this purpose. (3890 tweets)

After converting scores into binary labels (1 for angry, 0 for non-angry) we fine-tuned the model to only predict the angry tweets in an accurate way, and also to speed up learning process.

Results

- Only one threshold value (0.7)
- Learning rate of 10^{-5}

after one epoch of fine-tuning:

- Training accuracy = 91.1%
 - Training loss = 0.25
- 
- Validation accuracy = 95.12%
 - Validation loss = 0.13

```
1it [00:04, 4.01s/it]Validation Loss per 50 steps: 0.04947724938392639
Validation Accuracy per 50 steps: 100.0
51it [02:33, 2.89s/it]Validation Loss per 50 steps: 0.1444336045716031
Validation Accuracy per 50 steps: 95.09803921568627
101it [05:00, 2.87s/it]Validation Loss per 50 steps: 0.1451419404071599
Validation Accuracy per 50 steps: 94.05940594059406
151it [07:25, 2.87s/it]Validation Loss per 50 steps: 0.15357861595092626
Validation Accuracy per 50 steps: 94.5364238410596
201it [09:51, 2.87s/it]Validation Loss per 50 steps: 0.14655140867058317
Validation Accuracy per 50 steps: 94.65174129353234
251it [12:16, 2.88s/it]Validation Loss per 50 steps: 0.1340377499123078
Validation Accuracy per 50 steps: 95.11952191235059
292it [14:15, 2.93s/it]Validation Loss Epoch: 0.13453946218902424
Validation Accuracy Epoch: 95.11568123393316
Accuracy on test data = 95.12%
```



```
0it [00:00, ?it/s]/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2329: FutureWarning:
FutureWarning,
Training Loss per 50 steps: 0.7349226474761963
Training Accuracy per 50 steps: 53.125
1it [01:19, 79.98s/it]--- 79.95675349235535 seconds ---
```

Waiting to finish the current execution.
Python 3 Google Compute Engine backend
RAM: 8.38 GB/12.68 GB Disk: 38.90 GB/107.72 GB

```
Epoch_loss = []
Epoch_Accu = []
Valid_Acc=[]
Valid_loss=[]

EPOCHS = 8
for epoch in range(EPOCHS):
    x = train(epoch)
    acc,loss = valid(model, testing_loader)
    Epoch_loss.append(x[0])
    Epoch_Accu.append(x[1])
    Valid_Acc.append(acc)
    Valid_loss.append(loss)
```

```
0it [00:00, ?it/s]/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2329: FutureWarning: The `pad_to_max_length` argument is deprecated
FutureWarning,
```

```
### Validation
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

```
; nb_tr_steps=0; nb_tr_examples=0
```

Activate Windows
Go to Settings to activate Windows.



Conclusion:

It is not so easy to fine tune this kind of large model with a huge volume of data. We should test various size of learning rate, batches, epochs and so on, to get a more reliable result.

Hurdles:

- Time-consuming process to test a larger epoch_size
- Limited GPU on colab / facing errors

future work:

- Using another threshold to omit random emotion in the spectrum
- Detecting different emotions and comparing the results we will get.
- Working on sarcasm detection

Our Team

