MMD Project Report

Sentiment Classification - IMDB movie reviews using RoBERTa

Authors - Madhuri Pujari: mp5583, Lakshana Kolur: lk2719@nyu.edu

---

Overview of problem and hypothesis:

Project Description: The goal is to train a deep neural network to predict the sentiment of the movies based on long IMDB movie reviews

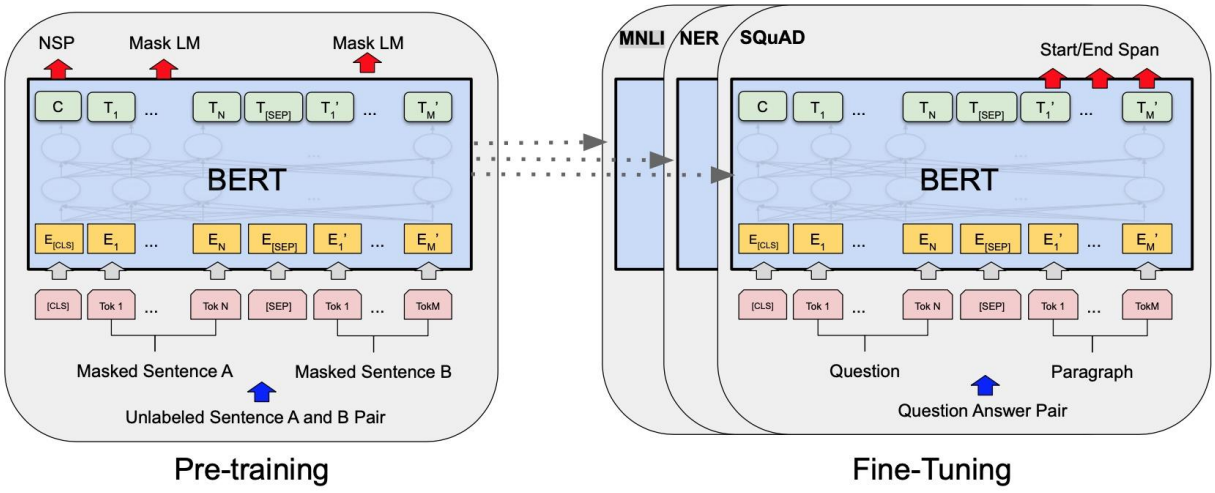
Solution Hypothesis:
Implementation of NLP models - Text mining

Often sentiment analysis uses fixed words which classify the movie into positive or negative emotions based on the words used in the review and works well with small texts.

We started our initial model with BERT, which stands for Bidirectional Encoder Representations from Transformers is a Large Language Model which in a whole is designed and built which understands context of language and is trained over huge data present over the internet, like Wikipedia pages, Research papers, and documents present over trusted websites. It is trained to perform two tasks at the base line:

1. Masked Language Model: It can predict a given token if it is masked in a sentence with the most accurate words
2. Next sentence prediction: This task given two sentences it predicts the likelihood of the second sentence following the first sentence.

The eventual use case of building BERT was to introduce a large language model which can be picked up as is and then implement and finetune it for various language tasks like Question answering, Named Entity Recognition and Multi Genre Natural Language Inference which can be seen in the figure below:



In our project we chose RoBERTa (A Robustly Optimized BERT Pretraining Approach) base model to generalize even better to downstream tasks compared to BERT internally to fine tune the model to train on our dataset for the purpose of ***classification of movie reviews into a positive or a negative sentiment***.

RoBERTa is a transformers model pre-trained on a large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labeling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts.

More precisely, it was pre-trained with the Masked language modeling (MLM) objective. Taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks: if you have a dataset of labeled sentences for instance, you can train a standard classifier using the features produced by the BERT model as inputs.

Overview of your data

About Dataset:

Large Movie Review Dataset. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing.

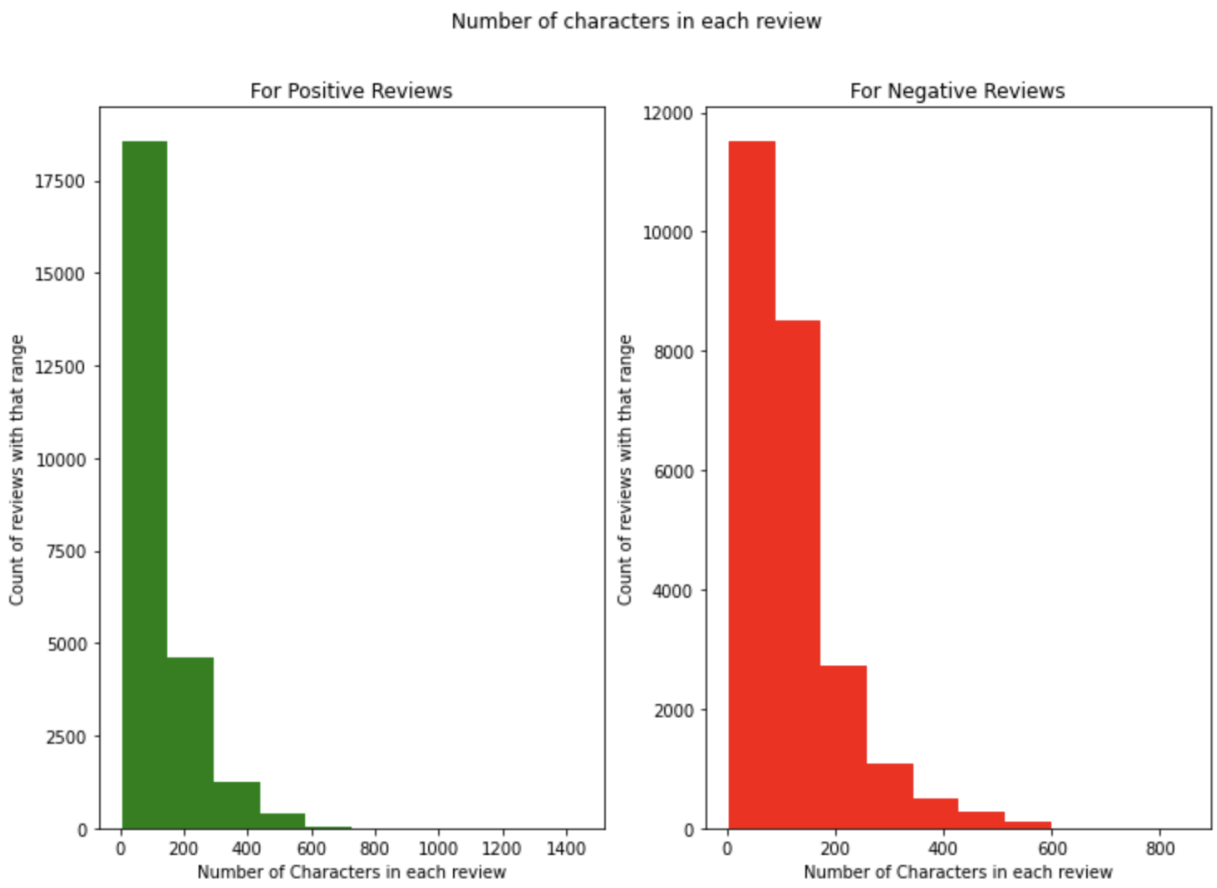
Paper References:
https://ai.stanford.edu/~ang/papers/acl11-WordVectorsSentimentAnalysis.pdf

Data Preprocessing:

1. Converting all text into lowercase for uniformity
2. Removing punctuation marks
3. Reviews extracted from the internet are mostly written on the phone and can contain emojis that may add noise, hence removing emojis
4. Removing stopwords. Stopwords are commonly used words (eg. 'the', 'a', 'and') that do not add meaning to the sentence hence removing them does not drastically change the sentence structure.
5. Removal of duplicate records from the dataset
6. Handling NULL values that may be present
7. Checking to see if there is a balanced representation of both classes of a binary classification. If imbalanced, we can implement SMOTE or another similar technique to effectively learn the decision boundary.
8. Applying Lemmatization:
   Lemmatization removes the grammar tense and transforms each word into its original form. Another way of converting words to its original form is called stemming. While stemming takes the linguistic root of a word, lemmatization is taking a word into its original lemma. For example, if we performed stemming on the word "apples", the result would be "appl", whereas lemmatization would give us "apple". We have used lemmatization over stemming as it's much easier to interpret.
9. Finding out the most common words in the dataset and most common bi-grams, removing the words which do not add value to the sentiment

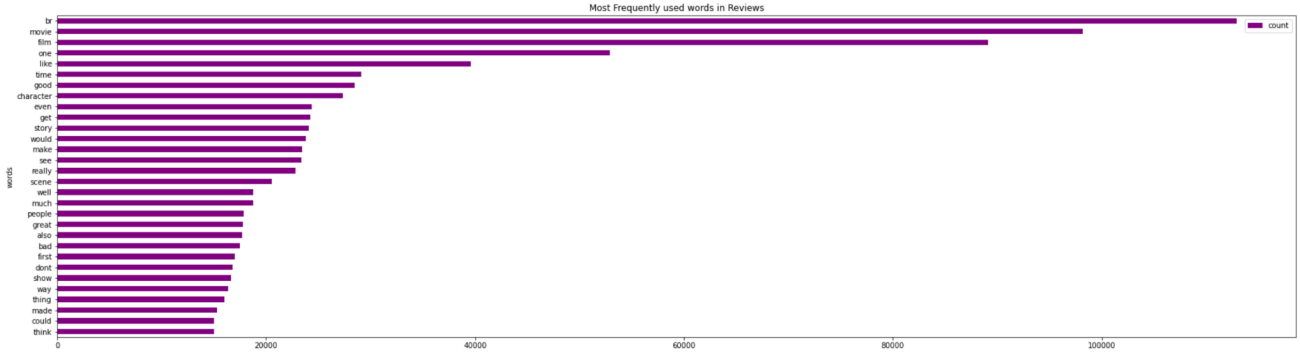
EDA Findings:

- We saw that positive reviews have a larger number of characters (~ 17500) as compared to negative (~ 12000). However, there are more negative reviews with a larger number of characters (in the range of 0-2000) as compared to positive reviews. This can be attributed to the fact that users/ movie-watchers more vehemently express their dislike, whereas people who enjoyed the movie usually do not have much to say. Ex. This can be seen for product reviews as well. People take time to write a negative review if they disliked the product, whereas if they like it, they most likely would not give a review.

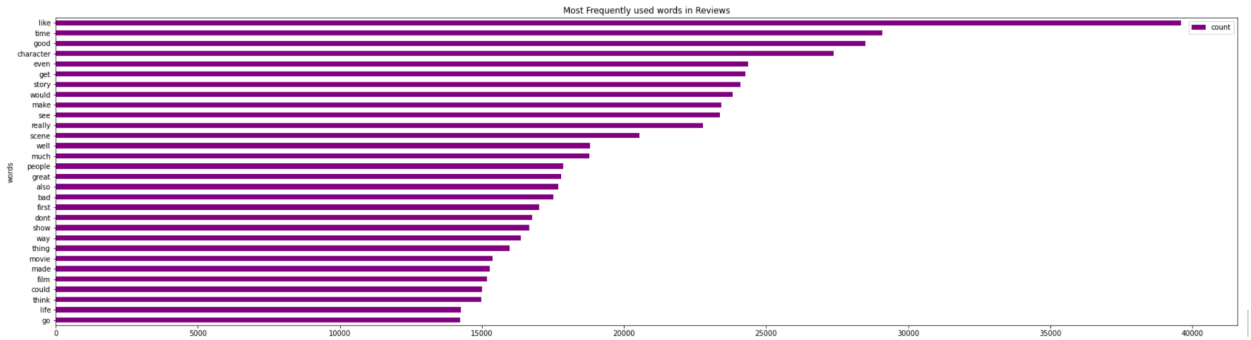


- The dataset was balanced with equal representation for both classes - positive and negative.
- Lemmatization greatly reduced the number of words that go through the model.
- We found out that words like 'br', 'movie', 'film' are the most frequently used in the reviews. However, they do not add value to understanding the sentiment of the review any better, therefore we removed these thereby reducing the size of the text by a considerable amount.

Before analyzing most frequent words:



After removing most frequent words that are unnecessary:



After removing words like ‘br’ and ‘movie’ that do not add any useful information for our model to learn to predict sentiment, the most frequent words (as we can see in the graph above) are ‘like’, ‘time’, ‘good’ and ‘character’.
This text preprocessing is especially helpful for NLP use-cases such as in our project. It removes noises and other text that may skew the performance of the model resulting in decreased performance.

Design Choices - Modeling techniques used and why

Transformer Model: (roberta-base)

As mentioned above, we picked RoBERTa because it can be used to generalize even better to downstream tasks compared to BERT internally to fine tune the model to train on our dataset for the purpose of ***classification of movie reviews into a positive or a negative sentiment***.

- **RobertaModel:**
  The bare RoBERTa Model transformer outputs raw hidden-states without any specific head on top. This model inherits from PreTrainedModel. This model is also a PyTorch torch.nn.Module subclass. Can use it as a regular PyTorch Module and that's what we have done. The model can behave as an encoder (with only self-attention) as well as a decoder, in which case a layer of cross-attention is added between the self-attention layers. To behave as an decoder the model needs to be initialized with the is_decoder argument of the configuration set to True.

- **Why RobertaTokenizerFast?**
  Construct a "fast" RoBERTa tokenizer (backed by HuggingFace's tokenizers library), derived from the GPT-2 tokenizer, using byte-level Byte-Pair-Encoding.
  This tokenizer has been trained to treat spaces like parts of the tokens (a bit like sentences piece) so a word will be encoded differently whether it is at the beginning of the sentence (without space) or not
  It has proven to be faster while processing the data, as compared to RobertaTokenizer

- **Activation Function used: ReLU**
  The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The ***main advantage*** of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0. Hence, our preference to use this function

- **CrossEntropyLoss:**
  Cross-entropy builds upon the idea of information theory entropy and measures the difference between two probability distributions for a given random variable/set of events. Cross entropy can be applied in both binary and multi-class classification problems. We'll discuss the differences when using cross-entropy in each case scenario.

- **Adam Optimizer:**
  Adam is a popular algorithm in the field of deep learning because it achieves good results fast, it is a replacement optimization algorithm for stochastic gradient descent for training deep learning models.

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
Adam is relatively easy to configure where the default configuration parameters do well on most problems.

- **Wandb:**
  A data visualization tool, we found this really useful to store our model metrics and display real time stats as the model is training/validating.

Code Walkthrough

1. We will start with creation of the **pre-process class** - This defines how the text is pre-processed before working on the tokenization, dataset and dataloader aspects of the workflow. In this class the data frame is loaded and then the 'sentiment' column is used to create a new column in the dataframe called 'encoded_polarity' such that if:
   'sentiment = positive' then 'encoded_polarity = 0'
   'sentiment = negative' then 'encoded_polarity = 1'
   Followed by this, the 'sentiment' column is removed from the dataframe.
   The 'dataframe' and 'encoded_polarity' dictionary is returned.
   This method is called in the 'run()' function.

2. We now define the Dataset class - This defines how the text is pre-processed before sending it to the neural network. This dataset will be used for the Dataloader method that will feed the data in batches to the neural network for suitable training and processing. We made the function call from run(), here the Dataloader and Dataset will be created. Dataset and Dataloader are constructs of the PyTorch library for defining and controlling the data pre-processing and its passage to the neural network.

3. CustomDataset Dataset Class
   - This class is defined to accept the Dataframe as input and generate tokenized output that is used by the Roberta model for training.
   - We are using the Roberta tokenizer to tokenize the data in the 'review' column of the dataframe.
   - The tokenizer uses the 'encode_plus' method to perform tokenization and generate the necessary outputs, namely: 'ids', 'attention_mask', 'encoded_polarity' transformed into the 'targets' tensor.
   - The CustomDataset class is used to create 2 datasets, for training and for validation.
   - Training Dataset is used to fine tune the model: 70% of the original data
   - Validation Dataset is used to evaluate the performance of the model. The model has not seen this data during training.

4. **return_dataloader:** Called inside the 'run()'
   'return_dataloader' function is used to for creating training and validation dataloader that load data to the neural network in a defined manner. This is needed because all the data from the dataset cannot be loaded to the memory at once, hence the amount of data loaded to the memory and then passed to the neural network needs to be controlled.
   Internally the 'return_dataloader' function calls the pytorch Dataloader class and the CustomDataset class to create the dataloaders for training and validation.
   This control is achieved using the parameters such as 'batch_size' and 'max_len'.
   Training and Validation dataloaders are used in the training and validation part of the flow respectively

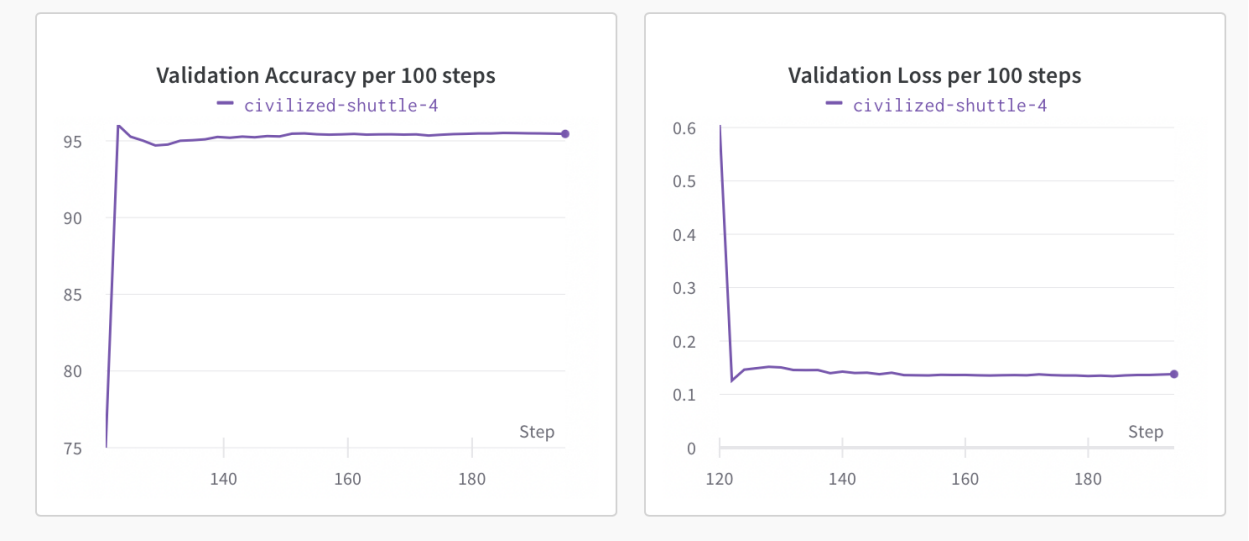
5. Defining a **Model**: Neural Network
   - We will be creating a neural network with the ModelClass.
   - This network will have the Roberta Language model and a few by a dropout and Linear layer to obtain the final outputs.
   - The data will be fed to the Roberta Language model as defined in the dataset.
   - Final layer outputs what will be compared to the encoded_polarity to determine the accuracy of model prediction.
   - We will initiate an instance of the network called model. This instance will be used for training and then to save the final trained model for future inference.
   - The return_model function is used in the run() to instantiate the model and set it up for TPU execution.

6. **Fine Tuning the Model**:
   - Here we define a training function that trains the model on the training dataset created above, specified number of times (EPOCH), An epoch defines how many times the complete data will be passed through the network.
   - This function is called in the run()
   - Following events happen in this function to fine tune the neural network:
   - The epoch, model, device details, testing_ dataloader, optimizer and loss_function are passed to the train () when its called from the run()
   - The dataloader passes data to the model based on the batch size.
   - The output from the neural network: outputs is compared to the targets tensor and loss is calculated using loss_function()
   - Loss value is used to optimize the weights of the neurons in the network.
   - After every 100 steps the loss value and accuracy is logged in the wandb service. This log is then used to generate graphs for analysis. Such as these
   - After every epoch the loss and accuracy value is printed in the console. Also, logged into the wandb service.

7. Validating the **Model Performance**:
   - During the validation stage we pass the unseen data(Validation Dataset), trained model, and device details to the function to perform the validation run. This step

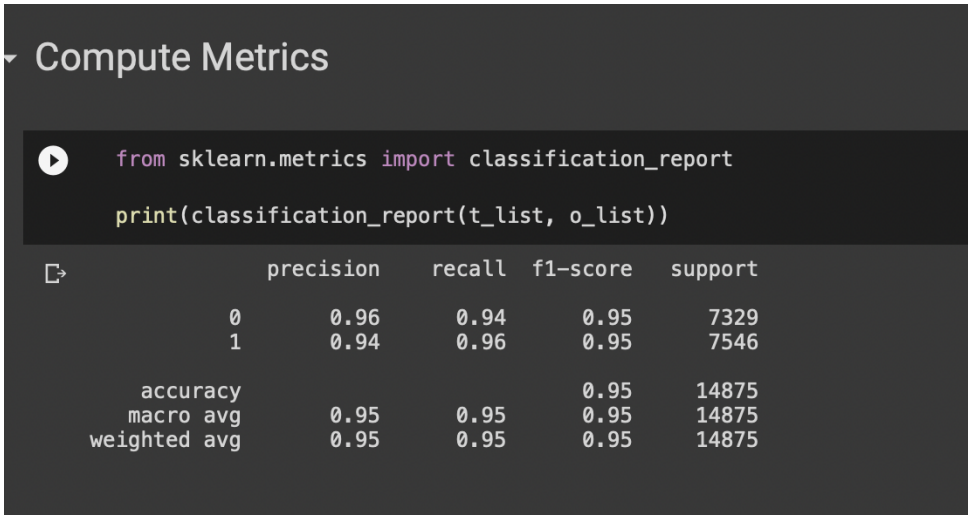
generates a new encoded_sentiment value for the dataset that it has not seen during the training session.
    - This is then compared to the actual encoded_sentiment, to give us the Validation Accuracy and Loss.
    - This function is called in the run()
    - This unseen data is the 30% of IMDB Dataset which was separated during the Dataset creation stage. During the validation stage the weights of the model are not updated. We use the generate method for generating new text for the summary.
    - The generated validation accuracy and loss are logged to wandb for every 100th step and per epoch.

8. Saving model as pickle file
9. Evaluation:



The validation accuracy keeps increasing upto a certain extent ~95% and then remains constant.
And the validation loss decreases per 100 steps and remains constant.
This shows that the model is performing well, as expected on the validation dataset (unseen data) as well.

10. Compute Metrics



Computing F1 score, precision and recall for the model shows that the model is predicting with high accuracy for both classes - 0 (negative) and 1 (positive).

Comparison with the model using RobertaSequenceClassifier - took us 6 hours to complete the fine tuning on a small_dataset of 1000 records, our model runs in 2 hours with the complete dataset.

Data Visualization:

Model performance

FullDatasetTrainedModel

This model was trained on the full dataset and gave an accuracy of 95%

Privacy	PRIVATE
Tags	+
Author	madhuripujari
State	finished
Start time	May 1st, 2022 at 10:50:50 pm
Duration	2h 39m 10s
Run path	madhuripujari/MMD-Project-IMDB-Sentiment-Analysis/3p2smh00
Hostname	5e639a184475
OS	Linux-5.4.188+-x86_64-with-Ubuntu-18.04-bionic
Python version	3.7.13
Python executable	/usr/bin/python3
Colab link	https://colab.research.google.com/notebook#fileId=14BoW86bW1oxLQkp-SbUJPDcZmAQsJCke
Command	`BERT - Sentiment Analysis - mp5583.ipynb`
System Hardware	CPU count 2
W&B CLI Version	0.12.15

Hyperparamaters:

Config

Raw

Config parameters describe your model's inputs. Learn more

Search keys | Ignore underscore keys

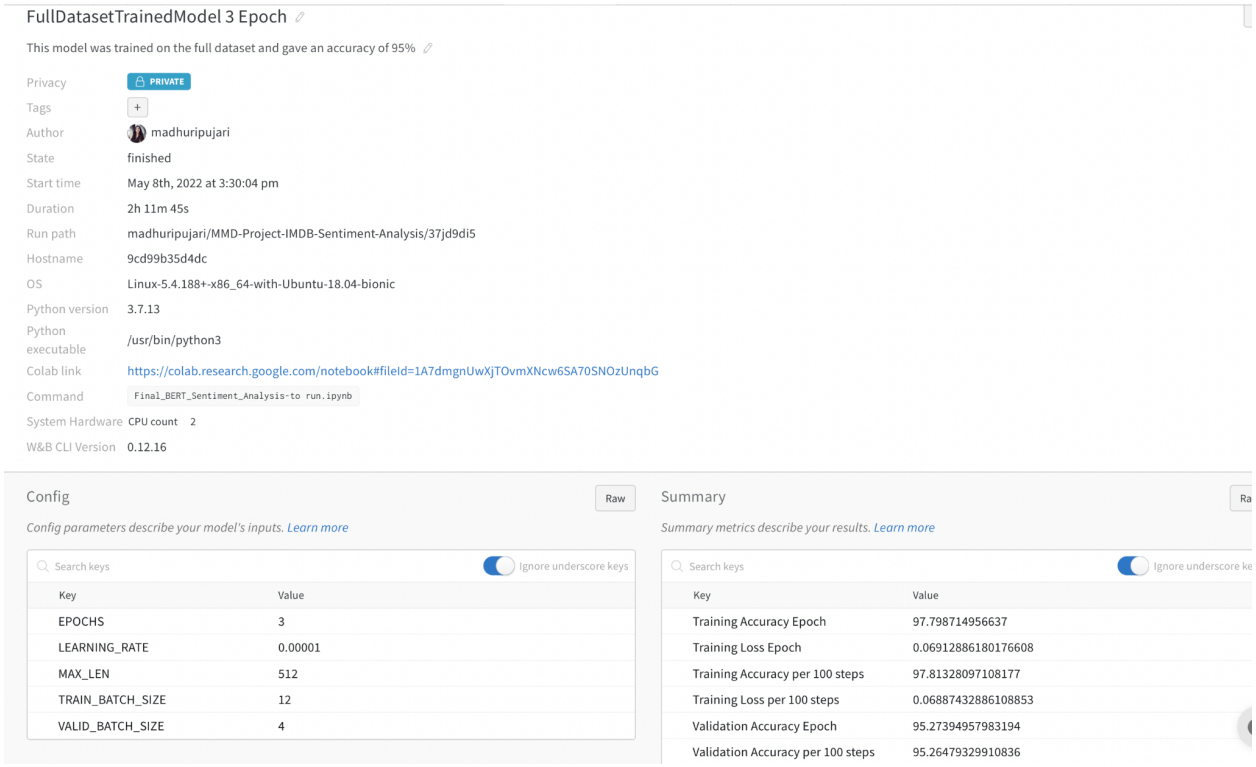
Key	Value
EPOCHS	2
LEARNING_RATE	0.00001
MAX_LEN	512
TRAIN_BATCH_SIZE	12
VALID_BATCH_SIZE	4

Summary

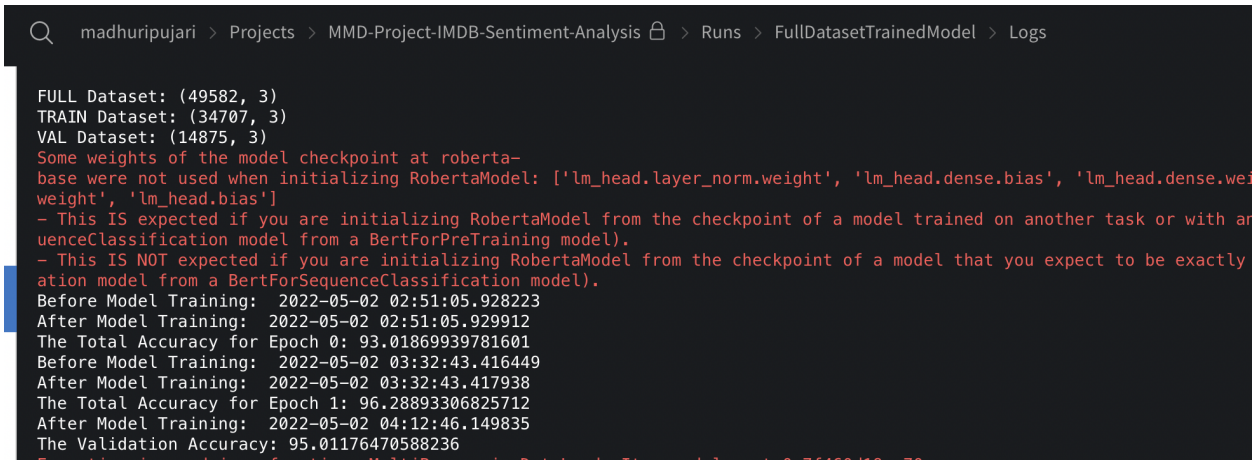
Summary metrics describe your results. Learn more

Search keys | Ignore unders

Key	Value
Training Accuracy Epoch	96.28893306825712
Training Loss Epoch	0.10682287338402892
Training Accuracy per 100 steps	96.28406521480424
Training Loss per 100 steps	0.10705891580444583
Validation Accuracy Epoch	95.01176470588236
Validation Accuracy per 100 steps	95.02161577951904
Validation Loss Epoch	0.1413670580947592
Validation Loss per 100 steps	0.1409207507471902



Output logs:



Accuracy For Epoch 0 - 93%
Accuracy for Epoch 1 - 96%
Validation Accuracy - 95%

Training loss Epoch: The cross-entropy loss per epoch while training has a downward trend.
Training accuracy Epoch: Conversely, the accuracy while training is increasing per epoch.
Training Loss/Accuracy per 100 steps: Same trend can be observed per 100 steps of the training, instead of per epoch in the next 2 graphs.


```
{
  "Training Loss per 100 steps": 0.10705891580444583,
  "_timestamp": 1651465177,
  "_runtime": 5328,
  "_step": 197,
  "Training Accuracy per 100 steps": 96.28406521480423,
  "\nTraining Loss Epoch": 0.10682287338402893,
  "\nTraining Accuracy Epoch": 96.28893306825712,
  "Validation Loss per 100 steps": 0.1409207507471902,
  "Validation Accuracy per 100 steps": 95.02161577951905,
  "Validation Loss Epoch": 0.1413670580947592,
  "Validation Accuracy Epoch": 95.01176470588236
}
```

Older Approach:

RobertaSequenceClassifier

This model was fine tuned with a very small dataset but it ran for 6+ hours - I used the hugginface model as it is, without optimizations

Privacy	PRIVATE
Tags	+
Author	madhuripujari
State	crashed
Start time	May 3rd, 2022 at 7:07:24 pm
Duration	6h 32m 14s
Run path	madhuripujari/huggingface/245dlr9g
Hostname	85746288480f
OS	Linux-5.4.188+-x86_64-with-Ubuntu-18.04-bionic
Python version	3.7.13
Python executable	/usr/bin/python3
Colab link	https://colab.research.google.com/notebook#fileId=16_Q-z-IDxt50CV35hIEU58drpVYx0zgp
Command	`RoBERTA with IMDB.ipynb`
System Hardware	CPU count 2
W&B CLI Version	0.12.16

Summary

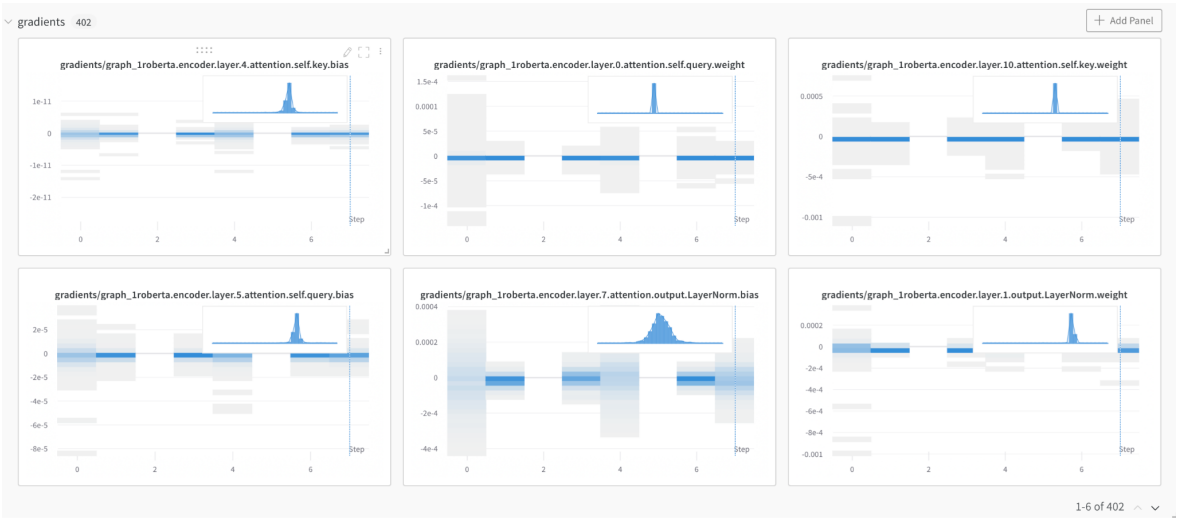
Summary metrics describe your results. Learn more

Search keys Ignore under

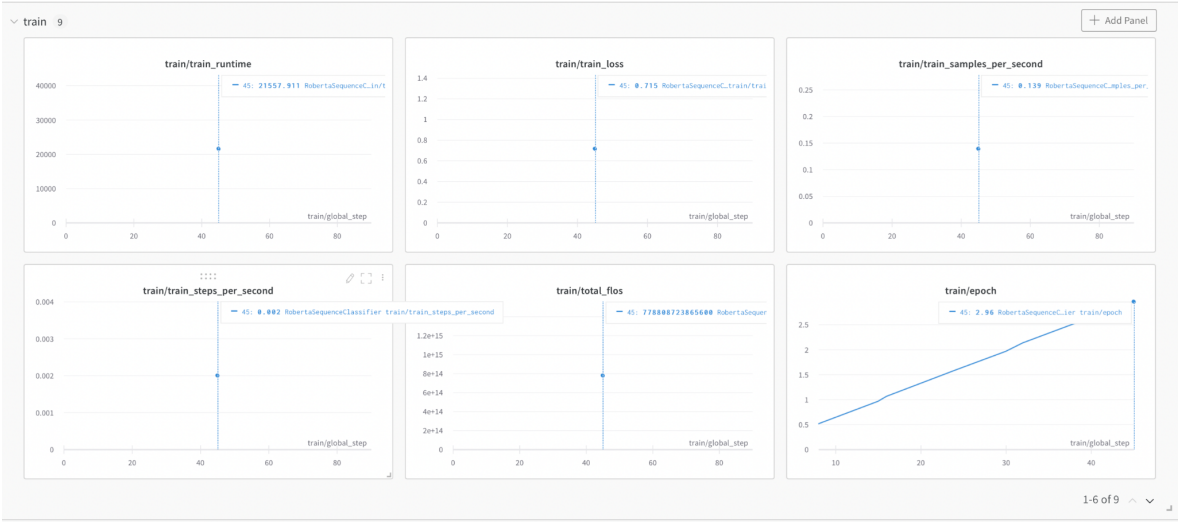
Key	Value
eval (8 collapsed)	
train	
epoch	2.96
global_step	45
learning_rate	0.00000400000000000001
loss	0.6919
total_flos	778808723865600
train_loss	0.7150134245554606
train_runtime	21557.9109
train_samples_per_second	0.139
train_steps_per_second	0.002

> root / wandb-metadata.json

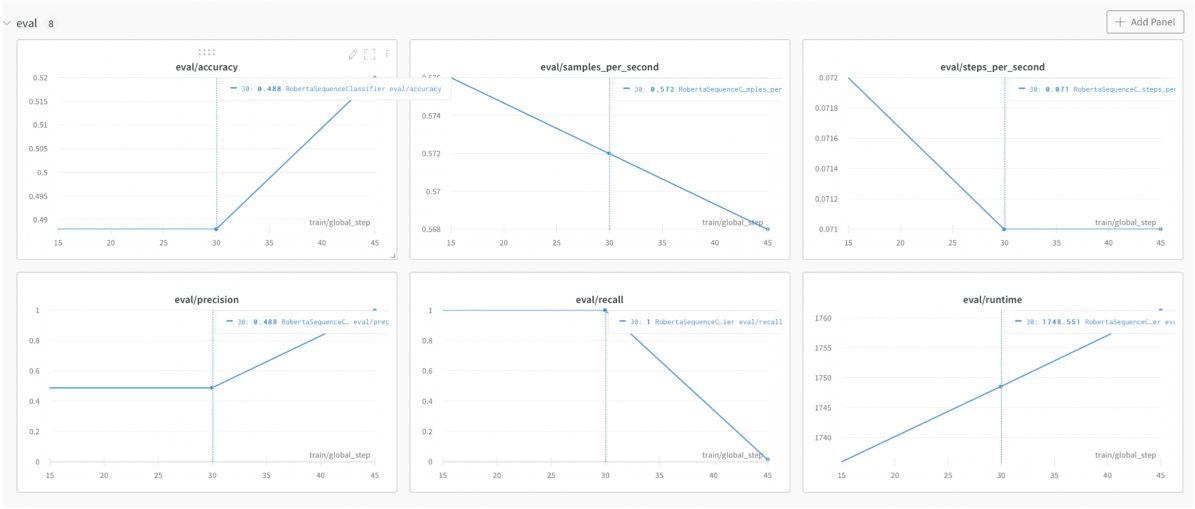

```
{
    "os": "Linux-5.4.188+-x86_64-with-Ubuntu-18.04-bionic",
    "python": "3.7.13",
    "heartbeatAt": "2022-05-03T23:07:28.198796",
    "startedAt": "2022-05-03T23:07:22.444087",
    "docker": null,
    "cpu_count": 2,
    "cuda": null,
    "args": null,
    "state": "running",
    "program": "RoBERTA with IMDB.ipynb",
    "colab": "https://colab.research.google.com/notebook#fileId=16_Q-z-IDxt50CV35hIEU58drpVYx0zgp",
    "host": "85746288480f",
    "username": "root",
    "executable": "/usr/bin/python3"
}
```

train :



Eval:



Business applications

Looking at the specific use case of movies, the findings obtained from our sentiment analysis model can be used to understand the public opinion about the movie and understand how they can make better movies in the future - understand what they did right and what they did wrong.

It can also be used for a smaller sample of population to assess their reactions to the movie before releasing it to the public.

Sentiment analysis has been most widely used for social media monitoring and brand reputation management. Brands, especially social media platforms are using this to track the perception of customers and keep a close eye on the direction their brand development is taking.

It can also be used by businesses for market research and competitor analysis.

Another application of sentiment analysis can be finding accurate target audiences and customizing their products to their needs.

Further sentiment classification can be done on any kind of data, say, tweets, political campaigns, what people are talking about the trends on social media etc.

Implementation of our project on a live system

The overall pipeline of our project in a live implementation:

- For production use, the front-end would be hosted on AWS S3 bucket. The end-users can access the data through a web application where they can input user reviews and get the sentiment of that review as predicted by the model.
- The Roberta model requires a large amount of memory, space and other resources so it would be hosted on the cloud, preferably on an EC2 instance with a Docker container

containing the model. (The model could also be run on Sagemaker as an alternate option)
- The data could be stored on the cloud too, on DynamoDB.
- As and when more data comes in, the model can start learning new parameters and can be tuned depending on them.
- Interactions between the S3 bucket and the backend can be done via the API Gateway.

This is for our specific use-case, however the most widespread applications of sentiment analysis are for large businesses trying to interpret the customer feedback and reviews and improving their products based on that.

Additions and improvements

Most of the movie reviews are more than 512 long even after data preprocessing, using a longformer transformer will significantly increase the performance of this model and a sliding window concept can achieve broader context attention span. This will significantly increase the model performance.
Using LSTMs for real-time data analysis (ex. Twitter feed)
Training the model for specific use-cases - understanding tone of the user, sarcasm detection, negation handling and more.

References:

https://arxiv.org/abs/1810.04805
https://arxiv.org/abs/1907.11692
https://arxiv.org/abs/2004.05150
https://arxiv.org/abs/2004.05150
https://arxiv.org/abs/2004.08483
https://arxiv.org/abs/1901.02860
https://arxiv.org/abs/1804.05685