Sentiment Analysis, Classification with BERT and Huggingface

Elias Adjei, Mohamed Khalid, Marat Muntinov

August 9, 2021

Summary

Abstract

Our model can accept a review and give us a sentiment classification of the review. We apply the hugging face transformer and transfer learning to make use of a pre-trained model to predict a sentiment out of a class of three, respectively negative, neutral and positive. Sentiment analysis and classification can help inform us in our business uses if users are appreciating our product or service.

1 Introduction

In this exercise, we will obtain and fine-tune BERT base model for sentiment analysis. We'll do the required text preprocessing such as adding special tokens, padding, and attention masks. Finally we will build a Sentiment Classifier using the amazing Transformers library provided by Hugging Face. We will make use of the reviews dataset from google play.

1.1 The BERT Model

BERT stands for Bidirectional Encoder Representations from Transformers. According to the BERT paper, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layersm much unlike recent language representation models, such as LSTM. As a result, the pre-trained BERT model we will download can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

Some important features of the BERT model are:

- Bidirectional - to understand the text you're looking you'll have to look back (at the previous words) and forward (at the next words)
- Transformers - The "Attention Is All You Need" paper presented the Transformer model. The Transformer reads entire sequences of tokens at once. This well preserves the context of our natural languages, allowing us to avoid the contextual loss problem. In a sense, the model is non-directional, while LSTMs read sequentially (left-to-right or right-to-left). The attention mechanism allows for learning contextual relations between words.
- (Pre-trained) contextualized word embeddings - The ELMO paper introduced a way to encode words based on their meaning/context. Nails has multiple meanings - fingernails and metal nails.

BERT was trained by masking 15% of the tokens with the goal to guess them. An additional objective was to predict the next sentence. BERT our of the box if very capable at asked Language Modeling (Where we let the model guess striked out words in our input) and Next Sentence Prediction (where

BERT predicts the next item in our sentence based on an input). BERT is simply a pre-trained stack of Transformer Encoders. There exists two versions of BERT, - one with 12 encoders (BERT base) and 24 encoders (BERT Large).

1.1.1 Is This Thing Useful in Practice?

The best part is that you can do Transfer Learning (thanks to the ideas from OpenAI Transformer) with BERT for many NLP tasks - Classification, Question Answering, Entity Recognition, etc. You will discover that BERT offers many useful use cases. We can train with small amounts of data and achieve great performance with just a few training epochs!

1.2 Our Objectives

We will:

- Preprocess text data for BERT and pytorch to understand
- Build PyTorch Dataset (tokenization, attention masks, and padding)
- Use Transfer Learning to build Sentiment Classifier using the Transformers library by Hugging Face
- Save our model as a binary file for later use
- Evaluate the model on test data
- Predict sentiment on raw text

2 Exploratory Data Analysis

Before we begin, we have to fetch and prepare our data in the correct format BERT can use. We do not have much work to do, and most of the heavy lifting will actualy be handled by tokenizer which has been conviniently already been provided for us.

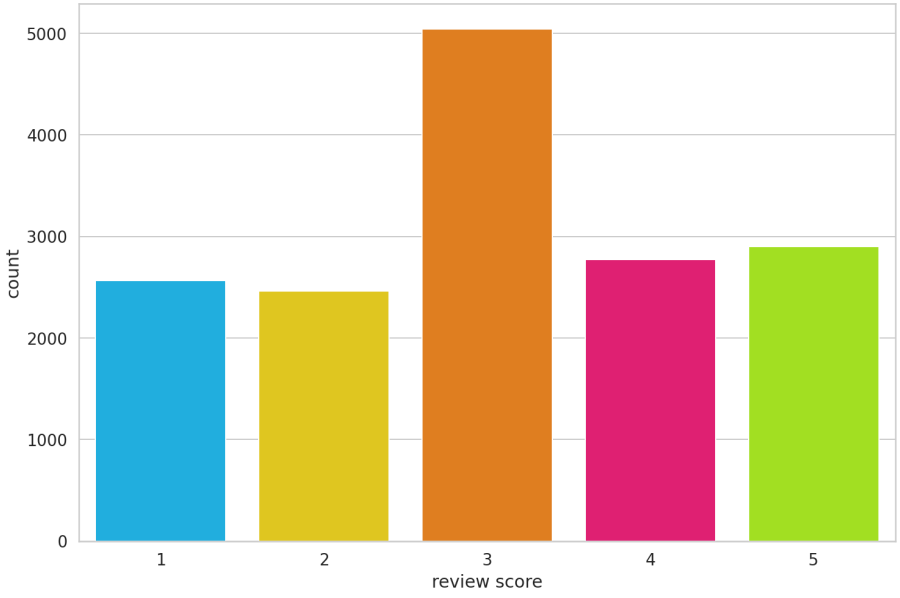
2.1 Setup

We'll perform some quick setup, these will come in handy later when we train and evaluate our model. We will also be using the GPU mostly for our mdoeling, as recommended by the BERT paper. We'll need the Transformers library by Hugging Face, so we'll go ahead and download it.

2.2 A Closer Look at Our Data

We'll load the Google Play app reviews dataset, gathered from the tutorial we watched on youtube. The google play reviews dataset is publicly available at here and here We have about 16k examples of reviews. Upon a closer inspection of our data for missing values, we realize that there aren't any.

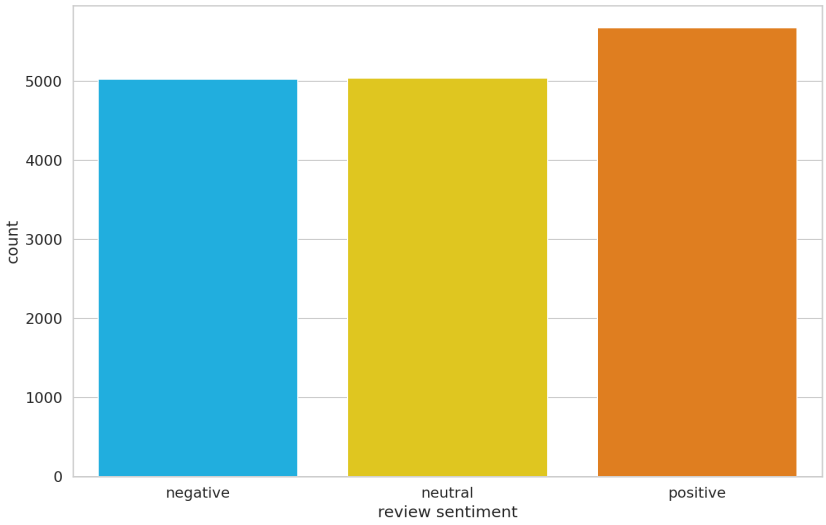
2.2.1 EDA review scores for our dataset



We are in luck since we do not have to drop any columns or imputate any missing data.

Next, we check if we have any class imbalances across our dataset. Our dataset is initialy hugely imbalanced, but that's fine, We will convert the dataset into negative, neutral and positive sentiment, totaling 3 classes.

2.2.2 EDA balanced review sentiments



Our diagram is paints a pretty clear picture of how our data looks. The balance is mostly restored after our custom scoring. Next, we need to pre-process our data so pytorch can handle it.

3 Data Preprocessing

Since Machine Learning models don't work with raw text, We need to convert all the text to numbers. BERT requires even more attention, pun intended. We need to effectively:

- Add special tokens to separate sentences and do classification
- Pass sequences of constant length (introduce padding to fill up empty spaces)
- Create array of 0s (pad token) and 1s (real token) called attention mask

The Transformers library provides a wide variety of Transformer models including BERT. It works with TensorFlow and PyTorch, for the purpose of our exercise we will be using pytorch. It also includes prebuilt tokenizers that will do the heavy lifting for us. We will use the case-sensitive model since more context may be attributed to cased words or sentences. The cased version simply works better. Intuitively, that makes sense, because "HEY!" might convey more sentimental meaning than "hey".

3.1 Tokenization

After balancing our dataset, we load a pre-trained Bert Tokenizer next to run some basic operations can convert the text to tokens and tokens to unique integers.

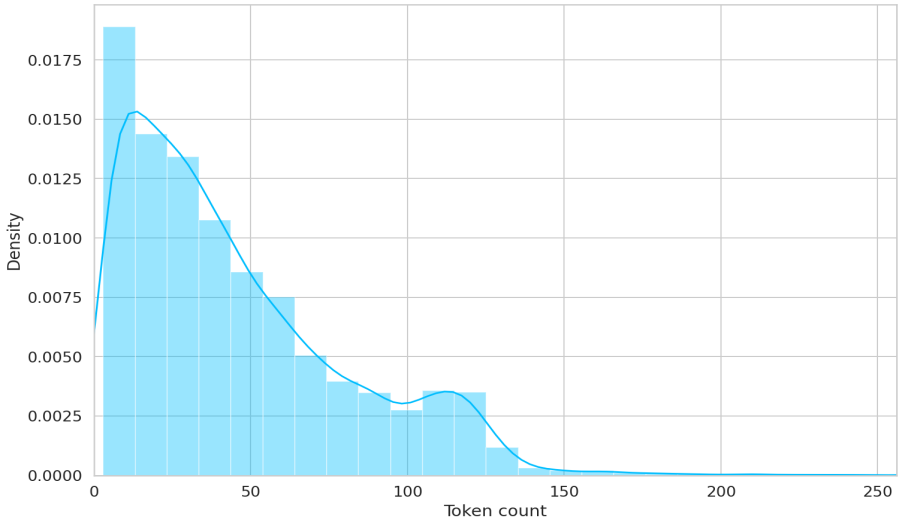
3.1.1 Special tokens

[SEP] - marker for ending of a sentence
[CLS] - marker indicating the start of each sentence, so BERT knows we're doing classification
[PAD] - the padding token

3.1.2 Perform tokenization

All of that work can be done using the encodeplus method, which we get from tokenizer. Our token ids are now stored in a Tensor and padded to a length of 32. The attention mask also has the same length BERT works with fixed-length sequences. We'll use a simple strategy to choose the max length. of our sequences. We will then store the token length of each review.

3.1.3 Token sequence distrubution across our dataset



Most of the reviews seem to contain less than 128 tokens, but we'll be on the safer side and choose a maximum length of 160

We'll then proceed to create the PyTorch dataset. The tokenizer does most of the heavy lifting for us in this part if our work. We also return the review texts, so it'll be easier to evaluate the predictions from our model. From there

we split our data into test, train and validation sets. We also need to create a couple of data loaders. Here's a helper function to do it. Our data loader will take our data set, divide them into batches and tokenize them. It will return a format that will be easier for us to handle.

4 Our Model

There are a lot of helpers that make using BERT easy with the Transformers library. Depending on the task you might want to use BertForSequenceClassification or BertForQuestionAnswering. For our use case we'll use the basic BERT model and build our sentiment classifier on top of it using transfer learning.

4.1 Sentiment Classification with BERT and Hugging Face

We have the hidden state for each of our 32 tokens (the length of our example sequence). But why 768? This is the number of hidden units in the feedforward-networks. We can verify this by checking the config of the BERT model. We will then create a classifier that uses the BERT model, which will give us the class that our given review falls into.

4.2 Building the Classifier

Our classifier delegates most of the heavy lifting to the BertModel. We use a dropout layer for some regularization and a fully-connected layer for our output. Note that we're returning the raw output of the last layer since that is required for the cross-entropy loss function in PyTorch to work. For this task we will use the GPU. Also, to get the predicted probabilities from our trained model, we'll apply the softmax function to the outputs.

4.3 Training our Model

To reproduce the training procedure from the BERT paper, we'll use the AdamW optimizer provided by Hugging Face. It corrects weight decay, so it's similar to the original paper. We will then use 4 epochs are recommended by the paper. We'll also use a linear scheduler with no warmup steps to train our model. Training the model has some interesting paramters. The scheduler gets called every time a batch is fed to the model. We're avoiding exploding gradients by clipping the gradients of the model by clipping the gradient when neccesary. We will also have another helper that will help us evaluate our model based on our data loader. Using these two, we can create a training loop. We'll also store the training history After training, we will store the state of the best model, indicated by the highest validation accuracy. We can retrieve this later and perform our classification instead of going through the whole training process again. We record an accuracy of 86% which is pretty good for our low number of epochs.

4.3.1 Model stats for nerds

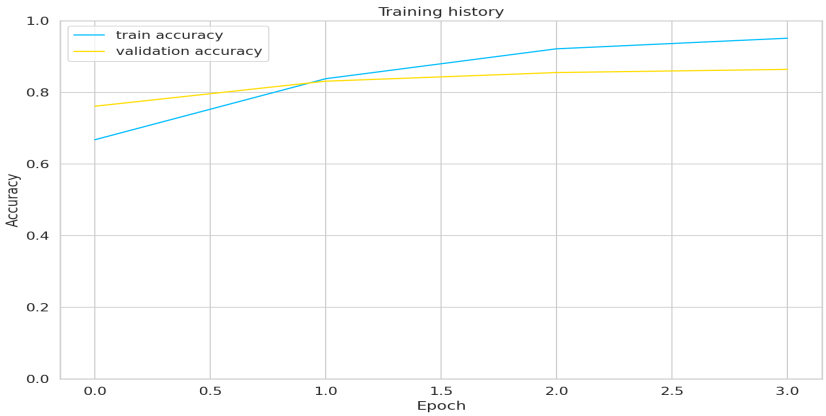
How did we come up with which hyperparameters to adjust?
The BERT authors have some recommendations for fine-tuning:
Batch size: 16, 32
Learning rate (AdamW): 5e-5, 3e-5, 2e-5
Number of epochs: 2, 3, 4

For the purpose of our exercise, we will stick with the recommendation. We realize that increasing the batch size reduces the training time significantly, but gives us lower accuracy.

4.4 Evaluating Our Model

It took a while to train our model. More epochs will definitely be better, if we spend more time training our model. We can look at the training vs validation accuracy:

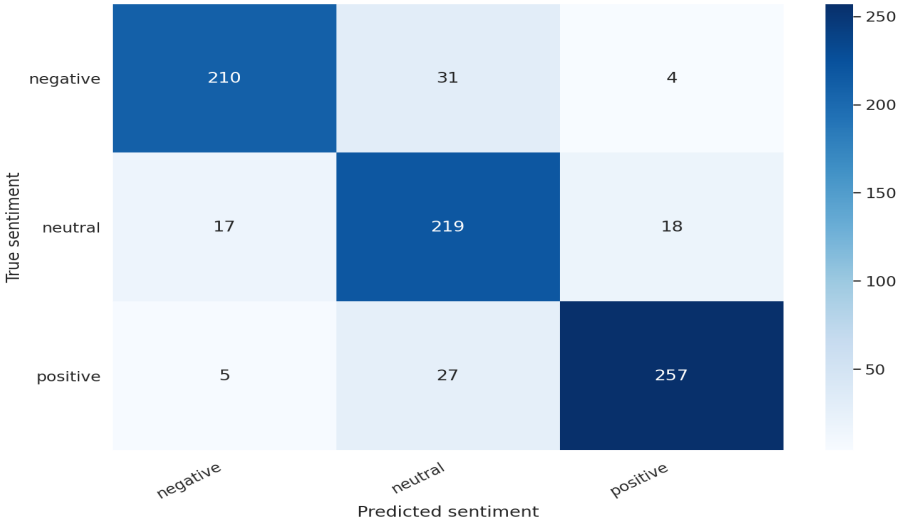
4.4.1 Our train, validation accuracy



We are guessing the training accuracy starts to approach 100% after 10 epochs or more. We could try to fine-tune the parameters a bit more, but this will be good enough for us.

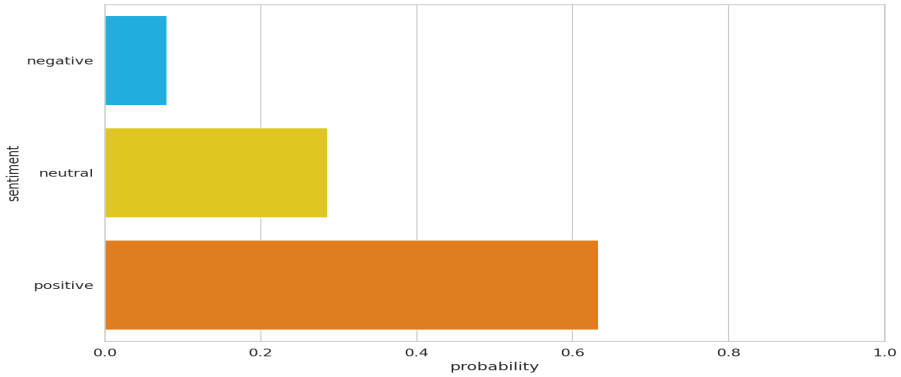
To actually figure out how good our model is on predicting sentiment, we start by calculating the accuracy on the test data. The accuracy is about 1% higher on the test set.
Our model seems to generalize well. Upon further scrutiny, our model finds it really hard to classify neutral (3 stars) reviews. Logically speaking, those should be hard to classify.

4.4.2 Confusion Matrix



The confusion matrix confirms that our model is having difficulty classifying neutral reviews. It mistakes those for negative and positive at a roughly equal frequency. That's a good overview of the performance of our model.

4.4.3 How Confident is Our Model?



Our model seems pretty confident with its classifications.

5 Conclusion

In this exercise, we used BERT for sentiment analysis. We built a custom classifier using the Hugging Face library and trained it on our app reviews dataset, and validated our model with the validation set. We achieved quite a high level of accuracy, with our model generalizing well. We achieve 86% our train set, and 87% on our test/validation set. Our attention mask helped us protect the context of our embeddings, overall resulting in a much accurate understanding of what we saying by our model. In conclusion, we did a lot less work than if we had to implement this from scratch.

5.1 References

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

L11 Language Models - Alec Radford (OpenAI)

Huggingface Transformers

Comprehensive tutorial on sentiment classification