Using BERT with SoftMax Regression and Logistic Regression to Categorize and Predict User Sentiment Towards Video Games

Zachary Watkins
Dept of Computer Science/Dept of Physics
Georgia State University
Atlanta, United States of America
Zwatkins1@student.gsu.edu

Ryan Peebles
Dept of Computer Science
Georgia State University
Atlanta, United States of America
Rpeebles1@student.gsu.edu

Abstract- *With its use of Transformer network architecture along with its huge training database, the Bidirectional Encoder Representations from Transformers, or BERT, has been put to great effect in many text-based classification applications. In typical scenarios, individuals using BERT or other natural language models would implement Transfer Learning to adapt the model to their needs. Here, we show that even the base pre-training of the BERT model is sufficient for binary and multiclass classification problems. As well as being sufficient in filtering and learning different features such as human noise.*

I. Introduction

Using a pretrained language model for classification has been common in the field of machine learning since its introduction in the early 2000s. Our goal with this application is to compare how well a model can pick up on human sentiment through both professional reviews and non-professional reviews, as well as demonstrate the effectiveness of learning despite the presence of noise. We chose to do this with the Bidirectional Encoding Representations for Transformers (BERT) along with SoftMax and logistic regression.

Background**:** As avid video game enjoyers, we have an in interest how the community perceives our favorite games. Many people leave reviews on several internet discussion boards and forums. These reviews can come from both Unprofessional journalists, i.e. average users, and professional journalist.

As humans, we tend to inject pop culture references, slang and other non-standard language features into our writings when writing casually. We will refer to this as Human Noise. When reading reviews for games on the popular video game marketplace *Steam,* a large portion of the user left reviews will contain human noise. On the other hand, profession reviews are usually grammatically correct and uses language that is straight forward and 'simple'. We as humans are typically able to recognize human noise and adjust our understandings, since we are exposed to it our entire lives. Even still we often fall victim to unexpected language tropes. Considering language models are mostly trained on professional writings such as books, articles, etc., is it able understand despite interference.

Objectives: Our object of this research is to see how well a pre-trained BERT model deals with a variety of uncommon features of text from online video game reviews containing professional text and unprofessional text. Comparing how accurately the model can categorize professional versus unprofessional language to gauge how well the model can understand human noise.

Contributions: The contribution of each team member were as follows:

Zachery Watkins:

- Lead Designer
- Lead Programmer
- Data collection
- Writer

Ryan Peebles:

- Writer
- Code Optimization
- Research Collection

For more information see section V.

II. Methods and Implementation

Prerequisites: In order to successfully recreate the implementation of this paper, you need the following:

- Knowledge and understanding of the Python coding language.

- Knowledge and understanding of common Machine Learning libraries such as PyTorch, TensorFlow, Transformers from Hugging Face.
- Knowledge and understanding of Data pre-processing and manipulation.
- GPU for computation time and efficiency.

Datasets: Two separate datasets are used in this study. The first dataset is a set of about 6 million player created reviews for several games across a multitude of genres. This set comes from the online video game marketplace known as Steam. Steam's review system allows anyone who owns a copy of a video game to review that game. Players can rate games as either "Recommended"(1) or "Not Recommended"(0), i.e. a binary classification. Along with a rating, players may also leave a short text review. As Steam reviews can be viewed as an open forum, many of the reviews left by players contain text or phrases that wouldn't be typical of a properly written body. Things such as sarcastic remarks, ironic statements, unrelated discourse, emojis, and community specific references that may not be found in the training set of a Natural language model. Sometimes this noise can imply to those reading to expect a certain rating from the reviewer only to be surprised with the opposite. For example, in a review for the popular game *'Counter-Strike',* the text says "Ruined my Life". A statement like this would typically imply a negative connotation towards the product, but to our surprise we find that the real review is "Recommended".

In contrast to the Steam dataset, we have the Metacritic dataset. Metacritic is a professional video game review publication/database. Pulled from reputable sources, Reviews found on Metacritic are often long formatted and contain straight forward typical correct English. The Metacritic dataset contains over 320 thousand reviews, also across several games and genres. Each game is given a decimal score ranging from 0 – 100. For simplification, we convert these ratings to a 0-9 scale with 0-9 = 0, 10-19 = 1, 21-29 = 2, 30-39 = 3, 40-49 = 4, 50-59 = 5, 60-69 = 6, 70-79 = 7, 80-89 = 9, 90 -100 = 9. This arrangement makes this dataset a multiclass classification problem.

Both Steam and Metacritic datasets contain information such as the game title or ID numbers. This information is not used in the model and as such is dropped from the datasets during pre-processing. Since we desire to compare the performance of BERT between the two datasets, we attempt to make the two more comparable by also converting the multiclass labels of the Metacritic review set into a binary classification. We chose to threshold the data as a score greater than 7(70-79) is considered "Recommended" or 1. This more accurately reflects the distribution of reviews found in the Steam dataset. We also chose to take a comparable sized sample of data, about 10%, from the Steam dataset when training, so as to have a better comparison and to ease the computational requirements.

BERT: BERT was created in 2018 by researchers at Google and was designed to handle sequential data such as text. BERT was built upon Transformer architecture, consisting of 12 sequential layers with 12 attention heads per layer. In each layer, each attention head assigns an attention score to each part of the input text.

BERT pretrains to two main pre-training tasks in mind. The first task is mass language modeling, and the other is next sentence prediction (NSP). In mass language modeling, a small random sample of words in the input data is selected and replaced with a [MASK] token. The goal of MLM (masked language modeling) is to predict which word goes into the masked spot given the context of the words around it. Like MLN, Next sentence prediction attempts to teach the model to make accurate predictions of what sentence would come after a given sentence sequentially. Both training tasks allow the model to obtain information about a word's positional dependencies on other words. For example, "I'm going to bed because I am [MASK]", might predict: "I am going to bed because I am tired.". In NLP, the input might be "I go to bed", and the output prediction for the next line would be "I am tired".

BERT then will perform tokenization and embedding from learned from its large corpus of pre-training data, often time using WordPiece. These embeddings take words and break them down into sub words or units and improve generalization, BERT does this by using a fixed size plethora of tokens in which each pertains to a unique word embedding. To continue, BERT can also include special tokens like [CLS] (classification) and [SEP] (separator). Special tokens have unique embeddings and are there to provide additional context to the model, BERT does this by adding special tokens to the input sequence to begin specialized tasks like text classification as demonstrated in our data research.

After this step, BERT sends the embedded data through each of its twelve attention masking heads. Each head applies an attention mask to a separate section of data in which the data mask helps the model know which words may have more influence or weight towards the prediction. For example, words that are nouns or verbs might have a higher attention mask value than words such as articles like "the" or "so". This masking is performed in parallel to maintain efficiency.

When this is completed, the data is sent into a feed forward network which implements two linear transformations, followed by a ReLU activation function. Upon completing each layer, the model reapplies its self-attention mask on the output of the previous layer

to learn better which words hold higher weight throughout the sentence. This use of transformer architecture, with masks and embedding, effectively trains BERT to successfully be used for several natural language processing tasks such as those demonstrated within this paper.

WordPiece: WordPiece is a tokenization process here words are broken down into parts. For example, "talking" might be tokenized into 'talk', '##ing'. In WordPiece, grammatical features such as prefixes and suffixes are separated from the root of the word. This allows the models to generalize vocabulary. In the example we've given, the model is able to know that 'talking' generally means the same as 'talk'.

Logistic Regression: Logistic Regression is used in binary classification to calculate the probability that an input feature belongs to a specific output category. This is also known as the sigmoid function.

$$P(y = Y|x) = \frac{1}{1 + e^{-z}}$$

- z represents the linear combination of input features

$$z = \sum \beta_n x_n$$

- β is the parameters learned during training via back propagation
- x is the input feature

SoftMax Regression: SoftMax Regression is used for multiclass classification. SoftMax computes the raw logit score for the probability each input feature belongs to specific class. SoftMax then divides each score by the sum of all scores to convert to probability and add up to 1.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_i}}$$

- z represents the raw logit score for each class i.

ReLU: Rectified Linear Unit is a non-linear activation function commonly found in feed forward networks. ReLU's purpose is to introduce non-linearity to a network to encourage learning.

$$ReLU(z) = \max(0, z)$$

ReLU assigns all values below 0 to 0, and all other values remain the same. 0 values aren't activated for the next layer and as such helps with overfitting.

Implementation: First, to implement BERT for classification, we need to pre-process our data to fit the model. As mentioned prior, we do 3 separate training runs to compare. Each run requires slight modifications to our datasets. For the first iteration of our research, we test the accuracy of BERT in context of a binary classification using the Steam dataset. For the second, we test the accuracy of the multiclass classification of the Metacritic dataset. And thirdly, we retest the Metacritic dataset, but this time converted to a binary classification, threshold in a way to which scores greater than and including 7 are considered a 1 and below 7 is considered a 0. We compare the last run directly to the results of our first Steam dataset test.

For Pre-processing our Steam dataset, we use the Pandas python library to read the data from a CSV file. We drop any data entries that contain empty or NaN review inputs, while also converting them to String data type. We also removed reviews that contained only special characters. Random sampling was used to properly represent the population, this however led to a majority positive reviews due to how people on Steam tend to vote. We sampled a portion of the dataset and split it into training and validation sets, with an 80-20 split using ScikKit-Learn.

Preprocessing for the Metacritic Dataset is done in a similar way. We still drop empty inputs and convert to strings, but here, as mentioned previously, we map the scores to a 0-9 scale. Panda's apply function is used along with our defined mapping function to accomplish this. Later for our final comparison run, we will re-process this as already stated to form binary classification.

Access to the pre-trained BERT model as well as the BERT tokenizer is obtainable through the Transformers library supplied from Hugging Face. After initializing a BERT tokenizer, we send out data into it in parallel batches for efficiency. Once the data has been encoded, we save it to memory so we can skip this step on sequential runs. We can now use a custom pyTorch dataset class to construct datasets for out model. The dataset consists of encoded text reviews and a list of scores as labels.

Grabbing our BERT model from Transformers, we can prepare special Trainer class provided from Transformers to pass our datasets into the model for training. The trainer class takes subclass of training arguments where we pass our hyperparmeters.

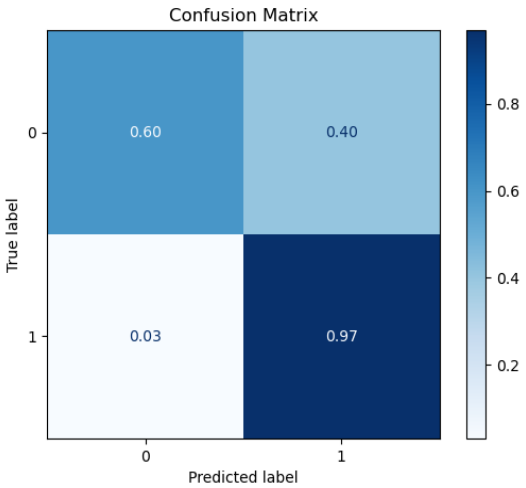
HyperParametes:

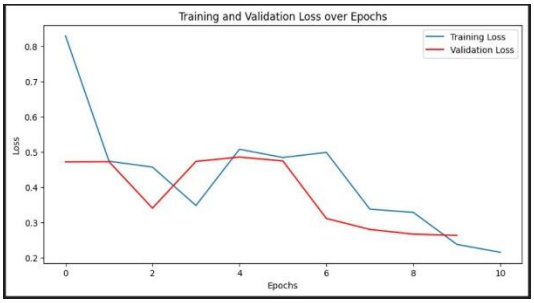
- Epochs =10
- Batch size = 64
- Weight decay = 0.01
- Learning Rate = 0.00005
- Warmup steps = 500

The parameters are kept the same for each run of our test for consistency of what we are measuring. After each training, the model is saved as a checkpoint, allowing us to easily re-access the model and use it to classify new inputs for validation. Training on the model for multiclassification is the same process, just with a SoftMax Regression as opposed to the Logistic Regression.

III. Results

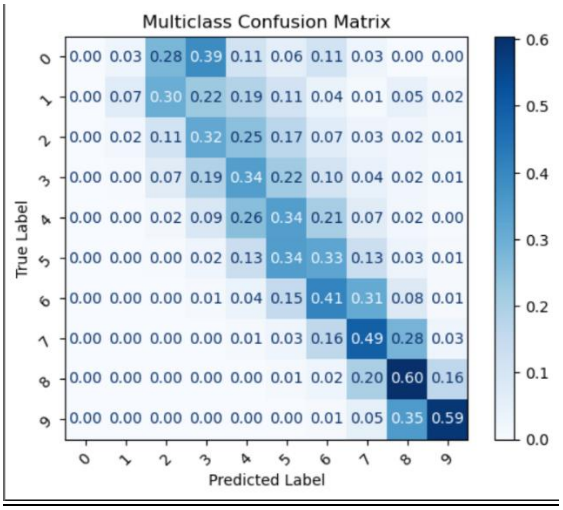
Binary Classification of Steam Reviews:

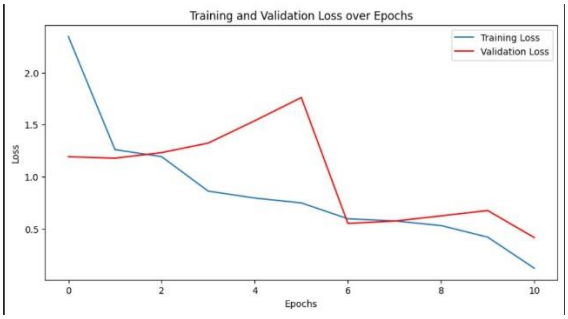




The result for binary classification of Steam Reviews shows promising results for classifying positive reviews with 97% accuracy but seems to struggle more with negative reviews with only 60% accuracy. As seen in the multiclass Metacritic run, when the review can be easily mislabeled due to the differences in categories being too small, this may lead to overfitting in training for negative revies.

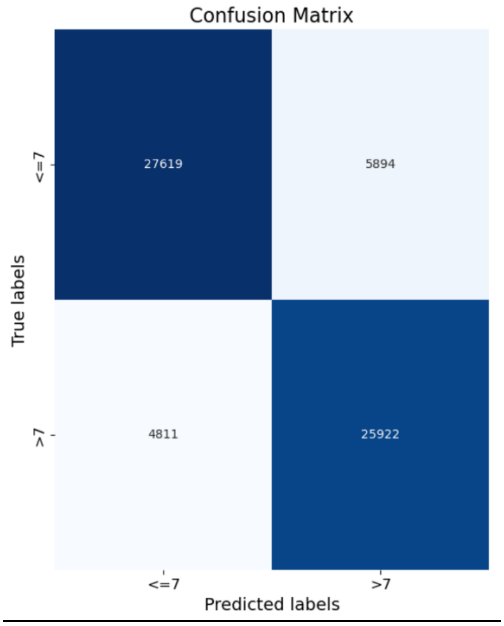
Multiclass classification of Metacritic Reviews:





The results of the multiclassification for Metacritic are less impressive. The model is still able to mostly predict correct labels, it struggles due to lack of difference between each category. For example, an entry with a score of 6 may have been scored as a 7 or a 5. The categories are too close to each-other, leading to incorrect correct labels.

Binary Classification of Metacritic:



Results after converting the multiclass in binary of the Metacritic reviews was able to correctly predict positive labels with about 84% accuracy and negative labels with about 82% accuracy. By comparison to the Steam binary classification, the model performed better here without the presence of human noise.

IV. Summary

BERT and other natural language models are extremely useful for many NLP applications such as classification. Using Tokenization and Encoding, the BERT model can be trained to classify both binary and multiclass classifications. Even without fine-tuning, the model is able to classify data assuming the data is appropriately distributed. It is shown that the model performs better on datasets that contain content more similar to that found in its pre-training corpus. When we introduce noise such as that found in the Steam dataset, the model is still able to accurately categorize.

BERT struggles the most when the input data is too similar to one another. As seen in the multiclass Metacritic run, when the review can be easily mislabeled due to the differences in categories being too small. When we switched to Binary Metacritic, the accuracy of the model was much higher. Less classes s to less confusion for BERT. With more appropriate fine-tuning and better data pre-processing, BERT will be able to accurately categorize reviews with or without the presence of human noise to a sufficient degree.

V. Teamwork

For this project, Zachary Watkins took on much of the responsibilities. He orchestrated and planned what our research project would encompass. He researched and chose which model to use as well as sourced our datasets. Zachary was also responsible of the proposal and update papers for our research. Zach also handled implementation of our model in Python.

Ryan took on much more assistive role as opposed to leading. Due to outside responsibilities, Ryan had to delegate much less time to this research than hoped for. Ryan took on a role in testing the code and implementing some optimization. Ryan also was responsible for a portion of the results analysis and for gathering our information for the final presentations and paper. HE is the main author of the final paper that was the sum of Zachs work.

VI. References

[1] “Transformers.” , huggingface.co/docs/transformers/en/index. Accessed 22 Apr. 2024.

[2] “Bert 101 - State of the Art NLP Model Explained.” BERT 101 - State Of The Art NLP Model Explained, huggingface.co/blog/bert-101. Accessed 22 Apr. 2024.

[3] “Tokenizers.” Hugging Face – The AI Community Building the Future., huggingface.co/docs/tokenizers/en/index. Accessed 22 Apr. 2024.

Link To Project:

https://drive.google.com/drive/folders/1kK83X9K8yIJ9aJIm03ppc9XKJBmIgqIV?usp=sharing