AMAZON RATING PREDICTION MODEL

-By Anjali Yadav

1. Methodology (Steps) used

Data Collection: The first step in building a multiclass classification model for Amazon ratings is to collect a data set of audio files. This is done by using a pre-existing dataset. It is important to ensure that the dataset is representative of the problem you are trying to solve and that there is enough data to build a robust model.

Data preprocessing: Once you have a dataset of reviews, the next step is to preprocess the data to extract useful features that can be used as input to our ensemble model. This usually involves removal of punctuation and special characters, lower-casing, lemmatization and tokenization.

Data Splitting: After preprocessing, the output feature is encoded using ordinal encoding because there is a hierarchical relationship between our output classes. This data is then divided into dependent and independent features. These are then split into Train and test set using stratified method.

Modeling: After the data is preprocessed and augmented, the next step is to create the BERT model called “bert-base-uncased”. Since we have already obtained a feature matrix, we will be directly feeding the Feature matrix into the BERT model.

Model Training: After building the model, the next step is to train it on the preprocessed data using appropriate loss functions and optimization algorithms. Common loss functions for binary classification include **Sparse categorical cross-entropy** and categorical cross-entropy. I have sparse categorical cross entropy for my model. The optimization algorithm used to train the model can be selected from a range of options, such as Stochastic Gradient Descent (SGD) or **Adam**.

Model evaluation: Once the model is trained, the next step is to evaluate its performance on the held validation set. It involves calculating metrics such as precision, accuracy, recall, and F1 score to determine the model's ability to classify review and predict the corresponding rating. The model can be further evaluated on the test set to assess its generalization performance.

2. Packages used

a. **Pandas:**
   It is a python library used for manipulating datasets. It helps us perform various operations on these datasets like data cleaning, transformation, and analysis. In this project, we used pandas to create a DataFrame containing waveform data of our audio signals.
b. **Matplotlib:**
   This library is used for plotting and visualization. We have used Matplotlib to create Histograms to check whether our dataset is balanced or not.
c. **Numpy:**
   It is a python library for numerical purposes and allows us to manipulate and access arrays and matrices. Some functions we used in our array are np.array(), np.mean(), etc.

d. **Scikit-learn:**
   It helps to build machine learning models, including algorithms for classification, regression, clustering, and dimensionality reduction, as well as tools for data preprocessing, model selection, and evaluation. We used this library for various functions. We used 'LabelEncoder' class for encoding our output classes into 0 and 1. We also used 'train_test_split' to split our training data into train set and validation set.
e. **TensorFlow:**
   TensorFlow is a low-level framework for building and training machine learning models, developed by Google. It provides a wide range of tools and functions for building and training different types of neural networks, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep neural networks (DNNs). In this project, we use it for CNN model.
f. **Keras:**
   Keras provides a wide range of building blocks for building neural networks, such as layers, activations, loss functions, and optimizers, which can be combined to build complex models.

3. Data Pre-Processing implemented

Loading the data:

Loading audio data is the first step in the data preprocessing pipeline. I loaded the data from the Amazon Reviews dataset and created a DataFrame for just two columns namely: "reviews_text" and "reviews_rating".

Lower casing:

Lower casing helps to standardize the data by converting it into a consistent form. This makes processing faster and improves accuracy.

Punctuation Removal:

This is also a step-in standardization of text data. It also helps to reduce the noise in the data, hence improving accuracy.

Stop Word Removal:

This method not only reduces noise, but also allows our model to focus only on the meaningful words hence improving efficiency.

Lemmatization:

This converts the words in our text data to a base form making it easier to process and analyze. It also reduces noise my converting similar form of words to a common base word which further reduces the redundancy while tokenizing. Since time and storage isn't a constraint, we will be using lemmatization instead of stemming.

Encoding:

This is done to convert our text data into numeric format to feed into our machine learning or deep learning models. We notice that our output labels have a hierarchical relationship among them, so we will perform ordinal encoding instead of one-hot encoding.

4. Code Explanation

I have divided my code into 4 sections:

- Importing the Dataset
- Data preprocessing
- BERT
- Evaluation of the Model and Accuracy Visualization

In the first, **"Importing the dataset",** I have defined the paths for the dataset of Amazon Reviews which contains 5000 samples and 75 features. I have also imported several libraries which will be used for accessing and manipulating this dataset.

Then, in **"Data Preprocessing",** I have performed some standardization techniques such as lower-casing, punctuation removal, stop-word removal, lemmatization and encoding.

Next, we **"Split the dataset"** to train the model. Before splitting the dataset I checked if we have unbalanced dataset. As we see, the dataset is slightly unbalanced, using a Stratified method for splitting the data will be a better fit as it will divide equal parts of both class labels into the training and testing datasets.

Now, we load the necessary libraries to load our pre-trained BERT model. These libraries and classes include "TFAutoModelForSequenceClassification" and "AutoTokenizer".

For **"Building the model",** I use the above-mentioned classes. I loaded the BERT model named "bert-base-uncased". I also created an object of the AutoTokenizer class to tokenize the preprocessed text data. We then used this object to separately perform tokenization on our test and train data. We set padding = True to allow padding so that every vector is of the same length. Also, we set Truncation = True to prevent any vector from exceeding the maximum length which is set at 300. Next, we create training and testing datasets for batch learning and parallelization in training.

After modeling was done, I compiled my model. I used 'Adam' as the optimizer because it prevents being stuck in local optima and is relatively easy to use. It requires very little hypertuning. Sparse_categorical_crossentropy was chosen as our loss function because our task is multiclass classification and our dataset is highly unbalanced. It also is not very effected when we have unbalanced data. As the dataset was not balanced, we could not use Accuracy as our performance metric, and F1-score and SparseCategoricalAccuracy were the best choice for this task. I chose SparseCategoricalAccuracy as it was designed for multiclass class predictions.

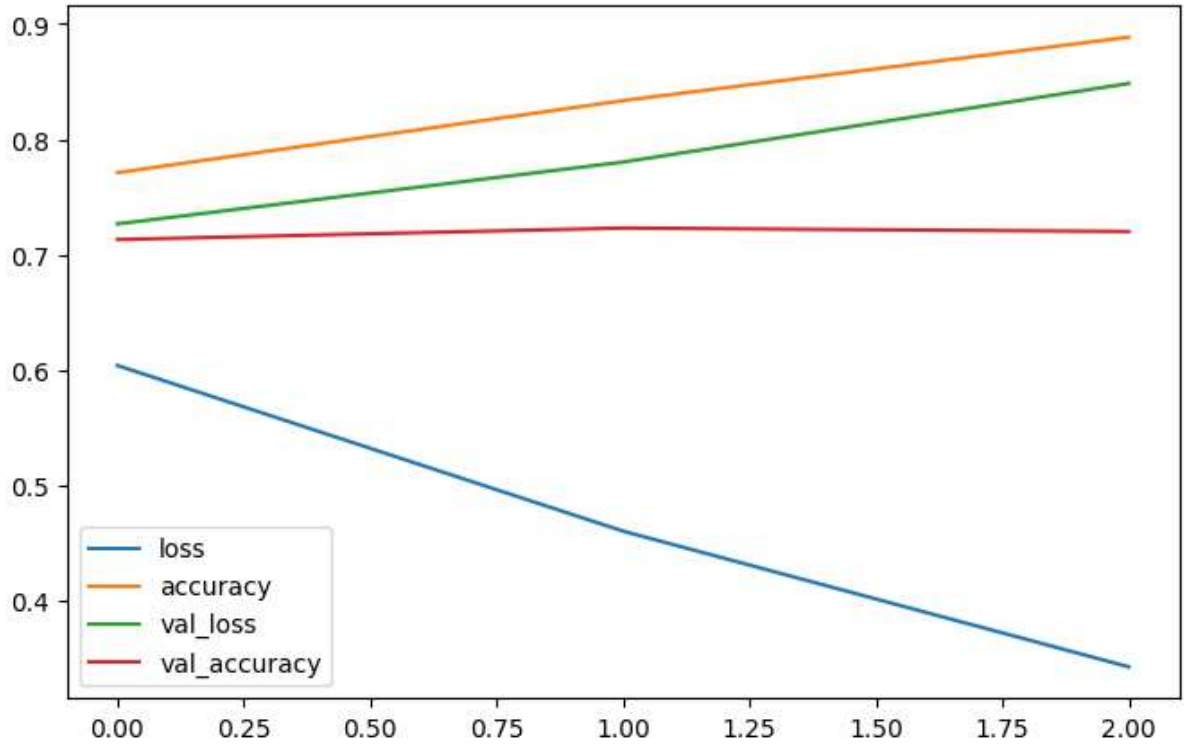
Then our model is trained on the training set and evaluated on the validation set after each epoch. The goal is to find the best set of hyperparameters that minimize the loss on the validation set while avoiding overfitting to the training set. I chose the number of epochs to be 10 as our dataset is slightly big and task at hand is quite simple.

Next, we **"Visualize and Evaluate our model".**

5. Results

We got a Training accuracy of **84.83%**

And testing Accuracy of **72%**



We find that our model is overfitting in every possible scenario. A simple CNN model to a pre-trained BERT Model, since our dataset contains a lot of imbalance and a lot of noise, it leads to overfitting every time.