Text Mining and Natural Language Processing 2022-2023

ABRC: Amazon Books Reviews Classifier

Michele Ventimiglia: 502230
Giovanni Michele Miranda: 507567

1 Introduction

Our project involves the analysis of a dataset composed of Amazon reviews for various types of books. The analysis we conduct is commonly known as SENTIMENT ANALYSIS, which focuses on training our model to predict ratings (typically referred to as "stars") based on the written reviews. This task is also known as text classification since the main goal is to assign different reviews to different classes (i.e. ratings within the range 1-5). In order to accomplish our goal, we have used different Deep Learning tools, mainly used for NLP tasks, and models which allowed us to conduct a very rewarding analysis.
On a final note, by studying the reviews and their corresponding ratings, we aimed at developing a model capable of reliably attributing ratings to new and previously unseen reviews.

2 Data

After exploring different dataset, we managed to find the best one on Kaggle (a data science platform), entitled *Amazon Book Reviews.* This dataset contains a total of 3.000.000 rows and 10 well defined columns. During EDA phase, we underlined the most interesting features for our analysis: *review/text* and *review/score*. So we extracted them from the dataset and reshaped respectively into two separate arrays named **ratings** and **reviews**, in order to proceed with our main goal of predicting ratings based on the text of reviews. In Figure 1 is shown the distribution of rating values per class (i.e. for each rating class 1-5 stars, the amount of reviews classified under that specific class). What actually emerges from this graph is that our reviews seems quite imbalanced, since we can clearly see an outrageous number of reviews classified as 5, which are nearly 9 times those classified as 1.

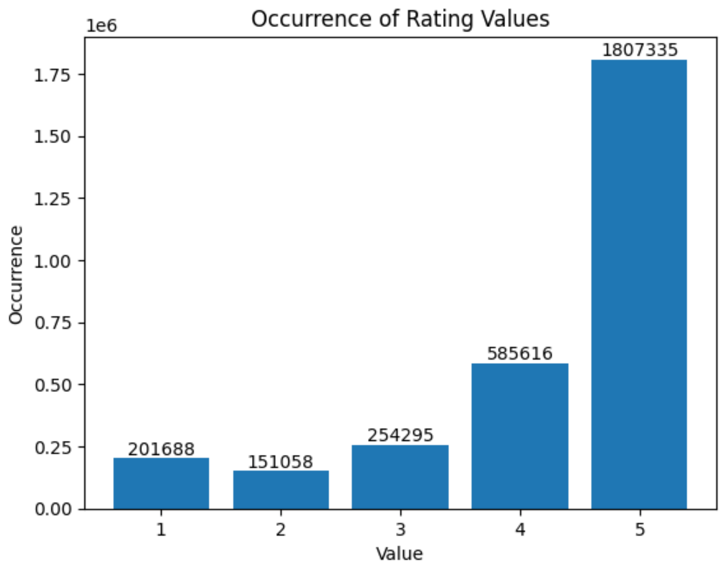


Figure 1: Occurence of Rating values.

To safeguard the model from training on heavily skewed data and to enhance its generalization ability, we have undertaken an sampling reduction approach. In the forthcoming section of this paper, we will investigate the precise definition and methodology of this technique.

3 Methodology

In this new section, we will clarify the complete process we have undertaken to develop a high-quality model, achieving a very good accuracy score. As illustrated in Figure 2, we have adhered to a well-established set of steps and methodologies. Naturally, each of these steps comprises a series of intricate sub-steps, adding complexity to the task at hand.

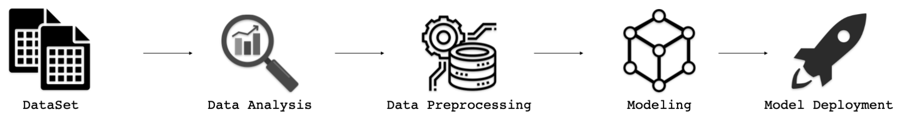


Figure 2: Model Pipeline

To gain a more comprehensive understanding of the architecture employed in our project, we can refer to Figure 3, which distinctly illustrates each operation properly categorized.

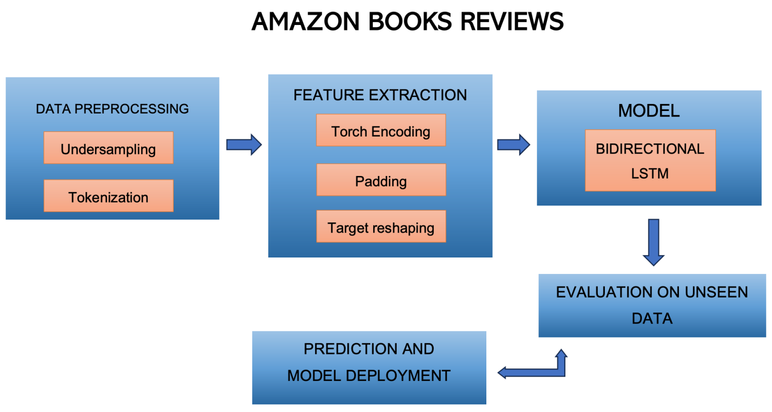
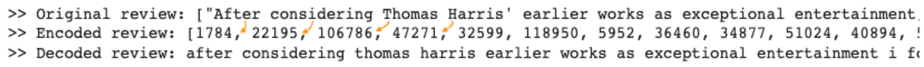


Figure 3: Model Architecture

Initially, we conducted a comprehensive analysis of the dataset, exploring its features and essential information. We identified a total of 10 feature columns, namely: ['Id', 'Title', 'Price', 'User id', 'profileName', 'review/helpfulness', 'review/score', 'review/time', 'review/summary', 'review/text']. One crucial step in addressing this type of problem involves handling **missing values**. To accomplish this, we performed data cleaning by removing null values from the pertinent columns of interest, specifically 'review/score' and 'review/text.' Then we **extracted** them into two new NumPy arrays respectively: *ratings* and *reviews*, in order to convert them in 2D arrays; this is often done to prepare the data for further processing or modeling. It is noteworthy that this initial analysis was conducted using the powerful Python library, NumPy, which facilitates array operations.
Subsequently, we moved on to the preprocessing step, where the initial action involved **class balancing** through undersampling of the arrays. This resulted in the creation of two balanced and reshaped features.
Another crucial step in NLP models is **tokenization**, which enables us to break down input textual sentences into individual units, namely words. This step holds great significance because in NLP models, we often require the processing of text at the finest level, making the treatment of individual words one of the most efficient approaches. We applied tokenization to each *review* (2D array of review/text feature). During this process, we filtered the text using an English vocabulary, ensuring that only English words were retained, thereby reducing complexity and eliminating unnecessary tokens. Afterward, we applied **case folding**, a process that converted all letters to lowercase, as clearly demonstrated in the *Decoded Review* shown in Figure 4).
To prepare for the encoding step, we chose to create an index mapping for

each word within every review. This involved mapping each word from the English Vocabulary to a unique index. Subsequently, we constructed the **Encoder Class**[1] using this indexed vocabulary as reference. We then applied this class to our tokenized corpus input, ensuring that each original input had its corresponding encoded version. For a visual representation of this process, please refer to Figure 4.



```
>> Original review: ["After considering Thomas Harris' earlier works as exceptional entertainment,
>> Encoded review: [1784, 22195, 106786, 47271, 32599, 118950, 5952, 36460, 34877, 51024, 40894, !
>> Decoded review: after considering thomas harris earlier works as exceptional entertainment i fo
```

Figure 4: Encoding Process

In order to conclude the preprocessing part, we conducted a statistical analysis of the reviews. Our goal was to establish a confidence interval (as shown in Figure 5) for the distribution of review lengths.

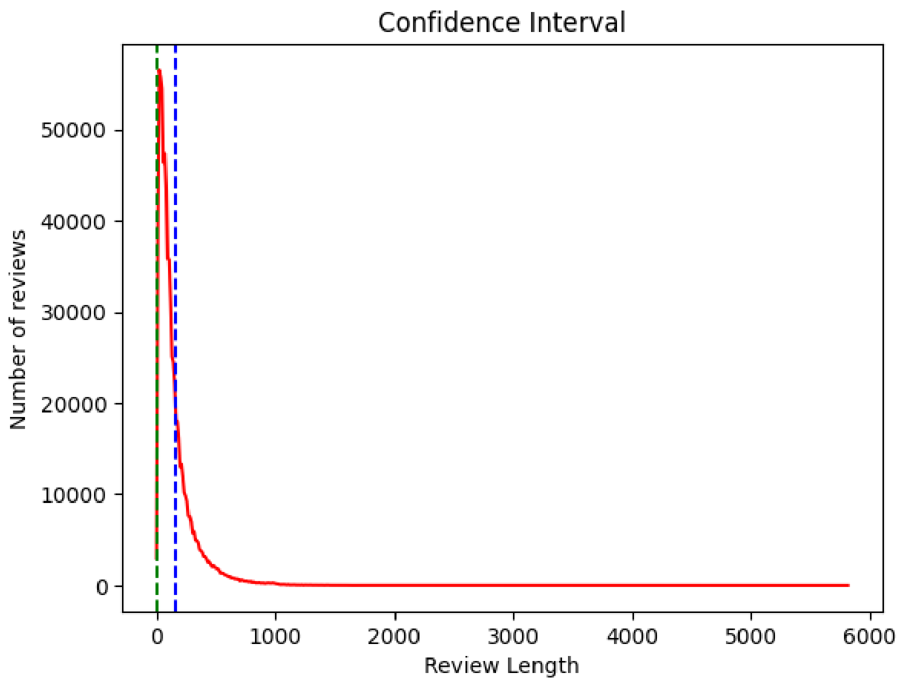


Figure 5: Confidence interval

This step was crucial because we aimed to maintain a high degree of unifor-

[1]This class has two methods: Encode and Decode. Encoding is the process of converting text into data that can be used in our models; it essentially involves mapping each input word to an encoding vector.

mity in our corpus, avoiding overly lengthy reviews that could lead to increased processing costs. To achieve this, we followed these steps:

1. Analyze the intervals and established the distribution of reviews length (in fact we noticed that 68.46% of reviews had lengths between the interval: [4, 161]); we then removed very long reviews, which results can be seen in Figure 6
2. Performed **Padding** [2] to fill shorter reviews and in order to have the same dimensionality among them.

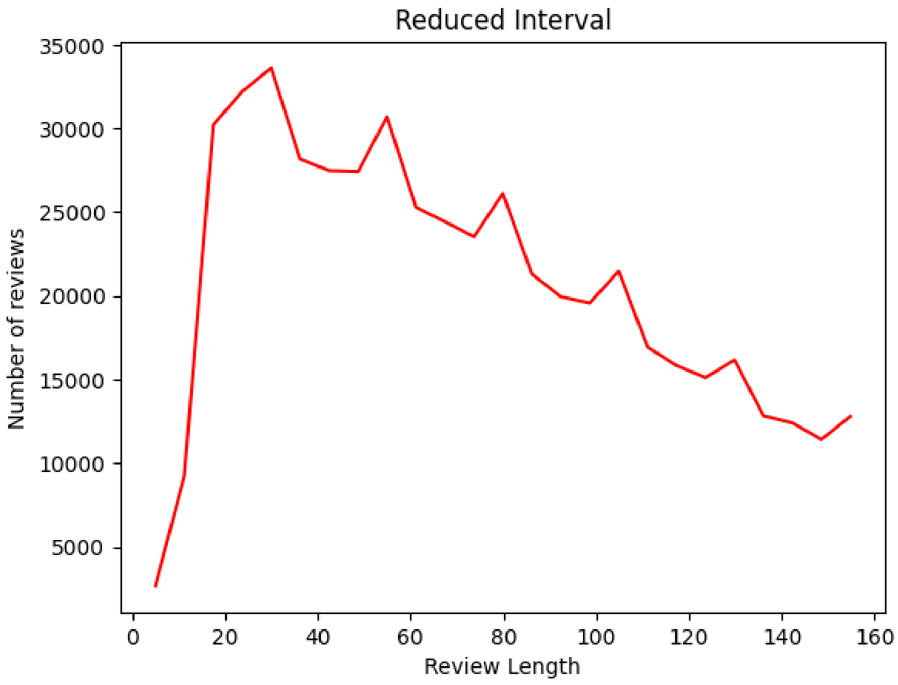


Figure 6: Reviews length after reduction

Next, we obtained our training, testing, and validation sets, preparing ourselves for the modeling phase.
Here we defined our **Model** by creating a class and by using PyTorch tools to recall pre-defined architectures; as subclass it takes a PyTorch class called Module [3], which serves as base for all neural network modules. This model

[2]Padding is a preprocessing step where sequences of text data, such as sentences or documents, are adjusted to have the same length by adding special tokens called "padding tokens" to the shorter sequences.

[3]It allows you to create complex neural network architectures by composing simpler modules into larger ones. You can define layers, subnetworks, and entire models as modules. This hierarchical composition is a key feature for building deep and structured networks.

results in a combination of different layers and techniques, each of which has its own role. In order to better understand the various element, let's dive into the model class:

1. **Embedding Layer** [4]: it uses the pre-set PyTorch class Embedding which stores embeddings of a fixed dictionary (i.e. the variable decoding-vocabulary) and size. Then you retrieves the stored word embeddings by using indices.
2. **Bi-directional Long Short-Term Memory, LSTM** [5]: it uses two parameters, embedding-dim and hidden-dim; the first one represents the dimensionality of the input embeddings, and is perhaps the same variable used in the previous layer, while hidden-dim specifies the dimensionality of the hidden states within the LSTM cells. The hidden states capture and store information from previous time steps and play a crucial role in the model's ability to learn and represent sequential patterns.
3. **Dropout Layer**: is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to zero during each forward pass; the *dropout_rate* parameter determines the fraction of units to drop during training phase.
4. **Linear Layers**: these are fully connected (dense) linear layers [6]. They introduce non-linearity into the model, enabling it to capture complex relationships between features. Within the architecture class, two distinct Linear Layers are observed, each assigned a different variable name, suggesting potential variations in their roles:
   - self.fc1: reduces the dimensionality of the output from the LSTM layer to 'hidden_dim', in order to get a suitable output for classification.
   - self.fc2: further reduces the dimensionality to the number of classes ('num_classes').

Leaky ReLU serves as the **activation function** following the fully connected layers, introducing non-linearity to the model. It plays a crucial role in mitigating the vanishing gradient problem, enabling the model to acquire the capacity to learn intricate mappings.
The **Class Model** has also a method called **forward**, which specifies how the input data should flow through the network and defines the forward pass of the

[4]In this case we decided to use two parameters: vocab-size and embedding-dim. Both are integer parameters that respectively indicate size of the dictionary of embeddings and size of each embedding vector.

[5]Bidirectional LSTM is an extension of the traditional LSTM architecture, which is a type of RNN; it processes input sequences in both forward and backward directions, allowing it to capture information from past and future time steps simultaneously. This bidirectional processing helps improve the model's ability to understand and represent context in the input data.

[6]A fully connected layer refers to a neural network in which each input node is connected to each output node.

model, that computes predictions based on the input data.

After explaining all the characteristics of our architecture, we want to underline the hyperparameters used with respect to the variables named before:

- *embedding_dim* = 256.
  This choice is a common one in NLP hyperparameters tuning, that here followed practical and empirical decision.

- *hidden_dim* = 64.
  It specifies how many hidden units or neurons the LSTM layer should have. To prevent overfitting, it has been shown that 64 is commonly a good number.

- *dropout_rate* = 0.5.
  It is a relatively strong regularization and is commonly used when working with small datasets or deep network.

- *learning_rate* = 10^{-4}.
  This is still a design choice, and such a number allows us to control overfitting and to stabilize the training process.

- *batch_size* = 64.
  It is a canonical choice in such NLP tasks, and we found out it suitable in our case too.

This sequence of steps culminates in the **model compilation**, which serves as the final setup phase before actual training begins. During this step, we define the **loss function** [7] as ***Cross Entropy***, a commonly employed loss function in multi-classification tasks. In this phase, we have also specified the **optimizer**, which is the optimization algorithm employed to update the model's weights during training. For this purpose, we opted for ***Adam***, a widely adopted and favored choice owing to its adaptive learning rate characteristics. As part of our design decision, we additionally introduced a custom learning rate scheduler named ***LinearLR***. This choice was motivated by our desire to closely monitor and track the learning rate adjustments throughout the training process, enabling us to gain deeper insights into and exert finer control over the model's behavior.
After configuring all these hyperparameters, we proceeded to the actual training phase. We loaded the dataset using PyTorch tools to maintain a clean, smooth, and easily accessible workflow. Additionally, we developed an **early stopping class** to continuously monitor the average validation loss. The purpose of this class is to halt the training process when this parameter ceases to improve for

[7] A loss function is a crucial component in training machine learning models. Its operativeness lies in its ability to quantify the disparity between the model's predictions and the actual target values. By minimizing this during training, the model adapts its parameters to approximate the underlying patterns in the data, ultimately leading to improved performance and accurate predictions

a specified number of consecutive epochs. This implementation is valuable as it prevents overfitting and saves computational time.
The training phase is then over, and in the next session we are going to analyze the forthcoming results.

4 Result

To assess the model's validity, we plotted training and validation metrics, enabling us to make observations about the model's effectiveness.
The first thing we have analyzed is the **Accuracy** [8], which is a metric used to evaluate the performance of a classification model. It represents the proportion of correctly predicted samples out of the total number of instances in the dataset.

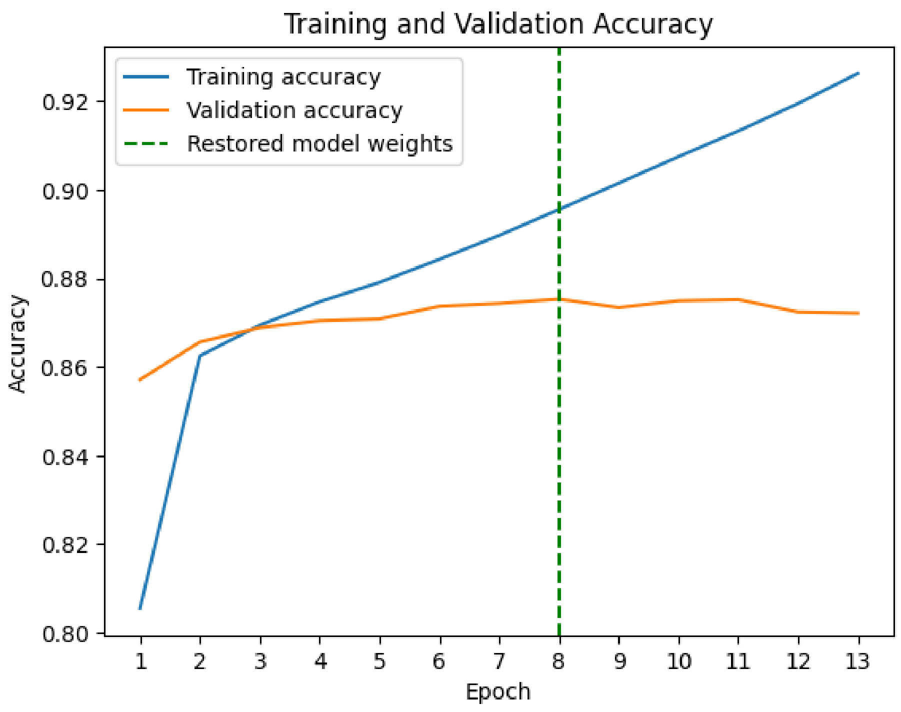


Figure 7: Accuracy metric

The training and validation accuracy plot indicates that the model's **generalization** is robust. We observe stable and satisfactory results in both the training

[8]It is calculated using the following formula: Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)

and validation phases, particularly up to *epoch 8*[9]. It is evident from the graph that we applied early stopping at this point to prevent overfitting, a decision well-supported by the plot.
Then we plotted also the Loss (Figure 8), which is a measure used to quantify the difference between the predicted values (output) and the actual target values (actual truth). These two metrics (*Accuracy* and *Loss*) provide a numerical representation of how well or poorly a machine learning model is performing in terms of its predictive accuracy.

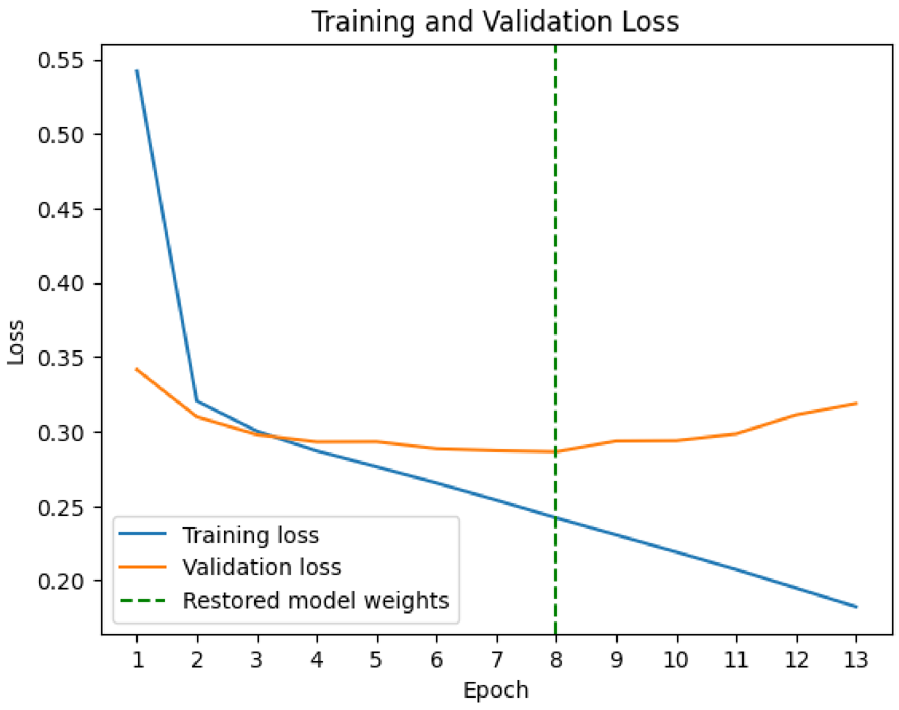


Figure 8: Loss metric

From the plot, we can observe a significant decrease in the loss terms during the first 8 epochs. This decrease is clearly visible in both the training and validation loss. However, after epoch 8, we notice that the model starts to exhibit signs of overfitting. This is evident as the training loss sharply decreases while the validation loss begins to increase. Nevertheless, it is still reasonable to assert that our model performed well from this perspective.
After taking all these factors into consideration, we conducted tests on previously unseen data (i.e. reviews). These tests provided us with actual results, offering a clearer visualization of our model's objectives, as clearly shown in

[9]After the training phase, we employed **early stopping** to determine which model, at which epoch, would have been the most suitable choice to select as the best possible model.

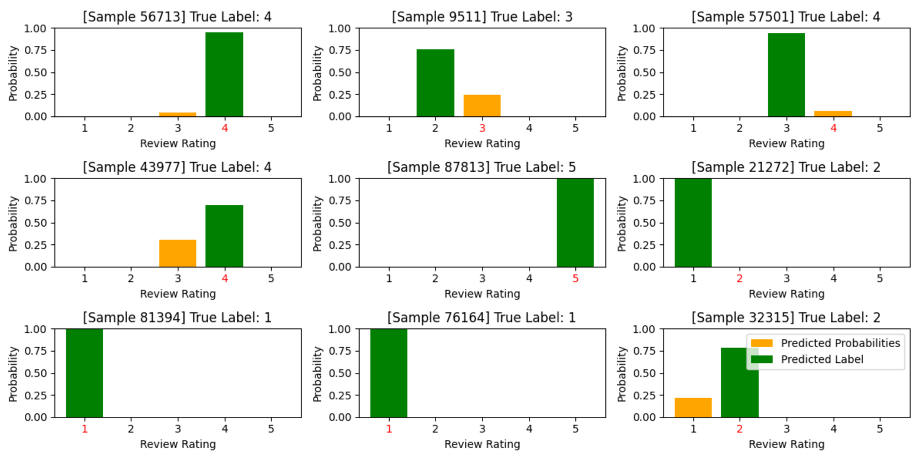
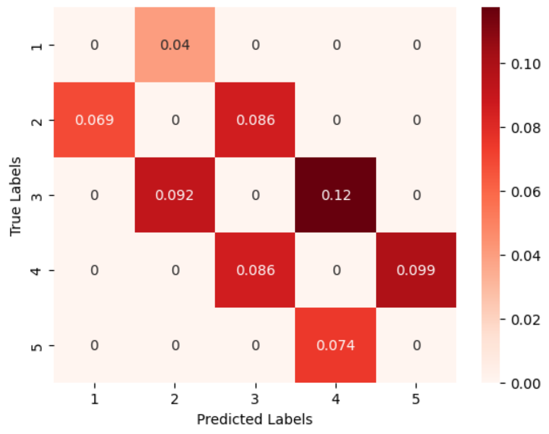


Figure 9: Model Prediction Test

Figure 9.
At the conclusion of this phase of our work, we made the decision to create a **Confusion Matrix**, as depicted in Figure 10, in order to assess the rate of misclassification errors. This allowed us to compare our model's predictions with the actual truth values.



	1	2	3	4	5
1	0	0.04	0	0	0
2	0.069	0	0.086	0	0
3	0	0.092	0	0.12	0
4	0	0	0.086	0	0.099
5	0	0	0	0.074	0

Figure 10: Confusion Matrix

5 Conclusion

As the concluding section of this report, we aim to provide a summary of all the work conducted so far. As mentioned briefly in the introduction, our project involved the analysis of a dataset consisting of Amazon book reviews, with the objective of classifying them into five distinct rating categories. This problem falls within the realm of multi-class classification and finds applications across diverse domains, extending well beyond its specific task. The primary challenges we encountered revolved largely around selecting optimal hyperparameters, which can be quite challenging. To address this, we conducted research on common choices made by other experts in the field for similar tasks and proceeded to manually configure them.

Another important decision pertained to the model's architecture, leading us to opt for a bidirectional LSTM. This choice yielded noticeable improvements in the model's performance compared to the standard LSTM. These decisions were strongly supported by the demonstrated results, prompting us to prioritize simplicity and effectiveness throughout the process.

In conclusion, it is essential to recognize that achieving a higher validation score is no trivial at all. As demonstrated by the confusion matrix (see Figure 10 for reference), our model primarily misclassifies within a single rating class, leading to errors that are somewhat understandable. The performance of our model has been inevitably damaged by the fact that our dataset contains reviews that were predominantly misclassified by the human reviewers themselves (assigning a five-star rating to a review more fitting for a one-star rating), making this task exceptionally challenging for our model. Taking all these factors into account and considering the substantial effort we dedicated to this project, we are pleased with the results presented in the preceding sections.