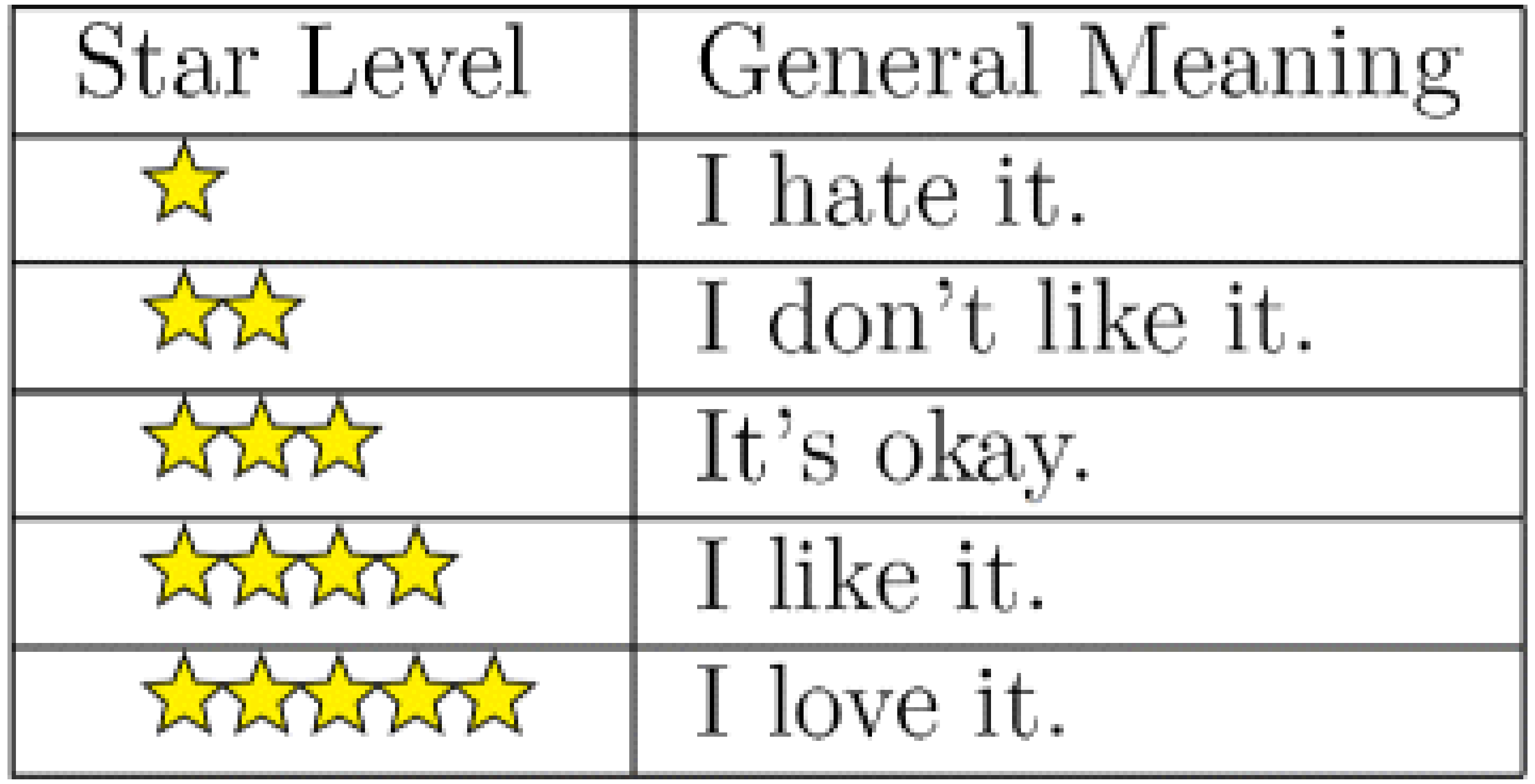
ABRC:
Amazon Books Reviews Classifier

Text Mining and
Natural Language Processing

Michele Ventimiglia: 502230
Giovanni Michele Miranda: 507567

Introduction

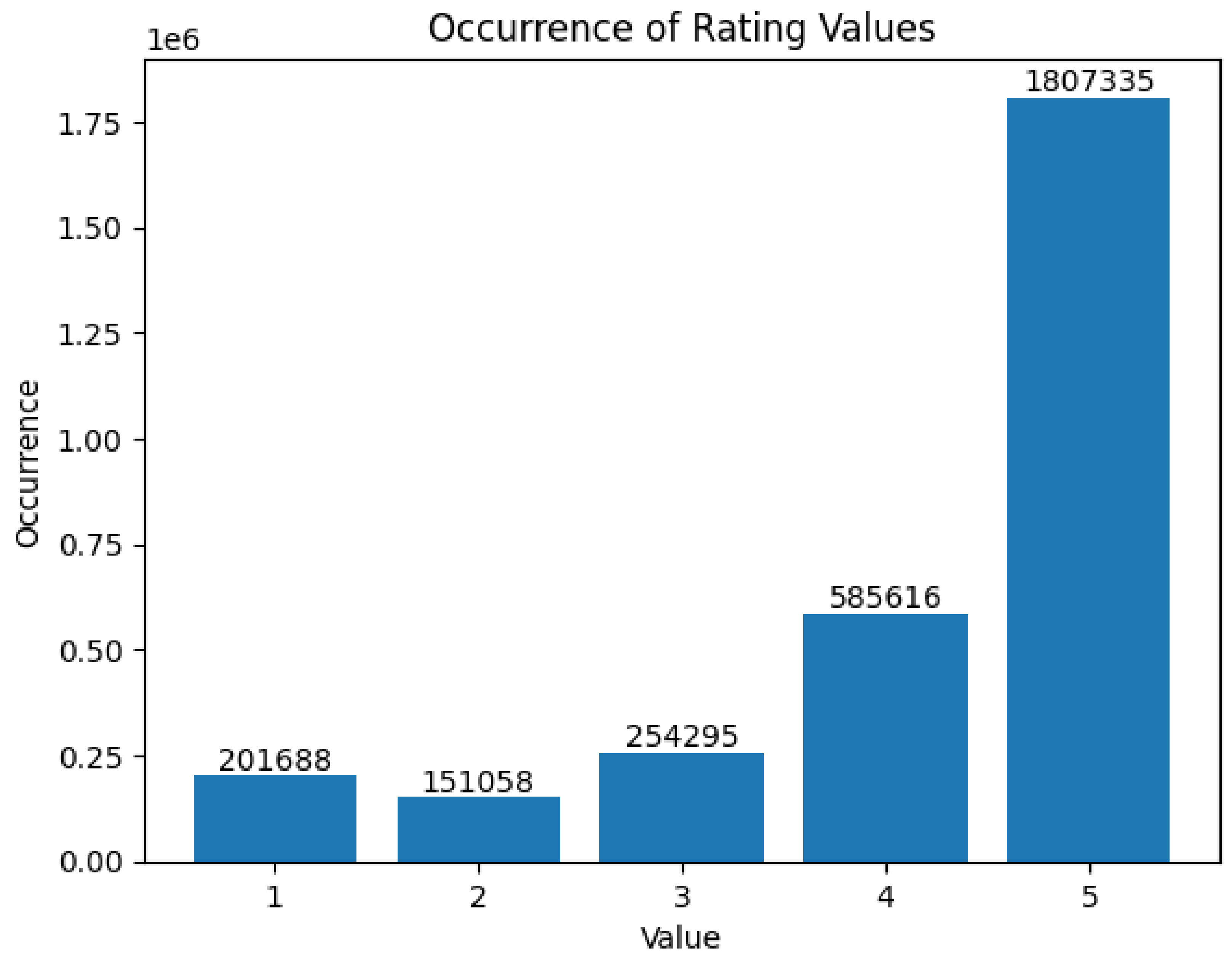
Our project involves the analysis of a dataset composed of Amazon reviews for various types of books. The analysis we conduct is commonly known as sentiment analysis, which focuses on training our model to predict ratings (typically referred to as ”stars”) based on the written reviews. The main **goal** is to assign the reviews to **ratings within a range 1-5**.
In order to do so, have used different Deep Learning tools, mainly used for NLP tasks, and models which allowed us to conduct a very rewarding analysis.



Star Level	General Meaning
★	I hate it.
★★	I don’t like it.
★★★	It’s okay.
★★★★	I like it.
★★★★★	I love it.

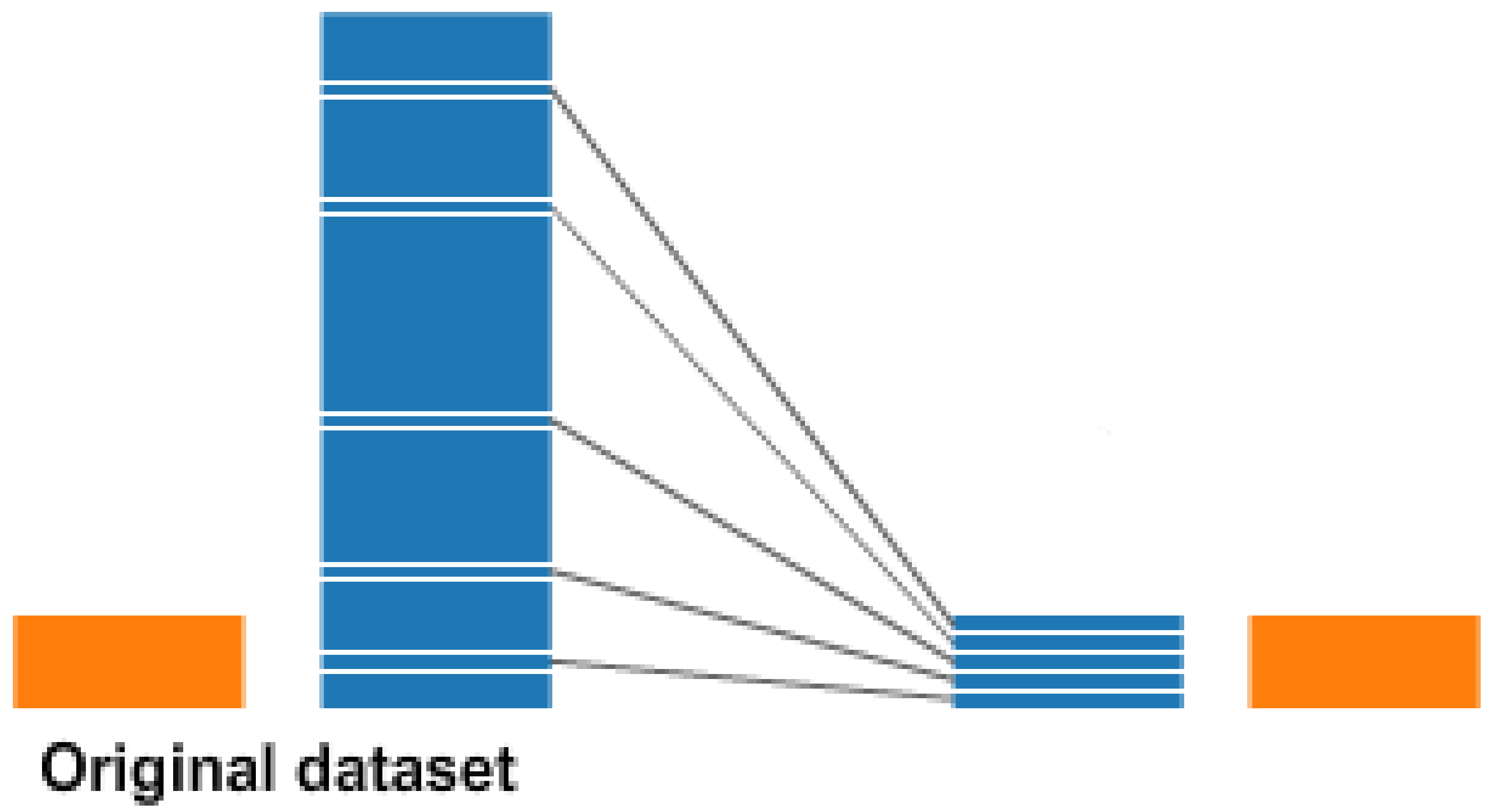
Data

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.
What actually emerges from the graph on the right is that the reviews seem quite imbalanced, since we can clearly see an outrageous number of reviews classified as 5, which are nearly 9 times those classified as 1.



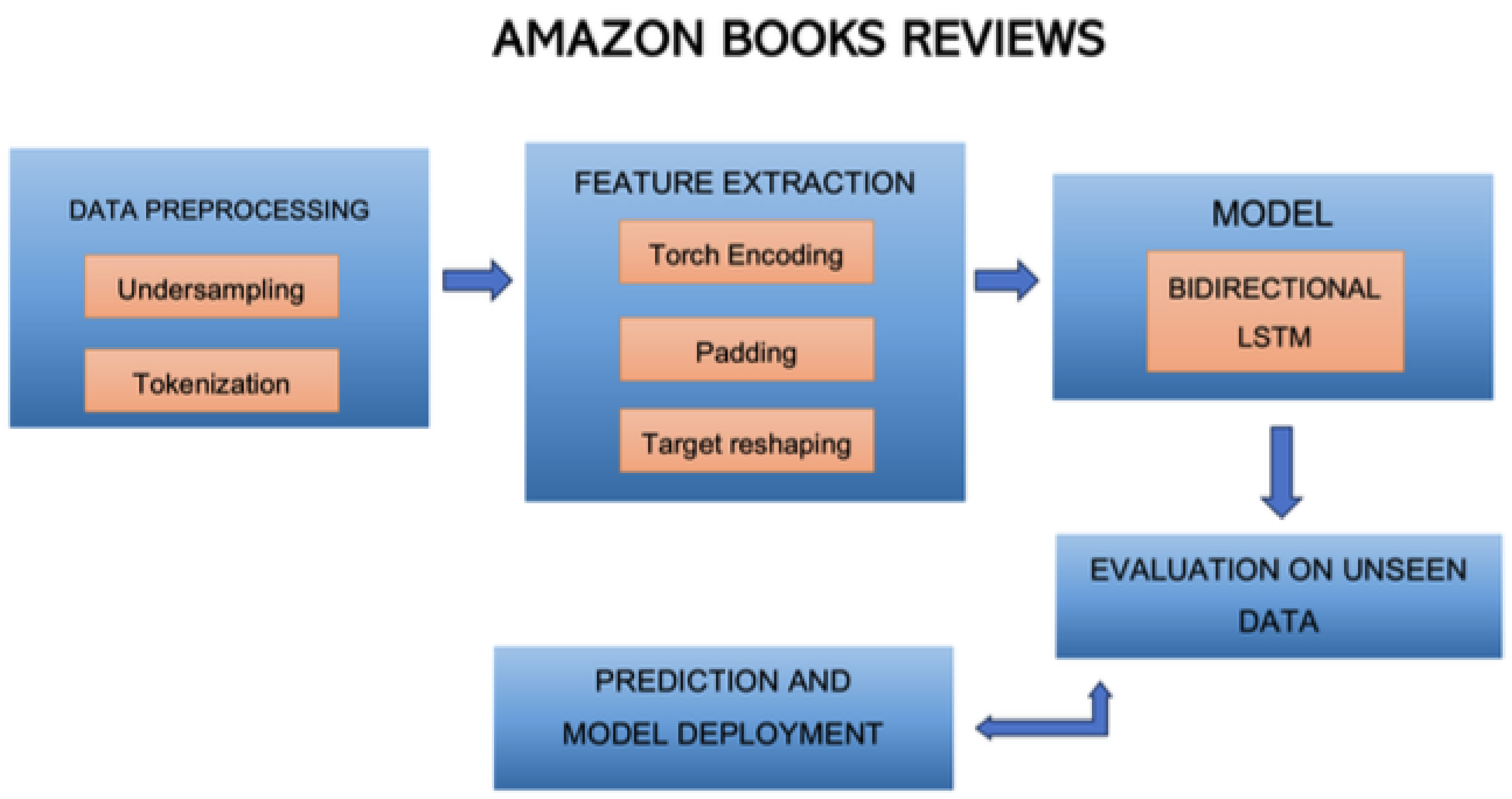
Data

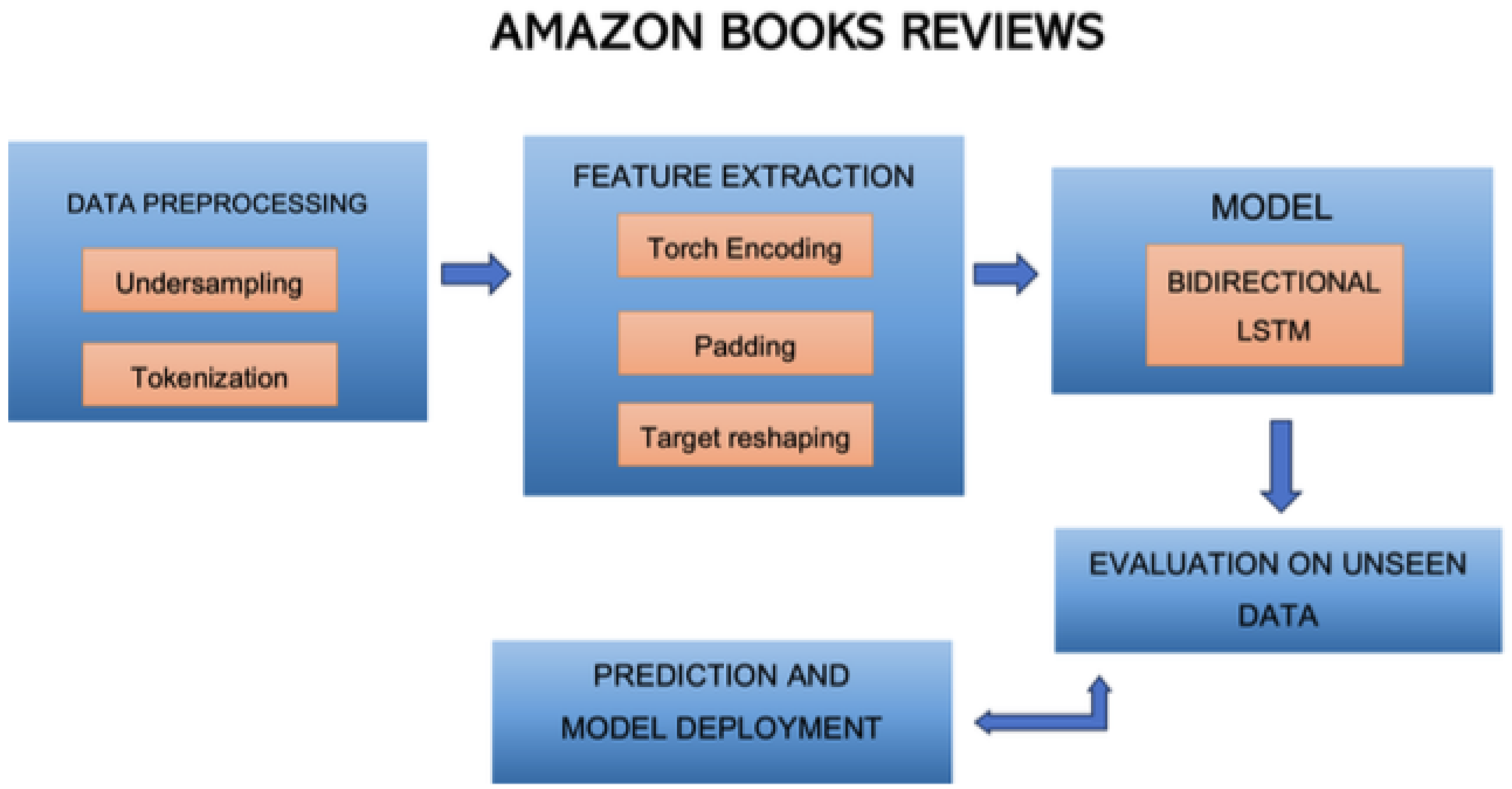
In order safeguard the model from training on heavily skewed data and to enhance its generalization ability, we have undertaken a **sampling reduction approach**. In the forthcoming section, we will investigate the precise definition and methodology of this technique.



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



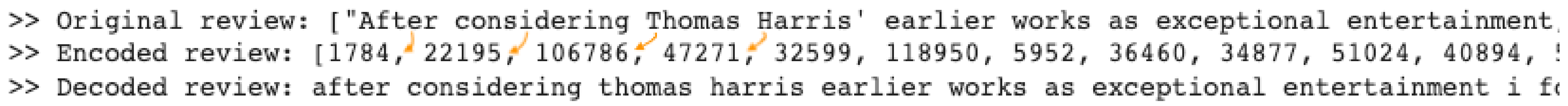

AMAZON BOOKS REVIEWS
DATA PREPROCESSING
Undersampling
Tokenization
FEATURE EXTRACTION
Torch Encoding
Padding
Target reshaping
MODEL
BIDIRECTIONAL LSTM
EVALUATION ON UNSEEN DATA
PREDICTION AND MODEL DEPLOYMENT

Methodology

Initially, we conducted a comprehensive analysis of the dataset, exploring its features and essential information. 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.

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. We applied tokenization to each review (2D array of review/text feature). In the meantime, 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.
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** 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.

Methodology

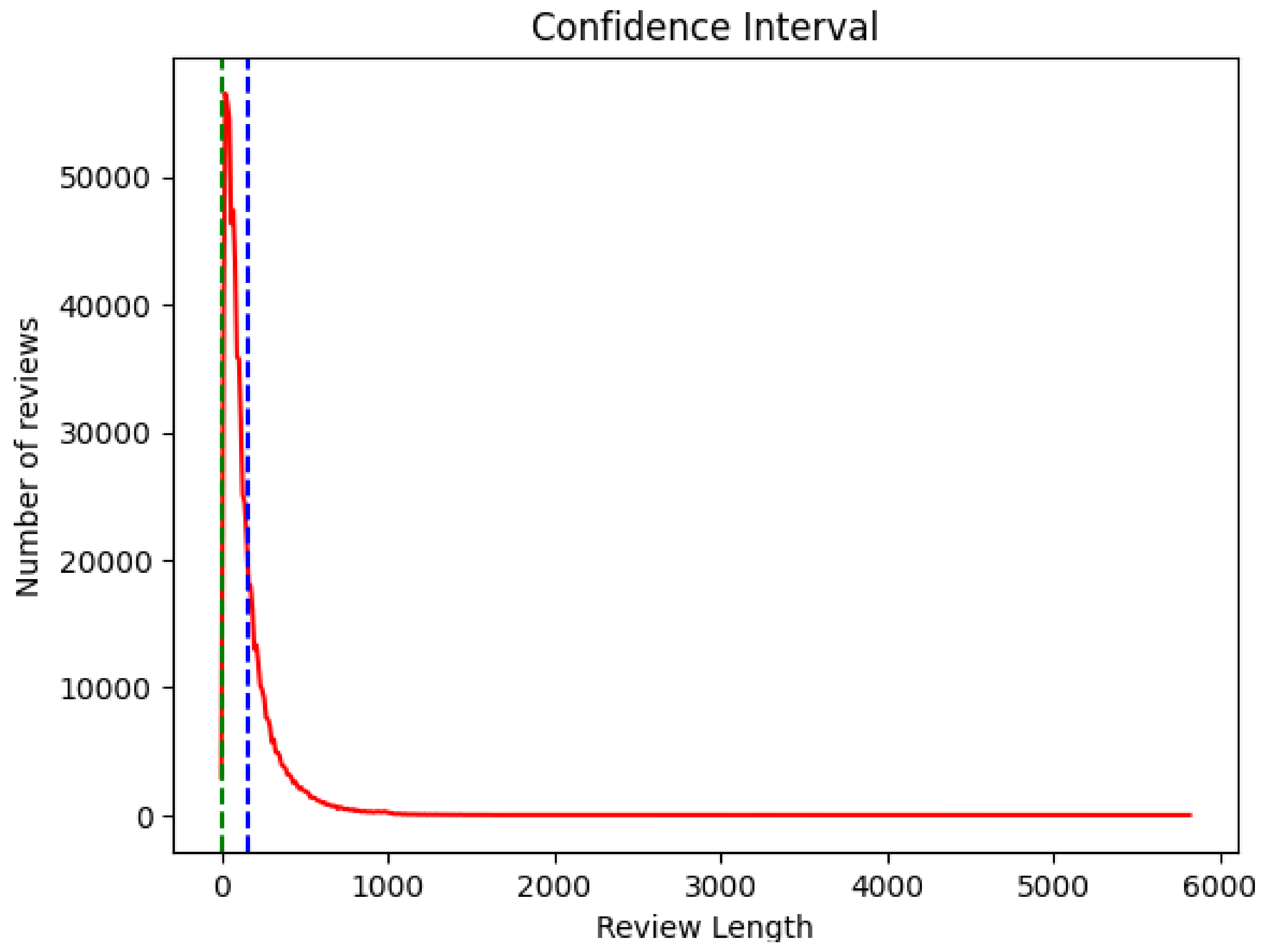


```
>> Original review: ["After considering Thomas Harris' earlier works as exceptional entertainment
>> Encoded review: [1784, 22195, 106786, 47271, 32599, 118950, 5952, 36460, 34877, 51024, 40894,
>> Decoded review: after considering thomas harris earlier works as exceptional entertainment i f
```

Encoding Process

Methodology

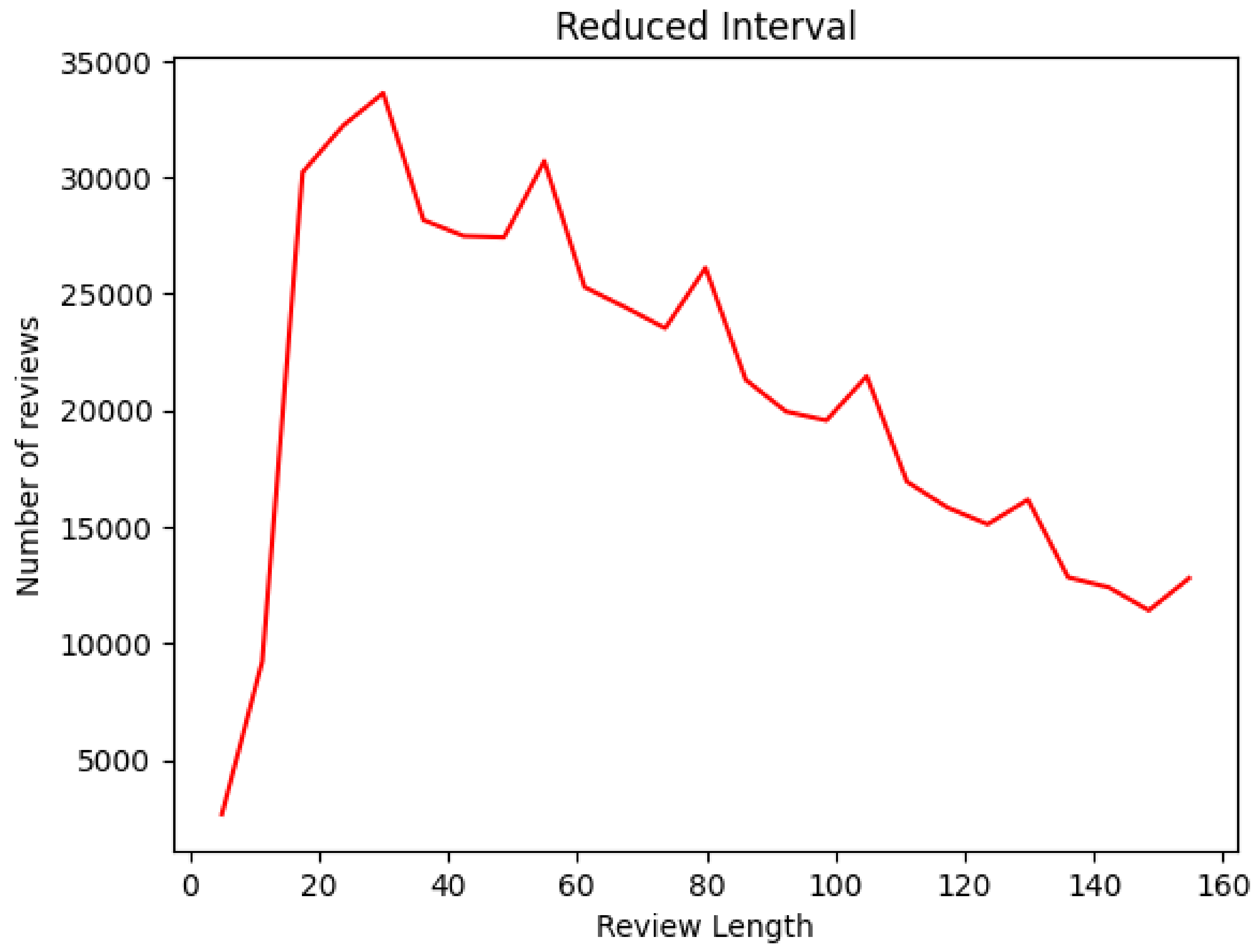
In order to conclude the preprocessing part, we conducted a statistical analysis of the reviews.
Our goal was to establish a **confidence interval** for the distribution of review lengths. This step was crucial because we aimed to maintain a high degree of uniformity in our corpus, avoiding overly lengthy reviews that could lead to increased processing costs.



Methodology

To maintain the degree of uniformity, we approached the problem through the following 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, as we can appreciate in the figure.
2. Performed **Padding** to fill shorter reviews and in order to have the same dimensionality among them.



Methodology

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 <u>Module</u>, which serves as base for all neural network modules. This model 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**
2. **BiLSTM**
3. **Dropout Layer**
4. **Linear Layers**

Leaky ReLU serves as the activation function following the fully connected layers, introducing non-linearity to the model. It plays a crucial role in mitigat- ing the vanishing gradient problem, enabling the model to acquire the capacity to learn intricate mappings.

Methodology

Let’s now have a better look into the different layers of our architecture by a piece by piece explanation:

- **Embedding Layer**: 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.
- **Bi-directional Long Short-Term Memory**, LSTM: 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.
- **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.
- **Linear Layers**: these are fully connected (dense) linear layers. 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’).

Methodology

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

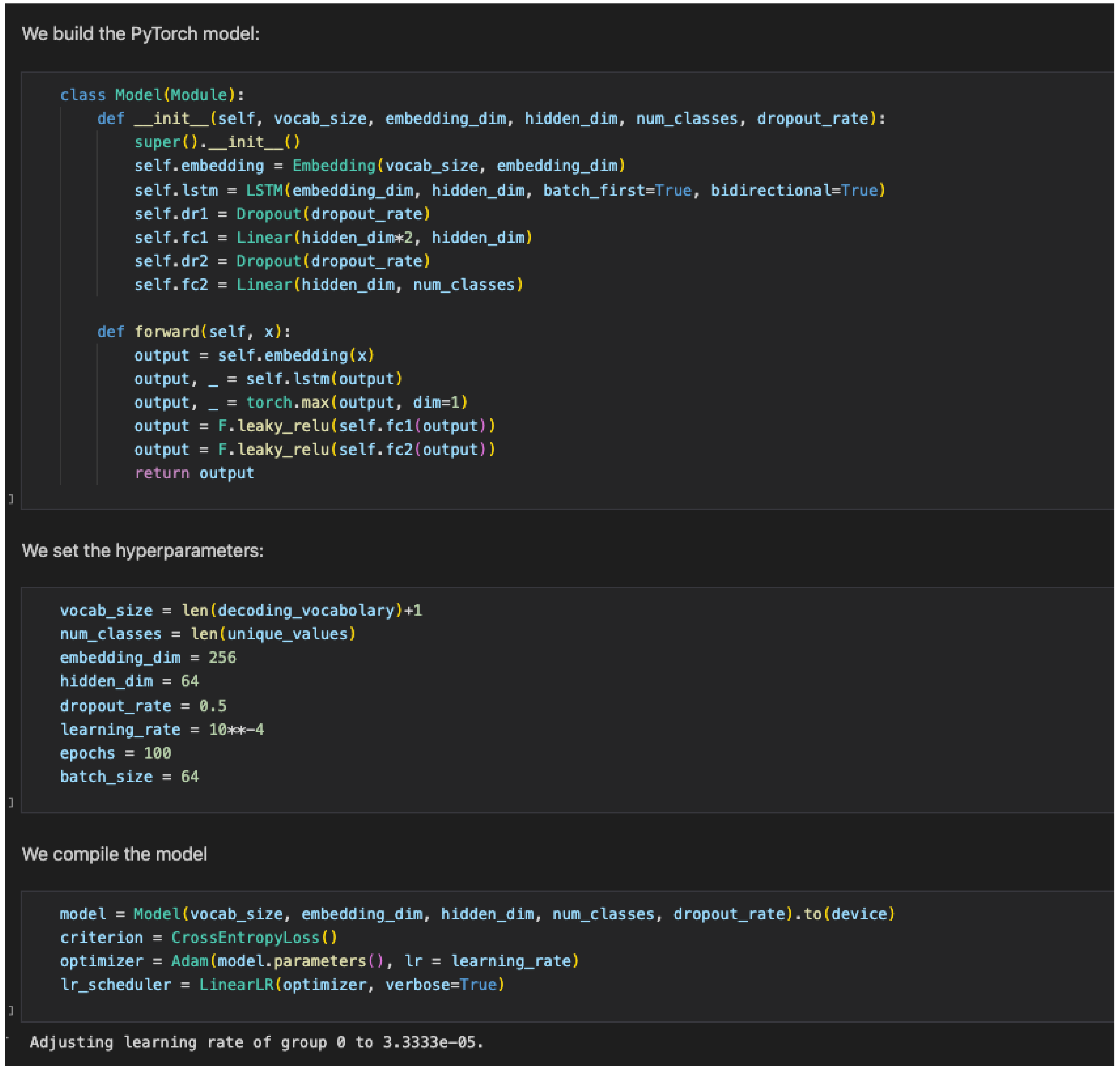
- *embedding_dim*
- *hidden_dim*
- *dropout_rate*
- *learning_rate*
- *batch_size*

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 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 preferred 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 behaviour.
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 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.

Methodology

On the right we can see the code representing the model architecture described so far.
As we can notice, of course, there are all the specific values selected for **hyperparameters** and all the methods and classes used.



We build the PyTorch model:

```
class Model(Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_classes, dropout_rate):
        super().__init__()
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.lstm = LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.dr1 = Dropout(dropout_rate)
        self.fc1 = Linear(hidden_dim*2, hidden_dim)
        self.dr2 = Dropout(dropout_rate)
        self.fc2 = Linear(hidden_dim, num_classes)

    def forward(self, x):
        output = self.embedding(x)
        output, _ = self.lstm(output)
        output, _ = torch.max(output, dim=1)
        output = F.leaky_relu(self.fc1(output))
        output = F.leaky_relu(self.fc2(output))
        return output
```

We set the hyperparameters:

```
vocab_size = len(decoding_vocabolary)+1
num_classes = len(unique_values)
embedding_dim = 256
hidden_dim = 64
dropout_rate = 0.5
learning_rate = 10**-4
epochs = 100
batch_size = 64
```

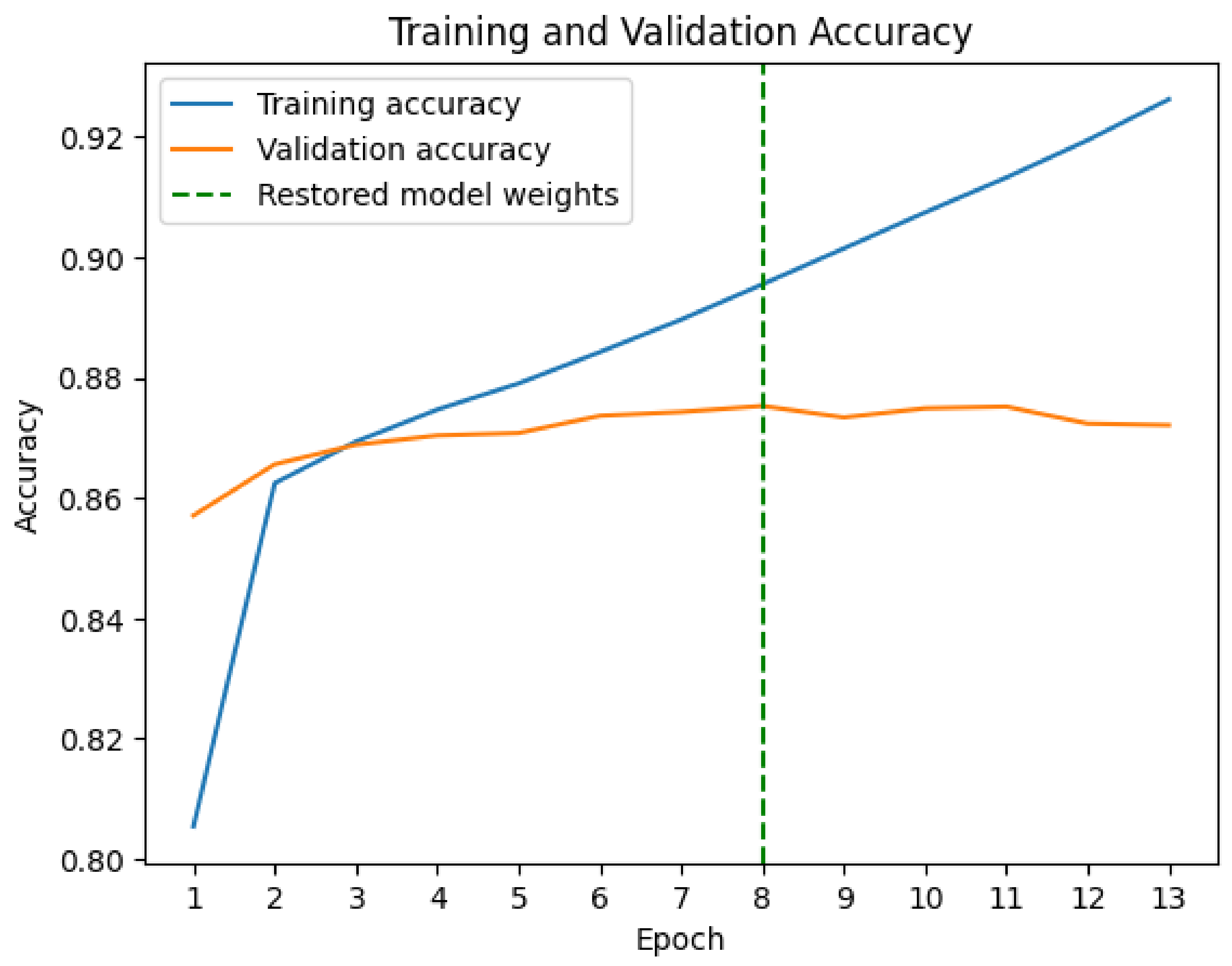
We compile the model

```
model = Model(vocab_size, embedding_dim, hidden_dim, num_classes, dropout_rate).to(device)
criterion = CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr = learning_rate)
lr_scheduler = LinearLR(optimizer, verbose=True)
```

```
Adjusting learning rate of group 0 to 3.3333e-05.
```

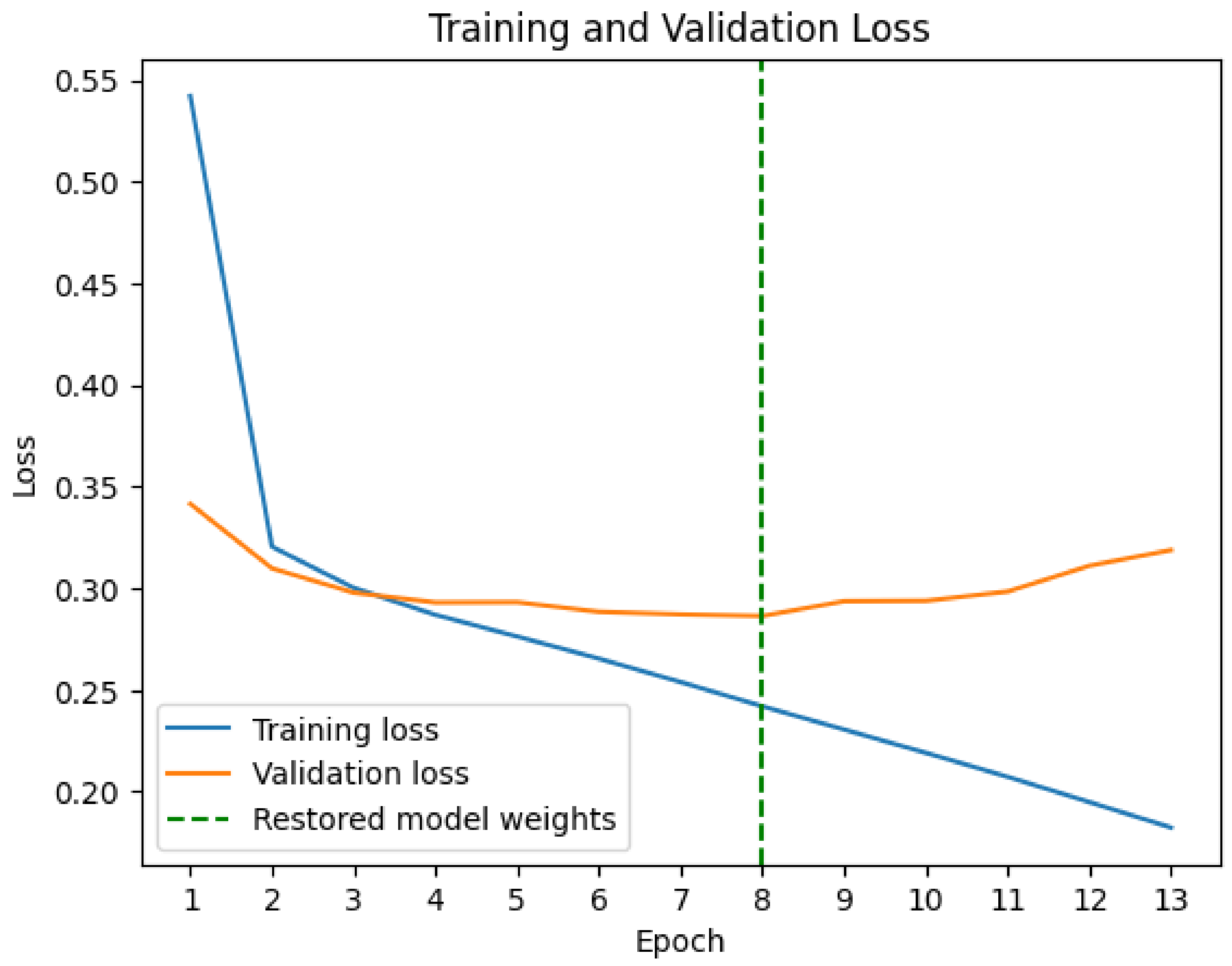
Result

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**, 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.

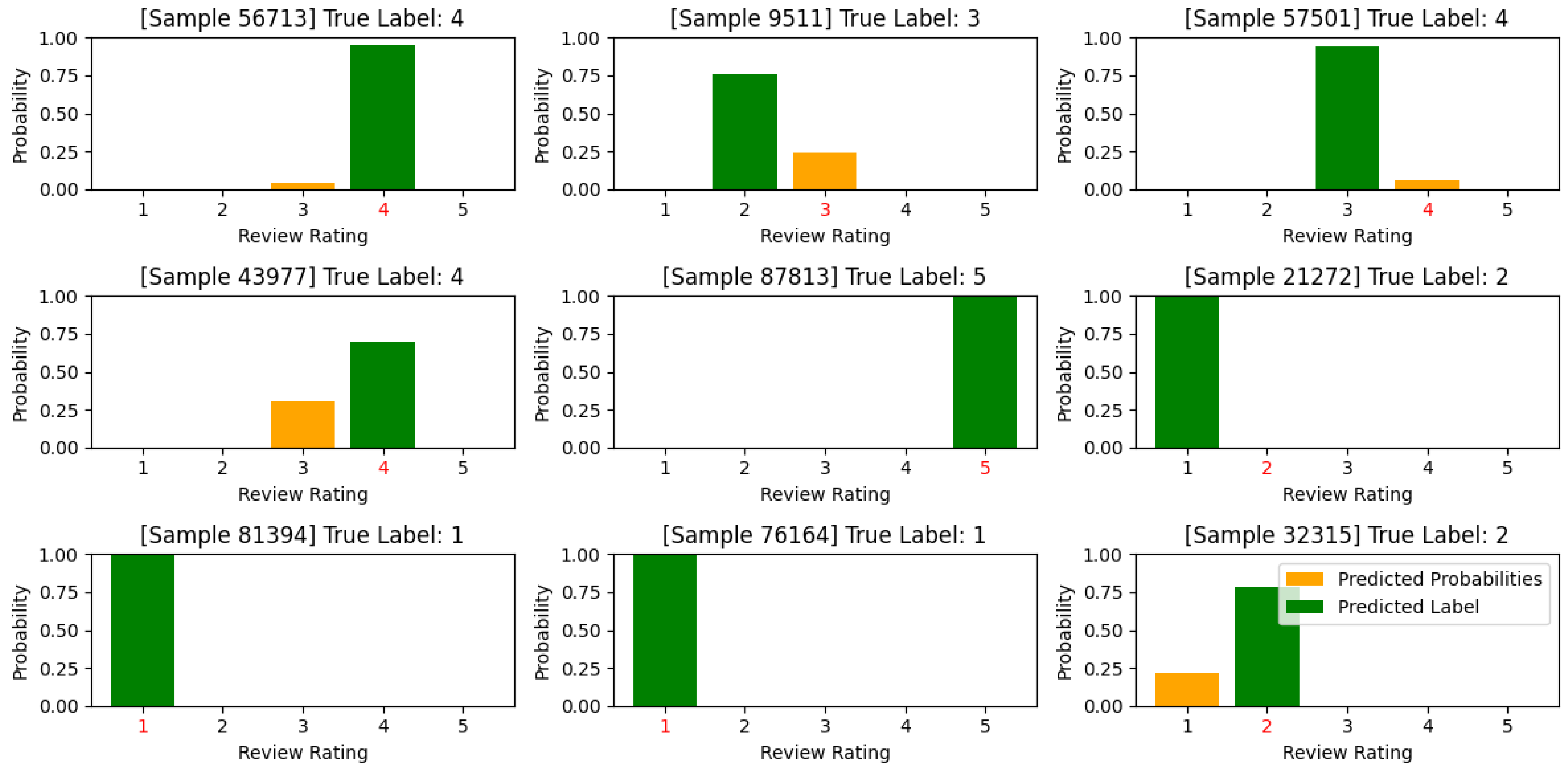


Result

The training and validation accuracy plot indicates that the model's **generalization** is robust. We observe stable and satisfactory results in both the training and validation phases, particularly up to *epoch 8*.
 It is evident from the graph that we applied **early stopping** at this point to prevent **overfitting**, a decision well-supported by the plot.
In order to have a complete visualization, we have also plotted the **Loss** (metric that quantifies the difference between the predicted values and the real ones).

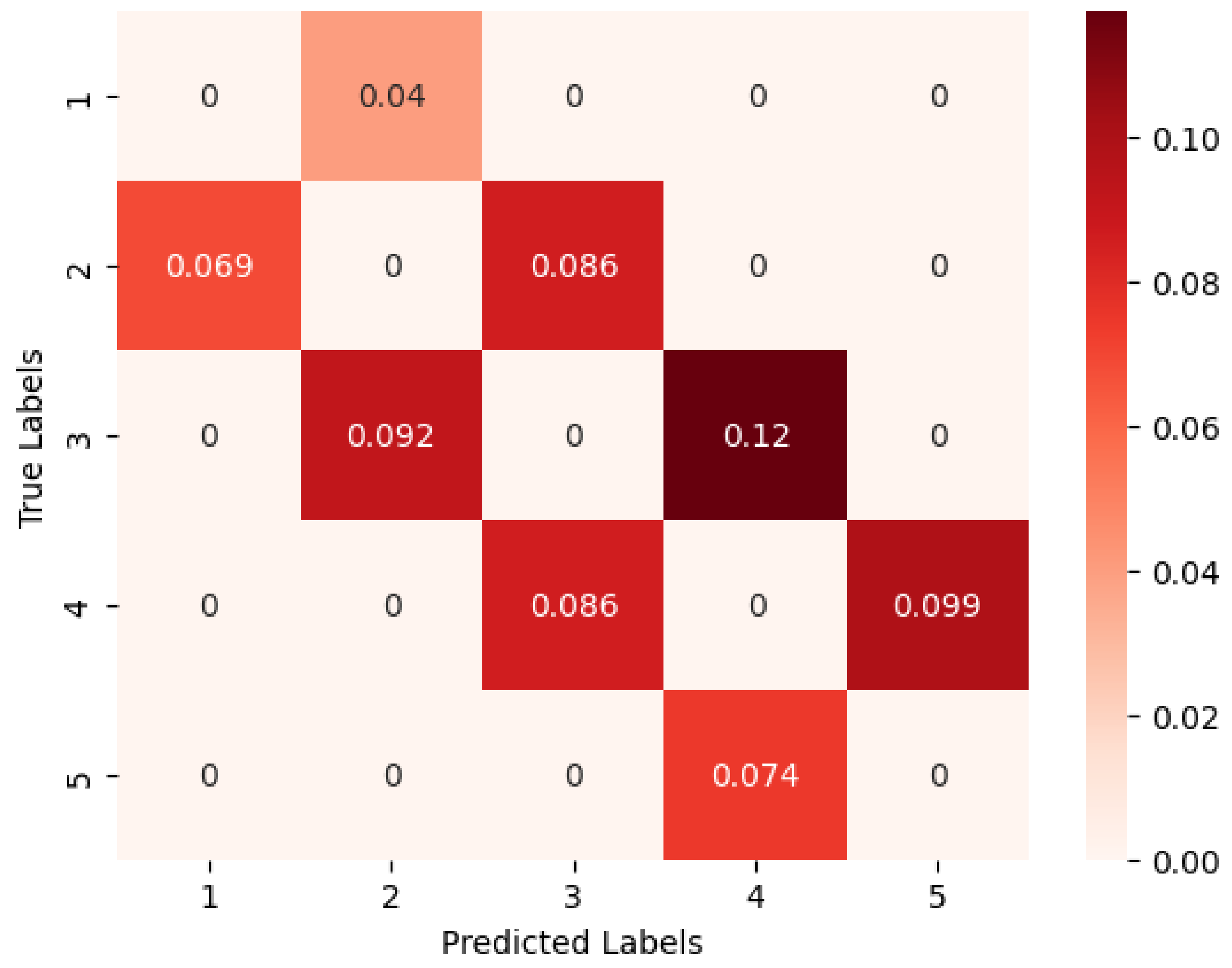


Result



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 the graph.

Result



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

Conclusion

As the concluding section of this report, we aim to provide a summary of all the work conducted so far. 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 hypeparameters**, 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, 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.

The End,
Thank you.