MDSAA

Master Degree Program in
Data Science and Advanced Analytics

Text Mining

Duarte Girão, Nº 20220670
Eduardo Palma, Nº 20221022
José Miguel Mato, Nº 20220607
Miguel Ramos, Nº 20210581

Group 11
NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

June, 2023

1. Introduction

This project aims to predict whether an Airbnb listing will be unlisted or listed in the following quarter. We will utilize textual fields associated with each unique Airbnb to extract valuable insights and develop a predictive model. The project involves data exploration, pre-processing, feature engineering, classification modelling, and result comparison. By analysing the textual data, we aim to uncover patterns and features that contribute to a listing's likelihood of being unlisted. The data exploration phase involves understanding the dataset's structure and content, including their relation with the target. Pre-processing techniques such as spell checking, stop word removal, and lemmatization will be applied to enhance the quality of the textual data. Feature engineering will extract relevant information from the textual fields and represent them numerically. Various classification models, including logistic regression, k-nearest neighbours, and long short-term memory networks, will be trained and evaluated. The project's outcome will provide valuable insights, aiding in proactive decision-making and strategies for host retention and guest satisfaction.

2. Data Exploration

2.1. General Overview

After conducting the initial analysis, it can be observed that each Airbnb listing possesses a distinct *"host_about"* and *"description"* textual field. However, the presence of *"comments"* textual field may vary, ranging from none to multiple comments per listing. In the original training dataset, comprising Airbnb's prior to the train-test split, there are 12,496 unique listings with corresponding *"host_about"* and *"description"* fields. Additionally, a total of 721,402 comments were recorded. While a few duplicated rows were identified, they were considered as multiple comments from different individuals or couples pertaining to the same Airbnb listing. Notably, no missing values were detected. Instead, occasional single instances of *"."* were found in certain textual fields, which we interpreted as instances where individuals opted to complete the form without providing any text.

During the general overview, two notable findings emerged regarding the target distribution and language usage across the textual fields of unique Airbnb listings. First, concerning the target distribution, it was observed that 72.3% of the listings in our training set remained listed in the subsequent quarter, while 27.7% were unlisted. This class imbalance will need to be addressed, either through under sampling the majority class or by applying class weights during the training process. Regarding language usage, English emerged as the predominant language across the various textual fields. However, the second most utilized language exhibited more variability depending on the field under consideration. In the *"host_about"* and *"description"* textual fields, Portuguese emerged as the second most commonly used language, which aligns with the assumption that these listings are primarily located in Portugal. Conversely, in the *"comments"* textual field, French emerged as the second most prevalent language, likely due to a significant number of tourists from French-speaking countries such as France, Belgium, and Switzerland.

2.2. Exploring the Patterns in the "Comments" Textual Field

Our analysis of the *"comments"* textual field is divided into three main sections. Firstly, we examine the size of the comments, followed by an exploration of the specific words used within these comments. Lastly, we investigate the frequency and distribution of the comments themselves.

Within the *"comments"* textual field, the comments vary in length, ranging from a single word to as long as 1137 words. On average, the comments consist of approximately 50 words. Through our analysis, we observed a trend where smaller comments tend to indicate user satisfaction (with the usage of emojis), while longer comments often reflect a sense of disappointment. However, it is important to note that this pattern is not universally applicable and there are exceptions to this general observation. In the subsequent Data Exploration phase, we will further investigate whether there exists a correlation between the comment length and the target variable, aiming to validate this initial finding.

During the analysis of the specific words used within the *"comments"* textual field, we opted to separate the comments made for listings that remained listed from those for listings that were unlisted. This division aimed to determine whether there were any notable differences in the most frequently used words in the reviews. It came as no surprise that the most common words encompassed articles, prepositions, conjunctions, adverbs, and nouns. However, upon removing stop words, which are commonly occurring words without significant contextual meaning, the most prevalent words in both the listed and unlisted comments turned out to be the same. Examples of such common words include *"apartment"*, *"stay"*, *"place"*, and "Lisbon".

Examining the comment count per Airbnb listing, we observe that the average number of comments is 57, with a maximum of 1175 comments. The 1st, 2nd, and 3rd quartiles indicate 0, 18, and 76 comments, respectively. Upon segregating the comments based on each label, several significant conclusions can be drawn. We find that for unlisted Airbnb's, the average number of comments is 8, whereas for listed Airbnb's, this average increases to 76. Analysing the distribution of comments in both cases, we note that it is predominantly concentrated around the value of 0 for unlisted Airbnb's. Conversely, for listed Airbnb's, the number of comments exhibits greater variability, with a higher occurrence of comments. These findings initially suggest a correlation between the comment count and the likelihood of an Airbnb being listed in the upcoming quarter. However, we will delve deeper into exploring this correlation in subsequent analyses.

2.3. Exploring the Patterns in the "Host About" and "Description" Textual Field

In the analysis of the "host_about" and "description" textual fields, we will not conduct a specific word analysis as it yielded inconclusive results in previous attempts. However, our focus will shift towards identifying potential relationships between the size of these textual fields and the corresponding labels.

Considering all the descriptions, the average length of each description is 142 words, with the smallest description containing 3 words and the longest consisting of 362 words. The 1st, 2nd, and 3rd quartiles are 105, 168, and 179 words, respectively. Upon segregating the descriptions based on the label, we observe that the distribution, though similar, tends to have smaller average descriptions for Airbnb's that will be unlisted in the following quarter. This indicates a potential correlation between the description size and the likelihood of being unlisted. We will further calculate and examine this correlation in subsequent analyses.

The average size of the host about description is 82 words, with a maximum length of 1597 words and a minimum of 1 word. The 1st, 2nd, and 3rd quartiles indicate 28, 61, and 113 words, respectively. From the analysis of the host about description size, we couldn't draw significant conclusions. There were no substantial differences observed when segregating the descriptions based on the different labels.

2.4. Correlation Analysis

The correlation between the number of comments and the label is -0.33, indicating a slight correlation. This suggests that as the number of comments on an Airbnb listing increases, it becomes more likely for it to appear listed in the upcoming quarter. Conversely, listings with fewer comments are less likely to stay listed. Similarly, the correlation between the size of the description and the label is -0.13, signifying a slight correlation. This implies that Airbnb's with longer descriptions are more likely to be retained in the upcoming quarter. On the other hand, smaller descriptions are less likely to appear listed. It is crucial to consider the possibility of spurious correlation in this context. It is plausible that the larger descriptions are a result of the Airbnb listings being of higher quality, offering better amenities and services. Consequently, these listings are more likely to receive positive ratings and remain listed in the upcoming quarter.

In section 2.2., we identified a potential correlation between the comment size and the listing status of the Airbnb. However, the resulting correlation was close to 0, suggesting no significant relationship between the two variables. Similarly, we assessed the correlation between the size of the host about and the label, which also yielded a correlation value near 0.

3. Pre-Processing

3.1. Data Sets Translation

Prior to commencing pre-processing, we were faced with two alternatives: either retaining the textual fields of our training and test sets in their respective original languages, and subsequently employing multilingual pre-processing text techniques and multilingual feature engineering techniques to generate embeddings; or translating all the textual fields of the training and test set into English, enabling the utilization of conventional pre-processing and feature engineering techniques. It was important to consider that the former approach presented certain limitations, as stop words in one language could be ordinary words in another language, and vice versa. After weighing the pros and cons of each method, we have opted for the second approach presented. Despite the additional time required in the initial stage, it is anticipated to enhance the effectiveness of subsequent stages.

3.2. Pre-Processing Pipeline

Subsequently, we will outline each pre-processing technique that we initially took into account, and ultimately present the ones that were included in our final pre-processing technique pipeline:

- **Lowercasing:** Changing all uppercase letters to their corresponding lowercase counterparts while leaving lowercase letters unchanged.
- **Stop Word Removal:** Commonly used words in a language that often do not carry significant meaning or contribute much to the understanding of the text. Examples of stop words in English include "a", "an", "the", "is", "and", "in", "to", and so on.
- **Named Entity Recognition:** Subtask of information extraction that involves identifying and classifying named entities in text into predefined categories such as person names, locations, organizations, dates, etc.

- **Stemming:** Reduces words to their base or root form, known as the "stem". The stem may not necessarily be a valid word itself, but it represents the core meaning or essence of the word. Helps to overcome challenges such as different tenses, plurals, or derived forms of words that may occur in a given text.
- **Lemmatizing:** Transforms different inflected forms of a word into a single base form while preserving the correct part of speech. The main advantage of lemmatization over stemming is that it produces valid words that exist in the language. This helps maintain the interpretability and semantic integrity of the text.
- **Spell Checking:** Involves automatically detecting words that are not spelled correctly and suggesting or applying appropriate corrections.
- **Punctuation Removal:** Punctuation marks, such as periods, commas, and question marks, don't carry significant meaning on their own. By removing punctuation, the focus can be placed on the actual content and structure of the text.
- **URL Links and HTML Tag Removal:** Not meaningful for textual analysis and can introduce biases or distractions.
- **Contractions Expansion:** By expanding contractions, abbreviated or contracted words are converted into their complete forms.

In relation to the final pipeline, our initial step was to convert all text to lowercase since it doesn't impact any other technique. Afterward, we focused on removing URLs and HTML tags to ensure proper handling of these elements. We then proceeded with transforming contractions to expand abbreviations. It was important to perform these steps before punctuation removal because removing punctuation marks beforehand might have resulted in the unintended deletion of URLs, HTML tags, and abbreviations. Next in our pipeline, we performed punctuation removal and stop word removal. Following these steps, we incorporated spell checking and lemmatization. Since lemmatization relies on corrected words, we made the decision to apply spell checking before lemmatization. We utilized the pipeline on both the training set and the test set. We can apply the pipeline to the test set is that we do not require any prior knowledge about the test set in order to apply these steps.

3.3. Train-Validation Split

In the end, we performed a standard train-validation split, allocating 70% of the data to the training dataset and setting aside 30% for the validation dataset. Since our dataset is highly imbalanced, we decided to apply a stratified train-validation split. This approach ensures that each class is proportionally represented in both the training and validation sets. Furthermore, we plan to address the issue of class imbalance during model training. We will assign greater weight to the minority class to ensure that our model generalizes as effectively as possible and captures the patterns and characteristics of the minority class.

Extra Work: Within the pre-processing, the extra work dedicated to the treatment of the datasets included the translation of all textual fields into English, spell checking, removal of URL links and HTML tags, as well as expanding contractions.

4. Feature Engineering

4.1. TF – IDF with Unigrams

The first feature engineering technique we utilized was TF–IDF (Term Frequency-Inverse Document Frequency), a very common numerical representation technique used in text mining, that measures the level of importance of each term within aggregated documents. Each document is composed of terms with weights that reflect their significance value. The weight is determined by considering two factors: TF, Term Frequency, which evaluates the frequency of a term within a document, with higher weights indicating greater occurrence within the document; and IDF, Inverse Document Frequency, which gauges the rarity or distinctiveness of a term across the entire collection of documents, with higher weights indicating less common occurrence across the document collection as a whole.

TF-IDF with 2 Grams approach, a variant of the TF-IDF technique, includes not only individual terms but also pairs of consecutive words (2-grams) as features. It calculates the importance of these 2 grams within a document based on their frequency and rarity. By considering 2-grams, the TF-IDF approach captures not only single words but also the pair word combinations, providing a more complex representation of the text. Unfortunately, due to limitations in computational power and time resources, we were unable to execute TF-IDF with 2-grams, despite our belief that it would enhance performance. Not only did storing all the features occupy over 10GB of space, but it also significantly extended the duration of model training.

Due to its simplicity, we have decided to apply this feature engineering technique exclusively to the *"host_about"* and *"description"* textual fields. We believe that alternative feature engineering methods will yield more favourable results. However, we still wanted to demonstrate the functionality of this technique, even though we expect imperfect results in the classification models. Additionally, we have chosen to concatenate both textual fields into a single column and run a single TF-IDF process, instead of performing TF-IDF separately for each field. This approach eliminates the possibility of duplicate columns for the same word when merging the results of the individual TF-DFs. We also decided to merge our training and validation datasets into a new dataset in order to train our TF-IDF model. This will allow for vocabulary consistency as if we applied separately on each dataset, we would have the possibility of having different vocabularies for each one of them. So this merge ensures that our vocabulary is consistent across the entire dataset. Moreover, this approach enhances the coverage of N-grams, as incorporating more text data increases the likelihood of capturing a wider range of N-grams.

After preparing our datasets, we proceeded to apply the TF-IDF technique. Firstly, we trained the TF-IDF model using the concatenated column of the "host_about" and "description" textual fields, along with the merged training and validation dataset. Subsequently, we independently applied the trained model to the train, validation, and test datasets. This resulted in three distinct matrices, where each row represents a document, and each column corresponds to a specific feature. The values within the matrices indicate the TF-IDF scores, reflecting the significance of a term within a document relative to the entire corpus. To facilitate accessibility and manipulation of the TF-IDF values, along with their corresponding feature names and indices, we converted the sparse matrices into three dense dataframes.

In the end of this section on the notebook we present the 20 most important features based on their TF-IDF scores. The three features with highest scores are “apartment”, “isbn” (representing “Lisbon”), and “room”.

4.2. GloVe Embedding Model

GloVe (Global Vectors for Word Representation) is a popular word embedding technique that represents words as dense vectors in a high-dimensional space. GloVe is designed to capture the semantic relationships between words based on their co-occurrence statistics in a given corpus. It starts by creating a co-occurrence matrix to measure how often words appear together in a corpus. From this matrix, a probability distribution is derived. Word vectors are initialized randomly and then trained using an iterative process to minimize a loss function that captures the mismatch between the dot product of word vectors and the logarithm of co-occurrence probabilities. The trained word vectors form a dense representation of words in a vector space, capturing semantic relationships. GloVe's advantage lies in its ability to incorporate both local and global co-occurrence statistics. These word embeddings are useful in various natural language processing tasks.

To incorporate GloVe embeddings into our project, we opted for a pre-trained GloVe model since our own corpus wouldn't encompass the entirety of the English dictionary. We specifically selected a pre-trained model that was trained on a massive dataset containing 6 billion words. Our approach involved applying the GloVe embeddings to each individual word of each of the three textual fields (*"host_about"*, *"description"*, and *"comments"*) within our three distinct datasets: training, validation, and test. Due to computational and time constraints, we made the decision to embed each word individually and then aggregate the embeddings by summing them within the same sentence. Furthermore, for the "comments" textual field, which may contain multiple comments per index, we also sum the embeddings of sentences that correspond to the same index. By adopting this approach, we can effectively address the computational and time limitations. However, it comes at the cost of losing efficiency and the inherent relationships between different words when inputting them into a model. In an ideal scenario, the optimal approach would involve concatenating the embeddings of each word within the same sentence and concatenating the sentences from the same index and textual field (*“comments”*). Unfortunately, we were unable to implement this method due to the memory constraints it would impose, resulting in excessive memory usage. As a result, we successfully obtained a single embedding consisting of 100 numerical values for each textual field corresponding (*"host_about"*, *"description"*, and *"comments"*) to each index.

The resulting output will be a dataset with three columns representing *"host_about"*, *"description"*, and *"comments"*. Each column will contain embeddings consisting of 100 numerical values for each unique index in the dataset. This output will be saved so that it can be used during the classification models phase.

4.3. DistilUSE Embedding Model

The DistilUSE model is a pre-trained sentence embedding model that captures the semantic meaning of sentences. It is based on the DistilBERT model, which is a distilled version of the original BERT mode. It uses a transformer architecture, which is a type of neural network known for its effectiveness in natural language processing tasks. The model is a distilled version of a larger model, designed to be more compact while maintaining performance. It provides universal sentence embeddings, meaning it can handle multiple languages and retain case information. By using this

model, we were able to obtain fixed-length vector representations (embeddings) of 512 numerical values for sentences. The sentence embeddings are typically derived from the second-to-last transformer layer. The input sentence is processed through multiple transformer layers, and the outputs of the second-to-last layer are averaged to obtain a fixed-length sentence embedding.

Due to the limitation of the embedding model to handle only 512 tokens at a time, we had to segment and embed textual fields that exceeded this limit separately. To reconstruct the original sentence, we summed the embeddings obtained from the segmented portions. In order to optimize memory usage, we implemented a specific strategy for embedding. In cases where multiple sentences existed within the same textual field and index (specifically, the "comments" field), we chose to sum the individual embeddings. Due to the limitation of the embedding model to handle only 512 tokens at a time, we had to segment and embed textual fields that exceeded this limit separately. To reconstruct the original sentence, we summed the embeddings obtained from the segmented portions. In order to optimize memory usage, we implemented a specific strategy for embedding. In cases where multiple sentences existed within the same textual field and index (specifically, the "comments" field), we chose to sum the individual embeddings. This choice was made to address the storage requirements that would have emerged from concatenating the embeddings of each sentence from the same index and textual field ("comments" field). Nonetheless, it is crucial to acknowledge that this approach could potentially lead to reduced model efficiency as it results in a loss of the interrelation between different words.

The resulting output will be a dataset with three columns representing *"host_about"*, *"description"*, and *"comments"*. Each column will contain embeddings consisting of 512 numerical values for each unique index in the dataset. This output will be saved so that it can be used during the classification models phase.

4.4. XLM - RoBERTa Embedding Model

The XLM - RoBERTa model is a pre-trained model based on XLM-RoBERTa architecture, specifically designed to produce multilingual sentence embeddings for estimating semantic textual similarity between sentences. It generates a fixed-length vector representation that captures the semantic content of the input text. In the case of XLM-RoBERTa, the sentence embeddings are typically obtained from the last layer of the XLM-RoBERTa encoder. Each token in the input sentence is encoded by passing through multiple transformer layers, and the output of the last layer for each token is averaged to obtain a fixed-length sentence representation of 768 numerical values

The methodology employed for sentences surpassing 512 tokens, along with the summation of embeddings based on corresponding indices and columns (*"comments"* textual field), closely resembled the approach utilized in the pre-trained DistilUSE embedding model. Despite its advantages, this approach has inherent drawbacks, including a potential loss of efficiency and a loss of the inherent relationships between words.

The resulting output will be a dataset with three columns representing *"host_about"*, *"description"*, and *"comments"*. Each column will contain embeddings consisting of 768 numerical values for each unique index in the dataset. This output will be saved so that it can be used during the classification models phase.

Extra Work: DistilUSE and XLM-RoBERTa embedding models as extra methods using transformer-based embeddings.

5. Classification Models

In the fourth stage of our project, we tested different machine learning models and compared their performance to identify the best fit for our dataset. Our approach involved training each model on the training datasets (the ones resulting from TF-IDF and each one of the embeddings) to learn patterns and relationships, aiming for more accurate predictions. Additionally, we employed the validation dataset to evaluate the models' performance, fine-tune parameters using grid-search, and prevent overfitting. By assessing the models on the validation dataset, we aimed to select parameter settings that would generalize well to unseen data. Finally, we applied the best-performing model to the test set to obtain our final predictions.

5.1. Logistic Regression

The first model we trained was the Logistic Regression with the training dataset resulting from the TF-IDF. Logistic Regression is a supervised learning algorithm commonly used for binary classification tasks and capable of handling multi-class classification. It models the relationship between the input features and the probability of belonging to a specific class, by applying the logistic function (which consists of the sigmoid function). Among the different parameters we included in the fine-tuning, we would like to highlight the *"C",* that controls the regularization strength, *"penalty",* a regularization mechanism that penalizes large coefficient values to prevent overfitting, and *"class_weight",* which determines the weights associated with each class. After evaluating different combinations of *"C"*, *"penalty"* and *"class_weight"* values, we achieved a best combination of *"0.1"*, *"L1"* and *"balanced"* (adjusts the class weights so the dataset is balanced), respectively, resulting in a final F1-Score of 0.76 when the trained model is applied in the validation set.

5.2. K-Nearest Neighbours

The second model we trained was the K-Nearest Neighbours with the training dataset resulting from the TF-IDF. KNN is a supervised machine learning algorithm that can be used for classification, although it can also be applied to regression tasks. "K" in KNN represents the number of nearest neighbours considered for making the prediction. The algorithm labels every new data point by assigning the label of most of its K nearest neighbours in the training data. Although this model may be simple to understand and implement, it can be computationally expensive for large datasets, which is what we verified when applying it. The only parameter we included in the fine-tuning was the number of neighbours, "K". After evaluating different K values, we found out that the best model was only considering 1 neighbour at a time. After applying this model to our validation dataset we were able to obtain an F1-Score of 0.76.

5.3. Multi-Layer Perceptron

The third model we trained was the multi-layer perceptron model with the training dataset resulting from the TF-IDF. The MLP is also a supervised learning algorithm typically used for classification tasks. It is composed by multiple layers of interconnected neurons, which receive inputs, apply an activation function to produce an output, and pass it to the neurons in the next layer. During training, the model adjusts the weights and biases of the neurons, a process known as backpropagation and is performed by an optimization algorithm. The hidden layers between the input and output layers enable the network to learn and capture complex patterns in the data, making this model more powerful

and capable of learning more sophisticated patterns, especially when compared to the two previous models (KNN and Logistic Regression). We applied a more robust fine-tuning with 8 parameters, from which we highlight *"hidden_layers_sizes",* which specifies the number of neurons that compose the neural network's hidden layers, *"activation",* defining the activation function used in the hidden layers, and *"learning_rate"*, which defines how the learning rate is adjusted during training. After performing the grid-search, we achieved a best combination of *"(50, 50)"*, *"relu"* (bringing non-linearity to the model) and *"adaptative"* (based on the training progress and validation score) respectively, resulting in a final F1-Score of 0.76 when the trained model is applied in the validation set. Note that due to time and computational constraints, we have decided to use only 30% of our original dataset size for training this specific model.

5.4. Long Short-Term Memory

For our final model we decided to change a bit the approach, and we are going to train three different LSTM, each one of them with a different embedding, the first one with the GloVe Embedding, the second one with the DistilUSE Embeddings and the third one with the XLM – RoBERTa model.

The LSTM is a type of recurrent neural network (RNN) that is designed to handle sequence data by capturing long-term dependencies. It overcomes some of the limitations of traditional RNNs, which struggle with capturing long-term dependencies due to the vanishing gradient problem. The LSTM architecture includes memory cells, which allow it to selectively remember or forget information over time. An LSTM cell consists of three main components:

- **Cell State (Ct):** This represents the memory of the LSTM and can retain information over long sequences. It is updated using gates that control the flow of information.
- **Input Gate (it):** This gate controls the input information to be stored in the cell state. It decides which parts of the input should be stored.
- **Forget Gate (ft):** This gate controls the forget operation on the cell state. It decides which information should be discarded from the cell state.

In our specific scenario, we will concatenate the embeddings of the three textual fields, namely "host_about," "description," and "comments," for each unique index. This process results in a single embedding for each index, with sizes of 300, 1536, and 2304 numerical values, corresponding to the usage of GloVe, DistilUSE, and XLM-RoBERTa embedding techniques, respectively. However, employing only one embedding per observation means that we lose the capability to capture contextual information within the sequence. Additionally, it fails to capture the sequential dependencies that exist in the data. By compressing the entire observation into a single vector representation, we inevitably encounter information loss. Despite these limitations, this approach also offers advantages, such as reduced computational demands.

After training each one of the three LSTM models (trained with class weights as we have an unbalanced dataset) with GloVe, DistilUSE and XLM-RoBERTa embeddings and when applying the trained model to our validation set we were able to obtain F1-Scores of 0.86, 0.87, and 0.88, respectively.

6. Evaluation

	Accuracy	Precision	Recall	F1-Score
Logistic Regression w/ TF-IDF	0,78	0,79	0,78	0,75
K-Nearest Neighbours w/ TF-IDF	0,78	0,77	0,78	0,77
Multiple Layer Perceptron w/ TF-IDF	0,76	0,74	0,76	0,75
LSTM w/ GloVe Embedding	0,86	0,87	0,86	0,86
LSTM w/ DistilUSE Embedding	0,87	0,87	0,87	0,87
LSTM w/ XLM - RoBERTa Embedding	0,88	0,88	0,88	0,88

Figure 1 – Evaluation Metrics from All the Models

In order to assess the performance of our models, we opted to utilize four distinct evaluation metrics:

- **Accuracy:** Measures the overall correctness of the model's predictions. It calculates the ratio of correctly predicted instances to the total number of instances in the dataset. However, it may be misleading if the classes are imbalanced, as it doesn't consider the distribution of predictions among different classes.
- **Precision:** Measures how many of the predicted positive instances are actually true positives. It calculates the ratio of true positives to the sum of true positives and false positives. A high precision value suggests that the model has a low rate of falsely labelling negative instances as positive.
- **Recall:** Also known as sensitivity, measures the proportion of true positives that are correctly identified by the model. It calculates the ratio of true positives to the sum of true positives and false negatives. A high recall value indicates that the model has a low rate of falsely labelling positive instances as negative.
- **F1-Score:** Combines precision and recall into a single metric. It is the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy and completeness. The F1-score is especially useful when the classes are imbalanced, as it considers both false positives and false negatives.

During the training and evaluation of our models, we focused primarily on the F1-Score metric due to the imbalanced nature of our dataset. Among all the models, the LSTM with RoBERTa Embedding demonstrated the best performance when evaluated on the validation set, achieving an F1-Score of 0.88. As a result, we selected this model to make predictions on the test set. The embedding of the test set and the trained model are stored in the submitted folder, and in the end of the notebook "Classification Models and Results" there is the code for the submitted predictions.

7. Conclusion

To sum up, the main objective of this project was to predict whether an Airbnb listing would be unlisted or listed in the following quarter. Valuable insights were derived by analysing the textual fields associated with each unique Airbnb, leading to the development of a predictive model. This model was then utilized to accurately predict the labels for the test set, achieving the primary goal of the project.