Text Mining Report

MASTER'S DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Predicting Airbnb Unlisting

Group 17

Iryna Savchuk - 20211310 - m20211310@novaims.unl.pt
Cátia Parrinha - 20201320 - m20201320@novaims.unl.pt
Pedro Anastácio - 20180040 - m20180040@novaims.unl.pt

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

1. Introduction

This report presents the final assessment for the Text Mining course within the master's degree Program in Data Science and Advanced Analytics at Nova IMS, Lisbon. The objective of this project is to utilize Natural Language Processing (NLP) techniques and models to predict the unlisting of properties on Airbnb in the upcoming quarter. The prediction will be based on a combination of the actual property descriptions, host descriptions, and guest feedback in the form of comments. The goal is to develop a classification model using the acquired NLP techniques from the course to determine whether a property will be unlisted (1) or remain listed (0).

To complete this task Python was used, along with libraries such as NLTK, Scikit Learn, PyTorch, and Transformers. The success of this project has the potential to provide valuable insights and predictions for property owners and Airbnb itself. This information can aid in making informed decisions regarding property management and optimizing listing strategies. Additionally, the project serves as an opportunity to apply and reinforce the knowledge acquired in the Text Mining course, specifically in the domain of NLP and classification techniques.

Throughout this report, the methodologies used will be discussed, we will also describe how the data was employed, present the implemented models, and evaluate their performance with a classification report and a confusion matrix.

2. Data Exploration

Four Excel files were provided, consisting of two types of data (Table 1 and Table 2) divided into train and test datasets. The target variable, "unlisted," was only present in the train.xlsx dataset, while the remaining datasets included only independent variables.

Although there was a relationship between the two train datasets, it was decided to explore them separately rather than merging them initially. To gain an overview of the data and better understand its characteristics, the Pandas profiling report was utilized. The 'train' dataset (Table 1) consisted of 4 columns and 12,496 rows, while the 'train_reviews' dataset (Table 2) consisted of 2 columns and 721,402 rows. All independent variables were in the form of text.

Regarding missing values, neither dataset had any, but there were duplicate rows in 5.14% of 'train' and 2.18% of 'train_reviews'. Upon inspecting some of these duplicates, it was observed that some individuals simply commented with a period ".", which could be considered for removal as it lacked meaningful information.

An initial observation revealed an imbalanced distribution of the target variable, "unlisted," with 72.3% of the data labeled as "0" (Figures 1 and 2). Despite the separate exploration of the datasets, the analysis and exploration of the three textual independent variables, namely "description," "host_about," and "comments," were conducted in parallel.

The first step in exploring the independent variables was to tokenize the text and count the tokens. In the "description" variable, the maximum number of tokens in a single row was 210, and the distribution (Figure 3) showed that a significant number of rows had token counts between 150 and 180. Further analysis of the frequency of individual tokens (Figure 4) revealed the need for cleanup to remove HTML tags such as "<b>". The "host_about" variable had a larger maximum token count, which seemed unusual as it should contain more information about the property owners/managers rather than the properties themselves (Figure 5). Examining the top 35 most frequent tokens in "host_about" (Figure 6) showed that not all tokens were meaningful words, including tokens like "X000D" generated during text extraction from Excel sheets. The "comments" variable, originating from a larger dataset, exhibited a distribution of the number of tokens per row primarily ranging from 0 to 63 tokens (Figure 7), reflecting the brevity of comments. Analyzing the most frequent tokens (Figure 8) in "comments" revealed the presence of languages other than English, such as French. Word clouds were generated for the three independent variables (Figures 9, 10, and 11), further highlighting the presence of Portuguese as another language in the data.

Based on the previous exploration, it became obvious that data contained multiple languages. Therefore, the next step involved language detection. In the "description" variable, there were 11 additional languages besides the initial three, accounting for only 2.38% of the data in that variable (Table 3). The "host_about" variable contained 29 other languages besides the initial three, representing 5.22% of the data in that column (Table 4). Conversely, the "comments" variable included 44 other languages, which constituted 14.6% of the data in the 'train_reviews' dataset (Table 5).

3. Data Preprocessing

The data cleaning process began with the utilization of the BeautifulSoup library to eliminate HTML tags, excessive white spaces, and newline characters from the "description," "host_about," and "comments" features using regular expressions.

The second step involved translating the "description" and "host_about" features to English using the "MarianMTModel" and "MarianTokenizer" classes from the "transformers" library. A multilingual to English translation model named "Helsinki-NLP/opus-mt-mul-en" was employed.

For the third step, a function named "clean()" was defined with three parameters: text_list, lemmatize, and stemmer. Within this function, missing values in the text_list were filled with empty values, so the function was bulletproof for future use in the test set or new data of any kind. The text was then converted to lowercase, emojis were transformed to text using the "demoji" library, URLs were removed, numerical data and punctuation marks were eliminated, and stopwords for English, Portuguese, and French were removed since they were the most prevalent languages in both datasets, since 'train_review' was not translated. The application of lemmatization and stemming was explored both individually and in combination, but ultimately, it was determined that lemmatization alone yielded the best performance.

After applying the aforementioned cleaning steps to the three independent variables, a comparison was made between the raw data and the cleaned data to observe the changes. In the "description" variable,

the number of unique words decreased from 1,648,726 to 16,387 after cleaning. Similarly, for "host_about", the number of unique words was reduced from 931,865 to 11,190. Lastly, in the case of "comments", the number of unique words decreased from 21,570,231 to 169,634.

Due to the extensive size of the 'train_reviews' dataset, the decision was made to solely include reviews that had been identified as written in English, considering the time constraints and the impracticality of translating the remaining data.

This updated version of the reviews dataset with only the English comments was then merged with the clean train dataset.

Finally, the train/validation split was applied using the stratify parameter on the target variable. This was done because, as observed previously, the distribution of the target variable is imbalanced, and using stratify ensures a similar representation proportion of classes in both the train and validation sets. After the split, the independent features were further split into 3 subsets corresponding to the textual features - 'description', 'host_about', and 'comments' - so that the next feature engineering steps could be applied to them independently.

Extra work:
1 - Language detection;
2 - Removal of HTML tags using the BeutifulSoup library;
3 - Converting emojis to their text description with the use of the 'demoji' library;
4 - Multilingual translation to English using transformers.

4. Feature Engineering

The first technique explored in this section involved creating binary Bags of Words (BoW) representations for the independent variables. CountVectorizer instances were used to transform the training and validation datasets into BoW vectors for the text features "description," "host_about," and "comments". These BoW representations were then merged into two feature matrices for training and evaluating the model.

The second technique utilized was Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, considering both 1-gram and 1-and-2-gram features. Similar to the previous technique, three separate instances were used to transform the training and validation data into TF-IDF vectors, which were then merged into two feature matrices for model training and evaluation.

The subsequent technique, which was not covered in class, involved applying Hashing Vectorization to the independent features in the training and validation datasets. This technique maps text features to a fixed number of dimensions using hash functions, disregarding the vocabulary size. We created hashed representations and combined them into feature matrices for training and evaluating the model.

Next, custom features were created. The first feature involved converting the previously identified languages of the independent variables ("description" and "host_about") in the train and validation datasets into a new binary feature: "1" for English text and "0" for any other language. Second, we built

a number of counts, such as counts of words, counts of reviews, counts of reviews languages, etc. Also, during the initial data exploration, it was noticed that the feature "description" contained important details about the properties, such as capacity, proximity to touristic places in minutes, number of rooms/bedrooms, additional costs associated with stay, and whether the property had a license number. By using regular expressions we extracted these details and added new columns to both the train and validation datasets. Another source of potentially valuable information was the sentiment of the customer reviews, so we calculated sentiment scores for the given comments using three different methods: TextBlob polarity, NLTK SentimentIntensityAnalyzer, and DistilBERT sentiment analyzer. All of the obtained sentiment scores ranged from -1 to 1 (where -1 represented negative sentiment, 0 denoted neutral sentiment and 1 indicated positive sentiment). In the end, all the custom features that we built were scaled using StandardScaler to further be used in modeling.

The final technique explored was the BERT tokenizer. It encodes textual features in the training and validation datasets using the pre-trained model "bert-base-uncased". The encoding process is performed with the get_encodings() function, which converts the resulting encodings into string arrays and creates a list of dictionaries containing the input IDs and attention masks. Then, a DictVectorizer is initialized and fitted using the encoding dictionaries to obtain separate feature matrices for training and validation. Similar to the previous techniques, this process is applied simultaneously to all three textual features and their respective training and validation sets. Finally, all feature matrices are combined to create the final training and validation datasets.

Extra work:
1 - HashingVectorizer;
2 - Numeric feature extraction using regular expressions;
3 - Sentiment with TextBlob;
4 - Sentiment with NLTK SentimentIntensityAnalyzer;
5 - Sentiment with transformers (DistilBERT);
6 - BERT Tokenizer.

5. Classification Models

Before delving into model experimentation, we defined a function to assess the model prediction performance by plotting the corresponding confusion matrix. In addition, for each classification model built, we will be using Sklearn 'classification_report' which provides a summary of key metrics such as precision, recall, F1-score, and support for each class in the classification problem.

This section of the project involves using different feature representations created in the previous section: Bag-of-Words (BoW), TF-IDF 1-gram, TF-IDF 1-and-2-gram, Hashing Vectorizer, Custom Features, and Encodings by BERT Tokenizer. These representations are fitted to various models using the training dataset, and predictions are made on the validation dataset.
The tested models include K-nearest Neighbours, Random Forest, Logistic Regression, Neural Networks, and XGBoost. The provided Jupyter Notebook includes the classification_report and confusion matrix for each feature representation with each tested model.

To explore some of the more advanced techniques, a BERT model (Bidirectional Encoder Representations from Transformers) was trained on the "descriptions" feature and subsequently pushed to the HuggingFace Model Hub. Its performance was evaluated using the validation dataset, but unfortunately, appeared to be really poor, with an accuracy score of about 0.69 and an average f1-score of 0.65.

Another more advanced technique that we decided to try was Ensemble Learning - a Machine Learning technique that involves combining multiple individual models to create a more powerful and accurate predictive model. In particular, we used a Voting Classifier that combined predictions from the three most promising models built in the previous step to make the final prediction.

Since the Voting Classifier turned out to improve the macro average F-1 score (see the Evaluation and Results section below), we ended up using GridSearch to tune hyperparameters for it.

Extra work:
1 - Neural Networks;
2 - XGBoost;
3 - Training BERT (Bidirectional Encoder Representations from Transformers);
4 - Voting Classifier
5 - GridSearch for Voting Classifier

6. Evaluation and Results

As previously mentioned, we used both the confusion matrix and Sklearn 'classification_report' to evaluate the performance of the trained models. However, to determine the best model, we chose to evaluate the performance through the macro avg of the f1-score metric, which returns the average f1 score for unlisted and listed predictions. This metric can be seen as an overall measure of the model's ability to perform well on both classes, regardless of their imbalance or difference in sample sizes.

Initially, we employed the K-nearest Neighbours Model to evaluate the effectiveness of each distinct feature ('description', 'host_about', 'comments', and 'combined') across the various representations that we developed (BOW, TF-IDF, Hashing Vectorizer). Our observations indicated that the 'combined' feature matrix, which merged all three features, demonstrated superior performance.

KNN	Bow	TF-IDF (1-Gram)	TF-IDF (2-Gram)	Hashing Vectorizer	Custom Features	BertTokenizer
description	0.68	0.7	0.7	0.69		
host_about	0.71	0.71	0.71	0.7		
comments	0.82	0.43	0.42	0.82		
combined	0.85	0.85	0.85	0.86	0.82	0.82

Following that, we chose to develop all other models with this 'combined' feature matrix that encapsulated all the data. The table below shows the results for the macro average f1-scores obtained for all the models. It can be noticed that, on average, Random Forest models exhibited better performance, along with the BOW and TF-IDF representations, in both 1-gram and 1-and-2 gram combinations.

Models / Feature Engineering	Bow	TF-IDF (1-Gram)	TF-IDF (2-Gram)	Hashing Vectorizer	Custom Features	BertTokenizer
KNN	0.85	0.85	0.85	0.86	0.82	0.82
Random Forest	0.86	0.86	0.86	0.85	0.84	0.86
Logistic Regression	0.84	0.86	0.86	0.84	0.83	0.84
Neural Networks	0.85	0.83	0.83	0.82	0.83	0.81
XGBoost model	0.78	0.8	0.81	0.8	0.75	0.78
BERT						0.65
Voting Classifier Ensemble			0.87			

In terms of comparing performance across created features, the table above shows that TF-IDF has rather good performance across all the models, especially TF-IDF which extracts a bit of context (1-and-2 gram).

Having these points in mind, we decided to go the extra mile and build a Voting Classifier based on TF-IDF 1-and-2 gram representations. This Voting Classifier is an example of Ensemble Learning that would use the predictions of multiple individual models (K-Nearest Neighbors, Random Forest, and Logistic Regression) and combine them to make the final prediction. The idea behind Ensemble Learning is that by combining the predictions of multiple models, the overall prediction can be more robust, accurate, and reliable compared to using a single model. We also have chosen to use a soft voting scheme, in which the predicted probabilities from multiple models are averaged to make the final prediction (in contrast to 'hard' voting where majority rule voting is used).

By training Voting Classifier, we achieved a small growth of the macro average f1-score, indeed, so the model has become a little more balanced in terms of its predictive performance on both classes.

Lastly, we run the Grid Search parameter tuning for the Voting Classifier to explore hyperparameters for its basic models, in particular: the number of nearest neighbors for the KNN classifier in [7, 9, 11]; the number of trees in the Random Forest in [50, 100]; and the inverse of the regularization strength in Logistic Regression in [1.0, 0.1, 0.01]. Fitting 5 folds for each of 18 combinations (totaling 90 fits) has been run and the best model has been built based on the best parameter combination. This best model has been evaluated in the same manner as all other models (Figures 12 and 13) and was used by our group to do predictions on the 'test' (unseen) dataset.

To ensure the reproducibility of the submitted prediction values, the 'dump' function from the 'joblib' package was used to save the final model to a file named 'best_model.pkl'. It also was used to archive the fitted TfidfVectorizers for each of the textual columns and generate three files - 'tfidf_desc_2g.pkl', 'tfidf_host_2g.pkl', and 'tfidf_reviews_2g.pkl' - corresponding to descriptions, host_about, and comments.

As a final note, we would like to share the link to the Google Drive folder, which contains the .csv files built throughout the project as well as the output pickle files for the model and vectorizers mentioned in the previous paragraph:
https://drive.google.com/drive/folders/11wCygDHOt6lq1SJhVEQfzh54-eqNV7yE

7. References

[1] - “Natural Language Processing with Python” tutorial, https://python-course.eu/machine-learning/natural-language-processing-with-python.php, retrieved June 18, 2023.

[2] - Hugging Face website, https://huggingface.co/learn/nlp-course/chapter3/4?fw=pt , retrieved June 18, 2023.

[3] - Scikit-learn documentation on VotingClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html, retrieved June 18, 2023.

8. Appendix

8.1 Tables

Variable Name	Description
index	Property identifier in the form of an integer
description	Description of Airbnb properties in the form of a string
host_about	Description of the host of an Airbnb property in the form of a string
unlisted	The dependent variable. If the property is unlisted (1) or not (0)

Table 1

Variable Name	Description
index	Property identifier in the form of an integer
comments	Comments left by guest of Airbnb properties

Table 2

	Language	Percentage
0	en	72.170882
1	pt	20.403696
2	Other Languages	5.220493
3	fr	2.204929

Table 3

	Language	Percentage
0	en	81.650128
1	pt	14.740717
2	Other Languages	2.376761
3	fr	1.232394

Table 4

	Language	Percentage
0	en	64.172583
1	fr	14.877922
2	Other Languages	14.598733
3	pt	6.350761

Table 5

8.2 Figures

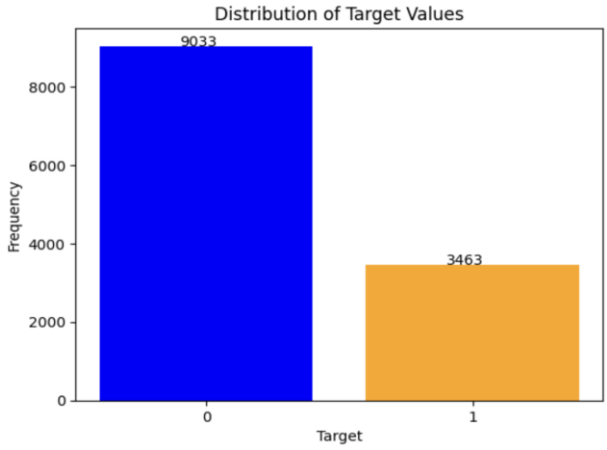

Figure 1

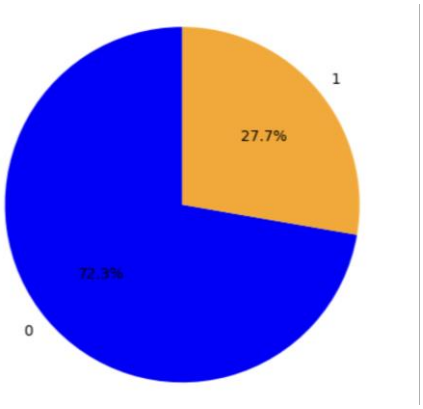

Figure 2

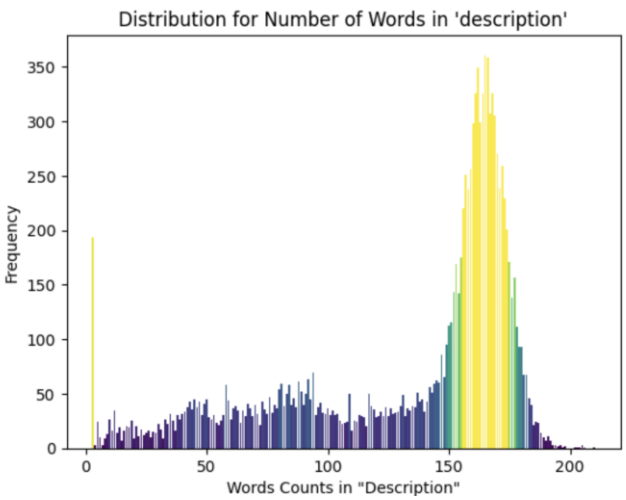

Figure 3

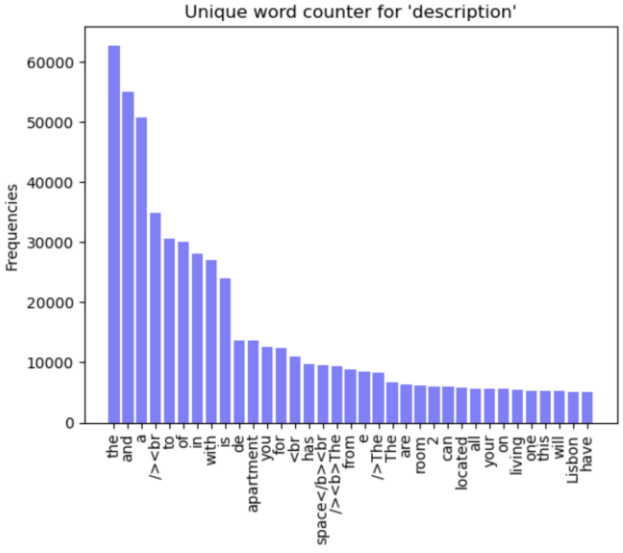

Figure 4

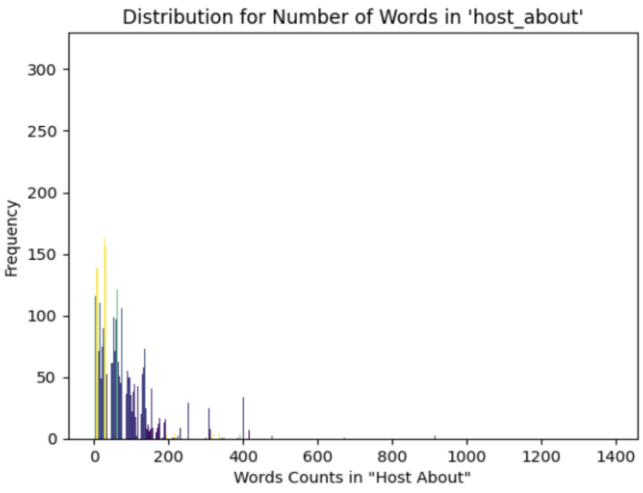

Figure 5

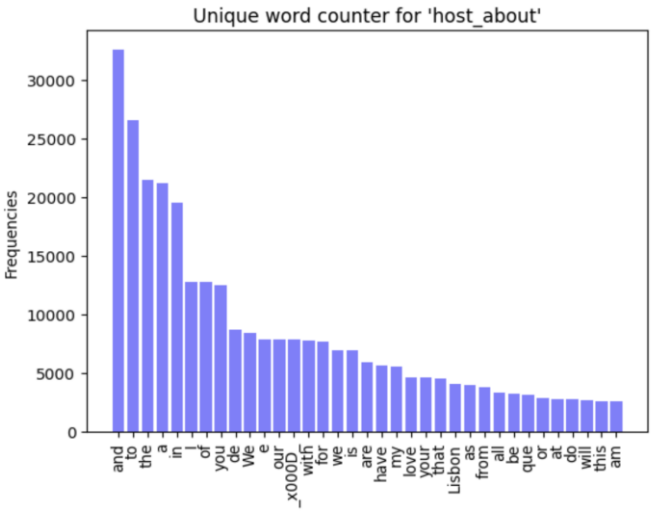

Figure 6

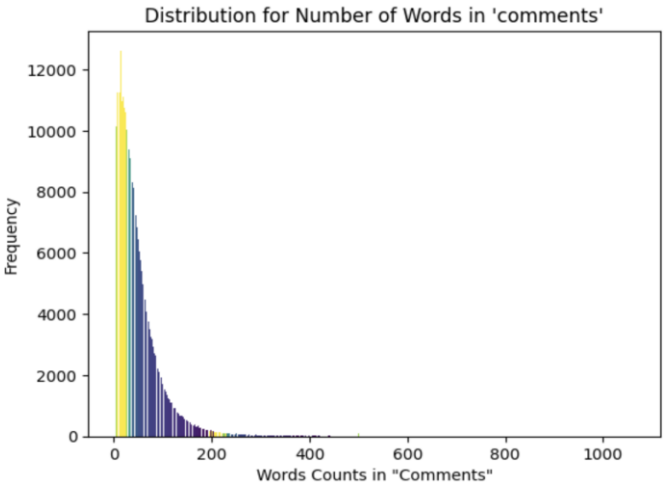

Figure 7

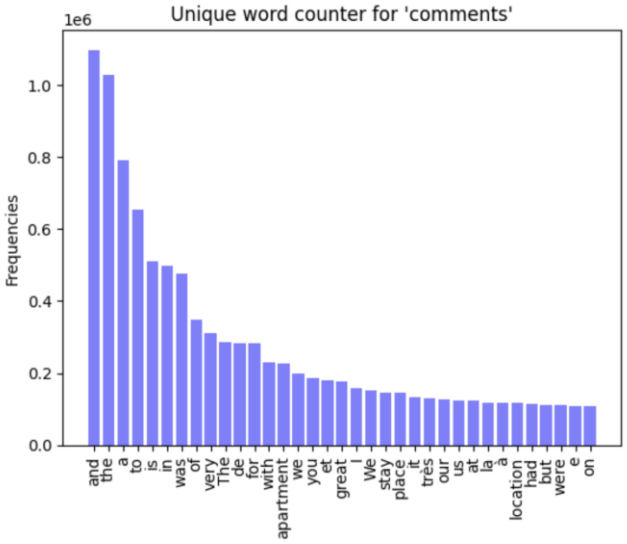

Figure 8

Figure 9

Figure 10

Figure 11

	precision	recall	f1-score	support
listed	0.92	0.93	0.93	1796
unlisted	0.81	0.80	0.81	704
accuracy			0.89	2500
macro avg	0.87	0.86	0.87	2500
weighted avg	0.89	0.89	0.89	2500

Figure 12

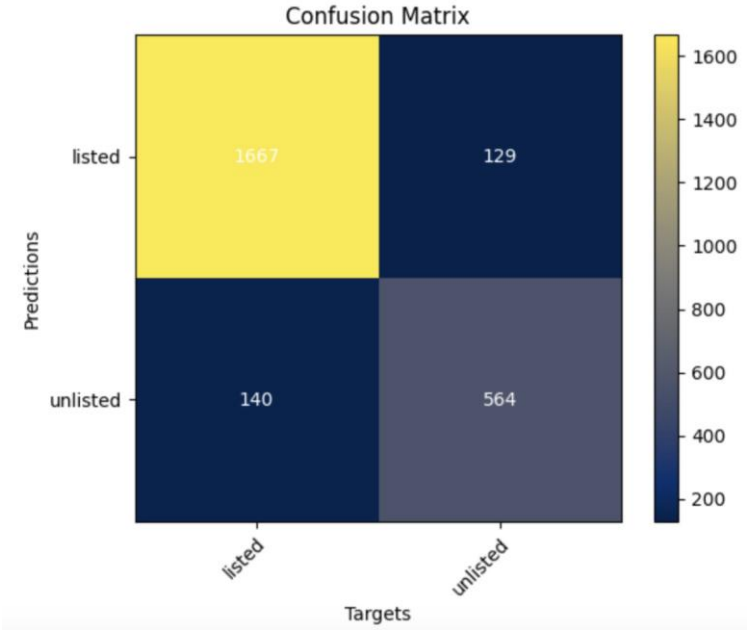

Figure 13