NOVA IMS

JUNE 2023

TEXT MINING PROJECT

USING NATURAL LANGUAGE PROCESSING TO PREDICT AIRBNB UNLISTINGS

GROUP 8:
FILIPE DIAS (R20181050)
INÊS SANTOS (R20191184)
MANUEL MARREIROS (R20191223)

Index

Introduction

The primary objective of this project is to leverage the power of Natural Language Processing (NLP) techniques to predict the likelihood of an Airbnb property being unlisted within the upcoming yearly quarter. By harnessing the wealth of information contained within past labeled data, we aim to develop models capable of learning from property descriptions, host descriptions, and guest comments. Through the analysis of these textual data sources, we can uncover valuable patterns, sentiments, and contextual information that may contribute to the event of a property getting unlisted. This approach allows us to go beyond traditional metrics and delve into the textual context, providing a more comprehensive understanding of the factors influencing unlistings.

Data Exploration

In this section, we looked into the data provided and decided the required preprocessing actions to be taken.

The datasets provided were the following:

- *airbnb_df*: Contains a description of each property and information about the respective host;
- *reviews*: Stores the comments left by guests on the properties.

Each set is divided in both *train* and *test,* with the difference being that the latter lacks a label.

Having imported the datasets, the first thing we wanted to check was how many of the 12496 AirBnB properties of our training set were unlisted after three months. We concluded that there were **3463 unlistings** in that time period, which accounted for **27%** of the total number of properties. This means we were dealing with a moderate level of class imbalance, which we mitigated further ahead.

We then looked at the word counts to understand what the average size of the descriptions was (132 words), what the size of the smallest descriptions was (only 3 words) and what the size of the largest descriptions was (210 words). We also wanted to perform a word-level analysis, concluding that the most common words were “the”, “and” or “a”, which do not provide any valuable information, or even html tags. Once we performed all these steps for the description, we repeated all of them for the comments on the reviews dataset. Finally, we created a word cloud with the most common words having a larger size and the less common ones being smaller.

Data Preprocessing

For data preprocessing, we implemented the following techniques:

- **Converting data types:** The values in the 'comments' column of both reviews dataframes were converted to strings. This operation ensures that all values in the 'comments' column are treated as strings, regardless of their original data type.
- **‘BeautifulSoap’:** This is a library that we utilized to remove all the html tags, as they did not contain any useful information and their presence would likely jeopardize the performance of our models.
- **Regular Expressions:** A regular expression is a sequence of characters that forms a search pattern. We noticed that in the properties’ descriptions there was usually a host license number. This is a common denominator in these descriptions, even if the number differs

between them. Because of that, we implemented regex to replace these license numbers with the normalized text "#LICENSENUMBER".

- **Stop Words:** We removed commonly used words, such as "the", "and", or "a", which provide little to no value in terms of conveying meaningful information.
- **Lemmatization:** This is the process of reducing words to their base or root form while considering their context.
- **Stemming:** While implemented, we opted for utilizing lemmatization instead. Stemming removes the last characters of a word, reducing it to a shorter form that doesn't necessarily have a meaning. Additionally, it does not consider the context or part of speech. While it is faster, it may result in less accurate results.
- **Lowercasing:** We converted all text to lowercase, in order to standardize the data.
- **Removal of Special Characters and Punctuation:** These characters hold little to no semantic meaning and can introduce noise. Removing them simplifies our data and makes it easier to process and analyse.

After doing that, we resorted to the ***langdetect*** library to filter the reviews dataset so that it included only English comments. Doing so allowed us to reduce the large number of comments, whilst maintaining enough of them to gain more insights about the unlistings. Then, we joined everything in a single dataset that contained four columns: *description*, *host_about*, *comments,* and the label. This is the dataset of which the columns are going to be vectorized and then passed through the models.

To mitigate the effects of the slight previously identified **class imbalance** in our training data, we opted for oversampling the minority class. To do so, we applied ***RandomOverSampler,*** which, by generating synthetic samples, aims to provide the model with a more balanced representation of the classes during training, that can help improve the model's performance, especially when the minority class is underrepresented.

Finally, we **split** our *train* data into *train* and *validation* datasets, where the former will help us train our models, and the latter will allow us to evaluate their performance. The split was done with a test size of 0.2, meaning **20%** of the data was used for validation, and the remaining **80%** was used for training. The split was also **stratified based on the *'unlisted'* column** to ensure a proportional representation of unlisted listings in both the *train* and *validation* sets.

Feature Engineering

We implemented the four following feature engineering techniques:

- **Bag-of-Words (BoW):** Represents the text data by counting the occurrence of each word in each document. We also utilized BoW to analyse some interesting n-grams, particularly bi-grams and tri-grams, of our data, and we plotted their frequency to gain insights into the importance or relevance of certain word combinations in your text data. Our findings lead us to consider unigrams, bigrams, and trigrams during the vectorization process of TF-IDF.
- **TF-IDF:** Converts the text comments into TF-IDF weighted numerical feature vectors.
- **GloVe:** Utilizes pre-trained word embeddings to represent words as dense vectors.
- **Multilingual Word Embedding:** Given the diverse range of languages in our project, including Portuguese, English, French, and German, we implemented multilingual embeddings. These embeddings capture the semantic meaning of words or phrases across multiple languages, allowing us to effectively employ NLP across language barriers. Implementing multilingual embeddings typically involves using pre-trained models or libraries specifically designed for

this purpose. The one we chose was **Sentence-BERT (SBERT**) [1], a widely used library known for its high-quality multilingual embeddings and efficient performance. As the name indicates, **SBERT is an extension of the BERT model**. Unlike traditional language models, BERT considers both the preceding and following words when encoding a given word, meaning it captures more contextual information. SBERT modifies this architecture to **produce embeddings at the sentence-level**, where semantically similar sentences are represented by vectors that are closer together. This is more computationally efficient than BERT. A task that BERT can take hours to performed can be performed with SBERT in seconds with a comparable level of accuracy.

These feature engineering techniques enable us to transform the text data into a numerical format that can be utilized by machine learning models for various NLP tasks. Each method offers different perspectives on capturing information from text, allowing us to extract meaningful features and improve the performance of our models.

To evaluate the effectiveness of the various techniques, we implemented them in different classifiers and conducted a comparative analysis of their performance. Based on the results, we concluded that the Bag-of-Words technique presented the worst performance, and for that reason it was only applied to the first two models, the most basic ones. Then, due to their stable and positive performance, both TF-IF and Multilingual Word Embedding were applied on all the models.

GloVe, specifically, was reserved for the Long Short-Term Memory networks. By leveraging GloVe, which provides pre-trained word embeddings, we aimed to improve the LSTM's ability to understand and represent the semantic meaning of words in the input data.

It is important to point out that these decisions were based both on the empirical results and the theoretical definitions of the different techniques. TF-IDF is often favoured when the emphasis is on highlighting the significance of rare words, allowing us to focus on the most informative features for predicting unlisting behavior. Bag-of-Words representation, although lacking in capturing semantic relationships, offers computational efficiency and simplicity. GloVe embeddings enrich LSTM models with semantic understanding, facilitating the capture of word meanings. Multilingual embeddings, on the other hand, enable cross-lingual generalization and knowledge transfer. By evaluating and understanding the empirical and theoretical aspects of these techniques, we made informed choices to enhance the effectiveness of the classifiers in our analysis.

Classification Models

We applied several classification models that can be used for text mining and NLP tasks. We began by implementing two of the simplest models, logistic regression and KNN. For these models, our main goal was to have a baseline that we could use to compare the techniques we used for converting words into numerical vector representations that can be used as features in machine learning models, as previously described.

We then moved on to more complex models, where we performed some very basic **hyper-parameter optimization with *GridSearch*** in order to find the ideal parameters and enhance performance.

Logistic Regression

This is a popular model for text classification tasks. It predicts the probability of an instance belonging to a certain class and works by learning a weight vector that represents the importance of different features (words in our case) in predicting the class label.

K-Nearest Neighbours (KNN)

This is a non-parametric classification algorithm that classifies instances based on their similarity to other instances in the neighbourhood. It works by measuring the distance between instances in the feature space, assigning the class label based on the majority vote of the K nearest neighbours. While easy to implement, its performance can be affected by the curse of dimensionality.

Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron is a type of artificial neural network that consists of multiple layers of interconnected neurons. MLPs detect complex patterns in the input data through backpropagation, and can capture non-linear relationships between features, as well as learning representations from raw text data. For this model some hyperparameter optimization was done, where we tried different combinations of hidden layer sizes, activation functions, solvers and alpha values.

Naive Bayes Classifier

Naive Bayes models are probabilistic classifiers that assume independence between features. They calculate the probability of a document belonging to a class based on the occurrence of different features (words or n-grams) in the document.

Support Vector Machines (SVM)

Support Vector Machines are binary classifiers that aim to find an optimal hyperplane that separates instances of different classes. SVMs can handle high-dimensional data effectively and are robust against overfitting.

Gradient Boosting (GB)

Gradient Boosting is an ensemble method that combines multiple weak learners (e.g., decision trees) to create a strong predictive model. GB iteratively builds a sequence of models, where each subsequent model corrects the mistakes made by the previous models. Gradient Boosting is known for its high predictive performance and ability to handle complex relationships in the data.

Long Short-Term Memory (LSTM)

They are a type of recurrent neural network (RNN) architecture that is designed to effectively capture long-term dependencies in sequential data. It was introduced to address the vanishing gradient problem in traditional RNNs, which struggle to retain and propagate information over long sequences.

We have implemented two LSTM-based architectures. The first architecture is a basic one, while the second architecture incorporates additional complexity with dropout regularization, and more LSTM units. As we have said, both the networks are trained on the GloVe word embeddings, so that the LSTM layers can leverage the learned embeddings to capture the semantic and contextual information within the text.

Evaluation and Results

Methodology

The evaluation of our models was mainly based on their respective confusion matrixes and individual metrics (accuracy, precision, and recall), as well as the **F1-Score**. This is a harmonic mean between precision and recall that uses the following formula: $F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$.

A confusion matrix, on the other hand, reflects the outcome of a classification problem by diving the results into four possibilities: the **True Positives** (1's predicted correctly), **True Negatives** (0's predicted

correctly), **False Positives** (1's predicted wrongly) and **False Negatives** (0's predicted wrongly). Then, based on these four possibilities, we can compute performance metrics, such as the ones previously mentioned.

Accuracy corresponds to the percentage of cases correctly predicted in the total number of cases. It must be used with caution because it can give faulty results when dealing with imbalanced datasets, which happens in our case. **Precision** reflects how many of our predicted positives were actually positive. When the cost of a false positive is very high and the cost of a false negative is low, precision is an effective evaluation metric to use. Finally, **recall** reflects how many of our true positives were identified as such. If a model does not produce false negatives, it will have a recall of 1.

Now that we understand how each of the individual metrics works, we can conclude that, on their own, they all have some limitations. For that matter, it is important to use a more robust metric to evaluate our models, and that is where the F1-Score comes in, incorporating both precision and recall. With this metric, we can limit both false positives and false negatives as much as possible. Therefore, we decided to use it as the ultimate model performance evaluator in the project, optimizing for it as much as possibly, namely on the *GridSearch*.

Results

Looking strictly at the F1-score, the best performing model was the **Multi-Layer Perceptron** with the features resulting from **MWE**, with a score of **0.92** for the label 'Listed' and **0.8** for the label 'Unlisted'. This may be because MLP models are capable of learning complex non-linear relationships between the input features and the target variable. The SBERT embeddings from MWE can provide a high-dimensional input representation, allowing the MLP to capture intricate patterns and dependencies in the data that may not be easily captured by simpler models.

Other models that obtained a performance worth pointing out were the Gradient Boosting and Naïve Bayes. both with MWE, and the Multi-Layer Perceptron using the TF-IDF vectorization. On the negative end, the worst models were, as expected, the most basic ones, Logistic Regression and KNN, as well as SVM. All the results can be found in the annex for further consultation.

Deployment and Prediction on Unseen Data

After determining what our best model was and training it with the labelled data, it was then time to make the predictions on the unseen data. To do so, we first had to replicate all the steps that were done in the train set on the test set to ensure our test records were as similar as possible to the ones we trained the models with. Once that was done, we were able to make the predictions for the labels on this data and save them to a csv containing only two things: the AirBnB index of the 1389 properties present on the test set, and the prediction calculated by our model (1 for unlisted, 0 for listed).

Conclusion

Through the development of this project, we were able to both consolidate the materials learned during the Text Mining course, as well as apply knowledge obtained through self-study. The project presented its challenges, particularly due to the complexity of the problem and the large volume of text data involved. One notable limitation we encountered was the lack of computational power, which posed constraints on processing and analysing such a substantial amount of text. Despite these challenges, the project was a valuable and enriching experience as it allowed us to delve deeper into various topics, engage in comprehensive discussions, and conduct thorough investigations. It

reinforced our understanding of text mining concepts and techniques while providing a practical application of the knowledge gained.

Annex

	precision	recall	f1-score	support
Listed	0.84	0.83	0.84	1807
Unlisted	0.58	0.59	0.59	693
accuracy			0.77	2500
macro avg	0.71	0.71	0.71	2500
weighted avg	0.77	0.77	0.77	2500

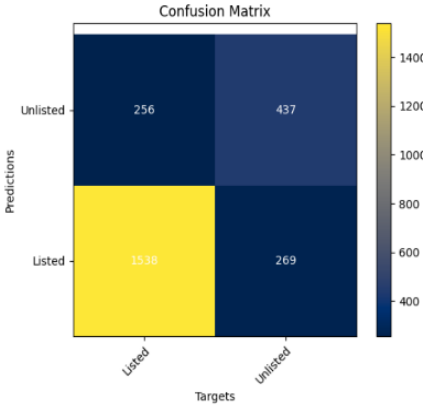

Figure 1 - Results of Logistic Regression with BoW

	precision	recall	f1-score	support
Listed	0.85	0.89	0.87	1807
Unlisted	0.66	0.58	0.62	693
accuracy			0.80	2500
macro avg	0.75	0.73	0.74	2500
weighted avg	0.80	0.80	0.80	2500

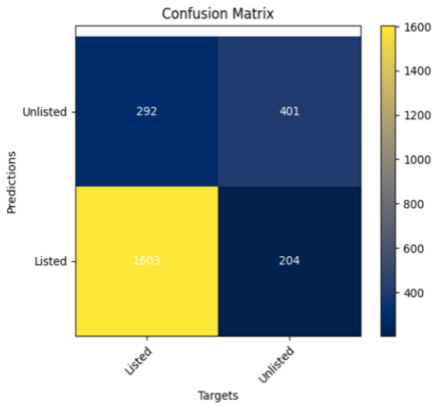

Figure 2 - Results of Logistic Regression with TF-IDF

	precision	recall	f1-score	support
Listed	0.86	0.85	0.85	1807
Unlisted	0.62	0.63	0.62	693
accuracy			0.79	2500
macro avg	0.74	0.74	0.74	2500
weighted avg	0.79	0.79	0.79	2500

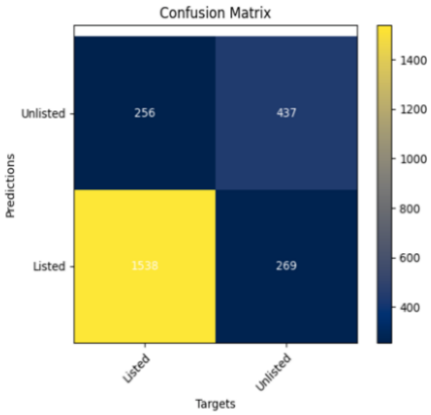

Figure 3 - Results of Logistic Regression with MWE

	precision	recall	f1-score	support
Listed	0.86	0.80	0.83	1807
Unlisted	0.56	0.66	0.60	693
accuracy			0.76	2500
macro avg	0.71	0.73	0.72	2500
weighted avg	0.78	0.76	0.77	2500

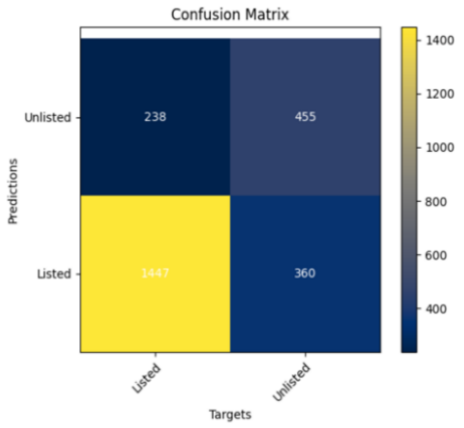

Figure 4 - Results of KNN with BoW

	precision	recall	f1-score	support
Listed	0.86	0.71	0.78	1807
Unlisted	0.48	0.70	0.57	693
accuracy			0.71	2500
macro avg	0.67	0.70	0.67	2500
weighted avg	0.75	0.71	0.72	2500

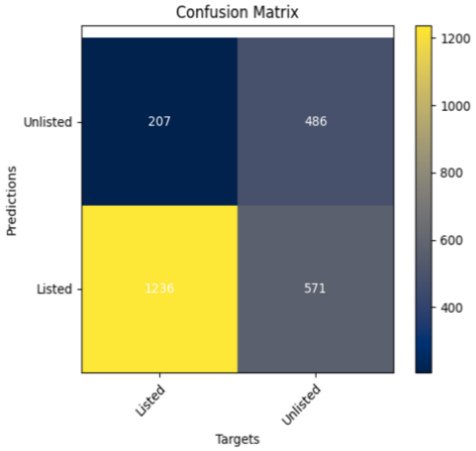

Figure 5 - Results of KNN with TF-IDF

	precision	recall	f1-score	support
Listed	0.88	0.85	0.86	1807
Unlisted	0.63	0.69	0.66	693
accuracy			0.80	2500
macro avg	0.76	0.77	0.76	2500
weighted avg	0.81	0.80	0.81	2500

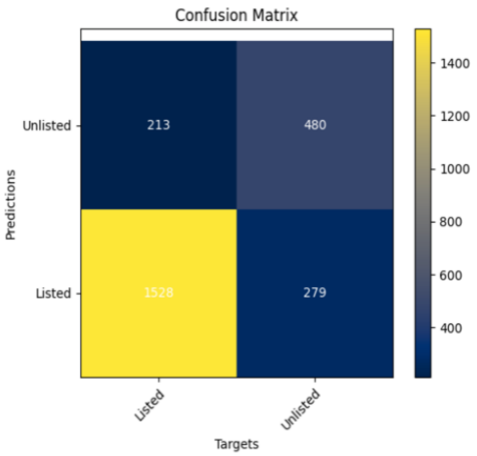

Figure 6 - Results of KNN with MWE

	precision	recall	f1-score	support
Listed	0.90	0.94	0.92	1807
Unlisted	0.81	0.72	0.76	693
accuracy			0.87	2500
macro avg	0.85	0.83	0.84	2500
weighted avg	0.87	0.87	0.87	2500

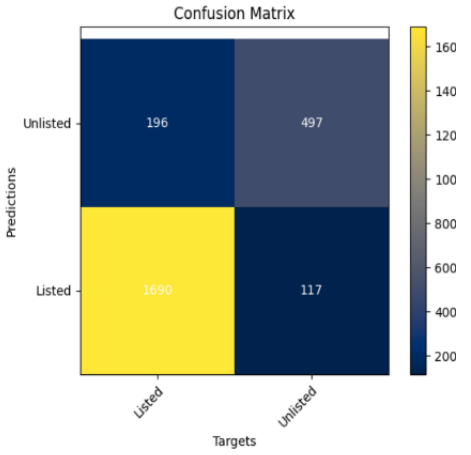

Figure 7 - Results MLP with TF-IDF

	precision	recall	f1-score	support
Listed	0.92	0.93	0.92	1807
Unlisted	0.81	0.79	0.80	693
accuracy			0.89	2500
macro avg	0.86	0.86	0.86	2500
weighted avg	0.89	0.89	0.89	2500

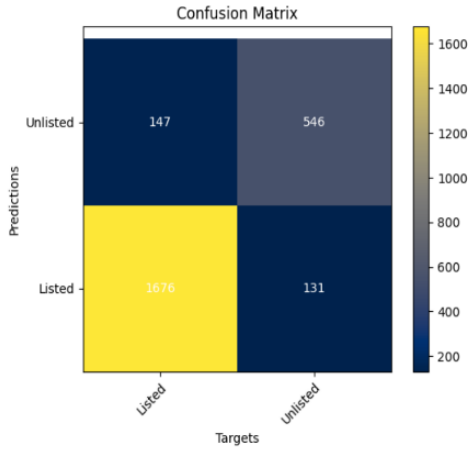

Figure 8 - Results MLP with MWE

	precision	recall	f1-score	support
Listed	0.79	0.94	0.86	1807
Unlisted	0.69	0.36	0.47	693
accuracy			0.78	2500
macro avg	0.74	0.65	0.66	2500
weighted avg	0.76	0.78	0.75	2500

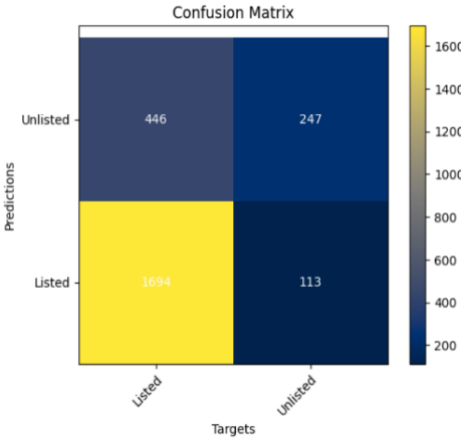

Figure 9 - Results of Naive Bayes with TF-IDF

	precision	recall	f1-score	support
Listed	0.93	0.88	0.90	1807
Unlisted	0.73	0.83	0.77	693
accuracy			0.87	2500
macro avg	0.83	0.85	0.84	2500
weighted avg	0.87	0.87	0.87	2500

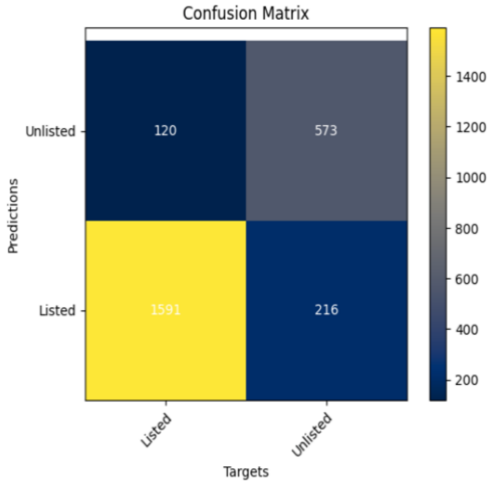

Figure 10 - Results of Naive Bayes with MWE

	precision	recall	f1-score	support
Listed	0.81	0.96	0.87	1807
Unlisted	0.78	0.40	0.53	693
accuracy			0.80	2500
macro avg	0.79	0.68	0.70	2500
weighted avg	0.80	0.80	0.78	2500

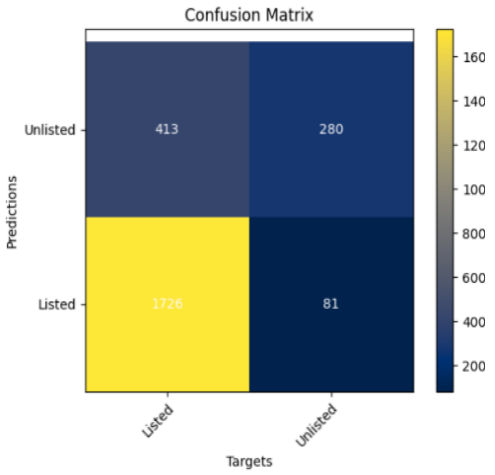

Figure 11 - Results of SVM with TF-IDF

	precision	recall	f1-score	support
Listed	0.85	0.93	0.88	1807
Unlisted	0.75	0.56	0.64	693
accuracy			0.83	2500
macro avg	0.80	0.74	0.76	2500
weighted avg	0.82	0.83	0.82	2500

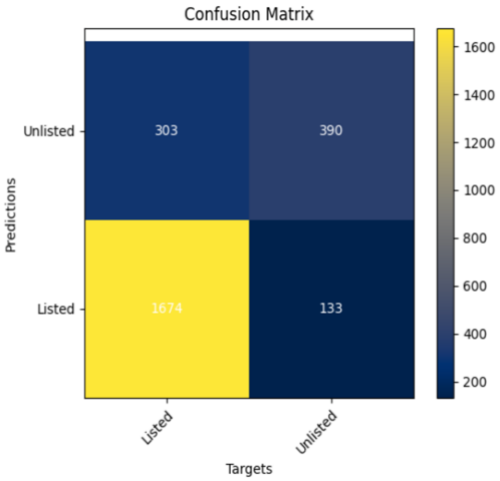

Figure 12 – Results of SVM with MWE

	precision	recall	f1-score	support
Listed	0.84	0.85	0.85	1807
Unlisted	0.60	0.58	0.59	693
accuracy			0.78	2500
macro avg	0.72	0.71	0.72	2500
weighted avg	0.77	0.78	0.77	2500

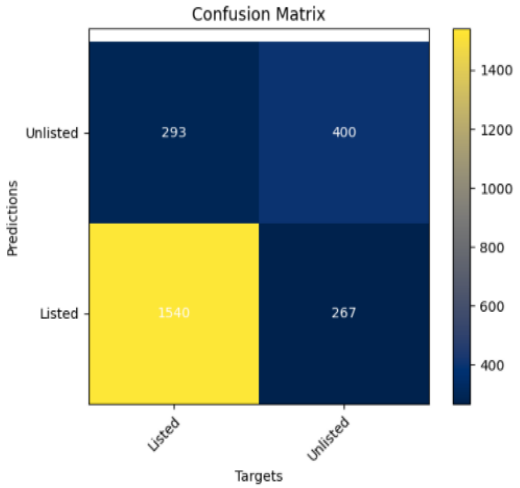

Figure 13 – Results Gradient Boosting with TF-IDF

	precision	recall	f1-score	support
Listed	0.94	0.89	0.91	1807
Unlisted	0.75	0.84	0.79	693
accuracy			0.88	2500
macro avg	0.84	0.87	0.85	2500
weighted avg	0.89	0.88	0.88	2500

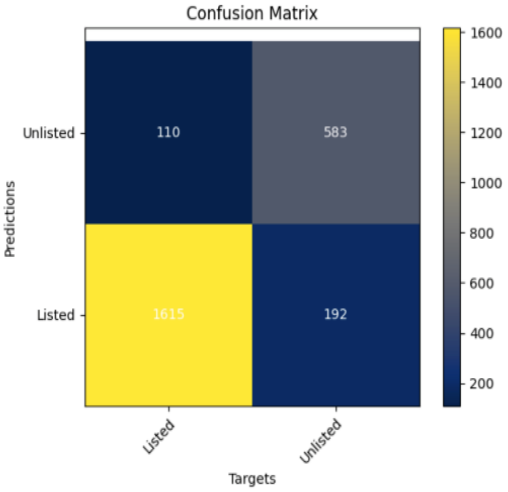

Figure 14 - Results Gradient Boosting with MWE

```
Epoch 20/20
125/125 [==============================] - 2s 16ms/step - loss: 0.5072 - f1_score: 0.7496 - val_loss: 0.7057 - val_f1_score: 0.6517
```

Figure 15 - Results of LSTM2 with GloVe

References

1. Reimers, N. (2019, August 27). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv.org. https://arxiv.org/abs/1908.10084