Text Mining

Project Report – Predicting Airbnb Listings

Authors:

Andriani Kakoulli (20230484) 20230484@novaims.unl.pt
Kenza Balsam Boukhris (20230604) 20230604@novaims.unl.pt
Michelle Buston Vega (20230590) 20230590@novaims.unl.pt
Pedro Paris (20191217) 20191217@novaims.unl.pt

Abstract

This report presents a comprehensive study using Natural Language Processing (NLP) techniques to predict the de-listing of properties on Airbnb in the subsequent quarter. Using datasets with property descriptions, host details, and guest reviews, we applied various text mining and machine learning models to develop predictive models. Our methodology included data preprocessing, feature engineering, and implementing multiple classification models. Techniques such as TF-IDF, GloVe embeddings, and Doc2Vec transformed textual data into actionable insights, while models like Logistic Regression, Multilayer Perceptron (MLP), Random Forest, SVM, XGBoost, LSTM, and Neural Networks were used for prediction. The evaluation focused on precision, recall, and F1-score, considering the dataset's imbalance.

The results showed that TF-IDF with Random Forest yielded the most effective outcomes, emphasizing the importance of appropriate feature engineering in text-based predictive modeling. Future directions might include exploring advanced deep learning models and additional feature incorporation to enhance predictive accuracy further. This project demonstrates the capabilities of NLP in real-world applications and provides a blueprint for similar studies leveraging textual data for predictive analytics.

Contents

1 Introduction

The goal of this project is to leverage Natural Language Processing (NLP) techniques to predict whether a property listed on Airbnb will be unlisted in the next quarter. To achieve this, we will analyze real-world data including Airbnb property descriptions, host descriptions, and guest reviews. This involves implementing various text processing and machine learning techniques to extract meaningful patterns and insights from the textual data provided. Throughout this project, we will apply the NLP techniques learned during the Text Mining Lecture but also some extra methods to preprocess the data, engineer on on relevant features, and build robust models.

2 Data Exploration

2.1 The Dataset

The first step in any scientific approach to a dataset is to understand the characteristics of the data we are dealing with. Our dataset is divided into four distinct subsets:

- **Train Set (train.xlsx):**
  - **Size:** 6,248 entries
  - **Attributes:** `index`, `description`, `host_about`, `unlisted`
  - **Description:** This dataset contains the descriptions of Airbnb properties and hosts. The `description` column provides details about the properties, while the `host_about` column contains information about the hosts. The `unlisted` column is the target variable, indicating whether a property was removed from the Airbnb list in the next quarter (`1` for unlisted, `0` for listed).
- **Train Reviews (train_reviews.xlsx):**
  - **Size:** 361,281 entries
  - **Attributes:** `index`, `comments`
  - **Description:** This dataset includes guest comments for the Airbnb properties in the training set. There can be multiple comments per property, comments may be in various languages, and not all properties have associated comments.
- **Test Set (test.xlsx):**
  - **Size:** 695 entries
  - **Attributes:** `index`, `description`, `host_about`
  - **Description:** The structure of this dataset is similar to the train set but without the `unlisted` column. This set is used to evaluate our model's predictions. The true labels are withheld by the teaching team and will be used to assess the accuracy of our predictions.
- **Test Reviews (test_reviews.xlsx):**
  - **Size:** 41,866 entries
  - **Attributes:** `index`, `comments`
  - **Description:** This dataset contains guest comments for the properties in the test set. The structure is identical to the train reviews set.

2.2 Initial Observations

The `description` and `host_about` columns are filled with textual data that vary in length and detail. These columns provide a rich source of information about the properties and hosts, which is crucial for our analysis. The column `comments` contains reviews in multiple languages and vary in length and sentiment, which makes them a valuable but challenging component to analyze.

Our Word-Count-Analysis for the `description` column shows that the longest descriptions are detailed, providing extensive information about the properties. The shortest descriptions often contain minimal information, such as license numbers, indicating a need for data validation and cleaning. For a detailed view of word counts, please refer to Table 6 in the appendix.

The top 25 most common words in the `description` column were identified, as shown in Figure 3. This figure reveals that common stop words such as "the", "and", "a", and HTML tags like `<br>` are among the most frequent words in the descriptions. This underscores the importance of preprocessing steps.

Furthermore, we can say that there are only two missing values in the `comments` column of the Training Reviews dataset. The rest of the data has no missing values.

Finally, we take a look at the balance of our target variable. An imbalanced dataset can be worrisome when predicting, as it will always be biased towards predicting the most common value of the target variable. In our `unlisted` target variable, we have 4540 'listed' to 1708 'unlisted', which gives us a class imbalance ratio of about 5:2. This class imbalance between listed and unlisted properties will be addressed during model training.

3 Data Preprocessing

3.1 Tokenization

Tokenization involves breaking down text into smaller units, typically words or phrases, known as tokens. This step is essential for converting textual data into a format that can be easily analyzed and processed by machine learning algorithms. The process of tokenization and text preprocessing in our project involves several key steps:

Firstly, we created a preprocessing function that handles various text cleaning and normalization tasks. This function removes HTML tags using the BeautifulSoup library, handles contractions (such as converting "can't" to "cannot"), removes non-alphabetic characters and URLs, and converts all text to lowercase to ensure uniformity. Once the text is cleaned, it is then split into individual words or tokens. This is done by splitting the text string using spaces as delimiters.

Additionally, the function removes commonly used words that do not carry significant meaning, known as **stop words** (such as "and", "the", "is") to focus on more meaningful words.

Next, **lemmatization** was applied to the text, which reduces words to their base or root form (for example, "running" to "run"). This step helps in normalizing the text and ensuring that different forms of a word are treated as a single item.

Stemming is also applied, which reduces words to their base form in a more aggressive manner (such as "running" to "runn").

After defining the preprocessing function, we proceed to handle the `comments` from the `train_reviews` and `test_reviews` datasets. The comments are treated as strings, grouped by property index, and concatenated into a single text string for each property, providing a comprehensive view of all comments related to each property.

Next, we merge the aggregated reviews with the main property data from the `train` and `test` datasets using the property index. This step ensures that all relevant information, including descriptions, host information, and guest comments, is combined into a single dataset. Subsequently, the `description`, `host_about`, and aggregated `comments` columns are combined into a single `full_text` column, providing a richer context for analysis and modeling.

Finally, we apply the previously defined preprocessing function to the `full_text` column. This step cleans and normalizes the text, ensuring it is ready for further analysis and modeling. By applying the preprocessing function to the combined text column, we ensure the text data is consistently processed and standardized.

3.2 Sentiment Analysis with VADER (Extra)

Sentiment analysis is performed on the `comments` from the `train_reviews` dataset using VADER (Valence Aware Dictionary and Sentiment Reasoner). It is a tool specifically designed for analyzing sentiment in text. The goal is to determine the emotional tone behind the comments, which can provide insights into guest experiences and potentially influence our predictive models. First, the comments had to be converted to string format. Afterwards, a function to calculate sentiment scores for each comment was designed. Based on its compound score, each comment was classified as positive, negative, or neutral. To analyze the distribution of sentiment scores for listed and unlisted properties and identify any patterns or differences in guest sentiment, comments were grouped by property index and the number of each sentiment category for each property was counted. It shows that positive comments are predominant for both listed and unlisted properties, with a slight variation in proportions (Table 1). However, since sentiment analysis did not show a significant impact on the target variable, it will not be used in the final model predictions.

Sentiment	**Listed Properties**	**Unlisted Properties**
Positive	190,148 (72.90%)	73,339 (73.21%)
Neutral	48,906 (18.75%)	18,570 (18.54%)
Negative	21,778 (8.35%)	8,265 (8.25%)

Table 1: Sentiment count and proportions of listed and unlisted properties

3.3 Translation (Extra)

Given the multilingual nature of the dataset, translating non-English text into English is essential for ensuring uniformity and improving the effectiveness of subsequent analyses. We use the DeepL API for translation, ensuring that the necessary libraries and API keys are set up. The process begins with detecting the language of each text entry using `langdetect`.

Initially, we attempted to merge the `description` and `host_about` columns into a single column for language detection and translation. However, we found that this approach caused us to lose information regarding the specific language of each part. Therefore, we decided to translate the `description` and `host_about` columns separately, which, although it increases runtime, preserves the accuracy of language detection.

Next, we define a function to translate text and apply it to non-English text entries. By using language detection, we translate only non-English text, significantly reducing processing time. This step ensures that all text data is uniformly in English, facilitating more effective analysis and modeling. Finally, we apply the optimized translation process to the entire dataset.

Efforts were made to translate the `train_reviews` and `test_reviews` datasets. However, the sheer volume of data proved to be a significant challenge. The `train_reviews` dataset is approximately 60 times larger than the `train` dataset, making the translation process extremely time-consuming. Despite running the translation process for many hours, we were unable to complete the translation for the entire dataset. Additionally, the kernel often disconnected due to the extensive processing requirements.

3.4 Exploration and Visualization after Preprocessing

After completing the tokenization and preprocessing steps, we further explore and visualize the text data to gain additional insights. This section presents an analysis of word counts, the most common words, bigrams, and a word cloud visualization of the `full_text` column.

Word Count Analysis

To understand the distribution of text length in the `full_text` column, we calculated the word count for each entry. The table below shows the top 5 and bottom 5 entries based on word count:

Index	Full Text	Word Count
264	offer comfort renov bedroom apart bairro alto ...	28,166
1146	welcom lot light right heart lisbon apart tota...	27,656
3728	alfama blue hous ii small charm apart alfama o...	24,387
4105	best view almost lisbon coolest flat great loc...	21,793
953	spacethi last floor apart near year old build ...	20,216
2469	quiet good room sympatico friend respons	6
1944	bedroom doubl bed nice polit	5
5682	licens number al lxbestsuit	4
5739	licens number al lxbestsuit	4
3872	licens number al lxbestsuit	4

Table 2: Word Count for top 5 and bottom 5 in full text

The entries with the highest word counts tend to have detailed descriptions and numerous comments, providing a comprehensive view of the property. In contrast, entries with the lowest word counts have minimal information.

Word Cloud

The word cloud offers a quick visual summary of the most important words in the dataset. It was generated to provide a visual representation of the most frequently occurring words in the `full_text` column. The word cloud is shown bellow. Words like "great", "locat" (location), "high", "recommend", "walk", and "distance" are prominently featured, suggesting that positive attributes and recommendations are common themes in the text data.

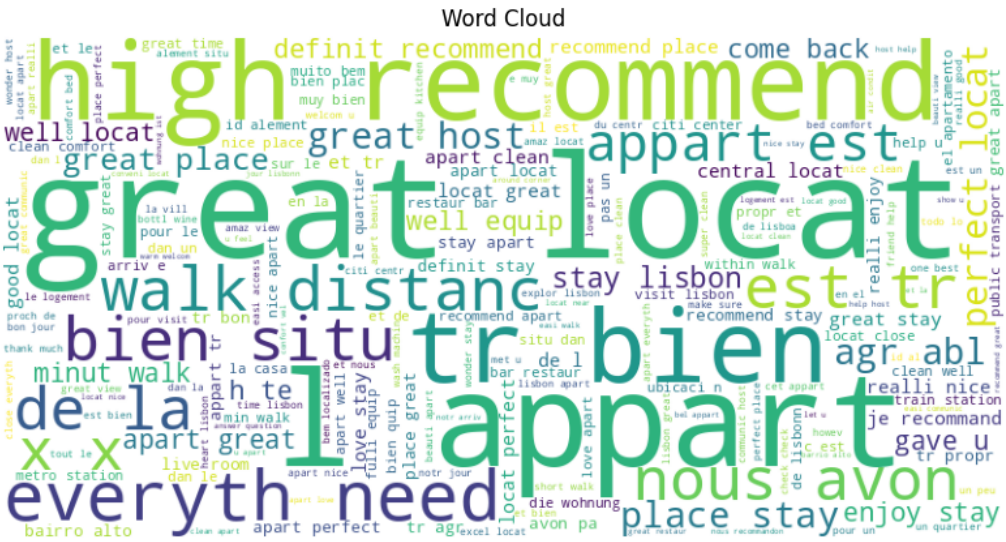

Figure 1: Word Cloud of Full_Text Column

Most Common Words and Bigrams

The top 25 most common words and top 20 most common bigrams (pairs of consecutive words) in the `full_text` column were identified. These visualizations are presented in Figures 2 and 3 in the appendix. The most common words and bigrams provide insights into the key themes and sentiments expressed in the text data. Similar to the word cloud, they show that positive attributes such as "great location" and recommendations are frequently mentioned, indicating the importance of these aspects in property reviews.

3.5 Balancing the Dataset with SMOTE

Recognizing the challenge posed by the class imbalance in our dataset, where most Airbnb listings were labeled as 'listed,' we implemented the Synthetic Minority Over-sampling Technique (SMOTE) to enhance the robustness of our predictive models. This technique synthetically generates new examples in the minority class, 'unlisted,' by interpolating existing examples. This approach not only helps to balance the class distribution but also aids in mitigating the model's bias towards the majority class.

By integrating SMOTE into our preprocessing pipeline, we ensured that both classes were equally represented, enhancing the generalization capability of our machine learning models. This crucial preprocessing step was applied right after the initial data preparation and before the data was split into training and testing subsets. This strategic placement ensures that the synthetic data does not leak into the testing phase, maintaining our evaluation's integrity and challenging nature.

4 Feature Engineering

4.1 TF-IDF

We use Term Frequency-Inverse Document Frequency (TF-IDF) to identify the most important words in the `full_text` column of our dataset. TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. It helps in highlighting words that are significant within specific documents while reducing the weight of commonly used words that are less informative.

The TF-IDF process begins by transforming the `full_text` data into numerical features using the TfidfVectorizer from the Scikit-Learn library. We adjust the max_features parameter to limit the number of features based on computational resources, setting it to 25 in this case. This step converts the text data into a TF-IDF matrix, where each entry represents the TF-IDF score of a word in a document. Since the TF-IDF matrix is initially sparse (i.e., it contains many zeros), we convert it to a dense array to facilitate further analysis. This conversion ensures that the data is in a suitable format for subsequent steps.

The bar plot bellow (Figure 4) shows the top 25 terms based on their TF-IDF scores for the first document. Words like "de", "stay", "nice", "clean", "place", "good", and "great" have the highest scores, indicating their significance in the context of that specific document.

4.2 GloVe Word Embeddings

To enhance our text analysis, we employed GloVe (Global Vectors for Word Representation) to generate word embeddings. Word embeddings are a type of word representation that allows words to be represented as vectors in a continuous vector space, capturing semantic meanings and relationships between words. Using the torchtext library, we loaded pre-trained GloVe vectors from the 6B dataset with 50 dimensions, containing 400,000 words. These vectors offer rich word representations based on large corpus usage. We defined the get_word_vector function to retrieve GloVe vectors for given words. The document_to_vectors function maps each token in a document to its GloVe vector, appending available vectors to a list and padding sequences to ensure uniform length. The corpus_to_glove function converts the entire corpus into vector representations using document_to_vectors, padding the sequences to a specified maximum length.

We applied corpus_to_glove to the `full_text` column of the training dataset, converting the text data into a 3-dimensional array of GloVe vectors. This array was then transformed into a NumPy array for easier manipulation. Finally, we flattened the 3-dimensional array into a 2-dimensional one, making it suitable for

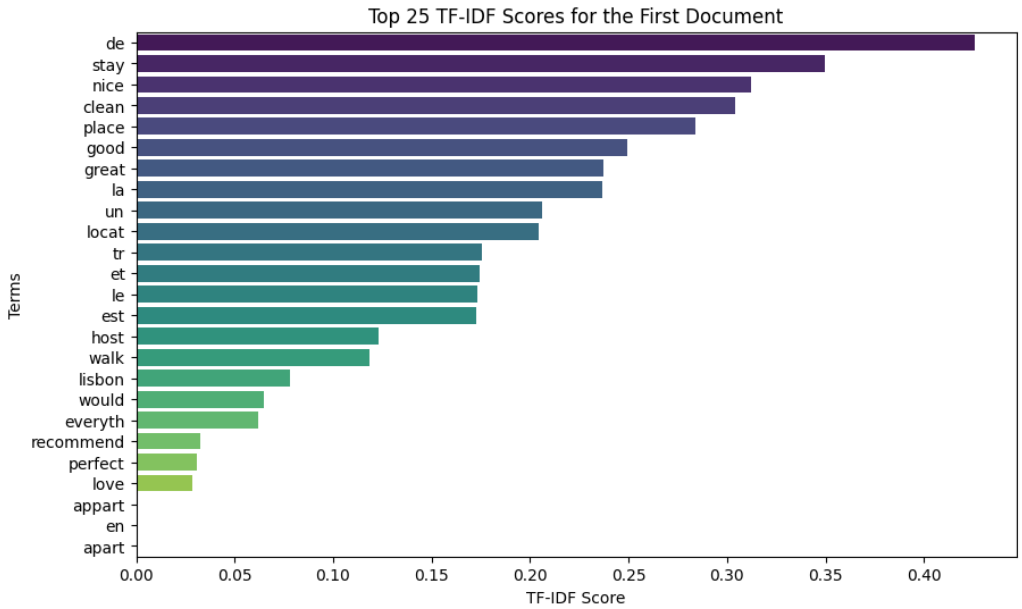

Figure 2: Top 25 TF-IDF Scores for the First Document

machine learning models, with each document represented as a fixed-length vector. The shape of the final flattened array is (6248, 1250), indicating that each of the 6248 documents is represented by a 1250-dimensional vector. This transformation enables us to leverage the semantic richness of GloVe embeddings for downstream tasks such as classification.

4.3 Doc2Vec

In addition, we employed the Doc2Vec model to generate document embeddings. Doc2Vec, an extension of Word2Vec, allows us to create vector representations for entire documents rather than individual words. This technique is particularly useful for capturing the semantics of a whole text.

We began by importing the necessary libraries, including `Doc2Vec` and `TaggedDocument` from `gensim.models.doc2vec` and `word_tokenize` from `nltk.tokenize`, and downloading the NLTK tokenizer models. Each document in the `full_text` column was preprocessed by converting text to lowercase and tokenizing it into individual words using `word_tokenize`. Each document was then tagged with a unique identifier using `TaggedDocument`.

We initialized the `Doc2Vec` model with a vector size of 25, a minimum word count of 2, and 50 training epochs. The `build_vocab` method constructed the vocabulary from the tagged documents, and the model was trained using the `train` method. This process iterates through the documents, adjusting model weights to learn document representations.

After training, the `Doc2Vec` model was used to infer vectors for each document in the `full_text` column. The `infer_vector` method applied to the tokenized version of each document provided its vector representation. The resulting document vectors were then converted to a NumPy array for easier manipulation.

By employing `Doc2Vec`, we transformed our textual data into dense vector representations that capture semantic relationships and overall context. This complements GloVe word embeddings, offering a richer understanding of the text.

4.4 XLM_RoBERTa (Extra)

In our exploration of advanced feature engineering techniques, we also experimented with RoBERTa (Robustly Optimized BERT Pretraining Approach) to enhance our text representation capabilities. Similar to our experiences with Doc2Vec, we encountered significant challenges with RoBERTa, primarily due to its extensive

computational demands. During the feature engineering phase, using RoBERTa significantly increased the runtime, ultimately leading to frequent kernel disconnections. This made it impractical to complete the process within the available infrastructure, and as a result, we were not able to integrate RoBERTa into our final modeling workflow fully. This exploration highlighted the limitations of our current setup when dealing with highly sophisticated NLP models in a resource-constrained environment.

5 Classification Models

Before training the models, we performed an additional **data split** into training and validation sets. The `train_test_split` function from Scikit-Learn was used for this purpose, with 20 % of the data reserved for validation. This split ensures that we can evaluate the model's performance on unseen data, helping to prevent overfitting and providing a more reliable estimate of the model's generalization ability.

We implemented several models from our text mining class, including Logistic Regression, Multilayer Perceptron (MLP), and Long Short-Term Memory (LSTM). Additionally, we explored extra models such as Random Forest, Support Vector Machine (SVM), XGBoost, and a basic Neural Network. This diverse set of models allowed us to compare traditional text mining approaches with more advanced machine learning techniques, ultimately aiming to identify the most effective model for our classification problem.

Each of these models offers unique strengths. Logistic Regression provides a simple and interpretable baseline. MLP and LSTM harness the power of neural networks for capturing complex patterns in the data. Random Forest and XGBoost leverage ensemble learning for robust predictions. SVM offers strong performance for binary classification tasks. The basic Neural Network model serves as an entry point into more complex deep learning approaches.

5.1 Logistic Regression

Logistic Regression is a straightforward and widely used classification algorithm that models the probability of a binary outcome based on one or more predictor variables. We used the Logistic Regression class from Scikit-Learn with default settings, which include the 'lbfgs' solver and no regularization. This model serves as a baseline for comparison with more complex models.

5.2 Multilayer Perceptron (MLP)

MLP is a type of feedforward artificial neural network that consists of multiple layers of nodes. We used the MLPClassifier from Scikit-Learn, specifying the 'adam' solver, 'logistic' activation function, and hidden layers with sizes (2, 2). This configuration includes two hidden layers with two neurons each, and a random state was set for reproducibility.

5.3 Long Short-Term Memory (LSTM)

LSTM is a type of recurrent neural network (RNN) capable of learning long-term dependencies. For our implementation, we used Keras with TensorFlow backend. The model included an embedding layer for word embeddings, an LSTM layer with 100 units, and a dense layer with a sigmoid activation function for binary classification. The model was compiled with the 'adam' optimizer and 'binary_crossentropy' loss function.

5.4 Random Forest (Extra)

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training. We used the RandomForestClassifier from Scikit-Learn with parameters including the number of estimators (100 and 200), max depth (None, 10, 20), min samples split (2, 5), and min samples leaf (1, 2). GridSearchCV was employed to find the best combination of these parameters.

5.5 Support Vector Machine (Extra)

SVM is a supervised learning model that analyzes data for classification and regression analysis. We used the SVC class from Scikit-Learn with a grid search over parameters such as C (0.1, 1, 10), kernel ('linear', 'rbf'), and gamma ('scale', 'auto'). This model is particularly effective for high-dimensional spaces.

5.6 XGBoost (Extra)

XGBoost is an optimized gradient boosting library designed to be highly efficient and flexible. We used the XGBClassifier from the XGBoost library, performing a grid search over parameters including the number of estimators (100, 200), max depth (3, 6, 10), learning rate (0.01, 0.1, 0.2), and subsample (0.8, 1.0). XGBoost's efficiency and scalability make it suitable for large datasets.

5.7 Neural Network (Extra)

A basic Neural Network was implemented using Keras with TensorFlow backend. The architecture included an input layer size of 500, one hidden layer with 512 neurons and 'ReLu' activation, and an output layer with a single neuron and 'sigmoid' activation. The model was compiled with the 'adam' optimizer and trained over 5 epochs with a batch size of 64.

6 Evaluation and Results

In evaluating our models on the validation set, we focused on several key metrics: precision, recall, accuracy, and F1-score. Due to the imbalance in our target variable, the F1-score was prioritized over accuracy, as it provides a better measure of a model's performance on imbalanced datasets. Specifically, the weighted F1-score was used, which calculates the F1-score for each class and gets the weighted average using the proportion of each class in the dataset. This approach ensures that the model's performance on the under-represented class is adequately considered.

We compared the before mentioned feature engineering methods to determine which would provide the best input for our models. This comparison is crucial because the choice of feature engineering method can significantly impact the performance of machine learning models.

TF-IDF (Term Frequency-Inverse Document Frequency) performed exceptionally well across all models, consistently providing high precision, recall, and F1-scores, showcasing its effectiveness in capturing the importance of words within documents relative to the entire corpus. The standout models using TF-IDF were Random Forest, which achieved one of the highest F1-scores at 0.862, and LSTM, which also had a high F1-score of 0.862 but with a significantly longer runtime. Overall, TF-IDF provided a balanced combination of performance and efficiency, making it a reliable choice for text feature extraction. The exact metrics for each model can be seen in Table 3.

GloVe (Global Vectors for Word Representation) did not perform as well as TF-IDF across the models, particularly in models with high runtime. As a result, we only reported the initial models (LR, MLP, RF, NN). The precision, recall, and F1-scores were generally lower, indicating that GloVe embeddings might not capture the specific nuances needed for this classification task as effectively as TF-IDF. However, Random Forest stood out with a respectable F1-score of 0.723, though it required a much longer runtime. Overall, while GloVe embeddings offer rich word representations, they were less effective for our specific text classification needs compared to TF-IDF. The exact metrics for each model can be seen in Table 4.

Doc2Vec (Document to Vector) performed reasonably well on some models. SVM and XGBoost both achieved an F1-score of 0.85, indicating their ability to leverage Doc2Vec's document-level embeddings effectively. However, Random Forest also performed well with an F1-score of 0.83. Due to high runtime and kernel disconnections, we did not further pursue Doc2Vec models after initial tests. The exact metrics for each model can be seen in Table 5.

The final comparison highlighted that TF-IDF was the most efficient and effective feature engineering method. It consistently yielded the best results across different models, with the Random Forest and SVM models performing particularly well in terms of F1-score and runtime balance. This comprehensive evaluation allowed us to identify the most suitable models for predicting Airbnb listing status, with a clear preference for the Random Forest model using TF-IDF features.

Model	Precision	Recall	F1-Score	Support	Runtime
Logistic Regression	0.858220	0.857379	0.857272	1250.0	0.033769
MLP	0.851342	0.851322	0.851315	1250.0	3.563473
Random Forest	0.888450	0.888216	0.888190	1250.0	23.705324
SVM	0.882710	0.882709	0.882710	1250.0	20.108435
XGBoost	0.887210	0.887115	0.887102	1250.0	47.654690
LSTM	0.845799	0.847200	0.846424	1250.0	283.323527
Neural Network	0.83	0.81	0.82	1250.0	3.0

Table 3: TF-IDF Results

Model	Precision	Recall	F1-Score	Support	Runtime
Logistic Regression	0.659918	0.6784	0.667452	1250.0	2.365106
MLP	0.661244	0.6744	0.666991	1250.0	13.817462
Random Forest	0.797643	0.7760	0.723084	1250.0	1294.301240
SVM	-	-	-	-	-
XGBoost	-	-	-	-	-
LSTM	-	-	-	-	-
Neural Network	0.706936	0.7320	0.711458	1250.0	14.648279

Table 4: GloVe Results

Model	Precision	Recall	F1-Score	Support	Runtime
Logistic Regression	0.77	0.77	0.77	1250.0	0.14
MLP	0.81	0.78	0.79	1250.0	2.33
Random Forest	0.83	0.84	0.83	1250.0	214.19
SVM	0.85	0.84	0.85	1250.0	80.58
XGBoost	0.84	0.84	0.84	1250.0	156.37
LSTM	-	-	-	-	-
Neural Network	-	-	-	-	-

Table 5: Doc2Vec Results

7 Conclusions

This project aimed to predict whether a property listed on Airbnb will be unlisted in the next quarter using various Natural Language Processing (NLP) techniques. By analyzing real-world data, including Airbnb property descriptions, host descriptions, and guest reviews, we built robust predictive models and gained several key insights.

Among the feature engineering methods, **TF-IDF** consistently provided the best results across various models, demonstrating high precision, recall, and F1-scores, and thus it was the concluded Feature Engineering method to be used in modeling. It effectively captured the importance of words within documents, making it a reliable choice for text feature extraction. In contrast, GloVe embeddings, while offering rich word representations, were less effective for this specific classification task. The performance with GloVe was generally lower, and the Random Forest model, although notable, came with significantly longer runtime. Doc2Vec aimed to create dense vector representations for entire documents, capturing the overall context. However, practical issues such as high runtime and kernel disconnections limited its usability, similarly as with XLMRoBERTa. Despite these challenges, some models like SVM and XGBoost performed well with Doc2Vec, indicating the method's potential.

In terms of model performance, the Random Forest and SVM models were the top performers, particularly when using TF-IDF features, which was the concluded Feature Engineering technique. These models balanced high F1-scores with reasonable runtime, making them suitable for predicting Airbnb listing status. Logistic Regression, while simple, provided a solid baseline, and MLP and LSTM models demonstrated the power of neural networks, though LSTM had a significantly longer runtime. XGBoost and basic Neural Networks also showed robust performance, highlighting the effectiveness of ensemble learning and neural network approaches. Based on runtime and metrics, the chosen model for predicting Airbnb listing status was **Random Forest**.

Several challenges and limitations were encountered throughout the project. The dataset's imbalance between listed and unlisted properties was a significant hurdle, which we addressed through careful evaluation metrics, prioritizing F1-scores to ensure balanced model performance. Handling multilingual data and varied sentiment in guest comments required additional preprocessing steps. Translation was particularly challenging due to the volume of data, and sentiment analysis did not significantly impact model performance.

In conclusion, this project demonstrated the potential of NLP techniques and various machine learning models in predicting Airbnb listing status. The comprehensive evaluation and comparison of feature engineering methods and models provide valuable insights for future applications in similar text mining tasks.

8 Future Work

For future work, exploring more advanced NLP techniques, such as transformer-based models like BERT, could potentially enhance model performance further. Additionally, incorporating extra features, such as property location and host experience, could provide more context and improve prediction accuracy. These features could help capture the broader context in which a property operates, considering factors like neighborhood desirability and host responsiveness. Integrating these elements with our current approach could offer a more holistic view and yield better results.

A Appendix

Index	Description	Word Count
3549	The villa in Ericeira has 2 bedrooms and has c...	210
2046	Sunny apartment 10 m from beach and train stat...	205
4256	The villa in Ericeira has 3 bedrooms and has c...	204
1367	Located in between Lisbon and Cascais, the Cas...	202
3154	This is a bright and design apartment in a new...	198
5243	License number Exempt	3
2999	License number 105154/AL	3
4306	License number 18235/AL	3
5769	License number 45360/AL	3
5682	License number 7797/AL	3

Table 6: Word Count for for top 5 and bottom 5 in description

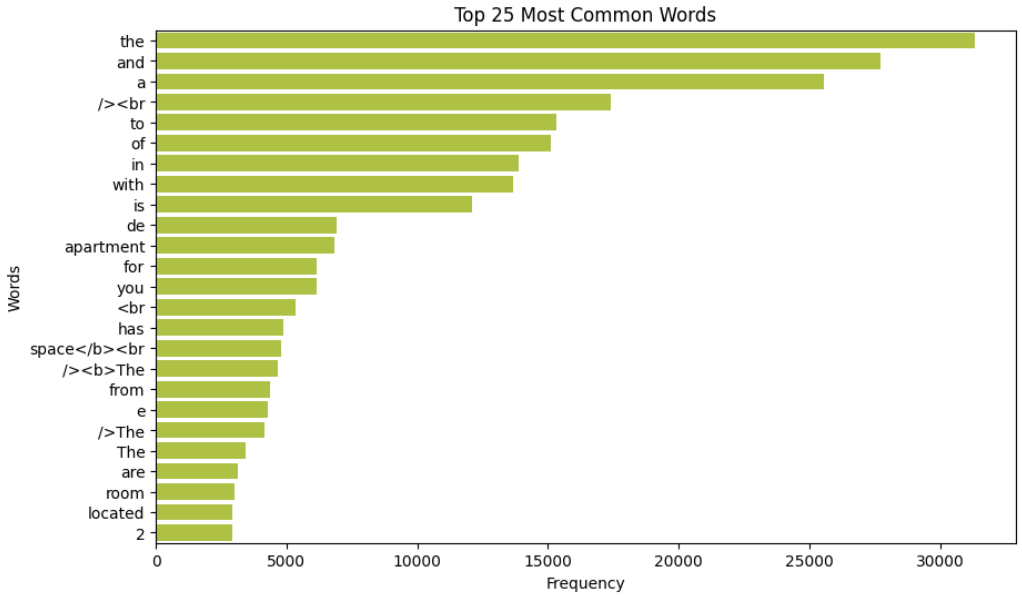

Figure 3: Top 25 Most Common Words in Description Column

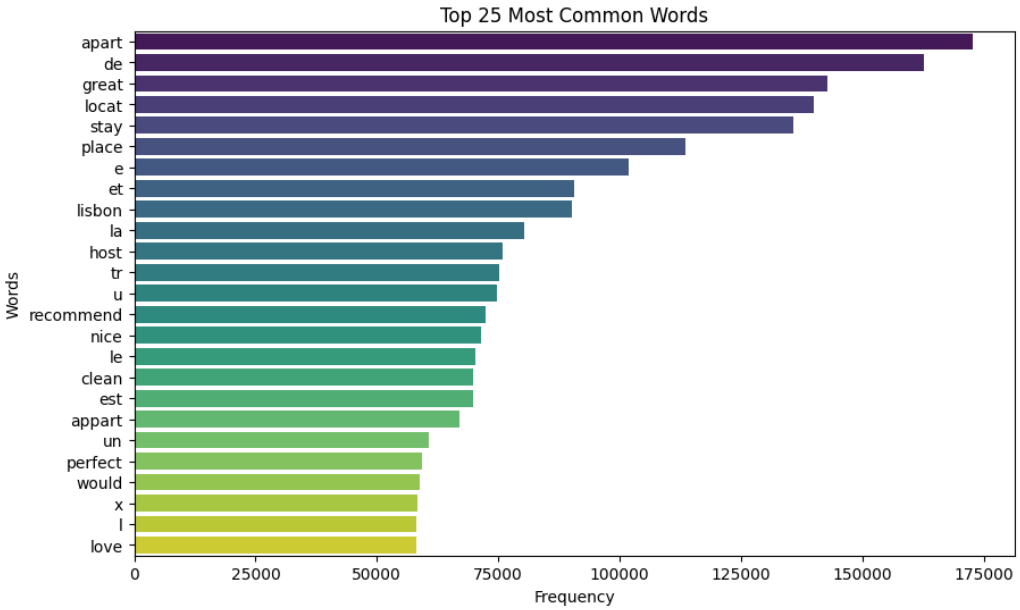

Figure 4: Top 25 Most Common Words in Full_Text Column

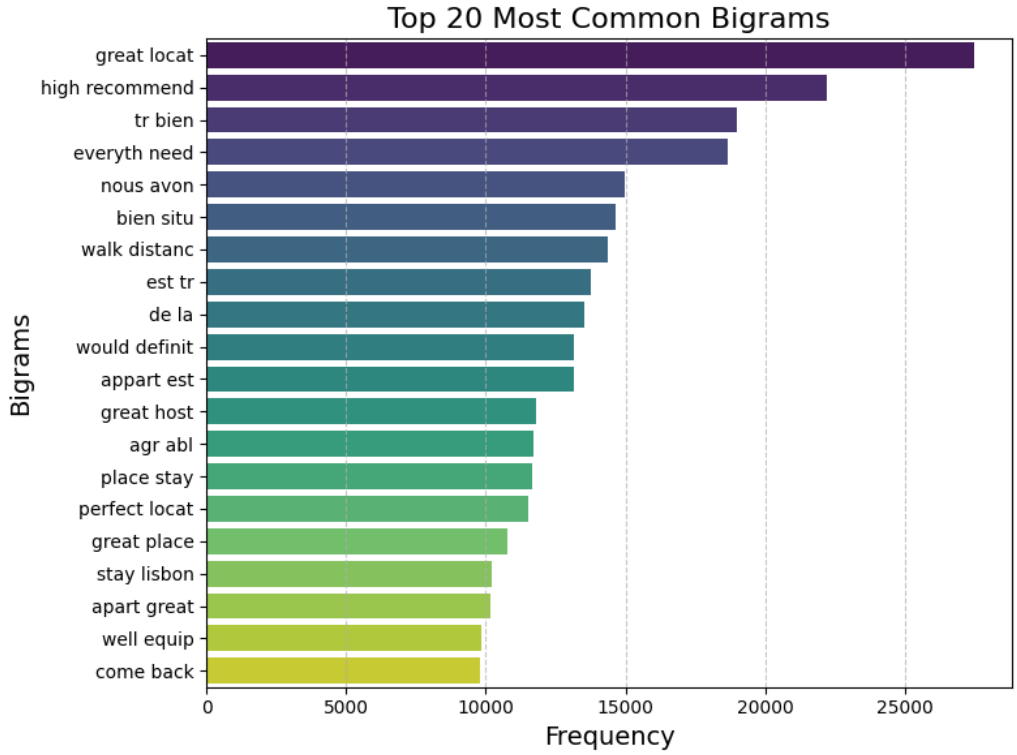

Figure 5: Top 20 Most Common Bigrams in Full_Text Column