NOVA IMS Information Management School

TEXT MINING

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Forecasting of Airbnb Listing Status

Afonso Anjos, 20230527

Filipe Pereira, 20230445

João Machado, 20230426

June, 2024

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

Index

Abstract

This project leverages Natural Language Processing (NLP) techniques to predict whether an Airbnb property will be unlisted in the next quarter. Using data from Airbnb property descriptions, host descriptions, and guest reviews, we developed a classification model to determine the listing status of properties. We employed two primary approaches: extracting sentiment-based numerical features from guest reviews and combining them with textual data from property descriptions and host information; Utilizing comprehensive textual features by concatenating the description, host information and all guest reviews into a single text variable for each property.

Our methodology encompassed extensive data exploration, preprocessing, feature engineering, and the implementation of various classification models. We experimented with multiple feature engineering techniques, including Bag of Words (BoW), TF-IDF, Word2Vec, GloVe, and advanced transformer-based embeddings like XLM-RoBERTa and Voyage AI. The evaluated models included Logistic Regression (LR), Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), BERT, and XLM-RoBERTa.

Evaluation was based on standard metrics such as precision, recall, F1-score, and accuracy. Our best-performing model, a Logistic Regression using TF-IDF (trigram) and SpaCy preprocessing, achieved an accuracy of 0.89, precision 0.90, recall 0.89 and f1-score of 0.89. This model was subsequently used for predictions on the test set, demonstrating the effectiveness of our comprehensive approach in capturing nuanced textual and numerical features for accurate prediction.

Data Exploration

For this task, we started by checking the missing values for the different datasets, where we detected that the variable "comments" of the reviews dataset had two missing values, so we chose to delete them. We then checked for duplicated rows across all the data frames we had and found that both the train and test reviews datasets had 142 and 15 duplicated rows each. With a deeper analysis, we concluded that these represented distinct reviews, meaning that one property could have multiple duplicated reviews from different people, so we chose to preserve these rows.

We also took a closer look at the train and test datasets, where we filtered the rows by the variables "description" and "host_about" and found that there were 341 instances where these two variables were duplicated with other rows. Thus, we concluded that many different properties had the same texts in these two variables, but since they represented different properties, we didn’t delete any of these rows.

After the initial analysis, we opted to represent the data in multiple plots and graphs to take a better look at its content and retrieve some meaningful insights. First, we examined the target variable "unlisted," where we made a simple bar plot of its labels to understand their proportion and potentially identify imbalance problems. We saw that the label "listed" was more than double the label "unlisted," so we took note of this in case we later needed to apply some sampling techniques to improve model performance.

We then did a few more plots, where we first plotted the word count distribution for the variables "description," "host_about," and "comments." We noticed that a significant percentage of the descriptions had anywhere between 150 and 180 words, with the remaining descriptions well distributed between the ranges of 0 to 150. For "host_about," we didn’t notice the same pattern, as most texts from this variable had less than 150 words. Likewise, the "comments" variable showed a similar trend (most comments had less than 150 words).

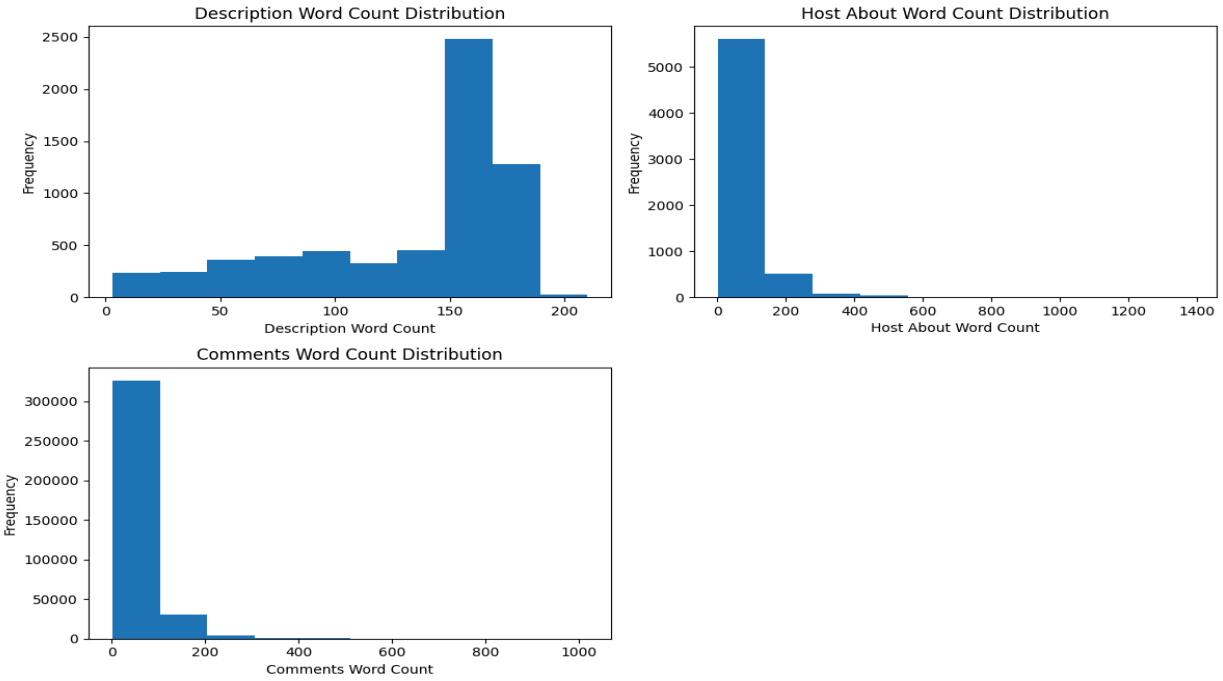

Fig. 1 - Word count distribution

We also plotted the frequency of words in each text variable, where we saw that articles, conjugations, and prepositions were by far the most frequent words across all the variables, along with HTML symbols such as "
." To further consolidate this observation, we also created word clouds for the different text variables, which showed the importance of these words and unrecognisable symbols. This allowed us to know that we had to address this issue in the preprocessing stages, as we will soon see.

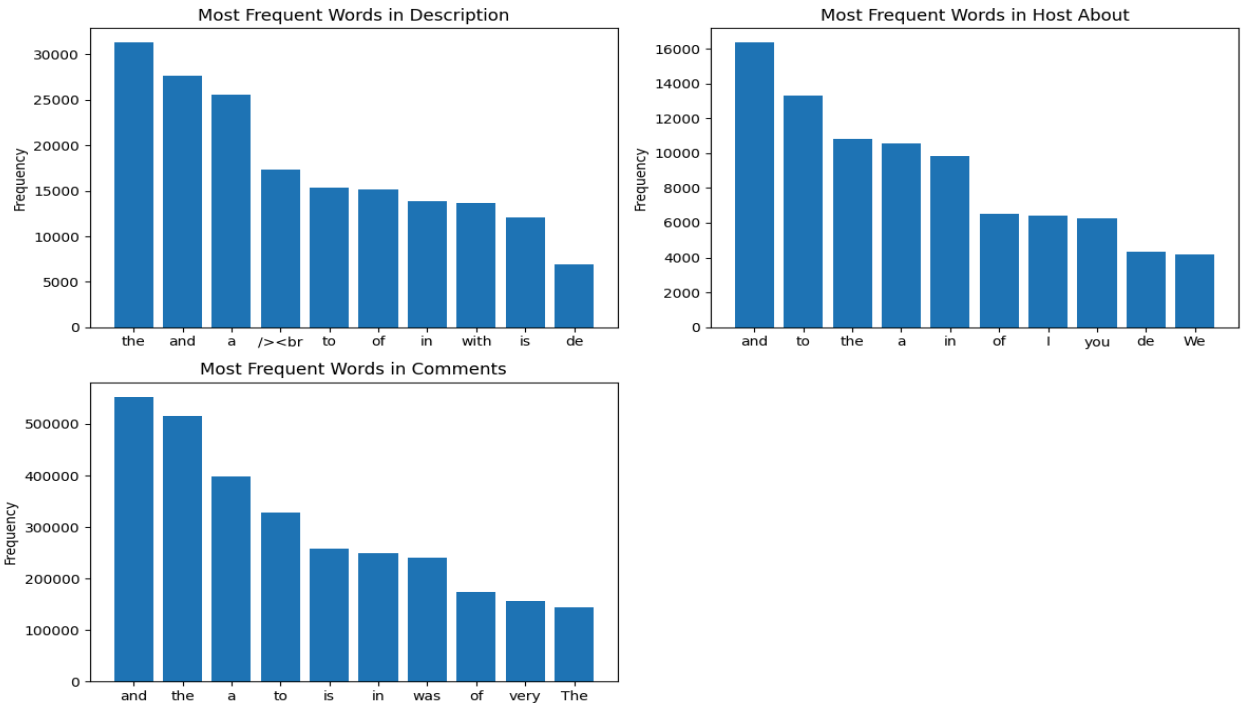

Fig. 2 - Most frequent words

Data Preprocessing

In our exploratory analysis, we detected that the text variables were written in multiple languages, some of which we couldn't even identify. Since the processing steps are typically done in one language (usually English), we had the option of either translating all the languages into English or taking a multi-language approach where we would apply different text preprocessing steps depending on the specific language. To do this, we first used a language detection library "langdetect" to create new variables that identified the language of a certain text label, for each of the text variables **(Extra Work)**. We detected that in the variables "description" and "host_about," English and Portuguese were the most prominent languages, with the remaining ones being much less frequent. The same couldn't be said for the "comments" variable, where English was still the most frequent, followed by French, Portuguese, Spanish, and German (in this order), with the remaining languages being less frequent.

We then noticed that for certain texts, the langdetect library couldn't identify the language, attributing the label "unknown." With further analysis, we concluded that this was because certain texts were merely symbols and/or representations of emojis like ":)", mainly in the "comments" variable. Since these rows didn't provide any useful information and would likely be removed during the preprocessing HTML and punctuation removal, we chose to delete them straight away to facilitate pipeline creation.

Next, we created two main pipelines because our goal was to have an easy way to call and transform our data using different preprocessing steps without having to run every line of code again. In the first pipeline, we applied all the different operators of the NLTK library, giving us more control over which techniques we were applying and to which specific variables/labels.

To do this, we decided to further split the first pipeline into two functions. The first function was used to preprocess the variables "description" and "host_about," and for this, we removed all emojis using the "emoji" library **(Extra Work)**, applied lowercase to all texts, removed any strange symbols, and tokenized everything. We then applied stopwords, lemmatization, and stemming, according to the specific language of each text, focusing on the English and Portuguese languages.

The second function was created to preprocess the "comments" variable due to the fact that this variable had a greater variety of frequent languages. The process was very similar to the previous function, only taking into account the extra languages: French, German, and Spanish. This way, we ensured that no matter the variable, we were applying the optimal preprocessing steps, and in case our results were not satisfactory, we could always just slightly change the code where we called the functions.

The second pipeline was a completely different approach where we chose to use the SpaCy library, as it's a more robust and simpler way to handle multiple scenarios **(Extra Work)**. Also, since this library uses pretrained models, trained on large amounts of text data, it typically provides better results even though it's less flexible for optimization.

For this task, we created a single function that would work on all three text variables, trained on the more frequent languages. This way, the SpaCy pipeline was able to apply different transformations to each token, like lemmatization, lowercasing, normalisation, and filtering, without having the need to either translate all languages to English or specify each language for each transformation.

To summarise, our final result were two main preprocessed datasets: one where a more custom-made preprocessing was done using the NLTK library, focusing on the more frequent languages and taking into

account the disparity of the number of main languages in the three different text variables; the other dataset was a simpler version, yet more robust, taking advantage of the different languages pretrained models provided with the SpaCy library.

Sentiment Analysis

The objective of this analysis was to perform sentiment analysis on the "comments" variable in our reviews dataset and create new variables based on the polarity of these comments.

Our first step in this stage was to use the original comments instead of the preprocessed ones. This decision was based on our utilization of the twitter-xlm-roberta-base-sentiment, a transformer model that can understand context and semantic meaning, handling raw text efficiently. This model was selected because it is adept at interpreting the nuances of raw text, which is crucial for accurate sentiment analysis. Additionally, the model was trained on Twitter data, making it particularly well-suited for analyzing informal comments similar to those found in our dataset.

We started by loading the tokenizer and model from the `transformers` library and created a sentiment analysis pipeline. This pipeline was designed to handle truncation and padding with a maximum length of 512 tokens, ensuring efficient processing of text data. To optimize performance and manage GPU memory, we processed the comments in batches of 64. The sentiment analysis results for each batch were aggregated into a comprehensive result set.

From this result set, we extracted the sentiment labels (negative, neutral, positive) and appended them as a new column in the original reviews dataset. After this, we grouped the reviews dataset by the index column and counted occurrences of each sentiment by property. These sentiment counts were merged back into the original dataset to include these sentiment-based variables. Any missing values were filled with zeros to maintain data integrity.

The resulting variables were the "reviews" which represented the number of reviews by property, and the "negative", "neutral" and "positive" variables, which represented the polarity of the comments made for each property.

We also created another binary variable, "reviews_exist", that tracked whether a property had any comments or not. This way we could later check if these comments, or lack of, had any impact on whether a property was listed or unlisted, as we will soon see.

By incorporating these new variables into our dataset, we have enriched it with valuable sentiment-based metrics. These metrics will be used to improve our machine learning models and help us achieve better results. By using advanced transformer models we managed to gain meaningful insights from raw text data, ultimately enhancing the analytical capabilities of our dataset.

More Data Exploration

Having done all our preprocessing, we once again did exploratory data analysis by plotting the data using a variety of different graphs with the intent of verifying if previous concerns had been solved and checking changes made to the different text variables, after all the transformations applied.

To do this, we created histograms to plot the word counts, for all the datasets. This way we could compare the original data to the other two preprocessed datasets that resulted from the NLTK and SpaCy, for the

"description" and "host_about" variables. Doing so, allowed us to notice that on the "description" variable we previously had a very prominent right-skewness and with the transformations it was corrected, now having the majority of texts with a word count between 60 to 100 words. On the "host_about" variable the left-skewness was preserved and in fact it appeared to be even more pronounced, with the majority of the text having anywhere between 0 to 100 words each.

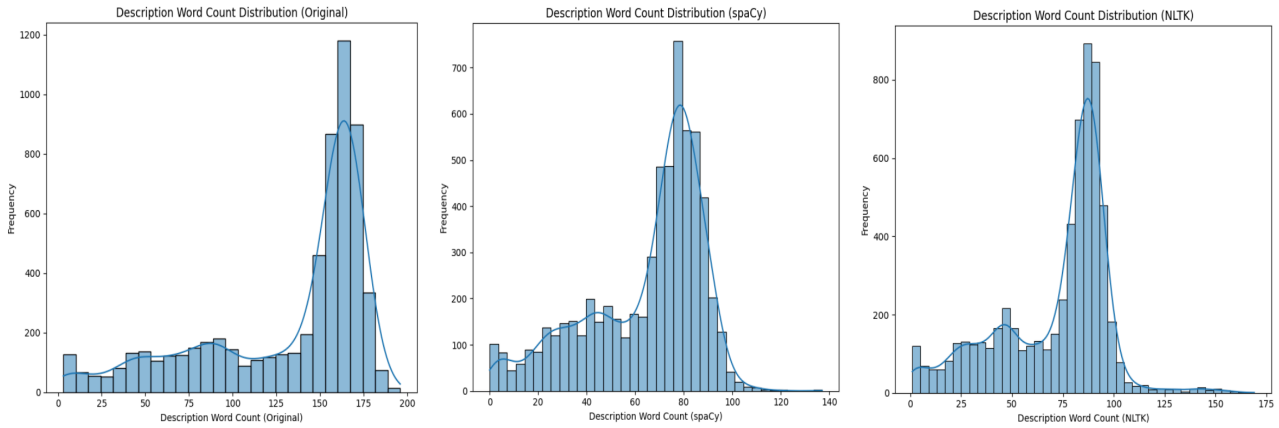

Fig. 3 - Word count distribution comparison

Then we created word clouds, to confirm the most frequent words that appear in each variable. While we previously had symbols and html text has some of the most recurrent words, with the transformations that was no longer the case. Also, we could now see that the words were lemmatized or stemmed correctly, mainly in the NLTK dataset.

Fig. 4 - Word clouds

We also plotted a simple pie chart where we could see the proportion of listed and unlisted properties, where reviews were written or not. The goal was to identify some type of pattern where properties that had less reviews would be more likely to be unlisted than otherwise, and that was precisely what we found. When reviews were made, there were 93.7% of listed properties, unlike the 28.1% of listed properties when there were no reviews. This was further confirmed when plotting a correlation matrix, where we noticed a -0.69 correlation between the variable "reviews_exist" and the target variable "unlisted".

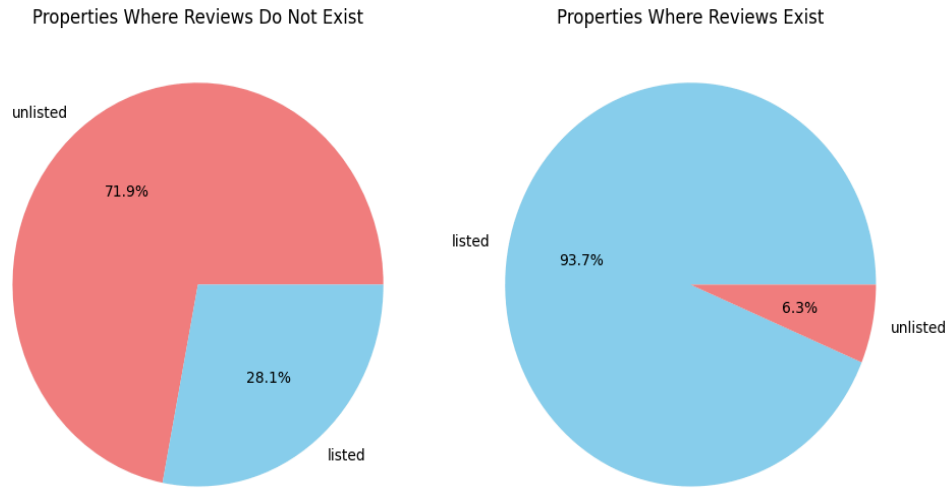

Fig. 5 - Pie Chart of reviews_exist in relation to the target variable

Finally, we plotted the average sentiment of reviews per property and noticed that on average the majority of reviews are positive, followed by negative and lastly neutral reviews. Following this approach, we compared the neutral and negative sentiments of the reviews with the listing statues, where we, curiously enough, noticed that when the sentiment was negative, there were much less unlisted (1) properties, contrary to what would be expected.

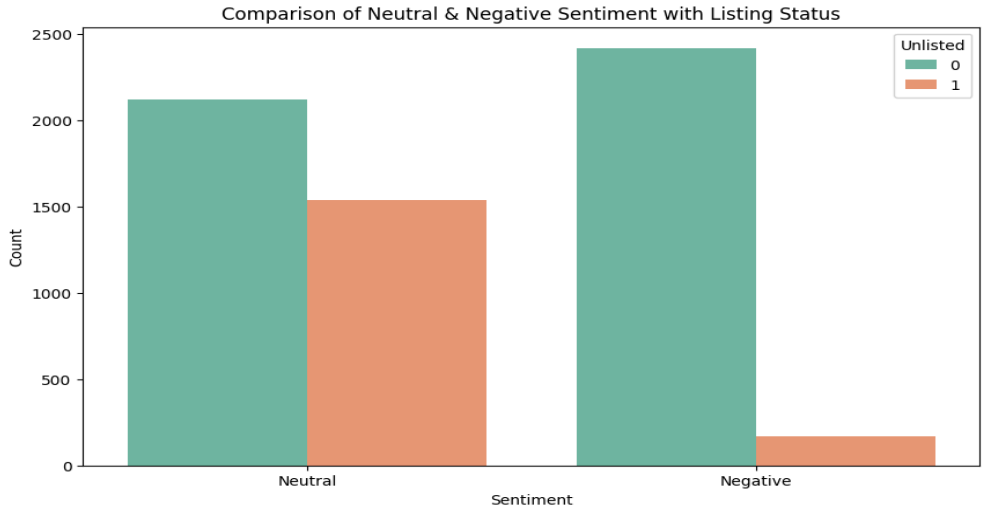

Fig. 6 - Comparison of neutral and negative sentiment in relation to the target variable

Feature Engineering

After completing the necessary preprocessing steps, we proceeded to feature engineering. For this, we employed several techniques to transform the text data into numerical representations suitable for machine learning models. The methods we used include Bag of Words (BOW), TF-IDF, Word2Vec, and GloVe. Each of these methods were applied to both the NLTK and SpaCy preprocessed datasets to ensure a comprehensive approach.

Transformer-Based Embeddings (Extra Work)

In this section, we implemented feature engineering techniques using transformer-based embeddings to enhance the performance of our NLP classification model. Specifically, we utilized the XLM-RoBERTa model and the Voyage AI model to generate embeddings for the raw text data from the "description" and "host_about" columns of the original dataset.

XLM-RoBERTa

XLM-RoBERTa is a transformer model designed for cross-lingual understanding, making it suitable for handling text in multiple languages without the need for pre-cleaning. This capability is leveraged to embed the raw text directly, capturing semantic and contextual information effectively.

We utilized the XLM-RoBERTa model from the Hugging Face library to extract embeddings. The process involved loading the pre-trained model and tokenizer, then embedding the text data from the "description" and "host_about" columns. The embeddings were generated by tokenizing the text and passing it through the model, with average pooling applied to obtain a single fixed-size vector for each input text. These embeddings were then added to our dataset.

Voyage AI

Voyage AI, recognized for holding the top position on the Hugging Face leaderboard at the time, offers state-of-the-art embeddings for a wide range of NLP tasks. We utilized the Voyage AI model to generate embeddings for the text data without pre-cleaning, leveraging its capability to process raw text effectively.

We began by obtaining our API key from Voyage AI, which is required to access their services. After setting up the API key, we initialized the Voyage AI client. We then defined a function to embed text using the Voyage AI model. This function sent the text to the Voyage AI API and retrieved the embeddings. The embeddings for both the "description" and "host_about" columns were generated using this function and added to our dataset.

By embedding the raw text using XLM-RoBERTa and Voyage AI, we aimed to retain the rich semantic information and context present in the descriptions and host information. This approach is beneficial because it handles multilingual text efficiently, reduces the need for extensive text preprocessing, and captures deeper contextual relationships compared to traditional embeddings.

Classification Models

After all the data was prepared, the objective of this analysis was to build and evaluate multiple classification models to predict whether a property is unlisted. This involved utilizing various text feature engineering methods and incorporating numerical features derived from sentiment analysis. The models implemented in

this study include Logistic Regression (LR), Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), BERT, and XLM-Roberta.

All our models were trained using an 80% training set and evaluated using a 20% validation set. Due to time constraints, we opted not to perform K-Fold validation.

Logistic Regression

For the Logistic Regression, we created models using the variables from the following feature engineering methods: Bag of Words (BoW), TF-IDF (Unigram, Bigram, Trigram), Word2Vec, and GloVe. As seen earlier, these feature engineering methods were applied separately to both NLTK and Spacy preprocessed datasets, so we had to create models for each one of these datasets. Additionally, we created models for the two datasets derived from the transformer-based embeddings XLM-Roberta and Voyage. All these feature engineering methods were used in the models alongside the numerical features derived from the sentiment analysis.

Multi-Layer Perceptron

For the Multi-Layer Perceptron (MLP), we employed a neural network-based approach to enhance our classification models. Similar to the Logistic Regression models, we utilized the same feature engineering methods: Bag of Words (BoW), TF-IDF (Unigram, Bigram, Trigram), Word2Vec, and GloVe. These methods were applied to both the NLTK and SpaCy preprocessed datasets, resulting in separate models for each dataset. We also created models for the transformer-based embeddings XLM-RoBERTa and Voyage AI. Each model incorporated the numerical features derived from sentiment analysis.

The MLP models were designed with a specific architecture tailored to our classification task. The architecture began with an input layer. This input layer was followed by a dense layer with 64 neurons and a ReLU activation function, which introduced non-linearity to help the model learn complex patterns in the data. The output from this dense layer was then fed into another dense layer, also with 64 neurons and ReLU activation, to further capture intricate relationships in the data.

Subsequently, a third dense layer with 32 neurons and ReLU activation was added to refine the learned representations. Finally, the model concluded with an output layer consisting of a single neuron with a sigmoid activation function, which produced a probability score indicating whether a property was listed or unlisted. The model's predictions on the validation set were converted to binary outcomes using a threshold of 0.5

The complete model was compiled using the Adam optimizer, known for its adaptive learning rate capabilities and efficiency in handling sparse gradients. The loss function used was binary cross-entropy, appropriate for binary classification tasks. The model's performance was tracked using accuracy as the primary metric.

For the training process we used 10 epochs, and a batch size of 32 to balance training speed and model performance. Throughout the training, the model's performance on the validation set was monitored to ensure it was generalizing well to unseen data.

Long Short-Term Memory

For the Long Short-Term Memory (LSTM) models we used advanced feature engineering methods including Word2Vec, GloVe, XLM-RoBERTa, and Voyage AI embeddings. We maintained the same approach as our earlier models with the NLTK and SpaCy datasets as well as the numerical features.

Unlike previous models where we used Bag of Words (BoW) and TF-IDF, these methods were not used for the LSTM models because they do not capture the sequential nature of text data effectively. LSTMs are designed to process sequences and understand the context over time, making embeddings like Word2Vec, GloVe, XLM-RoBERTa, and Voyage AI more suitable as they preserve the order and semantic relationships of words in the text.

The architecture began with two input layers for the embeddings: one for the "description" embeddings and one for the "host_about" embeddings. Each input layer was followed by a masking layer to handle any padding in the sequences, ensuring the LSTM layers only processed the actual data and not the padded values.

The masked sequences were then passed through bidirectional LSTM layers, which had 4 units each. Using bidirectional LSTMs allowed the model to capture dependencies in the text data from both forward and backward directions, enhancing the context captured by the embeddings.

The outputs from the bidirectional LSTM layers were then concatenated with the input numerical features, which included attributes like "reviews_exist," "negative," "neutral," "positive," and "reviews". This concatenation combined the sequential text data with the numerical features into a single representation.

The combined features were then passed through a dense layer with 64 neurons and a ReLU activation function to introduce non-linearity and help the model learn complex patterns in the data. Finally, the model concluded with an output layer consisting of a single neuron with a sigmoid activation function, which produced a probability score indicating whether a property was listed or unlisted. The model's predictions on the validation set were converted to binary outcomes using a threshold of 0.5, as we had seen earlier with the MLP.

The complete model compilation, performance tracking and training process were the same as the MLP.

The use of LSTM models allowed us to explore the benefits of sequence-based neural network architectures in our classification task. By applying advanced feature engineering techniques such as Word2Vec, GloVe, XLM-RoBERTa, and Voyage AI embeddings, we ensured that the models captured the rich, sequential nature of the text data. This approach, combined with the incorporation of numerical features from sentiment analysis, enabled us to capture complex patterns and relationships in the data, ultimately enhancing the predictive power of our models.

Transformers and other advanced models (Extra Work)

For transformer-based models, we used the original text variables because these models are pre-trained and can leverage the rich context provided by raw text. Our approach involved using the description and host_about variables, and creating an additional text variable that concatenates all reviews associated with each property.

To ensure that these models capture the full context of the reviews, we concatenated all comments for each property into a single text variable. This comprehensive text input was formed by merging the original dataset with the concatenated reviews dataset, ensuring that the reviews were included in the main dataframe. Any missing review comments were filled with an empty string to prevent issues during text processing. Finally, a new text variable was created by concatenating the description, host_about, and comments fields.

BERT

We chose BERT (Bidirectional Encoder Representations from Transformers) for its powerful language representation capabilities and its ability to understand the context and semantics of the text more effectively than traditional models. BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers, making it highly suitable for understanding nuanced text data and capturing the relationships between words in a sentence.

To implement the BERT model, we used the BertTokenizer for tokenizing the text data. The tokenizer from the pre-trained BERT model (bert-base-uncased) was loaded, and the text data was tokenized with appropriate padding and truncation to fit the model's input requirements.

We defined a custom dataset class to handle the tokenized data and corresponding labels. This class, CustomDataset, converted the tokenized encodings and labels into PyTorch tensors, facilitating the use of the data in model training. The training and validation datasets were created using the CustomDataset class, which allowed the models to be trained and evaluated effectively.

After preparing the data, we loaded the pre-trained BERT model for sequence classification, specifying that the model has two labels. The model was then moved to the appropriate device (GPU if available).

We defined the data collator to handle padding and set up the training arguments, specifying parameters such as output directory, evaluation strategy, learning rate, batch sizes, number of epochs, and weight decay.

A Trainer instance was created to handle the training and evaluation processes. The model was trained using the training dataset, and the evaluation was conducted using the validation dataset.

XLM-Roberta

For our second transformer model we decided to use XLM-Roberta for its exceptional ability to understand context and semantic meaning across multiple languages, which makes it particularly powerful for our task. XLM-Roberta, a variant of the RoBERTa model, is pre-trained on a large multilingual corpus, allowing it to capture nuanced meanings and relationships in text more effectively than many other models. This capability is especially important in our dataset, which may contain varied linguistic patterns and expressions within the reviews and descriptions. The XLM-Roberta model was fine-tuned with our dataset, using input layers for text embeddings and numerical features.

In terms of the implementation, the approach was the same as the one used in the BERT model.

Evaluation and Results

The evaluation of our classification models involved using a set of standard metrics to measure the performance of each model. The key metrics used were precision, recall, f1-score, and accuracy.

The models were evaluated using the validation set, and the metrics were calculated to provide a comprehensive assessment of their performance. Below are tables comparing the results of each model, highlighting the best performers. For this comparison we used the weighted average for the precision, recall and f1-score metrics.

Model	Feature Engineering	Preprocessing	Accuracy	Precision	Recall	F1-Score
Logistic Regression	BoW	NLTK	0.88	0.88	0.88	0.88
Logistic Regression	BoW	Spacy	0.87	0.87	0.87	0.87
Logistic Regression	TFIDF (unigram)	NLTK	0.88	0.89	0.88	0.89
Logistic Regression	TFIDF (bigram)	NLTK	0.89	0.89	0.89	0.89
Logistic Regression	TFIDF (trigram)	NLTK	0.89	0.89	0.89	0.89
Logistic Regression	TFIDF (unigram)	Spacy	0.88	0.89	0.88	0.89
Logistic Regression	TFIDF (bigram)	Spacy	0.88	0.89	0.88	0.89
Logistic Regression	TFIDF (trigram)	Spacy	0.89	0.90	0.89	0.89
Logistic Regression	Word2Vec	NLTK	0.86	0.85	0.86	0.85
Logistic Regression	Word2Vec	Spacy	0.84	0.84	0.84	0.84
Logistic Regression	GloVe	NLTK	0.81	0.81	0.81	0.81
Logistic Regression	GloVe	Spacy	0.83	0.83	0.83	0.83
Logistic Regression	XLM-Roberta	-	0.88	0.89	0.88	0.89
Logistic Regression	Voyage	-	0.89	0.89	0.89	0.89

Table 1 - Logistic regression models

Model	Feature Engineering	Preprocessing	Accuracy	Precision	Recall	F1-Score
MLP	BoW	NLTK	0.88	0.88	0.88	0.88
MLP	BoW	Spacy	0.87	0.86	0.87	0.86
MLP	TFIDF (unigram)	NLTK	0.88	0.88	0.88	0.88
MLP	TFIDF (bigram)	NLTK	0.87	0.87	0.87	0.87
MLP	TFIDF (trigram)	NLTK	0.88	0.88	0.88	0.88
MLP	TFIDF (unigram)	Spacy	0.87	0.87	0.87	0.87
MLP	TFIDF (bigram)	Spacy	0.85	0.85	0.85	0.85
MLP	TFIDF (trigram)	Spacy	0.87	0.87	0.87	0.87
MLP	Word2Vec	NLTK	0.87	0.87	0.87	0.87
MLP	Word2Vec	Spacy	0.88	0.89	0.88	0.88
MLP	GloVe	NLTK	0.88	0.88	0.88	0.88
MLP	GloVe	Spacy	0.87	0.87	0.87	0.87
MLP	XLM-Roberta	-	0.87	0.87	0.87	0.87
MLP	Voyage	-	0.87	0.87	0.87	0.87

Table 2 - Multi-Layer Perceptron models

Model	Feature Engineering	Preprocessing	Accuracy	Precision	Recall	F1-Score
LSTM	Word2Vec	NLTK	0.89	0.89	0.89	0.89
LSTM	Word2Vec	Spacy	0.87	0.87	0.87	0.87
LSTM	GloVe	NLTK	0.88	0.88	0.88	0.88
LSTM	GloVe	Spacy	0.88	0.88	0.88	0.88
LSTM	XLM-Roberta	-	0.88	0.89	0.88	0.89
LSTM	Voyage	-	0.88	0.88	0.88	0.88

Table 3 - LSTM models

Model	Feature Engineering	Preprocessing	Accuracy	Precision	Recall	F1-Score
Transformers	BERT	-	0.87	0.87	0.87	0.87
Transformers	XLM-Roberta	-	0.84	0.84	0.84	0.84

Table 4 - Transformers models

Best Performing Models

Model	Feature Engineering	Preprocessing	Accuracy	Precision	Recall	F1-Score
Logistic Regression	**TFIDF (trigram)**	**Spacy**	**0.89**	**0.90**	**0.89**	**0.89**
MLP	**Word2Vec**	**Spacy**	**0.88**	**0.89**	**0.88**	**0.88**
LSTM	**Word2Vec**	**NLTK**	**0.89**	**0.89**	**0.89**	**0.89**
Transformers	**BERT**	**-**	**0.87**	**0.87**	**0.87**	**0.87**

Table 5 - Best performing models

From all the models, the best performer was the Logistic Regression. This specific model used TF-IDF trigram as the feature engineering method and SpaCy as the preprocessing method. After identifying the best-performing model based on these evaluation metrics, we proceeded to use this model to make predictions on the test set.

Conclusion

This project successfully leveraged Natural Language Processing (NLP) techniques to predict whether an Airbnb property would be unlisted in the next quarter. By integrating data from property descriptions, host descriptions, and guest reviews, we developed and evaluated multiple classification models, including Logistic Regression (LR), Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), BERT, and XLM-RoBerta. We employed various text feature engineering methods and numerical features derived from sentiment analysis to enrich our dataset.

Our extensive preprocessing efforts, including handling multilingual text, tokenization, and sentiment analysis, ensured that the models received high-quality inputs. The best-performing model was a Logistic Regression using TF-IDF (trigram) and SpaCy preprocessing, achieving an accuracy of 0.89, precision of 0.90, recall of 0.89, and F1-score of 0.89. This model's strong performance demonstrates the effectiveness of our comprehensive approach in capturing nuanced textual and numerical features for accurate prediction.

Future Work

While this project has demonstrated promising results, there are several avenues for future research and improvements. Translating the documents used in Word2Vec and GloVe could improve results, as it would ensure that all text data is processed in a consistent language, thereby reducing noise and enhancing the quality of word embeddings. Implementing K-Fold cross-validation would provide a more robust evaluation of model performance by ensuring that all data points are used for both training and validation. Conducting extensive hyperparameter tuning for each model, especially for neural network-based models like MLP and LSTM, could further enhance model performance. Additionally, exploring more advanced NLP techniques, such as sentence transformers or domain-specific pre-trained models, might capture deeper semantic relationships in the text data.

By addressing these areas, future research can build on the foundation laid by this project, further enhancing the accuracy and applicability of models predicting Airbnb property unlistings. This continued exploration will contribute valuable insights to both researchers and industry practitioners, ultimately benefiting the broader Airbnb community.

References

[1] Hugging Face. (n.d.). MTEB Leaderboard. Retrieved from https://huggingface.co/spaces/mteb/leaderboard

[2] Voyage AI. (n.d.). API Key and Installation. Retrieved from https://docs.voyageai.com/docs/api-key-and-installation

[3] Voyage AI. (n.d.). Homepage. Retrieved from https://www.voyageai.com/