Group Project Text Mining 2023/2024

Predicting Listing Status of AirBNB Properties

Masters in Data Science and Advanced Analytics

Text Mining

Afonso Gorjão (20230575), Diogo Almeida (20230737),
Frederico Portela (20181072), Pedro Carvalho (20230554)

June 9, 2024

Contents

I. Introduction

Over the past few decades, global tourism has surged, leading to a significant increase in the demand and supply of accommodations. Traditional hotels have seen their monopoly eroded by the rise of internet-based booking platforms like Airbnb. With this growing diversity in lodging options, hosts are increasingly focused on satisfying customers to garner positive online reviews. For a platform like Airbnb, analyzing these reviews is crucial as such analysis provides deeper insights into consumer experiences and the factors influencing their satisfaction or dissatisfaction. In this project, we will leverage reviews and property and host descriptions to predict the likelihood of a property being unlisted in the next quarter.[1]

II. Data Exploration

This section provides an overview of the dataset and the initial findings from our exploratory data analysis (EDA). The aim is to understand the data's distribution, patterns, and characteristics. There are four datasets: *train* (that we will refer to as *train_info*) containing 6,248 records and information about property descriptions, host descriptions, and listing status; *train_reviews* containing more than 300,000 guest comments; *test* (that we will refer to as *test_info*) with 695 records structured like the train set but without the listing status; *train_reviews* with more than 40,000 guest comments.

2.1 Missing Values

Initially, we observed that some fields contained blank spaces not recognized as nulls. To address this issue, we implemented a method to replace such pseudo-NaNs with actual NaN values. This method involved using regular expressions to identify empty or whitespace-only strings and replacing specific strings like 'Nan' and 'Nul' with NaNs. Additionally, we also considered documents solely composed of punctuation, or of only one character's length or just the same character repeat multiple times as pseudo-NaNs. Initially, only the *train_reviews* dataset contained 2 empty comments, but after applying the *replace_pseudo_nans* function, we discovered more comments (901) and *host_about* (73) descriptions were missing, mostly on the train dataset (only 130 out of the 1031 missing comments are on the test dataset). Given that these values are actual missing values the data will be analysed as it is after pseudo-NaN replacement.

2.2 Checking for Duplicates

We then checked for duplicate rows in both the *info* and *reviews* datasets. The results indicated that *train_reviews* and *test_reviews* datasets had 253 and 35 duplicates, respectively. These duplicate rows were subsequently removed to maintain data quality for a clearer EDA.

2.3 Language Detection

Given the multilingual nature of the dataset, we used the *FastText* library in order to detect the language of the text documents. The analysis revealed 11 unique languages in the *description* column, 17 in the *host_about* column, and 59 in the *comments* column.

The plots showed the distribution of these languages, being the most common ones: English (en), French (fr), Portuguese (pt), Spanish (es), German (de), Italian (it), Dutch (nl), and Russian (ru). These common languages covered the majority of the text data, approximately 98.5% of the comments were in

these languages, as can be seen in Figure 1. This visualization helped us determine which languages were prevalent and should be focused on for translation and further pre-processing.

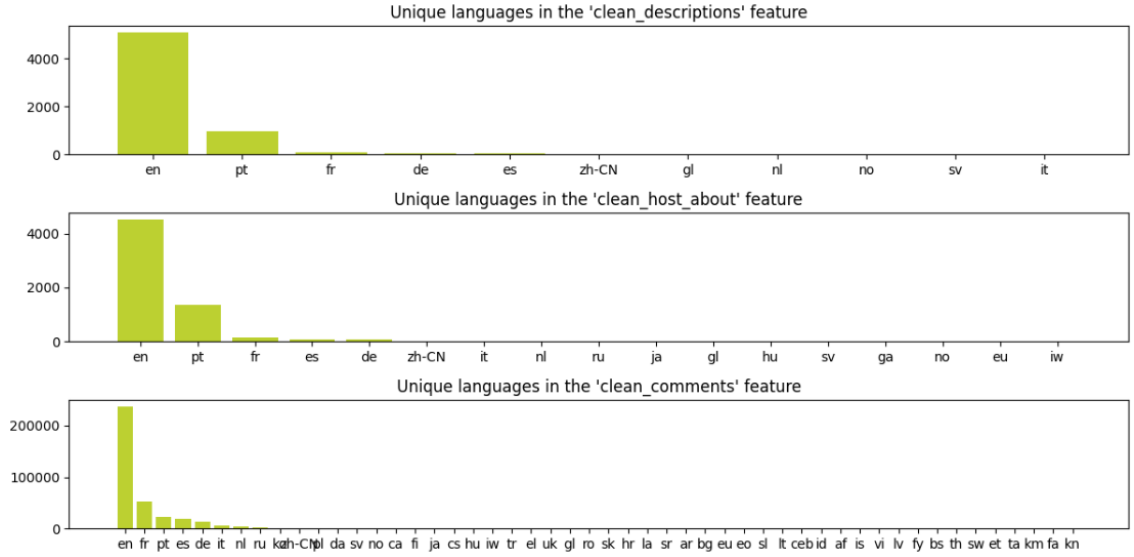

Figure 1: Languages present in training dataset

2.4 Distribution of Listing Status and Possible Relationships

We analyzed the distribution of properties listed vs. unlisted in the *train_reviews* dataset. The results revealed that 27.34% of the properties unlisted (1708 out of 6248), while 72.66% were kept listed (4540 out of 6248), indicating a slight data imbalance. Also, 67.94% (4245 out of 6248) of properties had reviews, with an average of 85.07 reviews per property. The median number of reviews per property was 48, with a mode of 1 review per property. In the test dataset, 68.35% of properties had reviews, with an average of 88.11 reviews per property. The median number of reviews was 45, the mode was 1. This analysis highlighted that a significant portion of properties had a substantial number of reviews, which could be informative for our predictive model.

We decided to check if the existence of any review had a potential relationship with the listing status. It proved to be true. We can observe that properties without any reviews are much more prone to be unlisted (Figure 2). More specifically, if a property has at least one review then its chance of being unlisted is around 6% whereas if a property has no reviews then its chance of being unlisted skyrockets to around 72%. We later used this information to create our baseline model.

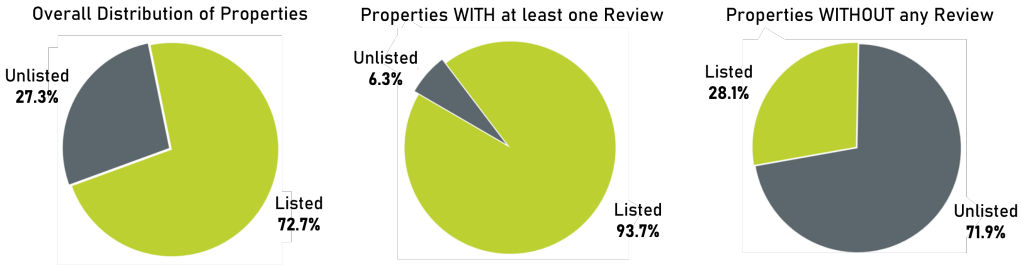

Figure 2: Difference in the distribution of properties with or without any review

We also explored the relationship between the number of reviews and the listing status. The analysis suggested that, despite a very similar distribution in both cases, there's a slight tendency for properties

with more reviews to remain listed (Figure 3). This insight could be valuable for our predictive modeling, indicating that properties with higher engagement from guests (as reflected by the number of reviews) are more likely to stay listed.

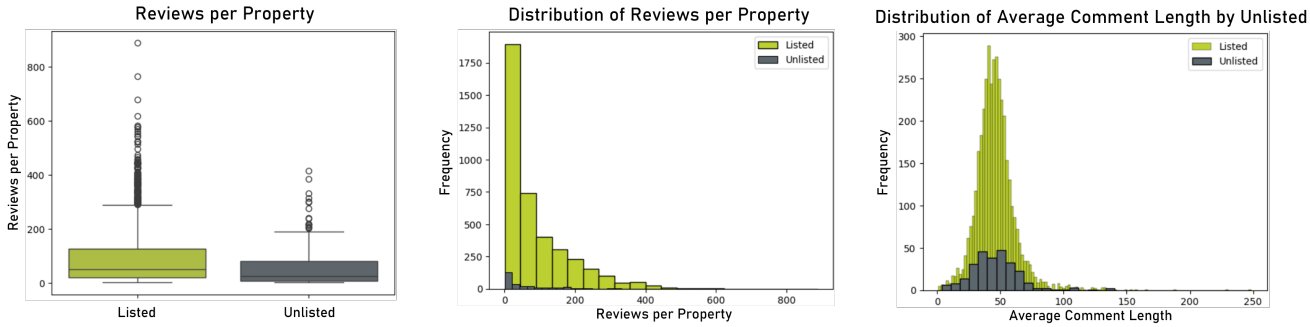

Figure 3: Left: Boxplot of reviews per property for listed and unlisted properties. Center: Histogram with reviews per property for listed and unlisted properties. Right: Histogram with average comment length

After analyzing the data and considering our knowledge of the Airbnb platform and how it works, we thought about using the super-host feature in our favor. Despite not having access to which hosts are labeled as super-host, we can use *comments*, *host-about*, and *description* and look for a "super-host". Then, we realized that around 10% of the properties had "super-host" and that they were less likely to be unlisted. The distribution can be seen in Figure 4. However, we also inferred that the likelihood of a host being considered a "super host" is highly correlated to if the property has reviews or not.

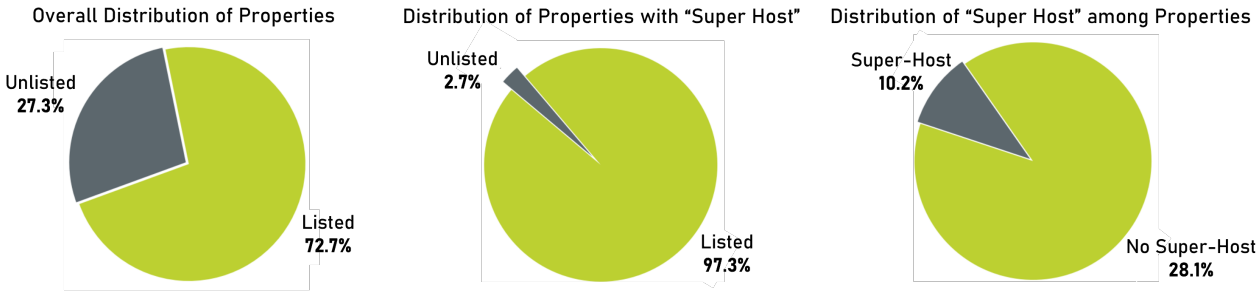

Figure 4: Left: Properties listed and unlisted distribution. Center: Distribution of properties with "super-host" in listed and unlisted. Right: Percentage of properties with and without "super-host"

Furthermore, we also discovered that the same host can be mapped to several properties and assessed whether if the number of properties being managed by the same host (an individual or an enterprise) affected the listing status, it seemed it did not.

III. Data Preprocessing

In this section, we outline the pre-processing steps applied to the data. To ensure the consistency of the pre-processing transformations, we merged the *info* and *reviews* datasets for both the training and testing data and reindexed the newly merged datasets (i.e. added what we called the *property_index* to mark properties and *comments_index* to mark row indexes). Additionally, we dropped duplicated rows and replaced pseudo-NaNs with actual NaNs. Our plan was to iteratively add new columns to the dataset, where each new column represents a transformed version of the original column. The names of these new columns indicate all the transformations that the original column has undergone to reach that point.

3.1 Text Cleaning

To clean the text, we applied several cleaning functions using regular expressions and common string methods. The cleaning process included removing default strings, e.g. the *COVID* tag in property descriptions, HTML tags, newlines, special characters, control characters, mentions, URLs, hashtags, replacing numbers by their string representation e.g. "1" to "one" and multiple white spaces. Common transformations such as lower casing the text and the removal of functional punctuation were postponed until after the additional step of Named-Entity Recognition, because after several empirical tests we arrived at the conclusion that the pre-trained NER algorithm used worked better with punctuation, mainly apostrophes, and upper cased names. For example, in the sentence "Silvia's place was awesome." it is easier to detect Silvia as PERSON entity rather than in the sentence "silvias place was awesome." or even in "silvia's place was awesome.".

Inevitably the cleaning function will trim some documents until they are blank strings. We made sure these strings were replaced by actual NaN values to ensure proper handling.

3.2 Handling Missing Values

As previously mentioned, there are a considerable amount of properties without reviews. Removing these properties would result in losing potentially important information. We addressed this issue by filling the missing values (in both the *comments* and *host_about* columns) with the word "unknown". We also ensured that this word was not adultered in subsequent pre-processing steps.

3.3 Additional Step #1: Translation of Rare Languages to English

As mentioned in 2.3, approximately 98.5% of the comments were written in any of 8 languages (see Figure 2.3). Although ideally our dataset should only have text in one unique language, translating all data to English would be too time consuming and compute expensive as well as it could introduce bias due to information being "lost in translation". Therefore we picked the most common languages to be kept as they are, but will translate the remainder into English. We chose the top 8 most common languages because, besides their high occurrence numbers, they are also all supported by the traditional language processing libraries such as *NLTK* and *spaCy*.

We used the *deep_translator* library for the translation, which essentially calls the Google Translator API. Before performing the translations, we made some quick fixes to handle language code discrepancies: for example, 'he' (Hebrew) was replaced with 'iw', and 'zh' (Chinese) with 'zh-CN' to match the library's requirements. Some extremely rare languages were not supported by the package and were dropped - they comprised <1% of the data and all properties they were mapped to had at least one other review in another, supported language, so we would not be dropping any properties by dropping those comments.

Translating rare languages also helped reduce our vocabulary size, which is beneficial for feature engineering. It's important to note that doing this step this early in the pre-processing stage, i.e. before cleaning and transforming the text data too thoroughly, prevented the loss of semantic context and the quality of the translation.

3.4 Additional Step # 2: Named Entity Recognition (NER)

Named Entity Recognition (NER) is a natural language processing technique that identifies and classifies key information (entities) in text into predefined categories such as names of people, organizations,

locations, dates, and more. It is useful for extracting structured information from unstructured text, enabling applications like automated content analysis, and information retrieval, and enhancing search engine accuracy. In our case, we used NER to identify people and locations (countries, cities, streets, between others, described by the PERSON, GPE, and LOC tags), which obviously were abundant across all the text fields in our data and to replace such entities with placeholder words like 'person' and 'location', respectively. This would help us greatly reduce the vocabulary size.

Although the algorithm seemed to work for several instances after manually reviewing the transformed data we observed the results were not perfect. Many entities were not recognized, one possible reason is the informal nature and orthography of the reviews. Nonetheless, although it may not always increase the quality of the text it did not seem to decrease it, i.e. we basically have various False Negatives - not picking up on entities - but few, if any, False Positives.

3.5 Stop Word Removal

Removing stop-words is essential to focus on the meaningful content of the text. The function *remove_stopwords* was defined to process the text, eliminating any words found in the respective language's stop-word list. This function tokenizes the text, checks each token against the stop-word list, and then reconstructs the text without the stop-words.

During this process, we discovered that some texts could become empty after removing all stop words, resulting in new pseudo-NaNs. To address this, we checked for blank strings after stop-word removal. If a string became blank, we replaced it with the token 'unknown' to maintain consistency and avoid issues in subsequent steps.

3.6 Stemming and Lemmatization

Stemming and Lemmatization help standardize the text data and reduce the language model's complexity. We utilized the SnowballStemmer from the *NLTK* library for stemming and the *spaCy* library for lemmatization. We structured our code so that retained both the stemmed and unstemmed, as well as, lemmatized and unlemmatized versions of the text, providing flexibility for further analysis.

3.7 Train-Test Split

Finally, at the end of pre-processing, we performed the train-test split. In the split, we wanted to make sure that the data was stratified in terms of the listing status but also in terms of houses with and without reviews - our split proportions were 80-20 and we did not create a validation set because throughout model development we employ k-fold cross validation techniques.

IV. Data Exploration After Pre-Processing

In this section, we analyzed the outcomes of the pre-processing steps.

4.1 Word Cloud and N-Grams

To further explore the data, we generated word clouds for some text columns to visually represent the most frequent words. This visualization helped us quickly identify the dominant terms and provided intuitive insights into the content of the descriptions, reviews, and host information. It also assured us of

the quality of cleaning processes (e.g. common HTML tags are not shown in the word clouds of heavily pre-processed columns).

In addition to word clouds, we conducted an N-gram analysis to understand the frequency of 2-grams and 3-grams within the text. This analysis revealed common sequences of words that appear together frequently. By identifying these patterns, we gained a better understanding of the typical language and expressions used by hosts and guests. We also examined the correlation between the most common N-grams in the pre-processed comments column and the target variable 'unlisted'. This analysis showed that certain N-grams had a noticeable correlation with whether a property remained listed or not. For instance, two specific N-grams, "great location" and "highly recommend", each showed a correlation of approximately 0.5 with the target variable.

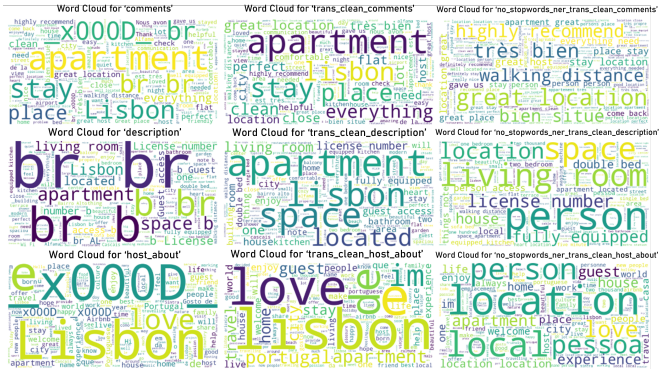

Figure 5: Word clouds for different stages of pre-processing of the original text data

4.2 Distribution Of Languages

This analysis aimed to determine if the language distribution of comments affects whether a property remains listed or becomes unlisted, focusing on whether certain properties cater more to tourists from specific countries. We plotted the distributions of comments in different languages for listed and unlisted properties. (Fig. 6) The languages we analyzed were the most common in the dataset. We found that all these languages showed similar distributions for both listed and unlisted properties. This indicates that the number of comments in any particular language does not significantly impact whether a property remains listed or becomes unlisted. Overall, the analysis revealed that properties do not cater disproportionately to tourists from specific countries in a way that influences their listing status.

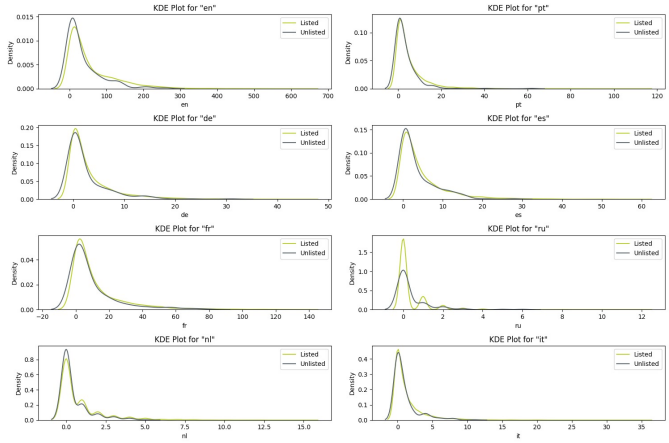

Figure 6: Word clouds for different stages of pre-processing of the original text data

4.3 Length of Description and Host About for Properties Without Reviews

Similarly to the topic before, here we tried to find a relationship between whether the length of the description and host information had any correlation with the property's listing status for properties that did not have any reviews. (Fig. 7) Like before, there was no clear separation between the distributions of the listed and unlisted properties based on the length of the descriptions or host information. Suggesting that the length of these text fields does not significantly discriminate between properties that remain listed and those that do not.

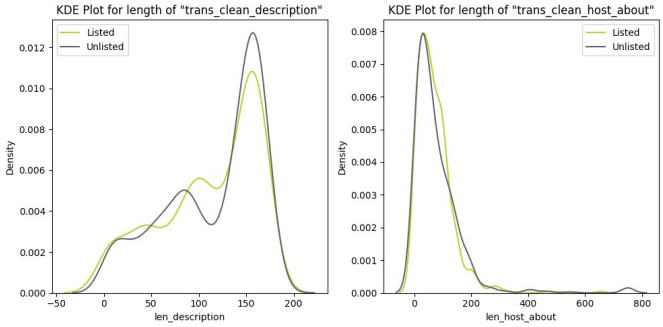

Figure 7: Word clouds for different stages of pre-processing of the original text data

4.4 Sentiment Analysis

In this step, we aimed to determine the relationship between the sentiment of reviews and whether a property remains listed or becomes unlisted. We sourced the pre-trained "nlptownbert-base-multilingual-uncased-sentiment" model from the HuggingFace library tuned for sentiment analysis. This model allows for multilingual data.

To determine if there was a significant relationship between sentiment and listing status, we conducted a chi-square test for independence. The null hypothesis (H0) states that there is no significant relationship between the sentiment of reviews and the listing status of properties. In contrast, the alternative hypothesis (H1) states that there is a significant relationship between these two variables.

The chi-square test results showed that for the columns *'ner_trans_clean_description'* and *'ner_trans_clean_host_about'*, we failed to reject the null hypothesis, for a significance level of 5%, indicating no relationship between the sentiment of these texts and the listing status of the properties. However, for the *'ner_trans_clean_comments'* column, we reject the null hypothesis, suggesting a significant relationship between the sentiment of comments and the listing status of properties. This indicates that the sentiment expressed across all comments for the same property may influence whether a property remains listed or becomes unlisted.

V. Feature Engineering

To enhance our model's understanding of the textual data, we performed several feature engineering steps to create extract embeddings.

5.1 TF-IDF

Firstly, we resorted to the *term frequency-inverse document frequency*. It weighs the importance of words based on their frequency in a document relative to the entire corpus. The main drawback of this

technique, when applied to our dataset, regards the existence of various languages in the corpus. The importance of a token would inevitably be affected by the proportion of tokens of its origin language in the corpus, and consequently, important tokens could be attributed lower importance scores if they were tied to underrepresented languages. To bypass this issue we only selected documents pertaining to English because the majority of houses (upwards of 95%) have at least one comment in English, and we assumed these comments were representative of the rest. Due to this technique's drawbacks compared to later techniques, we only created one dataset comprised of all properties, with or without reviews, using the *lemma_no_stopwords_ner_trans_clean_comments* feature, filtered by English comments. The TF-IDF vectorizer was constrained to output vectors of dimensions 1000, to avoid common issues that prey on this technique, such as the curse of dimensionality. Regarding the code we would like to highlight that TF-IDF is the only FE technique used that needs to first be fitted to the train data and then transform the train and test datasets - latter techniques extract embeddings from pre-trained models - therefore we were cautious in avoiding data leakage (for example when doing K-fold cross-validation) by passing the vectorizer object to subsequent training functions and fit and transform the data accordingly during model training. To assess the quality of this approach we fitted and transformed our train data to a vectorizer object and computed the average importance scores for each feature (1000 features, as previously mentioned). The results can be seen in the figure below (Fig. 8). We would like to highlight the expected high importance of the 'unknown' word - the token we used as a placeholder for missing values - and a confirmation of the value of the NER technique (although it has its flaws it did work for recognizing several entities).

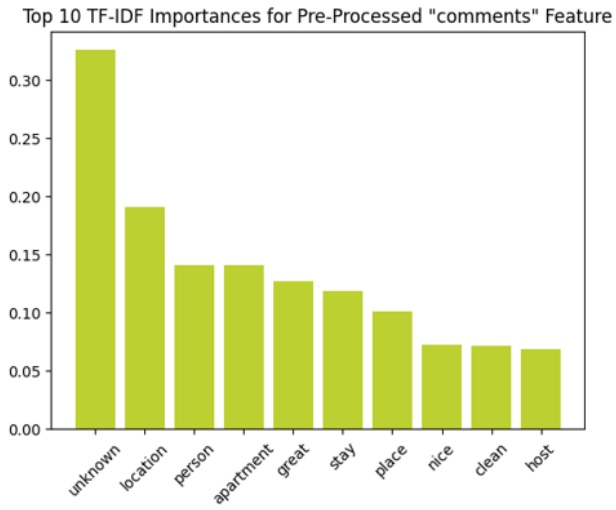

Figure 8: Top 10 TF-IDF Importances for Pre-Processed "comments" feature

5.2 Glove

Global Vectors for Word Representation (GloVe, for short) is an unsupervised learning algorithm for generating word embeddings. The key idea is to leverage the co-occurrence probabilities of words within a given context to produce meaningful word representations. We created three new datasets using this technique. For them all we resorted to the *torch-vocab* library to extract pre-trained GloVe models and used the 6B model. To extract GloVe embeddings, we implemented a class designed to load pre-trained vectors and generate sentence embeddings by averaging the vectors of the words in each sentence. If the word is not found in the GloVe vocabulary, it returns a zero vector of the same length as the model's vector dimension (user-defined parameter). We resorted to averaging the word vectors to form a 'sentence' embedding since the models we used (besides Transformer-based models)

are not able to handle sequentiality. All three datasets regard all properties (with and without reviews), the main difference between them is that the first one produces sentence embeddings of length 50 for the *lemma_no_stopwords_ner_trans_clean_comments* feature, which might compress too much information, and to deal with this we recreated this dataset but using word representations of length 200. The difference between the first and second datasets is not the dimensionality of the embeddings but of the feature used, in the latter, we sourced the stemmed version of the feature to be able to accurately compare both pre-processing techniques.

5.3 Additional #1: XLM-RoBERTa Feature Extraction

XLM-RoBERTa, Cross-lingual Model - RoBERTa, is a transformer-based model designed for natural language understanding across multiple languages[2]. It builds on the RoBERTa model, an optimized version of BERT (Bidirectional Encoder Representations from Transformers), and is trained on a diverse multilingual dataset. XLM-RoBERTa excels in capturing contextual nuances in text data, making it highly effective for tasks involving multiple languages and complex textual information. Its embeddings are contextualized word embeddings; unlike static embeddings, like GloVe's, these ones capture the meaning of a word based on the surrounding context within a sentence.

To extract the embeddings we resorted to HuggingFace's transformers library and sourced the pre-trained model and its corresponding tokenizer, the model checkpoint is '*xlm-roberta-base*'. We started by tokenizing our corpus. During tokenization, we ensured all sequences were of the same length by padding shorter sequences and truncating longer ones; this standardizes the input for the model. Then forward-passed them once over the network to extract the embeddings: we used the hidden state associated with the [CLS] token (marking the beginning of an input sequence), the last hidden state, to extract a representation akin to a sentence embedding. Each of our documents was now represented by a multi-dimensional vector of length 768.

At this stage, we created 5 new datasets. The driving rationale is that, as seen during the initial EDA, there was a big discrepancy between the target labels for houses with and without reviews. Therefore, we ought to create datasets, filtered by those two groups, to try and tackle both group's predictions separately. The first dataset acted as a medium of comparison with the previous techniques, it contains all properties in the dataset. The second dataset only contains properties with reviews. Datasets three, four, and five were all sliced to only have houses without reviews and differed solely by the features used. The third dataset's feature was a concatenation of the description and host_about features, whereas the fourth only comprised the description feature and the last one only the host_about. The pre-processed version of all features was the one clean, translated, and with NERs replaced, i.e. *ner_trans_clean_¡feature¿*.

Contrary to the previous embedding techniques, transformer-based models do not require heavily pre-processed data (e.g. stop word removal and lemmatization) to lend accurate representation. Reasons entail the massive amounts of data these models were trained on; the use of subword tokenization, which breaks down words into subwords, allowing the models to handle rare and out-of-vocabulary words effectively; and, as previously mentioned, the fact these models generate contextualized embeddings, that reduce the need for lemmatization or stemming, between others, because they capture the meaning of words based on their context, allowing the model to understand different forms of a word without reducing them to a base form.

Additionally, we visualized the datasets using the UMAP algorithm, as can be seen in the image below (Fig. 9. In simple terms, this algorithm reduces high-dimensional data to lower dimensions (in our case 2-dimensional), preserving structure and relationships, and making patterns and clusters easier to identify

[3]. The idea and code implementation were sourced from the book Natural Language Processing with Transformers[4].

The resulting plots are highly interesting (image is read from left-to-right and top-to-bottom), for the dataset with all properties, the first one, we see a clear split between the embeddings regarding each target label, correlated to the target distribution over properties with and without reviews dilemma. The second plot is much more homogeneous, implying a less pronounced distinction between properties with reviews based on their listing status. The last three plots seem to show mild heterogeneousness, hinting at the possible discriminatory power of the features.

Figure 9: UMAPs for XLMR Embeddings

5.4 Additional #2 - mBERT Feature Extraction

Multilingual BERT (*mBERT*) is a variant of the BERT model designed to handle multiple languages[5]. The model is trained on *Wikipedia* data from 104 languages, while *XLM-RoBERTa* is trained on a larger *CommonCrawl* corpus covering 100 languages. Additionally, they use different tokenizers, each with its own characteristics and methods. To utilize *mBERT* for feature extraction, we employed the same approach as the one used with *XLM-RoBERTa*, the only difference being the model checkpoint, which now was *'google-bert/bert-base-multilingual-uncased'*. The extracted embeddings also form uni-dimensional arrays of length 768.

Five new datasets were also created using the same data as the previous model. And its UMAP representations can be seen in the image below (Fig. 10). It seems both models tend to generate similar

embeddings as the qualities of each plot from before can be applied to the new array of plots.

Figure 10: UMAPs for mBERT Embeddings

A list of each of the embeddings datasets created and its main properties can be seen in the table below (Fig 11):

Embedding Type	Embedding Dataset Var. Name	Feature(s) Used	Dimensionality	Dataset Type (all / reviews only / no reviews only)	Language (all / <language code>)	Num. properties in Train set	Num. properties in Test set
TD-IDF	tfidf	lemma_no_stopwords_ner_trans_clean_comments	1000	all	en	4952	1239
GloVe	glove_50d	lemma_no_stopwords_ner_trans_clean_comments	50	all	all	4998	1250
	glove_stem	stem_no_stopwords_ner_trans_clean_comments	50	all	all	4998	1250
	glove_200d	lemma_no_stopwords_ner_trans_clean_comments	200	all	all	4998	1250
XLMR	xlmr_reviews	ner_trans_clean_comments	768	all	all	4998	1250
	xlmr_reviews_only	ner_trans_clean_comments	768	reviews only	all	3396	849
	xlmr_concat_no_reviews	lemma_no_stopwords_ner_trans_clean_description,lemma_no_stopwords_ner_trans_clean_host_about	768	no reviews only	all	1602	401
	xlmr_description_only	lemma_no_stopwords_ner_trans_clean_description	768	no reviews only	all	1602	401
	xlmr_host_about_only	lemma_no_stopwords_ner_trans_clean_host_about	768	no reviews only	all	1602	401
MBERT	mbert_reviews	ner_trans_clean_comments	768	all	all	4998	1250
	mbert_reviews_only	ner_trans_clean_comments	768	reviews only	all	1602	401
	mbert_concat_no_reviews	lemma_no_stopwords_ner_trans_clean_description,lemma_no_stopwords_ner_trans_clean_host_about	768	no reviews only	all	1602	401
	mbert_description_only	lemma_no_stopwords_ner_trans_clean_description	768	no reviews only	all	1602	401
	mbert_host_about_only	lemma_no_stopwords_ner_trans_clean_host_about	768	no reviews only	all	1602	401

Figure 11: Embeddings created and their main properties

VI. Model Development

6.1 Rule-Based Model

The first model we developed was a rule-based model. As we've seen during the initial EDA we knew that if a property had a least one review then its chance of being unlisted was around 6% (and 94% of remaining listed). Whereas, if a property had no reviews then its chance of being unlisted skyrocketed to around 72% (28% of remaining listed). Therefore, the rule-based model we developed, which was afterward used as our baseline, was as simple as predicting the positive class (unlisted) if the property had no reviews and the negative class (listed) otherwise, i.e. had reviews.

This model, as expected because it is not being fitted to any data, achieved virtually equal results in our train (left) and test datasets (right), achieving an f1-score for the positive class of 0.78 and an accuracy of 0.87, as can be seen in the image below (Fig 12).

	precision	recall	f1-score	support
0	0.94	0.88	0.91	3631
1	0.72	0.84	0.78	1367
accuracy			0.87	4998
macro avg	0.83	0.86	0.84	4998
weighted avg	0.88	0.87	0.87	4998

	precision	recall	f1-score	support
0	0.94	0.88	0.91	909
1	0.72	0.84	0.78	341
accuracy			0.87	1250
macro avg	0.83	0.86	0.84	1250
weighted avg	0.88	0.87	0.87	1250

Figure 12: Top 10 TF-IDF Importances for Pre-Processed "comments" feature

It is also worthwhile to look at the confusion matrix below (Fig. 13). We can observe a well-balanced model across both labels. Nonetheless, we see that it commits more Type I errors (i.e. false positives: predicting that the property will be unlisted where in reality it will not) than Type II errors (i.e. false negatives, predicting that the property will remain listed whereas it will be unlisted). These results are a direct consequence of the rule-based approach: if the house has reviews and it predicts that they'll remain listed it will get it right 94% of the time, meaning a low FN rate. However, when making predictions on properties without reviews, that they'll be unlisted, it will only get them right about 72% of the time, implying a higher FP rate. As a consequence, Precision is the metric that takes the biggest hit (0.72) as it is closely related to the FP rate - it measures how often the model correctly predicts the positive class, i.e. Precision = TP / (TP + FP).

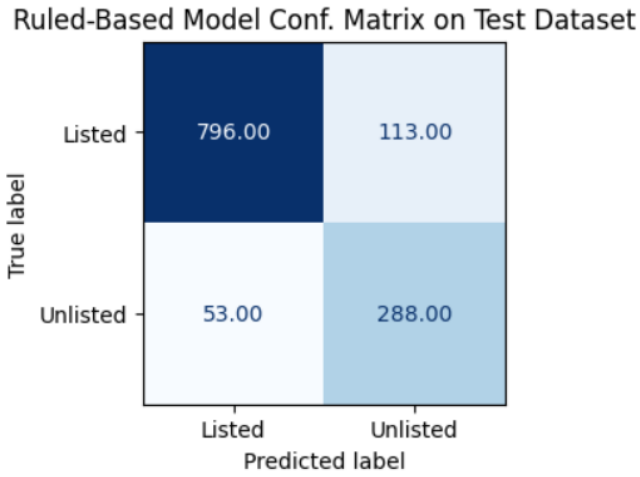

Figure 13: Confusion Matrix of Rule-Based Model

6.2 Non-sequential Models

6.2.1 K-Nearest Neighbors, Logistic Regression and Random Forest

For all the next scikit-learn classifiers we employed the same methodology. We moved all embedded datasets (14 total: 1 using TF-IDF, 3 using GloVe, 5 using XLM-RoBERTa, and the other 5 using mBERT) to a dictionary and, for each model, fitted the embedded train data and then evaluated the model on the train and test data (to check for overfitting and generalization issues) and stored the results. For all models, we opted to output prediction probabilities rather than binary values. However, the following results are all based on the default probability prediction threshold of 0.5. The models are: (1) 'KNeighborsClassifier()', (2) 'LogisticRegression()', and (3) 'RandomForestClassifier()'. The lack of explicit parameters means that we used the default and sensible values of the algorithm's implementation of scikit-learn. For models 2 and 3, we also tried adding balanced class weights due to target imbalance in our dataset - as a comparison medium, for both models, we used the glove dataset, for all reviews, and 50 dimensions. It seemed that the results converged to the same local minima.

We also tuned the hyper-parameters of these models using a grid search approach with 5-fold cross-validation (here enters the TF-IDF particularity mentioned in chapter 5.1., to avoid data leakage we fitted the vectorizer to the train folds and transformed the train and test folds, at every iteration of CV). The results did not improve for any of the models. Further reasoning will be provided in the next chapter, 6.2.2.

The results in the test sets can be seen in the following tables - Fig. 14 (for simplicity's sake we will not be showing the results in the train set to avoid a too large table as well as only showing the results for accuracy and F1 scores of the pos. class). Please note that the results are grouped not only by model and embedding but also by dataset type (all properties, only with reviews or only without reviews), models trained on differently sliced datasets should not be compared against others for now (they will be in chapter 7.1).

		KNN		LR		RF	
Dataset Type	Embedding Dataset Var. Name	Accuracy	F1	Accuracy	F1	Accuracy	F1
All reviews	tfidf	0.866	0.776	0.867	0.776	0.866	0.776
	glove_50d	0.866	0.775	0.867	0.776	0.866	0.775
	glove_stem	0.866	0.775	0.867	0.776	0.867	0.776
	glove_200d	0.867	0.776	0.867	0.776	0.867	0.776
	xlmr_reviews	0.866	0.775	0.867	0.776	0.866	0.774
	mbert_reviews	0.866	0.775	0.867	0.776	0.866	0.775
Reviews only	xlmr_reviews_only	0.938	0.000	0.938	0.000	0.936	0.036
	mbert_reviews_only	0.938	0.000	0.938	0.000	0.938	0.000
No reviews only	xlmr_concat_no_reviews	0.743	0.836	0.718	0.836	0.776	0.865
	xlmr_description_only	0.741	0.840	0.718	0.836	0.750	0.850
	xlmr_host_about_only	0.741	0.834	0.718	0.836	0.786	0.862
	mbert_concat_no_reviews	0.741	0.834	0.763	0.856	0.791	0.871
	mbert_description_only	0.721	0.825	0.728	0.837	0.751	0.850
	mbert_host_about_only	0.751	0.834	0.758	0.852	0.781	0.856

Figure 14: Results for the Scikit-Learn Models across all Embedded Test Datasets

Overall, the majority of models did not show signs of overfitting, except for RF models, and most interesting of all models of the dataset type 'All Reviews' achieved very similar results between them and the rule-based model (recall the 0.78 f1-score achieved by the rule-based model). The same can be said for the other dataset types.

6.2.2 Comparing Predictions

Initially, our hypothesis for such similar results was that there was a surreptitious bug in our code. However, after deep diving into error analysis, we concluded that our models all converged to the same local minimum. That local minimum was characterized by our models predicting virtually always the positive class for properties represented by embeddings of the 'unknown' word/token, i.e. if that property was not reviewed by anyone then the model would predict it would be unlisted and the negative class otherwise, i.e. if not represented by the 'unknown' embedding representation then it would predict that house would remain listed. This is exactly like the logic proposed in our rule-based model. The image below (Fig. 15) contains proof of the model's predictions. To tie the loose ends of the hyperparameter inefficiency rationale, empirical tests showed that the models would always converge to similar minima given a very broad array of hyperparameters (albeit some combinations could prove so terrible that the model would not be able to get there).

```
HOUSES WITHOUT REVIEWS:

True Labels:
unlisted
1    0.718204
0    0.281796
Name: proportion, dtype: float64

KNN Predictions:
knn_pred
1    1.0
Name: proportion, dtype: float64

LR Predictions:
lr_pred
1    1.0
Name: proportion, dtype: float64

RF Predictions:
rf_pred
1    1.0
Name: proportion, dtype: float64
```

```
HOUSES WITH REVIEWS:

True Labels:
unlisted
0    0.937574
1    0.062426
Name: proportion, dtype: float64

KNN Predictions:
knn_pred
0    0.998822
1    0.001178
Name: proportion, dtype: float64

LR Predictions:
lr_pred
0    1.0
Name: proportion, dtype: float64

RF Predictions:
rf_pred
0    0.998822
1    0.001178
Name: proportion, dtype: float64
```

Figure 15: Comparing Predictions of the KNN, LR and RF Models

As for the other dataset types a similar logic was observed: for datasets consisting of only properties with reviews they always converge to predicting the 'listed' label. And for datasets consisting of properties with only reviews, they would favor the 'unlisted' outcome, as they should.

6.3 Ungrouped Data

We also experimented with not grouping our data straight away by property index but rather making predictions on 'independent' comments and properties pairs. And then aggregate our predictions using the mode. The rationale for grouping the comments data by property was that it allowed the models to capture a holistic view of each property that would be less affected by noisy data, e.g. the fact that individual reviews could contain outliers/extreme opinions. However, we also hypothesized that using grouped datasets would possibly make the models unable to capture minute nuances that could affect the overall listing of the house, e.g. repeated mentions of a severe issue.

Therefore, we developed a simple Logistic Regression model, using *GloVe's 6B* embeddings of 50 dimensions of the *lemma_no_stopwords_ner_trans_clean_comments* feature for the ungrouped data. For this model, we were compelled to use class weights because the target imbalance ratio would skyrocket from around 70-30 to 99-1. After making predictions on the independent pairs they were aggregated by property using the mode. The resulting model, albeit slightly worse, still achieved respectable results: it returned an f1 score of 0.71 and an accuracy score of 0.81 on the test dataset (all properties) compared to 0.78 and 0.87 f1 and accuracy scores of the rule-based model, respectively.

6.4 Additional #1: XLM-RoBERTa Fine-Tuning

Due to their simplicity and inability to handle sequential data the previous models might have lacked the capability to capture complex patterns and contextual meanings in the data. Moreover, the use of pre-trained transformer-based models would help us leverage the existing pre-trained knowledge from large-scale language models and possibly aid us in enhancing our results.

Due to GPU constraints (these models had to be run in Google Colab, where there is a limit to the compute we have access to), we only fine-tuned transformer-based models in one dataset - only properties without reviews using the *ner_trans_clean_description* feature. The use of that dataset was based on its smaller size and the fact that achieving a model better than one that gets its predictions right 72% of the time would be easier than beating one that gets them right 93% of the time (all properties). Additionally, the use of that feature was based on the UMAP analysis (showed possible discriminatory prowess) and the fact that the *host_about* column had various repeated documents (as seen during EDA the same host can be mapped to several properties) besides missing values.

To fine-tune these models we resorted the HuggingFace's transformer libraries to extract the models and their tokenizers and wrote the training and evaluation loops using PyTorch (initially we tried using the Trainer class from HF however we got stuck with an issue we were not able to solve, at every iteration the model would output the exact same values for the tracked metrics. We believe it might have something to do with the '*output_dir*' argument).

Regardless of the implementation, the first model fine-tuned was XLM-RoBERTa. We tried different combinations of batch sizes (8, 16, and 32) and learning rates (1e-3, 1e-4, 1e-5, and 1e-6). Additionally, we also implemented the possibility of class weighting. The used optimizer was Adam and the loss function was Cross-Entropy Loss.

The results for the different combinations (batch sizes, learning rates, and class weighting) all pointed towards the same conclusion. Although it might have taken less or more epochs to reach there all model's configurations always ended up overfitting to the train data, never being able to accurately generalize to unseen instances. It is not that the configurations did not learn, because they achieved similar f1 scores to the expected values, around 0.84, they just stopped learning and started memorizing somewhere around the end of the first epoch. Below is an example of a full model config. run over 15 epochs for a batch size of 16, the learning rate of 1e-5, and without class weighting. (Fig. 16

Figure 16: Fine Tuning History of XLMR Model

Possible reasons for the inability of fine-tuned model configurations to generalize are: (1) the feature used had no discriminatory power, (2) albeit fine-tuning does not require massive amounts of data the data passed might be too small, and (3) not freezing any layers allowed for the complexity of the model

to take charge and quickly and easily memorize all our data, between others.

6.5 Additional #2: mBERT Fine-Tuning

The results obtained from fine-tuning mBERT configurations were very similar to the previous model. We faced the same issues and their existence is most likely due to the same reasons. Nonetheless, the image below (Fig. 17) pictures a full run of 15 epochs a batch size of 16, a learning rate of 1e-5, and with class weighting this time (as opposed to before where none was applied).

Figure 17: Fine Tuning History of mBERT Model

6.6 Bonus Additional #3: Text Summarization and Classification

Out of curiosity, and not counting as an additional step, we also wondered if we could first summarize the descriptions data to ideally enhance signal and reduce noise since the majority of descriptions pertain to the same topics, e.g. house size and amenities description, location and how far from tourist places, between others. We first extracted the data, and as an additional pre-processing step we used the model '*sshleifer/distilbart-cnn-12-6*', pre-trained on summarization tasks, to summarize the English descriptions, to a minimum length of 25 tokens and a maximum of 50. We then extracted the GloVe embeddings of the documents (again, used the 6B model with 50 dimensions) and fitted a Logistic Regression classifier. Although no breakthrough was achieved, the results are on par with the expected from models fitted to this dataset type (no reviews only). (Fig. 18)

	precision	recall	f1-score	support
0	0.83	0.02	0.03	320
1	0.74	1.00	0.85	904
accuracy			0.74	1224
macro avg	0.79	0.51	0.44	1224
weighted avg	0.77	0.74	0.64	1224

	precision	recall	f1-score	support
0	1.00	0.03	0.07	88
1	0.73	1.00	0.84	229
accuracy			0.73	317
macro avg	0.86	0.52	0.45	317
weighted avg	0.80	0.73	0.63	317

Figure 18: Model Results with additional Pre-Processing step of Text Summarization

6.7 Extending Rule-Based Model

Lastly, we tried two other approaches to enhance our scores.

6.7.1 Sentiment Analysis

We concluded from the Chi-Square independence test in the EDA post pre-processing chapter - chapter 4 - that the comments feature and the target were dependent. Therefore, we aimed at building a model that instead of being trained on predicting listed vs. unlisted properties would solely predict the sentiment of all the comments of a property, and if the sentiment was positive then the house would remain listed and unlisted otherwise. For houses without reviews, we followed our logic and predicted that they would be unlisted.

This model, once again, did not beat the rule-based model but got pretty close, achieving an f1 score of 0.76 on the test set. It now seemed that this model, if combined with other features, would surpass our baseline. Hence we set out to create it (next chapter).

6.7.2 Text Attribute Extraction

Instead of only resorting to the sentiment of a group of comments we added the following features: (1) the number of comments on the property, (2) the average comment length, (3) the number of different languages the reviews are in, (4) the description length, (5) the host about length and finally (6) the number of houses belonging to that house's host. We had already explored each of these features before and/or after pre-processing and although they did not seem to have discriminatory power individually, when combined they might have had. This is a typical aspect of statistical modeling/machine learning.

The data was fitted to a Decision Tree classifier (for us to easily check feature importance) that was tuned over a grid search - the best parameters are: no class weighting, maximum depth of 5, 16 minimum samples per leaf, and 2 minimum samples to split. Unfortunately, the model achieved virtually the same results as the rule-based model and after an error analysis, it seems that once again it got stuck in the same local minima. This is evidenced by looking at the importance plot (Fig. 19) and inferring that the most important feature, by an extremely wide margin, is the comments sentiment feature, which contains the sentiment of the comments of the property but would assume a value of -1 as a placeholder for houses with no comments. This means the model learned that if the sentiment was different from -1 it would predict the house would remain listed and unlisted otherwise.

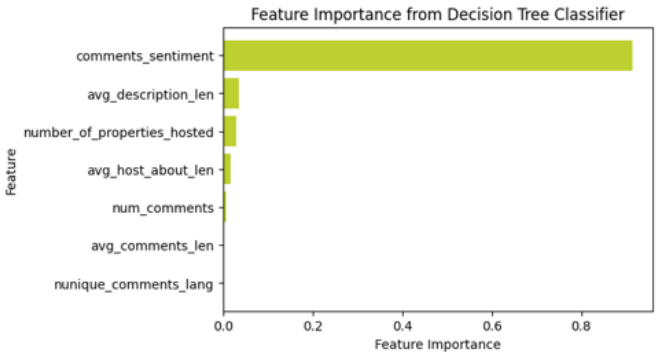

Figure 19: Feature Importance Plot

VII. Model Comparison

7.1 Fixing Predictions

Despite no model seemingly beating the rule-based model we still gauged their potency on the whole of the train and test datasets. As mentioned several times before some datasets pertain to different houses

than others - some to houses with either reviews or no reviews, some to only houses with reviews, and others solely to houses without reviews. Therefore, we 'fixed' the model's predictions; by fixed we mean that if a prediction array, be it from the train or test sets, has fewer instances than the maximum number of the set, i.e. predictions do not pertain to all the possible instances, then synthetic predictions were made following the aforementioned logic.

To illustrate with an example: train datasets composed only of properties without reviews (across all languages) only pertain to 1602 properties whereas there are a total of 4998 properties in the train set, meaning 4998 - 1602 = 3396 properties are missing a prediction. To be able to evaluate models accurately we have to make sure they all make predictions on the full train and test datasets. How to? By keeping the predictions as they are for the houses they've been trained and tested on but for houses not in the datasets then fabricate a prediction following the logic in the rule-based approach. We would now complete the prediction array by making predictions on the 3396 houses with reviews (according to the logic we would predict that they would remain listed). We also made sure the index order matched to ensure consistency.

7.2 Threshold Tuning and Cross-Validation of Results

As a last resource measure to beat the rule-based model, we experimented with tuning the prediction threshold of all models, ranging from LR + glove_50d to fine-tuned XLMR on the description of properties without reviews. Models without the ability to output prediction probabilities were skipped.

The majority of models did not seem to beat the baseline except for the following:

Classifier	Feature	Threshold	Accuracy	F1 Score	Precision	Recall
KNeighborsClassifier	xlmr_host_about_only	0.1	0.885	0.800	0.845	0.760
RandomForestClassifier	xlmr_host_about_only	0.2	0.890	0.806	0.841	0.774

Figure 20: Best Models after Threshold Tuning

Please note that if the thresholds presented seem to be too low it is attributed to the fact that models like KNN and RF calculate the prediction probabilities in different ways from models that are trained by optimizing a gradient, like LR and Transformers. KNN probabilities are calculated based on the proportion of neighbors that belong to each class and for RFs the probability is calculated based on the proportion of trees that voted for the predicted class.

Since the scores from this step are solely based on a fixed test set we had to ensure that these models' scores were actually due to being better models rather than no more than a mere coincidence given the unrepresentativeness of the test data. We enforced this idea by performing a 5-fold cross-validation loop over the data, where at each iteration we would fit the fold data to the model and evaluate its predictions after being tuned for their 'ideal' threshold. The results are as follows:

	KNN		RF	
	Mean	Std	Mean	Std
Accuracy	0.746	0.010	0.780	0.009
Precision	0.742	0.007	0.769	0.007
Recall	0.993	0.005	0.992	0.004
F1	0.849	0.005	0.867	0.005

Figure 21: Cross-validation Results of the best models after Threshold Tuning

Be aware that these results ought not to be compared to the results across all properties but rather against the results on the same dataset (in both cases for properties without reviews). KNN xlmr_host_abo ut_only got a score of around .834 and RF xlmr_host_about_only .8617. This is by no means a statistically backed test but looking at the result's f1 standard deviation we can see that those values (.849 and .866, for KNN and RF respectively) are probably not significantly higher than the ones retrieved from the test set and therefore they may not trump the rule-based model after 'fixing' the predictions.

7.3 Final Solution

After a thorough model development, error analysis, and tuning steps, we ended up with the same model we started with. Although several models came really close to beating the baseline, none seemed to soar above it, across any of the tracked metrics of accuracy, f1 score, precision, and recall. Therefore, we chose the baseline model as our final model - a thorough explanation of this model, the rationale for its creation, and its evaluation can be revisited in Chapter 6.1.

Moreover, by the initial EDA, we know that the ratio of houses with reviews to houses without reviews is virtually the same between the real train and test sets. And our predictions on the real test set also point to a similar ratio of listed to unlisted present in the train dataset. Therefore, we can safely assume our model is minimally robust and should be able to generalize to unseen instances.

VIII. Conclusion and Limitations

In this project, we explored a variety of methods to predict whether a property would remain listed or be unlisted. Our journey through different natural language processing (NLP) techniques included traditional methods like TF-IDF vectorization, GloVe, and some more advanced transformer-based models like XLM-RoBERTa and mBERT.

Despite the advanced capabilities of transformer models, our results revealed that the simpler, rule-based model achieved the best performance - a very interesting inference to make. This model, which relied on the straightforward logic that properties with at least one review were more likely to remain listed, achieved an F1-score of 0.78 on the positive class and an accuracy of 0.87.

In summary, our analysis underscores the importance of balancing model complexity with practical performance. The rule-based model's success demonstrates that in certain cases, simple models grounded in logical assumptions can be highly effective.

The main limitations of our work can be attributed to the lack of features in the datasets. Possibly with additional data, like review ratings and super host status, we could be able to create an ensemble model that would feed of the best attributes of each model and finally beat the rule-based model.

References

[1] Rodrigues A., (2021, June). Text Mining of Airbnb Reviews, A holistic approach to reviewers' opinions and topics distribution, NOVA IMS

[2] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., & Stoyanov, V. (2019, November 5). Unsupervised cross-lingual representation learning at scale. arXiv.org. https://arxiv.org/abs/1911.02116

[3] McInnes, L., Healy, J., & Melville, J. (2018, February 9). UMAP: uniform manifold approximation and projection for dimension reduction. arXiv.org. https://arxiv.org/abs/1802.03426

[4] Tunstall, L., von Werra, L., & Wolf, T. (2022). Natural Language Processing with Transformers: Revised Edition. O'Reilly Media

[5] Papers with Code - mBERT Explained. (n.d.). https://paperswithcode.com/method/mbert