Predicting Airbnb Unlisting

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Text Mining 2023

Group 23

Lukas Stark, number: 20220626

Felix Gayer, number: 20220320

David Halder, number: 20220632

Ricardo Montenegro Dona, 20221359

June, 2023

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

INDEX

Table of figures

1. Introduction

Utilising a dataset of 12.496 Airbnb listings with their description and 721.402 comments from past visitors, the objective of this project is to predict whether a property will be unlisted next quarter or if it will remain listed on the platform. To assess the quality of the approach taken, two more datasets were given containing 1.389 unlabeled Airbnbs and 80.877 comments. These are the properties for which the listing status for next quarter needs to be predicted.

To solve this problem, several preprocessing and feature engineering steps were conducted as well as several embeddings and models developed, tried, and assessed. Finally, a Random Forest with Skip-Gram embeddings was hyperparameter-tuned to predict the test data given.

The report is structured as follows. First, the data is presented, and the findings of the data exploration are explained. Next, the developed pre-processing pipelines and the methods within are explained in detail. Following the feature engineering methods and the created features are presented and explained. Subsequently, all the classification models tried are shown and discussed. Finally, the results are presented, and the final predictions are provided for the assessment.

2. Data Exploration

In this section, we present the findings of our exploratory data analysis conducted on two datasets, train and train_review. The primary objective of this analysis was to gain a comprehensive understanding of the data, identify any anomalies or inconsistencies, and take insights that could be important for further data processing and modelling tasks.

First, Data Types, Missing Values, Empty String, and Duplicates were checked. In the train dataset, the columns index and unlisted are composed of integers. On the other hand, description and host_about are considered objects. In relation to missing values, and empty string none were found in any column. The same result was seen when the existence of duplicates was tested. Regarding the train_review dataset, the values in index and comments columns were considered as integers and objects, respectively. Besides, no missing values or empty strings were found in this dataset. However, oppositely to the previous mentioned dataset, in train_review there were 305 duplicates, which can be logically explained, since it is possible to have the same comment for the same Airbnb property more than once.

Next a graphical analysis was conducted. In relation to the train dataset, we started by doing a balance check to the target variable, unlisted. The results were that 72.3% of the Airbnb properties remained in the quarterly Airbnb list, and 27.7% got removed (see Annex 1).

Next, the most frequent tokens were analysed in the description and host_about columns. The analysis was split in two (See Annex 2, left). On the left of, all tokens were considered for the bar plots. However, as it was noticed that there were many unnecessary symbols as tokens in both columns, we created the charts on the right, not considering them. As a result, for the column description, the most common tokens in the analysis that considered every item were “<”, ”>”, “/”, br, “-“, and “etc” (Annex 2, left). In host_about column it was “.”, “‘”, “!”, “*”, “_x000d”, and “etc”. Regarding the second analysis, the one that only considered important items, the most common tokens in the description column were “apartment”, “de”, “Lisbon”, “space”, “e”, and “etc”. In host_about column were “e”, “de”, “Lisbon”, “love”, “Portugal”, and “etc”.

The same analysis was done again grouped by the target variable. Meaning that, we analyzed the most common tokens in a dataset that only contained information about the listed properties (See Annex 2

right). The same was done for the unlisted ones (See Annex 3). The results do not differ a lot when compared with the most frequent tokens in the entire dataset.

Subsequently, the most common languages in the dataset were detected. In the description column the most frequent ones were English, Portuguese, French, and Dutch. In relation to the column host_about the same results were English, Portuguese, French, and Spanish (See Annex 4).

Concerning the train_reviews dataset, we started by visualizing throughout bar plots the most common tokens. In the analysis that considered all the possible tokens, the most present ones in the comment's column were ".", "'",!", "<", ">", and "etc". In the analysis that disregard symbols and other unnecessary tokens, the most frequent were "Apartment", "de", "great", "place", "stay", and "etc" (See Annex 5, left).

In relation to the visualisations in Annex 5 (right) and 6, they correspond to the most frequent tokens only for unlisted and listed properties, respectively. Regarding, the comments considering all possible tokens the top 6 comments are the same if we do not consider them in any order, ".", "'", "<", ">", br, and "!". On the other hand, when unecessary tokens are discarded the top 6 most common tokens used in comments are the same for listed and unlisted properties, "apartment", "de", "great", "location", "stay", and "place".

Again, the most common languages were analysed, where once again English was the most dominant one.

Finally, it was analysed which property had the highest number of comments (See Annex 7). Besides, on the right side of Annex 7 it is possible to see what the most common comments were. The "." is the most present one, and great location is another one that appears many times.

3. Preprocessing

For the data preprocessing two extra methods were conducted. First, language detection was applied, to lemmatize words from nine different languages and to conduct a sentiment analysis. Furthermore, part-of-speech tagging was used for the English texts to understand the grammatical structure of the sentences provided by and to improve lemmatizing.

For the pre-processing two different pipelines were created, which will be explained in detail. The first one the "Standard-Pipeline" cleans the data and applies methods like lemmatization and stemming. This pipeline is also part of the "Bert-Pipeline". However, the "Bert-Pipeline" applies additional steps only used by pre-trained encoder model BERT. This will be discussed later when explaining the implementation of BERT.

Before defining the first pre-processing function several Regex-Patterns were set, to remove expressions which do not contain any valuable information for the analysis. Next the preprocessing function was defined. After extracting the two features, a set of stop words for the English language is saved utilising the NLTK-library (See NLTK, 2023). Next, an instance of a Lemmatizer is created and saved in a variable for later use, as well as stemming dictionary for nine other languages. In the next step the data gets transformed to all lowercase as a first normalising step and to reduce the vocabulary size and make the data compatible with the later used machine learning models. In the same step the categorical data is encoded using American Standard Code for Information Interchange to further normalise it to ensure that for example words like "café" and "cafe" are recognized as similar. In the next step duplicates are dropped and empty cells are filled with empty strings, so that they are

compatible with the data modelling. Now, the Regex-Patterns described earlier are used to remove unwanted regular expressions from the data. In the next step, the language of the description and the comments is detected and added as a column to the dataframe. This information is then later used for part-of-speech (POS) and lemmatization. Next, the text is tokenized, and the earlier defined stop words are removed. This is done to remove noise in the data and to focus on the tokens which contain useful information.

Now the POS-tagging is conducted for the English comments and descriptions. To do so it is iterated through the two independent variables and the respective observations available and if the language is detected as English, the POS-tagging assigns a grammatical label to each word available based on the context of the sentence. Using this information, a dictionary is built to transform the abbreviations which were created in the previous step into WordNet POS tags. The information gained through the POS-tags is then used to reduce the tokens to their root form through lemmatization. Since this information was only retrieved for the English words, the tokens belonging to another language are stemmed, which means that the last few characters of a given word are removed. This is done so that tokens which contain the same meaning are also recognized as the token.

4. Feature Engineering

In this chapter the following techniques were applied on top of the minimum requirements. First, Skip-Grams were made as input for the defined machine learning models. Furthermore, BERT embeddings were created for the eponymous encoder model. Additionally, a sentiment analysis was developed to create more features for machine learning purposes.

4.1. Sentiment Analysis

The first feature engineering step is a sentiment analysis, utilising the Valence Aware Dictionary and Sentiment Reasoner (VADER) library. VADER is a rule- and lexicon-based sentiment analysis library that is particularly attuned to sentiments expressed in social media. (See Hutto & Gilbert, 2014) This is also the main reason why we decided to use tool, since the comment section can be classified as part of social media.

The motivation behind conducting this analysis was to capture the opinions of the users but also to assess their experience staying at the Airbnb. Especially, for the given objective this information is crucial.

The function works as follows. First, an instance of the “SentimentIntensityAnalyzer” class is created and saved as a variable. The output of the class is a dictionary containing the compound score, and the probabilities that the comment is positive, neutral or negative in a softmax-matter. The category with the highest probability is then taken to classify the comment. Comments which are not written in English or are empty are classified as neutral.

The data is then grouped by the Airbnb-ID and the compound score and the numbers of comments per category and overall are calculated and added as features. This dataframe is saved as “train_review_df”.

In the final steps of the feature-engineering the preprocessed data, containing the target variable, the comments, and the description of the Airbnb, and the newly created “train_reviews_df” are merged

on the Airbnb ID. Before returning the independent and dependent variables the empty numeric columns are imputed with the value “0”. For the categorical values this is not necessary, since it was done during the preprocessing. This procedure is then repeated with the data for the BERT Model.

Finally, a stratified train test split for the training of the models is applied to both the “Standard” and the “BERT” dataset. To make validation results comparable, both datasets are split in the exact same way, utilising the indexes of the first split to do so. In the end both datasets were separated in a 80 to 20 Train-Test split.

4.2. Vectorization & Embeddings

After creating our train-test split, the next step in our text mining process involves vectorization and embeddings. This crucial step aims to transform the textual columns into a numeric representation. Since most machine learning algorithms operate on numeric inputs, it is necessary to convert text into a numerical form that captures the semantic and contextual information present in the data.

To accomplish this, we explored several options, namely Bag of Words, TF-IDF as well as the extra methods stated at the beginning of the chapter (Skip Grams & DistilledBERT). Each method takes the input of x_train, x_val, x_test, as well as technique-specific hyperparameters, and returns the transformed inputs.

4.2.1. Bag of Words

This procedure is the simplest option among the techniques we examined. It represents text by creating a "bag" of words and counting their occurrences. To address memory constraints, we limited the bag of words to the most common 1000 words using the max_features parameter. This restriction not only helped optimize memory usage but also sped up the modelling calculations. Importantly, this limitation did not lead to worse scores, which is why we decided to stick with it.

4.2.2. TF-IDF

Term Frequency-Inverse Document Frequency is a slightly more advanced method compared to Bag of Words. It considers not only the frequency of words but also their importance by incorporating inverse document frequency. Similar to Bag of Words, we used the max_features parameter, set to 1000, to mitigate memory constraints. Again, this approach ensured efficient processing and did not compromise the model's performance.

4.2.3. Skip Grams

This was the first embedding method we applied. Skip Grams aim to capture the semantic meaning of words by considering the context in which they appear. Unlike Bag of Words or TF-IDF, Skip Grams create a vector representation for each word based on its neighboring words. This approach provides a more nuanced understanding of the text's meaning. The additional parameters and their values for Skip Grams include vector_size=100, window=5, and min_count=1.

4.2.4. DistilledBERT Word Embeddings

In addition to these traditional approaches, we also explored the use of pre-trained BERT (Chaumound et al.,2019) for text vectorization. Since our objective was to obtain embeddings rather than perform generative tasks, the smaller variant of BERT was sufficient for our project.

When transforming the text data into vectors using the pre-trained BERT model, we adopted a unique approach. Initially, we removed stop words and applied standard preprocessing steps to enhance the density of meaningful tokens. To handle the computational intensity of the BERT model and preserve the distinction between the "host_about," "description," and "comments" columns during training, we devised a new strategy.

We observed that the first section of the "description" and "host_about" columns often contained important information, while the second part frequently consisted of the same text in a different language. Conversely, for the "comments" column, we aimed to standardize the length of comments and summarize their content effectively. To achieve this, we divided the texts of the "host_about" and "description" columns into sentences. Additionally, we combined all comments for each host into one large text and divided it into sentences as well. From these sentences, we selected the first twenty for "description" and "host_about" and 20 random sentences for the comments column to provide a representative representation of the content.

Next, separate chunks were formed for each column, with a limit of 512 tokens per chunk. These chunks were vectorized using the BERT model. If multiple chunks were created for the selected sentences, the resulting vectors were aggregated. By implementing this approach, we generated a vector representation (with 768 components) for each host and for each of the three columns.

Finally, we saved the outputs of these vectorization and embedding functions as a checkpoint. Before proceeding with modelling, we additionally applied SMOTE oversampling to address the dataset's imbalance. By oversampling the minority class, we aimed to improve the model's ability to learn from the underrepresented samples.

5. Modelling

In this chapter, we delve into various modeling techniques to analyze the dataset using vectorization and embedding methods. We begin by loading the pre-saved vectorization and embeddings, including Bag-of-Words, TFIDF, Skip-Grams, and DistilledBERT Embeddings, that were generated in the feature engineering chapter. In addition to the obligatory models, a Recurrent Neural Network Model along with a column-based stacking approach have been developed as extra work.

5.1. Scikit-Learn Classifiers

To start our analysis, we employ several applicable models from the sklearn library, namely KNNClassifier, Logistic Regression, Random Forest, and MLPClassifier. We evaluate their performance by comparing ROC/AUC curves, utilizing the default hyperparameters for each model. To further assess the classifiers, we train them on different train splits associated with the vectorization and embedding methods. Subsequently, we evaluate the individual models on the validation split to identify the optimal classifier and vectorization/embedding method for our specific problem.

The K-Nearest Neighbor Classifier, Logistic Regression, and Random Forest were trained using their respective default parameters. However, for the Multi-layer Perceptron Classifier, we decided to adapt the hyperparameters to the following:

- hidden_layer_sizes=(100,50,10)
- max_iter=500
- early_stopping=True

Despite applying Synthetic Minority Oversampling (SMOTE), it became evident that all classifiers struggled with correctly labelling the minority class. As a result, we experimented with an additional technique called threshold shifting, employing a Random Forest in conjunction with TF-IDF as the feature engineering technique. Unfortunately, this approach did not yield any improvements in the scoring metrics.

5.2. Tensorflow Recurrent Neural Network

For the RNN model, we focused on utilising distilledBERT word embeddings. To incorporate these embeddings into the Tensorflow library, we first needed to preprocess and convert them into tensors for training. Regarding the RNN model architecture, we opted for a simple design consisting of three layers:

- SimpleRNN Layer (units = 50, activation = tanh, return_sequences = True)
- SimpleRNN Layer (units = 20, activation = tanh)
- Dense Layer (units = 1, activation = sigmoid)

We experimented with BatchNormalization to address the risk of vanishing gradients, but it did not lead to any improvements, so we excluded it from the final model. The RNN model utilized BinaryCrossentropy as the loss function and the Adam optimizer with a learning rate of 0.01.

5.3. Tensorflow Stacking

In the stacking approach, we directly used DistilledBERT word embeddings within the Tensorflow model. Throughout experimentation, we recognized the importance of comments in the dataset, even though they were only available for a limited number of hosts. To focus the models on the "description," "host_about," and "comments" columns, we adopted a stacking model that places more emphasis on the text within these individual columns.

The stacking model consisted of three BERT classification models, each dedicated to one of the mentioned columns. All three models shared the same architecture. The predictions from these models were then used as input to a simple MLP model (stacking model) for the final classification. To implement this approach, we organised the data into separate text files and converted them into a Keras dataset. The models followed a flow where textual data passed through pre-processing layers provided by Google's pre-trained BERT model, resulting in encoded data that was then passed through an output dense layer.

5.4. Final Model

After exploring two additional advanced models in our text mining project, we ultimately selected a RandomForestClassifier from the sklearn library as our final model for making predictions on the test data. This model, in combination with SkipGrams embeddings, demonstrated good performance compared to the other options considered. Despite performing a GridSearch to optimize the hyperparameters, it did not result in significant improvements in the evaluation metrics. Nevertheless, we utilized the best parameters obtained from the grid search, which differed from the default values and were set to max_depth=30 and n_estimators=300.

6. Evaluation and Results

To compare the performance of the classifiers and word representation methods used, a matrix was created with all the combinations tested. The evaluation metrics (accuracy, precision, recall, F1 score, values of the confusion matrix) are based on the described train-test split.

Due to the uneven distribution of classes in our training data, the metric accuracy should be used with caution. For us, a balance between recall and precision is important. For this reason, we focus on the metric F1 in the evaluation.

Index	Classifier	Word Representation	Accuracy	Precision	Recall	F1-Score	TN	FP	FN	TP
1	K-Nearest-Neighbor	Bag-of-Words	0.67	0.45	0.93	0.61	862	620	38	516
2	K-Nearest-Neighbor	TF-IDF	0.80	0.59	0.90	0.71	1095	351	57	497
3	K-Nearest-Neighbor	Skip-Grams	0.84	0.66	0.88	0.75	1194	252	68	486
4	K-Nearest-Neighbor	Word Embeddings (BERT)	0.82	0.63	0.88	0.73	1160	286	68	486
5	LogisticRegression	Bag-of-Words	0.83	0.64	0.82	0.72	1196	250	100	454
6	LogisticRegression	TF-IDF	0.87	0.71	0.89	0.79	1250	196	63	491
7	LogisticRegression	Skip-Grams	0.87	0.73	0.86	0.79	1273	173	80	474
8	LogisticRegression	Word Embeddings (BERT)	0.88	0.73	0.86	0.79	1274	172	77	477
9	RandomForest	Bag-of-Words	0.90	0.80	0.84	0.82	1328	118	89	465
10	RandomForest	TF-IDF	0.90	0.80	0.83	0.82	1330	116	93	461
11	RandomForest	Skip-Grams	0.89	0.79	0.84	0.82	1322	124	87	467
12	RandomForest	Word Embeddings (BERT)	0.89	0.79	0.84	0.82	1322	124	87	467
13	MLPClassifier	Bag-of-Words	0.88	0.79	0.78	0.78	1331	115	122	432
14	MLPClassifier	TF-IDF	0.89	0.79	0.80	0.80	1331	115	110	444
15	MLPClassifier	Skip-Grams	0.88	0.76	0.82	0.79	1302	144	102	452
16	MLPClassifier	Word Embeddings (BERT)	0.87	0.75	0.82	0.78	1293	153	98	456
17	RNN	Word Embeddings (BERT)	0.86	0.72	0.84	0.77	1578	229	112	581
18	BERT Stacking	Text (Word Embeddings (BERT))	0.86	0.83	0.63	0.72	1718	89	257	436

Figure 1: Results per classifier/word representation pair

In our research, a total of 18 different approaches were tested, including six different classifiers in combination with four different word representation methods. Here, Index 1 represents our baseline model, which is based on the K-Nearest-Neighbour algorithm and the Bag-of-Words method.

Overall, the following findings can be formulated:

- We observe that all the approaches examined perform better than our baseline model (Index 1).
- Of particular interest is the stacking approach (index 18), which performs exactly opposite to our baseline model.
    - Furthermore, it can be seen that the choice of word representation method has a different impact on the result depending on the classifier.

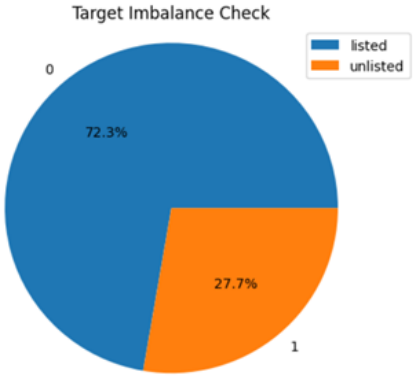
- K-Nearest-Neighbor performs significantly better with vector representations compared to the Bag-of-Words method and TF-IDF.
- In Logistic Regression, the Bag-of-Words method performs worse than the other methods, which otherwise behave similarly.
- Interestingly, RandomForest and MLPs show little sensitivity to the word representation methods applied.
- The RNN and the stacking approach, both performed only moderately to poorly.

Based on the evaluation, we decided to use the final combination of RandomForest and Skip-Grams. For this combination, a GridSearch was used to determine the ideal hyperparameters, but this did not improve performance.

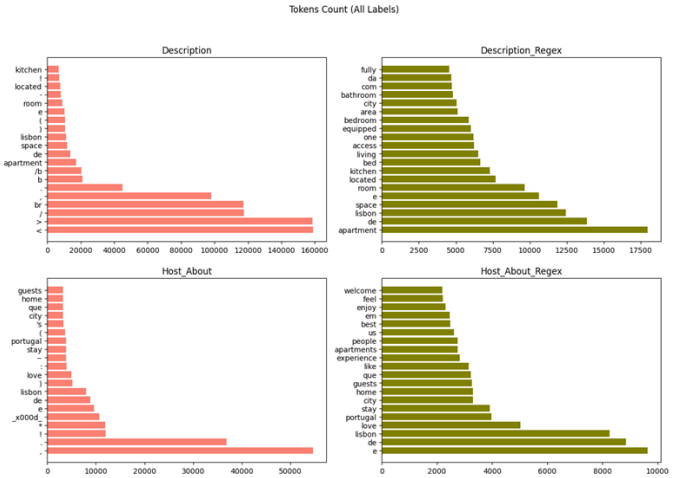
The final step was to merge the training and validation data to train the model with all the data. This model was then used to predict the test data. The percentage of Airbnbs classified as unlisted is 23.11%. This means that the class distribution is similar to the distribution in the training data, where 27.7% of Airbnbs are labelled as "unlisted".

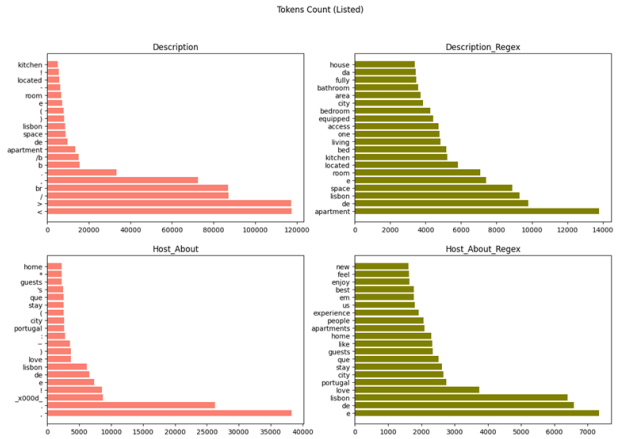
7. Annex

7.1. Annex 1 - Target Imbalance Check

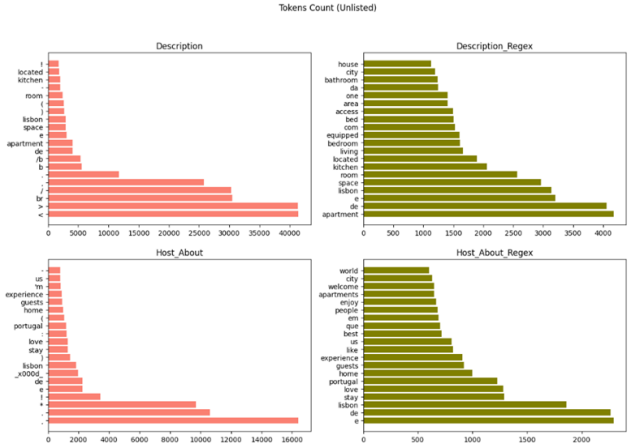


7.2. Annex 2 - Most Frequent Tokens – All properties (left all, right listed properties)

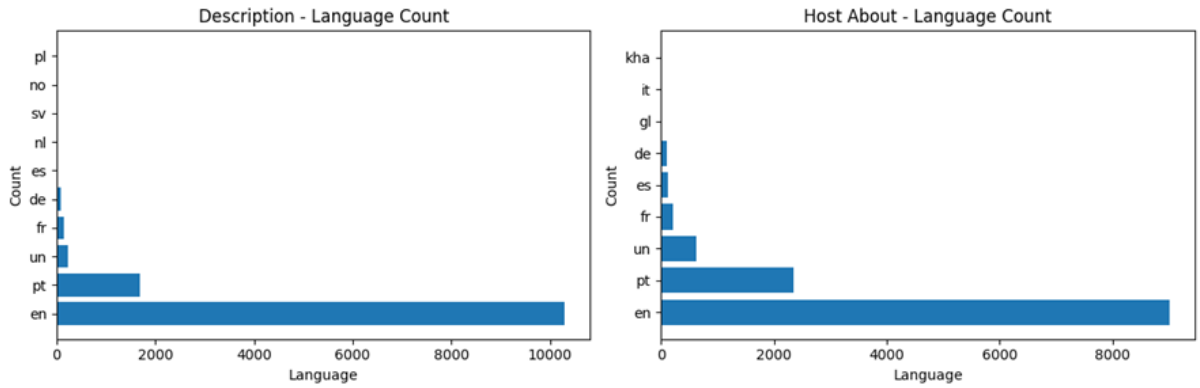




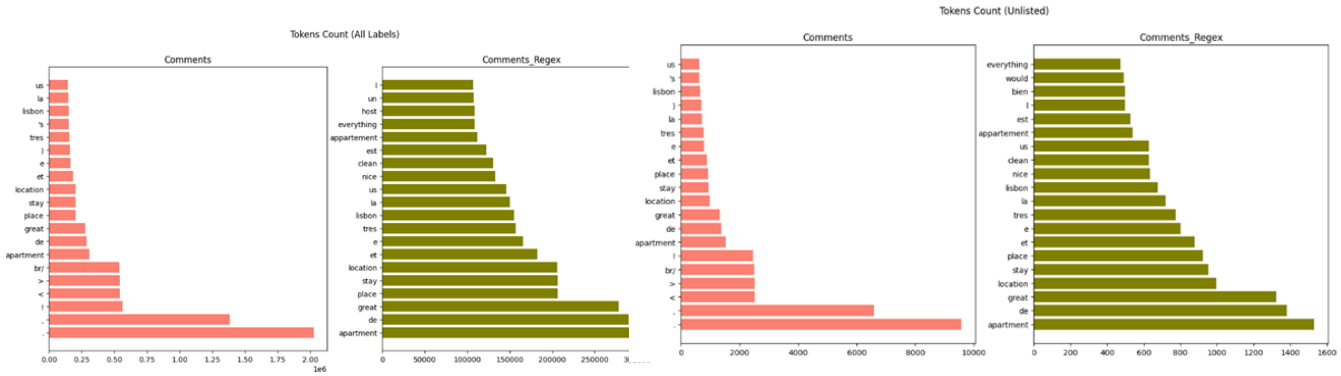
7.3. Annex 3 - Most Frequent Tokens – unlisted properties



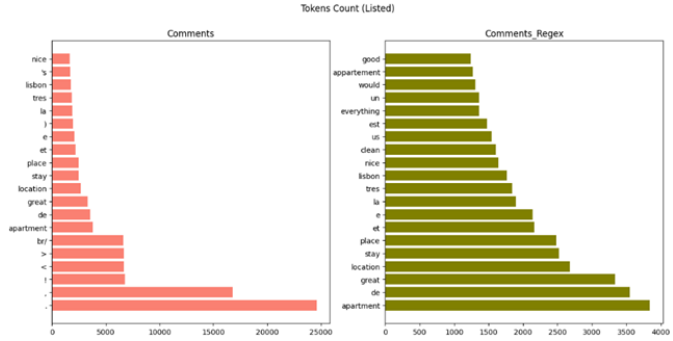
7.4. Annex 4 – Most common Languages in the Dataset



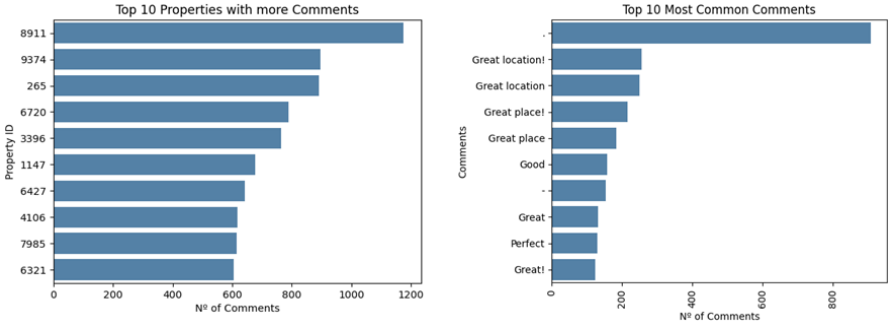
7.5. Annex 5 – Most Frequent Tokens - Train_Reviews (left all, right unlisted properties)



7.6. Annex 6 – Right: Most Frequent Tokens - Train_Reviews (listed properties)



7.7. Annex 7 - Top 10 Properties with more Comments & Top 10 most common Comments



8. References

Hutto, C.; Gilbert, Eric (2014): VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In *ICWSM* 8 (1), pp. 216–225. DOI: 10.1609/icwsm.v8i1.14550.

NLTK (2023): NLTK :: Natural Language Toolkit. Available online at https://www.nltk.org/index.html, updated on 1/2/2023, checked on 6/17/2023.

Sanh, Victor; Debut, Lysandre; Chaumond, Julien; Wolf, Thomas (2019): DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. Available online at https://arxiv.org/pdf/1910.01108.