Sentiment Prediction and Topic Modeling of Listings in London

Master of Applied Data Science

University of Michigan School of Information (UMSI)

SIADS 696: Milestone II

October 2022

Table of Contents

About the Authors

Nishant Singh is Assistant Vice President - Analytics with Applied Data Finance. He has over 13 years of experience in using analytics to solve business problems, primarily in the financial services and e-commerce domains. Nishant has his undergraduate degree in Instrumentation and Control Engineering and is currently pursuing the Master in Applied Data Science from UMSI. He is based out of New Delhi, India

Elias Kim is Data Analyst. He works in the finance sector specialising in building automated models for Anti-Money Laundering (AML) / Bank Secrecy Act. He is currently working with the government to improve the accuracy of AML automation models. He holds a Bachelor in Applied Mathematics and is currently pursuing the Master in Applied Data Science from UMSI. He is based out of New York, USA.

Michael Wynn is Manager, Manpower & Employment Research. He works with government agencies, providing analytical insights to assist manpower development policies. He was a Data Analyst at a market indices provider in infrastructure investments. He holds a Bachelor of Economics and is currently pursuing the Master in Applied Data Science from UMSI. He is based out of Singapore.

Statement of Work

Nishant Singh: Set up Cloud Service for Data Storage, GloVe Word Embedding, Supervised (Deep) Learning
Elias Kim: Language Detection, Data Cleaning & Preprocessing, Unsupervised Learning
Michael Wynn: Set up Github Repository and Maintain Workflows, Sentiment Tagging, Supervised (Machine) Learning
All members: Content Writing and Design for Project Proposal, Standup Videos and Final Report

Introduction

Airbnb is an online marketplace that provides a platform for homeowners (hosts) to rent out their properties to guests looking for accommodation. As at March 2022, there were over 6 million active listings on the platform. The type of property varies from a shared room, to a private room and to an entire apartment. The platform boasts of 4 million hosts who have welcomed over 1 billion guests in almost every country across the globe. Every time guests complete their stay and checkout from a property, they are encouraged to rate their experience. The star rating system is a quick way for guests to provide feedback on multiple parameters including their overall experience, allowing future prospective guests to get an idea on the property's quality. Guests are also encouraged to write a review. Comparing these two metrics, star ratings themselves do not fully reflect a guest's unique experience. On the other hand, text reviews contain more information such as polarity that include both negative and positive sentiments, that fully captures how a guest felt about that particular listing. To this end, we plan to capture these underlying emotions in text reviews data and use it to compute an entirely new review score for each listing. These review scores will assess all sentiments expressed within each review. We will then develop a sentiment prediction model that will be deployed to hosts who can use it to predict sentiment of future reviews. Additionally, we plan to identify key themes and topics in the reviews data. This would provide useful insights to hosts on what exactly the guests care about and help them identify key areas of improvement to make their properties more attractive to prospective guests. We focused on Airbnb listings in London, which contains 66,000 properties spanning approximately 1.2 million reviews. As a metropolitan city, London constantly attracts millions of tourists from all over the world with different demographic characteristics. Therefore, the population of reviews for London listings adequately captures a complete set of preferences and priorities when it comes to accommodation.

Part A: Supervised Learning

Motivation

Star ratings and reviews on the Airbnb platform aim to provide clarity about a listing's qualities. Unfortunately, the nearly ubiquitous star rating scale can foster assumptions to prospective guests and readers, and lacks clearly defined measurements. Different people rank star ratings differently. This ranking system therefore inherently lacks clarity because it is based on images (i.e. number of stars) that seem to represent the full breadth of a guest's emotional experience. Representations can do better with interpretations. Therefore, detailed text reviews provide a better picture for each listing. In an attempt to go beyond star ratings, we will use text reviews to compute a new compound score that accurately measures a guest's unique experience with a particular listing. To achieve this, we first need to classify each Airbnb guest review into positive, neutral or negative sentiments. From here, we will build a sentiment prediction model that can be deployed for all hosts to fully evaluate future reviews made at their listings.

Data Cleaning & Pre-Processing

The dataset was retrieved from a publicly accessible site (http://insideairbnb.com/get-the-data/). Since our goal is to analyze listings in London, we chose the London dataset which was compiled on the 10th of September, 2022. We specifically chose 'reviews.csv.gz' that contained a full and detailed collection of text reviews. The file was in CSV format, containing 1,216,212 rows and 6 columns: listing_id, id, date, reviewer_id, reviewer_name, and comments. For our analysis, we mostly used the 'comments' column since our task is majorly sentiment analysis.

Before processing the sentiment tagging, we observed that the reviews were containing words that had no meaning such as HTML tags, punctuation, NAN values and so forth.

The cleaning stages were composed of 5 steps:

1. Drop rows containing NAN values
2. Drop rows with empty string for 'comments' column
3. Remove HTML tags from string in 'comments' column
4. Remove special characters including emojis from strings
5. Drop rows containing non-English string in 'comments' column

From stage 1, 92 rows containing NAN values were dropped. From stage 2, 1477 rows were dropped because they were either empty or simply contained "." (dot). Next step was to identify the HTML tags and drop the rows that contained such tags. Initially, regular expression (regex) was used to identify and drop the responsible rows but it failed to detect some tags and the rows still contained HTML tags. After extensive research, *BeautifulSoup,* a HTML parser python package was used to successfully filter the HTML tags from the reviews. The following step was to identify special characters and emojis. The exclamation mark and question mark were excluded from the filter list because it was needed from sentiment tagging in later steps. Regex was used to filter out the special characters and emojis with flags=re.UNICODE.

The last step was to drop rows containing non-English strings from review. Upon inspecting rows, some rows contained non-English strings such as Chinese, Korean, Spanish, French, Arabic, and so forth. It was evident that successfully removing those non-English strings was a key to higher accuracy of a model for both supervised and unsupervised learning.

To accomplish such a task, *fastText*, an open-source python module from Facebook AI Research was used. The module is capable of recognizing more than 170 languages, so it was the right module for the task. A full pre-trained model was downloaded and loaded into the notebook for our usage.

```
# test model with example text
fasttxt_model.predict('이 집은 정말 좋아요! 맘에 쏙 드네요')[0][0][-2:]
```

```
'ko'
```

The model was tested using an example phrase in Korean and it successfully detected and classified the language. Using the module, each row was correctly identified and classified according to their language. Upon completing the classification task, the proportion of non-English rows turned out to be 10.28% so it was safe to drop the rows with non-English comments. After this step, a close examination on the dataset was carried out to further filter out possible mis-labeled rows . Quite a few reviews were written in two languages and yet, they were identified as English.

index	listing_id	id	date	reviewer_id	reviewer_name	comments	lang
1216207	14832630	184884203	2017-08-20	54407484	丽云	干净舒适适合家庭入住地段优越景点全部很多都可以步行到达节省了交通费房间设施齐备能满足我们的一切需求房东十分热情周到联系及时入住期间有个灯坏了跟房东联系后当天出游回来已经完美解决完全没有影响我们的行程 Clean and comfortable, suitable for family occupancy, lots of advantages, many attractions can be reached on foot, saving traffic costs. The room is well equipped. Can meet all our needs. Landlord very warm and thoughtful, timely contact. Check in, there is a lamp broken, and after contact with the landlord, the day of travel back, has been perfect solution. It didnt affect our schedule at all.	en

To identify those rows with two languages, CLD2 (Compact Language Detector 2), an open-source language detection library was used since the library supports mixed-language input. By running CLD2 to each row, it returned the top three languages found and their approximate percentages of the total text bytes for each comment. For example, the index 1216207 from the above CLD2 output yielded 34% zh and 66% en. Using that label, the rows containing more than one label were dropped using the below code snippet.

```
# drop rows containing multiple languages
reviews_df['Deteced_lang'] = reviews_df['comments'].map(lambda a: cld2.detect(a,bestEffort=True)[2][1][0])
```

Through the multiple stages of cleaning, we were able to acquire a clean dataset with plain English text.

After the cleaning process, the data pre-processing steps were performed which included :Punctuation removal, Stopwords removal, Lowercase, Lemmatization, andTokenization

For these steps, two separate columns were made for different usages in later modeling part.The 'comments' column was created using basic python string manipulation, which yielded str values.The 'cleaned_nltk_comments' was created using regex and NLTK, which yielded tokens. In result, we were able to acquire a pre-processed dataset, which can be used as an input for various models later in the modeling part.

	comments	cleaned_comments	cleaned_comments_nltk
0	Great location, and the host was very responsi...	great host responsive	[great, location, host, responsive, helpfulhugh]
1	Duccio is a lovely and friendly host. From arr...	duccio lovely friendly from arrival instructio...	[duccio, lovely, friendly, host, arrival, inst...
2	Duccio is a good communicator he was very help...	duccio good communicator helpful making stay g...	[duccio, good, communicator, helpful, making, ...
3	Not entirely compliant to the pics.Good locati...	not entirely compliant location basic shower w...	[entirely, compliant, picsgood, location, basi...
4	Great place and great host	great place great host	[great, place, great, host]
...	...	...	...

Sentiment Tagging

Valence Aware Dictionary for Sentiment Reasoning (VADER) model was used for text sentiment analysis. This rules-based tool is sensitive to polarity (i.e Positive, Negative) and intensity of emotion expressed in each text review. It uses a dictionary that maps lexical features to emotion intensities known as sentiment scores. It is on these sentiment scores that we will use as replacement to the default star rating system. VADER is imported from the NLTK package and we have applied it directly to the raw comments column in the dataframe (to capture sentiments and emotional intensity fully).

The result generated is a dictionary of 4 keys neg, neu, pos and compound: neg, neu, and pos refer to negative, neutral, and positive respectively. Compound corresponds to the sum of the valence score of each word in the lexicon and determines the sentiment degree. The value of -1 indicates most extreme negative sentiment and +1 indicates most extreme positive sentiment. This score is then used to determine the underlying sentiment of a text. The tagging rule is based on the following measurement:

```
def sentiment_tag(row):
  if row['VADER_compound_score'] >= 0.05:
    sentiment = "Positive"
  elif row['VADER_compound_score'] <= -0.05:
    sentiment = "Negative"
  else:
    sentiment = "Neutral"
  return sentiment
```

Text review is tagged as a positive sentiment so long as the compound score is less than or equal to +0.05, negative sentiment for compound score less than or equal to -0.05 and neutral sentiment tagged to the remaining reviews that fall within the two range of values.

The elapsed run time for the tagging process came up to be 2 hours 37mins.

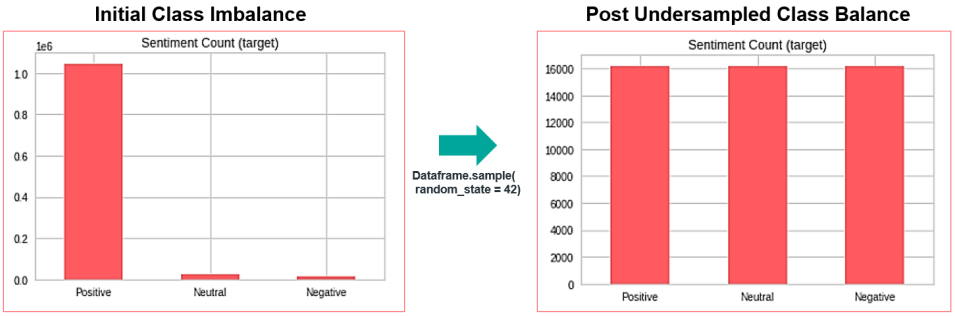
```
%%time
df['VADER_compound_score'] = [round(SentimentIntensityAnalyzer().polarity_scores(x)['compound'], 2) for x in df['comments']]

CPU times: user 2h 34min 28s, sys: 2min 53s, total: 2h 37min 22s
Wall time: 2h 37min 29s
```

After the tagging process is completed, the descriptive statistics of the tagged sentiment show a class imbalance in the data, with the majority of reviews being positive, accounting for 96% of all reviews. Additionally, they were overwhelmingly positive with mean and median compound scores of 0.86 and 0.92. We also note that negative compound mean and median values were not as skewed as the positive class. To deal with the severely skewed class distribution for the supervised classification task, we should not use accuracy as an evaluation metric. However, the classification models fitted on the full data took a long time and caused the team multiple time-outs (Google Colab has a session idle timeout of only 1hr 30mins). Hence, in efforts to increase computational efficiency for model training, we performed a technique (Undersampling) to balance the uneven distribution by keeping all of the data in the minority class and shrinking the size of the majority classes down to the minority class count.

Sentiment	Count	Mean	Median
Negative	16185	-0.52362	-0.51
Neutral	25786	0.00012	0
Positive	1045528	0.85646	0.92

Undersampling

As the presence of a class Imbalance poses a problem with our supervised classification task, we conducted random undersampling for both positive and neutral sentiments. As we have removed random samples in both the imbalance classes, the shortcoming of the method means we lose information and particularly in the positive sentiment class. However, the supervised learning results show that through our initial seed randomisation, we managed to capture and retain the variation needed in all three sentiments. As negative sentiment has the lowest proportion, we shrunk the other two sentiments down to get an equally distributed and balanced dataset for model training.



Methods & Evaluation (Machine Learning)

Feature Representation

After both input (cleaned_comments) and target (sentiment) have been created for machine learning, we need to convert the text representation of reviews into a numerical representation. Term frequency-inverse document frequency (TF-IDF) is used to quantify words and the word vector is extracted from the TF-IDF vectorizer. In addition to its simplicity, this feature representation is used because TF-IDF score does allow for an accurate representation of the overall polarity. Terms are given

weights based on their uniqueness and relevance, rather than just the frequency at which they appear. However, TF-IDF score is just a statistical measure and ignores word order in a sentence. Hence, we have also developed a Long Short-Term Memory (LSTM) model to account for this in the deep learning section using GloVe word embedding.

Using the undersampled data, we split it into three separate sets via random sampling - Training, Validation & Testing using the standard ratio of 80%, 10% and 10%. The TF-IDF vectorizer is then applied separately to each set to avoid data leakage. Additionally, a limit of 25 words was applied through the min_df parameter in the TF-IDF vectorizer to ensure that each token showed up at least that number of times.

Multiclass Classification Model

To begin with, we will treat the Logistic Regression model to act as our baseline model. Logistic regression as a baseline is advantageous as it trains quickly even on large datasets and provides robust results. We trained a total of 9 classifiers (including base model), fitted on the matrix of features extracted from the list of reviews (cleaned_comments) in the training set. The features extracted were performed by calling the fit_transform. Each fitted classifier is then tested for its performance by comparing several evaluation metrics computed based on the validation target with the labels predicted in the validation set. At this stage, all classifiers are given the default hyperparameters. The results of all 9 classifiers are the following:

Classifier	Accuracy (Balanced)	Precision (Macro)	Recall (Macro)	F1 Score (Macro)	Jaccard Score (Macro)	F-beta Score (Macro)	AUC ROC Score	Log Loss
LogisticRegression (Base)	0.917	0.917	0.917	0.917	0.848	0.917	0.979	0.275
SupportVectorMachine	0.918	0.918	0.919	0.918	0.850	0.918	0.979	0.265
MultinomialNB	0.806	0.805	0.837	0.801	0.669	0.799	0.966	0.465
DecisionTreeClassifier	0.845	0.845	0.845	0.845	0.735	0.845	0.872	5.192
RandomForestClassifier	0.897	0.897	0.898	0.897	0.815	0.897	0.971	0.356
GradientBoostingClassifier	0.877	0.877	0.877	0.876	0.781	0.877	0.966	0.409
AdaBoostClassifier	0.846	0.846	0.848	0.843	0.731	0.844	0.862	1.019
ExtraTreesClassifier	0.905	0.905	0.906	0.905	0.828	0.905	0.973	0.380
KNeighborsClassifier	0.406	0.409	0.672	0.312	0.198	0.340	0.506	16.702

Total elapsed run-time: 33:41 mins, 224.56s/it

All classifiers except KNeighborsClassifier performed well with >0.8 balanced accuracy, macro precision, macro recall, F1, F-beta and Area Under the Curve Receiver Operating Characteristic (AUC ROC)scores. The below-expectation evaluation results for KNeighborsClassifier seem to be due to underfitting. We performed several iterations for improving the classifier (on default parameters) through finding the optimal 'K' using the error rate and accuracy score. However, results continue to still show underperformance (Please refer to section 1.3 of Appendix for the result). When it comes to Jaccard score, all but the DecisionTreeClassifier, GradientBoostingClassifier and AdaBoostClassifer achieved >0.8. Jaccard score provides a good indication that these three classifiers are performing more poorly in certain sentiment classes.

The best performing model, by virtue of the most number of best performing evaluation metrics, is SupportVectorMachine (SVM). However, since SVM computes a distance function between each point in the training data, the storage of the distances takes a burden on memory and we noticed a long elapsed run-time while performing the model fitting process. Hence for efficiency, the team has moved forward to perform hyperparameter tuning on the second best performing classifier - LogisticRegression (which is our base model).

Hyperparameter Tuning

Selecting the optimal model architecture for the LogisticRegression was performed through RandomisedSearchCV, based on the scoring criteria on macro F1 score. The default model was fitted to the training set and the defined search space for the RandomisedSearchCV instance is the following:

```
# Define search space
space = {"solver": ['lbfgs', 'liblinear'],
         "penalty": ['none', 'l1', 'l2', 'elasticnet'],
         "C": [0.001, 0.01, 0.1, 1, 10],
         "multi_class": ['auto', 'ovr']}
```

The decision to optimize the model's performance based on macro F1 is due to the earlier observation that listing reviews consists of highly imbalanced sentiment classes. Through macro F1 score, a classification model can still reflect true model performance even when the classes are skewed. The best parameters after executing search results is the following:

```
{'solver': 'liblinear', 'penalty': 'l1', 'multi_class': 'auto', 'C': 1}
```

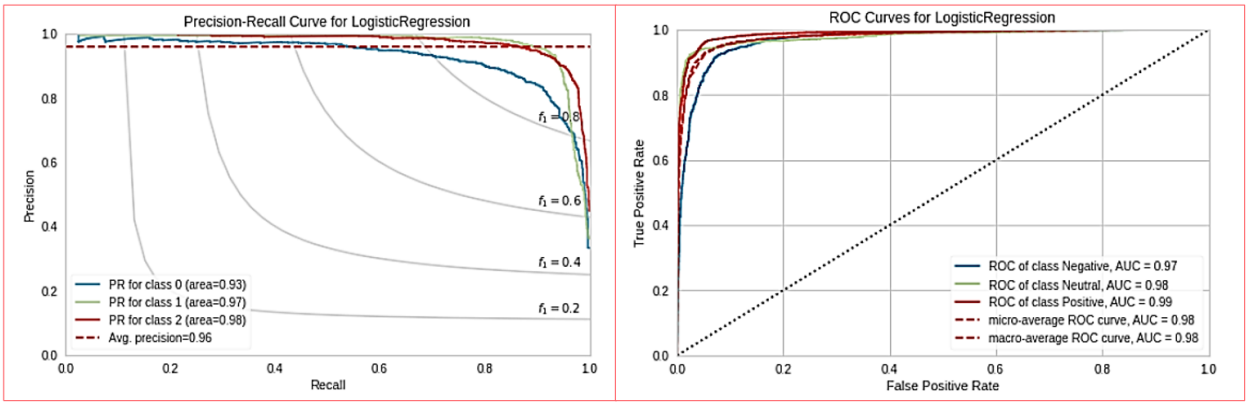
Evaluation

Both default and tuned LogisticRegression models were compared against each other on the test sample.

Classifier	LogisticRegression (Default)	LogisticRegression (Tuned)	Difference
Accuracy (Balanced)	0.913	0.915	0.248%
Precision (Macro)	0.912	0.914	0.223%
Recall (Macro)	0.912	0.914	0.224%
F1 Score (Macro)	0.912	0.914	0.220%
Jaccard Score (Macro)	0.839	0.843	0.421%
F-beta Score (Macro)	0.912	0.914	0.221%
AUC ROC Score	0.979	0.978	-0.131%
Log Loss	0.277	0.286	-3.195%

Note: The higher the log-loss value, the more the predicted probability diverges from the actual value. Log-loss is indicative of how close the prediction probability is to the corresponding true value.
Source: Dembla, G. (2020). Intuition behind Log-loss score. Towards Data Science

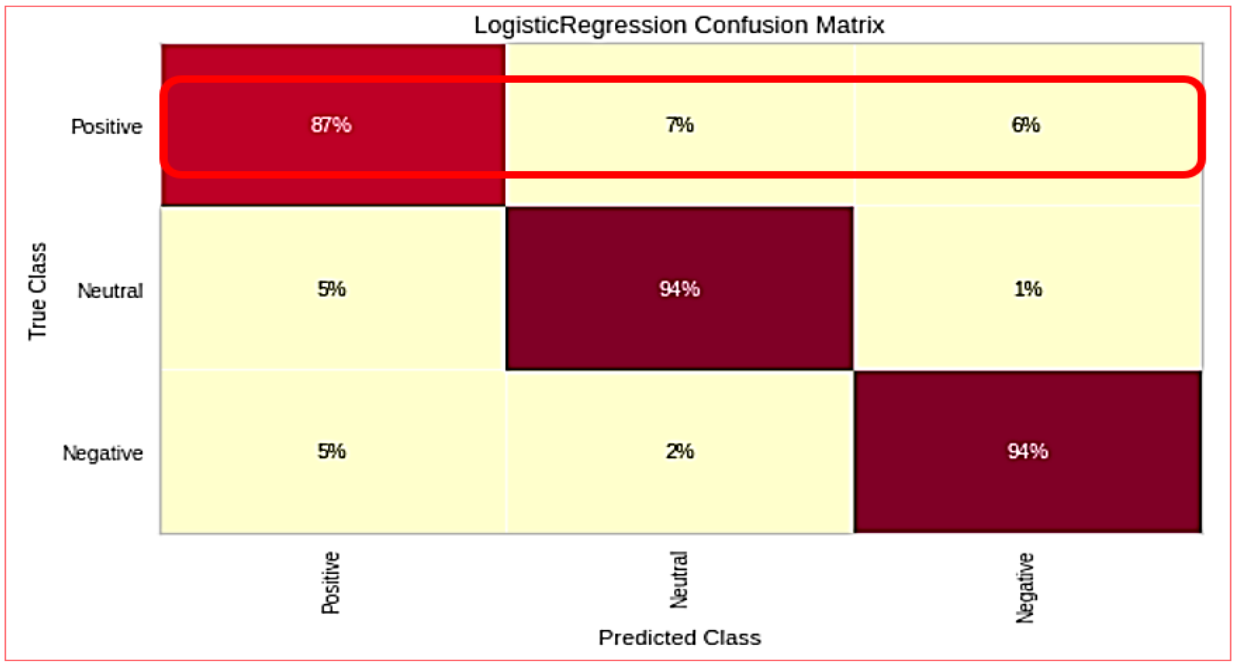
The results of the evaluation metrics show that the tuned model performed slightly better in all performance metrics except AUC ROC and log loss value, with most improvement in macro Jaccard score. It is worth noting that the only difference between the two models is the 'solver' parameter. Both models exhibit the same Precision-Recall and ROC curves.



Note: Classes in Precision-Recall Curve refer to the following: 0 (Positive), 1 (Neutral) and 2 (Negative)

Both models perform well on unseen data with >0.9 in most performance measurements. The tuned RegressionModel performs marginally better and we pick this model to accomplish the sentiment prediction task for deployment.

Failure Analysis



	precision	recall	f1-score	support
Positive	0.89	0.87	0.88	1561
Neutral	0.92	0.94	0.93	1685
Negative	0.93	0.94	0.93	1610
accuracy			0.91	4856
macro avg	0.91	0.91	0.91	4856
weighted avg	0.91	0.91	0.91	4856

As we observe from the confusion matrix above, predictions were poorer for positive sentiment as compared to neutral and negative sentiments. As also seen from the earlier Precision-Recall Curve, the precision for positive sentiment was lowest as compared to the other two classes (reinforced by the classification report here). The model has predicted 7% to neutral

class and 6% to negative class. Throughout the model building, we have prioritized efficiency by performing undersampling on an imbalance dataset. Bringing down the classes to equal sizes has allowed us to attain a generally high precision and recall scores for the tuned RegressionModel that are >0.9. Hence, the poorer performance particularly for the positive class cannot be attributable to a class imbalance. We believe that the reason could be due to the inherent feature extraction method for model training through TF-IDF. We could also test the classification result by setting the ngram_range to (1, 2) which can take into account words like “not good” or “not bad”. We have also developed a deep learning LSTM model using GloVe Word Embedding, that takes into account that certain words generally co-occur more often with one word than another, to see if the prediction results improve.

Methods & Evaluation (Deep Learning)

Introduction

Traditional Machine learning models that use features created by bag-of-words techniques like TF-IDF vectorization, have some limitations.

- Traditional ML models for Sentiment classification ignore the order of the words in the document. Consider a comment: *Nothing is good about this apartment. The apartment is not good, the furniture is not good and the location is not good as well.* A traditional ML model (using features created by TF-IDF vectorizer) sees multiple occurrences of the word “good” but it does not capture the fact that the word “good” is always preceded by the word “not”. A sequence model would help capture the order of occurrence of the words. Long Short-Term Memory (LSTM) model is a type of sequence model which helps capture long range dependencies in a sequence of words. In particular, we plan to use a Bi-directional LSTM model because a bidirectional LSTM uses information from words both earlier and later in the sequence to make predictions.

- Representing a document using TF-IDF vectorizer does not capture the semantics of a word. Presence of words like *excellent*, *marvelous and wonderful* in a text document might convey similar sentiments however the TF-IDF feature representation does not capture this. We can use a word embedding representation of a word to capture this aspect.

Feature Representation

For building our Bi-directional LSTM model, we used the cleaned train-validation-test split created above (from the supervised machine learning models). The words in the training set constitute our vocabulary.

The next step is to represent each word in the vocabulary with a word embedding vector. However, instead of creating our own word embedding representation, we chose to use the pretrained GloVe (Global vectors for word representation) word embeddings created by Stanford University. This is trained on a rich corpus of 6 billion tokens and has a vocabulary size of 400,000 words. There are 4 versions of this model with a 50-D, 100-D, 200-D and 300-D vector representation for a word. For our purpose, we choose the 300-D representation.

Next, we tokenize each row of the data and represent each of the tokens using its word embedding vector (from the pre-trained GloVe model). For words in our vocabulary which we do not have a word embedding vector in the pre-trained GloVe model, we make imputations. We represent them using the average embedding vector (by taking the average of all 400,000 words embeddings).

Next, we make the length identical for each sequence (i.e., each row of our data). We compute the 90^{th} percentile of the sequence lengths observed in the training data. The 90^{th} percentile for sequence length is 118. Sequences longer than this are truncated while sequences smaller than this are padded with zero vectors.

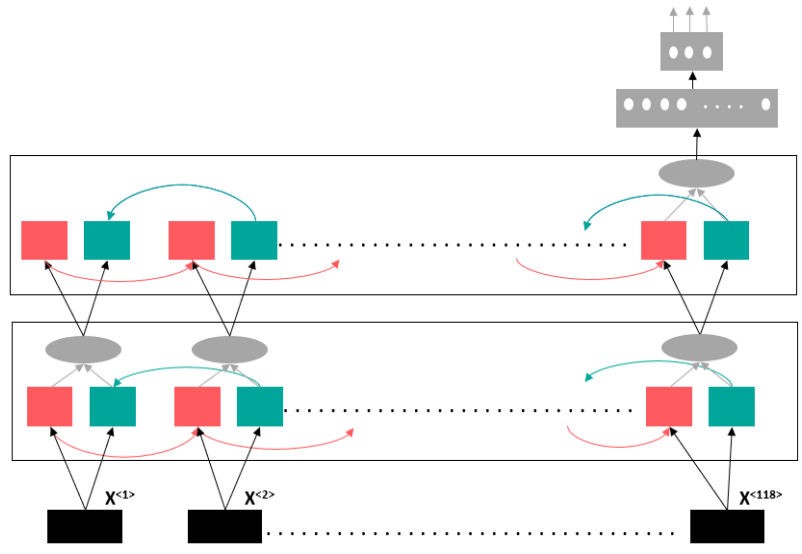
The feature creation is done. We are done representing each word in each sequence with an embedding vector. Next, we move to defining the model architecture and training the model.

Model Architecture and Training:

The word embedding representation created above is our input layer. The output of our model needs to be a Softmax layer with 3 outputs, one each to predict the propensity of Negative, Neutral and Positive Feedback respectively. Between these two layers we need the hidden layers. As a part of our model training exercise and hyper parameter tuning, we tested multiple architectures and based on the validation sample performance (Cross entropy loss and Accuracy), decided on the following architecture:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 118, 300)          7078800

 bidirectional_8 (Bidirectio  (None, 118, 200)         320800
 nal)

 bidirectional_9 (Bidirectio  (None, 200)              240800
 nal)

 dense_8 (Dense)             (None, 128)               25728

 dense_9 (Dense)             (None, 3)                 387

=================================================================
Total params: 7,666,515
Trainable params: 587,715
Non-trainable params: 7,078,800
_________________________________________________________________
```

The model architecture diagram below explains this architecture:



Final Output Layer: A Softmax layer with 3 outputs

Hidden Layer 3: A 128 output dense layer

Hidden Layer 2: Another 100 cell Bi-directional LSTM layer. Both the forward path and backward paths are computed independently and their output are concatenated. The parameter return_sequences is set to False to return only the last output

Hidden Layer 1: 100 cell Bi-directional LSTM layer. Both the forward path and backward paths are computed independently and their output are concatenated. The parameter return_sequences is set to True to return the full sequence

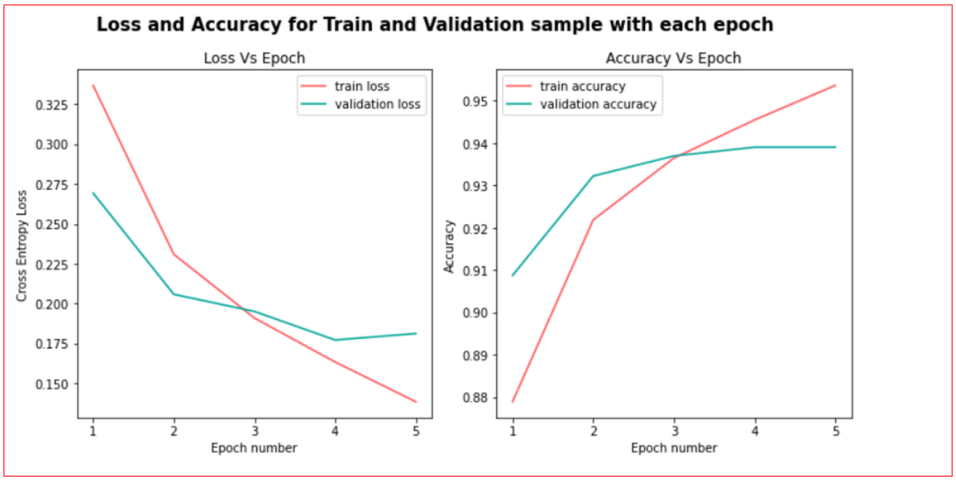
Input Layer: Each word is an embedding vector of dimension 300 and each sentence is truncated to length of 118 words

Model architecture diagram

Evaluation

The loss function for our model is Categorical crossentropy. The model tries to train the parameters to minimize this loss function. We tested different optimization algorithms to minimize the loss function and to update the model parameter values (Weights and biases) and finally decided to use the Adam optimizer.

We trained the model on 5 epochs and compared the training and validation losses after each epoch. Below are the Categorical crossentropy losses and accuracy for Training and Validation samples after each epoch of training. For the final model, after 5 epochs of training, the training accuracy is 95.36% and the validation accuracy is 93.90%. Training and validation loss after each epoch is shown below:



After multiple rounds of hyperparameter tuning, we tested the final model on the test sample. Below is the performance on the test data

Performance – Test Data

Accuracy	94.07%
Precision (Macro average)	0.9418
Recall (Macro average)	0.9415
F1 Score	0.9417

Confusion Matrix – Test Data

Actual Labels	Predicted Labels: Negative	Neutral	Positive
Negative	1,526	25	10
Neutral	97	1,534	54
Positive	79	23	1508

Failure Analysis

The BLSTM model performed pretty well with a test set accuracy of 94.07%. Shown above is the confusion matrix on the test data:

Positive comments – Wrongly tagged as Negative

Good spot to reside in. Shoutout to all the homeowners on airbnb building their equity in real estate. Stay on your grind. I mean everyone can be a homeowner. Invest in something else. You can do many things.

Darshan went out of his way to make our stay comfortable. We were having a very bad day, but Darshans consideration made things a little better.

Negative comments – Wrongly predicted as positive

location is great kitchen needs an upgrade in cutlery and cook ware mattresses could be upgraded for sure very affordable accommodation for London but Will look for another place next time especially as we travel as a family.

Pro: Euston/Kings Cross is a good location, lots of public transport. Many small restaurants nearby mostly Indian food. The room was OK. The room had its own lock with code pad, so safe and private. Also code pad on main entrance lets us come and leave anytime. There were two bathrooms w toilets, one toilet and one kitchen shared with other guests.Cons: when researching options for the trip we saw comparable prices for staying at hotels with breakfast included. One of the reasons we chose this place to stay was the advertised breakfast - which turned out to be only muesli and milk available for self-service.The room was described as quadruple. Its not that big IMO.So, OF place to stay but not great value for money compared to our expectations and the competition.

Though at an overall level the model performed well, we still had some wrong predictions (highlighted in green above) and we analyzed the comments in some of these wrong predictions to see what we could have improved further. Below are some examples of wrong comments.

One of the issues was that despite the fact that the pre-trained GloVe vector is trained on a rich text of 6 billion tokens, we still had words in our data which were 'out-of-vocabulary words'. Training a word2Vec/GloVe model on hospitality industry data (AirBnb/Booking.com/Tripadvisor etc.) would have given a more relevant word embedding model.

Discussion

Key Learnings

- Through experimenting with multiple models, we learnt that some models perform optimally based on different pre-set parameters and feature extraction methods used.
- Looking at multiple performance evaluation metrics has helped us identify the strengths and weaknesses of classifiers, allowing us to narrow down on picking the most efficient and best performing classifier for the task at hand.
- Deep learning models like LSTM took a long time to train. One possible solution to speed up the training process is to use local RAM instead of the cloud service. Another solution is we could have also experimented with different batch sizes and dropout rates.

Scope for Further Improvement

- The sentiment tagging was done using the VADER model. If time permits, we can explore manual tagging the entire data (or at least a sizable portion of it). This method will allow us to address issues arising due to incorrect classification by a rules-based black box model.
- Given more time, we could have also experimented with oversampling of the negative and neutral reviews to see how the results for both machine and deep learning would have come up.
- Using the sentiment predicted by our model, we can compute a positive feedback score (percentage of feedback which are positive) and compare its distribution to the existing star reviews distribution (which is highly skewed!!). We would need additional time to score every review in our data and compute the positive feedback percentage.
- For the deep learning model, we could have trained our own word embedding vector instead of relying on a pre-trained one. This would have helped mitigate the issue of out-of-vocabulary words.
- We also could have been more efficient in organizing the machine learning scripts by employing pipelines when running the models. This will remove work redundancy and simply the workflow.

Potential Ethical Consideration

- Given that we have removed reviews for languages other than English, it could subject non-english speaking hosts, who regularly have positive reviews from non-english speaking guests in London, to the concept of cumulative disadvantage. The non-english reviews cannot be used for sentiment prediction in our current model. To address this, we can therefore introduce the idea of using machine translation models as the first step in the process. This will help translate all reviews to English.

Part B: Unsupervised Learning

Motivation

Oftentimes, popular listings get fully booked months ahead, even a year in advance in some locations. On the other hand, some listings barely get booked by guests. Many hosts are lost on how to advertise their listings to guests even if they have a great place to share. Through a series of unsupervised learning methods, our team hopes to uncover underlying keywords that reviewers (or guests) look for when they use Airbnb. By providing keywords that guests look for, hosts would be able to gain some level of insight into which area they can improve upon. This extends beyond their listings and can also apply to their actual rental units. When keywords are provided, hosts could market their listings more advantageously with the provided keywords in order to attract more potential prospective guests. Further, hosts can focus on certain areas or attributes to make their rental unit more desirable, as opposed to the alternative of room/unit renovation. This analysis will be most beneficial to hosts who are new to the platform and intend to improve their ratings.

Methods & Evaluation

For this part of the analysis, topic modeling was performed using several different methods. Topic modeling is a process to automatically identify topics present in a text object and to derive hidden patterns exhibited by a text corpus. It is an unsupervised approach used for finding and observing the bag of words (called “topics”) in large clusters of texts. Topics can be defined as a repeating pattern of co-occurring terms in a corpus. Thus throughout this analysis, an attempt to identify topics for each sentiment is made.

Latent Dirichlet Allocation

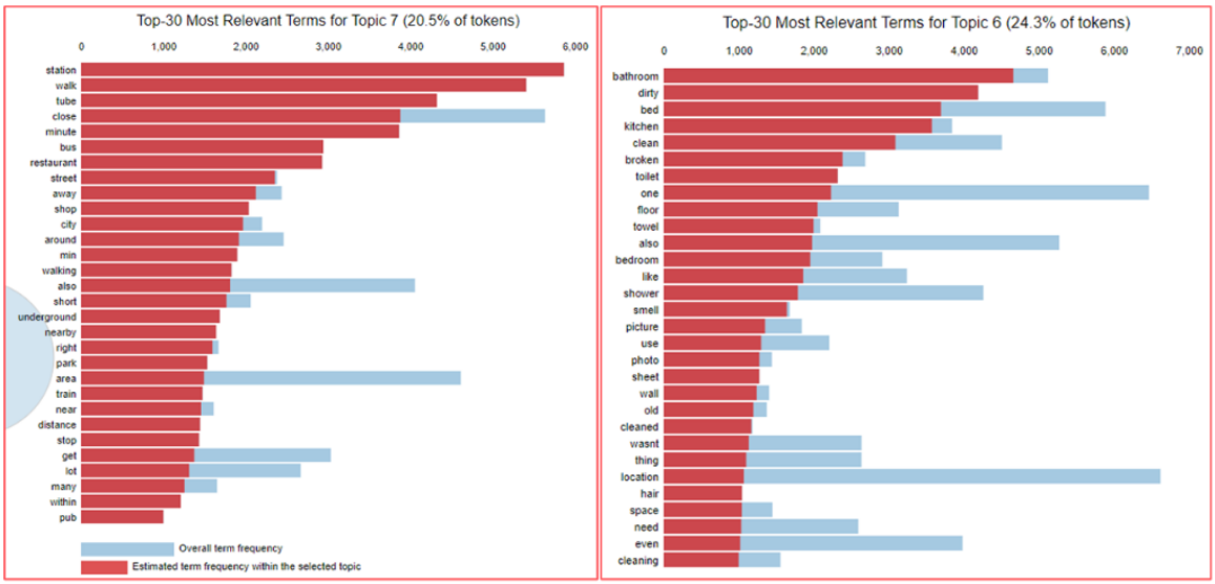
Latent Dirichlet Allocation (LDA) is an algorithm used to discover the topics that are present in a collection of text, corpus. Initially, LDA was performed using the sklearn package. To run LDA, first, the text was converted into a matrix representation using the Tfidf vectorizer. Various parameters were used to improve the model such as strip_accents, ngram_range, min_df, use_id, smooth_idf, and max_features. The ngram_range = (1,2) was used to cover the bigram extraction. Also, min_df = 2 was used to ignore terms with a frequency less than 2. The parameter, use_idf = True was used to enable inverse-document-frequency reweighting. Similarly, the parameter, smooth_idf = True was used to add one to document frequencies as if an extra document was seen containing every time in the collection exactly once, which prevents zero divisions.

After the text was converted into a matrix, it was passed to the LDA model using the fit method. The parameters were n_components, random_state, n_jobs, and learning_method. The parameter, n_components was used to identify the number of topics the model needs to retrieve. Here, one of the challenges of this method was setting the right number of topics. Since there was no fast and light way to implement an optimal number of topics, initially it was set to 10. Later in the steps, this value was fine-tuned accordingly using the *gensim* package. This step returned topics and their distribution.Finally, the topics were sorted by distribution and printed using *.compenents_* and *get_feature_names()* method.

The retrieved topics were still unclear and hard to label (Appendix 1.5). Further, the coherence score was 0.38 which was below average for a topic modeling task. Thus, we decided to use another method using the gensim package. After an unsuccessful analysis, the dataset was divided into three sub-datasets according to the sentiment: positive, negative, and neutral. Then LDA using *gensim* was carried out on each sub-dataset in hopes of better performance.

For the gensim model, firstly, a dictionary was created using the provided function *corpora.Dictionary.* Then, the text was converted to a document-term matrix using *doc2bow* function. The matrix representation then passed into LdaModel using parameters such as corpus, id2word ,chunksize, alpha, eta, iterations, num_topics, passes, and eval_every. To fine-tune the parameters, different values were tested for both 'alpha' and 'eta' parameters but none of them performed better x than when they were set to 'auto' in terms of coherence score. Thus, the model had both parameters set as 'auto'. Also, the num_topic parameter was the most concerning parameter since it directly affected coherence score. It was set to 8 according to the evaluation metric, which will be discussed in detail in the 'Evaluation' section. Finally, the topics and term-frequencies were printed but it was hard to interpret. Thus, visualization using pyLDAvis was used for a better interpretation. Topic modeling using *gensim* on positive sentiment sub-dataset showed keywords such as location, host, clean, station, walk, home, and tube as the top salient terms. (Appendix 1.6). While we tried to label each topic with a word that represents each cluster the most, one topic stood out.

Topic 7, which has 20.5% of tokens, consisted of keywords such as station, walk, tube, close, minute, bus, restaurant, and street.



This topic could be labeled as "Public transportation accessibility". From this topic modeling, we could gain an insight that the customers who use Airbnb in London are looking for listings within walking distance(possibly a minute away) of different public transportation such as tube and bus stations, followed by various shops, restaurants, and parks.

Topic modeling using *gensim* on negative sentiment sub-dataset showed keywords such as location, host, bed, dirty, door, night, clean, and bathroom as the top salient terms. Topic 6 which was 24.3% of tokens, represents what customers hated most during their stay. This topic could be labeled as " The most hated

part of the rental unit". From this topic, we could see that customers' number one complaint is the bathroom being dirty, followed by bed, kitchen, broken, and toilet.

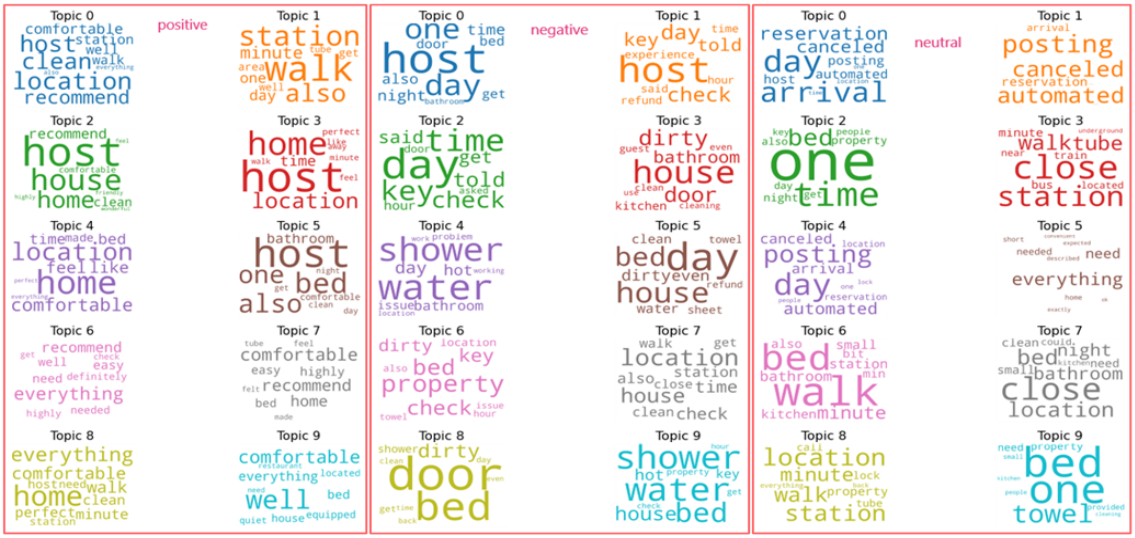
Topic modeling using *gensim* on neutral sentiment sub-dataset showed keywords such as location, close, photo, and quiet and described as the top salient terms. From the neutral sub-dataset topic modeling, there were not enough insights that we could get because many keywords were already covered by either positive or negative sentiment. One thing to note is that the keyword 'location' was still the most salient term for neutral sentiment as well. From this analysis, we could guess that location is the key to being a successful host.

Latent Semantic Indexing

Latent Semantic Indexing (LSI) analyzes the relationship between a set of documents and the terms these documents contain. It constructs a matrix that contains word counts per document from a text. The steps taken were very similar to the LDA model. A function named "prepare_corpus" was created to make a dictionary and document term matrix according to the sentiment. Therefore, using the created function, different dictionaries and document term matrixes were created accordingly.

Using dictionaries and document term matrixes, the LSI model was built using the parameters such as doc_term_matrix for document term matrix input, num_topics for specifying the number of topics, and id2word for created dictionary input. After which, each frequency and the topic was printed out. However, it was challenging to interpret the result on its own(Appendix 1.4). Thus, the WordCloud package was used to interpret the result in a more intuitive way.

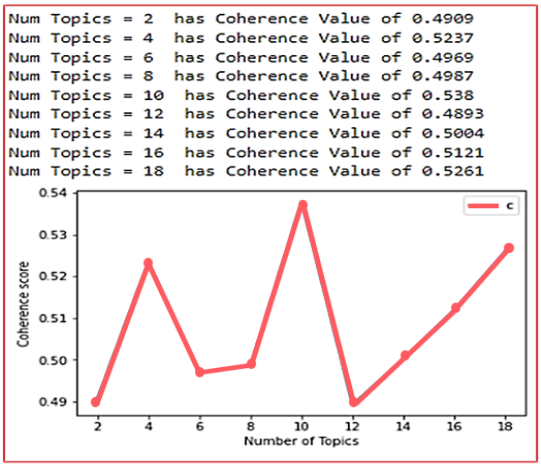
Topic modeling using LSI on positive sentiment sub-dataset showed distinct keywords such as host, location, home, station, walk, clean, and bed. One could notice that the keywords are quite similar to the ones extracted from LDA. Although, it is challenging to label each topic with a word that could represent each topic the most, it is intuitive to see the keywords in each sentiment



Evaluation

For the evaluation of LDA and LSA models, both perplexity and coherence scores can be used. Perplexity captures how surprised a model is of new data it has not seen before and is measured as the normalized

log-likelihood of a held-out test set. However, recent studies have shown that predictive likelihood (or equivalently, perplexity) and human judgment are often not correlated, and even sometimes slightly anti-correlated. Optimizing for perplexity may not only yield a human-interpretable topic but also our task did not involve any new data introduced to the model. Hence, it made topic coherence a more viable evaluation metric for our analysis.



Topic coherence measures score a single topic by measuring the degree of semantic similarity between high-scoring words in the topic. These measurements help distinguish between topics that are semantically interpretable topics. For this part of the analysis, the topic coherence score was used to evaluate the model and fine-tune the parameter. The coherence score largely depends on the number of topics retrieved. Since there is no automatic process to determine the optimal number of topics, different values had to be tested. Instead of plugging different values for num_topics back and forth, 'for loop' was used to calculate the number of topics and coherence score using that value. The values were plotted using plt. By inspecting the plot, we learned that the optimum number of topics was 10. Thus, using this value, we fine-tuned our LDA and LSA model retrospectively. Using this evaluation metric, we were able to achieve a coherence score of 0.538 from a score of 0.38.

Discussion

Key Learnings

- After a series of topic modeling methods, we were able to identify separate keywords for all guests who had a positive, negative or neutral experience. Guests who had a positive experience with the listing predominantly praised the location, host, cleanliness, proximity to public transportation and to various attractions such as shops, restaurants, pubs, and parks. Also, a lot of keywords were found to be related to abstract words such as 'feel', 'home',' family', and 'welcome'.
- On the other hand, customers who had a negative experience with the listing mostly hated similar features such as the location, host, cleanliness, and the common area like the bathroom, kitchen, toilet, and floor. Interestingly, 44% of the negative experiences were tied to the bathroom and cleanliness. Additionally, there were quite a large portion of keywords indicating that the rooms were not similar to what was depicted in the pictures.
- Lastly, customers who had a neutral experience with the listing also had similar things to say about its location, proximity to transportation, bed, communication, and price.
- As result, hosts who are looking into renting out their place or are in need of more attraction and positive reviews should primarily focus on the location of the place (which preferably includes a minute walk to public transportation), communication with the customer, welcoming vibe, cleanness of the place, especially the bathroom and price point.

- The challenging part of the analysis was balancing between the right number of topics, coherence score, and model runtime. The team ran the evaluation metric using a coherence score of up to 40 number of topics and the score kept increasing up to 0.57 at num_topic = 22. However, it took very long to process the model and print out the visualization. Although the model performed well, we had to find the right number of topics that would not reduce the score by much. After a few iterations, we were able to have a model that has a good coherence score of 0.538, moderate runtime of 6 mins and a decent number of topics (num_topics = 8) to interpret.

Scope for Further Improvement

- If more time and resources were allowed, this topic modeling could be improved by implementing a frequency filter that filters out terms with extreme frequency (either too low or high), part of speech tag filter which tags each token and filters out non-essential tags such as IN, CD, or MD. Both ideas could potentially improve the current model significantly.

Potential Ethical Consideration

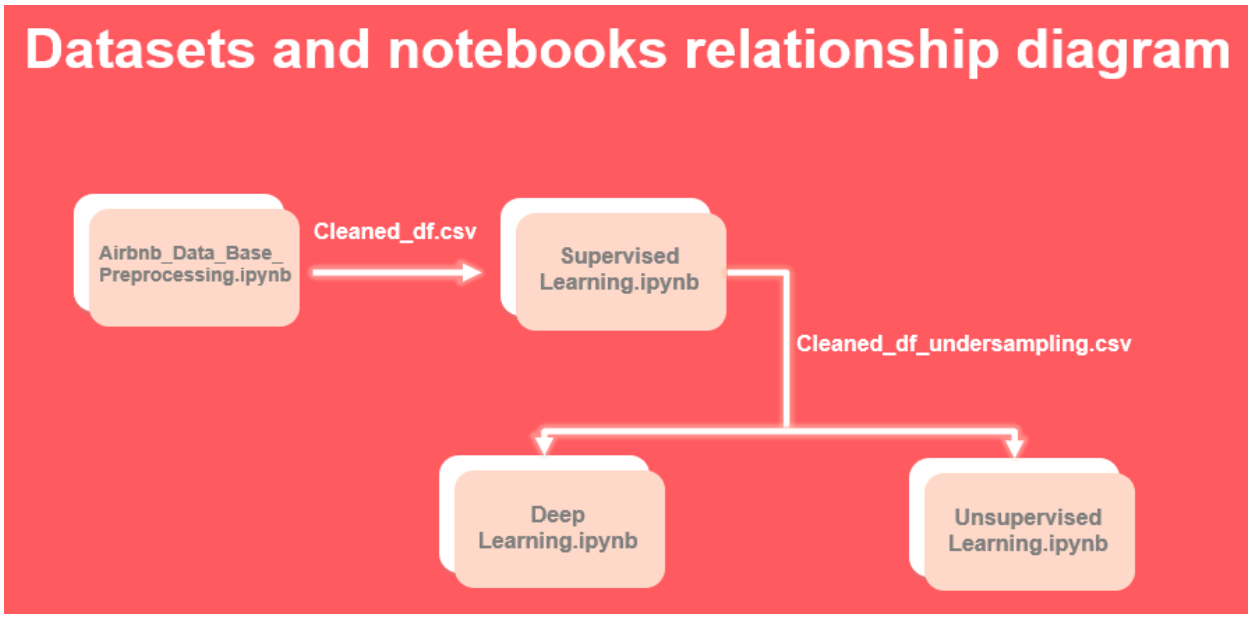
- Regarding guest preferences, some hosts offer below-average places with exceptionally cheaper prices compared to their neighbors, yet some customers still complain about their stay. We all thought that it is unfair to include those comments for our sentiment analysis and used them as a part of the negative sub-dataset. The proportion is unclear at this stage but given more time and resources, this ethical problem could be addressed too by analyzing another publicly available London Airbnb dataset that contains the ‘price’ column from ‘listings.csv’ whilst filtering out the ones with exceptionally low prices compared to neighbor listings.

Appendix

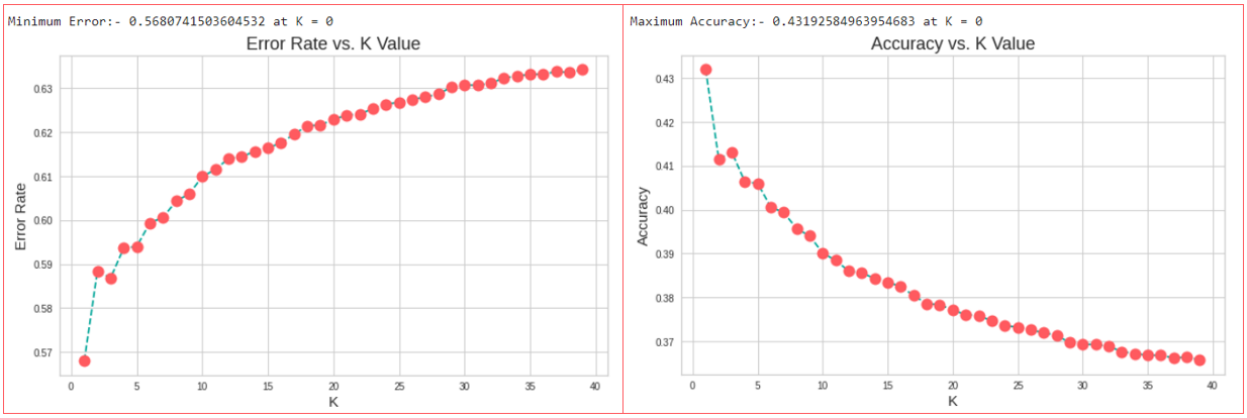
1.1 Github Repository

URL: https://github.com/Michwynn/London-Airbnb-Analysis---2

1.2 Relationship Diagram



1.3 Optimal "K" through Error Rate & Accuracy Score for KNN



1.4 Output of LSA

Printing Topics for positive reviews

Topic 0:, '0.292*"location" + 0.265*"host" + 0.256*"clean" + 0.177*"recommend" + 0.175*"comfortable" + 0.172*"station" + 0.160*"well" + 0.156*"walk" + 0.153*"everything" + 0.145*"also"'

Topic 1: '-0.563*"location" + -0.458*"host" + 0.200*"walk" + 0.198*"station" + -0.166*"clean" + 0.150*"also" + -0.147*"recommend" + 0.145*"minute" + -0.121*"perfect" + 0.104*"day"'

Topic 2: '-0.636*"location" + 0.476*"host" + 0.215*"house" + 0.198*"home" + -0.173*"walk" + -0.167*"station" + 0.149*"recommend" + -0.119*"tube" + 0.116*"clean" + -0.110*"minute"'

Topic 3: '-0.814*"clean" + 0.353*"host" + 0.186*"home" + -0.170*"comfortable" + 0.132*"location" + -0.110*"well" + 0.087*"time" + 0.077*"perfect" + -0.071*"everything" + -0.071*"bed"'

Topic 4: '-0.385*"host" + -0.359*"station" + -0.336*"walk" + 0.325*"home" + 0.235*"location" + -0.226*"minute" + -0.196*"tube" + 0.161*"comfortable" + -0.145*"close" + 0.132*"feel"'

Topic 5: '-0.489*"recommend" + 0.367*"host" + -0.260*"home" + -0.254*"house" + -0.235*"highly" + -0.230*"perfect" + -0.230*"everything" + 0.156*"bed" + 0.147*"also" + 0.137*"one"'

Topic 6: '-0.630*"house" + 0.409*"everything" + 0.255*"recommend" + -0.196*"comfortable" + 0.187*"easy" + 0.175*"need" + -0.163*"location" + 0.163*"well" + -0.155*"home" + -0.132*"station"'

Topic 7: '-0.516*"everything" + -0.415*"house" + 0.361*"comfortable" + 0.310*"recommend" + 0.271*"home" + -0.234*"need" + -0.169*"perfect" + 0.163*"highly" + 0.157*"easy" + 0.145*"bed"'

Topic 8: '0.418*"home" + -0.399*"recommend" + -0.377*"house" + 0.299*"everything" + 0.208*"comfortable" + -0.198*"well" + 0.198*"walk" + -0.167*"highly" + 0.146*"minute" + 0.141*"perfect"'

Topic 9: '0.591*"well" + 0.432*"comfortable" + -0.314*"clean" + -0.238*"time" + 0.187*"everything" + 0.164*"bed" + -0.132*"also" + 0.127*"house" + 0.111*"equipped" + -0.111*"get"'

1.5 Output of LDA using sklearn

```
Topic #1:
nice expected nice place come really back place come back stay great stay

Topic #2:
convenient location everything stay perfect convenient location home exactly described recommend

Topic #3:
canceled automated posting posting automated host canceled mold min walk staircase term alright

Topic #4:
good place website ive convenient place hidden hyde hyde park tin website hidden kensington

Topic #5:
dirty conveniently located conveniently reception would stay staff handy landlord confortable apt

Topic #6:
place great place stay smaller spot great provide dark thanks customer

Topic #7:
stay great london place location us really central definitely close

Topic #8:
not reservation arrival host there us he room didnt check

Topic #9:
great location good great location good location value highly recommended communication place
```

1.6 Output of LDA visualization pyLDAvis