Group 13: Airbnb New User Bookings

Abdulaziz Alkhalefah
asaa1g18@soton.ac.uk
University of Southampton
Southampton, UK

Rohan Bungre
rsb1g17@soton.ac.uk
University of Southampton
Southampton, UK

Harry Goatman
hjg1g17@soton.ac.uk
University of Southampton
Southampton, UK

Jack Pennington
jcp3g17@soton.ac.uk
University of Southampton
Southampton, UK

Andrea Sanchez Fresneda
asf1g17@soton.ac.uk
University of Southampton
Southampton, UK

William Savage
ws1g17@soton.ac.uk
University of Southampton
Southampton, UK

ABSTRACT

This report outlines the steps taken to predict new Airbnb users' first booking destination, with justifications and conclusions made throughout the different processes. We investigated a range of data mining techniques to; explore the dataset, including feature instance distributions and correlations between features; prepare the dataset through feature engineering, scaling and encoding; and predict the classes using supervised algorithms able to handle multiple classes. Ultimately, an $F1 - score$ of 0.83 and an $nDCG$ score of 0.86 on the official Kaggle competition was achieved, beating our initial baseline model.

CCS CONCEPTS

• **Computing methodologies → Classification and regression trees**; *Supervised learning*; **Machine learning**; **Ensemble methods**; **Classification and regression trees**; **Supervised learning by classification**.

KEYWORDS

data mining, machine learning, classification, supervised learning, kaggle, airbnb

ACM Reference Format:
Abdulaziz Alkhalefah, Rohan Bungre, Harry Goatman, Jack Pennington, Andrea Sanchez Fresneda, and William Savage. 2021. Group 13: Airbnb New User Bookings. In *Proceedings of COMP6237: Data Mining (COMP6237).* ACM, UoS, Southampton, UK, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The problem we chose to tackle was Airbnb's Kaggle competition: New Users Bookings [1]. The objective was to predict which country new Airbnb users will book as their first destination. Accurately predicting a user's booking, Airbnb can help their users tackle the overwhelming task of exploring stays in over 34,000 cities, ensure personalised advertisements reach the appropriate audience, and overall further understand their users.

This is a predictive data mining problem complemented with rich enough data to allow us to explore a range of user habits and apply a variety of techniques and models to evaluate and compare results.

[1] https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings/data

1.1 Data Description

Airbnb provided 5 datasets, including user demographics, web session records and some general summary statistics about the destination countries (details on these are provided in section 2). However we needed not to use all the datasets to build the predictive model. Thus our first challenge was to decide what data was useful to the problem (based on analysis). Note: all users in the datasets are from the USA.

1.2 Baseline Model

A preliminary baseline model was built prior to any exploration or model-building to help understand the task/data, for use during the evaluation stage, and to set an initial value to improve from. It was built following these steps:

(1) Find top 5 most common destination countries amongst all users in train dataset
(2) For each user in test, predict their destination as the top 5, to calculate nDCG with $k = 5$ (refer to subsubsection 1.3.2).

The model was submitted to the Kaggle competition and obtained an nDCG score of 0.85 (refer to subsubsection 1.3.2) and a cross-validation score of 0.807.

1.3 Evaluation Strategy

1.3.1 Cross-validation. To validate our model, we opted for k-fold cross validation (with $k = 5$), allowing us to assess how the results of the models will generalise to an independent dataset and flag problems such as overfitting or selection bias. In k-fold cross validation, the training data is randomly partitioned in k equal sizes (non-overlapping), wherein a single partition is held out for validation and $k - 1$ remaining partitions are used for training. This process is repeated k times (for each of the partitions is to be used as testing data once), the results are then averaged to produce a single score.

1.3.2 Metrics. Our key metric to evaluate model performance was **normalised discounted cumulative gain** (nDCG). It is a measure of ranking quality and it is used by the official competition (with $k = 5$), thus it was set as our key performance metric:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}$$
$$nDCG_k = \frac{DCG_k}{IDCG_k}$$

where:

rel_i = relevance of result at position i

$IDCG_k$ = maximum possible (ideal) DCG for a given set of queries

Additionally, F1 score was used to measure classification accuracy of our models and compare them.

2 DATA INSPECTION

This section aims to provide a description into our data exploration techniques with justification, gained insights from the data and overall conclusions to get data ready for the next stage.

2.1 Countries & Age-gender buckets datasets

These two provided datasets supplied with general, summary statistics on the destination countries. The former dataset included details on latitude, longitude, language and total area per country. While the latter provided with population distribution as broken down by demographics (age and gender wise) of the destination countries.

2.2 Train dataset

This dataset was the main dataset containing the potential features to be used and the target column (destination country), i.e. the examples to be mainly used during the learning stage of the model. Hence, we proceeded to explore the dataset in a rigorous manner. We found there to be 213,451 entries (no duplicate user id's nor rows) and 15 features (not including the target column), with both categorical and continuous data types.

2.2.1 Univariate. For this analysis, we looked at individual features, such as counts of the different options for every feature, percentage distribution of the different options for every feature and histograms of age distribution.

Our main results and conclusions from this were how the data was highly imbalanced, with 'NDF' dominating with almost 60% of presence over the other 14 countries (refer to Figure 1). With regards to age, there were outliers which were to be dealt with such as extreme values over 100 (2000+) and values under 18, which are illegal as you must be at least 18 to create an account and book on Airbnb. Additionally, the majority (45%) of users chose not disclose their gender and Apple devices are the most popular amongst users. Regarding dates, June was the most popular month to create an account, perhaps users wait until the start of summer to book their holidays, as seen in Figure 2. Finally, there was an extensive amount of `first_browser` instances (52) with the majority appearing < 100 times, thus this will be looked into during feature engineering.

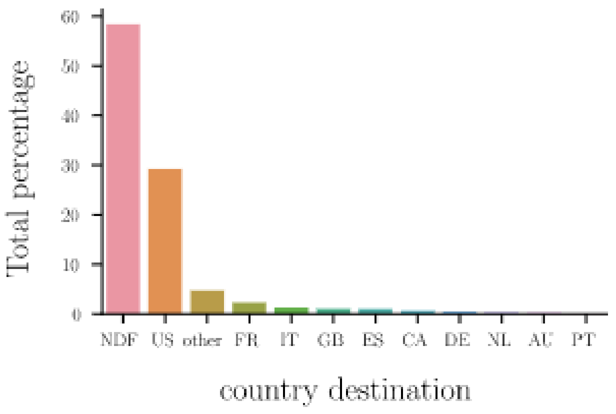

Figure 1: Target column options distribution

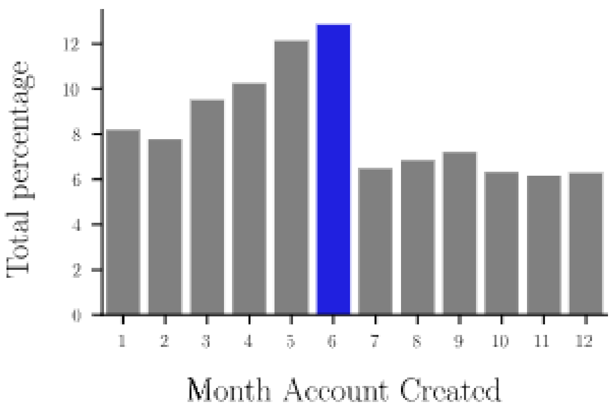

Figure 2: Accounts created per month distribution

2.2.2 Bivariate. The main aim of this was identify any correlations or patterns between different features and the target column, for feature selection and/or feature engineering.

Our findings were as follows. During the winter months, Australia gets booked the most, perhaps as it summer time there. There is a high correlation between 'NDF' and non-disclosed user information on their gender and/or age (plots for these seen in Figure 3 and Figure 4).

2.2.3 Multivariate. We used `t-SNE` with two dimensions to visualise multivariate analysis [2]. Ultimately, no clear clustering was observed as seen in Figure 5. It is to note there might be some clustering found in higher dimensions, however for visualisation purposes, we performed it with only two dimensions.

2.3 Sessions dataset

Contains a log of the users' (in train and test dataset) actions from 2014. Provides details on every click, search, page visited and time between actions. The amount of sessions details per user varies vastly (between 1000 – 6.5million rows per user), therefore we had

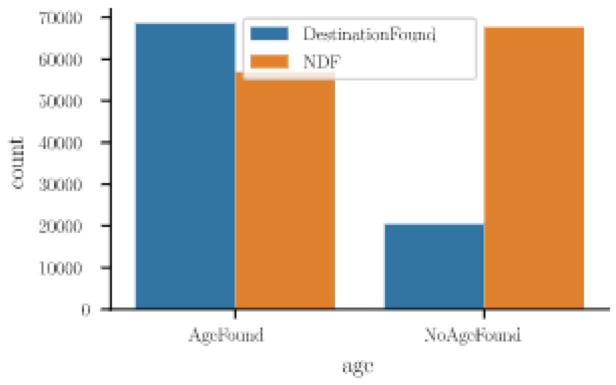

Figure 3: Age Disclosed and Country destination found

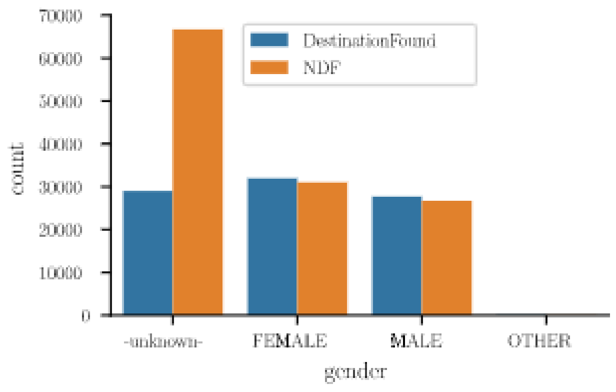

Figure 4: Gender options for destination found or not

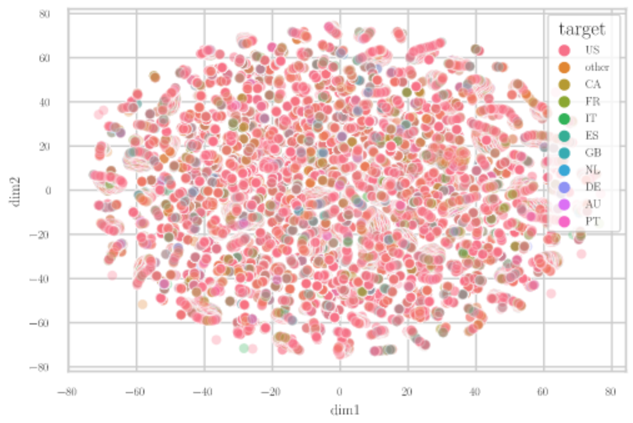

Figure 5: t-SNE on train dataset

to consider how to approach this dataset, and how to condense to a single entry per user (for efficiency purposes and joining with the train dataset). Furthermore, only around 30% of the users in the training dataset were also found in this dataset, entailing we would have to either drop the remaining entries, or extrapolate/fill in the data on sessions for the remaining training subset without such data. We considered having a clustering algorithm to artificially generate this data based on their assigned cluster, however this was deemed as too time consuming and the potential improvements not rewarding enough.

2.4 Conclusions

Ultimately, the training dataset (`train_users_2.csv`) was the only in use during the learning phase of the models, discarding the remaining datasets as they were deemed as data dumps (countries' demographics) or too time consuming (sessions) to clean and prepare, the problems it brought outweighed the possible improvements gained. Within the train set, we would look into engineering features from the devices, browsers and dates. We were also to drop `date_first_booking` feature as the problem involved predicting users who have not yet booked, thus could lead to overfitting (this column will most likely be empty for testing sets, or not present at all).

3 DATA PREPARATION

This section outlines the pre-processing techniques applied to the train set. The overview of our pipeline can be seen in Figure 6.

3.1 Data Cleaning

The dataset was relatively clean, no duplicate id's entries were found and empty values were only found in 3 of the columns. This lead to us ensuring those missing values were all of the same type: `np.nan`, as they were marked as '-unknown-' or 'NaN'.

3.2 Feature engineering

One-hot encoding raw features produced an extremely sparse training set, mainly from the larger number of values in `first_browser` feature (52 options, some used significantly infrequently). There was found to be values containing the mobile version of the browser, which was excess data as user's hardware type was provided in another feature. Casting the browser with a relative frequency of $< 0.5\%$ to `other` and mobile browsers to the desktop version (e.g. `Chrome_mobile` to `Chrome`), we reduced the values from 52 to 5.

We also engineered features from `first_device_type` by splitting it into two columns: hardware type and manufacturer. As the original contained both in one string, making it hard for the machine to understand, as well as providing more useful information (e.g.: `Mac_Desktop` split into `Mac` and `Desktop`).

Regarding date features, they were engineered into 4 other features:

- `year`: year of the date e.g. 2020, 2021
- `month`: numerical representation of the month e.g Janurary = 01, August = 08
- `day`: numerical representation of the day of the month e.g 04/05/2021 = 04
- `dayofweek`: numerical representation of the day of the week e.g. Monday = 01, Friday = 05

Encoding the date features in this way will allow, for potential patterns or seasonality to be inferred from the models.

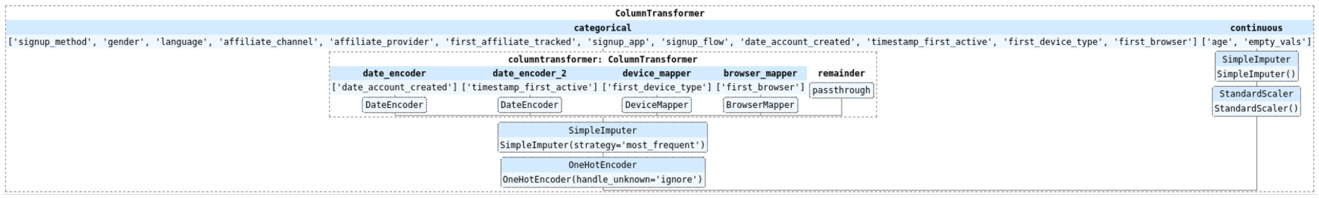
ColumnTransformer
categorical
['signup_method', 'gender', 'language', 'affiliate_channel', 'affiliate_provider', 'first_affiliate_tracked', 'signup_app', 'signup_flow', 'date_account_created', 'timestamp_first_active', 'first_device_type', 'first_browser']
continuous
['age', 'empty_vals']
columntransformer: ColumnTransformer
date_encoder
['date_account_created']
DateEncoder
date_encoder_2
['timestamp_first_active']
DateEncoder
device_mapper
['first_device_type']
DeviceMapper
browser_mapper
['first_browser']
BrowserMapper
remainder
passthrough
SimpleImputer
SimpleImputer(strategy='most_frequent')
OneHotEncoder
OneHotEncoder(handle_unknown='ignore')
SimpleImputer
SimpleImputer()
StandardScaler
StandardScaler()

Figure 6: Pre-processing Pipeline

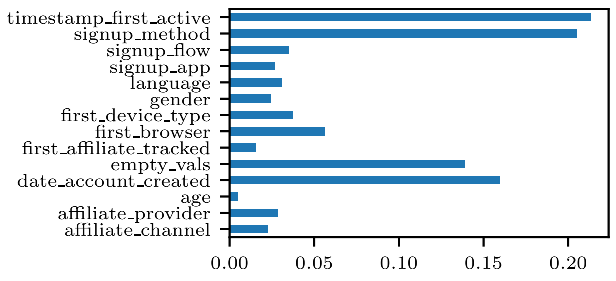

Figure 7: Feature importance plot

The final engineered feature was a simple column which counted the number of missing values per entry, as it was found there was high correlation between missing values and the most popular class 'NDF', during the exploration phase.

3.3 Feature Selection & Importance

As previously mentioned in subsection 2.4, the timestamp of first booking feature was dropped, additionally to this, the `id` feature was also dropped as it was redundant (only used to identify users, no substantial information obtained from it).

To identify which features were supplying the most information, we plotted a feature importance chart: Figure 7 (using our best model `XGBClassifier`). The engineered date features were amongst the most important, as it is a time-sensitive problem, this was expected (e.g. different months of the year influence due to seasons, weekends v weekdays, only create account to book...). Based on our analysis, 81% of users with 'Google' as `signup_method` had an 'NDF' country destination, whereas for the other two options, it was much more balanced. Therefore, as 'NDF' is the largest class, it is relatively easy to predict for this value from this feature. Finally, the feature `missing_value_counts` was also highly important, likely because users may fill in less information when they are just browsing ('NDF') or only Airbnb requires certain information to book.

3.4 Data Transformations

Depending on the data type, the features were dealt with accordingly. Most features were found to be categorical, with only age and the empty values counter (`empty_vals`) deemed as continuous.

3.4.1 Categorical. For features of this type, we applied a `Column Transformer` with the following steps:

(1) `Simple Imputer` for imputing missing values using the mode - cannot produce a mean or median as they are discrete classes
(2) `One Hot Encoding` to encode values, so no relationship between values can be inferred by the model (as compared to using ordinal encoding)

3.4.2 Continuous. For this type, we applied a `Column Transformer` with the following steps:

(1) `Simple Imputer` for imputing missing values using the mean
(2) `Standard Scaler` to scale the features - standardised features by removing the mean and scaling unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where:

m = mean

σ = standard deviation

3.5 Dimensionality Reduction

We looked into investigating how performing PCA on our cleaned dataset would affect our most effective models' performance. However, as most of the data is categorial and `One Hot Encoded`, we opted to use `Truncated SVD` as it works efficiently with sparse matrices. We aimed to keep 50% most effective dimensions. The results of this are found in section 5.

3.6 Balancing Data

After analysing the dataset it was clear the classes were imbalanced, as seen in Figure 1. Thus we explored running our best performing model on a rebalanced dataset. The two ways we rebalanced our dataset were as follows:

- **Random under-sampling:** Random under-sampling removes samples from the majority class, with or without replacement. However, it can lead to increase in variance of the classifier and can degrade results due to discarding potentially useful or important data.
- **Random over-sampling:** Random over-sampling involves supplementing the training data with multiple copies of some of the minority classes. However, this can lead to dramatically increased dataset size.

The results can be found in section 5.

4 PREDICTIVE MODELS

As the problem in hand is a supervised (labelled data to train on), multi-class classification problem, thus we considered models appropriate for this. Furthermore, it required to obtain the top 5 predictions (for the metric), so models able to to predict probabilities for each class was another requirement. Following are the different model types we tried:

Linear - Used as a starting point to gain familiarity with the data and how to work with the pipeline. Scikit-learn's `LogisticRegression` [6] model was used with the parameters set to work with a multi-class classification problem. Performed poorly, only marginally beating the baseline. This is expected as the data is not linearly separable, thus kernels would have been needed (rather, we decided to move on to different models).

Decision Trees - Supervised learning models able to work with multiple classes, good starting point before approaching more complex models, met our requirements for a model. Although easy to understand, they required a substantial amount of fiddling parameters to train, which did not seem like a good approach. Additionally, we found them to be overfitting based on the difference between training and validation scores. However the scores were more promising, therefore the decision was made to investigate Random Forests.

Forests - Basic model of a Random Forest provided in Python library scikit-learn [6], with `entropy` as splitting criterion (this decision was trivial, as choosing between the two options has no significant difference in the results [7]). The idea of building a collection of trees to decide the outcome predicted class reduces the overfitting observed in Decision Trees [4], an issue we came across with. Advanced Forest models were explored given the encouraging results obtained from the basic model:

- LightGBM - based on decision tree algorithms using gradient boosting, used for ranking and classification, thus an appropriate option. Popular choice for Kaggle data science competitions.
- AdaBoost - most popular boosting algorithm [7], used for classification rather than regression. Although it was originally for binary problems, is now able to work with multi-class problems; we used scikit-learn's model [6].
- XGBoost - an optimised algorithm based on parallel tree boosting [3] (rather than bagging like Random Forest algorithms), boosting is a standard when dealing with data mining problems, particularly classification problems [7].

Artificial Neural Network - Particularly, we used a feedforward one (i.e. information only moves forward): an MLP (Multi-layer Perceptron classifier). This was due to its ability to distinguish non-linearly separable data [1], something we discovered our data was from analysis. As choosing a learning rate is an intricate task, we used an adaptive learning rate for its general outperformance over configured ones [5].

5 RESULTS & EVALUATION

The models' results can be found in Table 1.

With dimensionality reduction, as described in subsection 3.5, we ran a `RandomForest` model and obtained a lower `nDCG` value of 0.844, compared to without: 0.882. We had under 20 features, thus we did not advantage from performing dimensionality reduction. Nonetheless, it did reduce time taken for training by over 20 minutes, which is a notable benefit.

With the balancing of the dataset as described in subsection 3.6, we ran our best performing model on the under-sampled dataset, which had a low `nDCG` score of 0.327. We suspected this to be due to the large decrease in the size of our dataset. When ran on the over-sampled data we got a `nDCG` score of 0.979. This led us to run our top 3 best performing models on the over-sampled dataset. Refer to Table 2 for the results. The high values for the tree methods we suspected to be due to over-fitting, due to the large duplication of data from over-sampling, as the difference between the size of the majority class and the smallest minority class is extremely large.

We also explored building a more complex ensemble model from our top 3 models: `XGBoost`, `MLP` and `RandomForest` (the first and last model are themselves types of ensemble methods). It performed worst than our best model (0.88) and took almost an hour to run, in addition to it being harder to tune parameters. Nonetheless, we suspected it to be less overfitting than some of the previously used models due to the ensemble's robust nature, and so for future works, further investigation into complex ensemble methods should undoubtedly continue. Due to the increase in computational cost and complexity and decrease in available time left for project completion, any further ensemble methods explorations were halted and we decided to concentrate on evaluating and discussing our achieved results.

Table 1: Table of results

MODEL	nDCG (validation)	F1 Score (validation)
Baseline	0.807	-
DecisionTree	0.825	0.754
LightGBM	0.821	0.605
LogisticRegression	0.814	0.533
MLPClassifier	0.900	0.816
RandomForest	0.882	0.800
XGBClassifier	0.903	0.828

Table 2: Table of top 3 performing models on over-sampled dataset

MODEL	nDCG (validation)	F1 Score (validation)
MLPClassifier	0.484	0.664
RandomForest	0.996	0.99
XGBClassifier	0.979	0.955

5.1 Reflection

Overall, our tree-based models performed better than other models. These models are popular amongst the Kaggle community and its competitions, due to the high scores they obtain. Ensembles of trees often work well due to the combination of weak learners derived from the dataset, to make a good strong learner that is able to generalise efficiently and well. The dataset provided allowed us to build these required highly uncorrelated weak learners, thus performing so satisfactorily. XGBoost and RandomForest work on this principle so performed the best as well as provided with the added benefit that they tend not to overfit. Yet our final submission to Kaggle was lower than the cross-validation scores we obtained. We suspect either the dataset to train on was not representative enough, or our implementation of nDCG may have not been entirely correct. This would have to be re-verified in future works. Similarly, the F1 Score on certain models was considerably higher than the nDCG one. We concluded this to be due to the former being a better metric when imbalanced class distribution exists.

Debatably, using the sessions dataset in addition may have improved our final scores. However based on our analysis and discussions, we concluded potential improvements achieved using this latter dataset too would have not been worth the amount of time it would have taken to get the data ready. This is because the sessions data needed to be condensed into single rows per user and perhaps use methods such as TF-IDF to deal with the action details, as well as necessary data cleaning and feature engineering. With the added problem of how only a small percentage of the sessions users are also present in the training set. Further investigation into the sessions dataset undoubtedly should be performed in future works.

We refrained from exhausting parameter tuning as we decided exploring and trying a range of techniques would be more beneficial, nonetheless, in future, techniques such as GridSearch would be an approach to try for this. Additionally, we considered experimenting with more complex Deep Learning models, but due to limited computation availability, we were unfortunately not able to. This is something else to be researched into in future works.

Ultimately, from a business-perspective, Airbnb do not lose much if any users are misclassified, as the user themself would easily be able to correct where they would like to book first, without Airbnb's guidance. Moreover, most users were predicted to be 'NDF', in both training and testing, making it hard for Airbnb to accommodate the website to the user, so techniques such as gathering additional information of the users, whether directly or indirectly would highly benefit the company with this first user booking problem. Another approach to these cases could be simply providing information on the most popular destinations, and thus using 'wisdom of the crowd'. Nevertheless, even if not used for predicting users' first booking destination, the datasets provided are rich enough to gain other insights into Airbnb users, potentially helping the company. As an example, most users' affiliate channel was basic, suggesting how Airbnb should improve their marketing strategies in other channels such as semi-brand and perhaps remove themselves from some which are significantly less present amongst users' data.

6 CONCLUSION

Rigorous data inspection unveiled interesting patterns amongst Airbnb users and highlighted weaknesses in the data quality. We were able to clean and engineer features, aiming to improve the data, as well as using state of the art techniques to pre-process the data. Exploring a range of models enabled us to learn a range of machine learning approaches used in industry and ultimately achieved an nDCG score of 0.86 with the XGBClassifier in the Kaggle competition, beating our initial baseline model. Given the time scope and experience, this is a rewarding result and most importantly, the exposure gained to a range of techniques will prove valuable for future works.

REFERENCES

[1] Mutasem Alsmadi, Khairuddin Omar, Shahrul Azman Mohd Noah, and Ibrahim Almarashdah. 2009. Performance Comparison of Multi-layer Perceptron (Back Propagation, Delta Rule and Perceptron) algorithms in Neural Networks. *2009 IEEE International Advance Computing Conference, IACC 2009*, 296 – 299. https://doi.org/10.1109/IADCC.2009.4809024

[2] Renesh Bedre. 2021. t-SNE in Python. https://www.reneshbedre.com/blog/tsne.html.

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug 2016). https://doi.org/10.1145/2939672.2939785

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, California.

[5] Russell D Reed; Robert J Marks. 1999. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* Bradford Book, Cambridge, Mass.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[7] Laura Raileanu and Kilian Stoffel. 2004. Theoretical Comparison between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence* 41 (05 2004), 77–93. https://doi.org/10.1023/B:AMAI.0000018580.96245.c6