Airbnb New User Booking

Meher Jyothi Karpurapu, Nikhilesh, Janani Suresh Kumar

Abstract - **Machine learning is an application of artificial intelligence (AI) that gives machines the capacity to learn consequently and enhance from experience without being unequivocally programmed. There's a lot of information being collected every time we go through any website and distinctive companies are interested in the collection of that information which increases the sum of information that companies have accessible for client profiling (Webb & Micheal J Pazzani, 2001). In this paper, we explain our current best solution to predict Airbnb's new users' booking destination which will help us to find where the new user will book their first travel experience with Airbnb. First, we show our findings of the dataset characteristics which are given by the Airbnb on Kaggle competition and going forward we will talk about our feature selection strategy and different modelling techniques used based on these observations. We compare these models by creating a classification report for each of these which captures the precision, recall and f1 score of the classification model. These are important metrics for our dataset as the data is imbalanced and accuracy score proves to be inadequate for evaluation.**

Keywords

XGBoost, Decision Tree, Random Forest, Ridge Regression, Extra Trees, Classification, Supervised Learning

1. INTRODUCTION

1.1 *Background*

Airbnb has become a really prevalent choice among travellers around the world for the kind of unique experiences that they give additionally for presenting cheap, pocket-friendly staying options in more than 34000+cities over 190 nations. [5] Airbnb has maintained its market dominance over the past three years. In 2017, the company paid$ 200 million for Luxury Retreats, followed by a$ 400 million purchase of HotelTonight in 2019. Some of these accessions, it's still considered a rather " light" asset company in comparison to its main rivals, Expedia andBookings.com [6] The COVID-19 has impacted negatively on Airbnb, which has been one of the hardest hit platforms.. New bookings were

down 85% at the peak of the pandemic, while app usage has progressively increased as governments have eased travel restrictions. Users can make their bookings either through the site application or using the iOS/Android application. Airbnb is consistently attempting to improve the user's first booking experience. [7]

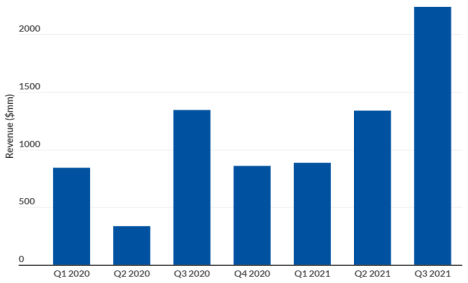

Fig 1. Airbnb Revenue

The exact forecast makes a difference to diminish the normal time required to book by sharing more personalized proposals additionally in superior determination of the request. Particularly, given the information of 210,000 users' behaviors, members were inquired to foresee in which nation a modern client will make his or her first booking.

With the essence of machine learning, the system can be designed which will accurately predict where the user will book for the first time and subsequently decrease the average booking time and improvise the booking experience. Airbnb can meaningfully utilize it to better personalize its marketing tactics and increase its booking rates.

1.2 Problem Definition

In this challenge, we are provided a list of people, their demographics, web session logs, and some summary statistics. We need to anticipate which countries a new user would travel to for their first booking. The users in this dataset are all from the United States. There are 12 possible outcomes of the destination country: 'US', 'FR', 'CA', 'GB', 'ES', 'IT', 'PT', 'NL','DE', 'AU', 'NDF' (no destination found), and 'other'. The 'NDF' is different from 'other' because 'other' means there was a booking, but it is to a country not included in the list, while 'NDF' means there wasn't a booking.[5] This

paper tackles how we approached the problem and created a model for predicting the user's intentions.

1.3 Our Solution

To begin with, we analyze the dataset by visualizing different features. We perform data cleaning and preprocessing by imputing missing values and outliers. Then, feature selection is done by plotting the correlation graph and dimensionality is reduced using Principal Component Analysis (PCA). For model selection, we utilize supervised learning algorithms. Based on the imbalance in data, we perform resampling of the dataset. We select five models based on different factors then analyze and compare the performances of each of these algorithms. We optimize these algorithms by tuning their hyperparameters and evaluating the performance using f1 scores.

1.4 Related Work

In [8], they utilized only the user dataset, neglecting the session dataset. They have followed the XG boost algorithm for the prediction of destination countries. Data cleaning was done by removing the NaN values, splitting the timestamps, and categorization of all non-numeric data such as gender, language, device type, browser type etc. The performance of this approach is low as they do not consider user sessions in their modelling. Plotting the confusion matrix depicts a high misclassification rate as the model only predicts two labels, i.e. NDF and US for all cases.

In another paper [9], they select the number of actions performed by each user and the type of action performed by them from the sessions dataset and map it with the user dataset. The data cleaning is performed by dropping a few columns and filling all the NaN values by -1. They have also followed the XG Boost with parameters learning rate as 0.1 , a number of estimators as 200, max depth as 5. This results in improved performance than [8].

2. DATA EXPLORATION

2.1 Dataset Analysis

This Airbnb dataset is featured in the Kaggle competition for the New User Bookings competition. It includes 5 datasets to help with the prediction of the travel destination country.

This dataset includes information like demographics of specified countries, users web session records and some summary statistics. All information is captured in four files[6]:

- train_users.csv - the training set of users
- test_users.csv - the test set of users. As columns, it includes id (user id), date_account_created (the date of account creation), timestamp_first_active (timestamp of the first activity, note that it can be earlier than date_account_created or date_first_booking because a user can search before signing up), date_first_booking (date of first booking), gender, age, signup_method (application/website signup method), signup_flow (the page a user came to signup up from), language (international language preference), affiliate_channel (what kind of paid marketing), affiliate_provider (where the marketing is e.g. google, craigslist, other), first_affiliate_tracked (what's the first marketing the user interacted with before the signing up), signup_app, first_device_type, first_browser, country_destination (this is the target variable you are to predict).
- sessions.csv - web sessions log for users. As columns it includes User_id (to be joined with the column 'id' in users table), action, action_type, action_detail, device_type (type of device used), secs_elapsed
- countries.csv - summary statistics of destination countries in this dataset and their locations. As columns it includes country_destination (destination countries), lat_destination(the latitude coordinates of the country), lng_destination ((the latitude coordinates of the country), distance_km(the distance of US from the destination country), distance_km2, destination_language (destination country native language) , language_levenshtein_distance.
- age_gender_bkts.csv - summary statistics of users' age group, gender, country of destination. As columns it includes age_bucket (21 age buckets e.g. 0-4, 100+), country_destination, gender, population_in_thousands, year.

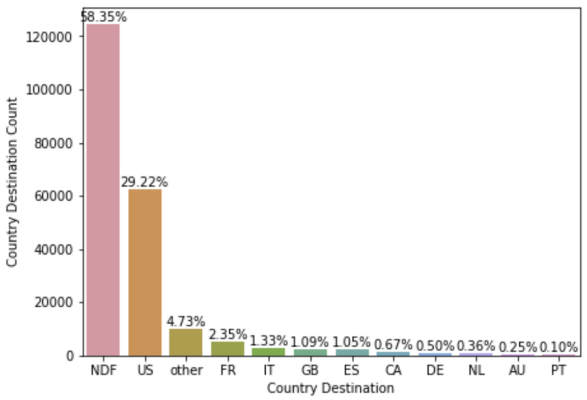

Fig 2. Distribution of destination countries

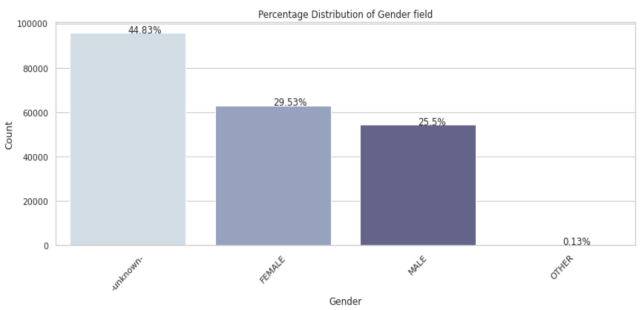

Fig 3. Distribution of gender

2.2 Data Pruning

Data Pruning is one of the techniques used for removing unused parameters to solve overfitting problems. In Figure 4, session_csv dataset has 6 columns user_id, action, action_type, action_detail, device_type and secs_elapsed. We dropped unnecessary fields like action_detail and device_type. In the sessions.csv dataset, there are a lot of repetitive users with different actions. We aggregated the actions by counting all actions a user performed and store that by mapping to user id. Lastly, we filled the missing values in session.csv with 0.

Out[3]:

	user_id	action	action_type	action_detail	device_type	secs_elapsed
0	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	319.0
1	d1mm9tcy42	search_results	click	view_search_results	Windows Desktop	67753.0
2	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	301.0
3	d1mm9tcy42	search_results	click	view_search_results	Windows Desktop	22141.0
4	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	435.0

Fig 4. sessions.csv before pruning

Out[9]:

	user_id	action	secs_elapsed
0	00023iyk9l	39	738079.0
1	0010k6l0om	63	586543.0
2	001wyh0pz8	90	282965.0
3	0028jgx1x1	31	297010.0
4	002qnbzfs5	782	6463327.0

Fig 5. sessions.csv after pruning

Data	Users	Number of Features
Dataset before pruning	10567737	6
Dataset after pruning	135483	3

Table 1: Sessions Dataset Information

2.3 Data Cleaning And Preprocessing

Data cleaning is the process of identifying and correcting errors in a dataset that may have a negative impact on the model. Data cleaning involves cleaning the null values and outliers from the dataset. The following steps are performed in cleaning the dataset.

1. Merged session dataset with training and testing dataset using user id
2. Filled missing values with data
3. Changed date format
4. Removed Outliers

We first merge the user details and sessions dataset to create one dataset by mapping them using user id. We then proceed with data cleaning and preprocessing.

A. *Filled missing values with data*

Firstly, cleaning a dataset includes filling or replacing missing values. Missing values in the dataset are by default represented as NaN (Not a number). We used the functions isnull() and notnull() to check for missing values. To fill null values in the columns of secs_elapsed, action and action_type columns, we have used -1. To fill the null values of age, the median of age is calculated, and the null values are filled using the median value which is 34. In the gender column, there are two variables used: other and unknown. We combined both 'unknown' gender and 'other' gender, as they are the minority classes with very low frequency.

B. *Changed date format*

We have three different fields in the user datasets that contain data on different dates: date_account_created, timestamp_first_active, and date_first_booking. Since this date first booking only has a value for users who have booked a trip, it is useless for prediction because none of the test users will have one. We will simply remove the field as it has 58% null values. The other date fields specify when the user first used the site and when they decided to create an account. Before we can clean these fields, we need to change the format to one that we can easily modify. This data is currently stored as strings, but we convert it to date objects that can be easily modified. Hence, we separated year, month, and date for timestamp_first_active and date_account_created.

C. *Removed Outliers*

Outliers increase the variability in the dataset, which decreases statistical power. Consequently, excluding outliers can cause the results to become statistically significant. Outliers in the data can sabotage and mislead the training process, resulting in longer training times, less accurate models, and poorer results. In the training dataset, the outliers present in the age field are below 15 and above 110.
Fig. 6 shows the plotting of the boxplot of 'age' to confirm the presence of outliers. Fig. 7 shows the plotting of the boxplot of 'age' after removing outliers

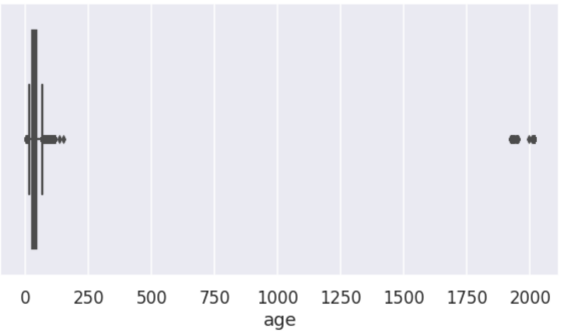

Fig 6. Age with outliers

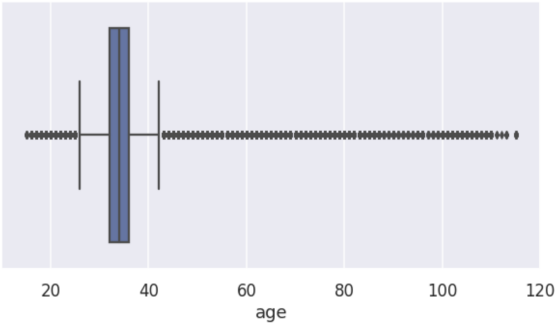

Fig 7. Age without outliers

Data Preprocessing

Data preprocessing is an essential step in Machine Learning because the quality of data and the useful information that can be extracted from it directly affects our model's ability to learn and thus, it is critical that we preprocess our data before feeding it into the model. As shown in Figure 8, the data for age is right-skewed (long tail to the right). Due to this skewness, we use median imputation to replace unknown ages with the median of the age, which is 34.

Checked skewness in age:

The data is positively skewed or skewed right if the skewness is positive. When skewness is negative, the data is negatively skewed or skewed left. When skewness equals zero, the data are perfectly symmetrical. Many statistical models fail when there is too much skewness in the data. The tail portion of skewed data may act as an outlier for the statistical model, and we know that outliers affect the model's performance.

- Initial skewness: 1.94737023 (right skewness)
- Removed skewness with applying log(age).
- Skew after correction: 0.52898

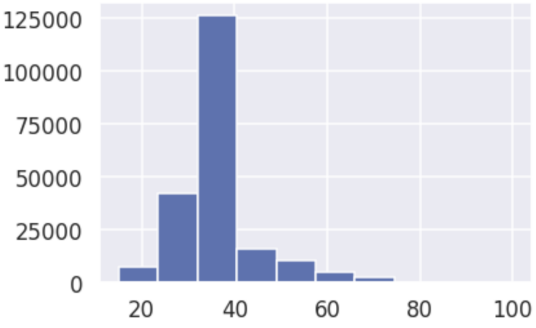

Fig 8. Right skewed data of age

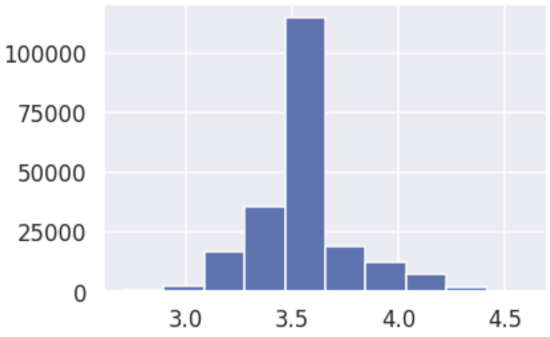

Fig 9. Removing skewness by taking log(age)

2.4 Feature Parsing

After discarding the NaN data, we begin to investigate the basic information of train_user_df. We've noticed that many of the features in train users are not numerical, but rather use string to present different values. To make the models work properly, we encode them using the one-hot technique for the computer to be able to see relevant patterns in the data. The standard scaler is used to transform data into distribution with a mean of 0 and a standard deviation of 1.

2.5 Feature Selection

We perform feature selection for three reasons:

- **To minimize overfitting -** With less redundant data, there is less chance of making decisions based on noise.
- **To increase Accuracy -** Less deceptive data gives better results.
- **Reduce training time -** Lesser data means less time for analysis.

We should remove features with low variance. After combining features of the session dataset with training and training dataset and parsing from the raw data, we have 18 features which include id, date_account_created, timestamp_first_active, date_first_booking, gender, age, signup_method, signup_flow, language, affiliate_channel, affiliate_provider, first_affiliate_tracked, signup_app, first_device_type, first_browser, country_destination, action, secs_elapsed.

The correlation matrix describes how two or more variables are related to one another. These variables could be input data features that were used to forecast our target variable. We can anticipate one from the other if two variables are significantly correlated. We create a correlation matrix to show which variables have a high or low correlation with one another as shown in figure 10. We visualized the distribution of independent variables to examine if there is any apparent correlation among the features. High correlation can lead to unreliable model results. There is a high correlation between dac_year, dac_month, dac_day and tfa_year, tfa_month and tfa_day respectively, along with action and second elapsed. We, therefore, remove the secs_elapsed, tfa_year, tfa_month and tfa_day to reduce correlation. We divide the entire data frame into two datasets: 80% for training and 20% for testing.

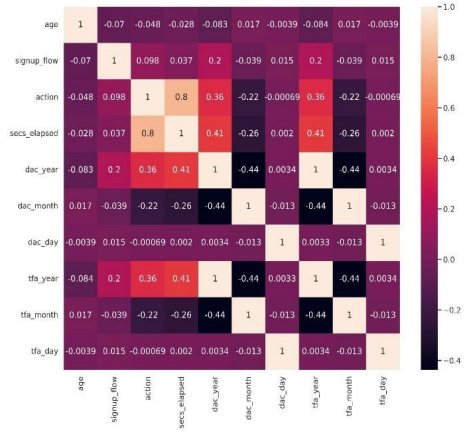

Fig 10. Correlation matrix

2.6 Dimensionality Reduction

Having a high number of dimensions in the data can reduce the performance of a machine learning model.[11] For dimensionality reduction, we employ Principal Component Analysis (PCA). It's a technique that involves projecting each data point onto only the first few principal components in order to obtain lower-dimensional data with as little fluctuation as feasible. The first principal component is a path that maximizes the variance of the projected data.

In our data, to capture 0.9529463612680115 variance, 57 principal components are used.

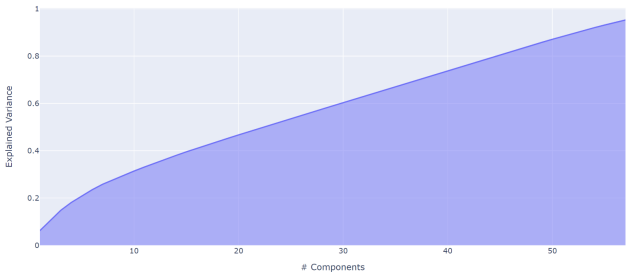

Fig 11. Variation in the variance of data with an increasing number of Principal components

3. HANDLING IMBALANCED DATA

In this dataset, the maximum examples are for 'NDF' and 'US' as destination countries. These majority classes can lead to the low performance of the models. We first use SMOTE to generate synthetic examples of minority classes to ensure class balance. *SMOTE* first selects a minority class instance at random and finds its k nearest minority class neighbours. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b. [4]

This approach aids in overcoming the problem of overfitting caused by random oversampling. It concentrates on the feature space and interpolates new examples between positive instances that lie together. It performs better than random oversampling as random oversampling chooses random samples from the minority class and replaces them in the training dataset. We combine SMOTE with random undersampling to reduce the samples of the majority class, i.e. 'NDF'. Random undersampling reduces the number of examples in the majority classes in the training dataset. One of the main disadvantages of random undersampling is it leads to information loss. Therefore, combining SMOTE with random under-sampling is advantageous. It improves bias towards the minority classes and reduces bias towards the majority class.

We compare the performance of the ridge algorithm and random forest models before and after resampling the data. The results are discussed in further sections.

4. METHODOLOGY

Our target variable is destination country, which is discrete and categorical in nature. For each entry, given the user data, we must predict the country in which the user would book their next destination in. Since there are more than three possible values for the destination country, this corresponds to a multinomial classification problem.

We employ five classification models to find the optimal fit for solving our problem and compare the performance of each of them. To improve the performance of each of these algorithms, we tune the hyperparameters of these algorithms to find the optimal performance.

4.1 Random Forest

Random forest is an ensemble algorithm that generates different training subsets from sample training data with replacement and builds decision trees on these samples. It uses the bagging technique where the final output is based on majority voting. Given the imbalanced nature of our dataset, tree-based algorithms, like the random forest, give better performance given their hierarchical structure. This allows the model to learn from all classes.

There are three parameters that enhance the performance and predictive power of the random forest algorithm. First is the number of decision trees to be built by the algorithm before taking the final vote. Next is the min_sample_leaf which depicts the minimum number of samples required to be at a leaf node. We analyze the performance for 1, 2, and 4 leaves. The final important parameter is the min_sample_split which calculates how many samples are needed to split an internal node. We tune this parameter for 2, 5, and 10 values. We choose a weighted f1 score for comparison as it accounts for class imbalance by computing the average of binary metrics in which each class's score is weighted by its presence in the

true data sample. This is particularly useful in case of class imbalance as in this case. We compare the performance of the algorithm for 200, 400, 600, and 800 trees and find that the performance is optimal when the algorithm generates 400 decision trees, and uses 1 sample at the leaf, considering at least 2 samples for splitting.

We implement the random forest algorithm in two scenarios. First, we implement this algorithm without sampling the data. In the second scenario, we perform sampling of the data by combining Synthetic Minority Oversampling Technique (SMOTE) and Random Under Sampling to synthesize new examples from existing examples of the minority class and reduce the examples of the majority class. We then use the random forest algorithm on the resampled data.

For the first case, implementing the algorithm without any sampling produces the result summarized in figure 12.

	precision	recall	f1-score	support
AU	0.76	0.34	0.47	125
CA	0.69	0.25	0.37	411
DE	0.67	0.21	0.32	295
ES	0.71	0.25	0.36	620
FR	0.72	0.23	0.35	1430
GB	0.67	0.25	0.36	642
IT	0.72	0.25	0.37	778
NDF	0.73	0.86	0.79	34772
NL	0.78	0.21	0.33	226
PT	0.75	0.26	0.39	68
US	0.59	0.55	0.57	17379
other	0.68	0.26	0.38	2781
accuracy			0.69	59527
macro avg	0.71	0.33	0.42	59527
weighted avg	0.69	0.69	0.68	59527

Fig 12. Random Forest Classification report (before sampling)

After resampling the data and then performing this algorithm does not improve the performance as summarized in Fig. 13.

	precision	recall	f1-score	support
AU	0.61	0.36	0.45	125
CA	0.38	0.27	0.32	411
DE	0.33	0.22	0.26	295
ES	0.47	0.26	0.33	620
FR	0.48	0.24	0.32	1430
GB	0.48	0.25	0.33	642
IT	0.49	0.26	0.34	778
NDF	0.78	0.66	0.72	34772
NL	0.49	0.23	0.31	226
PT	0.62	0.26	0.37	68
US	0.48	0.70	0.57	17379
other	0.36	0.29	0.32	2781
accuracy			0.62	59527
macro avg	0.50	0.33	0.39	59527
weighted avg	0.65	0.62	0.63	59527

Fig 13. Random Forest Classification report (After sampling)

We plot the confusion matrix to analyze the results after resampling.

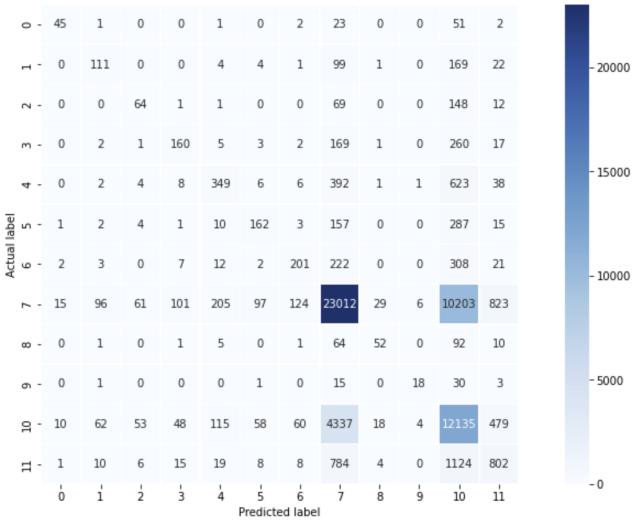

Fig 14. Random Forest Confusion matrix with resampled data

4.2 Ridge Regression

Linear regression minimizes the loss function by selecting coefficients for each independent variable. However, if the coefficients are excessively large, it may overfit the training dataset and lead to model overfitting. Such a model will not be able to generalize adequately to new data. To overcome this, regularization is used to penalize the coefficients. Ridge algorithm is a variant of linear regression which adds a penalty term to the loss function, equivalent to the square of the magnitude of the coefficients, which is also known as L2 regularization.

The way the ridge algorithm is implemented in python is it first converts the target values into {-1, 1} and then treats the problem as a regression task (multi-output regression in the multiclass case). For multi-class classification, n_class classifiers are trained in a one-versus-all approach. [12].

The loss function used by the ridge algorithm is:

$$cost(w) = 1/(2 * n) \sum_{i=1}^{i=n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{j=D} w_j^2$$

where lambda is the regularization parameter. With the increase in lambda, bias increases but on the other hand, variance decreases. This also means that a low lambda value can cause over-fitting, whereas a high lambda value can cause under-fitting.. Since this algorithm regularizes the loss function, it is particularly useful when dealing with unbalanced data, like in this situation.

For this dataset, we implement the ridge algorithm in two cases. First, we implement the ridge algorithm on training data without any resampling. Next, we apply the ridge algorithm after resampling the data using SMOTE and random under-sampling.

When we trained the model without sampling, we tuned the value of alpha for different values to find the optimal performance. We use the value of alpha as 0.01, 0.1 , 0.5 , 0.8 , and 1 and choose the class weights as balanced as it automatically adjusts weights which are inversely proportional to class frequencies in the input data as

$$\frac{n\_samples}{(n_{classes} * np.bincount(y))}$$

wheren_samples is the total number of samples or rows in the dataset, n_classes is the total number of unique classes in the target, and the last term calculates the total number of rows of the corresponding class. [12]

It takes into consideration the frequency of the class and assigns the class weights which are inversely proportional to them. This is useful for imbalanced data. We evaluate the performance based on the weighted f1 score. The results of model performance with different values of alpha are summarized in the diagram below. Optimal performance is found when alpha is set to 1.

```
Best: 0.537204 using {'alpha': 1}
0.537204 (0.003125) with: {'alpha': 0.1}
0.537204 (0.003125) with: {'alpha': 0.3}
0.537204 (0.003125) with: {'alpha': 0.6}
0.537204 (0.003125) with: {'alpha': 1}
```

Fig 15. F1 score with a standard deviation of Ridge algorithm by changing alpha parameter

The performance is summarized below. The f1-score seems to be quite low even after optimizing the hyperparameter.

	precision	recall	f1-score	support
AU	0.00	0.12	0.01	125
CA	0.01	0.04	0.01	411
DE	0.01	0.17	0.02	295
ES	0.02	0.11	0.03	620
FR	0.05	0.04	0.04	1430
GB	0.02	0.06	0.03	642
IT	0.02	0.07	0.03	778
NDF	0.69	0.52	0.60	34772
NL	0.01	0.08	0.02	226
PT	0.00	0.12	0.00	68
US	0.34	0.01	0.01	17379
other	0.06	0.03	0.04	2781
accuracy			0.31	59527
macro avg	0.10	0.11	0.07	59527
weighted avg	0.51	0.31	0.36	59527

Fig 16. Ridge Algorithm Classification report (before resampling)

In the second case, when we employ a combination of SMOTE and random under-sampling to balance the dataset before the ridge algorithm, the results seem to be comparable to unsampled data, with no evident gain. For this experiment as well we keep the class weight as balanced and set alpha as 1. The results are summarized below in the form of the classification report. As clearly evident, the performance does not improve with resampling data, probably because it results in addition of noise. It also results in increased processing time as the number of samples increases. The performance of the ridge algorithm seems to be worse than random guessing. Confusion matrix in figure depicts that while the algorithm detects minority classes, there is a high amount of misclassification.

	precision	recall	f1-score	support
AU	0.00	0.13	0.01	125
CA	0.01	0.06	0.01	411
DE	0.01	0.16	0.02	295
ES	0.02	0.12	0.03	620
FR	0.04	0.03	0.03	1430
GB	0.02	0.06	0.03	642
IT	0.02	0.06	0.03	778
NDF	0.70	0.50	0.58	34772
NL	0.01	0.10	0.02	226
PT	0.00	0.15	0.00	68
US	0.38	0.01	0.02	17379
other	0.06	0.04	0.05	2781
accuracy			0.30	59527
macro avg	0.11	0.12	0.07	59527
weighted avg	0.52	0.30	0.35	59527

Fig 17. Ridge Regression Classification report (after resampling)

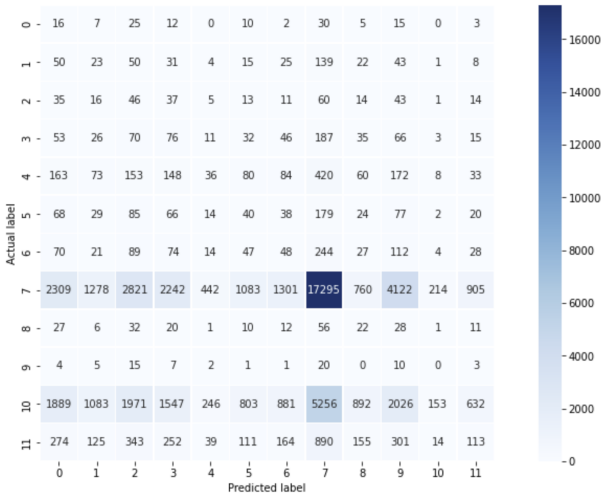

Fig 18. Confusion matrix of Ridge Regression after resampling data

As the results show no improvement in performance by resampling, we implement the other models without resampling data.

4.3 Decision Tree

By learning simple decision rules inferred from prior data, a Decision Tree generates a training model that can be used to predict the class or value of the target variable.

The hyperparameters for this model are as follows:

1. Max depth: This indicates the depth of the tree. The deeper the tree, the more splits it has and it captures more information about the data. But larger depth can lead to overfitting. Keeping other parameters default, we tune the model for depths 12,15,50,80, and 100. By plotting the box and whisker plots as well, performance is found optimal for depth set to 12 as depicted in the table below and visualize it as shown in Fig. 19.

Max depth	Average weighted f1 score
>12	0.558
>15	0.554
>50	0.507
>80	0.505
>100	0.505

Table 2. Evaluation of F1 score by changing max depth, other parameters set to default

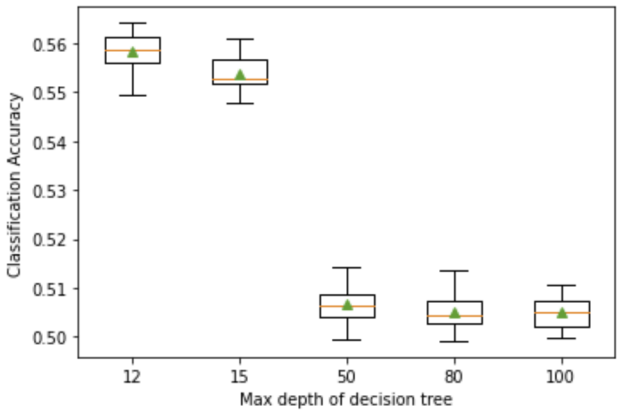

Fig 19. Box and whisker graph of the effect of max depth of decision tree vs its evaluation score

2. Criterion: The function to measure the quality of a split. It decides which attribute to place at the root or at internal nodes. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Keeping depth as 12, we find that our model performs better for entropy.

Entropy is calculated by:

$$Entropy = -\sum_{j} p_j \cdot log_2 \cdot p_j$$

Where p_j is the probability of class j. It is a measure of the disorder of features in the target. The result for the decision tree algorithm is summarized in the classification report in Fig. 20.

	precision	recall	f1-score	support
AU	0.05	0.01	0.01	125
CA	0.13	0.01	0.01	411
DE	0.15	0.01	0.02	295
ES	0.27	0.02	0.04	620
FR	0.22	0.01	0.03	1430
GB	0.10	0.01	0.01	642
IT	0.23	0.02	0.03	778
NDF	0.67	0.84	0.75	34772
NL	0.75	0.01	0.03	226
PT	0.00	0.00	0.00	68
US	0.47	0.43	0.45	17379
other	0.21	0.01	0.02	2781
accuracy			0.62	59527
macro avg	0.27	0.11	0.12	59527
weighted avg	0.56	0.62	0.57	59527

Fig 20. Decision Tree Classification report

4.4 Extra Trees Classifier

Extra Trees Classification is short for Extremely Randomized Trees. This algorithm takes the training dataset and generates a huge number of unpruned decision trees. For classification problems, majority voting is used to decide the final label. It is a variation of the random forest classifier. They both create several trees and separate nodes based on random feature subsets. There are two key distinctions between both: unlike Random forest, extra trees do not bootstrap the observations. This means that it fits the decision on the entire training set. Also, to select an optimal split point, Random forest uses a greedy approach while extra trees select the split point randomly. It also leads to faster execution of Extra trees as compared to random forest algorithm. [1]

This random selection leads to less correlated decision trees and increases variance. Due to the randomness added by this algorithm, we implemented it to compare it with the random forest algorithm.

For optimizing the results, we tune the following hyperparameters in this algorithm. We choose a weighted F1 score for comparison.

1. *The number of trees:* This parameter decides the maximum number of trees to be used in the ensemble. We set the other parameters as default and use the weighted f1 as the evaluation score. We evaluate the model using repeated stratified k-fold cross-validation, with three repeats and 10 folds. We change the maximum number of trees to 10, 50 and 100 and summarize the results as shown in Table 3. We also plot the box and whisker plot to visualize the distribution of the f1 scores. While the average is comparable for all three values, the performance is found to be most stable when the value of maximum trees is set to 50. The following table depicts the effect of changing the number of trees in the ensemble based on the f1 score. This depicts that performance is stable when there are

50 decision trees used. The result has been visualized using box and whisker plots as shown in figure 21.

Ensemble size	Average Weighted F1 score
>10	0.528
>50	0.535
>100	0.535

Table 3. Evaluation of F1 score by changing number of trees in extra trees algorithm, other parameters set to default

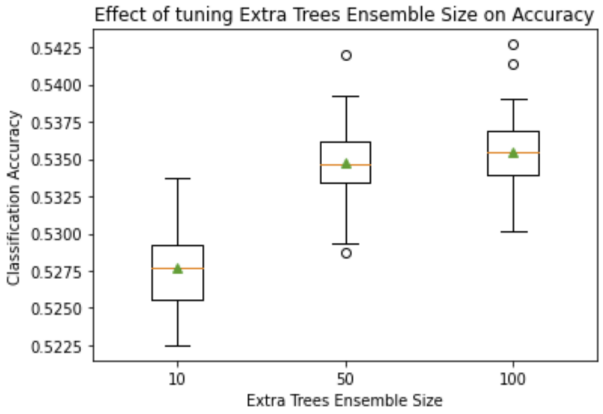

Fig 21. Box and whisker graph of number of the effect of trees in extra trees ensemble vs its evaluation score

2. *Maximum number of features:* This parameter controls the maximum number of features to consider when looking for the best split. At default, it considers the square root of the number of input features. Keeping the number of trees set to 50, we set the value of max features as 1, 2 and 4 to compare the performance of the model based on f1 scores. The results are summarized in Table 2. Box and whisker plots are plotted to understand the distribution of feature set size. Performance is found to be optimal when the maximum number of features is set to 4.

Max features	Average weighted f1 score
>1	0.535
>2	0.536
>4	0.536
>6	0.535

Table 4. Evaluation of F1 score by changing max features in extra trees ensemble, other parameters set to default

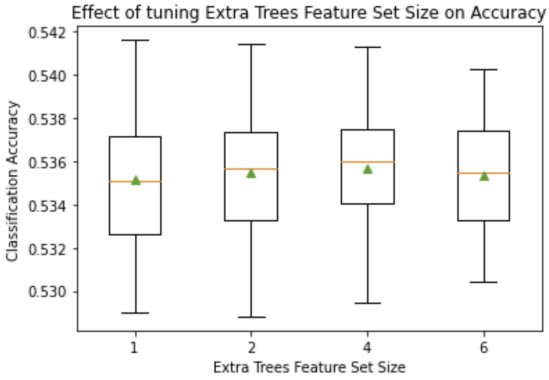

Fig 22. Box and whisker graph of the feature set of varying feature size of extra trees vs its evaluation score

The performance of the algorithm is summarized as shown in Fig. 23.

	precision	recall	f1-score	support
AU	0.56	0.37	0.44	125
CA	0.52	0.27	0.35	411
DE	0.46	0.22	0.30	295
ES	0.52	0.27	0.35	620
FR	0.54	0.25	0.35	1430
GB	0.56	0.26	0.36	642
IT	0.54	0.26	0.36	778
NDF	0.73	0.86	0.79	34772
NL	0.59	0.21	0.31	226
PT	0.64	0.26	0.37	68
US	0.60	0.52	0.55	17379
other	0.58	0.26	0.36	2781
accuracy			0.68	59527
macro avg	0.57	0.33	0.41	59527
weighted avg	0.67	0.68	0.66	59527

Fig 23. Extra tree classifier Classification Report

4.5 Gradient Boosting Classifier

Gradient Boosting is a type of ensemble machine learning algorithm. This type of algorithm builds a strong model by aggregating a collection of weaker models. A weak model is one that performs slightly better than random chance. Boosting is an ensemble strategy that involves adding new models to existing models to correct errors. Models are added in sequential order until there are no more improvements to be made. Gradient boosting gets its name from the fact that it uses a gradient descent approach to minimize loss when adding new models.

Among several implementations of the gradient boosting algorithms, we have implemented the most popular Xgboost algorithm. It stands for Extreme Gradient Boosting. In each round of training, the error function's steepness is described by gradient, which is the partial derivative of the loss function. By "descending the gradient," the gradient is used to determine which direction to adjust the model parameters in order to (maximally) reduce the error in the next round of training. It's built to be both computationally efficient (i.e., quick to execute) and extremely effective.

Since we have imbalanced data, we choose the weighted f1 score to evaluate the performance. We tune the following hyperparameters to optimize the model:

1. *Number of Trees:* In xgboost, the model sequentially adds decision trees to try to correct and improve the predictions made by prior trees. This parameter controls the number of boosting stages to perform. Keeping the other parameters as default, we tune the n_estimates parameter to 10, and 50 and evaluate the model performance. The results are summarized in the diagram below. The optimal performance is viewed at 10 trees as shown in Fig. 24 .

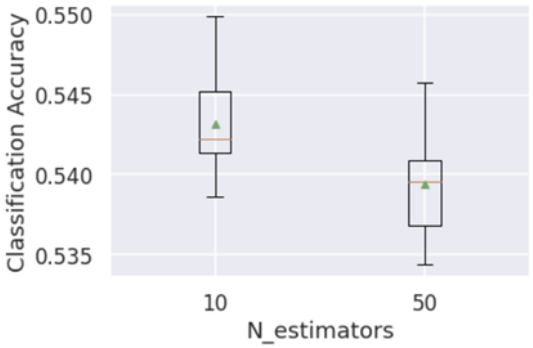

Fig 24. Box and whisker graph of the feature set of a varying number of estimators of extra trees vs its evaluation score

2. *Learning rate:* The learning rate determines how much each model contributes to the ensemble prediction. Smaller rates may need the inclusion of more decision trees in the ensemble. Setting the number of estimators to 10, we explore the learning rate between 0.0.1 to 1. The optimal performance is found when the learning rate is set to 0.01 as summarized in Table 5.

Learning Rate	Average weighted F1 score
0.01	0.545
0.1	0.543
0.3	0.537
0.5	0.545

Table 5. Evaluation of the performance of XGBoost algorithm with varying learning rate

The performance of xgboost algorithm is shown in figure 25.

	precision	recall	f1-score	support
AU	0.00	0.00	0.00	145
CA	0.00	0.00	0.00	386
DE	0.00	0.00	0.00	316
ES	0.00	0.00	0.00	595
FR	0.00	0.00	0.00	1442
GB	0.00	0.00	0.00	670
IT	0.00	0.00	0.00	777
NDF	0.64	0.87	0.74	34461
NL	0.00	0.00	0.00	220
PT	0.00	0.00	0.00	63
US	0.46	0.33	0.38	17292
other	0.00	0.00	0.00	2727
accuracy			0.60	59094
macro avg	0.09	0.10	0.09	59094
weighted avg	0.51	0.60	0.54	59094

Fig 25. Classification report of xgboost algorithm

5. OPTIMIZATION (Parameter Tuning), EVALUATION (Metric, Performance on Different Models)

5.1 Evaluation

For evaluation, we analyze the performance of all the models on the test data based on precision, recall and f1 score. We also create a confusion matrix for each of them to understand their performance better.

The goal of the F1-score measure is to balance between precision and recall, which is particularly useful in situations when dealing with unbalanced datasets. It is calculated as:

$$f1 = \frac{2 * (PRE * REC)}{(PRE + REC)}$$

Where PRE stands for precision and REC stands for recall. Precision is the number of true positives divided by the total number of positive predictions. It is depicted by the following formula

PRE = TP / (TP + FP)

The recall is the measure of True Positives being correctly identified by our model. It is computed using the following formula:

REC = TP / (TP + FN)

Where TP is the true positive. It represents the cases in which the classes were correctly predicted. FP is false positive, which represents the classes that have been incorrectly labelled as positive. FN stands for false-negative where positive samples have been incorrectly labelled as negative.

For multiclass prediction, we have calculated f1 by taking the weighted average. This takes the average of f1 scores for all classes by taking into consideration the support for each label. The support represents the total number of samples present for that label. The results for f1 score of all models is summarized in table 5.

Algorithm	Weighted F1 score
Random forest without resampling data	0.675
Random forest with resampled data	0.625
Decision tree	0.57
Ridge algorithm without resampling data	0.536
Ridge algorithm with resampled data	0.354
XGBoost algorithm	0.54
Extra Trees algorithm	0.664

Table 6. Evaluation of the performance of different classification algorithms

We also plot the confusion matrix for each algorithm to visualize the performance of the models. It is displayed in the figures below.

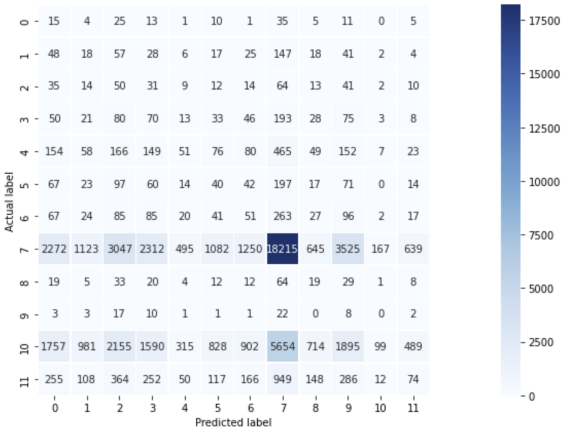

Fig 26. Confusion matrix for Ridge Algorithm

Fig 27. Confusion matrix for Extra trees

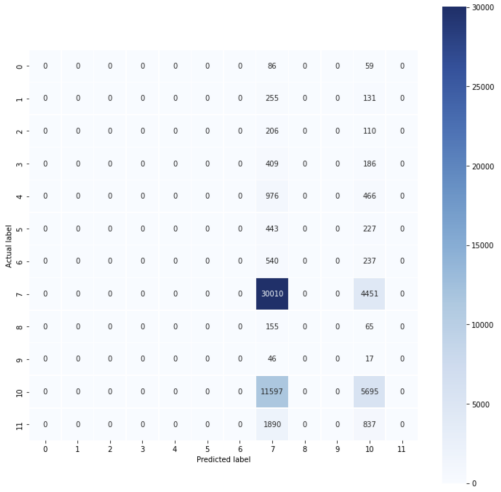

Fig 28. Confusion matrix for XGBoost

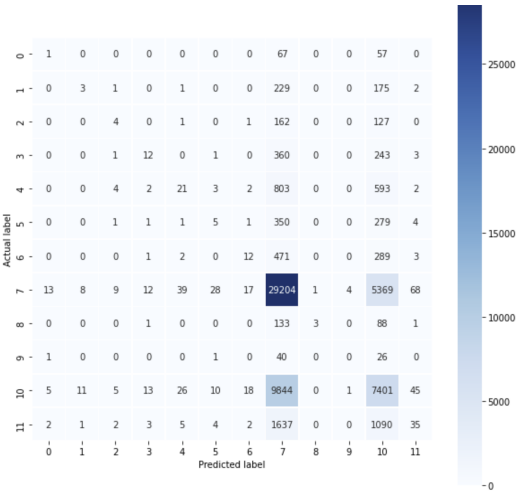

Fig 29. Confusion matrix for Decision trees

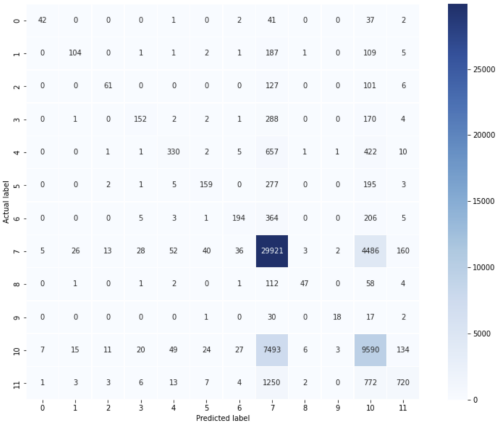

Fig 30. Confusion matrix for Random forest

In this experiment, we perform resampling of data using SMOTE and random under-sampling and compare the performance on unsampled and resampled data using random forest algorithm with a number of estimators = 400, and ridge regression algorithm, with alpha (penalty) set to 1. The performance of algorithms without resampling has been better in this experiment. A decision tree algorithm has been implemented with max depth = 12 and criterion = entropy. We have also analyzed the performance of the extra trees classifier with the number of

trees set to 50, and max features to consider for split as 4 and xgboost algorithm with estimators set to 10 trees, and a number of features while splitting set to 4. We find that the best performance is given by random forest and extra trees classifier, but we propose to utilize extra trees classifier as it is computationally faster.

6. FUTURE WORK

Most of the accommodation results are either NDF(No destination found) or the US. More data for individual users' web activities should be collected that will surely be useful to improve the accuracy of the prediction. Hyperparameter tuning of the various other different parameters can be performed. The session data and training data can be more utilized by extracting, improvising more features.

7. CONCLUSION

We propose to use an Extra trees classifier algorithm for such travel destination prediction tasks. This algorithm gives the best result on test data with an F1 score of 0.664. While its result is similar to the random forest algorithm in terms of F1 score and misclassification rate, the extra trees algorithm is computationally faster. The optimal performance of Extra trees is recorded when the number of trees in the ensemble are set to 50 and maximum features to look for while splitting is set to 4.

Data cleaning, preprocessing and feature selection greatly improve the performance of the model.

8. ACKNOWLEDGEMENT

A huge thanks to the University of Windsor, Ontario, Canada, the computer science department and to Dr. Robin Gras for the guidance.

9. REFERENCES

[1] Geurts, P., Ernst, D. & Wehenkel, L. "Extremely randomized trees". *Mach Learn* 63, 3–42, 2006.

[2] Hugo Ulfsson "Predicting Airbnb user's desired travel destinations", Stockholm, 2017.

[3]Srinivas Avireddy, Sathya Narayanan Ramamirtham, Sridhar Srinivasa Subramanian "Predicting Airbnb user destination using user demographic and session information", UC San Diego

[4] H. He, Y. Ma, "Imbalanced Learning: Foundations, Algorithms, and Applications", 2013.

[5] Airbnb Revenue and Usage Statistics (2021)

https://www.businessofapps.com/data/airbnb-statistics/

[6] Kaggle competition
https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings/overview

[7] Airbnb Revenue Image

https://infogram.com/airbnb-revenue-1hxr4zx9plzdo6y

[8] Kaggle competition

https://www.kaggle.com/manas13/airbnb-new-user-bookings

[9] Kaggle competition

https://www.kaggle.com/justk1/airbnb

[11] Kevin P. Murphy, Machine Learning: A Probabilistic Perspective, 2012.

[12]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html