

Airbnb New User Bookings

Udacity Capstone Project

Mohammed Shinoy
mshinoy@uwaterloo.ca

Abstract

Many businesses rely on customer data to provide better services. Predicting the first booking of a new customer on Airbnb based on previous user data containing demographics and sessions data is a similar procedure of high importance to an establishment. This information may be crucial to the company to provide relevant recommendation and support for the customer. This project examines the data provided by Kaggle and Airbnb to training such a machine learning model to predict user behavior. The creation methodology, feasibility, and results of such a predictive algorithm are also analyzed.

Keywords: Airbnb, supervised learning, sci-kit learn, seaborn, matplotlib, XGBoost, Kaggle.

1. Introduction

The customer is the focus of any business, keeping them happy and engaged will benefit both the customer and the business equally. Therefore it pays to know your customer's needs before they ask for it. As it helps the business to keep inventory, support, and recommendations on hand just before the customer asks for it. This concept is implemented by many technology companies and industries to provide recommendations to the consumer before they even think about it. Airbnb being a revolutionary technology company which has disrupted the short time rental space is no exception, It would be helpful to know the user's behavior in order to efficiently cater their products and services.

1.1. Problem Statement

Airbnb is a trusted community marketplace for people to list, discover, and book unique accommodations around the world online or from a mobile phone or tablet [1]. Utilizing prior user data effectively can help Airbnb to serve newcomers to the website in a better manner. User experience, customer retention, and customer satisfaction will shoot up.

There is a possibility that the needs of a new user can be predicted based on the demographic data of all the previous users. The problem statement is to predict the location a new user would book based on demographics and other data. User data contains the date, device type, geographical data etc. Based on this given data, create a machine learning model that will be able to accurately predict where a new user will book.

1.2. Datasets and Inputs

The dataset for this study is provided by Kaggle [2]. This dataset was used in a Kaggle competition in collaboration with Airbnb which can be obtained [here](#). The dataset contains 5 .csv files with information necessary to make a prediction. They are:

- countries.csv - summary statistics of destination countries in this dataset and their locations
- age_gender_bkts.csv - summary statistics of users age group, gender, country of destination
- train_users.csv - the training set of users
- test_users.csv - the test set of users
- sessions.csv - web sessions log for users [2]

1.3. Proposed Solution

Most probably there will be similarities in user behavior for people in the same demographics. These similarities will be helpful in creating an ML model using supervised learning to predict new user behavior. ML techniques such as SVM, Random Forest, AdaBoost etc along with Grid-SearchCV can be used to model an algorithm which can predict the destination a new user would book.

Prior to training, data cleaning and wrangling is to be performed on the dataset in order to extract features which will be beneficial to train an ML algorithm.

1.4. Evaluation Metric

As this is a multi-class classification problem, several evaluation methods can be used for this given ML model valuation. One of the most common metrics used for classification is Accuracy from sklearn.metrics.accuracy_score which compares the predicted values to the actual values and returns the ratio of Correctly Classified Values to Total Number of Predictions.

$$\text{accuracy_score} = \frac{\text{number of correctly predicted labels}}{\text{total number of labels}} \quad (1)$$

This metric though it is very simple, it has a major disadvantage. If the dataset is highly skewed, i.e data has about 90% x labels and the rest as y labels and the model predict x all the time. The accuracy will be 90% even though the model fails to recognize any y labels thereby rendering the model useless.

An alternative to this would be the F1 score from sklearn.metrics.f1_score. f1_score solves the problem of the dataset being skewed as it works on Precision and Recall.

Precision calculates the ratio between the number of labels correctly predicted to the total number of labels predicted. i.e if the algorithm is asked to classify white / black balls from a group of 20 balls which have 16 black and 4 white balls. If it classified 4 white balls and 6 black balls as white. Then the precision would be 4 out of the total 10 picked balls i.e $4/10 = 0.2$ which is a bad accuracy.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (2)$$

Now in recall, it is the ratio between the number of labels correctly predicted to the total number of labels of that particular type. i.e for the same example as above. If it classified 4 white

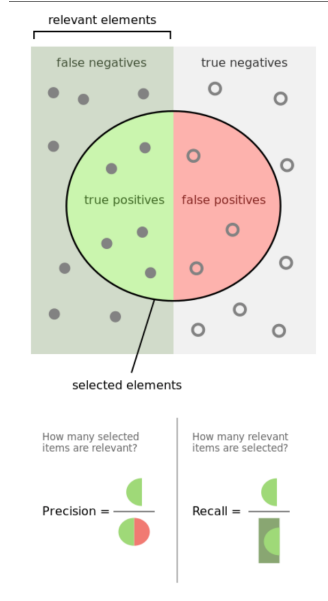


Figure 1: Precision - Recall [3]

balls and 6 black balls as white. Then the recall would be 4 out of the total 4 white balls. i.e $4/4 = 1$. This is a perfect recall.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (3)$$

A visual representation for precision and recall is provided in figure 1. The f1_score is based on this precision and recall. It is calculated as follows.

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

f1_score with weighted average will be used to evaluate the model.

The official evaluation of this project is done by Kaggle using Normalised Cumulative Discounted Gain (NDCG) [2]. The submission file to Kaggle can contain a list of maximum top 5 predicted destinations for every user in the test set.

$$DCG_k = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)}, \quad (4)$$

$$nDCG_k = \frac{DCG_k}{IDCG_k} \quad (5)$$

In the above equation k is 5 officially or the number of predictions per user the model provides. Where rel_i is the relevance of the result at position i. $IDCG_k$ is the maximum possible (ideal) DCG for a given set of queries. All NDCG calculations are relative values on the interval 0.0 to 1.0.

2. Data Analysis, Cleaning and Visualization

A total of 5 different datasets are provided.

- countries.csv - summary statistics of destination countries in this dataset and their locations
 - Number of rows : 10
 - Number of columns : 7
 - Might not be a highly relevant dataset as the data it contains can be dropped, especially distance won't affect the prediction as the users in question are all from the US and all countries are at the same distance for every user.
- age_gender_bkts.csv - summary statistics of users age group, gender, country of destination
 - Number of rows : 420
 - Number of columns : 5
 - Might not be relevant dataset as most of this data is obtained in the train_user.csv such as age, gender, country etc.
- train_users.csv - the training set of users
 - Length of rows : 213451 rows
 - Number of columns : 16
 - Highly relevant data as we are supposed to predict based on these columns. It contains user id, dates of account creation, first booking dates, gender, age, signup method, signup app, destination etc.
- test_users.csv - the test set of users
 - Number of rows : 62096
 - Number of columns : 15
 - test_users.csv has the same data columns as train_users.csv but not the destination and date first booking as they are new users. We have to predict the destination for this set.
- sessions.csv - web sessions log for users
 - Length of rows : 10567737
 - Number of columns : 6
 - Information regarding every user session with respect to the actions taken, device used and secs_elapsed being some of the important columns.

2.1. Training Data - Analysis

The total number of rows in the testing data is 213451 and number of columns is 16, which makes this a medium size dataset. The columns in the training data are:

```
['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']
```

The dataset needs to be cleaned and imputed as it contains a lot of NaN and unknown values. The basic process that we will follow:

1. Converting all the dates column to datetime format.
2. Changing the "-unknown-" values to -1.
3. Checking for bogus data in columns.
4. One hot encode the dataset and prepare the data for fitting into an ML algorithm.

The 'timestamp_first_active' has very large numbers as the timestamp is in a numerical format and not python datetime. This is changed to datetime format for easier computation and understanding. When dates are involved it is always better to extract information like year, month, day, week, etc. In this case the year, month and the day for each of the date columns i.e 'date_account_created' 'timestamp_first_active' 'date_first_booking' are made into dummy columns.

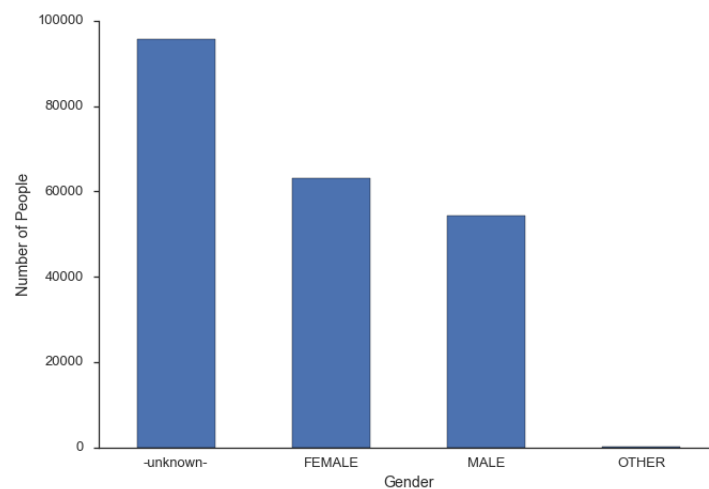


Figure 2: Gender

"-unknown-" values in the dataset are changed to np.nan. Columns 'gender' and 'first_browser' seems to have some "-unknown-" values. The figure 2 clearly shows that there are more than 95000 profiles with "-unknown-" gender. This has to be cleaned and replaced with NaN values. They might be "-unknown-" because people might not have specified it or maybe it was not a field that was necessary to be filled.

As seen in figure 3, the age column has a few outliers too, the maximum value is 2014 and there are a few values below 15 too, which are clearly bogus values. Users might have input the current year instead of the age by mistake or they withheld this information for privacy reasons. These have to be removed and replaced with NaN values. Even though the age is wrong, it would be nice to get the data concerning these individuals who would like to browse privately and know their behavior. Maybe these people might have similar booking tendencies. After cleaning this column, it follows a bell curve with high density over the age group of 20-50.

First browser column as shown in figure 5 has a lot of "-unknown-" data. Once they are converted it is observed that around 130000 users use Chrome, Safari and Firefox combined. There are about 20000 users whose first browser is unknown. This would suggest that most of the bookings are coming through desktop platforms.

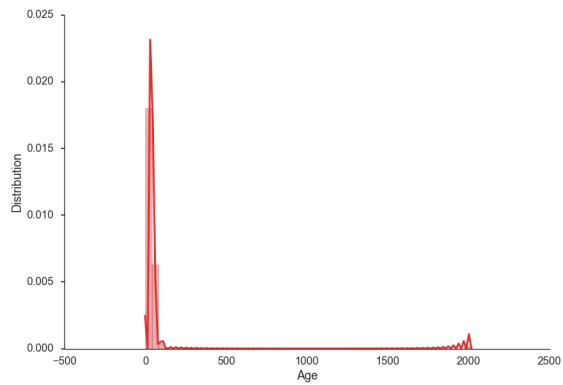


Figure 3: Age Column (Prior to cleaning)

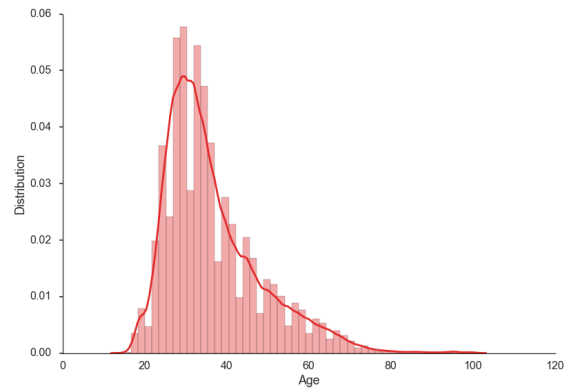


Figure 4: Age Column Cleaned

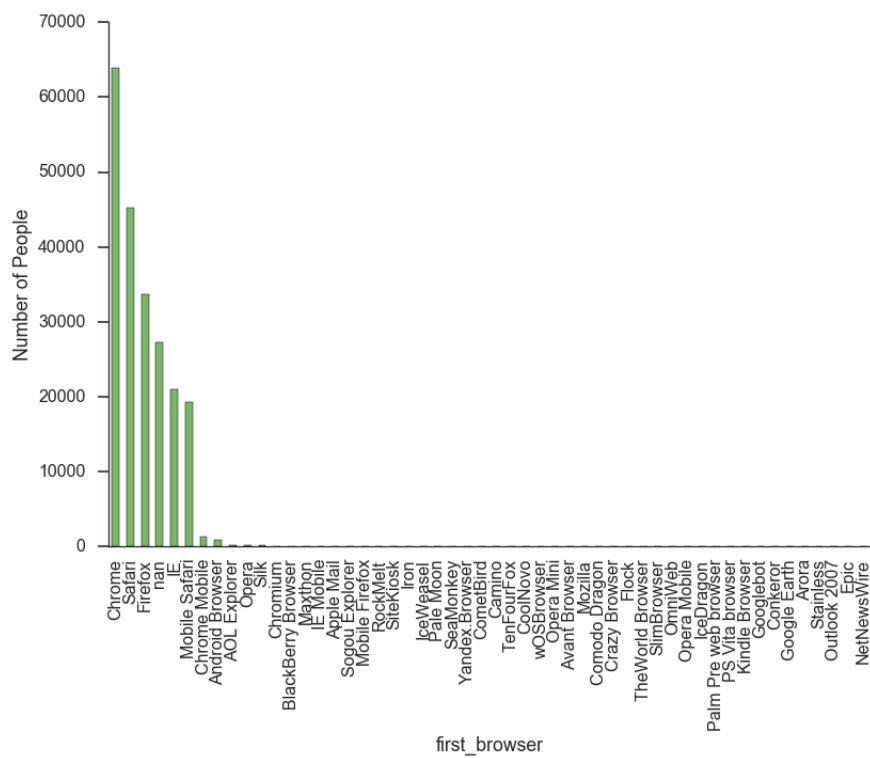


Figure 5: First Browser

One hot encoding is a process in which every unique categorical value in a column is converted to a feature column and for each row 0 or 1 is assigned based on whether that row has that feature. Once the basic cleaning is done, OHE is done.

After OHE, there seem to be NaN values only in age, the presence of NaN values will halt the ML training process. Therefore it is necessary to impute it. Imputing with the mean, median or mode of the distribution is a possible solution, but imputing it with any one of the above-mentioned methods will skew the data towards one end as it will drastically increase the number of users having that particular mean, median or mode age. Therefore, imputing it with -1 is a better choice.

2.2. Test Data - Analysis

Since the test data i.e "test_users.csv" also is in the same format as of Train data. Similar process is followed and data exploration and cleaning is done. There are only two columns missing from this dataset with respect to "train_users.csv" i.e country_destination and date_first_booking as these are only populated when a booking is done.

One interesting point to note is that the test data unlike train data contains only a subset of months.

```
test_data contains these years: [2014]
train_data contains these years: [2010 2011 2014 2012 2013]
test_data contains these months: [7 8 9]
train_data contains these months: [ 6  5  9 12  1  2  3  4 11  7  8 10]
```

Because of this there might be a chance that the prediction algorithm might heavily rely on the datasets from these months to predict and possibly cause over fitting if a suitable algorithm is not trained.

2.3. Sessions Data - Analysis

Sessions data contains every action a user took when he/she was on the airbnb website, what type of action it was, the device used and how many seconds was spent for each task. The features contained are:

```
{ 'user_id' 'action' 'action_type' 'action_detail' 'device_type'
  'secs_elapsed' }
```

The actions each user takes on the website will most likely correspond to a booking or no booking. The sec_elapsed feature is quite important as longer time spent on the website can mean that the person was interested in finding a booking and possibly lead to a booking. This dataset has the potential to contain data which pertains to whether a person will book or not. The sessions data has duplicate user_id's as this dataframe contains every session of a user.

The columns of high interest are action, action_type and action_detail. These have categorical data, therefore, these columns are to be one hot encoded. The secs_elapsed data is also important, therefore total secs_elapsed time for each user is calculated and a groupby is performed on the user_id which makes the user_id column in sessions_data unique.

After the transformation, we left join this sessions data with the training as well as testing data on user_id. A left join is explicitly selected because we do not want to lose users in the training set nor the test set. About 65% of training data does not have sessions data values. but for the testing dataset, only 0.68% has no sessions data. This difference suggests that maybe we should train the model only on the training data rows that have session_data to get a better accuracy. Therefore in order to test this theory, we can create two training datasets. One that does an inner join and the other with a left join on the sessions dataset.

The two training datasets are named training_data_merged and training_data_merged_inner. These training sets will be run on the same algorithm to check which one performs better initially and then tune it and discard the other. This would be a good experiment to know whether "more data" or "less data with good features" is better.

After OHE, It is to be noted that the training and testing data has different columns of features as there is a difference in the unique values in some columns between training_data and test_data. This needs to be handled.

2.4. Other Datasets and Summary

The countries.csv dataset just contains the 9 countries listed and contains latitude, longitude, distance from US, area in square km, the language in the country and a metric value which is the measure of how much each country's language is different from English.

Some of the use cases of this dataset that I can think of:

1. The distance of the destination from US, People tend to travel smaller distances more frequently. Thereby this could maybe be of potential value.
2. The language_levenshtein_distance can also be a factor, people might tend to go to places that speak in English for ease of travel or maybe there might be people who want to experience new cultures altogether.

The age_gender_bkts.csv dataset contains the amount of people who selected different destinations based on the age brackets and gender. This dataset might not contain data that is highly important as almost all the data is from 2015. We are trying to train on 2010 - 2014 and predict some months in 2014.

The countries.csv and age_gender_bkts.csv have been explored but the data found is not included into the training dataset as they will not add any features that are of high value. Already the training dataset and testing dataset has 661 and 636 features respectively. It is better not to add any more and increase the curse of dimensionality.

2.5. Code Challenges and Clarifications

In this section, some of the challenges of data cleaning, exploration and visualization is explored and how it was overcome.

One major coding challenge was to verify the working of the code. Especially when there a lot of inner joins, joins on user_id etc. Step by step verification is needed to know that the merging and concatenating that is being done is correct.

In line 77 of the part 1 notebook. Dummyfying the required columns in session data did not work as expected using sklearn.OneHotEncoder. The iPython Notebook Crashed and a restart of

kernel was suggested. The underlying reason for the problem is unknown. Therefore I wrote a `create_dummy` function to handle it for me.

First , The columns that has to one hot encoded is created which is

```
"to_dummy = ['action','action_type','action_detail']"
```

First `create_dummy` function will take the dataframe and column to one hot encode as the parameters. Then a new dataframe which contains the `user_id` and the column is created. Create a new column named "count" and assigns 1 to every row. Groupby the newdf by `user_id` and column pair, which will lead to counting up the values that were created in count column. Now pivot the newdf based on `user_id` , where columns are col and column values will be the count. This basically creates columns for all the unique values. Now just rename the columns based on the unique value and the initial column. For the three columns that are being one hot encoded. They are concatenated outside the function. One of the points that is to be noted is that , once you concatenate in that manner, the `user_id` becomes the index. so you have to re index these dataframe to easily join with the other dataframes

2.6. Benchmark Model

As this is a Kaggle competition, a good benchmark model would be the best Kaggle score for the test set, which comes in at .88697 NDCG score. If the model trained by this method has a score of 0.86 NDCG the model can be deemed useful and ready. The testing will be done on Kaggle and the results will be compared to best models performance. A personal goal would be to be in the top 20% ie above 0.88183 NDCG score of the Kaggle Private Leaderboard.

3. ML Algorithm Analysis and Implementation

At this stage there are 3 datasets as created above. The column to be predicted i.e `country_destination` is dropped to another dataframe as a validation set. As mentioned above the number of columns in the testing and training datasets are different. If training is done on 661 columns, at the time of predictions the algorithm will search for 661 columns to predict on. As our testing data only has 636 columns, an error will be thrown. So it is of primary importance to remove the columns which are mutually exclusive to each dataset. Compare the columns between training and testing datasets and drop the ones that are not part of each other.

3.1. Possible ML Algorithms

The objective of this machine learning model is to predict the `country_destination` that a new user will book based on previous data. Therefore this is a multi-classification problem and there are a plethora of classification algorithms that can be used. Some of the algorithms that have been considered for this problem are shown below, The hyper parameters used for the below classifier are default values as mentioned in sklearn, only the `random_state` is set at 27 so that we get the same seed which produces the same pseudo random numbers.

1. Random Forest

- A random forest is a collection of random decision trees. In which at each node you will randomly draw a subset of features and the decision tree will predict the classification. Then the same is done with several trees and bagged. This ensemble method will reduce overfitting and provide good classification. The bias-variance trade-off for this algorithm is good and therefore the possibility of overfitting is drastically reduced.

```
1 clf_A = RandomForestClassifier(n_estimators=10, criterion='gini',
    max_depth=None, min_samples_split=2, min_samples_leaf=1,
    min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
    =None, min_impurity_split=1e-07, bootstrap=True, oob_score=False,
    n_jobs=1, random_state=27, verbose=0, warm_start=False,
    class_weight=None)
2
```

Listing 1: RandomForestClassifier Classifier

2. Support Vector Machines (SVM)

- This is a good classification algorithm for high dimensional spaces. It uses a subset of training points in the decision function (called support vectors). It plots the X, Y values on the graph and then tries to find a hyperplane that separates the classes. If a hyperplane is not found, kernel trick is implemented and classification is made. One of the drawbacks is that if there are a higher number of features with respect to the number of samples, It is likely to give poor results. They also do not provide direct probability estimates which might be an issue for this problem as it is expected to make a maximum of top 5 possible predictions for every given user_id. Though this is a drawback, the probability estimates can be obtained indirectly.

The function used along with the parameters are given below. Its the default implementation in sklearn.svc.svm.

```
1 clf_B = SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
    shrinking=True, probability=False, tol=0.001, cache_size=200,
    class_weight=None, verbose=False, max_iter=-1,
    decision_function_shape=None, random_state=27)
2
```

Listing 2: SVM.SVC Classifier

3. Logistic Regression

- Multinomial Logistic Regression is based on plotting the independent variable X on the X-axis and then using a logit function or a progit function and cost function to classify in which class it will fall based on other features. [4]

LogisticRegression default values are for binary classification. This problem especially requires multi-nomial logistic regression. Therefore solver is changed to 'newton-cg' and multi_class to 'multinomial'.

```

1 clf_C = LogisticRegression(penalty='l2', dual=False, tol=0.0001, C
    =1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
    random_state=27, solver='newton-cg', max_iter=100, multi_class='
    multinomial', verbose=0, warm_start=False, n_jobs=1)
2

```

Listing 3: LogisticRegression Classifier

4. XGBOOST

- XGBoost is short for Extreme Gradient Boosting. This is based on Gradient boosted trees. Boosted trees are basically an ensemble of decision trees which are fit sequentially so that each new tree makes up for errors in the previously existing set of trees. The model is "boosted" by focusing new additions on correcting the residual errors of the last version of the model. Then you take an approximate step in the gradient direction by training a model to predict the gradient given the data. This algorithm is very popular for having really good accuracy and is quite common among Kaggle Competitions. [5] [6]

The default implementation of XGBoost is used at a standard low learning rate, just 2 n_estimators which is the number of sequential trees to be modeled and a low max_depth which controls the depth of each tree in order to control overfitting is used. The objective is set as 'multi:softprob' as this is a multi-class classification task which needs to output probabilities. The classifier parameters and results are given below.

```

1 xgb1 = XGBClassifier(
2     learning_rate = 0.1,
3     n_estimators=2,
4     max_depth=2,
5     min_child_weight=3,
6     gamma=0,
7     subsample=0.8,
8     colsample_bytree=0.8,
9     objective= 'multi:softprob',
10    nthread=-1,
11    scale_pos_weight=1,
12    seed=27)
13
14 xgb1_model = fit_and_results(xgb1, X_train, y_train, X_test, y_test)

```

Listing 4: xgb1_model Classifier

Based on the f1_score of these four models a decision will be made with respect to which algorithm is supposed to be tuned and finalized.

3.2. Stratified Shuffle Split

As this is a Kaggle Dataset, we have the liberty to train on the whole training data and then leave the testing and metrics to Kaggle. But, Kaggle only provides 5 submissions a day, also it is beneficial to test on a local machine with the metrics of our choosing. This helps to confirm that the model training has improved with each step.

StratifiedShuffleSplit is one of the best sklearn cross-validation methods. Even though this is a dataset that contains 213452 rows, Many of the destination labels are "NDF" and there is a possibility that an unbalanced dataset might fall into the training data. Therefore, It is always better to go with StratifiedShuffleSplit rather than train_test_split. [7]

2 different datasets for training data have been loaded. One which contains a left join between the given training set and session data, the other which has data from an inner join between the session data and training data. Usually it is true that more data will result in better results. But a good experiment would be to train the algorithm on both these datasets to predict on the test data, as the test data only has 0.68% of its rows without session data. Therefore it might perform better. Therefore, split both the training datasets i.e training_data_merged and training_data_merged_inner into testing and training set for Cross-Validation. The new datasets are :

- training_data_merged - Split into X_train, y_train, X_test, y_test
- training_data_merged_inner - Split into X_train_inner, y_train_inner, X_test_inner, y_test_inner

3.3. Preliminary Model Selection

The basic default model for the above-said algorithms is tested. Out of the four algorithms tested, SVC did not terminate training within any specified amount of time. On further exploration, it is noted in sklearn docs that the SVM.SVC implementation is based on libsvm. "The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to a dataset with more than a couple of 10000 samples" [8]. Therefore it makes sense that it is taking very long. SVM model fails to be a useful algorithm in our case.

The test results are given below:

```
**-----New Classifier-----**

Training a RandomForestClassifier using a training set size of 192105. . .
Trained model in 30.3826 seconds
Made predictions in 3.2560 seconds.
F1 score for training set: 0.9517.
Made predictions in 0.7060 seconds.
F1 SCORE FOR TEST SET: 0.5590.
-----

**-----New Classifier-----**

Training a LogisticRegression using a training set size of 192105. . .

Trained model in 1401.0322 seconds
Made predictions in 3.1293 seconds.
F1 score for training set: 0.5167.
Made predictions in 0.2494 seconds.
F1 SCORE FOR TEST SET: 0.5172.
-----
```

Out of these two, the RandomForestClassifier seems to be the better one in terms of speed as well as accuracy.

XGBoost is based on a Gradient Boosting Algorithm and is well known among Kagglers as a trained model usually works out to have really good accuracy when compared to the rest. It also provides a plenty of tuning factors which makes it a powerful ML training algorithm. Testing the default implementation of `xgboost.sklearn.XGBoostClassifier` on the `X_train`, `y_train`, `X_test`, `y_test`

The basic implementation of XGBoost at a standard low learning rate, just 2 `n_estimators` which is the number of sequential trees to be modeled and a low `max_depth` which controls the depth of each tree in order to control overfitting is used. The objective is set as `'multi:softprob'` as this is a multi-class classification task which needs to output probabilities. The classifier parameters and results are given below.

```
1 xgb1 = XGBClassifier(  
2     learning_rate = 0.1,  
3     n_estimators=2,  
4     max_depth=2,  
5     min_child_weight=3,  
6     gamma=0,  
7     subsample=0.8,  
8     colsample_bytree=0.8,  
9     objective= 'multi:softprob',  
10    nthread=-1,  
11    scale_pos_weight=1,  
12    seed=27)  
13  
14 xgb1_model = fit_and_results(xgb1, X_train, y_train, X_test, y_test)
```

Listing 5: xgb1_model Classifier

Fitting the model at 2017-02-08 03:21:13.421553

--- 1.76975533565 minutes ---

Fitting Completed

Model Report

Accuracy (Train) : 0.6325

Accuracy (Test) : 0.6364

F1 Score (Test): 0.592713

The `f1_score` increased by a value of 0.05 when compared to RandomForestClassifier, that is 55% to 59%. Out of the classifiers tested the lower than default implementation of XGBoost performs the best. Now the classifier can be improved by tuning the parameters.

3.4. XGBoost Classifier Tuning

XGBoost algorithm tuning is a tricky process. It has several parameters that can be tuned. Using a GridSearchCV is the best way to tune this classifier. It is not feasible to try out every single parameter as the training time will exponentially increase with added parameters. Heavy computation power is required for such level of tuning. Therefore in order to make this process easy it is suggested to use parameters that have been tried and tested by the ML community. The tuning parameters and methodology for this project follow some aspects from a blog write up in AnalyticsVidhya.com [9] by Arshay Jain.

It is noted that XGBoost performs well with higher number of n_estimators. The exact number of n_estimators that will work best for this project has to be found by GridSearchCV, but a higher value somewhere around 100-200 gives good results. In this case, 150 is selected as n_estimators. The default value for max_depth is usually 5. The model is run on both the training datasets which results in two classifiers i.e xgb2_model and xgb2_inner_model. Test results of xgb2_model (training_data_merged):

```
Fitting the model at 2017-02-08 03:29:45.786502
--- 180.176677748 minutes ---
Fitting Completed
```

Model Report

```
Accuracy (Train) : 0.657
Accuracy test (Test) : 0.6563
F1 Score (Test): 0.607926
```

Test results of xgb2_inner_model (training_data_merged_inner):

```
Fitting the model at 2017-02-09 18:46:07.141532
--- 62.1681851466 minutes ---
Fitting Completed
```

Model Report

```
Accuracy (Train) : 0.6067
Accuracy test (Test) : 0.5973
F1 Score (Test): 0.541131
```

From the above results, based on the f1_score, training on the larger dataset results in a better score. Therefore lets continue our tuning on xgb2_model.

3.5. GridSearchCV on the Model

There are many parameters which can be tuned to get a model with very good accuracy, but it requires a computer with high performance. The current setup takes around 1515 minutes to do a CV of 3 folds with 3 different parameters for max_depth.

A GridSearchCV for max_depth with three parameters were tested.

```

1 parameters1 = { 'max_depth':[5,7,10]}
2
3 xgb3 = XGBClassifier(
4     learning_rate =0.1,
5     n_estimators=150,
6     max_depth=5,
7     min_child_weight=3,
8     gamma=0,
9     subsample=0.8,
10    colsample_bytree=0.8,
11    objective= 'multi:softprob',
12    nthread=4,
13    scale_pos_weight=1,
14    seed=27)
15
16 xgb3_model, training_time_1 = trainer(xgb3, parameters1, X_train, y_train)

```

Listing 6: xgb3_model Classifier with GridSearchCV

After a day and half of training the model did not terminate and was not responding, the reason is unknown. This led to the requirement of a user interrupt. The partial results are seen and can interpreted, the results clearly show max_depth of 10 is the best f1_score among the three possibilities of 5,7 and 10.

```

Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV] max_depth=5 .....
[CV] ..... max_depth=5, score=0.602711 - 14.1s
[Parallel(n_jobs=1)]: Done 1 out of 1 |
elapsed: 126.9min remaining: 0.0s
[CV] max_depth=5 .....
[CV] ..... max_depth=5, score=0.603535 - 16.4s
[Parallel(n_jobs=1)]: Done 2 out of 2 |
elapsed: 254.9min remaining: 0.0s
[CV] max_depth=5 .....
[CV] ..... max_depth=5, score=0.601635 - 17.9s
[CV] max_depth=7 .....
[CV] ..... max_depth=7, score=0.603667 - 20.8s
[CV] max_depth=7 .....
[CV] ..... max_depth=7, score=0.604673 - 20.0s
[CV] max_depth=7 .....
[CV] ..... max_depth=7, score=0.602632 - 19.9s
[CV] max_depth=10 .....
[CV] ..... max_depth=10, score=0.603349 - 41.2s
[CV] max_depth=10 .....
[CV] ..... max_depth=10, score=0.603117 - 35.7s
[CV] max_depth=10 .....
[CV] ..... max_depth=10, score=0.603097 - 36.1s

```

```
[Parallel(n_jobs=1)]: Done    9 out of    9 | elapsed: 1515.0min finished
```

3.6. Final Model Parameters

With the results of the GridSearchCV and calculating the mean of the f1_scores it can be inferred that a max_depth of 10 will probably give the best Kaggle submission results. The final model parameters are given below:

```
1 xgb_final = XGBClassifier(  
2     learning_rate =0.1,  
3     n_estimators=150,  
4     max_depth=10,  
5     min_child_weight=3,  
6     gamma=0,  
7     subsample=0.8,  
8     colsample_bytree=0.8,  
9     objective= 'multi:softprob',  
10    nthread=4,  
11    scale_pos_weight=1,  
12    seed=27)  
13  
14 xgb_final_model = xgb_final.fit(training_data_merged , destination_merged)
```

Listing 7: xgb_final Classifier with GridSearchCV

3.7. Code Challenges and Clarifications

One of the changes that had to be made to the training and testing dataset is the maintain the same name and number of columns in both the datasets. The importance of this was mentioned earlier. In line 13,14 and 15 of the part 2 notebook , A list of columns in both the training and testing datasets are made. A list comprehension is used to make the list of columns exclusive to that dataset. This list is used to iterate through the respective column and drop the columns.

After fixing the final parameters of XGBoost and predicting on the dataset. Creating the submission file is crucial. In line 46 of the part two notebook. the predict_for_submission function takes in 5 arguments. Once the classifier is used to predict the probabilities, then the task of arranging those probabilities come into play. An argument sort has to be done on the prediction. Then a LabelEncoder is used to change all the probabilities to countries. After that, select the top n number of prediction by reversing and splicing the list. Then using the ids from id list the required dataframe is made.

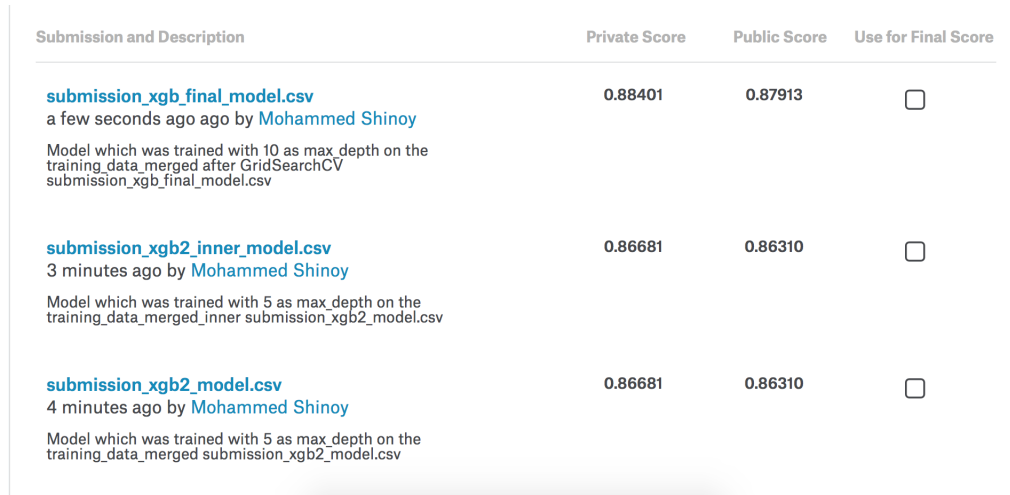
4. Results and Analysis

The results and evaluation given in the previous section is just based on f1_score. The kaggle leaderboard metric is Normalised Cumulative Discounted Gain (NDCG) [2].

The testing data for this project is testing_data_merged dataframe. All the above created models are used to predict the country destination for the test dataset. In order to submit the predictions, the predicted list is converted into the submission format. The submission format is a supposed to be a .csv file which contains a list of ids, predicted_country.

4.1. Submission Results

Once the list is created, the results are submitted to Kaggle, As shown in the figure 6 final_model has the best score of 0.87913.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_xgb_final_model.csv a few seconds ago ago by Mohammed Shinoy Model which was trained with 10 as max_depth on the training_data_merged after GridSearchCV submission_xgb_final_model.csv	0.88401	0.87913	<input type="checkbox"/>
submission_xgb2_inner_model.csv 3 minutes ago by Mohammed Shinoy Model which was trained with 5 as max_depth on the training_data_merged_inner submission_xgb2_model.csv	0.86681	0.86310	<input type="checkbox"/>
submission_xgb2_model.csv 4 minutes ago by Mohammed Shinoy Model which was trained with 5 as max_depth on the training_data_merged submission_xgb2_model.csv	0.86681	0.86310	<input type="checkbox"/>

Figure 6: Kaggle Submission Screenshot

4.2. Final Classifier Quality

One of the important points to notice when training a machine learning algorithm is whether it will scale, feasible for the scenario and performance is consistent at all times with varied data. All in all whether the classifier is robust.

For this project, StratifiedShuffleSplit was used and the reasoning for that was explained earlier. Due to this even we can be sure that the dataset used to train and test does not contain skewed datasets and will perform similarly even if different datasets are pushed into it. The GridSearchCV results seen above proves that even after doing a cross fold CV three times for every single parameters gives consistent results. This proves that the model does not overfit nor there will be any performance hit when it starts to see new datasets

This Final classifier predicted the first booking destination for 62016 users in just 14 sec. Though this classifier cannot be called into action every time a new person joins Airbnb. It is possible that the prediction algorithm can be run every few minutes for the new users that joined. Based on these predictions services can tweaked for every user.

5. Conclusion

The results obtained by the final model is satisfactory. This model is at position 318 out of 1400 submissions. The best score in Kaggle which is 0.88697. The xgb_final_model is just 0.00784 shy of the best score. with a score of 0.87913 on Kaggle. This makes the xgb_final_model classifier a good model for predicting the country a new user will book based on the demographic data and the sessions data.

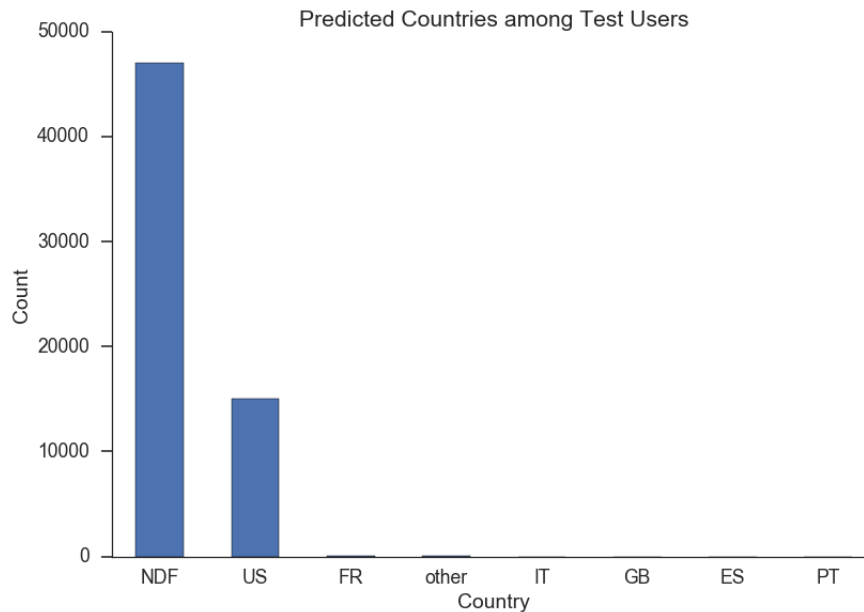


Figure 7: Prediction

The graph which shows the first predictions made by the classifier as seen in figure 7 shows us that from the 62095 values of new users 47022 people would opt for NDF and 14988 people select the US. This can be due to the fact that most users are in the US and since most of the people usually travel small distances, they seem to book for rentals in the same country.

Based on this model which predicts the top 5 places a person would book, Airbnb can provide rental listings in these countries. The fact that this same model trained with a city dataset can be used to predict cities rather than countries and provide users a much more personalized Airbnb listings. Such models will definitely drive up revenue and customer satisfaction.

5.1. Airbnb - New User Booking in a Nutshell

- Made sense of the data provided, Visualizations to detect outliers, bogus data etc. All datasets are studied and visualized for better understanding using matplotlib and seaborn.
- Cleaning the data based on the previous step. Selected the datasets of importance and cleaned them. Modified columns and other data fields by either dropping or imputing unwanted columns, outliers etc.
- Merge datasets based on common columns and cleaned and imputed unnecessary values. One hot encoded the datasets and later checked for consistency.
- Split the training data into test set and training set for cross-validation on local machine using StratifiedShuffleSplit
- Considered RandomForestClassifier, SVM, Logistic Regression, XGBoost to solve the problem. The best one i.e XGBoost is selected to tune
- Improve the model with cross-validation and GridSearchCV and test them with f1_score

- Computing limitations slowed down GridSearchCV —item Predicted on the test set and a submission is made to Kaggle.
- NDCG score of 0.87913 is obtained which puts the classifier at 318 out of 1400 teams.

5.2. Limitations faced and Improvements

Computing power was a major limitation. The training was done on a Macbook Pro with 8GB RAM and Dual Core Intel i5 processor. Due to this the training time for XGBoost model was very high when classifiers with `n_estimators` of 150 and `max_depth` of 5+ were run. I was unable to utilize GridSearchCV to the best possible extent. More parameters and tuning could be done with better computing power.

The Jupyter/Ipynb Notebook would not respond at times due to heavy RAM usage, which eventually led to force quits and restarts. This was highly inconvenient. Later on, I started saving the models trained to the file. In order to overcome these difficulties, Kaggle scripts seem promising. Models that take too long to train on a local machine can be done online. Maybe they have better computation prowess. Using AWS servers for ML is also a good idea as they have ML specific servers.

In addition to the hardware problems. The dataset exploration and cleaning took a considerable amount of time. The fact that training datasets and testing datasets needed the same columns values was known only after I had trained a few models. The training and testing datasets have to be manipulated in a similar fashion and consistency between them has to be checked after every major manipulation.

My Implementation model for this particular problem is good in terms of NDCG score but by no means the best. There might be underlying relationships between the other columns and datasets that I have not explored. Better data exploration might help uncover such relationships and thereby can be used to train a better model. All in all, this project was an amazing learning experience and now I have a clear base of support to tackle more Kaggle and real world datasets.

6. Tools and Libraries

- Tools : PyCharm, Jupyter Notebook , Kaggle
- Libraries: pandas, sci-kit learn, seaborn [10], matplotlib, XGBoost, sklearn.joblib.

References

- [1] Airbnb, "Airbnb About Us".
URL <https://www.airbnb.ca/about/about-us>
- [2] Kaggle, "Kaggle : Airbnb Recruiting New User Bookings" (2017).
URL <https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings/details/evaluation>
- [3] Walber, Precision - Recall (2017).
URL <https://commons.wikimedia.org/w/index.php?curid=36926283>
- [4] Multinomial Logistic Regression (2017).
URL https://en.wikipedia.org/wiki/Multinomial_logistic_regression
- [5] T. Chen, Introduction to Boosted Trees (2017).
URL <http://xgboost.readthedocs.io/en/latest/model.html>

- [6] U. of Washington, [Introduction to Boosted Trees](#) (2017).
URL <http://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf>
- [7] S. Exchange, [Stratified Cross Validation](#) (2017).
URL <http://stats.stackexchange.com/questions/117643/why-use-stratified-cross-validation-why-does>
- [8] scikit learn, [sklearn.SVM.SVC](#) (2017).
URL <http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- [9] A. Jain, [Complete Guide to Parameter Tuning in XGBoost \(with codes in Python\)](#) (2017).
URL <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes/#comment-116830>
- [10] M. Waskom, ["seaborn"](#) (2017).
URL <http://seaborn.pydata.org>