Predicting Airbnb User Bookings with Supervised Machine Learning
Capstone #1 Final Report

Anthony Johnson

Introduction

Airbnb is an online tool for connecting homeowners (hosts) with people looking for a place to stay (users). More successful connections, or bookings, means more revenue, and so it is in Airbnb's interest to make the user experience as smooth as possible. Airbnb thus uses data to optimize user experience and increase the number of bookings.

Airbnb would likely benefit from a predictive model of how *soon* a user will make their first booking, based on typical user actions, profiles, etc. That is, there could be a strong correlation between specific user profiles/actions and the likelihood that the user will make a booking or not.

Such a predictive model would be very useful to Airbnb in order to pick out which users need a little extra incentive to get them to make their first booking. Airbnb could, for instance, provide some discount to incentivize quicker action, but only to a select population. Applying such a discount to everyone would be unnecessarily expensive.

In this report, Airbnb user data was collected, cleaned, explored, and ultimately used to train a random forest classifier to predict if a user will make a booking or not. The final metric of interest, the ROC AUC, was calculated to be 0.76.

Data Source

The data was acquired from a Kaggle competition from 2015. The data contains information about users' signup and booking information, as well as detailed logs of user actions. The data is relatively clean, though not perfect. Here is a summary of the tables:

1) Train_users.csv - user data for training the model
2) Test_users.csv - user data for testing the model
3) Sessions.csv - detailed logs of user actions
4) Countries.csv - summary statistics of destination countries
5) Age_gender_bkts.csv - summary statistics of users age, gender, country

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

Each of these CSV files were read into Jupyter using the Pandas read_csv() function, creating a Pandas DataFrame for each CSV file.

Approach

This problem involves digging into the user data (train_users.csv) and activity logs (sessions.csv) and training models to predict how soon a booking will be made. The train_users table contains dates for account creation and dates of first booking. The date of first booking minus the date of account of creation is thus the main variable we want to model. Here are some questions we want to address, to help us tease out useful information:

1) Are there any correlations between user profile information and first booking date, e.g. gender, age, signup_method, etc? Looking for correlations here would be the simplest place to start. Perhaps, for instance, men of age 50-55 with a language preference of Czech or Japanese are unlikely to make bookings. This could indicate, among other things, that Airbnb needs more customer support in those languages.
2) Are there specific user actions that indicate that a user will not make a booking?
3) Are there specific strings of user actions that indicate that a user will not make a booking? This is where some type of machine learning model may come in handy, since it would be challenging to "manually" check all the possible user action combinations.
4) Can we train classifiers to distinguish between users who will make a booking from users who will not? Or should we use regression to predict exactly when the user may make a booking?

Once the data is fully cleaned and explored, we will start with logistic regression classification, and then move to random forest classifiers with advanced hyperparameter optimization.

Data Assessment

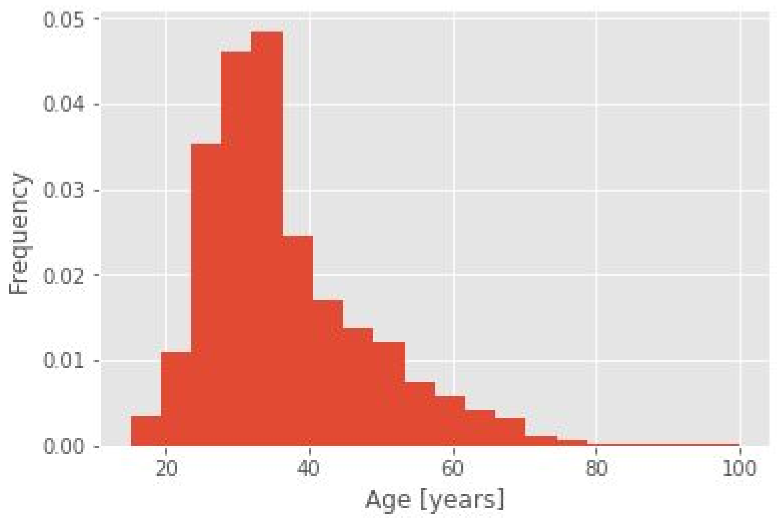
The initial assessment of the DataFrames (df) involved calling the following functions.

1) **df.head()**, to get an idea of the data content, column names, etc.
2) **df.info()**, to assess if the data types for each column were appropriate and to get a count of the number of null entries. From here, I discovered that the columns that should be DateTime objects were not all of the correct type. In the train_users and test_users tables, the "date_first_booking" and "date_account_created" columns were object type, and "timestamp_first_active" was integer type. Furthermore, about ⅔ of the "date_first_booking" entries were null, which makes sense for this data since not all of the registered users ended up making a booking. About half of the "age" entries were null, and a handful of the "first_affiliate_tracked" entries were null.
3) **df.describe()**, to assess the columns with float or int types, looking for outliers. In the train_users and test_users tables, we see statistics for "timestamp_first_active", which shouldn't be integer type. In addition, we see "age" statistics, and there a number of red flags here. First, the standard deviation is 156, which makes no sense for an age distribution unless there are a number of false outliers skewing it. The maximum age is 2014, which of course explains the high standard deviation. It seems likely that a user accidentally put the year in for their age. We'll have to check how often this occurred. Additionally, there were "age" entries of 1, which also need to be fixed.

Data Cleaning

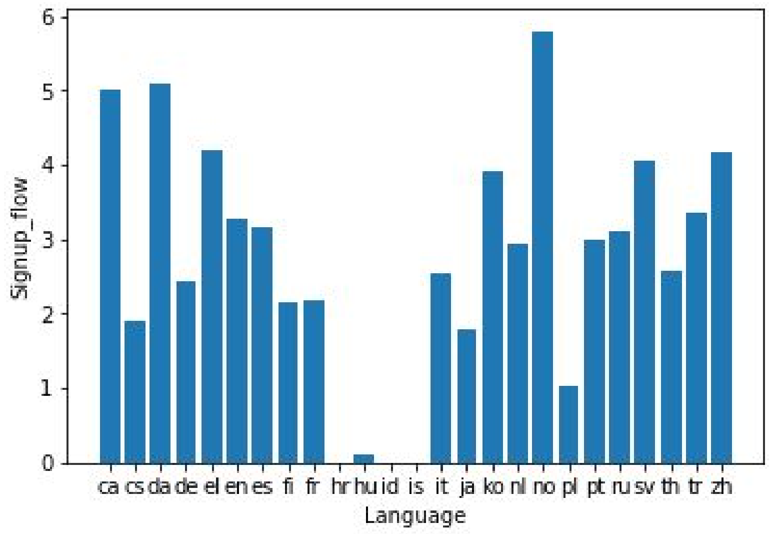
From my assessment, it seems that train_users and test_users are the only tables that need a bit of cleaning. It's possible that more cleaning will need to be done later.

1) First I needed to convert any datetime columns into datetime objects. The "date_account_created" and "date_first_booking" columns were the most straightforward. I called the Panda's to_datetime function on those columns, using the format='%Y-%m-%d' setting. The "timestamp_first_active" column was then converted by first converting to a string using astype(str) and then inputting into the to_datetime function, with the setting format='%Y%m%d%H%M%S'.
2) Next, the "age" column needed to be fixed. The simplest way to do it was to convert any number above 100 or below 10 into the median age. It's unclear if this is the best method, but an alternative could be to replace those false entries with NaN. I initially tried setting the age threshold to 120, since it's conceivable there are actual users over 100. However, upon looking at the value_counts() for people over 100, I noticed there is a bizarrely large number of entries with value 105 and 110. There were also a number of entries that could have been birth years, e.g. 1952, 1938, etc. If this was the case, I could extract their age. But since that's unclear, I just set all of the values over 100 to the median age.

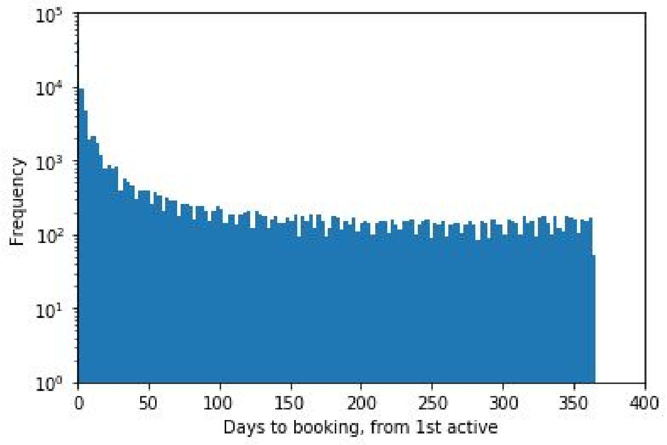


Data Exploration

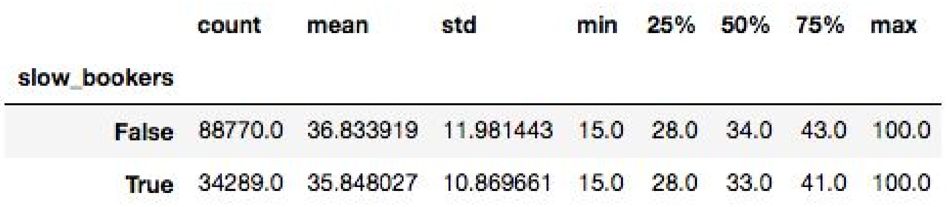
1) I first applied "groupby" functions to the train_users data, to assess the differences in "age" and "signup_flow" between different groups.
Example: train_users.groupby('signup_method').describe()
    a) Between "basic" and "facebook" signup_methods, there is a 1.7 year age difference as well as a slight difference in mean signup_flow.
    b) There were also differences in signup_flow between different browsers and different languages. I made some bar plots to illustrate some of the differences.



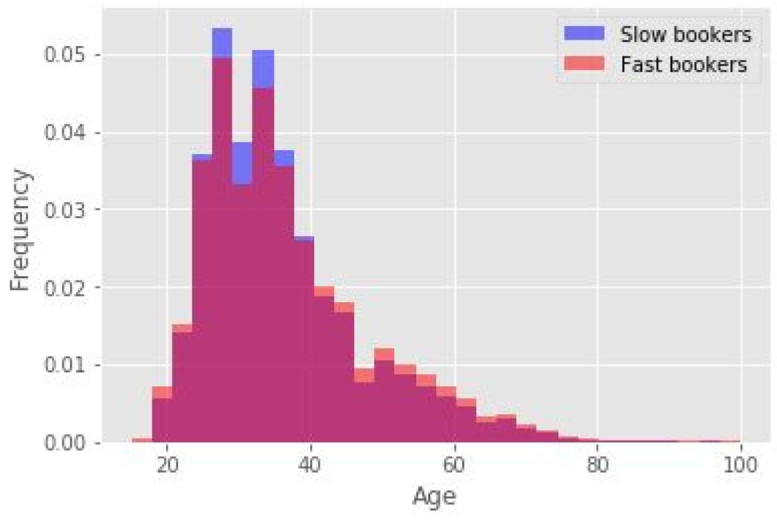
2) In the Sessions table, I grouped by user and action to get the total number of specific actions for each user.
   Example: sessions.groupby('user_id').count()
3) Next, I created columns representing the amount of time between account creation and first booking, as well as between time_first_active and first booking. Since we're interested in predicting this time difference, it is crucial that we understand any trends related to it. Of the people who make a booking, about 50% do it within the first 2 days from when they start looking.



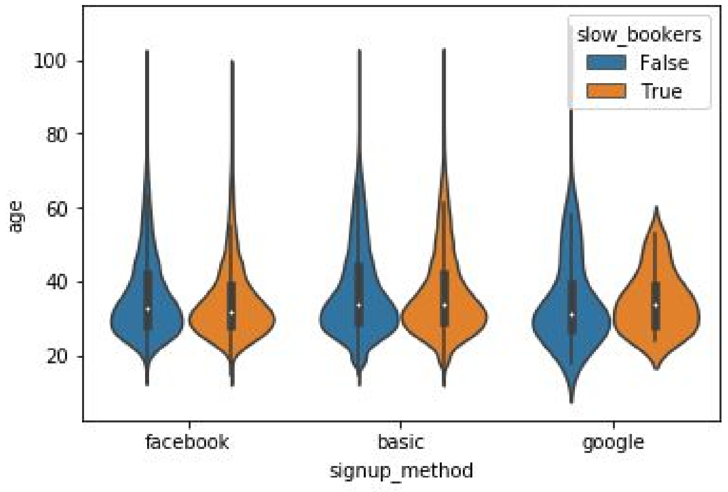
4) From this data, it became straightforward to compare different metrics between the "slow" bookers and "fast" bookers, like age for instance. Generally, fast bookers appear to be a slightly older population than the slow bookers, as seen in the data and histogram below.



slow_bookers	count	mean	std	min	25%	50%	75%	max
False	88770.0	36.833919	11.981443	15.0	28.0	34.0	43.0	100.0
True	34289.0	35.848027	10.869661	15.0	28.0	33.0	41.0	100.0



5) I then made a number of violin plots to compare different metrics. Here's an example.



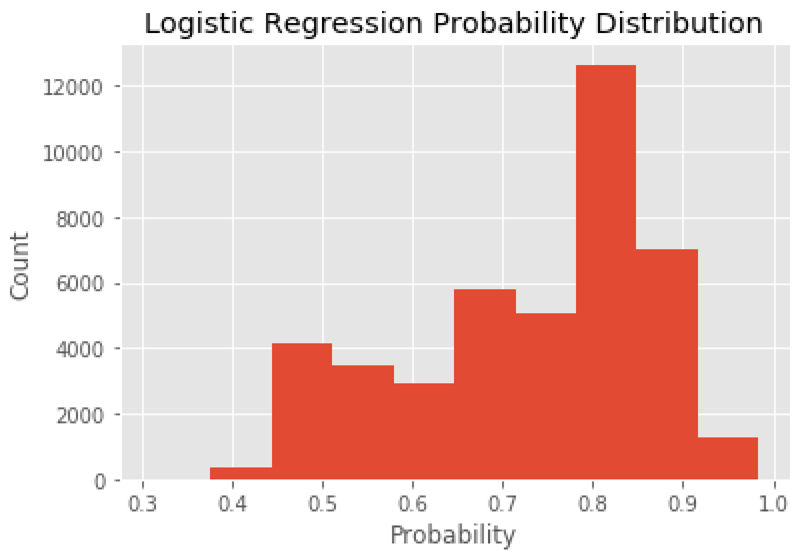
6) Finally, I left-joined the grouped-by-user sessions table with the train_users table, such that all of the users now have detailed actions along with their important date-time information for account creation and first booking.

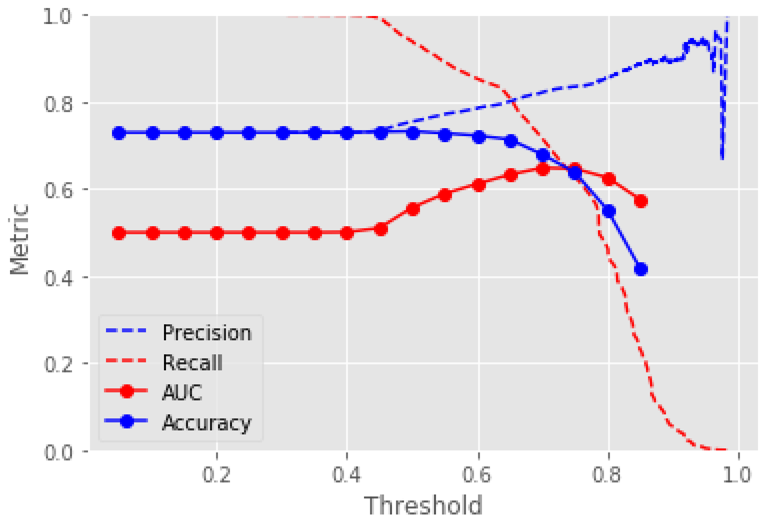
Data Modeling

We want to be able to distinguish "slow" bookers from people who never made a booking, so I first removed all users who made a "fast" booking. This "fast" threshold was arbitrarily set to 2 days.

To further prepare the data for modeling, the categorical variables (such as signup_method) were converted such that each possible value had its own column, with a one or zero as an entry. This was done using one-hot-encoding.

1) **Logistic Regression Classifier**
I first tried using a Logistic Regression Classifier from Scikit-learn. With a default Logistic Regression Classifier, I got an ROC AUC of 0.55. Because the classes are not perfectly balanced, the area under the ROC curve (ROC AUC) is a better metric to use than accuracy. Below are plots of the predicted probability distribution, as well as various metrics vs. the probability threshold.





Based on the plot of ROC AUC and accuracy vs. threshold, above, it is clear that at a threshold of around 0.45 or below, there is overfitting. The maximum ROC AUC occurs around a threshold of 0.75.

Furthermore, the most extreme coefficient values are displayed below, so we can see which variables have the greatest effect on the booking speed. The most important variables seem to be gender, affiliate_channel, signup_method, and language.

Variable	**Coefficient**
affiliate_provider_3	-0.486442
language_8	-0.406453
affiliate_provider_9	0.402620
signup_app_4	0.410161
first_browser_8	0.560238
language_5	0.630218
signup_method_1	0.757938
affiliate_channel_5	0.900174
gender_1	1.344208

To improve the model, we can try setting the threshold to 0.75, as well as by carrying out 5-fold cross-validation in order to help avoid overfitting. With these changes, we get a new ROC AUC of 0.65. This is clearly a step in the right direction, but we can possibly do better by moving to a random forest.

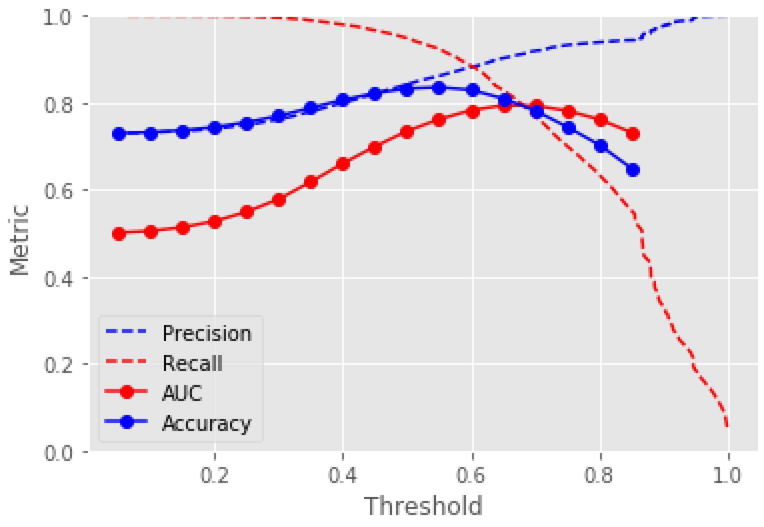
2) **Random Forest Classifier**
Next, I used a Random Forest Classifier from Scikit-learn. Initially, I tried 100 estimators, with otherwise default settings. This gave us an ROC AUC of 0.66, a slight improvement. To optimize the hyperparameters, I used a RandomizedSearchCV, which applies pseudo-random values to the specified hyperparameters in param_grid (below).

```
param_grid = {
        "max_depth":            [3, None],
        "max_features":         randint(1, 11),
        "min_samples_split":    randint(2, 11),
        "bootstrap":            [True, False],
```

"**n_estimators**":	randint(50,200),
"**criterion**":	["gini", "entropy"]}

After running the RandomizedSearchCV, the best model had an ROC AUC of 0.72, a significant improvement. Below are the hyperparameters for that best model, along with a plot of various metrics vs. threshold. The AUC value of 0.72 occurred at a threshold of 0.5, but the maximum AUC of 0.80 occurred around a threshold of 0.7, as seen in the plot below. The precision and recall at this threshold were about 0.92 and 0.74, respectively.

"**max_depth**" =	None
"**max_features**" =	9
"**min_samples_split**" =	9
"**bootstrap**" =	True
"**n_estimators**" =	189
"**criterion**" =	“gini”



A Monte Carlo algorithm was then implemented to further optimize the hyperparameters in the random forest classifier. The initial hyperparameters were taken from the RandomizedSearchCV. During each iteration, each hyperparameter was altered slightly and a new ROC AUC score was calculated with the function cv_score_prob. If the AUC score increased, the hyperparameters were accepted as the new best. Otherwise, they were reverted back to the previous best. The optimal hyperparameters changed slightly, and are shown below:

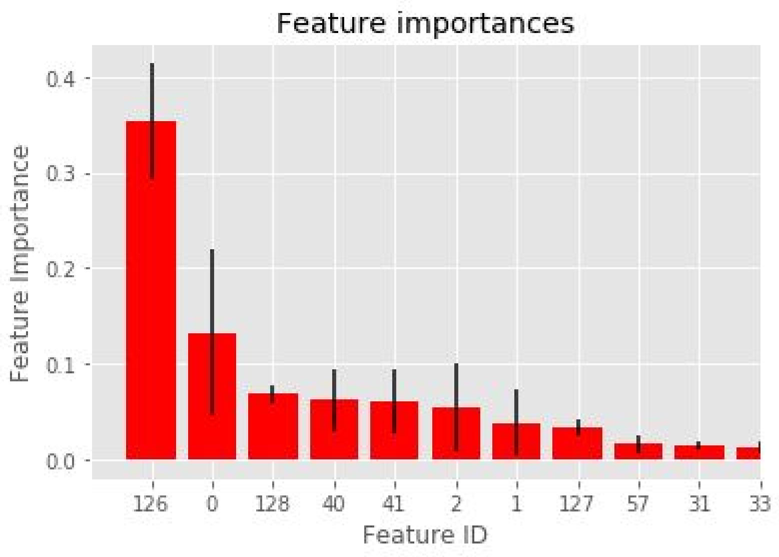
"**max_depth**" = 22
"**max_features**" = 16
"**min_samples_split**" = 9

"**min_samples_split**" = 3
"**bootstrap**" = True
"**n_estimators**": = 163
“Threshold”: 0.73

The final random forest model had a training AUC of 0.80 and a testing AUC of 0.76. From the feature rankings, below, we see that age and gender are the biggest predictors, followed by "action" and "signup_method." Generally, people who never made a booking tended to be slightly older than those who did.

Feature ranking:

1. feature 126: **age** (0.352915)
2. feature 0: **gender_1** (0.132394)
3. feature 128: **action** (0.067985)
4. feature 40: signup_method_1 (0.061338)
5. feature 41: signup_method_2 (0.060057)
6. feature 2: gender_3 (0.053979)
7. feature 1: gender_2 (0.038183)
8. feature 127: signup_flow (0.032856)
9. feature 57: first_browser_5 (0.015301)
10. feature 31: first_affiliate_tracked_1 (0.013746)

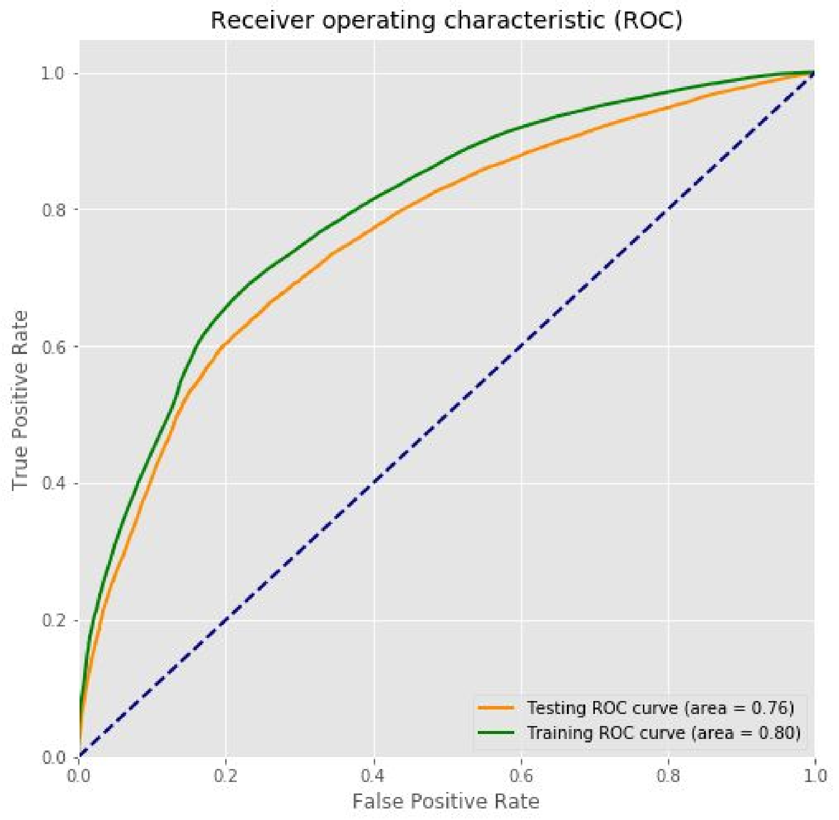


Below are the confusion matrices for the training and testing sets.

TRAINING SET	PREDICTED - Slow booking	PREDICTED - No booking
ACTUAL - Slow booking	27903	6766
ACTUAL - No booking	32655	60740

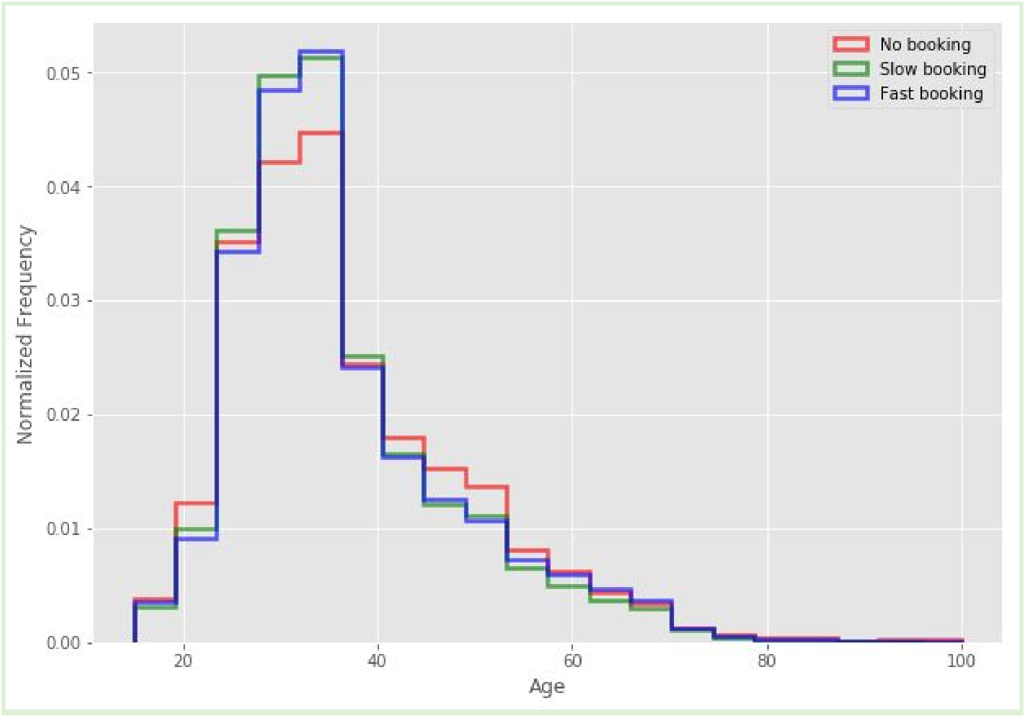
TESTING SET	PREDICTED - Slow booking	PREDICTED - No booking
ACTUAL - Slow booking	8876	2664
ACTUAL - No booking	11414	19734

Below is the ROC curve for the final random forest classifier. The training and testing AUC values are 0.80 and 0.76, respectively.



In the table and histogram below, we see the average age for slow and fast bookers, and for people who never made a booking.

	Average Age
Slow bookers	35.9
Fast bookers	36.5
No booking	37.1



Discussion and Future Work

An ROC AUC value of 0.76 is significantly better than random chance, but errors will still be made. For this application, an AUC of 1.0 would mean that we know exactly who will not make a booking. This group of users could then be sent coupons to incentivize a booking. Low accuracy means that coupons/promotions may be sent to people who are planning to make a booking anyway.

In the final model, we see that age is the most important variable for predicting speed of booking. Specifically, slow bookers tend to be slightly younger than fast bookers. It could be useful to dig into this further, to better understand why this is the case.

In this report, the user actions were simply counted. More work could be done to better model how specific searches, for instance, lead to bookings. Are there specific actions, or strings of actions, that correspond with slow vs. fast bookings?

Fine tuning of the random forest classifier could be done with new users as more data is collected. Other models such as gradient boosting might also provide accurate predictions of user bookings.

Furthermore, instead of treating this as a classification problem, regression models could be used to predict exactly how far into the future each user will make a booking. Because of the scatter in the data, it was simpler in this case to use classification.

Summary & Conclusions

Overall, the best model was a random forest classifier with an AUC of 0.76. This model can be used to predict which users will likely make a booking in the future, separating them from the users which will never make a booking. User age turned out to be the most important variable for predicting how soon a user will make a booking.
Generally, as seen in the averages above, people who made fast bookings tended to be older than people who made slow bookings (36.5 vs. 35.9 years). And people who never made a booking tended to be older still, at 37.1 years. Airbnb should place a high priority on collecting age data whenever possible.

Github link to source code:
https://github.com/Aejohnso/Springboard/tree/master/Capstone_1_Project