Airbnb's New User Bookings Kaggle Competition

Lisa Oshita

April 16, 2018

1 About the Competition

Founded in 2008, Airbnb is an online accommodation marketplace featuring millions of houses, rooms and apartments for rent or lease in over 200 countries. As part of a recruiting competition, Airbnb partnered with Kaggle to host a data science competition starting in November 2015 and ending in February 2016. The task of this competition was to build a model to accurately predict where new Airbnb users will make their first bookings. Participants were given access to six data sets containing information about the users and the 12 possible destinations to predict: Australia, Canada, France, Germany, Italy, Netherlands, Portugal, Spain, United Kingdom, US, other and no destination found (NDF). No destination found indicates that a user did not make a booking. Other indicates that a booking was made to a country not listed. This paper describes the process used to develop several models to predict booking destinations, with the goal of exploring and implementing machine learning concepts.

2 XGBoost, Random Forest, Stacked Models

This paper explores three different models: Extreme gradient boosting (XGBoost), random forests, and stacked models.

XGBoost algorithms are a fairly new method of supervised ensemble learning that performs consistently better than single-algorithm models. It is a form of gradient boosting that introduces a different, more formal method of regularization to prevent overfitting—enabling it to outperform other models. Additionally, XGBoost algorithms are parallelizable, allowing it to fully utilize the power of computers, which effectively decreases computation time [1]. As XGBoosts have been used to place highly in Kaggle competitions, including this Airbnb competition, this was the first model implemented.

Random forest models are another powerful form of ensemble modeling frequently used in Kaggle competitions.

Stacking, also called meta ensembling, is a technique used to combine information from individual predictive models to create a new model. Because stacked models are able to correct and build upon the performance of those base models, they usually achieve better results. This technique is also frequently

used in Kaggle competitions and was used by many of the top participants of this competition.

3 About the Data

Of six available data sets, two were utilized: train_users and sessions. The sessions data, with 1,048,575 rows and 12,994 unique users, contains information regarding user's web session activities (e.g. what pages were viewed on Airbnb's site, what actions were taken by the user). To minimize computation time, 10% of rows from each unique user was randomly sampled. This new sampled data, containing 104,424 rows, was used for all following analyses.

The train_users data, with 213,415 rows and 16 original columns, contains user information from 2010 to 2014. Information includes features like age, gender, date of account creation, sign up method (Facebook, Google...), and language preference.

Destination	**Percentage of the data (%)**
NDF	58.35
US	29.22
other	4.73
FR	2.35
IT	1.33
GB	1.09
ES	1.05
CA	0.67
DE	0.50
NL	0.36
AU	0.25
PT	0.10

Table 1: Percentage of data each destination accounts for

Exploring the feature to predict, booking destinations, revealed a significant issue. Table 1 highlights how extremely imbalanced the classes are. Together, NDF and US account for almost 90% of the entire data, indicating that a majority of users either have yet to make a booking, or book to locations in the US. The remaining 10, under-represented destinations make up less than 5% of the data each. The extremity of this imbalance presented several challenges for the machine learning algorithms explored in this paper.

4 Feature Engineering

From the train_users data, 17 features were created. Date features were pulled apart into month, year, day of the week, and season features. The differences in days between date features (days between the date an account was created and

the date of first booking, and days between the date a user was first active and the date of first booking) were also calculated. Missing and erroneous values in the gender feature were cleaned and recoded. The age feature contained several issues. 87,990 users (41% of users) did not enter an age, while 2,345 users (1% of users) entered an age of greater than 100. These values were recoded, while missing values were imputed using an XGBoost to predict those ages, based on ages available in the data.

For the sessions data, rows were aggregated by user ID and features counting the number of unique levels of each categorical feature were created. From secs_elapsed, mean, median, standard deviation, minimum and maximums were calculated for each user. A total of 320 features were created and joined by user ID to the train_users data.

Original Features	Sign up method (Facebook, basic, Google)
	Sign up flow (17 levels: 1, 2, 3...)
	Language preference (25 levels)
	Affiliate channel (8 levels: direct, re-marketing, API...)
	Affiliate provider (18 levels: craigslist, Bing, email-marketing...)
	First affiliate tracked (8 levels: un-tracked, linked, local ops...)
	Sign up app (Web, Moweb, iOS, Android)
	First device type (9 levels: Mac Desktop, iPad, iPhone...)
	First browser (52 levels: Chrome, Firefox, Safari...)
Derived Features	Year, month, day of the week, season features of date account created, date first active, date first booking
	Days between date account created and date of first booking
	Days between date first active and date of first booking
	Age
	Gender
	Count features created from the sessions data (314 features: number of times a user viewed recent reservations, number of times a user viewed similar listings...)
	Mean, median, standard deviation, minimum, maximum of seconds elapsed for each user's web activity

Table 2: Original and Derived Features

One-hot encoding was used to convert categorical features to machine learning compatible forms. Essentially, a Boolean column was generated for each level of a categorical feature. Continuous features were left as is. After one-hot encoding, there were a total of 596 features. Table 2 displays all features, both original and derived, that were included in the training data for model building.

5 Model Building and Results

Although the methods used for model building were circular and iterative processes, rather than linear. the following describes the general processes used to develop the XGBoost, random forest, and stacked models.

The full data was partitioned into one training set, containing 149,422 rows (70% of the full data), and one test set containing 64,029 rows. All model building was performed on just the training set. For both XGBoost and random forest models, five fold cross-validation was performed, including all 596 features. Both models achieved 87% classification accuracy, as well as a Normalized Discounted Cumulative Gain (NDCG) score (the metric used in the actual competition) of 0.92. However, both models only made predictions for NDF and the US.

Examining feature importance for each model showed that not all features were necessary. So, the models were refit to just the 200 most important features and cross-validation was performed again. While accuracy and NDCG scores remained the same, computation time decreased tremendously. All following models were built on just those 200 features.

Feature	**Mean decrease accuracy**	**Mean decrease Gini**
firstbook_sn.1	0.06	7655.61
firstbook_wkd.1	0.05	6501.57
firstbook_y.1	0.05	6491.36
lag_acb_binNA	0.05	5597.83
lag_acb_bin.1..365.	0.03	4631.19
firstbook_m.1	0.04	4493.25
lag_bts_bin.1..1369.	0.03	4272.94
lag_bts_binNA	0.04	4197.65
firstbook_snspring	0.01	1932.55
firstbook_y2013	0.01	1700.19
firstbook_y2014	0.01	1163.31
age_clean	0.00	949.90
lag_acb_bin0	0.01	786.03
lag_bts_bin0	0.01	752.03
firstbook_snsummer	0.01	614.88
firstbook_wkdMonday	0.00	599.84
firstbook_snwinter	0.00	506.20
firstbook_m05	0.00	480.97
firstbook_wkdSaturday	0.00	411.84
firstbook_wkdThursday	0.00	376.21

Table 3: Feature importance of the random forest model

Tables 3 and 4 show feature importance output of the top 20 most important features from the random forest and XGBoost models, respectively. Tables are ordered by decreasing importance. Note that different columns are displayed

because feature importance for random forest and XGBoost models output differently. Also note that 0s in the columns are just extremely small decimals. For Table 3, mean decrease accuracy is a measure based on the hypothesis that if a feature is not important, than rearranging values of that feature should not harm or decrease accuracy. Mean decrease gini refers to the total decrease in node impurities that result from splitting on a variable, averaged across all trees. For both, higher values indicate greater feature importance. For Table 4, gain refers to the increase in accuracy achieved by splitting on a feature. Cover refers to the number of observations that were assigned a class after splitting on the feature. For both gain and cover, higher values indicate importance. Interestingly, Table 4 shows that only one feature is accurately contributing to classification with a gain score of 0.96, and that is whether or not a user made their first booking.

Feature	**Gain**	**Cover**
firstbook_y.1	0.96	0.23
age_clean	0.01	0.08
signup_flow	0.00	0.02
firstbook_snspring	0.00	0.02
affiliate_channelother	0.00	0.03
gender_cleanFEMALE	0.00	0.00
gender_cleanMALE	0.00	0.01
age_bucket.1	0.00	0.01
lag_acb_bin.1..365.	0.00	0.01
firstbook_y2014	0.00	0.01
first_device_typeMac.Desktop	0.00	0.00
first_device_typeWindows.Desktop	0.00	0.01
first_affiliate_trackeduntracked	0.00	0.00
lag_bts_bin.1..1369.	0.00	0.01
first_browserSafari	0.00	0.01
affiliate_providergoogle	0.00	0.00
affiliate_channelsem.non.brand	0.00	0.01
signup_methodbasic	0.00	0.00
first_affiliate_trackedomg	0.00	0.00
firstbook_wkdFriday	0.00	0.01

Table 4: Feature importance of the XGBoost

To account for the highly imbalanced classes, two over-sampling techniques were explored. The first technique involved over-sampling with replacement from observations of the under-represented destinations, and under-sampling from observations of the over-represented destinations. The second combined under-sampling with Synthetic Minority Oversampling Techniques (SMOTE). SMOTE involves generating synthetic examples of those under-represented destinations by selecting from the k-nearest neighbors [3]. For this analysis, k = 5 was chosen. For both techniques, over-sampling was performed until each under-

represented destination constituted at least 4% of the data (any higher and it was found that model performance on the test set decreased dramatically). The same model fitting processes (cross-validation with only the top 200 features) were again performed on the new training sets. For both models on both data sets, accuracy and NDCG scores remained the same. However, both models were able to make predictions for more under-represented destinations (rather than just NDF and US), although those predictions were not significantly accurate. Table 5 displays the confusion matrix of predictions from the XGBoost built on the over-sampled training data (not with SMOTE). Confusion matrices for the random forest models, and for models built on the SMOTE training data, were similar.

	AU	CA	DE	ES	FR	GB	IT	NDF	NL	other	PT	US
AU	0	0	0	0	1	0	0	0	1	5	0	22
CA	1	2	0	0	3	0	1	0	1	2	0	14
DE	0	0	0	0	1	1	1	0	0	4	0	14
ES	0	0	1	0	1	1	1	0	1	3	0	17
FR	0	0	2	1	2	0	0	0	0	1	0	9
GB	0	0	0	0	1	2	0	0	0	1	0	8
IT	0	0	1	0	0	0	0	0	1	3	0	7
NDF	0	0	0	0	0	0	0	26154	0	0	0	0
NL	0	0	1	0	5	0	2	0	0	5	0	10
other	1	0	3	1	5	1	1	0	0	6	1	25
PT	0	4	1	0	4	0	3	0	1	7	1	43
US	111	294	213	470	1032	483	586	0	155	2082	43	12930

Table 5: Confusion matrix for predictions made with XGBoost built on the over-sampled training data. Predicted values lie on the Y axis, actual values lie on the X axis.

The XGBoost and random forest were then combined in a stacked model [2]. This process was performed twice: once on the regular over-sampled data and once on the SMOTE over-sampled data. Each time, the following process was followed. The training data was partitioned into five folds, each fold containing about 20% of the data. Each model was then built on the training folds, and tested on the held out fold. For example, the XGBoost was first built on folds one, two, three and four, and used to make predictions on fold five. In the second iteration, the XGBoost was built on folds two, three, four and five, and used to make predictions on fold one. This process was repeated for both models until each fold had been used as a test fold. Predictions from these models were stored as two columns in the training data. Then, each model was fit to the full training data (ignoring the folds and predictions created in the previous step), and used to predict on the held out test set. Predictions from these models were stored in two columns in that test set. A final XGBoost was used as the model to combine, or stack the information from the previous processes. The model was fit to the predictions stored in the training set and tested on the predictions

stored in the test set. Resulting accuracy and NDCG scores for both stacked models were the same as all previous models (accuracy: 87%, NDCG score: 0.92). Table 6 displays the confusion matrix of predictions from the stacked model, built on the over-sampled training data (not SMOTE). The confusion matrix for the stacked model fit to the SMOTE training data was similar.

	AU	CA	DE	ES	FR	GB	IT	NDF	NL	other	PT	US
AU	0	0	0	3	0	2	2	0	0	11	0	42
CA	0	2	0	2	3	1	2	0	0	7	0	47
DE	0	3	2	2	2	2	2	0	1	6	0	52
ES	0	2	1	4	2	3	1	0	1	7	0	30
FR	0	0	0	0	1	0	0	0	0	0	0	2
GB	0	1	4	1	2	3	0	0	0	5	0	26
IT	0	1	0	3	3	1	4	0	1	1	0	20
NDF	0	0	0	0	0	0	0	37362	0	0	0	0
NL	1	2	1	3	5	0	6	0	0	5	1	35
other	1	2	3	1	8	2	6	0	0	26	0	95
PT	1	2	3	4	7	2	3	0	1	13	0	73
US	158	413	304	651	1473	681	824	0	224	2947	64	18290

Table 6: Confusion matrix for predictions made with the stacked model, built on the over-sampled training data. Predicted values lie on the Y axis, actual values lie on the X axis.

Lastly, all three models—the XGBoost, random forest, and stacked model—were used to perform five-fold cross-validation on the full data. Metrics achieved in this step were the same as all previous metrics (Accuracy: 0.87, NDCG score: 0.92). Run times for the XGBoost, random forest and stacked models were 27.5 minutes, 28.7 minutes, and 3.24 hours, respectively.

Although the NDCG scores achieved by models in this paper are higher than the competition's top performing models, it does not reflect how they would perform on the same data used to evaluate those model submissions.

6 Discussion and Conclusion

Although various strategies for improving model performance were explored, no effective strategy was found—each model achieved around 87% accuracy and NDCG scores of about 0.92. One explanation for this is how imbalanced the target classes are, as previously described. Essentially, there is not enough information available about the under-represented destinations to enable models to make accurate predictions. Instead, each model classified them mainly as bookings to the US. Since NDF and US constitute such a large portion of the data, the accuracy of each model remained fairly high at 87%. In order to create a model that can accurately predict all 12 possible destinations, a strategy to resolve this issue of highly imbalanced classes would need to be implemented.

To explain why the stacked model did not result in better performance, the base models need to be examined. Stacked models generally perform well when its base models differ. For example, suppose base model 1 could predict five countries well, but not the other 7. Suppose the opposite was true for base model 2: it could predict those 7 countries well, but not the other 5. In this case, a stacked model would be useful because of its ability to build upon and correct the base models' performances. But since both the random forest and XGBoost could only make accurate predictions for NDF and the US, the stacked model's performance was not a significant improvement.

There are several possibilities for continued work on this project. One idea is to stack more than just two models, as the top three participants in this competition combined numerous algorithms in multiple layers. Another possible strategy, used by the second place participant of this competition, is to build additional models to predict and impute missing values for certain features. Missing values were a significant issue with several features, like age and gender. Building models to address this may be helpful with model performance. Another strategy is to find a way to incorporate the other two available data sets: countries, which contains summary statistics about the destinations, and age_gender_bkts, which contains statistics describing the users' age groups, gender, and destinations. One idea, as implemented by the second place winner, is to build models to predict features within the countries data, and used those features to predict booking destinations. Exploring parameter tuning may also be helpful with model performance.

Overall, through the completion of this project, I have gained a strong understanding of machine learning. Through this process, I learned about different strategies for addressing imbalanced classification tasks, different methods for addressing missing values, various ways to optimize model performance, and have greatly improved my coding skills.

References

[1] Introduction to Boosted Trees - xgboost 0.7 documentation.

[2] Ben Gorman. A Kaggler's Guide to Model Stacking in Practice | No Free Hunch, 2016.

[3] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique, 2002.