Derek Siker
March 28, 2018

Predicting Airbnb New User Bookings with Gradient Boosted Trees

Project Overview

By accurately predicting where new users will book their first travel experiences, Airbnb can improve the personalization of user content, increase the likelihood of first time user bookings, and better gauge global user demand. "With over 34,000+ cities to choose from across 190+ countries, understanding user demand poses an important challenge to business development and plays a critical role in the improvement of the user experience."[1]

In this project I built an application capable of predicting the country in which a new Airbnb user will make his or her first booking. The application uses an ensemble classifier trained on in-house Airbnb data that looks for patterns between user demographics and initial booking destinations and then uses this information to make predictions on new users. The project was inspired by Airbnb's recruiting competition hosted on Kaggle.[2]

Problem Statement

The goal is to predict the country in which a new Airbnb user will make his or her first booking using in-house data provided by Airbnb; the tasks involved are the following:

- Download and preprocess user demographic and session data[3]
- Train a classifier that can predict where a new user will make their first booking
    - All of the users in the data are from the US
    - There are 12 possible outcomes of the destination country ('US', 'FR', 'CA', 'GB', 'ES', 'IT', 'PT', 'NL','DE', 'AU', 'No Destination', and 'other')

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

- For every user in the dataset, submission files should contain two columns: id and country. The final destination country predictions must be ordered such that the most probable destination country goes first (with a maximum of five predictions per user)[4]

The main algorithms that will be used to solve this challenge are the LightGBM and XGBoost algorithms, both of which are variants of gradient boosted decision trees. An in-depth explanation of the justification for using these algorithms and the details of the solution path are outlined in the Algorithms & Techniques section and Implementation section respectively.

Metrics

The competition allows up to 5 predictions for each new user in the test dataset. The metric used to grade these predictions is *normalized discounted cumulative gain* (NDCG), which measures the performance of a recommender based on the relevance of the recommended entries. The main reason for selecting this metric is that it measures the usefulness or certainty of a guess based on its position in the results list, and incrementally penalizes additional guesses. The NDCG formula and definition used by the competition are both copied below:

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

$$nDCG_k = \frac{DCG_k}{IDCG_k},$$

"where rel_i is the relevance of the result at position i. and $IDCG_k$ is the maximum possible (ideal) discounted cumulative gain for a given set of queries. All NDCG calculations are relative values on the interval 0.0 to 1.0. The ground truth country is marked with relevance of 1, while the rest have relevance of 0. For example, if for a particular user the destination is FR, then the predictions become:" [5]

[FR] gives a $NDCG = \frac{2^1-1}{log_2(1+1)} = 1.0$

[US, FR] gives a $DCG = \frac{2^0-1}{log_2(1+1)} + \frac{2^1-1}{log_2(2+1)} = \frac{1}{1.58496} = 0.6309$

[4] https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings/data
[5] https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings#evaluation

Analysis

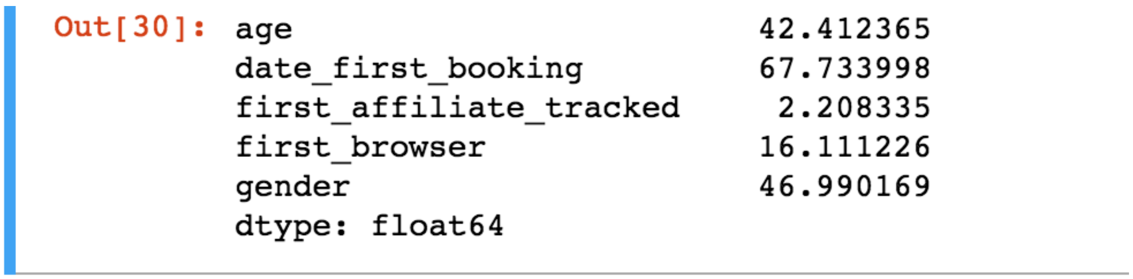
Data Exploration

The data provided by Airbnb and used in this project consists of the following:

- train_users.csv
    - Training set that contains user demographics and signup data
    - Contains 213,451 unique users
    - Has the following fields:
        - 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
- test_users.csv
    - Test set that contains user demographics and signup data
    - Contains 62,096 unique users
    - Has the same fields as the training set except for date_first_booking and country_destination
- sessions.csv
    - Contains user web session data and will be appended to user data to make further inferences
    - Has the following fields:
        - action
        - action_type
        - action_detail
        - device_type
        - secs_elapsed

I began by merging the training and test data to get a better understanding of the data distribution as a whole. First I wanted to understand the total amount of missing data in order to get a better idea of which fields could either be dropped or needed further exploration.

The categories for age, date_first_booking, and gender all contained the highest percentage of missing values. Since date_first_booking is not represented in the user test data it can be dropped.



```
Out[30]: age                          42.412365
         date_first_booking           67.733998
         first_affiliate_tracked       2.208335
         first_browser                16.111226
         gender                       46.990169
         dtype: float64
```

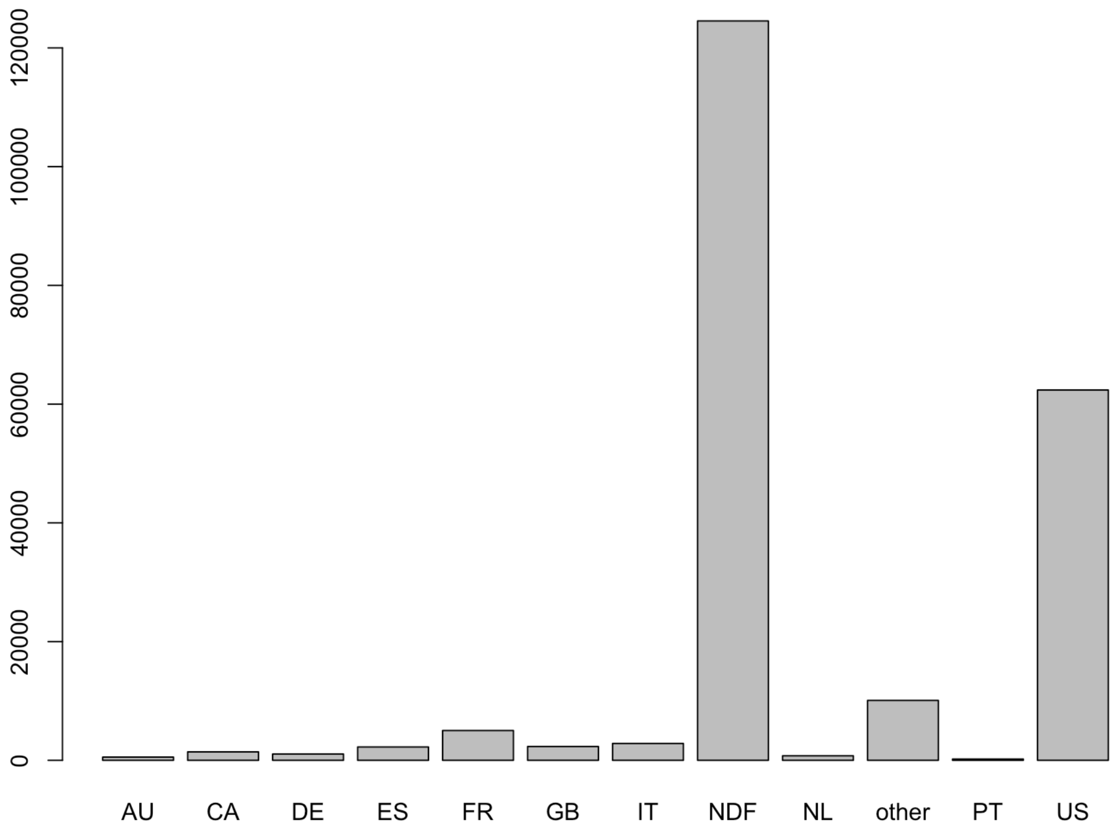
The user age field contained a number of outliers that are either below Airbnb's age limit (18) or above the longest recorded human life (122). As it turns out, many of the ages given above the longest recorded human life turned out to be years of birth, and so will need to be reformatted. For all of the other users with age outliers, I converted the their age field to NaN so that the remaining data could still be used.

The remaining fields are all either dates or categorical and will need to be transformed before they can be further explored. I began by transforming each of the date fields to DateTime Index. Then I transformed the remaining features to Python category types.

At this point I also concluded that visualizing the data would prove a better method for drawing further insights. In the following section I used matplotlib and seaborn to further explore the missing user data as well as the remaining categorical features.

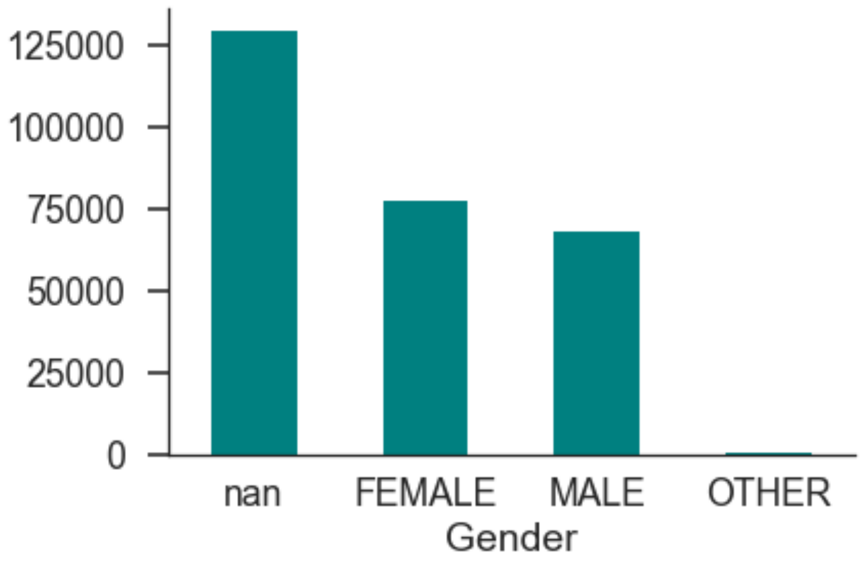
Exploratory Visualization

Now equipped with a better understanding of the missing data I decided to visualize some of feature distributions. First I wanted a better understanding of the countries that had been visited by users in the training data. As seen in the chart below, the vast majority of users either did not choose a destination or traveled within the United States. This could be explained by the fact that Airbnb is a relatively new company based out of the US and that they only started to expand in foreign markets between 2011-2014.[6] Given the unequal distribution of country destinations I will need to resample during the preprocessing stage in order to insure that the model isn't overly biased.

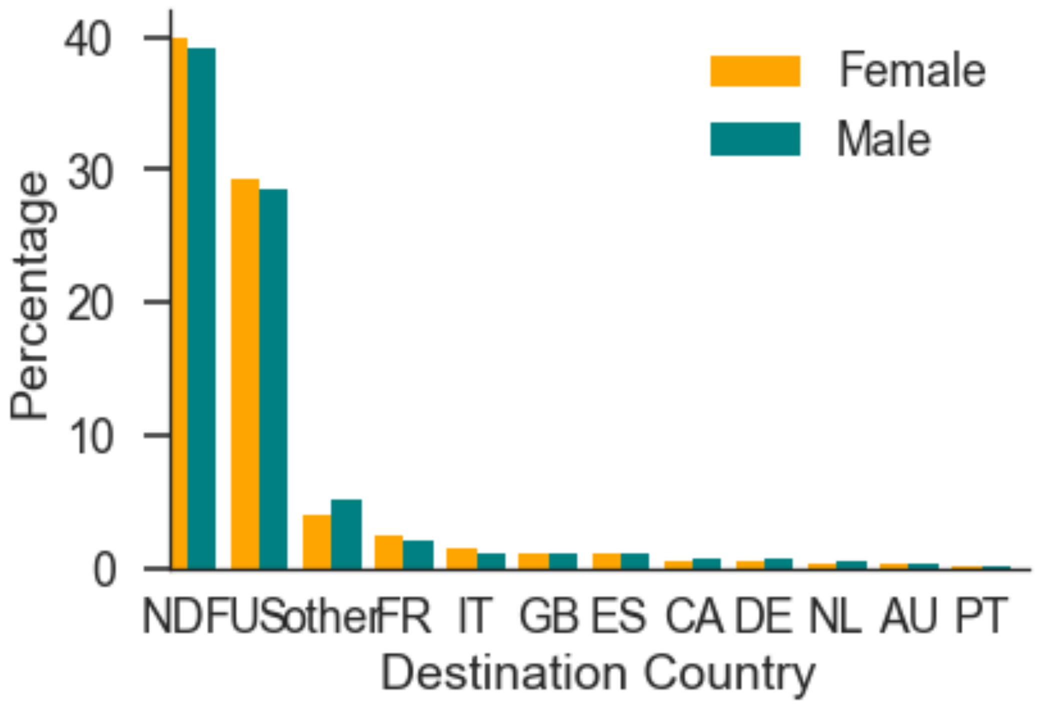


I also wanted to see whether this distribution persisted with regards to gender and age. After summing up the total number of unique male and female users, I found that the gender distribution was roughly equal – approximately 10k more females than males – and that a large percentage of users simply chose not to give their information regarding sex.

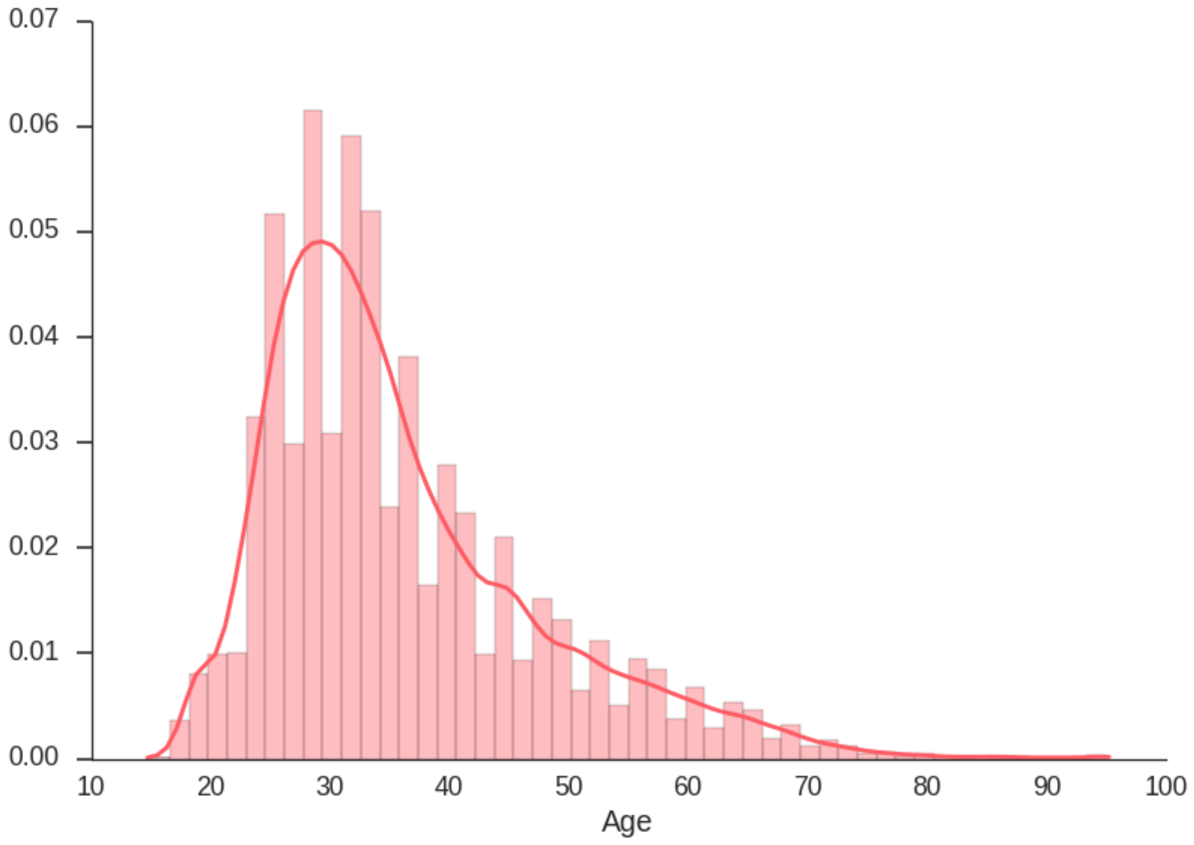
6 https://en.wikipedia.org/wiki/Timeline_of_Airbnb?oldformat=true



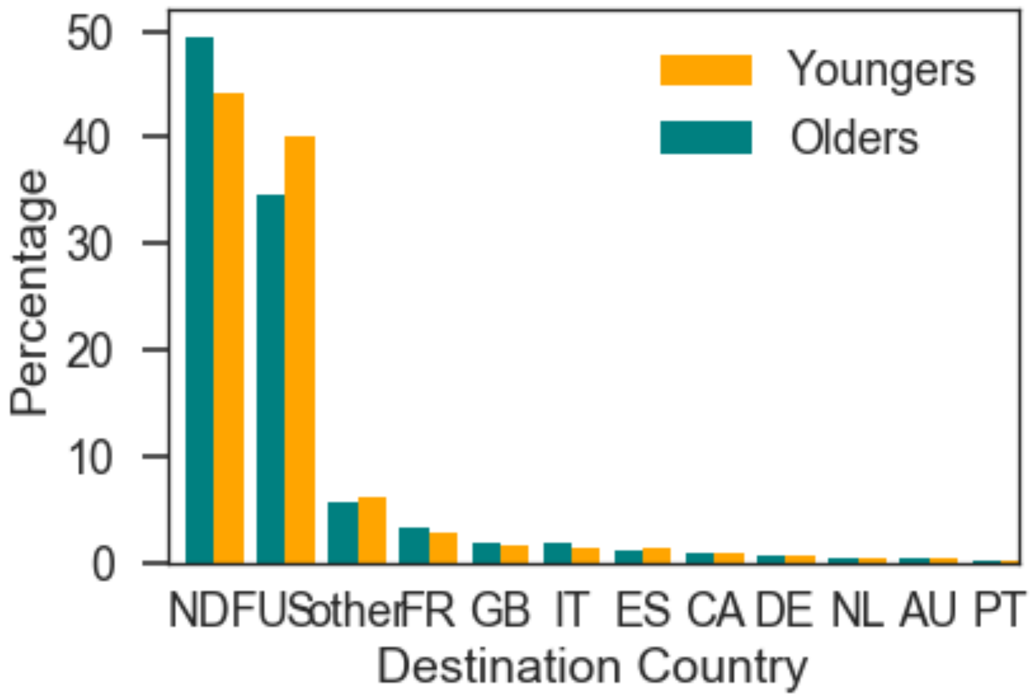
I then split each country destination into a male and female column and plotted the chart below. As previously mentioned, there were no significant differences in the preferences between the sexes, although men were slightly more likely than women to travel to other countries beyond the scope of those listed.



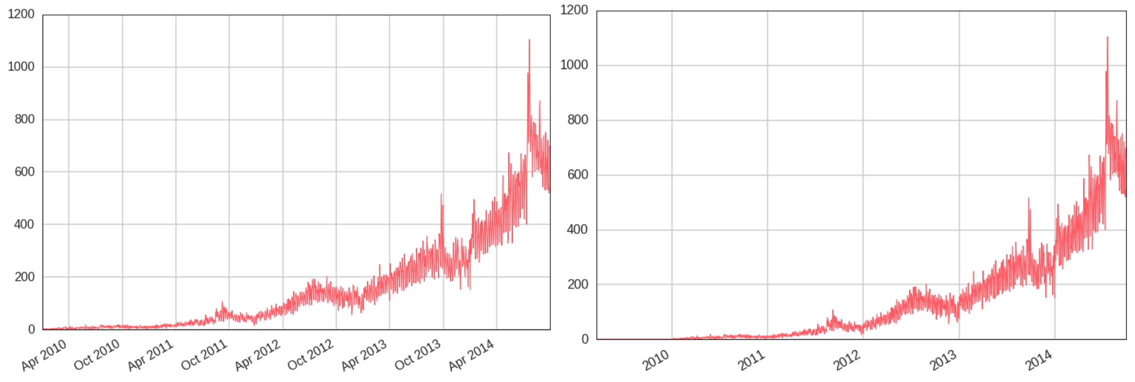
Next I explored the age distribution with the same intent of seeing whether any travel preference existed between older and younger users. First I plotted the total age distribution and found that the majority of users were between the ages 25 – 40.



This result wasn't that surprising given the reputation of Airbnb as a relatively young Silicon Valley startup with a disruptive business model and a burgeoning following of digitally-native millennials. But now I was faced with the more difficult task of arbitrarily deciding whom to classify as an old user. After trying a number of different ages I settled on 45 as a good delimiter because it pointed most clearly to a trend among younger users to travel more in the US, and for older users to not pick a destination at all. But one could also argue that this trend was based on a massaging of the data, and that if one chose a more flattering age cutoff, then the pattern would appear far more muted.



Given the disproportionate number of users traveling to the US, I figured that language might also play a role in destination choice. Lending credence to this hypothesis was the fact that over 96% of users selected English during their session. I also wanted to see whether Airbnb's status as a relatively new company based in San Francisco contributed to the low amount of foreign travel. While I wasn't able to find a direct causal link, I did notice correlation between the sharp increase in user account creation and activity between 2011-2014, signifying that a possible explanation for the relatively low amount of travel to countries outside the US could be attributed to Airbnb's status as a growing startup.



It was also just around this time that Airbnb began its initial push into foreign markets, so it's plausible that bookings outside of the US didn't start to pick up until a few years into the business expansion plan.

To further corroborate this hypothesis I wanted to see whether new users – after 2014 – behaved differently from old users. After plotting the data, I noticed that new users booked less often, and when they did make a booking it was primarily in the United States (thereby refuting my initial hypothesis).

After exploring the user data, I also wanted to see whether any insights could be gained from the user session data. The session data contained 135,484 unique user IDs, and after combining it with the user training data I was unsurprisingly able to determine that the majority of bookings were made from Macbooks and Windows desktops, and that the primary user action consisted of viewing listings.

Algorithms & Techniques

In this project I applied a gradient boosting approach and used both the LightGBM and XGBoost algorithms. Broadly speaking, boosting is an ensemble learning method that builds a strong classifier by adding together a number of weaker classifiers. One might call this a "strength by numbers" approach where the 'weakness' of a learner refers to the relatively small correlation between it and the target variable (in this case the target variable is user booking destination).

Ilan Reinstein on KDNuggets goes on to describe the process of boosting in more detail:

> "By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor, until the training data is accurately predicted or reproduced by the model. Now, gradient boosting also comprises an ensemble method that sequentially adds predictors and corrects previous models. However, instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. So, in the end, you are updating

> your model using gradient descent and hence the name, gradient boosting. This is supported for both regression and classification problems."[7]

LightGBM is one example of an algorithm that implements decision tree boosting. Pushkar Mandot, a data scientist at Aera Technology gives the following summary of LigthGBM:

> "LightGBM is a gradient boosting framework that uses a vertical tree-based learning algorithm. It differs from other gradient boosting algorithms because it grows vertically or leaf-wise by choosing the leaf with max delta loss to grow. Leaf-wise algorithms can reduce more loss than a level-wise algorithm. LightGBM is also prefixed as 'light' because of its high speed. It can handle large data sets, but also focuses on accuracy and supports GPU learning."[8]

XGBoost is another powerful algorithm that implements decision tree boosting. Jason Brownlee from Machine Learning Mastery gives the following explanation:

> "XGBoost stands for e**X**treme **G**radient Boosting and is another implementation of gradient boosted decision trees. It was developed for the sole purpose of model performance and computational speed and was engineered to exploit every bit of memory and hardware resources for tree boosting algorithms. XGBoost is unique in its ability to add regularization parameters, which allows it to be extremely fast without sacrificing accuracy."[9]

I decided to use both LightGBM and XGBoost because of their strong track records in previous Kaggle competitions and their ability to yield relatively high accuracy on large datasets without sacrificing performance speed. But more broadly speaking, I went with a boosted tree strategy because it generally tends to outperform other ensemble methods when dimensionality is low (less than 4000 dimensions)[10] while the only major drawback is the need to tune a number of hyperparameters.

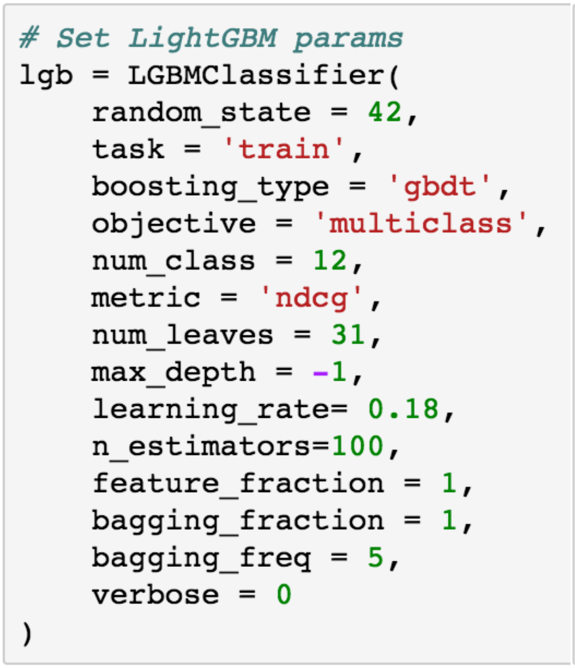
The number of parameters that can be tuned for each of the algorithms is quite extensive so I decided to begin with the default settings and proceed from there. Luckily both the

---

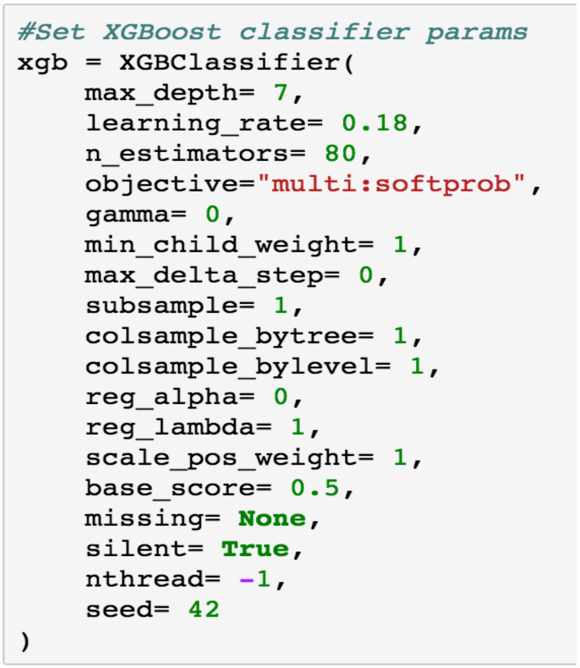
7 https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html
8 https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
9 https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
10 http://fastml.com/what-is-better-gradient-boosted-trees-or-random-forest/

LightGBM and XGBoost algorithms have simple Sci-kit Learn APIs whereby parameters are passed into an LGBMClassifier or XGBClassifier object and then can be parsed by any of Sci-kit Learn's other cross validation or testing functions. After experimenting with a number of different parameters I ended up using the formulations shown below, which I then passed into Sci-kit Learn's cross_val_score function. Within this function I also specified the x-label preprocessed training data, encoded y-label target feature, K-fold cross validation parameters, and custom-built NDCG scorer to be used.

Parameters for LightGBM and XGBoost Classifier Objects:

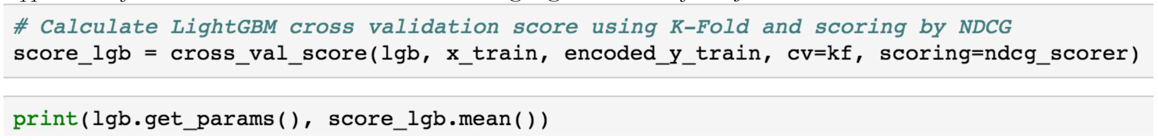


```
# Set LightGBM params
lgb = LGBMClassifier(
    random_state = 42,
    task = 'train',
    boosting_type = 'gbdt',
    objective = 'multiclass',
    num_class = 12,
    metric = 'ndcg',
    num_leaves = 31,
    max_depth = -1,
    learning_rate= 0.18,
    n_estimators=100,
    feature_fraction = 1,
    bagging_fraction = 1,
    bagging_freq = 5,
    verbose = 0
)
```



```
#Set XGBoost classifier params
xgb = XGBClassifier(
    max_depth= 7,
    learning_rate= 0.18,
    n_estimators= 80,
    objective="multi:softprob",
    gamma= 0,
    min_child_weight= 1,
    max_delta_step= 0,
    subsample= 1,
    colsample_bytree= 1,
    colsample_bylevel= 1,
    reg_alpha= 0,
    reg_lambda= 1,
    scale_pos_weight= 1,
    base_score= 0.5,
    missing= None,
    silent= True,
    nthread= -1,
    seed= 42
)
```

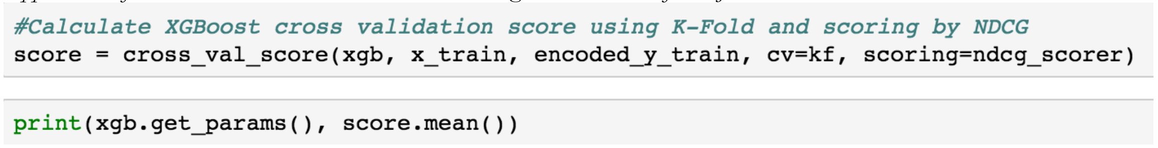
Application of Sci-kit Learn cross_val_score method using LightGBM classifier object:



```
# Calculate LightGBM cross validation score using K-Fold and scoring by NDCG
score_lgb = cross_val_score(lgb, x_train, encoded_y_train, cv=kf, scoring=ndcg_scorer)
```

```
print(lgb.get_params(), score_lgb.mean())
```

Application of Sci-kit Learn cross_val_score method using XGBoost classifier object:



```
#Calculate XGBoost cross validation score using K-Fold and scoring by NDCG
score = cross_val_score(xgb, x_train, encoded_y_train, cv=kf, scoring=ndcg_scorer)
```

```
print(xgb.get_params(), score.mean())
```

I ended up leaving most of the parameters in their default settings because the accuracy achieved with default settings was already quite high. However, I did add a random state variable, increase the learning rate to 0.18, and specify a multiclass objective for both algorithms. After observing the inferior performance of XGBoost in the default setting trial run, I also made further adjustments to it by reducing the number of estimators to 80 and setting max depth at 7.

Benchmark

To create an initial benchmark for the classifier, I used the out-of-the-box random forest algorithm from Sci-Kit Learn. It can easily handle feature interactions and is non-parametric, so one doesn't have to worry greatly about outliers or whether the data is linearly separable. It's also fast and scalable, and doesn't require tuning a large amount of parameters like other classifiers. In this case, if the data includes feature interactions that are not linearly separable, then a random forest will still perform well. By incorporating an ensemble method that doesn't rely on gradient boosting this model also has less chance of overfitting the data while still providing a tree-based benchmark comparison. The out-of-the-box random forest with all parameters set to default achieved the best accuracy with a mean NDCG score of 0.81.

Methodology

Data Preprocessing

The preprocessing done in the 'capstone_cv_model' Jupyter notebook consists of the following steps:

1. User training and test data is joined into a pandas DataFrame and indexed by user id
2. User age data is reformatted to convert years of birth into ages, remove any remaining outliers, convert missing data into a readable format, and bin users by age group
3. All dates are cast to proper datetime format, converted to DateTime index, and then subdivided so that time lag between user account creation and first activity can be created as a new column
4. New column is created in user table to sum missing data for each row
5. All duplicated columns are dropped from the user table
6. One hot encoding is applied to user categorical features
7. User data is split back into train and test sets

Additionally a helper function is imported before the preprocessing stage to enable one hot encoding.

Implementation

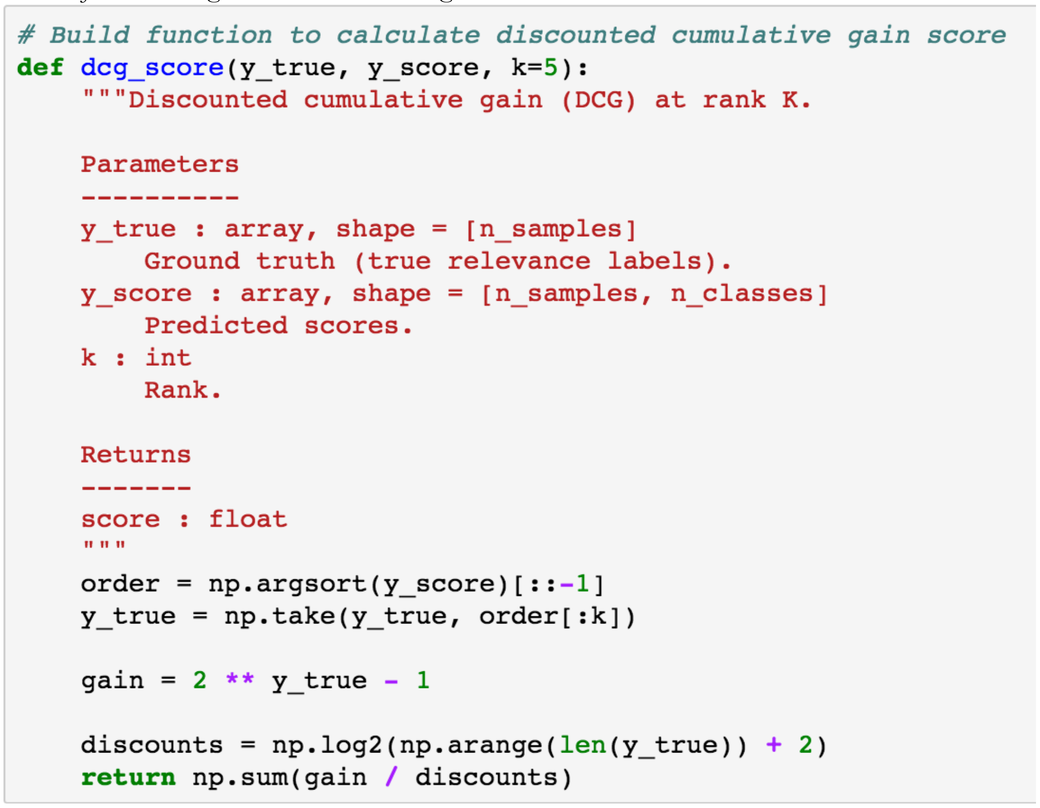
The implementation process can be split up into six main stages and was completed using the preprocessed data outlined in the previous section (further details can be found in the accompanying 'capstone_cv_model' Jupyter notebook):

1. The performance metric scoring function is built by defining a method for normalized discounted cumulative gain and then feeding it into Sci-kit Learn's make_scorer feature (pictured below)
    a. First a method is built to calculate discounted cumulative gain score at rank K=5 (allowing up to 5 country destination predictions)

   b. Next a method is built to calculate normalized discounted cumulative gain score at rank K=5 that utilizes the discounted cumulative gain method
   c. Finally the ndcg_score method is passed into Sci-kit Learn's make_scorer

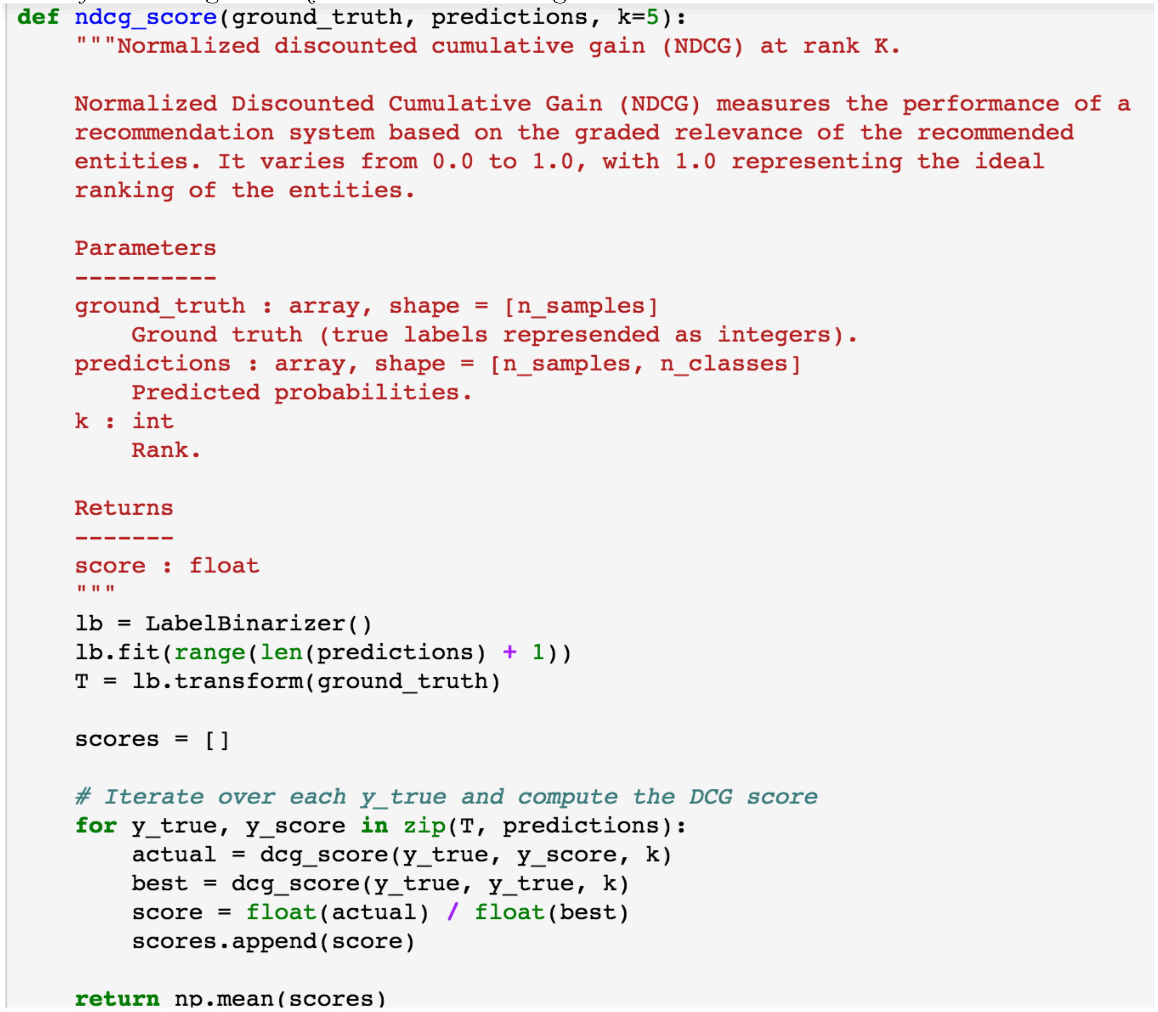
2. The training data is split into training features (x label) and the target feature (y label), after which they are resampled using SMOTE in order to balance the data
3. The oversampled target feature is encoded using LabelEncoder, and then K-fold cross validation parameters are set (in this project I used 10-fold validation)
4. The benchmark classifier is trained on the preprocessed data and scored using Sci-kit Learn's cross_val_score function, after which the mean NDCG score is printed to the terminal
5. The LightGBM classifier is trained on the preprocessed data and scored using Sci-kit Learn's cross_val_score function, after which the mean NDCG score is printed to the terminal
6. The XGBoost classifier is trained on the preprocessed data and scored using Sci-kit Learn's cross_val_score function, after which the mean NDCG score is printed to the terminal

Method for calculating discounted cumulative gain score:



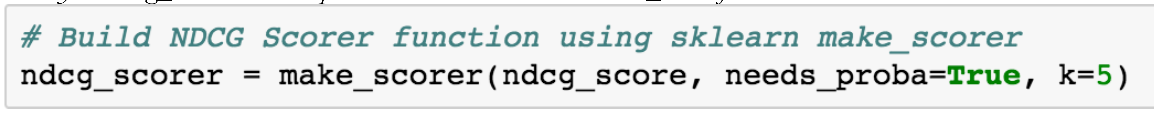
```
# Build function to calculate discounted cumulative gain score
def dcg_score(y_true, y_score, k=5):
    """Discounted cumulative gain (DCG) at rank K.

    Parameters
    ----------
    y_true : array, shape = [n_samples]
        Ground truth (true relevance labels).
    y_score : array, shape = [n_samples, n_classes]
        Predicted scores.
    k : int
        Rank.

    Returns
    -------
    score : float
    """
    order = np.argsort(y_score)[::-1]
    y_true = np.take(y_true, order[:k])

    gain = 2 ** y_true - 1

    discounts = np.log2(np.arange(len(y_true)) + 2)
    return np.sum(gain / discounts)
```

Method for calculating normalized discounted cumulative gain score:



```python
def ndcg_score(ground_truth, predictions, k=5):
    """Normalized discounted cumulative gain (NDCG) at rank K.

    Normalized Discounted Cumulative Gain (NDCG) measures the performance of a
    recommendation system based on the graded relevance of the recommended
    entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal
    ranking of the entities.

    Parameters
    ----------
    ground_truth : array, shape = [n_samples]
        Ground truth (true labels represended as integers).
    predictions : array, shape = [n_samples, n_classes]
        Predicted probabilities.
    k : int
        Rank.

    Returns
    -------
    score : float
    """
    lb = LabelBinarizer()
    lb.fit(range(len(predictions) + 1))
    T = lb.transform(ground_truth)

    scores = []

    # Iterate over each y_true and compute the DCG score
    for y_true, y_score in zip(T, predictions):
        actual = dcg_score(y_true, y_score, k)
        best = dcg_score(y_true, y_true, k)
        score = float(actual) / float(best)
        scores.append(score)

    return np.mean(scores)
```

Finally the ndcg_score method is passed into Sci-kit Learn's make_scorer function :



```python
# Build NDCG Scorer function using sklearn make_scorer
ndcg_scorer = make_scorer(ndcg_score, needs_proba=True, k=5)
```

The main complication that arose during implementation was figuring out how to score each of the algorithms. Both LightGBM and XGBoost had in-built features for running cross-validation, but only LightGBM had an in-built NDCG scorer. Before realizing that both LightGBM and XGBoost had Sci-kit Learn APIs, I was faced with the far more difficult task of figuring out how to implement the customized NDCG scoring function, because neither algorithm could be passed into Sci-kit Learn as a parameterized classifier object. After discovering the API documentation, I was able to treat both algorithms in a similar manner to the Sci-kit Learn random forest algorithm that was being used as a benchmark. This made for much cleaner and aesthetically pleasing code.

Refinement

As mentioned in the Benchmark section, the out-of-the-box random forest classifier achieved a decent accuracy on the validation set, around 0.807 using the NDCG scoring function and then taking the mean score. Given the relatively high accuracy of this score in comparison to others on the Kaggle leaderboard, I decided to keep it without making any further adjustments to the benchmark parameters.

The LightGBM algorithm on default settings with a multiclass objective parameter achieved a slightly improved mean NDCG score of 0.825. The XGBoost algorithm on default settings with a 'multi:softprob' objective parameter achieved the a mean NDCG score of 0.809. I refined both algorithms by systematically adjusting the following parameters for each and then taking the best respective accuracy score:

- learning_rate – the boosting learning rate
- max_depth – maximum tree depth for base learners (-1 means no limit)
- n_estimators – number of boosted trees to fit

Given the long training time for each iteration, I had to be conservative about how to test different parameters. Because each of the algorithms already performed quite well under default parameters, I decided not to stray far and only made incremental changes to each parameter. Ultimately I ended up using a grid search approach for both algorithms and testing three different variations of each parameter by increasing the values minimally according to the scale of their default settings.

The LightGBM algorithm's highest cross validation score was 0.83 with a learning rate of 0.18, a max_depth of -1, and n_estimators set to 100.

The XGBoost algorithm's highest cross validation score was 0.824 with a learning rate of 0.18, a max_depth of 7, and n_estimators set to 100.

Results

Model Evaluation & Validation

During development, a validation set was used to evaluate the final model. The final LightGBM architecture and parameters were chosen because they achieved the highest score among the tried combinations. In addition to the parameters for which the final model was chosen, I also experimented with new parameters to achieve a higher score. The full parameter list for the final model is outlined below:

- random_state = 42,
- task = 'train',
- boosting_type = 'gbdt',
- objective = 'multiclass',
- num_class = 12,
- metric = 'ndcg',
- num_leaves = 31,
- max_depth = -1,
- learning_rate= 0.18,
- n_estimators=100,
- feature_fraction = 1,
- bagging_fraction = 1,
- bagging_freq = 5,
- verbose = 0

The final model generalized quite well on unseen data with a mean NDCG score of 0.831 on the validation set. However, to test whether the model was robust enough for the problem I also conducted sensitivity analysis by removing various features from the training data such as user age bins, and by leaving out various stages of the preprocessing phase. As expected, this had quite a drastic negative effect on model performance, bringing it as low as 0.5 in one variation. That said I still believe the model can be trusted because it performed relatively well on both the training and validation sets without leaving any indication of overfitting or bias, and I accounted for sensitivity bias by oversampling the data using SMOTE. I also believe that the chosen parameters were appropriate because they seemed to have the largest effect on model performance in comparison to the parameters added after model selection.

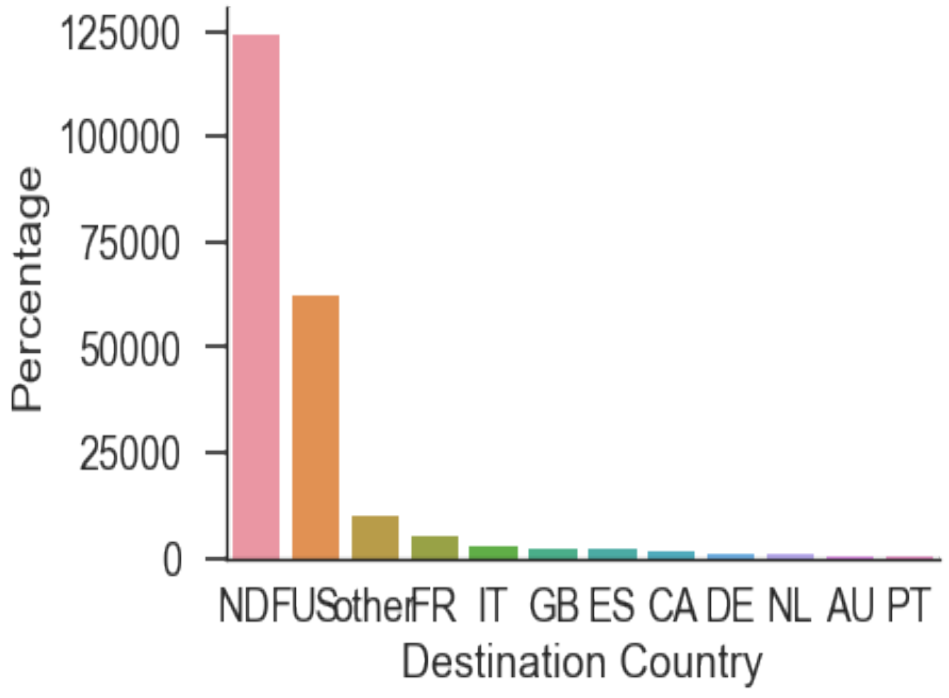
Justification

Using the final LightGBM model I achieved a mean NDCG score of 0.831. The benchmark random forest algorithm with default parameters only scored 0.807, meaning the LightGBM algorithm yielded a 3% improvement. In return for this improvement in accuracy the LightGBM algorithm only showed a slight increase in performance time (not more than one minute). When comparing both algorithms with only default parameters, LightGBM still yielded a 2.2% improvement in accuracy with a score of 0.825. This leads me to believe that the final solution is significant enough to have at least partially solved the problem within the domain of data being used. Had I had more time, I would have also used features from the user sessions dataset, as well as constructed new features to further analyze destination country as a function of proximity to holidays and vacation seasons. That said, the LigthGBM model performed well on the problem and would be a recommendable improvement to the benchmark model.

Conclusion

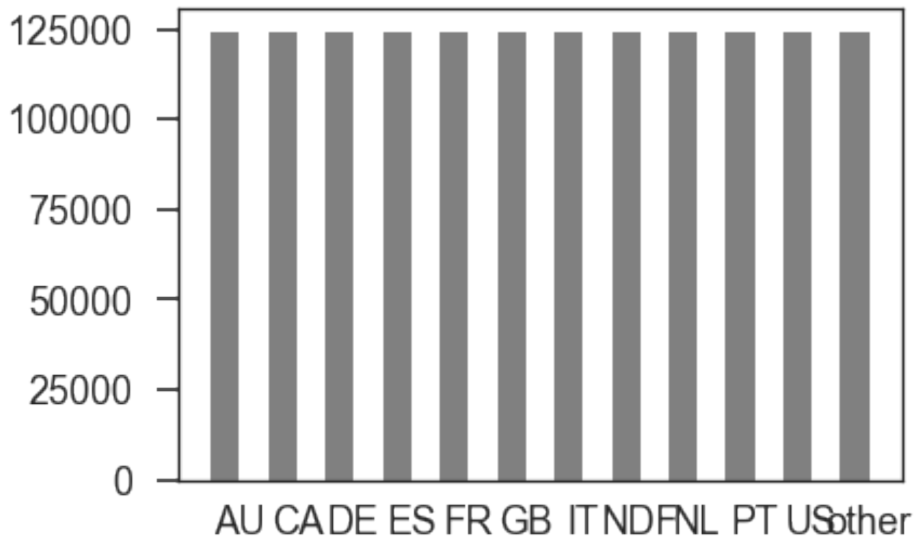
Free Form Visualization

One of the most important steps taken in the project was to account for the uneven distribution of country destinations in the user data. I did this by using a resampling technique from Sci-kit Learn's imbalanced-learn module called SMOTE, which oversamples the underrepresented classes by creating new instances of them. However, I also wanted to visualize this process to make sure I understood exactly how much new data was being added, and whether it would have any impact on the overall performance speed of the algorithms. Below is the chart from the exploratory analysis section showing the original distribution of country destinations



As previously discussed, the users without bookings accounted for the majority of users in the data with an average ratio of 25:1 against most other country destinations.

After using SMOTE I re-plotted the frequencies and found that new data had been synthesized for each underrepresented country such that all of the labels were now equally represented.



This did make for a longer performance time of about a minute, but also insured that I was using balanced data going forward.

Reflection

The process for this project can be summarized in the following steps:

1. The initial problem, metrics, and datasets were obtained through Airbnb's competition hosted on Kaggle
2. The data was imported, explored, and visualized using pandas, numpy, matplotlib, and seaborn
3. The LightGBM and XGBoost algorithms were selected as possible solutions to the challenge
4. The default Sci-kit Learn Random Forest algorithm was selected as a benchmark classifier

5. The data was preprocessed using the insights gathered during the exploratory analysis phase, and then split into a training and test set and further processed using Sci-kit Learn's imbalanced-learn SMOTE module.
6. Each classifier was trained using the preprocessed data and default parameters
7. The LightGBM and XGBoost algorithms were then retrained using the preprocessed data and a grid search cross validation approach that took into account learning rate, number of boosted trees, and maximum tree depth as parameters
8. A variation of the LightGBM algorithm was selected as the final model and updated with additional parameters and then retrained
9. The final model was then subjected to sensitivity analysis and tested to see how well it could generalize onto unseen examples

I found the data preprocessing stage to be the most labor intensive and challenging aspect of the project because there were so many irregularities. In addition to being extremely unbalanced, the data also had a number of irregular entries and large amounts of data were missing. This meant that significant amount of time had to be spent cleaning the data and munging it before I could get to the more interesting work of drawing insights. Nevertheless, the most interesting conclusion I drew from the project was that both younger and newer users tended to travel more in the United States than abroad. My hypothesis is that this has to do with Airbnb being a relatively new company based in the United States, and that younger travelers might have less disposable income to travel abroad.

The final model did an adequate job of predicting user destinations but probably would have performed even better if the user sessions data had been taken into account. I imagine that this model could also be generalized to similar kinds of problems such as predicting demand for hotels around the world, but would certainly need to be further tuned before yielding any meaningful results.

Improvement

The biggest improvement that I wish I had implemented earlier in this project was running the data pipeline through a GPU instead of a CPU. When running the algorithms locally I quickly realized that testing anything more than a few parameters would take countless hours, and at one point my computer even crashed. I would have saved at least a few days had I used a framework like Hadoop with Apache Spark and rented server space through Amazon Web Services, not to mention I would have been able to test many more parameters for each algorithm. Given how close the performance was between LightGBM and XGBoost, it's entirely conceivable that XGBoost would have come out ahead.

With more time I would also utilize the user session data and integrate it with user demographics to produce a more robust model. In particular, it might be useful to examine whether a relationship exists between the type of device being used to make bookings and the country destination.

Lastly, I would conduct further research into different resampling techniques such as Tomek links. Undersampling could also be a viable strategy for balancing the data.