# APPLIED DATA SCIENCE - PROJECT REPORT


## UTKARSH KUMAR SINGH - utksingh@iu.edu


## Kaggle Competition - Airbnb New User Bookings

### 1. INTRODUCTION & PROBLEM DESCRIPTION

Airbnb is a web service that provides customers with the ability to list, find, and rent lodging in locations across the globe (34,000+ cities across 190+ countries). In order for Airbnb to provide a personalized experience for its customers, it has explored the possibility of predicting the country destination in which a user will make a booking. With this information, Airbnb can create more personalized content with its member community, decrease the average time to first booking, and better forecast demand. These goals provide mutual benefit to Airbnb and its customers: personal recommendations can improve customer engagement with the platform, thus encouraging both repeat bookings and referrals to Airbnb for those in a customer's network of close friends and family.

In the context of the dataset we have been provided, our specific task is to predict the first booking destination for new Airbnb customers travelling from the United States. The response variable is the destination where the booking is made. This can be one of 12 possible values: 'US', 'FR', 'CA', 'GB', 'ES', 'IT', 'PT', 'NL','DE', 'AU', 'NDF', and 'other'. All of these correspond to two letter country abbreviations with the exception of 'NDF', which corresponds to 'No destination found' and implies that the user has not made a booking. Thus, embedded in the multiclass classification problem is a binary classification. In addition to knowing where a customer will make a booking, predictive modeling can also help Airbnb determine which customers are likely to make a booking. Response 'other' implies that the user made a booking to a country other than the ten specified.

### 2. DATA

#### 2.1 Datasets Provided

For this competition, Airbnb provided 6 different files. Two of these files provide background information (*countries.csv* - summary statistics of destination countries in this dataset and their locations and *age_gender_bkts.csv* - summary statistics of users' age group, gender, country of destination), while *sample_submission.csv* provides an example of how the submission file containing our final predictions should be formatted. The three remaining files are the key ones:

1. *train_users_2.csv* – This dataset contains data on Airbnb users, including the destination countries. This is the primary dataset that I will use to train the model.
2. *test_users.csv* – This dataset also contains data on Airbnb users, in the same format as *train_users_2.csv*, except without the destination country. These are the users for which we will have to make our final predictions.
3. *sessions.csv* – This data is supplementary data that can be used to train the model and make the final predictions. It contains information about the actions (e.g. clicked on a listing, updated a wish list, ran a search etc.) taken by the users in both the testing and training datasets above. I haven't used this file since the data dates back to 1/1/2014, while the users dataset dates back to 2010, so there is not enough data.

This is a summary of the training and test datasets available. Please see appendix for details:

- *id, timestamp first active, date account created, and date first booking, age, gender, signup_method, signup_flow, language, affiliate_channel, affiliate_provider, first_affiliate_tracked, signup_app, first_device_type, first_browser, country_destination*(not in test dataset).

Training set: 213,451 users - Jan 2010 to Jun 2014

Test set: 62,096 users - July 2014 to Sep 2014

**2.2 Data Exploration**

- Missing Values in Combined(Training + Test) dataset (Refer Fig 2.2.1 in Appendix) -
    - Gender - I observed that gender column had missing values as '-unknown' values. Missing value percentage was around 47%. (Refer Fig 2.2.2 in Appendix)
    - Age - Age had approximately 42% missing values. (Refer Fig 2.2.3 in Appendix)
    - *date_first_booking* - There were 68% data missing for this column. On further analyzing, I observed all the *date_first_booking* values were missing in the test data.
- Country Destination By Itself - My first insight into the behavior of new users on Airbnb's website was that most of the bookings are split between No Destination Found (i.e., a non-booking) and the United States. (Refer Fig 2.2.4 in Appendix)
- Gender-wise Age distribution of users - The gender-wise age distribution of customers was pretty similar and followed a similar curve. (Refer Fig 2.2.5 in Appendix)
- *date_account_created* - Fig 2.2.6 provides excellent evidence of the explosive growth of Airbnb, averaging over 10% growth in new accounts created per month. In the year to June 2014, the number of new accounts created was 125,884 – 132% increase from the year before. (Refer Fig 2.2.6 in Appendix)
- *first_device_type* - The interesting thing about the data in this column is how the types of devices used have changed over time. Windows users have increased significantly as a percentage of all users. iPhone users have tripled their share, while users using 'Other/unknown' devices have gone from the second largest group to less than 5% of users. Further, the majority of these changes occurred between 2011 and 2012, suggesting that there may have been a change in the way the classification was done. (Refer Fig 2.2.7 in Appendix)

**2.3 Data Cleaning**

- I dropped *date_first_booking* column form the combined dataset as it was completely missing from the test dataset.
- Treating missing Values:
    - Gender - Replaced all the '-unknown' values with *nan*.
    - Age - Replaced all the users with age < 14 & >132 with *nan*.
    - Finally, I replaced all the missing values i.e. *nan* with -1.
- *date_account_created* - I extracted day, month, year from this column and created *dac_day, dac_month & dac_year* columns respectively and dropped the *date_account_created* column.
- *timestamp_first_active* - I extracted day, month, year from this column as well and created *tfa_day, tfa_month & tfa_year* columns respectively and dropped the *timestamp_first_active* column.
- One hot encoding feature - As the dataset had lots of categorical variables, therefore, I chose the one-hot-encoding as it can improve the performance of the classifier and makes it easy to fine tune it. The main motivation for the one-hot-encoding was to have a unique representation of the data that can be used by different algorithms, so I can easily compare their performance. I created dummies for all the following variables -
  *'gender', 'signup_method', 'signup_flow', 'language', 'affiliate_channel', 'affiliate_provider', 'first_affiliate_tracked', 'signup_app', 'first_device_type', 'first_browser'*
  and dropped the original columns.

After these steps the data was prepared for model creation.

**3. METHODS**

**1. Random Forest Algorithm -** This is the very first algorithm that I used to train my model. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is

the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

In this particular case dimensionality is less than the sample data size, so the random forest might offer a more complex model with accurate predictions. It also has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. These factors influenced me to try out this model. The results demonstrated that it did pretty well with the test data predictions which is discussed in detail in the results section.

**2. Boosted Tree Methods**

**2.1. Gradient Boosting Algorithm -** Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

**2.2. AdaBoost Algorithm** - Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

**2.3. XGBoost Algorithm** - This is our final model which gave us the best predictions/performance out of all. XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. Standard Gradient Boosting has no regularization like XGBoost, therefore it also helps to reduce overfitting. It implements parallel processing and is blazingly faster as compared to GBM. It also allow users to define custom optimization objectives and evaluation criteria**.**

I was committed to using boosted-tree based methods for a few **reasons**:

- Boosted Trees can outperform Random Forests once they plateau.
- Tree-based methods can handle missingness very well.
- Tree-based methods can handle both categorical and continuous variables.

The reason I chose XGBoost and AdaBoost for our predictive models is that both are variants of Gradient Boosted Machines. AdaBoost, among other things, is GBM with an exponential loss function instead of a deviance loss function. XGboost, among other things, is GBM with regularization. Regularization would be useful in determining feature importance across the unstacked model. I further fine tuned the parameters of the XGBoost model to improve the model performance.

**4. RESULTS**

The competition allowed up to 5 predictions for each new user in the test dataset. For example: United States, France, Australia, Italy, and Portugal. The metric used to grade these predictions was *normalized discounted cumulative gain* (NDCG)**,** which measures the performance of a recommendation system based on the relevance of the recommended entries. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities. This metric is commonly used in information retrieval and to evaluate the performance of web search engines.

To check initial accuracy of models, total training data is split into two parts, 70 percent to train the model and remaining for testing operations. I observed that, Random Forest model gave an accuracy of 57.4%, while Gradient boost and Adaboost provided 57.2% and 63.17% respectively. XGBoost (after tuning the parameters) provided the best performance with 64.77% accuracy.

On submitting the submission file generated by the above mentioned algorithms on Kaggle I obtained the following scores: (Please refer Fig 4.1 in Appendix)

So, XGBoost ( BestParameters : max_depth=6, learning_rate=0.3, n_estimators=25, objective='multi:softprob', subsample=0.5, colsample_bytree=0.5, seed=0) was chosen as my final model. Overall, the final result obtained was pretty good (NCDG Score - 0.86523) and was just 0.01274 lower than the best obtained score i.e. 0.88697.

## 5. DISCUSSION

In all the models, both missing age/gender are some of the most important features to prediction. This is consistent with the exploratory data analysis, which appears to show associations between both (i) missing age/gender and a country of first destination and (ii) missing age/gender and NDF (the latter of which we assume contributed more to the models). Sessions data would also have been handy if we had the data for the pasts users as well.

## 6. CONCLUSION & FUTURE WORK

In a nutshell, I performed exploratory data analysis on Airbnb new user information, wrangled and munged data in python, used python & Tableau for visualizations and graphs, feature engineered time-lag-based variables using python, fit models (XGBoost/Random Forest/AdaBoost/Gradient Boost) using scikit-learn module in python & finally performed predictions on users using XGBoost that provided a NCDG score of 0.86523 on Kaggle.

In the course of performing the analysis I recognized features missing from the dataset that I think could help improve the performance of the models. Following are some **recommendations** to Airbnb :

- Invest in collecting more demographic data to differentiate country destinations.
    - A possible solution is to collect age and gender data from users who sign-in using their e-mail (instead of Google and Facebook) on the same page where they enter credit card information to complete a request to book a reservation. That way, AirBnB will have demographic data on all of its users who book reservations.
- Flag users who decline to enter age and gender. There is correlation between such users and those who do not book.
- Continuously collect browser session activity; such data could be very helpful for predictions. This data was available only for newer users.
- Provide information about the potential number of travellers for a single booking. Knowing that a user has a spouse and two kids in the household may suggest a higher likelihood of booking in different places.

**Future work** - Were I to continue this project, there are several approaches that I had initially planned but did not have time to implement:

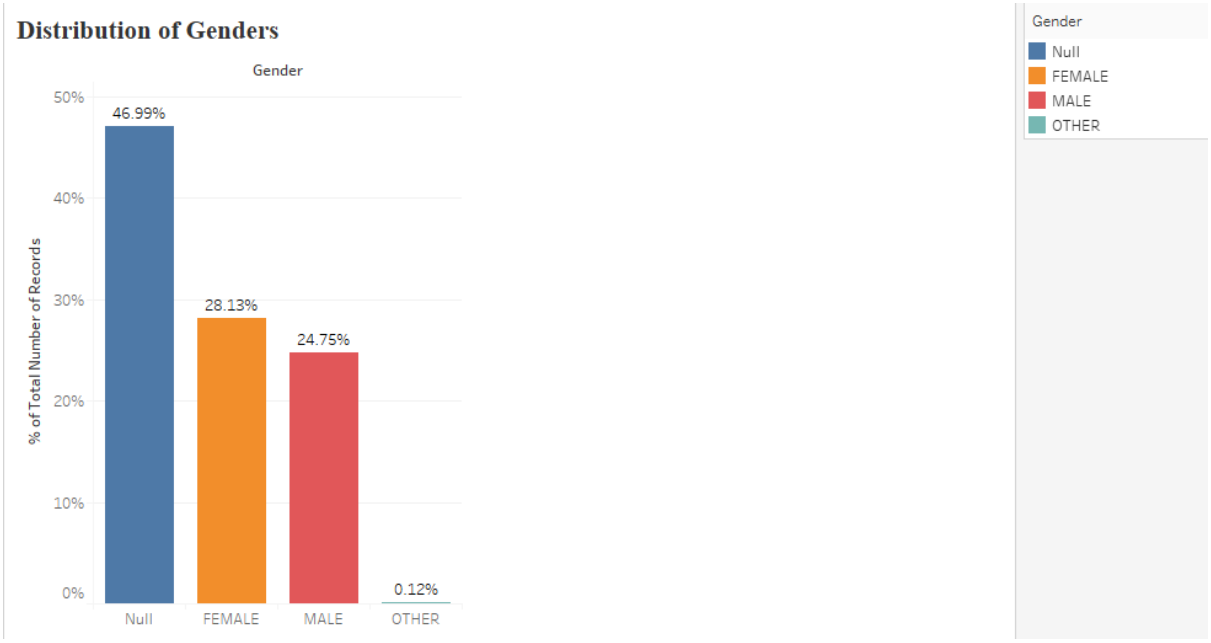
- Will try to use multiple XGBoost models and AdaBoost models with different set seeds and ensemble them as it should certainly improve the performance.
- Stack country of destination predictions to dataset as features to improve predictions.
- Design a strategy to deal with the imbalanced classes by either:
    - Determining whether some NDFs in the training set are hosts on Airbnb on the basis of their web activity, or
    - Predicting NDF versus all other classes; then US versus all other classes; and so on until all the classes are predicted individually. Then add those predictions as new features.

# APPENDIX

## Fig 2.2.1

| | |
|---|---|
| age | 42.412365 |
| date_first_booking | 67.733998 |
| first_affiliate_tracked | 2.208335 |
| gender | 46.990169 |

## Fig 2.2.2




## Fig 2.2.3

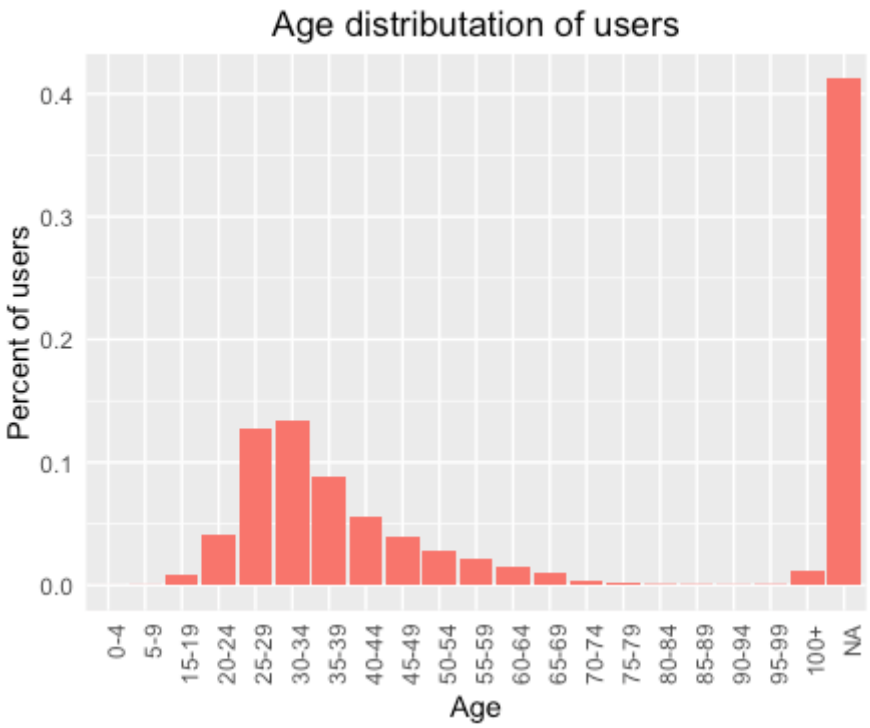



**Fig 2.2.4**

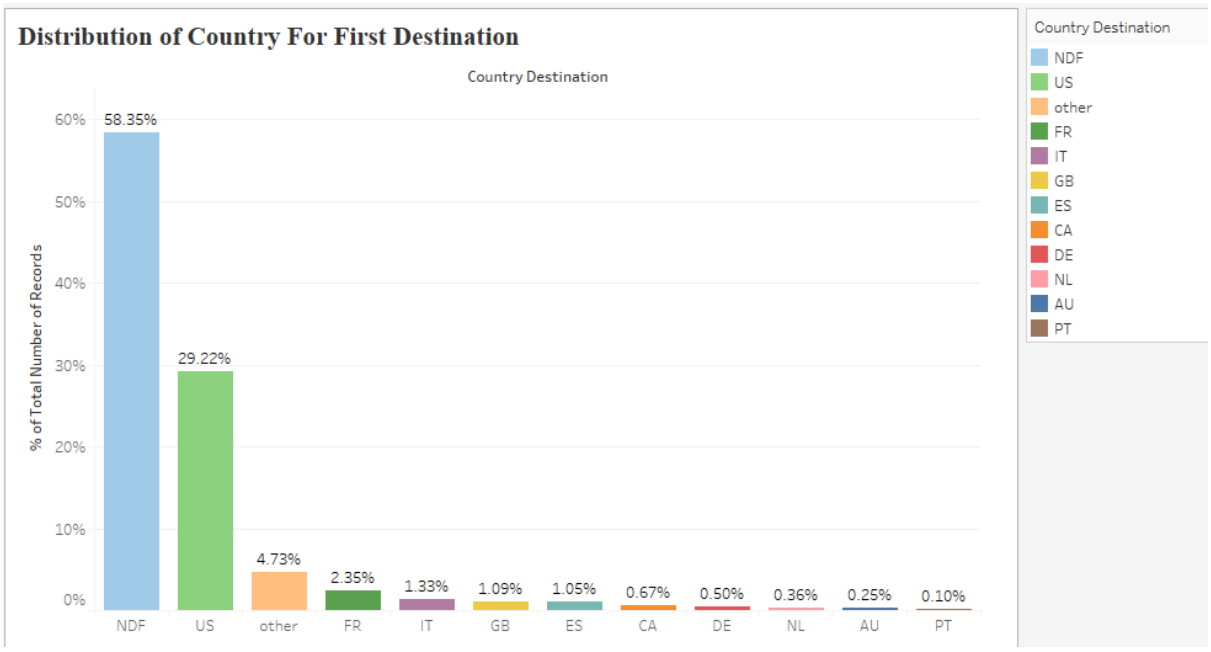



**Fig 2.2.5**

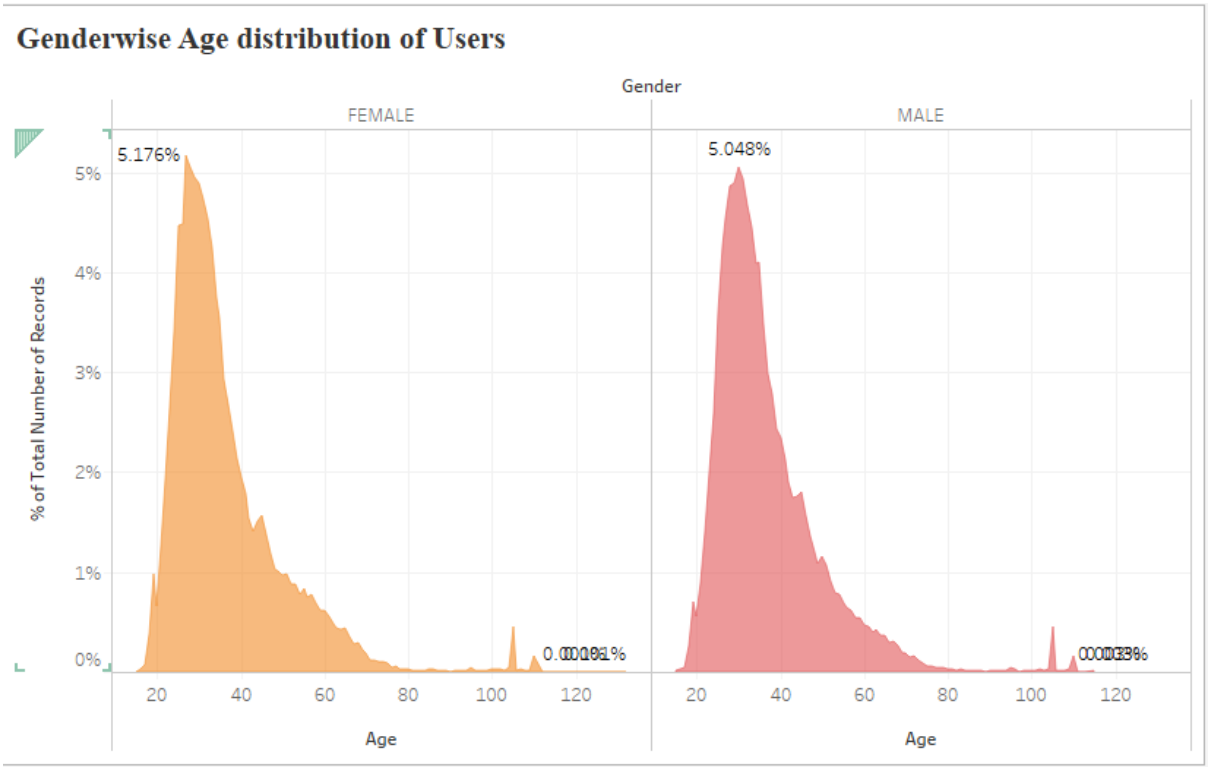



**Fig 2.2.6**

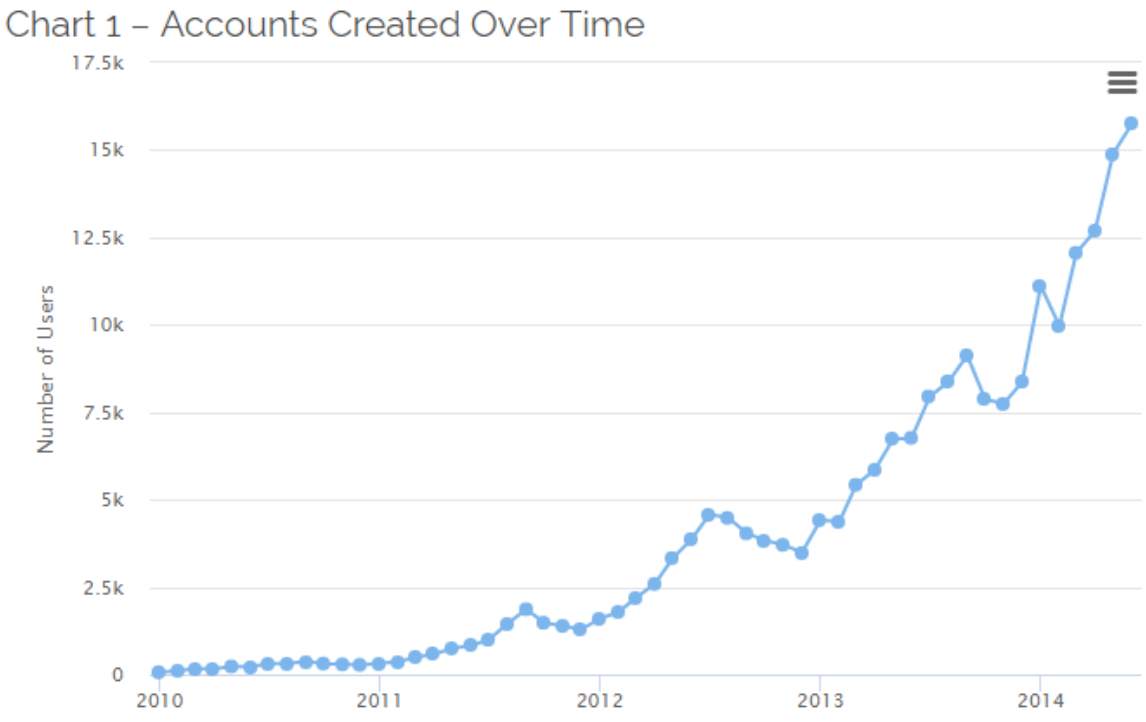
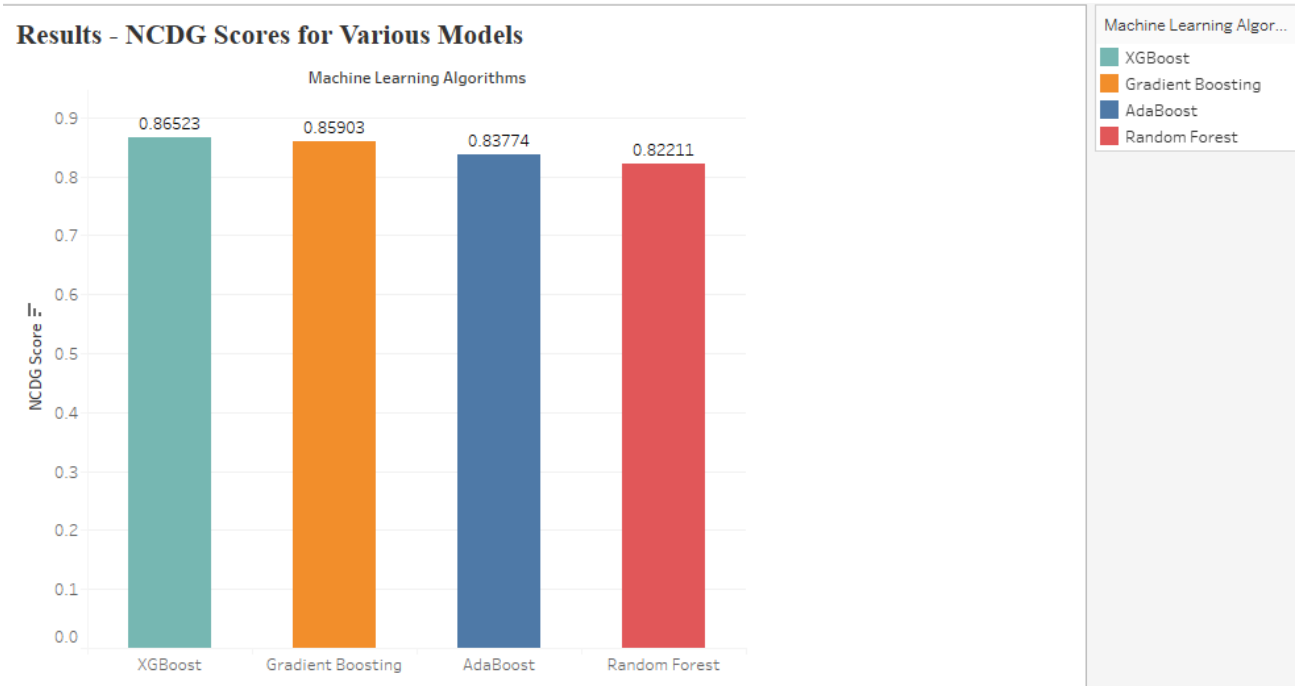



**Fig 2.2.7**

Table 5 – First Device Used

| Device | 2010 | 2011 | 2012 | 2013 | 2014 | All Years |
|---|---|---|---|---|---|---|
| Mac Desktop | 37.2% | 40.4% | 47.2% | 44.2% | 37.3% | 42.0% |
| Windows Desktop | 21.6% | 25.2% | 37.7% | 36.9% | 31.0% | 34.1% |
| iPhone | 5.8% | 6.3% | 3.8% | 7.5% | 15.9% | 9.7% |
| iPad | 4.6% | 4.8% | 6.1% | 7.1% | 7.0% | 6.7% |
| Other/Unknown | 28.8% | 21.3% | 3.8% | 2.8% | 4.6% | 5.0% |
| Android Phone | 1.1% | 1.2% | 0.7% | 0.4% | 2.6% | 1.3% |
| Android Tablet | 0.4% | 0.4% | 0.3% | 0.5% | 0.9% | 0.6% |
| Desktop (Other) | 0.4% | 0.4% | 0.4% | 0.6% | 0.7% | 0.6% |
| SmartPhone (Other) | 0.0% | 0.1% | 0.1% | 0.0% | 0.0% | 0.0% |
| **Total** | **100.0%** | **100.0%** | **100.0%** | **100.0%** | **100.0%** | **100.0%** |

**Fig 4.1**



**Notebook Code**

```
# coding: utf-8
```

```
# In[ ]:
```

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score
from time import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB
```

```
# In[ ]:
```

```
# Loading data
df_train = pd.read_csv('train_users_2.csv')
df_test = pd.read_csv('test_users.csv')
labels = df_train['country_destination'].values
df_train = df_train.drop(['country_destination'], axis=1)
id_test = df_test['id']
piv_train = df_train.shape[0]
```

```
# In[ ]:
```

```
# Creating a DataFrame with train+test data
df_all = pd.concat((df_train, df_test), axis=0, ignore_index=True)
#Removing id and date_first_booking
df_all = df_all.drop(['id', 'date_first_booking'], axis=1)
#Filling nan
df_all = df_all.fillna(-1)



# In[ ]:


df_all['date_account_created'] =  pd.to_datetime(df_all['date_account_created'])



# In[ ]:


# Extracting day, month & year from date_account_created
df_all['dac_year'] = df_all['date_account_created'].dt.year
df_all['dac_month'] = df_all['date_account_created'].dt.month
df_all['dac_day'] = df_all['date_account_created'].dt.day
df_all = df_all.drop(['date_account_created'], axis=1)



# In[ ]:


# Extracting day, month & year from timestamp_first_active

def trim_fraction(text):
    if '.0' in text:
```

```
        return text[:text.rfind('.0')]
    return text


list1 = []
for i in range(0,len(df_all)):
    list1.append(df_all.timestamp_first_active[i].astype(str))


list2 = []
for i in list1:
    list2.append(trim_fraction(i))


df_tfa = pd.DataFrame({"tfa":list2})



tfa = np.vstack(
    df_tfa.tfa.apply(
        lambda x: list(map(int, [x[:4], x[4:6], x[6:8],
                                 x[8:10], x[10:12],
                                 x[12:14]]))
        ).values)


df_all['tfa_year'] = tfa[:,0]
df_all['tfa_month'] = tfa[:,1]
df_all['tfa_day'] = tfa[:,2]
df_all = df_all.drop(['timestamp_first_active'], axis=1)



# In[ ]:
```

```
#Age

av = df_all.age.values
df_all['age'] = np.where(np.logical_or(av<14, av>132), -1, av)



# In[ ]:

# One-hot-encoding features

ohe_feats = ['gender', 'signup_method', 'signup_flow', 'language',
             'affiliate_channel', 'affiliate_provider',
             'first_affiliate_tracked', 'signup_app',
             'first_device_type', 'first_browser']
for f in ohe_feats:
    df_all_dummy = pd.get_dummies(df_all[f], prefix=f)
    df_all = df_all.drop([f], axis=1)
    df_all = pd.concat((df_all, df_all_dummy), axis=1)



# In[ ]:

# Splitting train and test

X = df_all.iloc[:piv_train,:]
le = LabelEncoder()
y = le.fit_transform(labels)
```

```
test = df_all.iloc[piv_train:,:]



# In[ ]:


def test_classifier(X_train, y_train, X_test, y_test, classifier):
    log("")
    log("=============================================")
    classifier_name = str(type(classifier).__name__)
    log("Testing " + classifier_name)
    now = time()
    list_of_labels = sorted(list(set(y_train)))
    model = classifier.fit(X_train, y_train)
    log("Learing time {0}s".format(time() - now))
    now = time()
    predictions = model.predict(X_test)
    log("Predicting time {0}s".format(time() - now))


    precision = precision_score(y_test, predictions, average=None, pos_label=None,
labels=list_of_labels)
    recall = recall_score(y_test, predictions, average=None, pos_label=None,
labels=list_of_labels)
    accuracy = accuracy_score(y_test, predictions)
    f1 = f1_score(y_test, predictions, average=None, pos_label=None, labels=list_of_labels)
    log("=================== Results ===================")
    log("F1       " + str(f1))
    log("Precision" + str(precision))
    log("Recall   " + str(recall))
    log("Accuracy " + str(accuracy))
```

```
    log("================================================")

    return precision, recall, accuracy, f1

def log(x):
    #can be used to write to log file
    print(x)



# In[ ]:

def cv(classifier, X_train, y_train):
    log("================================================")
    classifier_name = str(type(classifier).__name__)
    now = time()
    log("Crossvalidating " + classifier_name + "...")
    accuracy = [cross_val_score(classifier, X_train, y_train, cv=8, n_jobs=-1)]
    log("Crosvalidation completed in {0}s".format(time() - now))
    log("Accuracy: " + str(accuracy[0]))
    log("Average accuracy: " + str(np.array(accuracy[0]).mean()))
    log("================================================")
    return accuracy



# In[ ]:

# Random Forrest Classifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.7, stratify=y)

precision, recall, accuracy, f1 = test_classifier(X_train, y_train, X_test, y_test,
RandomForestClassifier())




# In[ ]:


# rf_acc = cv(RandomForestClassifier(),X, y)




# In[ ]:


rf = RandomForestClassifier()

rf.fit(X, y)

y_pred = rf.predict_proba(test)




# In[ ]:


# Gradient Boosting Classifier


from sklearn.ensemble import GradientBoostingClassifier


X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.7, stratify=y)

precision, recall, accuracy, f1 = test_classifier(X_train, y_train, X_test, y_test,
GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1))
```

# In[ ]:

```
# gbm_acc = cv(GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1),X, y)
```

# In[ ]:

```
gbm = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)

gbm.fit(X, y)

y_pred = nb.predict_proba(test)
```

# In[ ]:

```
# ADA Boost Classifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,

                                    train_size=0.7, stratify=y)

precision, recall, accuracy, f1 = test_classifier(X_train, y_train, X_test, y_test,
AdaBoostClassifier(n_estimators=100, base_estimator=
DecisionTreeClassifier(),learning_rate=1))
```

# In[ ]:

```
# ada_acc = cv(AdaBoostClassifier(n_estimators=100, base_estimator=
DecisionTreeClassifier(),learning_rate=1),X, y)



# In[ ]:


ada = AdaBoostClassifier(n_estimators=100, base_estimator=
DecisionTreeClassifier(),learning_rate=1)

ada.fit(X, y)

y_pred = nb.predict_proba(test)



# In[ ]:


# XGBoost Classifier


X_train, X_test, y_train, y_test = train_test_split(X, y,

                                    train_size=0.7, stratify=y)

precision, recall, accuracy, f1 = test_classifier(X_train, y_train, X_test, y_test,
XGBClassifier())



# In[ ]:


# xgb_acc = cv(XGBClassifier(),X, y)



# In[ ]:


# Classifier
```

```
xgb = XGBClassifier(max_depth=6, learning_rate=0.3, n_estimators=25,
                    objective='multi:softprob', subsample=0.5, colsample_bytree=0.5, seed=0)
xgb.fit(X, y)
y_pred = xgb.predict_proba(test)



# In[ ]:


# Tuning the XGBoost Classifier parameters


def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            log("Model with rank: {0}".format(i))
            log("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                results['mean_test_score'][candidate],
                results['std_test_score'][candidate]))
            log("Parameters: {0}".format(results['params'][candidate]))
            log("")


def best_fit(X_train, y_train, n_iter=5):


    parameters = {
        "n_estimators":[25,103,201, 403],
        "max_depth":[3,10,15, 30],
        "objective":["multi:softmax",'multi:softprob'],
```

```
        "learning_rate":[0.05, 0.1, 0.15, 0.3]
    }

    rand_search =
RandomizedSearchCV(XGBoostClassifier(seed=seed),param_distributions=parameters,
                                n_iter=n_iter,scoring="accuracy",
                                n_jobs=-1,cv=8)

    import time as ttt
    now = time()
    log(ttt.ctime())
    rand_search.fit(X_train, y_train)
    report(rand_search.cv_results_, 10)
    log(ttt.ctime())
    log("Search took: " + str(time() - now))



# In[ ]:


# best_fit(data_model.iloc[:, 1:], data_model.iloc[:, 0], n_iter=10)



# In[ ]:


# Taking the 5 classes with highest probabilities

ids = []  #list of ids
cts = []  #list of countries
for i in range(len(id_test)):
```

```
    idx = id_test[i]
    ids += [idx] * 5
    cts += le.inverse_transform(np.argsort(y_pred[i])[::-1])[:5].tolist()



# In[ ]:


# Generate submission


sub = pd.DataFrame(np.column_stack((ids, cts)), columns=['id', 'country'])
sub.to_csv('submission_xgb.csv',index=False)
```