Name: Vipin Singh
Email: vipsingh@iu.edu

Applied Data Science

Airbnb Data Science Report

Final Assignment

Introduction

Airbnb is an online market place to lease or rent short term lodging including vacation rentals and hotel rooms. It has become very popular in last couple of years. Airbnb does not own any lodging, it is a broker and earn commission from both guests and hosts for every booking. Every day thousands of users are booking places to stay in more than 65000 cities in approximately 191 countries. These users can do their booking at different places in different countries. This report shows how can we predict a country of stay for a new airbnb user booking.

Airbnb can better forecast the demands, decrease the average time of the booking and can share more personalize content in their community by accurately predicting the new users travel country. The Data which will be used to predict and train the prediction models is available on kaggle. There are various methods to do data exploration, each will be discussed along with the data visualization. The result section will cover any specific findings in the data. In the discussion section, important finding will be explained. Finally, the conclusion will provide the answer how accurately and precisely the country of a new user booking can be predicted by different algorithms.

Data

There are 5 datasets available to download at kaggle site, train_users_2, test_users, sessions, age and countries. As a part of this excericse I will be only using the train_users_2 data set. This data set contains 213451 users from Airbnb. The data is gathered in this set based on the first recorded visit of the user. I have done the initial analysis of the train_users_2 by using Tableau. The Visualizations are shown in the Appendix. Fig 1.1 represents the gender distribution across the data, the gender is unknown for many users. It is also evident from Fig 1.2 that most of the user's destination is NDF (no destination found) and US. As per Fig 1.3, most of the airbnb users are between the age of 30 and 65 years. I observed that the trends of booking in years from 2010 to 2015 it seems, as per

Name: Vipin Singh
Email: vipsingh@iu.edu

Fig 1.4, that the booking seems to be higher in summer season and it has reached its peak in the year 2014. More data exploration is done to identify the app usage, age and the devices used for booking, refer Fig 1.5 and Fig 1.6. The interesting observations is that the most countries booked are NDF and US. Interestingly there were some date fields which needs to be handled properly. The Age field is little off which need some preprocessing like handling the outlier in the age by converting it to proper age.

Methods

A supervised machine learning approach is required for this type of dataset. I have used python to do the analysis of the data and to compare the performance of the machine learning algorithms on training data set. Supervised machine learning is a technique where computer uses test data with known outcomes to build a model which can be applied to a data where the outcomes are unknown. I have used few classification algorithms like XGBoost, Naïve Bayes, Random Forest, Logistic Regression and Neural Network to compare the performance.

To apply the algorithms and calculate the performances, per Fig 1.7, I have loaded the train_users_2.csv in a df_train_data data frame. This data is then split in to 80:20 train and test in the data frames df_train and df_test. The split is done using the iloc for data frames. The column country_destinations of train and test are stored in labels_train and labels_test for later usage in the script. As per Fig 1.8, since the country destination of train and test is saved the column country_destination is dropped from df_train and df_test data frames. The data frame data_test is split using the iloc hence we need to reset its index. The train and test data frames are concatenated to create a new df_all data frame. This data frame is processed to fix the age column and to remove the columns id and date_first_booking, as they don't seem to be so important. The nan values are filled with -1 in df_all data frame. It is shown in Fig 1.9, how the columns date_account_created, timestamp_first_active are split in the year, month and day. The other columns in variable ohe_feats are also modified to generate the sparse matrix using pandas get_dummies. Now the sparse matrix df_all is again split back to df_train_tdm and df_test_tdm. As shown in Fig 1.10, a LabelEncoder is used to transform the labels_train values, these values are the labels of the splitted training data set from the full training data set. This transformation is done to pass the class of the training data set to train the model. A matrix dtrain is created using the df_train_tdm and y, which is passed to train method of XGB to train the model.

Once the model is trained, a feature score Fig 1.11, is generated to understand the important features in the data set. The important features are used to dice the df_all data frame and then again df_train_tdm and df_test_tdm are generated to consider only the important features.

Name: Vipin Singh
Email: vipsingh@iu.edu

XGB Boost

This model is trained with the training set with important features see Fig 1.12. The model is saved as xgboost_train and it predicts on the values in df_test_tdm using xgb.predict method. The two functions get_predicted_vals and convert_2_binary are created to calculate the performance numbers.

get_predicted_vals

This function accepts the predicted values from the model to inverse transform the predicted values. The output of this function will be a list having the predicted destination countries.

convert_2_binary

I have created this function to run the score function. It accepts two parameters true_val and predicted. When I ran the score function for multiclass variables, it gave the error of expecting a binary. I referred the QA video from JT and he suggested that we can convert the parameters to binary using a loop and setting the value to 1 if the output matches between actual and predicted otherwise to 0. I used the two lists label_true and labe_predicted. I assumed all 1's in label_true and updated the label predicted as per the match and non-match.

Naïve Bayes

The Gaussian Naïve Bayes classifies is used to fit the training data df_train_tdm along with the train class y. This model is then used to predict the test data in df_test_tdm. The output y_nb_pred is then passed to function get_predicted_vals to get the predicted countries of destinations for test data. The function f1_score is called to show the score based on the actual and predicted multiclass output. To see the precision, recall and fscore, the convert_2_binary function is called to convert the input of score function from multiclass to binary ref Fig 1.13

Random Forest

The RandomForestClassifier is used to fit the training data df_train_tdm along with the train class y. This model is then used to predict the test data in df_test_tdm using predict_proba function. The output y_rf_pred is then passed to function get_predicted_vals to get the predicted countries of destinations for test data. The function f1_score is called to show the score based on the actual and predicted multiclass output. To see the precision, recall and fscore, the convert_2_binary function is called to convert the input of score function from multiclass to binary ref fig 1.13.

Logistic Regression

Name: Vipin Singh
Email: vipsingh@iu.edu

The LogisticRegression classifier is used to fit the training data df_train_tdm along with the train class y. This model is then used to predict the test data in df_test_tdm using predict_proba function. The output y_lr_pred is then passed to function get_predicted_vals to get the predicted countries of destinations for test data. The function f1_score is called to show the score based on the actual and predicted multiclass output. To see the precision, recall and fscore, the convert_2_binary function is called to convert the input of score function from multiclass to binary ref fig 1.13.

Neural network

The sklearn neural_network classifier is used to fit the training data df_train_tdm along with the train class y. This model is then used to predict the test data in df_test_tdm using predict function. The output y_nn_pred is then passed to function get_predicted_vals to get the predicted countries of destinations for test data. The function f1_score is called to show the score, based on the actual and predicted multiclass output. To see the precision, recall and fscore, the convert_2_binary function is called to convert the input of score function from multiclass to binary ref fig 1.14.

Results

The Results are shown below from best to worst perfromance numbers. If we observe the below results the XGB is the best algorithm for this data set. This algorithm is also very good in terms of execution time. Random forest is the next to XGB but little slow in execution time compared to it. The Logistic Regression is next to Random Forest, execution time wise it is same as Random Forest. Naïve Bayes and Neural network are not the best algorithm for this type of data set, if we see timings Naïve bayes is fast compared Random Forest and Logistic Regression but its score is bad. In all the algorithms tested in this report neural network is slowest and does not have good score.

XGB Performance

XGB Score 0.655961583509
Precision: 1.0
Recall: 0.656
F-Score: 0.792

Random Forest Performance

Random Forest Score: 0.624502225346
Precision: 1.0
Recall: 0.625
F-Score: 0.769

Logistic Regression Performance

Name: Vipin Singh
Email: vipsingh@iu.edu

Logistics Regression Score: 0.612063715156
Precision: 1.0
Recall: 0.612
F-Score: 0.759

Naive Bayes Performance
Naive Bayes Score -> 0.00180370110096
Precision: 1.0
Recall: 0.0018
F-Score: 0.0036

Neural Network Performance
Nueral Network Score: 0.00180370110096
Precision: 1.0
Recall: 0.0018
F-Score: 0.0036

Discussion

I want to highlight few challenges which I faced in analyzing and deciding the algorithms. The first challenge which I faced was how to get the score of multiclass classification data set. Tried few algorithms, SVM was very slow for this data type, so ignored it. I learned an interesting feature of XGB to get the scores of the features in the data set, this helped to ignore the less important features which fitting the model. One important observation is that, there is a class imbalance in the dataset, more than 50% of cases are NDF (no destination found). The machine learning algorithm are the best when the data set is balanced and have enough data for each class to train the model.

Conclusion

We can finally conclude that the XGB algorithm, based on the scores, is the best algorithm to solve this problem. I think XGB Boost passed all the test which we have done so far, it performed well in terms of execution time and score. I have observed some of the predictions on the test data, it seems that algorithms are predicting more NDF, other and US as the country of destination for a new user. This is because of class imbalance in the data set, another way to analyze the data can be by filtering out the classes which are creating imbalance and study the performance of various algorithms. I created another script to apply the XGB algorithm to the actual test data and generated the file name Prediction_on_Test_data.csv to understand the output values ref Fig 1.15 and Fig 1.16. It was a great experience to learn different algorithms and understand their performance on the multi class data set.

Name: Vipin Singh
Email: vipsingh@iu.edu

References

https://kth.diva-portal.org/smash/get/diva2:1108334/FULLTEXT01.pdf

Apppendix

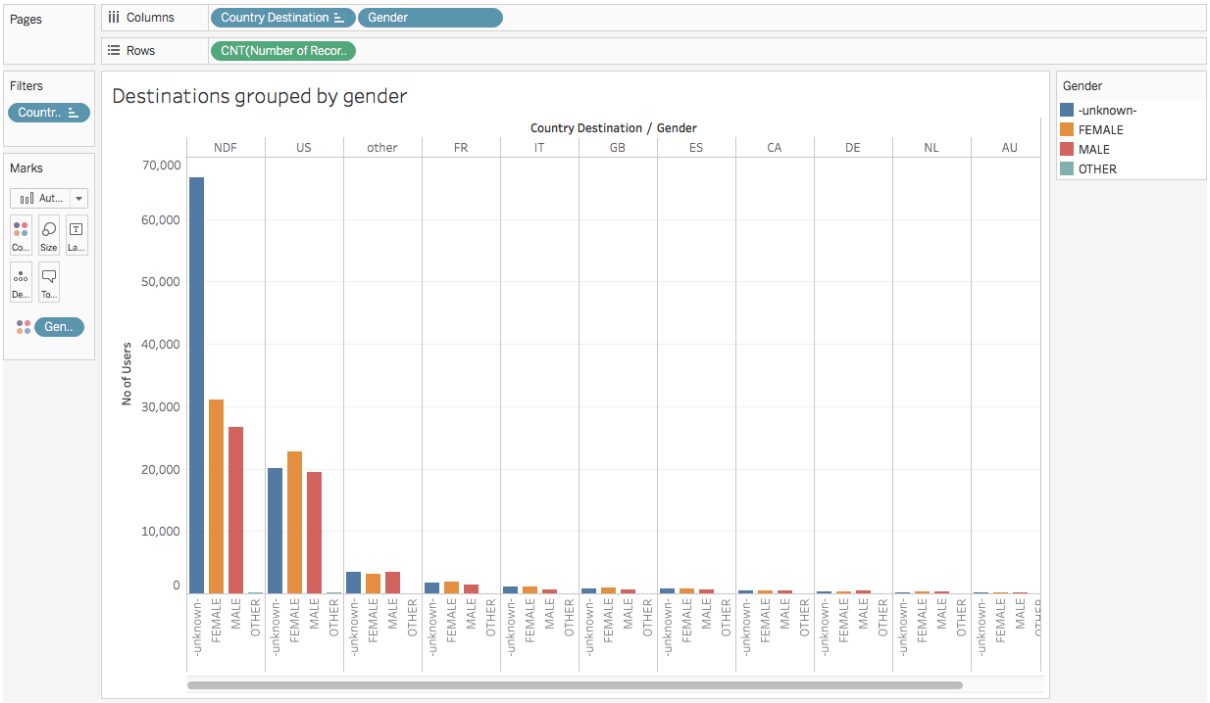


Fig 1.1 This Figure shows how genders of users are distributed across various countries

Name: Vipin Singh
Email: vipsingh@iu.edu

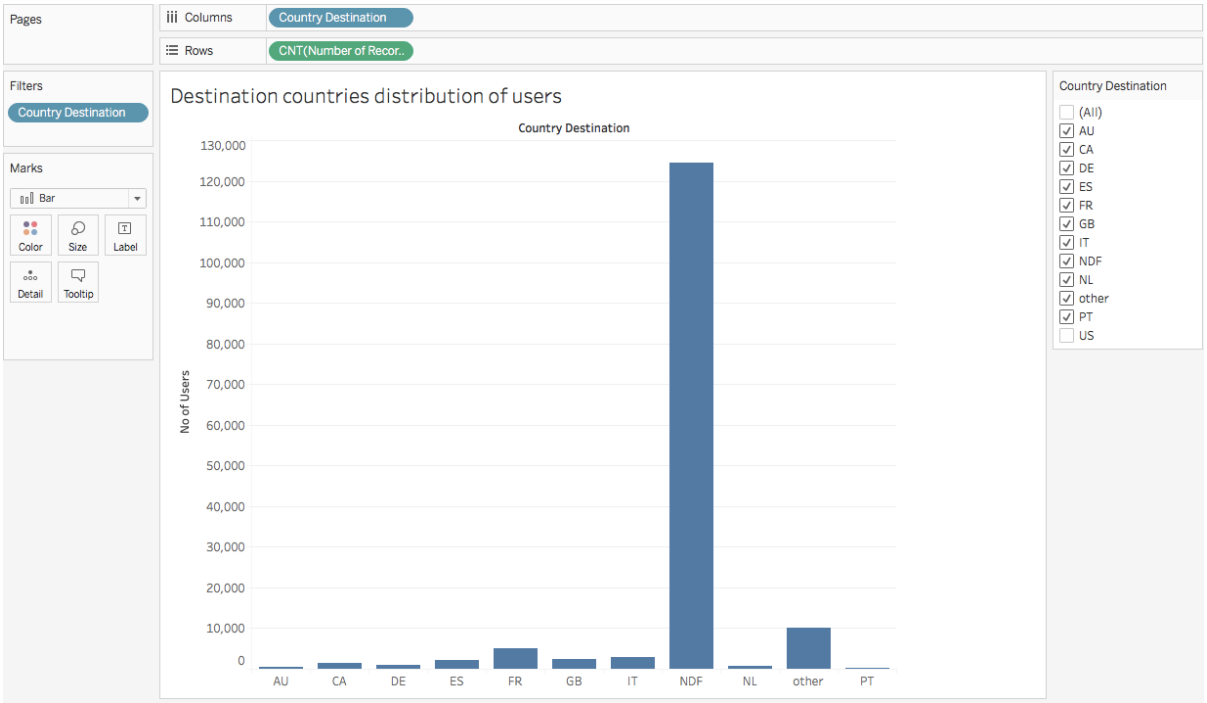


Fig1.2 This figure shows maximum chosen destination country

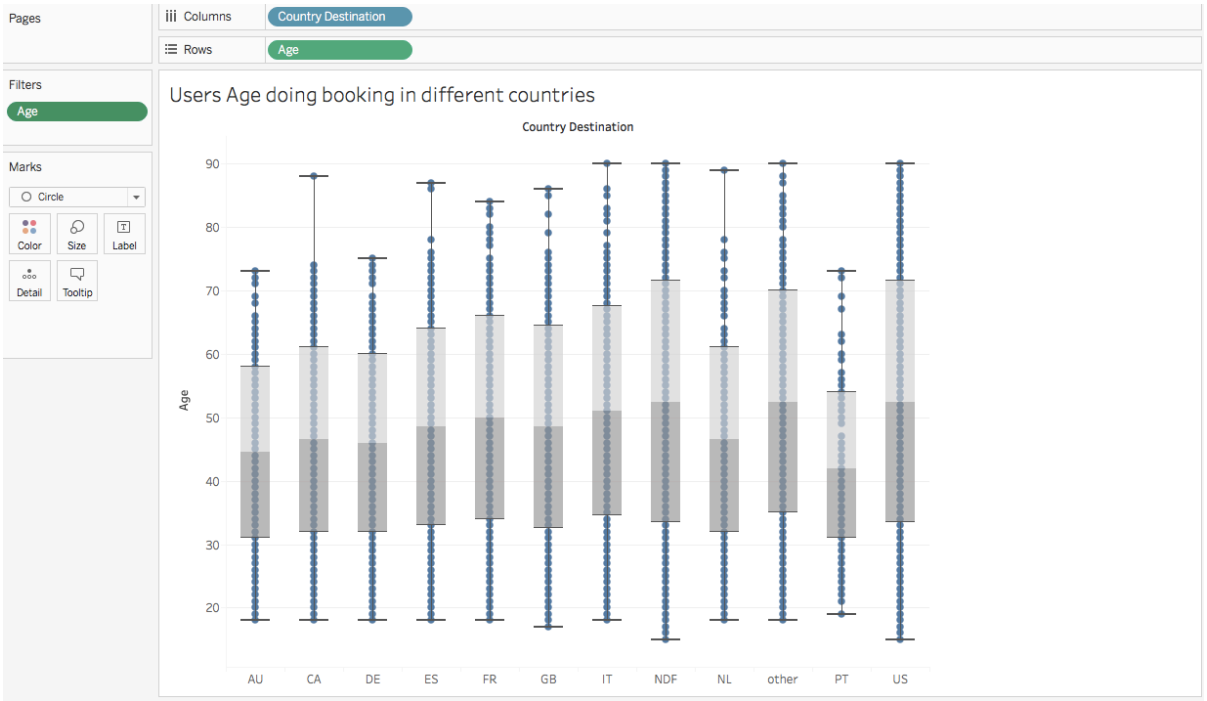


Fig 1.3 Grouped by various countries and the age of the users

Name: Vipin Singh
Email: vipsingh@iu.edu

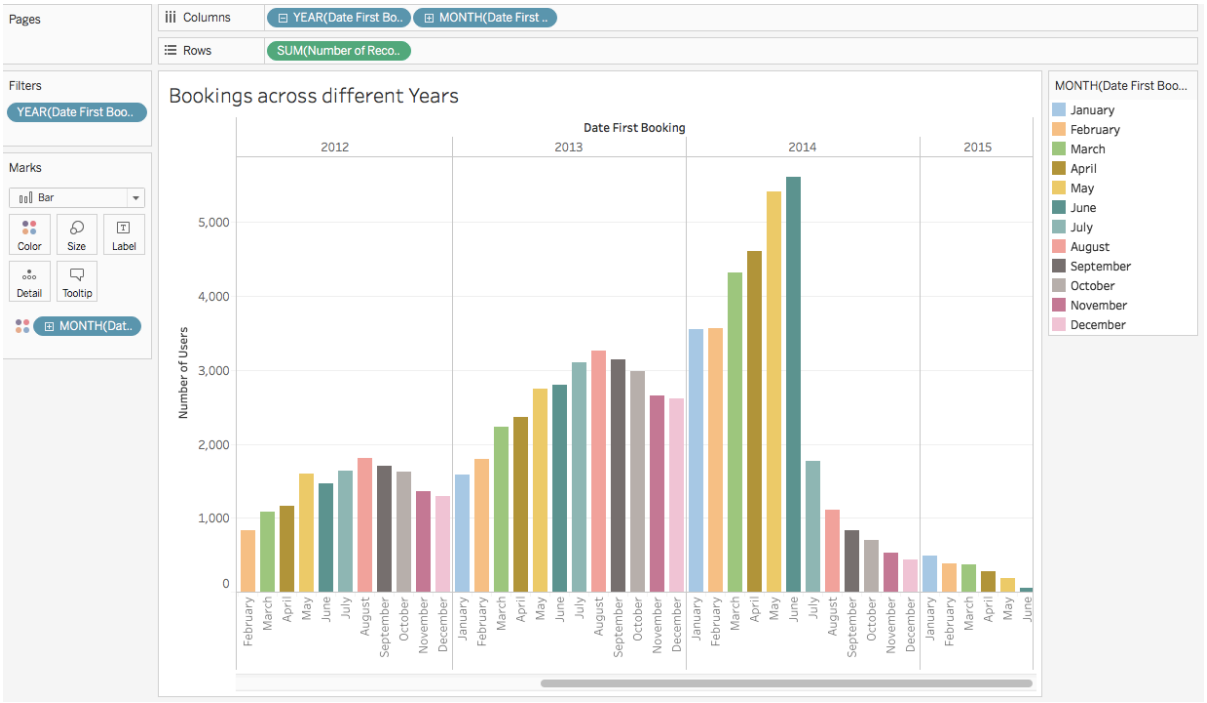


Fig 1.4 The booking trend in various years grouped by month

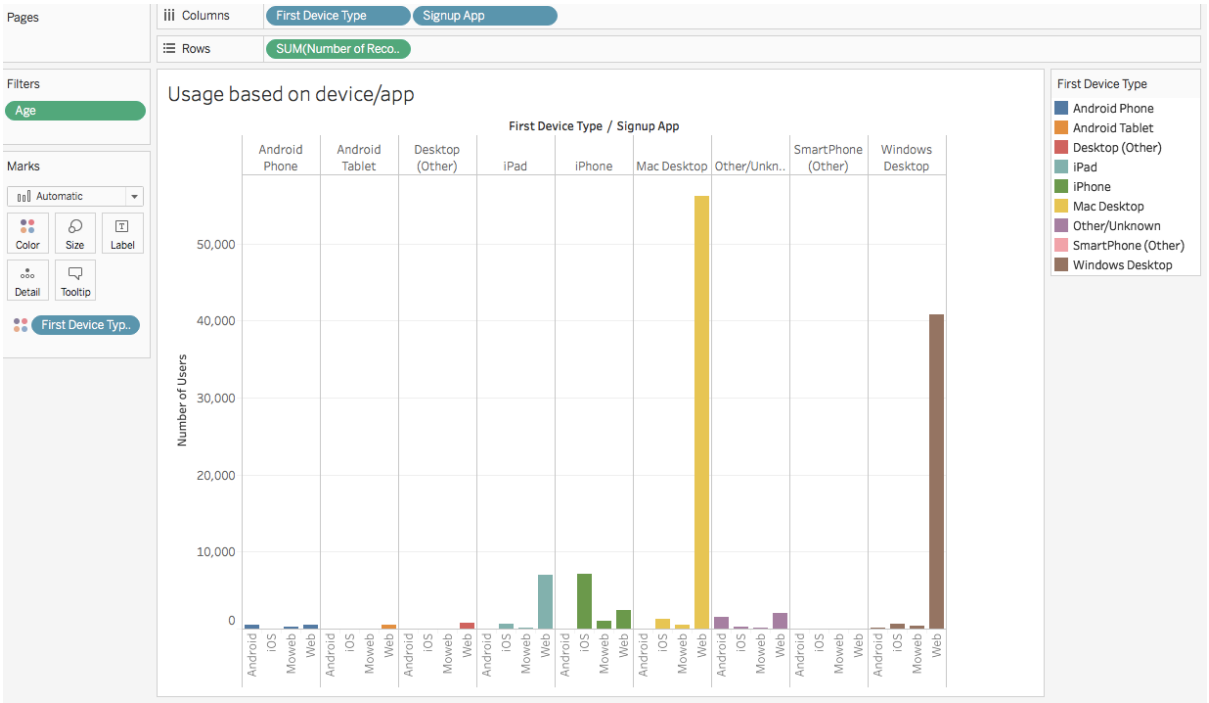


Fig 1.5 Apps used per device by users

Name: Vipin Singh
Email: vipsingh@iu.edu

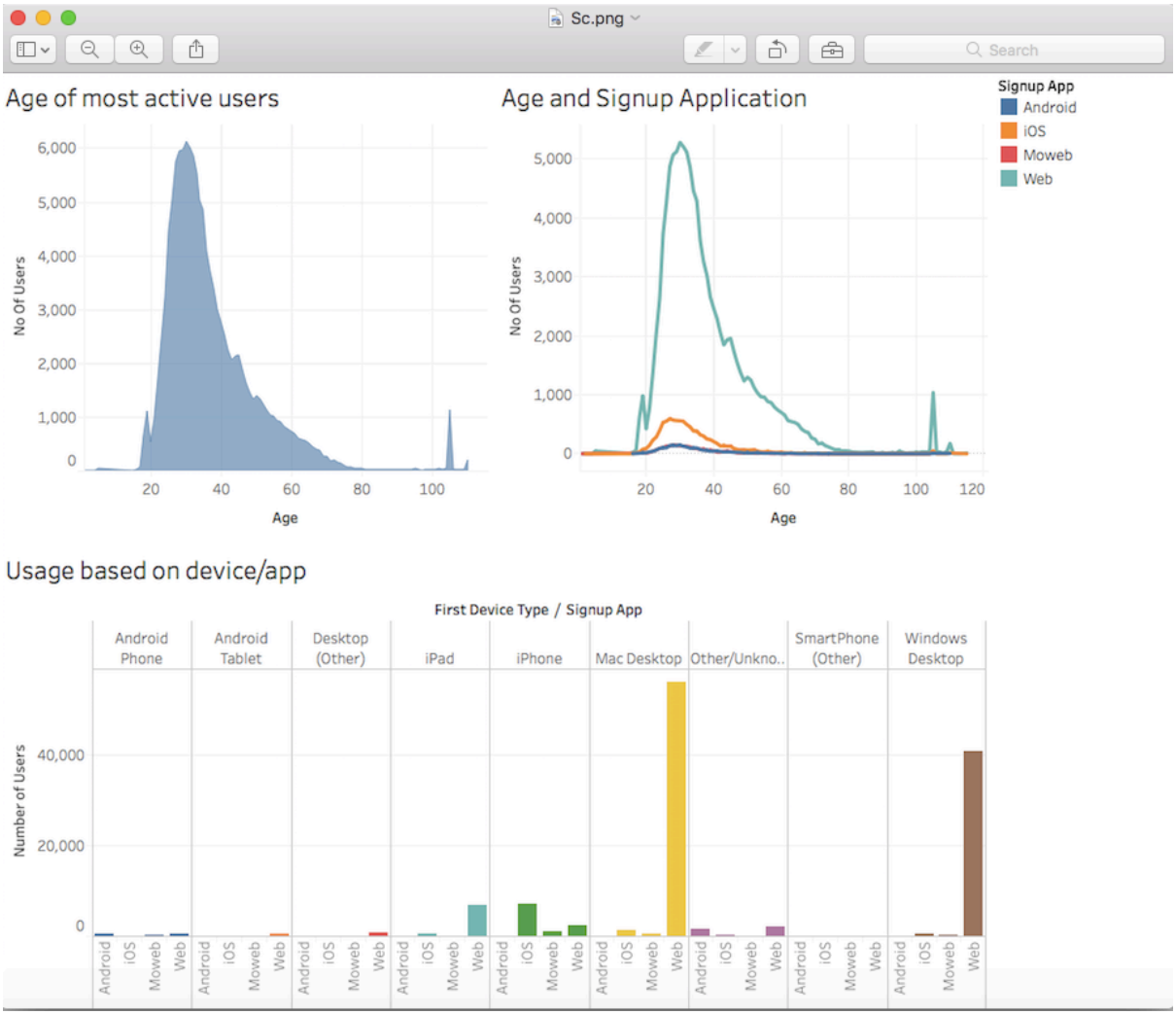


Fig 1.6 Age of most active users, user's app for booking and users per devices and apps

Name: Vipin Singh
Email: vipsingh@iu.edu

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, f1_score, recall_score,mean_squared_error
from sklearn.model_selection import train_test_split
import xgboost as xgb
import operator

def score(true,pred):
  return (precision_score(true,pred),
          recall_score(true,pred),
          f1_score(true,pred))

def print_score(s):
  print("""
Precision:  {:0.3}
Recall:     {:0.3}
F-Score:    {:0.3}
""".format(*s))

# Loading data
df_train_data = pd.read_csv('train_users_2.csv')
# Getting a label at 80% test data
piv_train = round(df_train_data.shape[0]*0.8)

# Printing the Value
print('piv_train -', piv_train)

# Splitting the Training and Test Data
df_train = df_train_data.iloc[:piv_train,:]
df_test  = df_train_data.iloc[piv_train:,:]

# Printing the shape of train and test data
print("df_train.shape -->" ,df_train.shape)
print("df_test.shape -->" ,df_test.shape)

# Storing the labels of train and test data in labels_train and labels_test
# These Values are stored since country destination will be dropped from
# df_train, and labels_test will be used as true values
labels_train = df_train['country_destination'].values
labels_test = df_test['country_destination'].values

```

Fig 1.7 shows how the train_users_2.csv is split in train and test using iloc

Name: Vipin Singh
Email: vipsingh@iu.edu

```
# Dropping the country destination from train and test data
df_train = df_train.drop(['country_destination'], axis=1)
df_test  = df_test.drop(['country_destination'], axis=1)

# Print the shapes

print("df_train.shape after drop of country destination" ,df_train.shape)
print("df_test.shape after drop of country destination" ,df_test.shape)

#
df_test = df_test.reset_index(drop=True)

## Check First three Rows
# df_test.iloc[:3]

## Getting the id's from df_test
id_test = df_test['id']

# Check first three rows of id
# id_test.iloc[:3]

## Creating a DataFrame With train and test data
df_all = pd.concat((df_train, df_test), axis=0, ignore_index=True)
print("df_all.shape -->" ,df_all.shape)


# Handling Age

df_all.loc[(df_all['age'] > 1900), 'age'] = 2017 - df_all['age']

#Identify age which is outlier
df_all.loc[(df_all['age'] <= 14) | (df_all['age'] >= 100), 'age'] = np.NAN


##Removing id and date_first_booking to make it ready for sparse matrix
df_all = df_all.drop(['id', 'date_first_booking'], axis=1)

## Filling nan with -1
df_all = df_all.fillna(-1)

```

Fig 1.8 Age Preprocess and removing the columns id and date_first_booking

Name: Vipin Singh
Email: vipsingh@iu.edu

```
87
88 # Creating the date_account_created by splitting it in to year, month and date
89 dac = np.vstack(
90     df_all.date_account_created.astype(str).apply(
91         lambda x: list(map(int, x.split('-')))
92         ).values
93     )
94 df_all['dac_year'] = dac[:,0]
95 df_all['dac_month'] = dac[:,1]
96 df_all['dac_day'] = dac[:,2]
97 df_all = df_all.drop(['date_account_created'], axis=1)
98
99 # creating the timestamp_first_active
100 tfa = np.vstack(
101     df_all.timestamp_first_active.astype(str).apply(
102         lambda x: list(map(int, [x[:4], x[4:6], x[6:8],
103                                  x[8:10], x[10:12],
104                                  x[12:14]]))
105         ).values
106     )
107 df_all['tfa_year'] = tfa[:,0]
108 df_all['tfa_month'] = tfa[:,1]
109 df_all['tfa_day'] = tfa[:,2]
110 df_all = df_all.drop(['timestamp_first_active'], axis=1)
111
112 # Check values, first three rows
113 # dfa_all.iloc[:3]
114
115 # Use the get dummies to create the sparse matrix
116 # This code will create binary 0 or 1 baased on category
117 ohe_feats = ['gender', 'signup_method', 'signup_flow', 'language',
118              'affiliate_channel', 'affiliate_provider',
119              'first_affiliate_tracked', 'signup_app',
120              'first_device_type', 'first_browser']
121 for f in ohe_feats:
122     df_all_dummy = pd.get_dummies(df_all[f], prefix=f)
123     df_all = df_all.drop([f], axis=1)
124     df_all = pd.concat((df_all, df_all_dummy), axis=1)
125
126 print("df_all.shape -->" ,df_all.shape)
127
128
129 #We need to split the sparse metrix
130 df_train_tdm = df_all.iloc[:piv_train,:]
131 df_test_tdm  = df_all.iloc[piv_train:,:]
132
```

Fig 1.9 shows how the columns date_account_creeated, timestamp_first_active and others

Name: Vipin Singh
Email: vipsingh@iu.edu

```
135 # Using the labelencoder to transform the country destinations of
136 # training set
137 le = LabelEncoder()
138 y = le.fit_transform(labels_train)
139
140 # Print the Value
141 print("y -->" ,y.shape)
142
143 # Classifier
144 params = {'eta': 0.2,
145           'max_depth': 6,
146           'subsample': 0.5,
147           'colsample_bytree': 0.5,
148           'objective': 'multi:softprob',
149           'num_class': 12}
150 num_boost_round = 1
151
152 dtrain = xgb.DMatrix(df_train_tdm, y)
153
154 # dtrain.num_row()
155 # dtrain.num_col()
156
157 clf1 = xgb.train(params=params, dtrain=dtrain, num_boost_round=num_boost_round)
158
159 # Get feature scores and store in DataFrame
160 importance = clf1.get_fscore()
161 importance_df = pd.DataFrame(
162     sorted(importance.items(), key=operator.itemgetter(1)),
163     columns=['feature','fscore']
164     )
165 # Plot feature importance of top 20
166 importance_df.iloc[-20:,:].plot(x='feature',y='fscore',kind='barh')
167
168 print("df_test.shape -->" ,df_test.shape)
169
170 # Only select features w/ a feature score (can also specify min fscore)
171 # Retrain model with reduced feature set
172 df_all = df_all[importance_df.feature.values]
173 print("df_all.shape -->" ,df_all.shape)
174
175 #We need to split the sparse metrix
176 df_train_tdm = df_all.iloc[:piv_train,:]
177 df_test_tdm  = df_all.iloc[piv_train:,:]
178
179 print("df_train_tdm.shape -->" ,df_train_tdm.shape)
180 print("df_test_tdm.shape -->" ,df_test_tdm.shape)
```

Fig 1.10 Training the XGB Model to get the important features

Name: Vipin Singh
Email: vipsingh@iu.edu

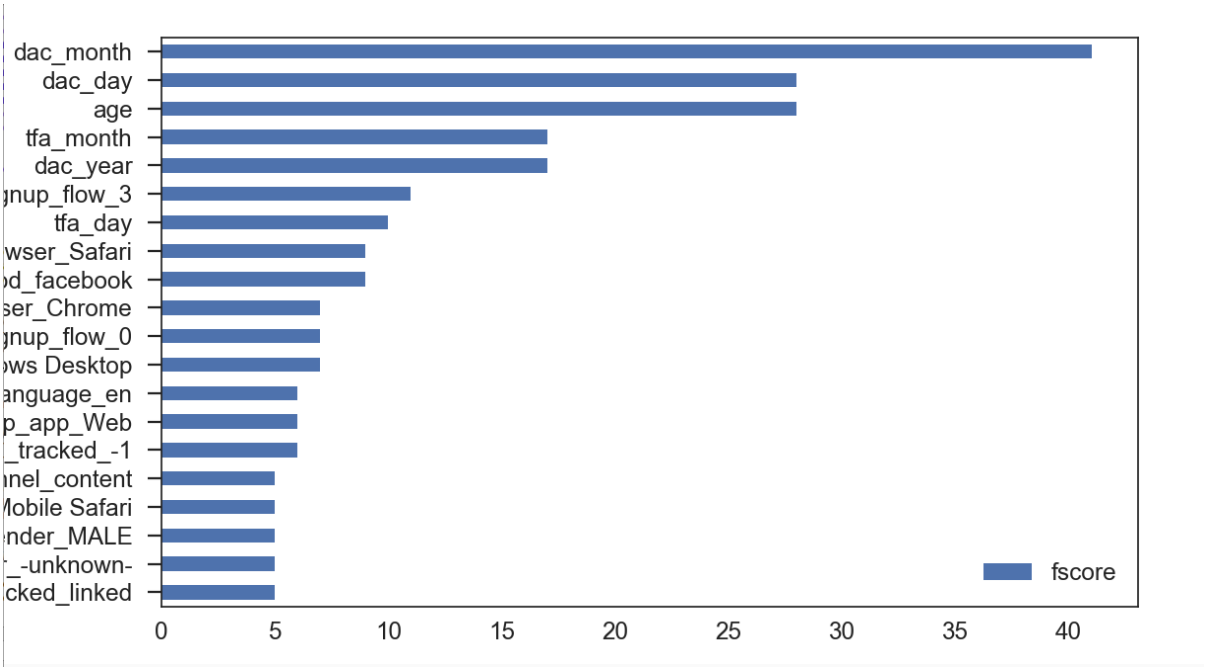


Fig 1.11 Features with the importance score

Name: Vipin Singh
Email: vipsingh@iu.edu

```
181
182 # Create a matrix to pass to the train along with the output Values
183 dtrain = xgb.DMatrix(df_train_tdm, y)
184 print("dtrain.rows -->" ,dtrain.num_row())
185 print("dtrain.cols -->" ,dtrain.num_col())
186
187 clf2 = xgb.train(params=params, dtrain=dtrain, num_boost_round=num_boost_round)
188
189 # Added this code for predict
190
191 clf2.save_model("xgboost_train")
192 bst = xgb.Booster(params)
193 bst.load_model("xgboost_train")
194
195 y_pred = bst.predict(xgb.DMatrix(df_test_tdm)).reshape(df_test.shape[0],12)
196
197 # Taking the 5 classes with highest probabilities
198 def get_predicted_vals(pred):
199     ids = []  #list of ids
200     cts = []  #list of countries
201     for i in range(len(df_test)):
202         idx = id_test[i]
203         ids += [idx] * 1
204         #print (idx,ids,i)
205         cts += le.inverse_transform(np.argsort(pred[i])[::-1])[:1].tolist()
206     return cts
207     #print (cts)
208
209 ## So cts are the predicted values for ids
210 # Check the values as cts[0]
211 # cts[1]
212
213 # Lets do a loop and ocnvert the true value into binary format, if it matches 1
214 # else 0
215 def convert_2_binary(true_val, predicted):
216     label_true = []
217     label_predicted = []
218     for i in range(len(cts)):
219         if true_val[i] != predicted[i]:
220             label_true.append(1)
221             label_predicted.append(0)
222         else:
223             label_true.append(1)
224             label_predicted.append(1)
225     return label_true, label_predicted
226
```

Fig 1.12 Shows the training the XGB model with important features using xgb.train and shows how it predicted the values using predict. This screen also shows two function created get_predicted_val and convert_2_binary.

Name: Vipin Singh
Email: vipsingh@iu.edu

```
226
227 print("\n\n XGB Performance")
228 cts = get_predicted_vals(y_pred)
229 print('XGB Score',f1_score(labels_test, cts, average ='micro'))
230
231 true_vals, pred_val = convert_2_binary(labels_test,cts)
232 s = score(true_vals,pred_val)
233 print_score(s)
234
235 #Naive Bayes
236 print("\n\n Naive Bayes Performance")
237 clfNB = GaussianNB()
238 clfNB.fit(df_train_tdm, y)
239 y_nb_pred = clfNB.predict(df_test_tdm)
240 cts = get_predicted_vals(y_nb_pred)
241 print('Naive Bayes Score -> ',f1_score(labels_test, cts, average='micro'))
242
243 true_vals, pred_val = convert_2_binary(labels_test,cts)
244 s = score(true_vals,pred_val)
245 print_score(s)
246
247 #Random Forest
248 print("\n\n Random Forest Performance")
249 forest = RandomForestClassifier(n_estimators=100)
250 forest.fit(df_train_tdm, y)
251 y_rf_pred = forest.predict_proba(df_test_tdm)
252 cts = get_predicted_vals(y_rf_pred)
253 forest_score = f1_score(labels_test, cts, average='micro')
254 print ("Random Forest Score: ", forest_score)
255
256 true_vals, pred_val = convert_2_binary(labels_test,cts)
257 s = score(true_vals,pred_val)
258 print_score(s)
259
260 #Logistic Regression
261 print("\n\n Logistic Regression Performance")
262 clf_lr = LogisticRegression(solver='lbfgs',multi_class='multinomial').fit(df_train_tdm,
263 y_lr_pred = clf_lr.predict_proba(df_test_tdm)
264 cts = get_predicted_vals(y_lr_pred)
265 score_lr = f1_score(labels_test, cts, average = 'micro')
266 print ("Logistics Regression Score: ", score_lr)
267
268 true_vals, pred_val = convert_2_binary(labels_test,cts)
269 s = score(true_vals,pred_val)
270 print_score(s)
```

Fig 1.13 showing the code to calculate the performances of algorithms

```
271
272 #Neural Network
273 print("\n\nNeural Network Performance")
274 from sklearn.neural_network import MLPClassifier
275 nn = MLPClassifier()
276 nn.fit(df_train_tdm, y)
277 y_nn_pred = nn.predict(df_test_tdm)
278 cts = get_predicted_vals(y_nn_pred)
279 score_nn = f1_score(labels_test, cts, average = 'micro')
280 print ("Nueral Network Score: ", score_nn)
281
282
283 true_vals, pred_val = convert_2_binary(labels_test,cts)
284 s = score(true_vals,pred_val)
285 print_score(s)
286
287
```

Fig 1.14 showing the performance of neural network

Name: Vipin Singh
Email: vipsingh@iu.edu

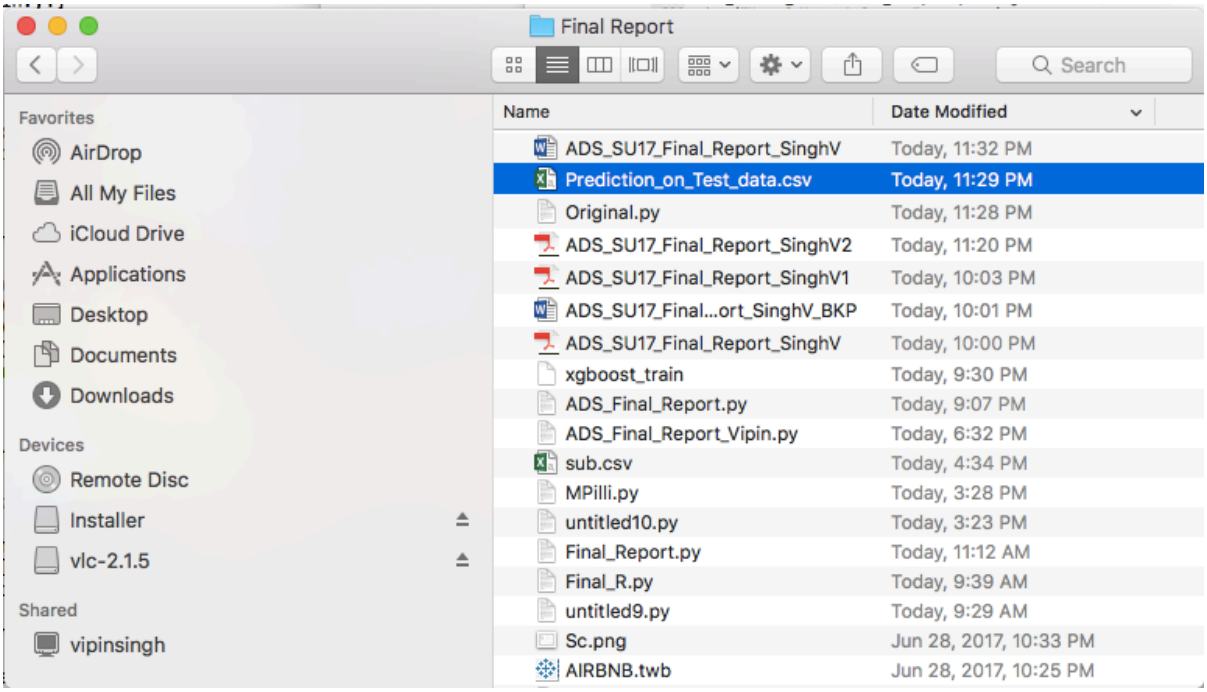
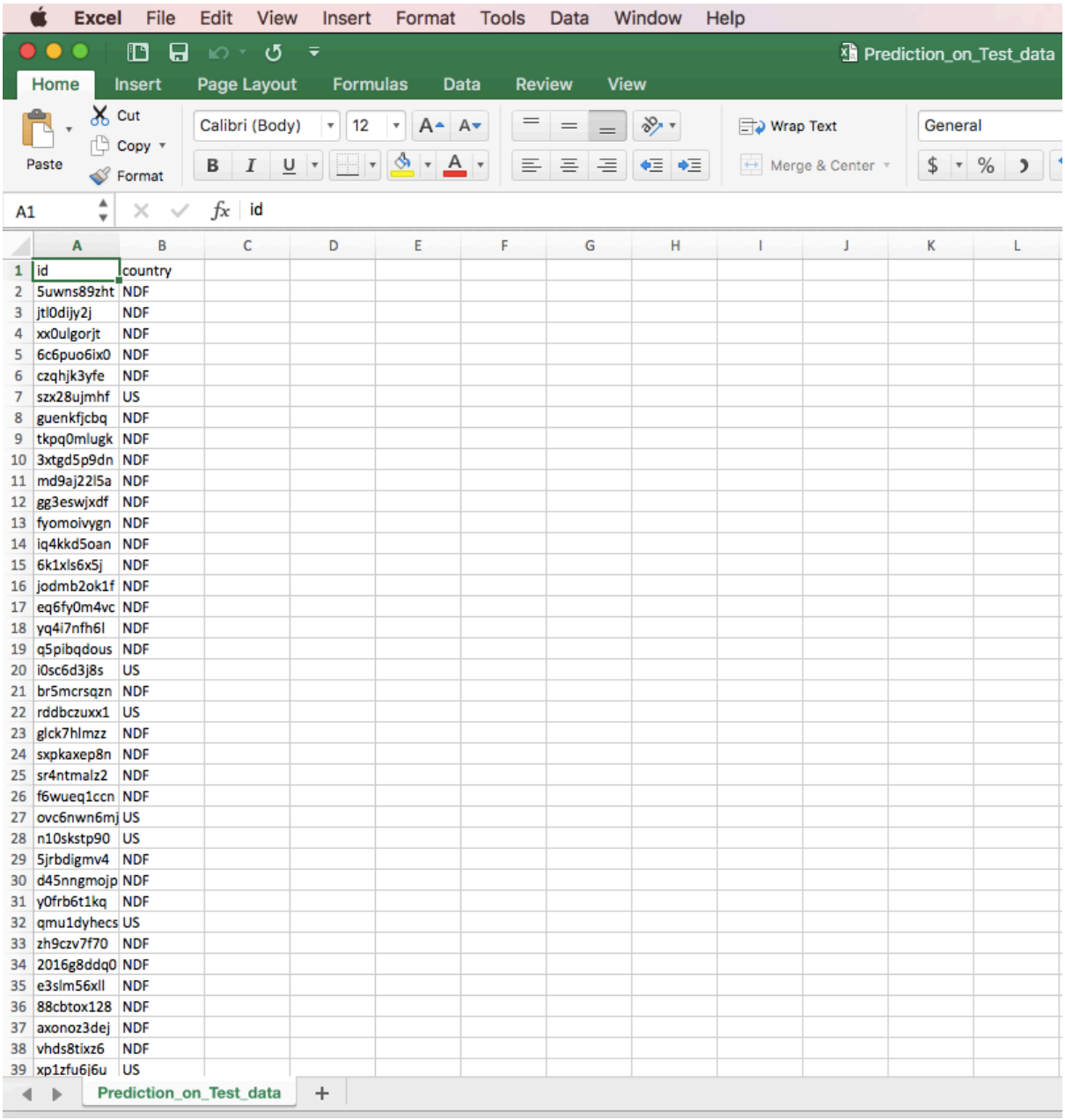


Fig 1.15 prediction file generated on actual test data

Name: Vipin Singh
Email: vipsingh@iu.edu



	A	B
1	id	country
2	5uwns89zht	NDF
3	jtl0dijy2j	NDF
4	xx0ulgorjt	NDF
5	6c6puo6ix0	NDF
6	czqhjk3yfe	NDF
7	szx28ujmhf	US
8	guenkfjcbq	NDF
9	tkpq0mlugk	NDF
10	3xtgd5p9dn	NDF
11	md9aj22l5a	NDF
12	gg3eswjxdf	NDF
13	fyomoivygn	NDF
14	iq4kkd5oan	NDF
15	6k1xls6x5j	NDF
16	jodmb2ok1f	NDF
17	eq6fy0m4vc	NDF
18	yq4i7nfh6l	NDF
19	q5pibqdous	NDF
20	i0sc6d3j8s	US
21	br5mcrsqzn	NDF
22	rddbczuxx1	US
23	glck7hlmzz	NDF
24	sxpkaxep8n	NDF
25	sr4ntmalz2	NDF
26	f6wueq1ccn	NDF
27	ovc6nwn6mj	US
28	n10skstp90	US
29	5jrbdigmv4	NDF
30	d45nngmojp	NDF
31	y0frb6t1kq	NDF
32	qmu1dyhecs	US
33	zh9czv7f70	NDF
34	2016g8ddq0	NDF
35	e3slm56xll	NDF
36	88cbtox128	NDF
37	axonoz3dej	NDF
38	vhds8tixz6	NDF
39	xp1zfu6j6u	US

Prediction_on_Test_data

Fig 1.16 Screen shot of the actual test data prediction file

Python Script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, f1_score, recall_score,mean_squared_error
from sklearn.model_selection import train_test_split
import xgboost as xgb
import operator

def score(true,pred):
  return (precision_score(true,pred),
        recall_score(true,pred),
        f1_score(true,pred))

def print_score(s):
  print("""
Precision:  {:0.3}
Recall:    {:0.3}
F-Score:   {:0.3}
""".format(*s))

# Loading data
df_train_data = pd.read_csv('train_users_2.csv')
# Getting a label at 80% test data
piv_train = round(df_train_data.shape[0]*0.8)

# Printing the Value
print('piv_train -', piv_train)

# Splitting the Training and Test Data
df_train = df_train_data.iloc[:piv_train,:]
df_test  = df_train_data.iloc[piv_train:,:]

# Printing the shape of train and test data
print("df_train.shape -->" ,df_train.shape)
print("df_test.shape -->" ,df_test.shape)

# Storing the labels of train and test data in labels_train and labels_test
# These Values are stored since country destination will be dropped from
# df_train, and labels_test will be used as true values
labels_train = df_train['country_destination'].values
labels_test = df_test['country_destination'].values

# Dropping the country destination from train and test data
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
df_train = df_train.drop(['country_destination'], axis=1)
df_test  = df_test.drop(['country_destination'], axis=1)

# Print the shapes

print("df_train.shape after drop of country destination" ,df_train.shape)
print("df_test.shape after drop of country destination" ,df_test.shape)

#
df_test = df_test.reset_index(drop=True)

## Check First three Rows
# df_test.iloc[:3]

## Getting the id's from df_test
id_test = df_test['id']

# Check first three rows of id
# id_test.iloc[:3]

## Creating a DataFrame With train and test data
df_all = pd.concat((df_train, df_test), axis=0, ignore_index=True)
print("df_all.shape -->" ,df_all.shape)


# Handling Age

df_all.loc[(df_all['age'] > 1900), 'age'] = 2017 - df_all['age']

#Identify age which is outlier
df_all.loc[(df_all['age'] <= 14) | (df_all['age'] >= 100), 'age'] = np.NAN


##Removing id and date_first_booking to make it ready for sparse matrix
df_all = df_all.drop(['id', 'date_first_booking'], axis=1)

## Filling nan with -1
df_all = df_all.fillna(-1)


# Creating the date_account_created by splitting it in to year, month and date
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
dac = np.vstack(
    df_all.date_account_created.astype(str).apply(
        lambda x: list(map(int, x.split('-')))
        ).values
    )
df_all['dac_year'] = dac[:,0]
df_all['dac_month'] = dac[:,1]
df_all['dac_day'] = dac[:,2]
df_all = df_all.drop(['date_account_created'], axis=1)

# creating the timestamp_first_active
tfa = np.vstack(
    df_all.timestamp_first_active.astype(str).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)

# Check values, first three rows
# dfa_all.iloc[:3]

# Use the get dummies to create the sparse matrix
# This code will create binary 0 or 1 baased on category
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)

print("df_all.shape -->" ,df_all.shape)

#We need to split the sparse metrix
df_train_tdm = df_all.iloc[:piv_train,:]
```

Name: Vipin Singh
Email: vipsingh@iu.edu

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

print("df_train_tdm.shape -->" ,df_train_tdm.shape)
print("df_test_tdm.shape -->" ,df_test_tdm.shape)

# Using the labelencoder to transform the country destinations of
# training set
le = LabelEncoder()
y = le.fit_transform(labels_train)

# Print the Value
print("y -->" ,y.shape)

# Classifier
params = {'eta': 0.2,
        'max_depth': 6,
        'subsample': 0.5,
        'colsample_bytree': 0.5,
        'objective': 'multi:softprob',
        'num_class': 12}
num_boost_round = 1

dtrain = xgb.DMatrix(df_train_tdm, y)

# dtrain.num_row()
# dtrain.num_col()

clf1 = xgb.train(params=params, dtrain=dtrain, num_boost_round=num_boost_round)

# Get feature scores and store in DataFrame
importance = clf1.get_fscore()
importance_df = pd.DataFrame(
   sorted(importance.items(), key=operator.itemgetter(1)),
   columns=['feature','fscore']
   )
# Plot feature importance of top 20
importance_df.iloc[-20:,:].plot(x='feature',y='fscore',kind='barh')

print("df_test.shape -->" ,df_test.shape)

# Only select features w/ a feature score (can also specify min fscore)
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
# Retrain model with reduced feature set
df_all = df_all[importance_df.feature.values]
print("df_all.shape -->" ,df_all.shape)

#We need to split the sparse metrix
df_train_tdm = df_all.iloc[:piv_train,:]
df_test_tdm  = df_all.iloc[piv_train:,:]

print("df_train_tdm.shape -->" ,df_train_tdm.shape)
print("df_test_tdm.shape -->" ,df_test_tdm.shape)

# Create a matrix to pass to the train along with the output Values
dtrain = xgb.DMatrix(df_train_tdm, y)
print("dtrain.rows -->" ,dtrain.num_row())
print("dtrain.cols -->" ,dtrain.num_col())

clf2 = xgb.train(params=params, dtrain=dtrain, num_boost_round=num_boost_round)

# Added this code for predict

clf2.save_model("xgboost_train")
bst = xgb.Booster(params)
bst.load_model("xgboost_train")

y_pred = bst.predict(xgb.DMatrix(df_test_tdm)).reshape(df_test.shape[0],12)

# Converts the Model Predicted values to list of destination countries
def get_predicted_vals(pred):
    ids = []  #list of ids
    cts = []  #list of countries
    for i in range(len(df_test)):
        idx = id_test[i]
        ids += [idx] * 1
        #print (idx,ids,i)
        cts += le.inverse_transform(np.argsort(pred[i])[::-1])[:1].tolist()
    return cts
    #print (cts)

## So cts are the predicted values for ids
# Check the values as cts[0]
# cts[1]
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
# Lets do a loop and ocnvert the true value into binary format, if it matches 1
# else 0
def convert_2_binary(true_val, predicted):
    label_true = []
    label_predicted = []
    for i in range(len(cts)):
        if true_val[i] != predicted[i]:
            label_true.append(1)
            label_predicted.append(0)
        else:
            label_true.append(1)
            label_predicted.append(1)
    return label_true, label_predicted

print("\n\n XGB Performance")
cts = get_predicted_vals(y_pred)
print('XGB Score',f1_score(labels_test, cts, average ='micro'))

true_vals, pred_val = convert_2_binary(labels_test,cts)
s = score(true_vals,pred_val)
print_score(s)

#Naive Bayes
print("\n\n Naive Bayes Performance")
clfNB = GaussianNB()
clfNB.fit(df_train_tdm, y)
y_nb_pred = clfNB.predict(df_test_tdm)
cts = get_predicted_vals(y_nb_pred)
print('Naive Bayes Score -> ',f1_score(labels_test, cts, average='micro'))

true_vals, pred_val = convert_2_binary(labels_test,cts)
s = score(true_vals,pred_val)
print_score(s)

#Random Forest
print("\n\n Random Forest Performance")
forest = RandomForestClassifier(n_estimators=100)
forest.fit(df_train_tdm, y)
y_rf_pred = forest.predict_proba(df_test_tdm)
cts = get_predicted_vals(y_rf_pred)
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
forest_score = f1_score(labels_test, cts, average='micro')
print ("Random Forest Score: ", forest_score)

true_vals, pred_val = convert_2_binary(labels_test,cts)
s = score(true_vals,pred_val)
print_score(s)

#Logistic Regression
print("\n\n Logistic Regression Performance")
clf_lr = LogisticRegression(solver='lbfgs',multi_class='multinomial').fit(df_train_tdm,y)
y_lr_pred = clf_lr.predict_proba(df_test_tdm)
cts = get_predicted_vals(y_lr_pred)
score_lr = f1_score(labels_test, cts, average = 'micro')
print ("Logistics Regression Score: ", score_lr)

true_vals, pred_val = convert_2_binary(labels_test,cts)
s = score(true_vals,pred_val)
print_score(s)

#Neural Network
print("\n\nNeural Network Performance")
from sklearn.neural_network import MLPClassifier
nn = MLPClassifier()
nn.fit(df_train_tdm, y)
y_nn_pred = nn.predict(df_test_tdm)
cts = get_predicted_vals(y_nn_pred)
score_nn = f1_score(labels_test, cts, average = 'micro')
print ("Nueral Network Score: ", score_nn)


true_vals, pred_val = convert_2_binary(labels_test,cts)
s = score(true_vals,pred_val)
print_score(s)
```

Output

Name: Vipin Singh
Email: vipsingh@iu.edu

```
>>> runfile('/Users/vipinsingh/Desktop/ADS/Final Report/ADS_Final_Report.py', wdir='/Users/vipinsingh/Desktop/ADS/Final Report')
piv_train - 170761
df_train.shape --> (170761, 16)
df_test.shape --> (42690, 16)
df_train.shape after drop of country destination (170761, 15)
df_test.shape after drop of country destination (42690, 15)
df_all.shape --> (213451, 15)
df_all.shape --> (213451, 155)
df_train_tdm.shape --> (170761, 155)
df_test_tdm.shape --> (42690, 155)
y --> (170761,)
[21:30:01] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 2 extra nodes, 0 pruned nodes, max_depth=1
[21:30:02] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 56 extra nodes, 0 pruned nodes, max_depth=6
[21:30:04] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 32 extra nodes, 0 pruned nodes, max_depth=6
[21:30:05] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 78 extra nodes, 0 pruned nodes, max_depth=6
[21:30:07] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 96 extra nodes, 0 pruned nodes, max_depth=6
[21:30:08] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 68 extra nodes, 0 pruned nodes, max_depth=6
[21:30:10] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 74 extra nodes, 0 pruned nodes, max_depth=6
[21:30:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 106 extra nodes, 0 pruned nodes, max_depth=6
[21:30:12] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra nodes, 0 pruned nodes, max_depth=4
[21:30:12] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 0 pruned nodes, max_depth=0
[21:30:14] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 102 extra nodes, 0 pruned nodes, max_depth=6
[21:30:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 94 extra nodes, 0 pruned nodes, max_depth=6
df_test.shape --> (42690, 15)
df_all.shape --> (213451, 70)
df_train_tdm.shape --> (170761, 70)
df_test_tdm.shape --> (42690, 70)
dtrain.rows --> 170761
```

Name: Vipin Singh
Email: vipsingh@iu.edu

dtrain.cols --> 70
[21:30:27] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth=6
[21:30:27] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 40 extra nodes, 0 pruned nodes, max_depth=6
[21:30:28] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 38 extra nodes, 0 pruned nodes, max_depth=6
[21:30:28] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 96 extra nodes, 0 pruned nodes, max_depth=6
[21:30:28] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 110 extra nodes, 0 pruned nodes, max_depth=6
[21:30:29] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 64 extra nodes, 0 pruned nodes, max_depth=6
[21:30:29] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 68 extra nodes, 0 pruned nodes, max_depth=6
[21:30:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 102 extra nodes, 0 pruned nodes, max_depth=6
[21:30:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra nodes, 0 pruned nodes, max_depth=6
[21:30:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 0 pruned nodes, max_depth=0
[21:30:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 112 extra nodes, 0 pruned nodes, max_depth=6
[21:30:31] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 108 extra nodes, 0 pruned nodes, max_depth=6

XGB Performance
XGB Score 0.655961583509

Precision: 1.0
Recall: 0.656
F-Score: 0.792

Naive Bayes Performance
Naive Bayes Score -> 0.00180370110096

Precision: 1.0
Recall: 0.0018

Name: Vipin Singh
Email: vipsingh@iu.edu

F-Score: 0.0036

Random Forest Performance
Random Forest Score: 0.625837432654

Precision: 1.0
Recall: 0.626
F-Score: 0.77

Logistic Regression Performance
Logistics Regression Score: 0.612063715156

Precision: 1.0
Recall: 0.612
F-Score: 0.759

Neural Network Performance
Nueral Network Score: 0.00180370110096

Precision: 1.0
Recall: 0.0018
F-Score: 0.0036