New User booking prediction on Airbnb dataset

Sri Naga Sai Sushma Kondabolu
Master's in Data Science, Northeastern University
Kondabolu.s@northeastern.edu

Abstract

To share more personalized content with users, decrease the average time of first booking and forecast demand, the Airbnb user's dataset was made available for predicting a new user's first booking destination. This prediction is made by the training the models using kNN, Decision Tree and Random Forest Classifier algorithms and choosing the best algorithm among the three to make the prediction on test users.

Introduction:

Airbnb is a globally operating company that focuses on lodging and vacation rentals through online booking. They have a lot of registered and unregistered users who have generate activity on their website or app in a daily manner. Here, Airbnb has provided a set of users that are split into train users and test users. The train users and test users were separated by the company based on first activity on their website after 7/1/2014. The users with first activity after the given date are the test users.

The train user's dataset has 213451 rows with fifteen features and a target variable. The target variable is the destination country for which the user makes first-ever booking. The destination countries outcomes for this dataset are limited to 'US' (United States of America), 'FR' (France), 'CA' (Canada), 'GB' (Great Britain), 'ES' (Spain), 'IT' (Italy), 'PT' (Portugal), 'NL' (The Netherlands),'DE' (Denmark), 'AU' (Australia), 'NDF' (no destination found), and 'other'. The 'NDF' value is different from 'other', as NDF means there was no booking at all. Whereas, 'other' means there was a booking to a country other than the specified ones. As described earlier there are fifteen features for the train user's dataset: id, 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 and first_browser.

The id field is unique for every user, and it is their user id. The date_account_created field is the date on which the account was created whereas the timestamp_first_active is the timestamp of the first activity on Airbnb platforms, note that it can be earlier than date_account_created or date_first_booking because a user can search before signing up. The date_first_booking is the date at which the first ever booking was made. Gender and age belong to the gender and age of the user. The signup_method corresponds to the account type which the user used to sign up and the signup_flow is the page a user came to sign up from. The language feature is the preferred international language of the user. The affiliate_channel and affiliate_provider are the kind of paid marketing and marketing medium provider the user came across. The first_affiliate_tracked is the first marketing the user interacted with before the signing up and signup_app is the application where the user signed up for the service. The first_device_type and first_browser are the first device where user activity was noticed and the web browser in which the user activity was first tracked. The same features are available with the test user's data too that has 62096 rows but without the country_destination feature as the users are new and have their destination is to be predicted.

Along with the train and test users' datasets, the sessions file, that contains the records of the web sessions of the users is provided and the countries data file that contains summary statistics of destination countries in this dataset and their locations. The age_gender_bkts file is also provided that has summary statistics of users' age group, gender, country of destination. These files are not employed for the solution developed in this project to lower the dimensionality of data and reduce computational complexity.

To predict the future user's first destination the models are trained using kNN, Decision Tree and Random Forest Classifier algorithms. To provide a uniform input to the algorithms, all the data is cleaned and converted to a numerical format. The same train user's data is split into model training data and model test data. The training chunk is to develop the model and make it learn parameters, whereas the test data is to evaluate the correctness of the learning of the algorithm.

Background:

The project involves the usage of the kNN, Decision Tree and Random Forest algorithms on the chosen dataset. Information on the implementation of the algorithms is available all over the web and the ready to use algorithm implementations are available on the Sci-kit learn library in Python. A brief introduction of the algorithms are stated below.

kNN: The kNN algorithm is known as k Nearest Neighbors algorithm. It is one of the popular and widely applied algorithms. The label for a data sample is allocated based on the majority class of its k nearest neighbors. There is no model essentially developed for training the algorithm. All the computation is performed at the time of the classification for a sample. The k nearest neighbors of the sample are based on the distance proximity to the sample. The distance can be assessed based on the different distance calculation methods like the Euclidean distance, Minkowski distance or Supremum distance among many other distance calculation methods. The most popular distance calculation is done by calculating the Euclidean distance. The choice of 'k' number of neighbors is critical for kNN. If 'k' is too small, there might be a lot of noise, whereas if 'k' is too large, there might be members of other classes interfering. It is better to choose, k = sqrt(n), where is 'n' is the number of samples in the training set when the number of samples are not too high. The kNN algorithm is readily implemented in the Sci-kit learn library of Python. Here that would be used.

Decision Tree: Decision Trees are non-parametric supervised machine learning algorithms. They are the building block of Ensemble methods. They are the basic structures for many algorithms like the Random Forest Classifier, AdaBoost, etc. They work on the strategy of splitting samples at the node of the tree based on a evaluation criteria. They split the samples until all the samples in a split have same class or there are no more samples/features to split. They can have binary split or multi way split based on our choice. The Decision Tree like kNN can be applied for classification as well as regression tasks.

Random Forest Classifier: Random Forests are a bagging method designed for Decision Trees. Each tree is built with a sample drawn with replacement and at each split the best feature to split is assessed from the selection of random features. This results in good tree diversity and reduced variance. Randomization reduces correlation between trees, this is important to reduce the generalization error.

Related Work:

The project is based on a Kaggle competition launched in the year 2016. Since then there are many solutions posted for the problem.

The majority of the works present on Kaggle used the XGBoost algorithm for training and testing the model. To assess the accuracy and power of simpler algorithms on the dataset kNN, Decision Tree and Random Forest Classifier are chosen for the problem. The accuracy of the developed model here may not be on par with the models developed using XGBoost, but they will give an idea on how to handle large datasets effectively using simple algorithms. Along with training the model using different algorithm, there is also a difference in a way the data is handled. The data has an irregular distribution, and the number of samples for certain classes are multiplied.

Project Description

The work on developing model begins at loading the datas et and inspecting it for any patterns. The visualization for t he dataset is performed for extracting data patterns and det ect any anomalies.

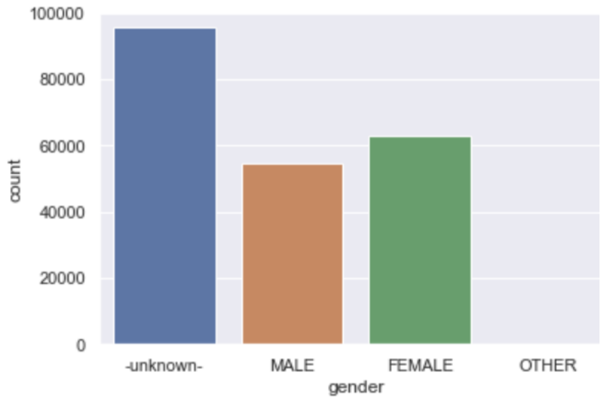


Fig 1: Distribution of train users by 'gender'

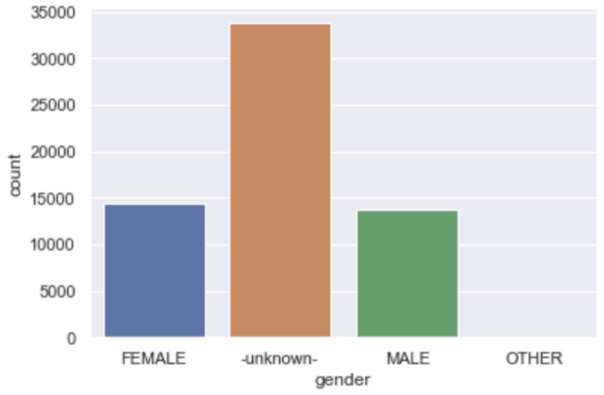


Fig 2: Distribution of test users by 'gender'

The Fig 1 illustrates the distribution of training dataset use rs based on the 'gender' feature. It is observed that close to half of the test users have not specified their gender. Amon gst the one's that specified their gender, the number of fem ales are more than male and other gender people. The same goes with the test user's dataset as well illustrated in Fig 2.

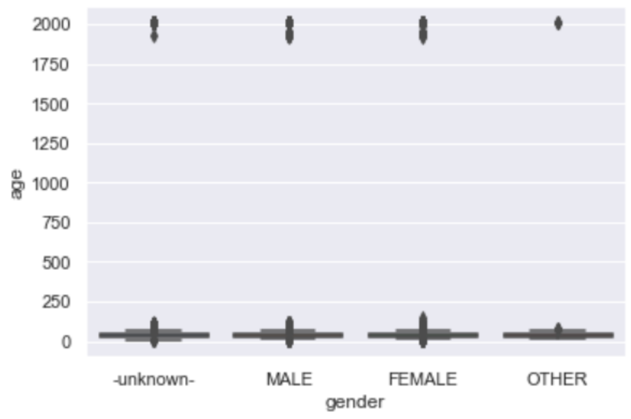


Fig 3 Age distribution of train users by ‘gender’

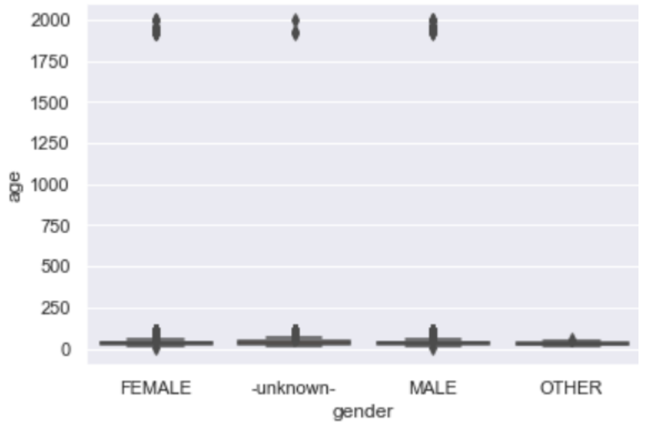


Fig 4 Age distribution of test users by ‘gender’

From Fig 3 & 4, it is comprehensible that there are good nu mber of outliers in the data for the ‘age’ field. The Fig 3 & 4 display the distribution of age of users by gender for the t rain and test users dataset respectively.

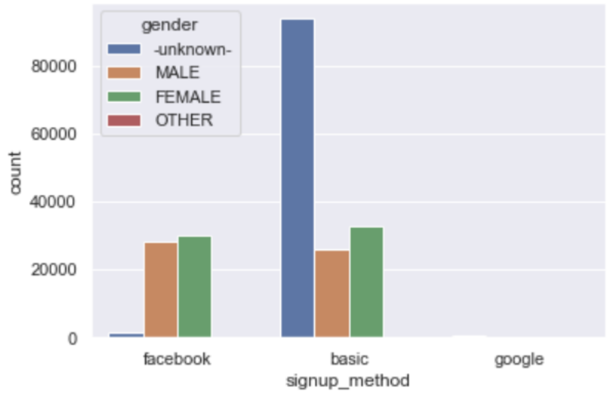


Fig 5 Distribution of ‘signup_method’ for train users based on ge nder

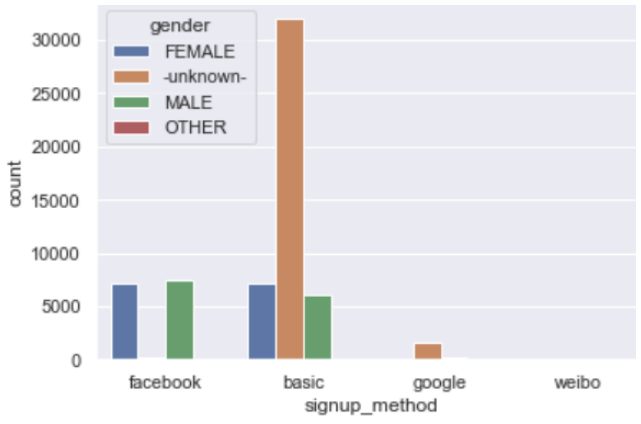


Fig 6 Distribution of ‘signup_method’ for test users based on gen der

Many people signed up using the basic method of signing i n/ registering for the service.

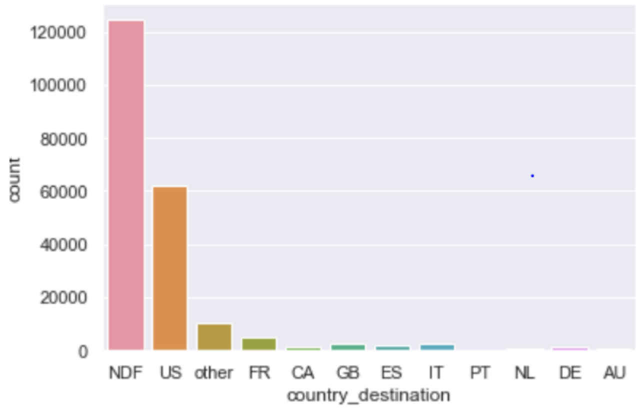


Fig 7 Distribution of the train users among destinations

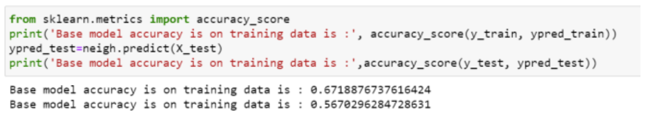
Fig 7 represents the distribution of target variable across di fferent classes. There are twelve targets available for the da taset. It can be clearly inferred that the data is highly skewe d to the right and has too less samples for other destination except for the US. This gives an opinion that the dataset is imbalanced for classification. Though the data has skewed nature, it can be tried to classify without alterations and ov ersampling/under sampling techniques can be applied, if ne cessary.

Now that there is a brief idea about the data, the next ste p is to clean it. The test user’s data is taken into considerati on and is inspected for null values. It is found that there 12 4543 null values for ‘date_first_booking’ field, this is expe cted as all the users might not have made a booking yet, so the null values for this feature aren’t imputed with any data . The ‘age’ field is also empty for 87990 users in the test us er’s dataset, the ‘age’ filed is to be handled more sensitivel y as it might impact a lot in the prediction of the users first destination, a different approach for the age field would be detailed further. The other feature that has null values is th e ‘first_affiliate_tracked’. It has 6065 null values. The null vales are replaced by ‘No data’ statement.

The same approach is applied for dealing with null values data of the test users' dataset. It has null values for the sam e features as train user's dataset but with a varying number of samples. Once the null values are handled, another step i n cleaning data is to make it more generic. To facilitate this the 'gender' feature that has '-unknown-' value is replaced with 'NotSpecified' value to make it look meaningful. The same is applied for the 'gender' field of test users data.
The next step to make data more beneficial for training the model is feature engineering. There are three date fields in the data: 'date_account_created', 'timestamp_first_active' and 'date_first_booking'. The 'date_account_created' and ' timestamp_first_active' fields are split into year, month, an d day. This will result in six additional features for the data set, that are Year_Account_Created, Month_Account_Crea ted, Day_Account_Created, Year_Timestamp_FirstActive, Month_Timestamp_FirstActive and Day_Timestamp_First Active. The 'date_account_created' and 'timestamp_first_a ctive' fields are split identically for test users dataset as wel l. Based on 'date_account_created' and 'timestamp_first_a ctive' fields another new feature 'days_difference_account' is created which gives the difference days between the time stamp when the user is active and the date when account is created.
The next feature to be handled is the 'age' feature. There ar e around 88000 null values for age already in the dataset, i nstead of using the age field directly, the users would be sp lit into age buckets which would group the users of same a ge bucket in one age range. The buckets for the age values are five years in length. There are certain assumptions mad e for assigning the age range to users, valid users are consi dered as those above fifteen years of age and below ninety years of age. Between these years the age buckets are made for every five years and assigned in the 'Age_Range' featu re. Through the visualization earlier, it is observed that ther e are a lot of outliers in the data that went up to 2014, etc. For the users that have given age between 1900 to 1999, it is assumed that they might have given their year of birth in stead of age, and their age is calculated from subtracting th e age given from year of time stamp first active, based on t his calculation they are assigned an 'Age_Range'. There ar e users who have given their age as less than fifteen years a nd greater than ninety years, these are grouped under '5-9' age range. All those users who left the age field blank i.e., have null values for age are assigned to a bucket ranging '1 1-14'. The same method is applied for the test user's data a s well. At this point, the test users data and train users data can be combined.
Now the data is clean but still has some features that are no t useful for classification, the next step would be dropping t hose features. The 'date_of_first_booking' feature is dropp ed as the test users would not be having a value for this and many of the train users have this empty. The 'id' column is also dropped as this would be unique for all users and woul d not contribute for classification. The target variable of 'c ountry_destination' is stored separately and dropped from t he main dataset.
Since many ML algorithms accept only numerical inputs, t here is a need to convert all non-numerical data features int o numbers and encode the categorical features. The pandas. get_dummies() function is chosen to convert the categorica l features to number based indicator values.
After converting the data, the test users are split from the tr ain users. The train users along with target values are again split into X_test, X_train, y_train and y_test using the skle arn.model_selection.train_test_split() functionality in sklea rn.
At first, the training was performed using the kNN algorith m with nearest neighbors as 5 on the X_test and y_train dat a. Then using RandomizedSearchCV(), the kNN algorithm was trained for nearest neighbors ranging from 7 to 12 with three folds each for the same data. Then using the best esti mator of RandomizedSearchCV, the accuracy was calculat ed on X_test. Using the best estimator, a prediction too was performed on the test user's dataset.
After using the data on kNN, the dataset was applied for te sting the Decision Tree algorithm for both criteria's of 'gin i' and 'entropy'. After testing the accuracy on the X_test, t he algorithm with entropy as criteria was used for predictin g the destination of test users' data.
Random Forest Classifier was also used to train the model with this data. The classifier was trained with 100, 200, 30 0, 700, 800, 1000, 1200 estimators. The best performing es timator was chosen for carrying out prediction for test user s' data.
The above algorithms were trained with the data given as it is. There is another training also performed by altering the number of samples of data. As quoted earlier the dataset is highly skewed with 'NDF' class samples totaling to more t han half of the dataset rows. To reduce the asymmetry, the sampled of other classes is being increased three-fold and t he duplicates of the NDF class are dropped. The test user's dataset is untouched. All the data processing techniques me ntioned earlier are applied on this enhanced dataset too. Th e kNN, DecisionTree and RandomForestClassifier algorith ms are applied on this dataset.

Empirical Results:

The kNN, DecisionTree and RandomForestClassifier are applied on the test data set.

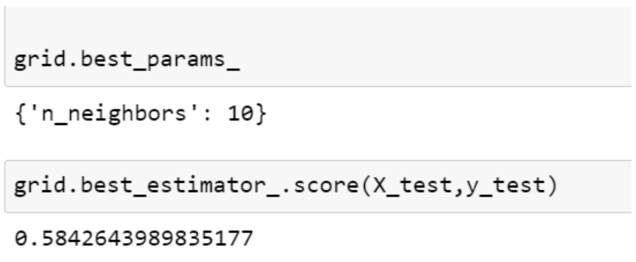


```
from sklearn.metrics import accuracy_score
print('Base model accuracy is on training data is :', accuracy_score(y_train, ypred_train))
ypred_test=neigh.predict(X_test)
print('Base model accuracy is on training data is :',accuracy_score(y_test, ypred_test))
```

```
Base model accuracy is on training data is : 0.6718876737616424
Base model accuracy is on training data is : 0.5670296284728631
```

Fig 8 kNN accuracy when neighbors=5

The accuracy using kNN on untouched data is around 67% on X_train and 56.7% percent on X_test. To check for better results, hyperparameter tuning is performed for neighbors values from 7 to 12, and best neighbor value was found to be at 10. The accuracy on X_test was found to be 58.4%.



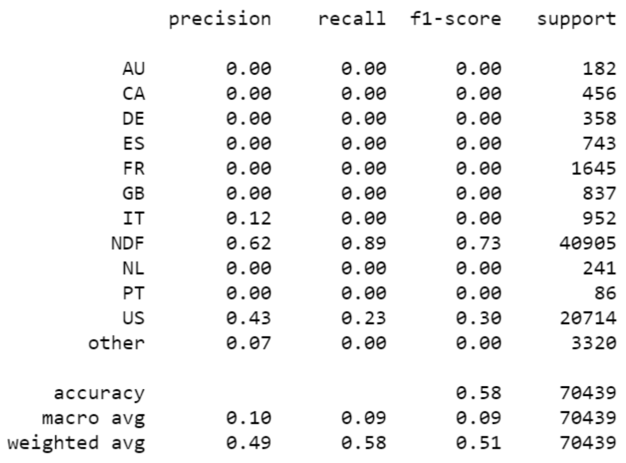
```
grid.best_params_
```

```
{'n_neighbors': 10}
```

```
grid.best_estimator_.score(X_test,y_test)
```

```
0.5842643989835177
```

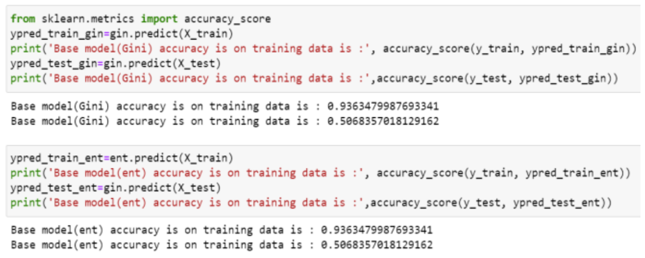
Fig 9 kNN accuracy on X_test with 10 neighbors



```
              precision    recall  f1-score   support

          AU       0.00      0.00      0.00       182
          CA       0.00      0.00      0.00       456
          DE       0.00      0.00      0.00       358
          ES       0.00      0.00      0.00       743
          FR       0.00      0.00      0.00      1645
          GB       0.00      0.00      0.00       837
          IT       0.12      0.00      0.00       952
         NDF       0.62      0.89      0.73     40905
          NL       0.00      0.00      0.00       241
          PT       0.00      0.00      0.00        86
          US       0.43      0.23      0.30     20714
       other       0.07      0.00      0.00      3320

    accuracy                           0.58     70439
   macro avg       0.10      0.09      0.09     70439
weighted avg       0.49      0.58      0.51     70439
```

Fig 10 Classification report for kNN with 10 neighbors

The classification report shows that the precision, recall and f1 score are too low for all other classes except for NDF and US



```
from sklearn.metrics import accuracy_score
ypred_train_gin=gin.predict(X_train)
print('Base model(Gini) accuracy is on training data is :', accuracy_score(y_train, ypred_train_gin))
ypred_test_gin=gin.predict(X_test)
print('Base model(Gini) accuracy is on training data is :',accuracy_score(y_test, ypred_test_gin))
```

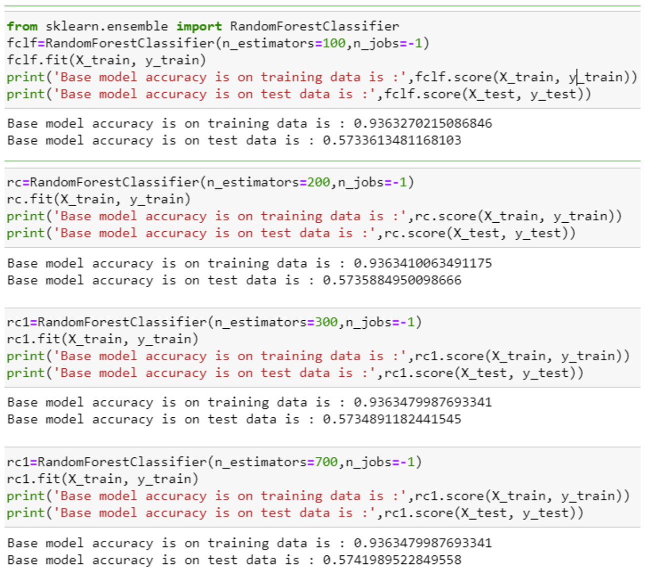
```
Base model(Gini) accuracy is on training data is : 0.9363479987693341
Base model(Gini) accuracy is on training data is : 0.5068357018129162
```

```
ypred_train_ent=ent.predict(X_train)
print('Base model(ent) accuracy is on training data is :', accuracy_score(y_train, ypred_train_ent))
ypred_test_ent=gin.predict(X_test)
print('Base model(ent) accuracy is on training data is :',accuracy_score(y_test, ypred_test_ent))
```

```
Base model(ent) accuracy is on training data is : 0.9363479987693341
Base model(ent) accuracy is on training data is : 0.5068357018129162
```

Fig 11 Accuracy of Decision Tree model on X_test and X_train using 'gini' and 'entropy' criteria's

The accuracy of the model using both the criteria's is identical, but the model has only fifty percent accuracy on X_test data.



```
from sklearn.ensemble import RandomForestClassifier
fclf=RandomForestClassifier(n_estimators=100,n_jobs=-1)
fclf.fit(X_train, y_train)
print('Base model accuracy is on training data is :',fclf.score(X_train, y_train))
print('Base model accuracy is on test data is :',fclf.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.9363270215086846
Base model accuracy is on test data is : 0.5733613481168103
```

```
rc=RandomForestClassifier(n_estimators=200,n_jobs=-1)
rc.fit(X_train, y_train)
print('Base model accuracy is on training data is :',rc.score(X_train, y_train))
print('Base model accuracy is on test data is :',rc.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.9363410063491175
Base model accuracy is on test data is : 0.5735884950098666
```

```
rc1=RandomForestClassifier(n_estimators=300,n_jobs=-1)
rc1.fit(X_train, y_train)
print('Base model accuracy is on training data is :',rc1.score(X_train, y_train))
print('Base model accuracy is on test data is :',rc1.score(X_test, y_test))
```

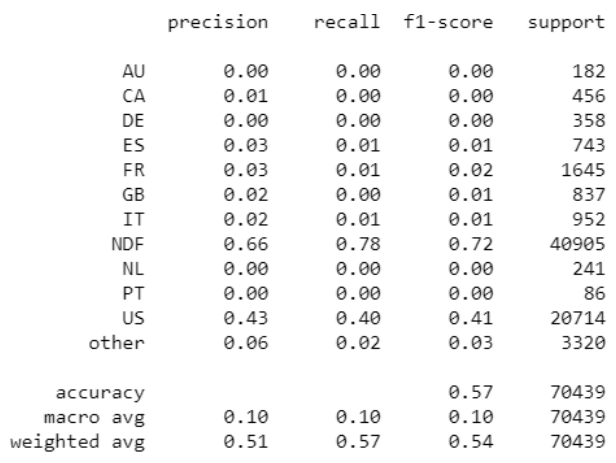
```
Base model accuracy is on training data is : 0.9363479987693341
Base model accuracy is on test data is : 0.5734891182441545
```

```
rc1=RandomForestClassifier(n_estimators=700,n_jobs=-1)
rc1.fit(X_train, y_train)
print('Base model accuracy is on training data is :',rc1.score(X_train, y_train))
print('Base model accuracy is on test data is :',rc1.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.9363479987693341
Base model accuracy is on test data is : 0.5741989522849558
```

Fig 12 Accuracies of Random Forest Classifier on different number of estimators

The Random Forest Classifier overfits the training data and has around 57% accuracy only for the test data.

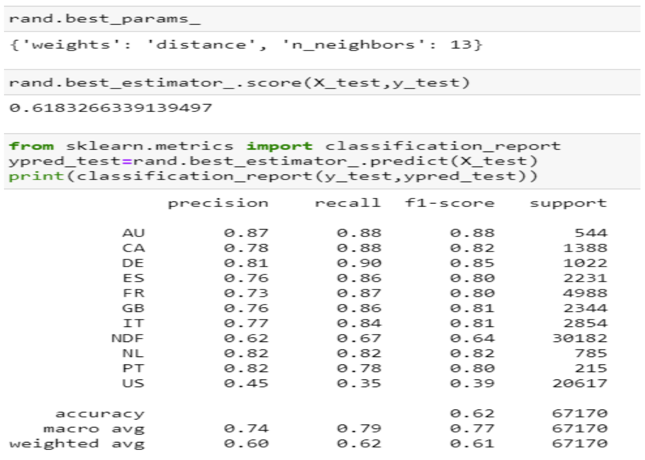


```
              precision    recall  f1-score   support

          AU       0.00      0.00      0.00       182
          CA       0.01      0.00      0.00       456
          DE       0.00      0.00      0.00       358
          ES       0.03      0.01      0.01       743
          FR       0.03      0.01      0.02      1645
          GB       0.02      0.00      0.01       837
          IT       0.02      0.01      0.01       952
         NDF       0.66      0.78      0.72     40905
          NL       0.00      0.00      0.00       241
          PT       0.00      0.00      0.00        86
          US       0.43      0.40      0.41     20714
       other       0.06      0.02      0.03      3320

    accuracy                           0.57     70439
   macro avg       0.10      0.10      0.10     70439
weighted avg       0.51      0.57      0.54     70439
```

Fig 13 Classification matrix for 700 estimators on Random Forest Classifier

As observed for kNN, the Random Forest Classifier too has low precision, recall and f1 score for classes other than NDF and US.

The upcoming results are based on the enhanced data that has oversampling of classes data other than NDF and USA. At distance weights and thirteen neighbors the kNN algorithm gives an accuracy off close to 62%. The accuracy is not that great, but the classification report shows that the sample classification for other classes than NDF has improved considerably.



```
rand.best_params_
```

```
{'weights': 'distance', 'n_neighbors': 13}
```

```
rand.best_estimator_.score(X_test,y_test)
```

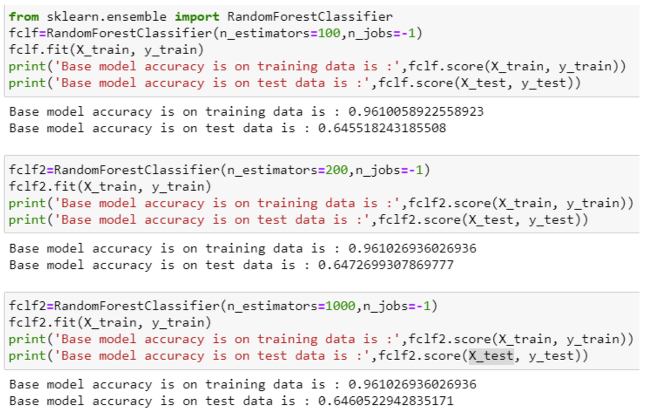
```
0.6183266339139497
```

```
from sklearn.metrics import classification_report
ypred_test=rand.best_estimator_.predict(X_test)
print(classification_report(y_test,ypred_test))
```

```
              precision    recall  f1-score   support

          AU       0.87      0.88      0.88       544
          CA       0.78      0.88      0.82      1388
          DE       0.81      0.90      0.85      1022
          ES       0.76      0.86      0.80      2231
          FR       0.73      0.87      0.80      4988
          GB       0.76      0.86      0.81      2344
          IT       0.77      0.84      0.81      2854
         NDF       0.62      0.67      0.64     30182
          NL       0.82      0.82      0.82       785
          PT       0.82      0.78      0.80       215
          US       0.45      0.35      0.39     20617

    accuracy                           0.62     67170
   macro avg       0.74      0.79      0.77     67170
weighted avg       0.60      0.62      0.61     67170
```

Fig 14 Accuracy and Classification report of kNN on oversampled data

The Random Forest Classifier too has an improved accuracy of about 7% when the data is oversampled as shown below.



```
from sklearn.ensemble import RandomForestClassifier
fclf=RandomForestClassifier(n_estimators=100,n_jobs=-1)
fclf.fit(X_train, y_train)
print('Base model accuracy is on training data is :',fclf.score(X_train, y_train))
print('Base model accuracy is on test data is :',fclf.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.9610058922558923
Base model accuracy is on test data is : 0.645518243185508
```

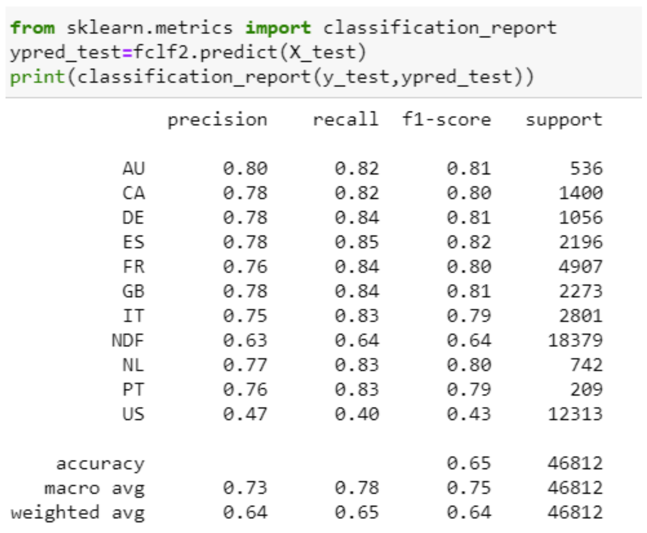
```
fclf2=RandomForestClassifier(n_estimators=200,n_jobs=-1)
fclf2.fit(X_train, y_train)
print('Base model accuracy is on training data is :',fclf2.score(X_train, y_train))
print('Base model accuracy is on test data is :',fclf2.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.961026936026936
Base model accuracy is on test data is : 0.6472699307869777
```

```
fclf2=RandomForestClassifier(n_estimators=1000,n_jobs=-1)
fclf2.fit(X_train, y_train)
print('Base model accuracy is on training data is :',fclf2.score(X_train, y_train))
print('Base model accuracy is on test data is :',fclf2.score(X_test, y_test))
```

```
Base model accuracy is on training data is : 0.961026936026936
Base model accuracy is on test data is : 0.6460522942835171
```

Fig 15 Random Forest Classifier accuracies on oversampled

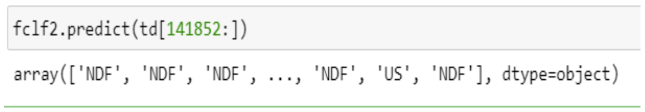


```
from sklearn.metrics import classification_report
ypred_test=fclf2.predict(X_test)
print(classification_report(y_test,ypred_test))
```

```
              precision    recall  f1-score   support

          AU       0.80      0.82      0.81       536
          CA       0.78      0.82      0.80      1400
          DE       0.78      0.84      0.81      1056
          ES       0.78      0.85      0.82      2196
          FR       0.76      0.84      0.80      4907
          GB       0.78      0.84      0.81      2273
          IT       0.75      0.83      0.79      2801
         NDF       0.63      0.64      0.64     18379
          NL       0.77      0.83      0.80       742
          PT       0.76      0.83      0.79       209
          US       0.47      0.40      0.43     12313

    accuracy                           0.65     46812
   macro avg       0.73      0.78      0.75     46812
weighted avg       0.64      0.65      0.64     46812
```

This shows that the classification on test data for different sample classes is over 70% except for US which has only around 47% and NDF that has 63%.

The prediction is carried out on the Random Forest classifier above for test users data as it has a good classification report and accuracy compared to other models.



```
fclf2.predict(td[141852:])
```

```
array(['NDF', 'NDF', 'NDF', ..., 'NDF', 'US', 'NDF'], dtype=object)
```

Fig 17 Predicted values for the test users using Random Forest Classifier

Conclusion/Future Directions

To handle classification or prediction for large datasets, there should be no highly skewed data. This makes the

trained algorithm poorly accurate. Oversampling of the data of the classes is a good idea, when the dataset to be handled after oversampling doesn’t have huge size.
To proceed with the problem in the future and achieve higher level of accuracies overall and with each class, a wider range of hyperparameters should be tried with hyperparameter tuning. This would be possible with systems having greater processing power. Also, the application on other available algorithms like AdaBoost, Gradient Boosting and XGBoost would give fairly good results for suitable hyperparameter values.

The Random Forest Classifier performed well on the oversampled data, among the algorithms considered and is applied for prediction of test users data.