PRML DATA CONTEST REPORT

Name : (1) VISHARAD BORSUTKAR -----> NA18B102
(2) RAHUL SHAKYA ----------> AE17B038

This report contains the approach we used to solve the Data contest problem.

Data contains :

(1) bikers.csv
(2) bikers_network.csv
(3) test.csv
(4) tour_convoy.csv
(5) tours.csv
(6) Train.csv

Libraries used :

Libraries used :
Importing required libraries

```
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
```

Reading all the csv files containing data :

```
train = pd.read_csv('data/train.csv')
tours = pd.read_csv('data/tours.csv')
tour_convoy = pd.read_csv('data/tour_convoy.csv')
bikers = pd.read_csv('data/bikers.csv')
bikers_network = pd.read_csv('data/bikers_network.csv')
test = pd.read_csv('data/test.csv')
```

Steps in Data Processing :

1. Selected unique biker id's from training data
2. Collected data for those unique biker id's from the bikers.csv
3. Checked unique tours_ids from training data
4. Collected data for those unique tour_ids from tours.csv
5. Made a list of organizers from tours.csv
6. Got data of those organizers from bikers.csv

7. Collected organizers friend from biker_network.csv
8. Also, collected bikers friends from biker_network.csv
9. From tour_convoy.csv took data for unique tours_ids which were obtained from tours.csv
10. Replaced the null values with male for gender as it was the mode for the gender category,
11. Replaced null values in born_in with 1993 as it was the mode for the Born_in column.
12. Replaced null values in member_since column with 30-10-2012 as it was the mode for the column.
13. Collected tour details and biker details and concat with X_train.

The code for the above mentioned steps is as follows :

```
# Unique biker_ids from train
train_biker_unique = train['biker_id'].unique()
# Data for unique biker_id
train_bikers = bikers[bikers['biker_id'].apply(lambda x: True if x in
train_biker_unique else False )]
# unique tours from train
train_tours_unique = train['tour_id'].unique()
# Data corresponding to the unique tours form tour.csv
train_tours = tours[tours['tour_id'].apply(lambda x: True if x in
train_tours_unique else False)]
# List of Organizers
organizers = train_tours['biker_id'].unique()
# Data of organizers
organizers_df = bikers[bikers['biker_id'].apply(lambda x: True if x in
organizers else False)]
organizers_friends =
bikers_network[bikers_network['biker_id'].apply(lambda x: True if x in
organizers_df else False)]
bikers_friends = bikers_network[bikers_network['biker_id'].apply(lambda x:
True if x in train_bikers else False)]
train_tour_convoy = tour_convoy[tour_convoy['tour_id'].apply(lambda x:
True if x in train_tours else False)]
# mode ---> male
train_bikers['gender'] = train_bikers['gender'].fillna('male')
# mode --> 1993
train_bikers['bornIn'] = train_bikers['bornIn'].apply(lambda x: '1993' if
x == 'None' else x)
train_bikers['member_since'] =
train_bikers['member_since'].fillna('30-10-2012')
X_train = train.copy()
# train_ids from tours
tours_details = X_train['tour_id'].apply(lambda x:
```

```
train_tours[train_tours['tour_id'] == x].iloc[0])
X_train = X_train.join(tours_details,rsuffix = '_organizer')
bikers_details = X_train['biker_id'].apply(lambda x:
train_bikers[train_bikers['biker_id'] == x].iloc[0])
X_train = X_train.join(bikers_details,rsuffix = '_biker')
X_train = X_train.drop(['biker_id_biker'], axis = 'columns')
```

Featured Introduced :

(1) Age :

```
def age(born_year,tour_date):
    tour_year = float(tour_date.split('-')[2])
    return tour_year - float(born_year)
```

(2) time :

```
def time(member_since,tour_date):
    tour = tour_date.split('-')
    member = member_since.split('-')
    year = float(tour[2])-float(member[2])
    month = float(tour[1])-float(member[1])
    day = float(tour[0])-float(member[0])
    return round(((year*12) + month + (day/30)),2)
```

(3) is_friend :

```
def is_friend(biker_id,organizer_id,x = 0):
    if(x == 0):
        try:
            organizer_friends =
(organizers_friends[organizers_friends['biker_id'] ==
organizer_id]['friends'].iloc[0]).split(' ')
            if(biker_id in organizer_friends):
                return 1
        except:
            pass
        try:
            biker_friends = (bikers_friends[bikers_friends['biker_id'] ==
```

```
biker_id ]['friends'].iloc[0]).split(' ')
            if(organizer_id in biker_friends):
                return 1
        except:
            return 0
        return 0
    else:
        try:
            organizer_friends =
(test_organizers_friends[test_organizers_friends['biker_id'] ==
organizer_id]['friends'].iloc[0]).split(' ')
            if(biker_id in organizer_friends):
                return 1
        except:
            pass
        try:
            biker_friends =
(test_bikers_friends[test_bikers_friends['biker_id'] == biker_id
]['friends'].iloc[0]).split(' ')
            if(organizer_id in biker_friends):
                return 1
        except:
            return 0
        return 0
```

(3) tour_relation :

```
def tour_relation(biker_id,tour_id,x=0):
    s = np.array([0,0,0,0])
    if(x == 0 ):
        try:
            friends = set((bikers_friends[bikers_friends['biker_id'] ==
biker_id]['friends'].iloc[0]).split(' '))
        except:
            return s
        try:
            this_tour_convoy =
train_tour_convoy[train_tour_convoy['tour_id'] == tour_id]
            going = set((this_tour_convoy['going'].iloc[0]).split(' '))
            maybe = set((this_tour_convoy['maybe'].iloc[0]).split(' '))
            invited = set((this_tour_convoy['invited'].iloc[0]).split(' '))
```

```
            not_going = set((this_tour_convoy['not_going'].iloc[0]).split('
'))
        except:
            return s
        try:
            s[0] = len(set(friends) & set(going))
            s[1] = len(set(friends) & set(maybe))
            s[2] = len(set(friends) & set(invited))
            s[3] = len(set(friends) & set(not_going))
        except:
            return s
        return s
    else:
        try:
            friends =
set((test_bikers_friends[test_bikers_friends['biker_id'] ==
biker_id]['friends'].iloc[0]).split(' '))
        except:
            return s
        try:
            this_tour_convoy = test_tour_convoy[test_tour_convoy['tour_id']
== tour_id]
            going = set((this_tour_convoy['going'].iloc[0]).split(' '))
            maybe = set((this_tour_convoy['maybe'].iloc[0]).split(' '))
            invited = set((this_tour_convoy['invited'].iloc[0]).split(' '))
            not_going = set((this_tour_convoy['not_going'].iloc[0]).split('
'))
        except:
            return s
        try:
            s[0] = len(set(friends) & set(going))
            s[1] = len(set(friends) & set(maybe))
            s[2] = len(set(friends) & set(invited))
            s[3] = len(set(friends) & set(not_going))
        except:
            return s
        return s
```

Added tours_details and bikers_details to X_train:

```
tours_details = X_train['tour_id'].apply(lambda x:
train_tours[train_tours['tour_id'] == x].iloc[0])
X_train = X_train.join(tours_details,rsuffix = '_organizer')
bikers_details = X_train['biker_id'].apply(lambda x:
train_bikers[train_bikers['biker_id'] == x].iloc[0])
X_train = X_train.join(bikers_details,rsuffix = '_biker')
X_train = X_train.drop(['biker_id_biker'], axis = 'columns')
```

Created languages column for different languages (dummies):

```
lang_columns = pd.get_dummies(X_train.language_id)
X_train = pd.concat([X_train,lang_columns],axis='columns')
```

Removed columns from X_train and created 'X' and 'Y' as follows :

```
X_train = X_train.drop(['bornIn','member_since','gender'],axis =
'columns')
X = X_train.drop(['biker_id', 'tour_id','timestamp','biker_id_organizer',
'tour_date', 'city', 'state', 'pincode','country', 'latitude',
'longitude','language_id',
      'location_id', 'area', 'like', 'dislike'],axis='columns')
Y = X_train['like']
```

Same procedure repeated for test data and code is as follows:

```
test_bikers = test['biker_id'].unique()
test_bikers_df = bikers[bikers['biker_id'].apply(lambda x: True if x in
test_bikers else False )]
test_tours = test['tour_id'].unique()
test_tours_df = tours[tours['tour_id'].apply(lambda x: True if x in
test_tours else False)]
test_organizers = test_tours_df['biker_id'].unique()
test_organizers_df = bikers[bikers['biker_id'].apply(lambda x: True if x
in test_organizers else False)]
test_organizers_friends =
bikers_network[bikers_network['biker_id'].apply(lambda x: True if x in
test_organizers else False)]
test_bikers_friends =
bikers_network[bikers_network['biker_id'].apply(lambda x: True if x in
test_bikers else False)]
test_tour_convoy = tour_convoy[tour_convoy['tour_id'].apply(lambda x: True
if x in test_tours else False)]
```

```
test_bikers_df['gender'] = test_bikers_df['gender'].fillna('male')
test_bikers_df['bornIn'] = test_bikers_df['bornIn'].apply(lambda x: '1993'
if x == 'None' else x)
test_bikers_df['member_since'] =
test_bikers_df['member_since'].fillna('30-10-2012')
X_test = test.copy()
tours_details_test = X_test['tour_id'].apply(lambda x:
test_tours_df[test_tours_df['tour_id'] == x].iloc[0])
X_test = X_test.join(tours_details_test,rsuffix = '_organizer')
bikers_details_test = X_test['biker_id'].apply(lambda x:
test_bikers_df[test_bikers_df['biker_id'] == x].iloc[0])
X_test = X_test.join(bikers_details_test,rsuffix = '_biker')
X_test = X_test.drop(['biker_id_biker'],axis='columns')
X_test['age'] = X_test[['bornIn','tour_date']].apply(lambda x:
age(x.bornIn,x.tour_date) ,axis=1)
X_test['mambership_time'] =
X_test[['member_since','tour_date']].apply(lambda x:
time(x.member_since,x.tour_date),axis = 1)
X_test['ismale'] = X_test['gender'].apply(lambda x: 1 if x == 'male' else
0)
X_test = X_test.drop(['bornIn','member_since','gender'],axis = 'columns')
X_test['isfriend'] =
X_test[['biker_id','biker_id_organizer']].apply(lambda x:
is_friend(x.biker_id,x.biker_id_organizer,1) ,axis = 1)
friendStatus_test = X_test[['biker_id','tour_id']].apply(lambda x:
tour_relation(x.biker_id,x.tour_id,1),axis = 1)
X_test['friends_going'] = friendStatus_test.apply(lambda x: x[0])
X_test['friends_maybe'] = friendStatus_test.apply(lambda x: x[1])
X_test['friends_invited'] = friendStatus_test.apply(lambda x: x[2])
X_test['friends_not_going'] = friendStatus_test.apply(lambda x: x[3])
lang_columns_test = pd.get_dummies(X_test.language_id)
X_test = pd.concat([X_train,lang_columns_test],axis='columns')
test_X = X_test.drop(['biker_id',
'tour_id','timestamp','biker_id_organizer', 'tour_date', 'city', 'state',
'pincode','country', 'latitude', 'longitude','language_id',
      'location_id', 'area', 'like', 'dislike'],axis='columns')
```

Training the model after hyperparameter tuning and Prediction :

```
Model = XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4, gamma=0.4,
```

```
gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.15, max_delta_step=0, max_depth=9,
             min_child_weight=10, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
Model.fit(X,Y)

Y_prediction = Model.predict_proba(test_X)
Y_prediction = pd.DataFrame(data = Y_prediction)

X_test['prediction'] = Y_prediction[1]

final_model_output = X_test[['biker_id', 'tour_id', 'prediction']]
```

Code to get tours for each biker :

```
Bikers_set = X_test['biker_id'].unique()
bikers_arr = []
tours_arr = []
for biker in Bikers_set:
   bikers_arr.append(biker)
   this_tours = []
   tour_list = X_test[X_test['biker_id'] ==
biker].sort_values(['prediction'],ascending=0)
   tour_list['tour_id'].apply(lambda x: this_tours.append(x))
```

Creating Submission files :

```
submission = pd.DataFrame(columns=["biker_id","tour_id"])
submission["biker_id"] = bikers_arr
submission["tour_id"] = tours_arr
submission.to_csv("NA18B102_AE17B038.csv",index=False)
```

Model could have improved :

(1) By adding features such as distance by using latitude and longitude data.

(2) By combining train and test at first before processing and Processing both at once. After processing we can separate them. This could have reduced the code as well as could have improved the model performance.

(3) If more features were deduced the dropped columns instead of just dropping them.