Data Contest Report

By: Ritvik Rishi (CS18B057)
Rushabh Lalwani (CS18B046)

Problem Statement:

The biker-interest group are the group of cyclists who travel and attend various tours throughout the year. We are provided with the basic personal information of all the bikers. Along with their details, we are also provided with the list of friends and their circle. In the past, the bikers have attended certain tours whose demographic details are provided. With this data of previous tours, bikers friend circle and their personal information, we have to make an ML model to predict the preference over the future tours whose information is given.

Data:

The six files provided are the following:

1. train.csv

train.csv has rows corresponding to tours shown to a biker, and data about whether he/she liked the tour or not.

biker_id	Unique identifier for a biker.
Tour_id	Unique identifier for a particular tour which the biker might be interested in.
invited	binary variable {0,1} indicating whether the biker has been invited to the tour or not
timestamp	Time string representing the approximate time when the biker was informed about the tour
like	A boolean representing whether the biker clicked the “like” option or not.
dislike	A boolean representing whether the biker clicked the “dislike” option or not. It could be possible that the biker didn’t click both options.

2. test.csv

test.csv is similar to train.csv except it does not contain the last two option rows because these are future tours and we don’t know whether the biker will like/dislike them or not.

biker_id	Unique identifier for a biker.

Tour_id	Unique identifier for a particular tour which the biker might be interested in.
invited	binary variable {0,1} indicating whether the biker has been invited to the tour or not
timestamp	Time string representing the approximate time when the biker was informed about the tour

3. tour_convoy.csv

tour_convoy.csv consists of a list of bikers that showed interest in a particular tour. It consists of the list of bikers that attended the given tours.

Tour_id	Unique identifier for the tour.
going	A space-delimited list of bikers who said they will go to the tour.
maybe	A space-delimited list of bikers who said they might go to the tour.
invited	A space-delimited list of bikers who were invited to the tour.
not_going	A space-delimited list of bikers who said they will not go to the tour.

4. bikers.csv

Bikers.csv has the demographic data of the bikers (includes all the bikers from train.csv and test.csv). The features in this file are mentioned below.

biker_id	Unique identifier for the biker person.

language_id	Identifier of the language the biker speaks.
location_id	Identifier of the location the biker resides in.
bornIn	Year of birth of the biker to estimate their age.
gender	Male/Female based on their bikers input.
member_since	Date of joining the bikers interest group.
area	Bikers location area.
time_zone	Time offset in minutes to GMT timezone of where the biker is. (For example, Indian Time is +5:30 GMT, so +330 minutes)

5. tours.csv

Tours.csv has the demographic features of the tours (includes all the tours from train.csv and test.csv). It has a total of 110 columns.

tour_id	Unique identifier for the particular tour.
biker_id	The ID of the biker who organized the tour.
tour_date	The date on which the tour was conducted.
city	Location of the tour - city (if known)
state	Location of the tour - state (if known)
pin code	Location of the tour - pin code (if known)

country	Location of the tour - country (if known)
latitude	Approximate location of the starting point of the tour (if known)
longitude	Approximate location of the starting point of the tour (if known)
w1..w100	Count of the number of occurrences of most common words in the description of the tour guide.
w_others	Count of other words

6. bikers_network.csv

bikers_network.csv consists of the social networks of the bikers. This is derived from the group of bikers that know each other via some groups.

biker_id	Unique identifier for the biker person.
friends	A space-delimited list of all friends of given biker_id.

Approach:

Given this data, we are required to predict the rankings of tours for a given biker. We tried various approaches to the problem, starting with collaborative filtering to get a score for a tour by a biker. We weren't able to effectively implement this, probably due to the sparseness of data regarding biker-tour scores. Another way could be regression to predict the like-dislike but we did not try that. **Binary classification for predicting the "like" for a biker-tour pair** was found to give the best accuracy. We also tried multiclass methods of classification to predict "like"(1), "dislike"(-1) or (0) but that did not perform better than binary classification.

We first build a feature matrix and input that to a binary classification model. The methods of building various features and the model are described below:

Preprocessing:

There are some missing values in many of the data files. We do not handle them during preprocessing, and instead, let the implementation of our model which uses the Gradient Boosting Machine handle them.

We notice that the number of tours in the tours.csv file is very large, and we require only a small number of them in the train/test files. So, we filter tours.csv according to the tours present in train/test files.

Features

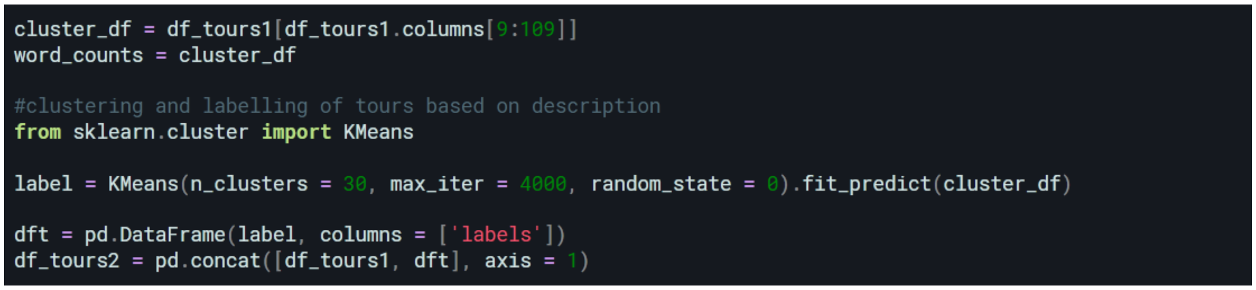
- **Gender**: The biker gender is added as a categorical feature. We converted it into numeric value using the *LabelEncoder* class from *sklearn.preprocessing* library. The missing values are not given any default values and assumed to be handled by the model.

· **Language id**: The language id of the biker is used as a feature from biker data. With 54 unique language ids, it formed an important feature for classification. We converted them into numeric values and used directly for classification.

· **Location id**: The location id of the biker is used as a feature from biker data. With 56 unique location ids, it formed an important feature for classification. We converted them into numeric values and used directly for classification.

· **Age:** The birth year of the biker is also used as a feature.

· **Labels:** Tours were clustered into 30 clusters using KMeans clustering and the cluster number allotted to a tour was added as a feature (categorical).



```
cluster_df = df_tours1[df_tours1.columns[9:109]]
word_counts = cluster_df

#clustering and labelling of tours based on description
from sklearn.cluster import KMeans

label = KMeans(n_clusters = 30, max_iter = 4000, random_state = 0).fit_predict(cluster_df)

dft = pd.DataFrame(label, columns = ['labels'])
df_tours2 = pd.concat([df_tours1, dft], axis = 1)
```

· **Words:** Common tour words frequency (101 columns in tours.csv) were processed and decomposed to 10 features using the *PCA* module of *sklearn.decomposition* library. These features are added in the classification model.

· **Number of friends:** For the biker, we add its number of friends as a feature.

· **Organiser's Friends**: This is a binary feature which tells if the organiser of the tour is the biker's friend or not.

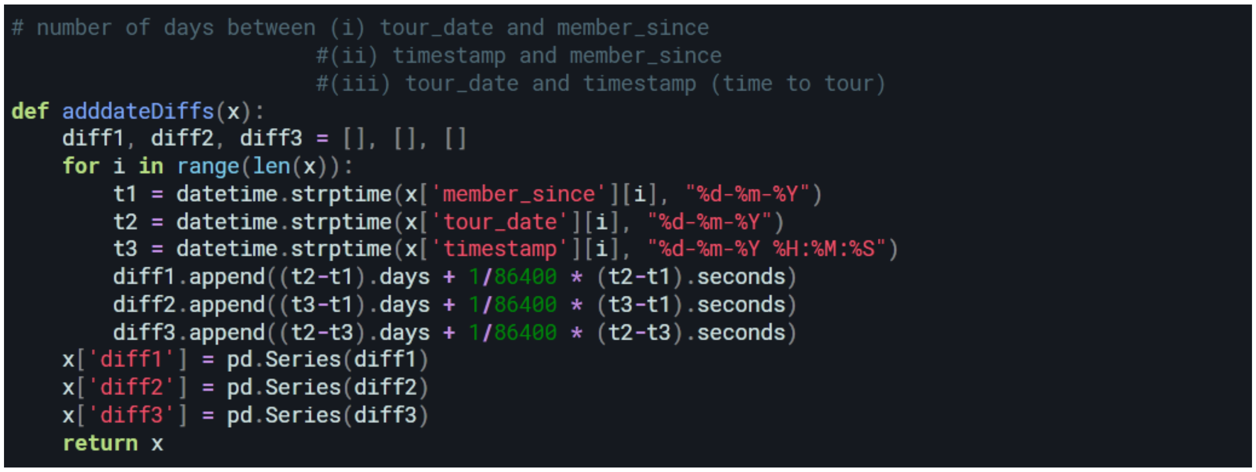
· **Distance:** The distance feature is calculated by using the *geodesic* module from *geopy.distance* library. It uses the latitude and longitude of both biker and tour.

· **Tour popularity**: tour_convoy.csv included four sets which consist of the bikers who are going/not going/maybe/interested for the given tour. We calculated the number of such bikers in a set for a given tour and made corresponding four features for a given tour as its popularity.

· **Friends influence:** Based on tour popularity and friends, we create four friends influence features. For the given tour, we got the set of bikers for each of the going/not going/maybe/interested fields and then we took the intersection of these four sets with the friends of the given biker. The number of friends forms four different features.

- **Duration features:** Here we considered these three dates provided: tour_date is the date of the tour, member_since is the date of joining the biker and timestamp is the approximate time when the invitation was given to the biker for this tour. There is no missing value in these dates. We calculate the difference in time of these three variables and correspondingly get the three features.

diff1	tour_date - member_since	The difference in date of joining and tour date.
diff2	timestamp - member_since	The difference in the invitation date and the date of joining.
diff3	tour_date - timestamp	The difference in the date of the tour and date of invitation.



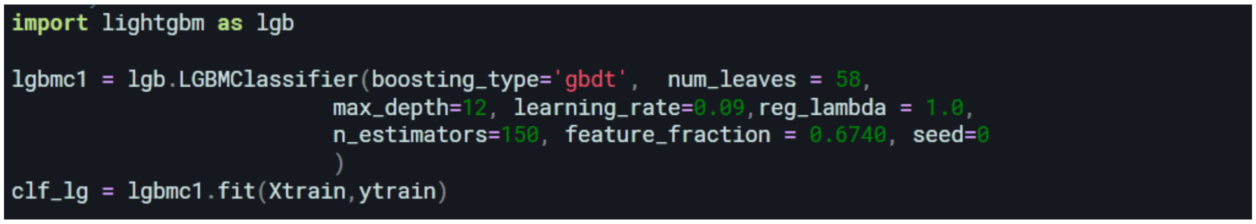
```python
# number of days between (i) tour_date and member_since
                        #(ii) timestamp and member_since
                        #(iii) tour_date and timestamp (time to tour)
def adddateDiffs(x):
    diff1, diff2, diff3 = [], [], []
    for i in range(len(x)):
        t1 = datetime.strptime(x['member_since'][i], "%d-%m-%Y")
        t2 = datetime.strptime(x['tour_date'][i], "%d-%m-%Y")
        t3 = datetime.strptime(x['timestamp'][i], "%d-%m-%Y %H:%M:%S")
        diff1.append((t2-t1).days + 1/86400 * (t2-t1).seconds)
        diff2.append((t3-t1).days + 1/86400 * (t3-t1).seconds)
        diff3.append((t2-t3).days + 1/86400 * (t2-t3).seconds)
    x['diff1'] = pd.Series(diff1)
    x['diff2'] = pd.Series(diff2)
    x['diff3'] = pd.Series(diff3)
    return x
```

The above features were chosen as an input to the binary classification model after careful study of their effect on accuracy using KFold cross-validation.

Model Building

We appended all the above features to every biker-tour pair in the train and test files to obtain a feature matrix. We are using binary classification to predict the 'like' column for a biker-tour pair using a gradient boosting decision tree. We used the LightGBM library to implement the gradient boosting decision tree. We studied the algorithm behind the LightGBM classifier, it's implementation, and parameter tuning through the documentation and various blogs, articles and Kaggle notebooks.

We trained the model using the train file, obtained the submission file by arranging the probability predictions of 'like' for all tours for a biker in decreasing order. The difference in the two submission files is due to different hyperparameter choices for training the gbdt model. A code snippet on training the model is shown below.



```
import lightgbm as lgb

lgbmc1 = lgb.LGBMClassifier(boosting_type='gbdt',  num_leaves = 58,
                         max_depth=12, learning_rate=0.09,reg_lambda = 1.0,
                         n_estimators=150, feature_fraction = 0.6740, seed=0
                         )
clf_lg = lgbmc1.fit(Xtrain,ytrain)
```

Observations and Takeaways:

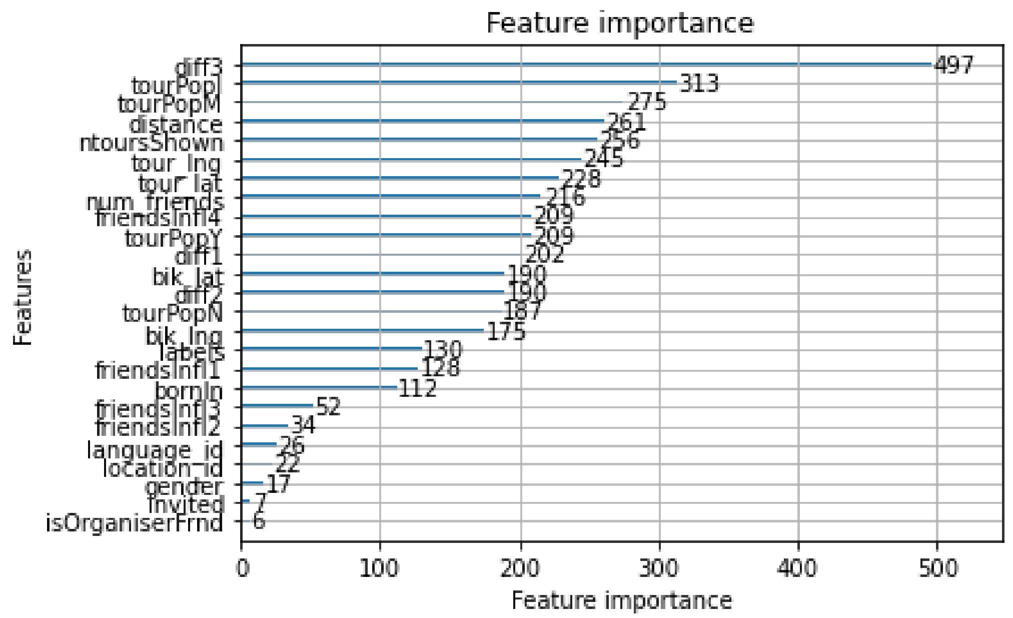
- ### Missing Values:

There were some missing values in the data but we did not handle them in the preprocessing. Arriving at a missing location for a biker by inferring it from the biker's friends' locations might have helped as locations and distances were found to be important features.

- ### Feature Engineering:

In our opinion, this data contest was mainly about feature engineering. We learnt how to extract useful information/features from noisy and voluminous and not entirely comprehensible real-world data. This involved identifying and considering useful features from the given data and constructing features that derive from the data.

- ## Feature Importance:



On plotting feature importance, we observed that the time to tour, tour popularity, distance, and the number of tours shown to the biker are considered the most important factors for splitting. However, not considering the distance feature did not affect the accuracy significantly, and that may be due to the presence of similar tour and biker latitude and longitude features. So, we used cross-validation and chose features that reduce the validation error.

- ## Model Selection and hyperparameter tuning:

Going from logistic regression to SVMs to decision trees to random forests to gradient boosting machines, gradient boosting decision trees using libraries such as xgboost and LightGBM were found to perform the best.

For hyperparameter tuning, we used Bayesian Optimisation as it is faster than GridSearchCV because it learns as it goes. We tuned parameters such as feature_fraction, max_depth and learning_rate to combat overfitting and increase validation score while reducing the gap between training and validation scores.

- **Overfitting and public/private leaderboard:**

Despite trying to tune hyperparameters such that they don’t overfit on training data and give a good score on the public leaderboard, there was a difference in our model’s performance on the private leaderboard. This was likely due to our model overfitting on the data considered for the private leaderboard. Our most general models with lesser complex features were way too close to the baseline 2 for comfort but they were observed to perform better than the final models with more complex features on the private leaderboard. While trying to get our model ahead of the baseline2, we overfit on the public leaderboard data (as did the baseline2 model, because it performed way worse on the private leaderboard). Not including very specific features and not trying to increase the scores on the public leaderboard but rather focussing on making a good generic model was a key takeaway from the contest.