Introduction

Thanks for looking.
This competition will predict the outcome of future soccer matches based on historical data!
In this notebook, we will show a simple implementation of the lightgbm method for beginners in machine learning.

Module Load

First, import required modules.

In [2]:
```
import numpy as np
import pandas as pd
import warnings
import time
warnings.simplefilter('ignore')
import math
from statistics import mean
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import timedelta
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import log_loss
```

Loading Data

Loading data by using read_csv in pandas.

In [3]:
```
train_df = pd.read_csv('C:/Users/HP/Downloads/football-match-probability-prediction/tr
test_df = pd.read_csv('C:/Users/HP/Downloads/football-match-probability-prediction/tes
train_target_score_df = pd.read_csv('C:/Users/HP/Downloads/football-match-probability-
```

Explanatory Data Analysis

Check the contents and shape

In [4]:
```
print(train_df.shape)
train_df.head()
```

```
(110938, 190)
```

Out[4]:

	id	target	home_team_name	away_team_name	match_date	league_name	league_id	is
0	11906497	away	Newell's Old Boys	River Plate	2019-12-01 00:45:00	Superliga	636	
1	11984383	home	Real Estelí	Deportivo Las Sabanas	2019-12-01 01:00:00	Primera Division	752	
2	11983301	draw	UPNFM	Marathón	2019-12-01 01:00:00	Liga Nacional	734	
3	11983471	away	León	Morelia	2019-12-01 01:00:00	Liga MX	743	
4	11883005	home	Cobán Imperial	Iztapa	2019-12-01 01:00:00	Liga Nacional	705	

5 rows × 190 columns

In [5]:
```
print(train_target_score_df.shape)
train_target_score_df.head()
```

(110938, 3)

Out[5]:

	id	score	target
0	11906497	2-3	away
1	11984383	1-0	home
2	11983301	2-2	draw
3	11983471	1-2	away
4	11883005	1-0	home

In [6]:
```
print(test_df.shape)
test_df.head()
```

(72711, 189)

Out[6]:

	id	home_team_name	away_team_name	match_date	league_name	league_id	is_cup	hom
0	17761448	team home	team away	2021-05-01 00:15:00	Division 1	755	False	
1	17695487	team home	team away	2021-05-01 00:30:00	Liga MX	743	False	
2	17715496	team home	team away	2021-05-01 01:00:00	Paulista A2	1314	False	
3	17715493	team home	team away	2021-05-01 01:00:00	Paulista A2	1314	False	
4	17715492	team home	team away	2021-05-01 01:00:00	Paulista A2	1314	False	

5 rows × 189 columns

Data Summary

train.csv...the training set. It contains data from 110938 matches from December 1, 2019 to May 1, 2021.
test.csv...the test set. No away and home team names.
train_target_and_scores.csv - contains each match's final score Home - Away in addition to the target which is also in the training set.

- target - The team that won that match
- home_team_name - The name of the Home the team. Hidden in test set, see this discussion
- away_team_name - The name of the Away the team. Hidden in test set, see this discussion
- match_date - The match date (UTC).
- league_name - The league name.
- league_id - The league id. Note that league names can be identical for two differents id.
- is_cup - If the value is 1 the match is played for a cup competition.
- home_team_coach_id - The id of the Home team coach.
- away_team_coach_id - The id of the Away team coach.
- home_team_history_match*date*{i} - The date of the last i-th match played by Home team.
- home_team_history_is_play*home*{i} - If 1, the Home team played home.
- home_team_history_is*cup*{i} - If 1, the match was a cup competition.
- home_team_history*goal*{i} - The number of goals scored by the Home team on its last i-th match.
- home_team_history_opponent*goal*{i} - The number of goals conceded by the Home team on its last i-th match.
- home_team_history*rating*{i} - The rating of the Home team on its last i-th match (pre match rating).
- home_team_history_opponent*rating*{i} - The rating of the opponent team on Home team last i-th match (pre match rating).
- home_team_history*coach*{i} - The coach id of the Home team on its last i-th match.
- home_team_history_league*id*{i} - The league name id by the Home team on its last i-th match.
- away_team_history_match*date*{i} - The date of the last i-th match played by Away team.
- away_team_history_is_play*home*{i} - If 1, the Away team played home.
- away_team_history_is*cup*{i} - If 1, the match was a cup competition.
- away_team_history*goal*{i} - The number of goals scored by the Away team on its last i-th match.
- away_team_history_opponent*goal*{i} - The number of goals conceded by the Away team on its last i-th match.
- away_team_history*rating*{i} - The rating of the Away team on its last i-th match (pre match rating).
- away_team_history_opponent*rating*{i} - The rating of the opponent team on Away team last i-th match (pre match rating).
- away_team_history*coach*{i} - The coach id of the Away team on its last i-th match.

- away_team_history_league*id*{i} - The league name id played by the Away on its last i-th match.

Specify id as line name

In [7]:
```
train_df.set_index(keys='id', inplace=True)
test_df.set_index(keys='id', inplace=True)
train_df.head()
```

Out[7]:

id	target	home_team_name	away_team_name	match_date	league_name	league_id	is_cup
11906497	away	Newell's Old Boys	River Plate	2019-12-01 00:45:00	Superliga	636	False
11984383	home	Real Estelí	Deportivo Las Sabanas	2019-12-01 01:00:00	Primera Division	752	False
11983301	draw	UPNFM	Marathón	2019-12-01 01:00:00	Liga Nacional	734	False
11983471	away	León	Morelia	2019-12-01 01:00:00	Liga MX	743	False
11883005	home	Cobán Imperial	Iztapa	2019-12-01 01:00:00	Liga Nacional	705	False

5 rows × 189 columns

For `train_target_score_df`, do the same with the line name as id and label encoding for `'target'`

In [8]:
```
train_target_score_df.set_index(keys='id', inplace=True)
train_target_score_df = train_target_score_df['target'].map({'home': 0, 'draw': 1, 'av
train_target_score_df.head()
```

Out[8]:
```
id
11906497    2
11984383    0
11983301    1
11983471    2
11883005    0
Name: target, dtype: int64
```

Here, visualize the relationship between target and other features using `seaborn`

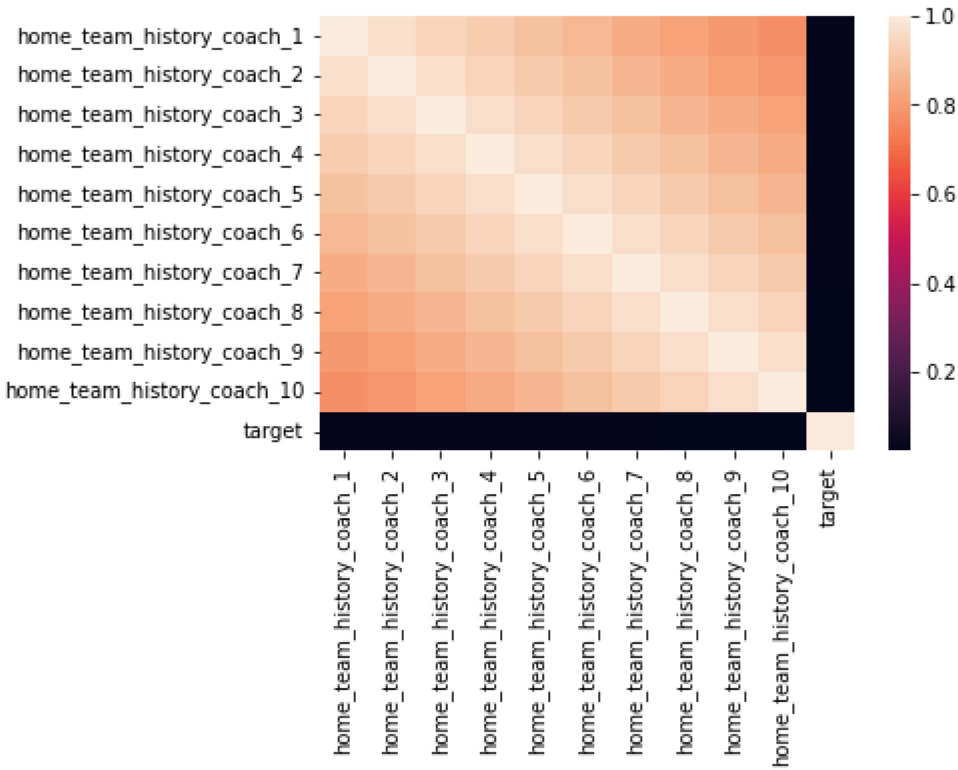
- Target and league_id do not correlate
- Targets and rating are slightly correlate . .

In [9]:
```
### Please change to the column that interests you. (ex)'home_team_history_rating'
target_columns = 'home_team_history_coach'
###
cols = []
for col in train_df.filter(regex=target_columns, axis=1).columns:
    cols.append(col)
```

```
print(cols)
corr = pd.concat([train_df[cols],train_target_score_df], axis=1).corr()
sns.heatmap(corr)
```

```
['home_team_history_coach_1', 'home_team_history_coach_2', 'home_team_history_coach_3', 'home_team_history_coach_4', 'home_team_history_coach_5', 'home_team_history_coach_6', 'home_team_history_coach_7', 'home_team_history_coach_8', 'home_team_history_coach_9', 'home_team_history_coach_10']
```

Out[9]: `<AxesSubplot:>`



Features

Next, finding and observing the features

First, except for the correct labels in `train_df`, combine with `test_df` for simultaneous processing

In [10]:
```
# Only correct labels should be separated.
train_df_y = train_df['target']
train_df.drop(['target'], axis=1, inplace=True)
train_df.head()
```

Out[10]:

	home_team_name	away_team_name	match_date	league_name	league_id	is_cup	home_t
id							
11906497	Newell's Old Boys	River Plate	2019-12-01 00:45:00	Superliga	636	False	
11984383	Real Estelí	Deportivo Las Sabanas	2019-12-01 01:00:00	Primera Division	752	False	
11983301	UPNFM	Marathón	2019-12-01 01:00:00	Liga Nacional	734	False	
11983471	León	Morelia	2019-12-01 01:00:00	Liga MX	743	False	
11883005	Cobán Imperial	Iztapa	2019-12-01 01:00:00	Liga Nacional	705	False	

5 rows × 188 columns

In [11]:

```
# Combine train_df and test_df
ntrain = train_df.shape[0]
all_data = pd.concat((train_df, test_df))
print(all_data.shape)
all_data.head()
```

```
(183649, 188)
```

Out[11]:

	home_team_name	away_team_name	match_date	league_name	league_id	is_cup	home_t
id							
11906497	Newell's Old Boys	River Plate	2019-12-01 00:45:00	Superliga	636	False	
11984383	Real Estelí	Deportivo Las Sabanas	2019-12-01 01:00:00	Primera Division	752	False	
11983301	UPNFM	Marathón	2019-12-01 01:00:00	Liga Nacional	734	False	
11983471	León	Morelia	2019-12-01 01:00:00	Liga MX	743	False	
11883005	Cobán Imperial	Iztapa	2019-12-01 01:00:00	Liga Nacional	705	False	

5 rows × 188 columns

Remove date, team_name and coach as they are not used.

In [12]:

```
all_data.drop(['home_team_name', 'away_team_name', 'league_name'], axis=1, inplace=Tru
all_data.drop(all_data.filter(regex='date').columns, axis=1, inplace = True)
all_data.drop(all_data.filter(regex='coach').columns, axis=1, inplace = True)
all_data.head()
```

Out[12]:

id	league_id	is_cup	home_team_history_is_play_home_1	home_team_history_is_play_home_2	I
11906497	636	False	0.0	1.0	
11984383	752	False	1.0	0.0	
11983301	734	False	0.0	1.0	
11983471	743	False	0.0	0.0	
11883005	705	False	0.0	1.0	

5 rows × 142 columns

Convert is_cup to numeric information since is_cup is character information.

In [13]:
```
all_data['is_cup'] = all_data['is_cup'].map({False: 0, True: 1})
all_data.head()
```

Out[13]:

id	league_id	is_cup	home_team_history_is_play_home_1	home_team_history_is_play_home_2	I
11906497	636	0.0	0.0	1.0	
11984383	752	0.0	1.0	0.0	
11983301	734	0.0	0.0	1.0	
11983471	743	0.0	0.0	0.0	
11883005	705	0.0	0.0	1.0	

5 rows × 142 columns

Check missing information

In [14]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(ascenc
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
print(len(all_data_na))
missing_data.head(30)
```

```
30
```

Out[14]:

	Missing Ratio
away_team_history_is_cup_10	11.564452
away_team_history_opponent_rating_10	11.184923
away_team_history_rating_10	11.184923
away_team_history_league_id_10	11.175667
away_team_history_opponent_goal_10	11.175667
away_team_history_goal_10	11.175667
away_team_history_is_play_home_10	11.175667
home_team_history_is_cup_10	11.082010
home_team_history_opponent_rating_10	10.697581
home_team_history_rating_10	10.697581
home_team_history_opponent_goal_10	10.685057
home_team_history_goal_10	10.685057
home_team_history_league_id_10	10.685057
home_team_history_is_play_home_10	10.685057
away_team_history_is_cup_9	10.373593
away_team_history_rating_9	10.035448
away_team_history_opponent_rating_9	10.035448
away_team_history_is_play_home_9	10.023469
away_team_history_opponent_goal_9	10.023469
away_team_history_goal_9	10.023469
away_team_history_league_id_9	10.023469
home_team_history_is_cup_9	9.916199
home_team_history_opponent_rating_9	9.559540
home_team_history_rating_9	9.559540
home_team_history_league_id_9	9.549739
home_team_history_is_play_home_9	9.549739
home_team_history_opponent_goal_9	9.549739
home_team_history_goal_9	9.549739
away_team_history_is_cup_8	9.156053
away_team_history_opponent_rating_8	8.864737

The missing information is as follows.

- ~ is_cup ~ ... fill with zero.
- ~ rating ~ ... fill with mean value.

- ~ is_play_home ~ ... fill with 0.5.
- ~ goal ~ ... fill with mean value.

In [15]:
```
# is_cup
for col in all_data.filter(regex='is_cup', axis=1).columns:
    all_data[col] = all_data[col].fillna(0)
# rating
for col in all_data.filter(regex='rating', axis=1).columns:
    all_data[col] = all_data[col].fillna(all_data[col].mean())
# is play home
for col in all_data.filter(regex='is_play_home', axis=1).columns:
    all_data[col] = all_data[col].fillna(0.5)
# league
for col in all_data.filter(regex='league', axis=1).columns:
    all_data[col] = all_data[col].fillna(0)
# goal
for col in all_data.filter(regex='goal', axis=1).columns:
    all_data[col] = all_data[col].fillna(all_data[col].mean())
```

Reconfirm missing values

In [16]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(ascend
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
print(len(all_data_na))
missing_data.head()
```

```
0
```

Out[16]:

Missing Ratio

Undo `test_df` and `train_df`

In [17]:
```
train_df = all_data[:ntrain]
test_df = all_data[ntrain:]
print(train_df.shape, test_df.shape)
train_df.head()
```

```
(110938, 142) (72711, 142)
```

Out[17]:

	league_id	is_cup	home_team_history_is_play_home_1	home_team_history_is_play_home_2	l
id					
11906497	636	0.0	0.0	1.0	
11984383	752	0.0	1.0	0.0	
11983301	734	0.0	0.0	1.0	
11983471	743	0.0	0.0	0.0	
11883005	705	0.0	0.0	1.0	

5 rows × 142 columns

Trainig

Now that feature engineering is done, create and train the model

Model Partitioning

Split `train_df` into **training data** (used to train the model) and **test data** (used to verify generalization performance of the model)

In [18]:
```
# pandas -> numpy
X = train_df.values
y = train_target_score_df.values
print(X.shape, y.shape)
# split training data and test data(20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(110938, 142) (110938,)
(88750, 142) (22188, 142)
(88750,) (22188,)
```

Hyperparameter tuning

Furthermore, the training data is split into training data and validation data, and hyperparameter tuning is performed using the validation data.
This process is time consuming, so skip this cell when implementing without tuning.

In [19]:
```
# import optuna.integration.lightgbm as lgb   #Models with tuning

# X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.
# # Create dataset for LightGBM
# lgb_train = lgb.Dataset(X_train, y_train)
# lgb_valid = lgb.Dataset(X_valid, y_valid)
# # setting parameter
# params = {'objective': 'multiclass',
#           'num_class': 3,
#           'metric': 'multi_logloss',
#           'verbosity': -1,
#           'num_leaves': 45,
#           'learning_rate': 0.05,
#           'feature_fraction': 1.0,
#           'bagging_fraction': 0.7,
#           'bagging_freq': 2,
#           'feature_pre_filter': False,
#           'lambda_l1': 0.005,
#           'lambda_l2': 1.0e-08,
#           'min_child_samples': 25}

# # Model creation from training data
# gbm = lgb.train(params, lgb_train, valid_sets=lgb_valid,
#                 verbose_eval=50, # Learning result output every 50 iter
#                 num_boost_round=10000, # max iteration
```

```
#                 early_stopping_rounds=100
#                )
# best_params = gbm.params
# print(best_params)
# # Check forecast accuracy
# y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
# print(y_pred)
```

Learning without Tuning.
Here, **k-fold cross validation** is performed to create a model with better generalization performance

In [20]:
```
import lightgbm as lgb   # No hyper-parameter tuning
# training parameter
params = {'objective': 'multiclass',
          'num_class': 3,
          'metric': 'multi_logloss',
          'verbosity': -1,
          'num_leaves': 6,
          'learning_rate': 0.05,
          'feature_fraction': 0.4,
          'bagging_fraction': 0.6780922358381494,
          'bagging_freq': 4,
          'feature_pre_filter': False,
          'lambda_l1': 1.0136734007272632,
          'lambda_l2': 5.110989943250052,
          'min_data_in_leaf': 25,
          'min_child_samples': 20,
          'num_iterations': 10000,
          'early_stopping_round': 100}

kfold = StratifiedKFold(n_splits=10,
                        random_state=1, shuffle=True).split(X_train, y_train)
scores = []
models = []
for k, (train, test) in enumerate(kfold):
    X_trainset_lgb = lgb.Dataset(X_train[train], y_train[train])
    X_validset_lgb = lgb.Dataset(X_train[test], y_train[test])

    gbm = lgb.train(params, X_trainset_lgb, valid_sets=X_validset_lgb,
                    verbose_eval=100,
                    num_boost_round=10000,
                    early_stopping_rounds=100)
    pred_t = gbm.predict(X_train[test], num_iteration=gbm.best_iteration)
    score = log_loss(y_train[test], pred_t) # Calculation of correct response rate for
    scores.append(score)  # append score
    print('Fold: %2d, loss: %.3f' % (k+1, score))
    models.append(gbm)
print('\nCV loss: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]   valid_0's multi_logloss: 1.01993
[200]   valid_0's multi_logloss: 1.01261
[300]   valid_0's multi_logloss: 1.01058
[400]   valid_0's multi_logloss: 1.00934
[500]   valid_0's multi_logloss: 1.00902
[600]   valid_0's multi_logloss: 1.0087
[700]   valid_0's multi_logloss: 1.00854
[800]   valid_0's multi_logloss: 1.00869
Early stopping, best iteration is:
[706]   valid_0's multi_logloss: 1.00849
Fold:  1, loss: 1.008
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]   valid_0's multi_logloss: 1.02168
[200]   valid_0's multi_logloss: 1.01532
[300]   valid_0's multi_logloss: 1.01384
[400]   valid_0's multi_logloss: 1.0133
[500]   valid_0's multi_logloss: 1.01287
[600]   valid_0's multi_logloss: 1.01271
[700]   valid_0's multi_logloss: 1.01225
[800]   valid_0's multi_logloss: 1.01222
Early stopping, best iteration is:
[765]   valid_0's multi_logloss: 1.01206
Fold:  2, loss: 1.012
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]   valid_0's multi_logloss: 1.02071
[200]   valid_0's multi_logloss: 1.01463
[300]   valid_0's multi_logloss: 1.01241
[400]   valid_0's multi_logloss: 1.01161
[500]   valid_0's multi_logloss: 1.01129
[600]   valid_0's multi_logloss: 1.01106
[700]   valid_0's multi_logloss: 1.01062
Early stopping, best iteration is:
[694]   valid_0's multi_logloss: 1.0106
Fold:  3, loss: 1.011
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]   valid_0's multi_logloss: 1.02267
[200]   valid_0's multi_logloss: 1.01651
[300]   valid_0's multi_logloss: 1.01463
[400]   valid_0's multi_logloss: 1.01383
[500]   valid_0's multi_logloss: 1.01361
[600]   valid_0's multi_logloss: 1.01344
[700]   valid_0's multi_logloss: 1.01343
Early stopping, best iteration is:
[651]   valid_0's multi_logloss: 1.01322
Fold:  4, loss: 1.013
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]   valid_0's multi_logloss: 1.02073
[200]   valid_0's multi_logloss: 1.01365
[300]   valid_0's multi_logloss: 1.011
```

```
[400]	valid_0's multi_logloss: 1.01015
[500]	valid_0's multi_logloss: 1.01008
Early stopping, best iteration is:
[440]	valid_0's multi_logloss: 1.00997
Fold:  5, loss: 1.010
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]	valid_0's multi_logloss: 1.02085
[200]	valid_0's multi_logloss: 1.01403
[300]	valid_0's multi_logloss: 1.01182
[400]	valid_0's multi_logloss: 1.0103
[500]	valid_0's multi_logloss: 1.0099
[600]	valid_0's multi_logloss: 1.00951
[700]	valid_0's multi_logloss: 1.00896
[800]	valid_0's multi_logloss: 1.00877
Early stopping, best iteration is:
[793]	valid_0's multi_logloss: 1.00868
Fold:  6, loss: 1.009
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]	valid_0's multi_logloss: 1.02338
[200]	valid_0's multi_logloss: 1.01684
[300]	valid_0's multi_logloss: 1.01457
[400]	valid_0's multi_logloss: 1.0137
[500]	valid_0's multi_logloss: 1.01346
[600]	valid_0's multi_logloss: 1.0131
[700]	valid_0's multi_logloss: 1.01275
[800]	valid_0's multi_logloss: 1.01246
Early stopping, best iteration is:
[790]	valid_0's multi_logloss: 1.01239
Fold:  7, loss: 1.012
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]	valid_0's multi_logloss: 1.02941
[200]	valid_0's multi_logloss: 1.02573
[300]	valid_0's multi_logloss: 1.02449
[400]	valid_0's multi_logloss: 1.02378
[500]	valid_0's multi_logloss: 1.02336
[600]	valid_0's multi_logloss: 1.02317
[700]	valid_0's multi_logloss: 1.02307
[800]	valid_0's multi_logloss: 1.02301
Early stopping, best iteration is:
[750]	valid_0's multi_logloss: 1.02296
Fold:  8, loss: 1.023
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]	valid_0's multi_logloss: 1.02099
[200]	valid_0's multi_logloss: 1.01451
[300]	valid_0's multi_logloss: 1.01232
[400]	valid_0's multi_logloss: 1.01131
[500]	valid_0's multi_logloss: 1.01084
[600]	valid_0's multi_logloss: 1.0104
[700]	valid_0's multi_logloss: 1.01026
[800]	valid_0's multi_logloss: 1.01029
Early stopping, best iteration is:
[763]	valid_0's multi_logloss: 1.01012
```

```
Fold:  9, loss: 1.010
[LightGBM] [Warning] min_data_in_leaf is set=25, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=25
Training until validation scores don't improve for 100 rounds
[100]	valid_0's multi_logloss: 1.02187
[200]	valid_0's multi_logloss: 1.01477
[300]	valid_0's multi_logloss: 1.01239
[400]	valid_0's multi_logloss: 1.01151
[500]	valid_0's multi_logloss: 1.01108
[600]	valid_0's multi_logloss: 1.01097
Early stopping, best iteration is:
[574]	valid_0's multi_logloss: 1.01084
Fold: 10, loss: 1.011

CV loss: 1.012 +/- 0.004
```

Create a function for predict, make predictions with k models, and average the predictions.

In [21]:

```
def predict(models, X_test):
#  Create array for storing test data
    y_pred = np.zeros((len(X_test), len(models), 3))
    for fold_, model in enumerate(models):
        # predict test
        pred_ = model.predict(X_test, num_iteration=model.best_iteration)
        # save predict
        y_pred[:, fold_] = pred_
    y_pred = y_pred.mean(axis=1)
    return y_pred
y_pred = predict(models, X_test)
y_pred_train = predict(models, X_train)
print('test loss = ', log_loss(y_test, y_pred))
print('train loss = ', log_loss(y_train, y_pred_train))
```

```
test loss =  1.0082318873355745
train loss =  0.9786918616071916
```

Submit prediction

Create data for submission.

In [22]:

```
X_submit = test_df.values
y_submit = predict(models, X_submit)
print(y_submit.shape)
```

```
(72711, 3)
```

In [24]:

```
submission_df = pd.read_csv('C:/Users/HP/Downloads/football-match-probability-predicti
submission_df.head()
```

Out[24]:

	id	home	draw	away
0	17761448	0.333	0.333	0.333
1	17695487	0.333	0.333	0.333
2	17715496	0.333	0.333	0.333
3	17715493	0.333	0.333	0.333
4	17715492	0.333	0.333	0.333

In [25]:

```
submission_df['home'] = pd.DataFrame(y_submit[:, 0])
submission_df['draw'] = pd.DataFrame(y_submit[:, 1])
submission_df['away'] = pd.DataFrame(y_submit[:, 2])
submission_df.head()
```

Out[25]:

	id	home	draw	away
0	17761448	0.443946	0.291687	0.264367
1	17695487	0.344119	0.316986	0.338895
2	17715496	0.374987	0.299761	0.325252
3	17715493	0.183615	0.323291	0.493094
4	17715492	0.461505	0.328985	0.209510

In [26]:

```
submission_df.to_csv("submission.csv", index=False, header=True)
```

In [ ]:

In [ ]: