1. **Task 2**

Run task\_2.py and input name of two teams

| import pandas as pd  import numpy as np  from datetime import datetime  import json  from tqdm import tqdm  def create\_history\_team():  df = pd.read\_csv(f"train.csv", low\_memory=False)  df.fillna(value=-1, inplace=True)  teams\_history = dict()  for i in tqdm(range(len(df.index))):  for x in ["home", "away"]:  team = df.iloc[i][f'{x}\_team\_name']  if team not in teams\_history.keys():  teams\_history[team] = []  value = dict()  value['match\_date'] = df.iloc[i]['match\_date']  value['league\_id'] = int(df.iloc[i]['league\_id'])  teams\_history[team].append(value)  for j in range(1, 11):  if df.iloc[i][f'{x}\_team\_history\_match\_date\_{j}'] != -1:  value = dict()  value[f'match\_date'] = df.iloc[i][f'{x}\_team\_history\_match\_date\_{j}']  value['league\_id'] = int(df.iloc[i][f'{x}\_team\_history\_league\_id\_{j}'])  teams\_history[team].append(value)  elif team in teams\_history.keys():  add\_current\_date = True  for ele in teams\_history[team]:  if df.iloc[i]['match\_date'] == ele['match\_date']:  add\_current\_day = False  break  if add\_current\_date:  value = dict()  value['match\_date'] = df.iloc[i]['match\_date']  value['league\_id'] = int(df.iloc[i]['league\_id'])    for j in range(1, 11):  add\_history\_day\_id = True  for ele in teams\_history[team]:  if df.iloc[i][f'{x}\_team\_history\_match\_date\_{j}'] == ele['match\_date']:  add\_history\_day\_id = False  break  if add\_history\_day\_id:  value = dict()  value['match\_date'] = df.iloc[i][f'{x}\_team\_history\_match\_date\_{j}']  value['league\_id'] = int(df.iloc[i][f'{x}\_team\_history\_league\_id\_{j}'])  with open("teams\_history.json", "w") as outfile:  json.dump(teams\_history, outfile, ensure\_ascii=False)  def create\_id2name\_league():  file = open("name2id\_league.json")  data = json.load(file)  id\_ = dict()  for key in data.keys():  for x in data[key]:  if x not in id\_.keys():  id\_[f"{x}"] = key  with open("id2name\_league.json", "w") as outfile:  json.dump(id\_, outfile)  def create\_name2id\_league():  df = pd.read\_csv("train.csv", low\_memory=False)  league\_df = df[["league\_name", "league\_id"]]  league\_name = dict()  for i in range(len(league\_df)):  if league\_df.iloc[i]["league\_name"] not in league\_name.keys():  li = []  li.append(str(league\_df.iloc[i]["league\_id"]))  league\_name[f'{league\_df.iloc[i]["league\_name"]}'] = li  else:  li = league\_name[f'{league\_df.iloc[i]["league\_name"]}']  if str(league\_df.iloc[i]["league\_id"]) not in league\_name[f'{league\_df.iloc[i]["league\_name"]}']:  li.append(str(league\_df.iloc[i]["league\_id"]))  league\_name[f'{league\_df.iloc[i]["league\_name"]}'] = li  with open("league\_name.json", "w") as outfile:  json.dump(league\_name, outfile)  if \_\_name\_\_ == '\_\_main\_\_':  f = open("teams\_history.json") # created by create\_history\_team function  data = json.load(f)  f1 = open("id2name\_league.json") # created by create\_id2name\_league function  id2name = json.load(f1)  print("Input team name 1: ")  team\_1 = input()  print("Input team name 2: ")  team\_2 = input()    result = []  for his\_1 in data[team\_1]:  for his\_2 in data[team\_2]:  id\_1 = his\_1['league\_id']  id\_2 = his\_2['league\_id']  if his\_1['match\_date'] == his\_2['match\_date'] and id2name[f'{id\_1}'] == id2name[f'{id\_2}']:  value = dict()  value['match\_date'] = his\_1['match\_date']  value['league\_name'] = id2name[f'{id\_1}']  result.append(value)  if result:  if len(result) > 1:  min = result[0]  for k in range(len(result)):  for i in range(1, len(result) - 1):  if datetime.strptime(result[i]['match\_date'], '%Y-%m-%d %H:%M:%S') < datetime.strptime(min['match\_date'], '%Y-%m-%d %H:%M:%S'):  min = result[i]  result[i] = result[i-1]  result[i-1] = min  print("Last match between two teams:")  if len(result) > 5:  for re in result[:5]:  print(f"Match date: {re['match\_date']} --- League name: {re['league\_name']}\n")  else:  for re in result:  print(f"Match date: {re['match\_date']} --- League name: {re['league\_name']}\n")  else:  print("No history match between two teams") |
| --- |

1. **Task 3**

Idea: train a model with output is the score. In my case i will label for

{“0-0”: 0 , “0-1”: 1,... }. You can refer at label\_scores.json for my own rules

I trained with XGboost model. You can load model\_task\_3.json for inference(refer at inference\_task\_3.py).

Here is my code in train\_task\_3.py

| from sklearn.model\_selection import train\_test\_split, GridSearchCV  import pandas as pd  import xgboost as xgb  from sklearn.metrics import accuracy\_score  from matplotlib import pyplot  from sklearn.preprocessing import LabelEncoder  import json  def create\_label\_scores():  result = dict()  count = 1  for i in range(6):  for j in range(6):  result[f'{i}-{j}'] = count  count += 1  result['other'] = 0  with open("label\_scores.json", "w") as outfile:  json.dump(result, outfile)  def create\_cleaned\_scores():  df = pd.read\_csv("cleaned\_train.csv", low\_memory=False)  df\_scores = pd.read\_csv("train\_target\_and\_scores.csv", low\_memory=False)  f = open("label\_scores.json")  labels = json.load(f)  scores = df\_scores['score'].to\_list()  for c, s in enumerate(scores):  scores[c] = labels[s]  df = df.drop(columns="target")  data\_join = dict()  data\_join["score"] = scores  df = pd.concat([df, pd.DataFrame(data\_join)], axis=1)  df.to\_csv(f"cleaned\_scores.csv", encoding='utf-8', index=False)  df = pd.read\_csv("cleaned\_scores.csv", low\_memory=False)  train, test = train\_test\_split(df, test\_size=0.1, random\_state=69)  x\_train = train.drop(columns="score")  y\_train = train["score"]  x\_test = test.drop(columns="score")  y\_test = test["score"]  eval\_set = [(x\_train, y\_train), (x\_test, y\_test)]  #model  early\_stop = xgb.callback.EarlyStopping(  rounds=5, metric\_name='mlogloss', data\_name='validation\_1', save\_best=True  )  xgb\_model = xgb.XGBClassifier(n\_estimators=10000, learning\_rate=0.005, max\_depth=2, objective='binary:logistic')  # optimization\_dict = {'max\_depth': [2,4,6,8],  # 'n\_estimators': [50,100,150,200]}  # model = GridSearchCV(xgb\_model, optimization\_dict,  # scoring='accuracy', verbose=1)  #eval\_metric="mlogloss"  xgb\_model.fit(x\_train, y\_train, eval\_set=eval\_set, verbose=True, callbacks=[early\_stop])  xgb\_model.save\_model("model\_score.json")  # make predictions for test data  y\_pred = xgb\_model.predict(x\_test)  predictions = [round(value) for value in y\_pred]  # evaluate predictions  accuracy = accuracy\_score(y\_test, predictions)  print("Accuracy: %.2f%%" % (accuracy \* 100.0))  # retrieve performance metrics  results = xgb\_model.evals\_result()  epochs = len(results['validation\_0']['mlogloss'])  x\_axis = range(0, epochs)  # plot log loss  fig, ax = pyplot.subplots()  ax.plot(x\_axis, results['validation\_0']['mlogloss'], label='Train')  ax.plot(x\_axis, results['validation\_1']['mlogloss'], label='Test')  ax.legend()  pyplot.ylabel('Log Loss')  pyplot.title('XGBoost Log Loss')  pyplot.savefig("chart\_task\_3.jpg") |
| --- |