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")
```

2. 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")
```