import torch

import pandas as pd

import numpy as np

import torch.nn as nn

import random

import torch.optim as optim

class MLP(nn.Module):

def \_\_init\_\_(self):

super(MLP, self).\_\_init\_\_()

self.fc1 = nn.Linear(9, 128)

self.fc2 = nn.Linear(128, 64)

self.output = nn.Linear(64, 1)

self.relu = nn.ReLU()

# self.sigmoid = nn.Sigmoid() # It is usually used for classification problems, especially binary classification problems, because it compresses the output into the interval [0, 1]. However, for regression tasks, the model's output should be a continuous value, so the Sigmoid function is not necessary.

def forward(self, x):

x = self.relu(self.fc1(x))

x = self.relu(self.fc2(x))

# x = self.sigmoid(self.output(x)) # Because this is a regression problem (we are predicting specific probability values), there is no need for sigmoid.

x = self.output(x) # The last layer directly outputs the result without using an activation function.

return x

def load\_model(model\_path):

model = MLP()

model.load\_state\_dict(torch.load(model\_path))

model.eval()

return model

def load\_data(csv\_file):

data = pd.read\_csv(csv\_file, encoding='GBK')

X = data.iloc[:, :-2].values

y = data.iloc[:, -2].values.reshape(-1, 1)

return X

def save\_predictions\_to\_csv(predictions, output\_file):

df = pd.DataFrame(predictions, columns=["Prediction"])

df.to\_csv(output\_file, index=False)

def predict(model, input\_data):

inputs = torch.Tensor(input\_data)

with torch.no\_grad():

outputs = model(inputs)

predictions = outputs.numpy()

return predictions

def main():

model\_save\_path = '\The model generated during training\model\_iteration\_{i + 1}.pth' # Here, manual settings are required for importing the models to be used

# input\_data\_path = '\train\_data.csv'

input\_data\_path = 'val\_data.csv'

# output\_predictions\_path = '\Validation set prediction effect / Prediction result\_validation\_{i}.csv'

output\_predictions\_path = 'Test set prediction effect\\Prediction result\_test\_{i}.csv'

model = load\_model(model\_save\_path)

input\_data = load\_data(input\_data\_path)

predictions = predict(model, input\_data)

save\_predictions\_to\_csv(predictions, output\_predictions\_path)

if \_\_name\_\_ == "\_\_main\_\_":

main()