import pandas as pd

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.metrics import mean\_absolute\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from scipy.stats import zscore

# Load dataset

df = pd.read\_csv("games.csv")

# Function to extract median from interval

def extract\_median(interval):

    try:

        low, high = interval.replace(',', '').split(' - ')

        low, high = int(low), int(high)

        return (low + high) / 2

    except Exception as e:

        return np.nan

df['Estimated owners'] = df['Estimated owners'].apply(extract\_median)

# Fill missing values with the median

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].median())

# Calculate number of supported languages

df["Nr lang"] = df['Supported languages'].apply(lambda x: len(x.split(',')) if pd.notna(x) and x != '' else 0)

# Calculate number of platforms

df["Nr plat"] = df[['Windows', 'Mac', 'Linux']].apply(lambda row: sum(row == "TRUE"), axis=1)

# Initialize 'Raport' column before applying calculations to avoid KeyError

df['Raport'] = 0

# Calculate the ratio of Positive to Negative reviews

df['Raport'] = df.apply(lambda row: row['Positive'] / row['Negative'] if row['Negative'] != 0 else df['Raport'].median(), axis=1)

# Fill missing values in 'Score rank'

df['Score rank'].fillna(40000, inplace=True)

# Remove outliers

z\_scores = np.abs(zscore(df[numeric\_cols]))

threshold = 3

outliers = (z\_scores > threshold).any(axis=1)

df = df[~outliers]

# Ensure 'Raport' column is created correctly

if 'Raport' not in df.columns:

    raise KeyError("'Raport' column is missing from the DataFrame")

# Define features and target

features = ['Estimated owners', 'Peak CCU', 'DLC count', 'Nr lang', 'Nr plat', 'Metacritic score',

            'User score', 'Positive', 'Negative', 'Raport', 'Achievements', 'Score rank', 'Recommendations',

            'Average playtime forever', 'Average playtime two weeks', 'Median playtime forever', 'Median playtime two weeks']

target = 'Price'

X = df[features]

y = df[target].values.ravel()

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Scale the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Convert to PyTorch tensors

X\_train\_tensor = torch.tensor(X\_train\_scaled, dtype=torch.float32)

X\_test\_tensor = torch.tensor(X\_test\_scaled, dtype=torch.float32)

y\_train\_tensor = torch.tensor(y\_train, dtype=torch.float32).view(-1, 1)

y\_test\_tensor = torch.tensor(y\_test, dtype=torch.float32).view(-1, 1)

# Define the ANN model

class SimpleANN(nn.Module):

    def \_\_init\_\_(self, input\_dim):

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

        self.fc1 = nn.Linear(input\_dim, 128)

        self.fc2 = nn.Linear(128, 64)

        self.fc3 = nn.Linear(64, 32)

        self.fc4 = nn.Linear(32, 1)

        self.dropout = nn.Dropout(0.5)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = self.dropout(x)

        x = torch.relu(self.fc2(x))

        x = self.dropout(x)

        x = torch.relu(self.fc3(x))

        x = self.dropout(x)

        x = self.fc4(x)

        return x

# Initialize the model, loss function, and optimizer

input\_dim = X\_train\_tensor.shape[1]

model = SimpleANN(input\_dim)

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model

num\_epochs = 1000

for epoch in range(num\_epochs):

    model.train()

    optimizer.zero\_grad()

    output = model(X\_train\_tensor)

    loss = criterion(output, y\_train\_tensor)

    loss.backward()

    optimizer.step()

    if (epoch + 1) % 100 == 0:

        print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')

# Evaluating the model

model.eval()

with torch.no\_grad():

    predictions = model(X\_test\_tensor)

    mae = mean\_absolute\_error(y\_test\_tensor, predictions)

    print(f'Mean Absolute Error: {mae:.4f}')

CNN:

import torch

import torch.nn as nn

import torch.optim as optim

# Define the CNN model

class SimpleCNN(nn.Module):

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

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

self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=32, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2, padding=0)

self.fc1 = nn.Linear(64 \* 7 \* 7, 128)

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

self.fc3 = nn.Linear(64, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

x = x.view(-1, 64 \* 7 \* 7)

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

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

x = self.fc3(x)

return x

# Initialize the model, loss function, and optimizer

model = SimpleCNN()

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

RNN

**Recurrent Neural Networks (RNN)**

RNNs are designed for sequential data, such as time series or natural language. They have loops that allow information to persist.

**Basic Structure:**

* Input Layer: Takes the sequential data.
* RNN Layers: Process the sequence step-by-step.
* Output Layer: Produces the final output.

**Example Code:**

python

Copy code

import torch

import torch.nn as nn

import torch.optim as optim

# Define the RNN model

class SimpleRNN(nn.Module):

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim):

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

self.rnn = nn.RNN(input\_dim, hidden\_dim, batch\_first=True)

self.fc = nn.Linear(hidden\_dim, output\_dim)

def forward(self, x):

h0 = torch.zeros(1, x.size(0), hidden\_dim).to(x.device)

out, \_ = self.rnn(x, h0)

out = self.fc(out[:, -1, :])

return out

# Initialize the model, loss function, and optimizer

input\_dim = 10 # Example input dimension

hidden\_dim = 20

output\_dim = 1

model = SimpleRNN(input\_dim, hidden\_dim, output\_dim)

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

cu ML:

import os

import pandas as pd

import numpy as np

import torch

import torch.nn

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.model\_selection import train\_test\_split

from scipy.stats import zscore

df = pd.read\_csv("games.csv")

df.describe()

def extract\_median(interval):

try:

low, high = interval.replace(',', '').split(' - ')

low, high = int(low), int(high)

return (low + high) / 2

except Exception as e:

return np.nan

df['Estimated owners'] = df['Estimated owners'].apply(extract\_median)

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].median())

df["Nr lang"] = 0

for i in range(len(df)):

supported\_languages = df.at[i, 'Supported languages']

if pd.notna(supported\_languages) and supported\_languages != '':

df.at[i, 'Nr lang'] = len(supported\_languages.split(','))

df["Nr plat"] = 0

for i in range(len(df)):

wind = df.at[i, 'Windows']

if wind == "TRUE":

df.at[i, "Nr plat"] += 1

mac = df.at[i, 'Mac']

if mac == "TRUE":

df.at[i, "Nr plat"] += 1

lin = df.at[i, 'Linux']

if lin == "TRUE":

df.at[i, "Nr plat"] += 1

df['Raport'] = 0

for i in range(len(df)):

if df.at[i, 'Negative'] != 0:

df.at[i, 'Raport'] = df.at[i, 'Positive'] / df.at[i, 'Negative']

df.loc[df['Negative'] == 0, 'Raport'] = df['Raport'].median()

df['Score rank'].fillna(40000)

z\_scores = np.abs(zscore(df[numeric\_cols]))

threshold = 3

outliers = (z\_scores > threshold).any(axis=1)

df = df[~outliers]

features = ['Estimated owners', 'Peak CCU', 'DLC count',

'Nr lang', 'Nr plat', 'Metacritic score',

'User score', 'Positive', 'Negative', 'Raport', 'Achievements',

'Score rank', 'Recommendations',

'Average playtime forever', 'Average playtime two weeks',

'Median playtime forever', 'Median playtime two weeks' ]

target = ['Price']

X = df[features]

y = df[target].values.ravel()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 42)

gradient = GradientBoostingRegressor()

gradient.fit(X\_train, y\_train)

y\_pred = gradient.predict(X\_test)

print(mean\_absolute\_error(y\_pred, y\_test))