import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, TensorDataset

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

from sklearn.preprocessing import StandardScaler

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, roc\_curve, ConfusionMatrixDisplay

from skopt import BayesSearchCV

from skopt.space import Real, Integer

# Load and preprocess data

data = pd.read\_csv("/content/drive/MyDrive/Colab Notebooks/newfinal4April.csv", names=["depDiab", "age", "smoke", "alcohol", "sex", "Urbanrural", "edu\_group", "castetribe", "bmi\_group", "wealthindx\_group", "religion\_group", "nonveg", "unhealthy", "edu\_continuous", "wealthindx\_continuous", "bmi\_cont", "state",])

data["smoke.Y"] = [1 if s == 1 else 0 for s in data["smoke"]]

data["smoke.N"] = [1 if s == 0 else 0 for s in data["smoke"]]

data["alcohol.Y"] = [1 if s == 1 else 0 for s in data["alcohol"]]

data["alcohol.N"] = [1 if s == 0 else 0 for s in data["alcohol"]]

data["sex.F"] = [1 if s == 1 else 0 for s in data["sex"]]

data["sex.M"] = [1 if s == 0 else 0 for s in data["sex"]]

data["urban"] = [1 if s == 0 else 0 for s in data["Urbanrural"]]

data["rural"] = [1 if s == 1 else 0 for s in data["Urbanrural"]]

data["caste.1"] = [1 if s == 1 else 0 for s in data["castetribe"]]

data["caste.2"] = [1 if s == 2 else 0 for s in data["castetribe"]]

data["caste.3"] = [1 if s == 3 else 0 for s in data["castetribe"]]

data["caste.4"] = [1 if s == 4 else 0 for s in data["castetribe"]]

data["religion.1"] = [1 if s == 1 else 0 for s in data["religion\_group"]]

data["religion.2"] = [1 if s == 2 else 0 for s in data["religion\_group"]]

data["religion.3"] = [1 if s == 3 else 0 for s in data["religion\_group"]]

data["religion.4"] = [1 if s == 4 else 0 for s in data["religion\_group"]]

data["religion.5"] = [1 if s == 5 else 0 for s in data["religion\_group"]]

data["religion.6"] = [1 if s == 6 else 0 for s in data["religion\_group"]]

y = data["depDiab"]

X = data[["age", "nonveg", "unhealthy", "edu\_continuous", "wealthindx\_continuous", "bmi\_cont", "smoke.Y", "smoke.N", "alcohol.Y", "alcohol.N", "sex.F", "sex.M", "urban", "rural", "caste.1", "caste.2", "caste.3", "caste.4", "religion.1", "religion.2", "religion.3", "religion.4", "religion.5", "religion.6"]]

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

# Standardize features

scaler = StandardScaler()

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

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

# Convert to PyTorch tensors

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

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

y\_train = torch.tensor(y\_train.values, dtype=torch.long)

y\_test = torch.tensor(y\_test.values, dtype=torch.long)

train\_dataset = TensorDataset(X\_train, y\_train)

test\_dataset = TensorDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

# Define the neural network model

class DNNModel(nn.Module):

def \_\_init\_\_(self, units=64, dropout\_rate=0.5):

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

self.fc1 = nn.Linear(X\_train.shape[1], units)

self.dropout = nn.Dropout(dropout\_rate)

self.fc2 = nn.Linear(units, units // 2)

self.fc3 = nn.Linear(units // 2, 3) # Output layer with 3 units for classification

def forward(self, x):

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

x = self.dropout(x)

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

x = self.fc3(x)

return x

# Train the model

def train\_model(model, train\_loader, criterion, optimizer, num\_epochs=50):

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, labels in train\_loader:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item() \* inputs.size(0)

epoch\_loss = running\_loss / len(train\_loader.dataset)

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

# Evaluate the model

def evaluate\_model(model, test\_loader):

model.eval()

all\_preds = []

with torch.no\_grad():

for inputs, labels in test\_loader:

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.numpy())

return all\_preds

# Create and train the model

model = DNNModel(units=64, dropout\_rate=0.5)

criterion = nn.CrossEntropyLoss()

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

train\_model(model, train\_loader, criterion, optimizer, num\_epochs=100)

# Evaluate the model

y\_pred = evaluate\_model(model, test\_loader)

# Metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

roc\_auc = roc\_auc\_score(y\_test, y\_pred, multi\_class='ovr')

print(f"Accuracy: {accuracy \* 100:.2f}%")

print(f"Precision: {precision \* 100:.2f}%")

print(f"Recall: {recall \* 100:.2f}%")

print(f"F1 Score: {f1 \* 100:.2f}%")

print(f"AUC: {roc\_auc \* 100:.2f}%")

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(confusion\_matrix=cm).plot()

# ROC Curve

test\_fpr, test\_tpr, \_ = roc\_curve(y\_test, y\_pred)

plt.plot(test\_fpr, test\_tpr, label=f"AUC = {roc\_auc:.2f}")

plt.plot([0, 1], [0, 1], 'g--')

plt.legend()

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("AUC (ROC curve)")

plt.grid(color='black', linestyle='-', linewidth=0.5)

plt.show()

# Compute SHAP Values (using PyTorch-based approach)

import shap

model.eval() # Set the model to evaluation mode

X\_train\_sample = shap.sample(X\_train.numpy(), 100)

# Define a PyTorch-based explainer for SHAP

explainer = shap.DeepExplainer(model, X\_train\_sample)

shap\_values = explainer.shap\_values(X\_test.numpy())

# Plot SHAP values

shap.summary\_plot(shap\_values, X\_test.numpy(), feature\_names=X.columns)

shap.summary\_plot(shap\_values, X\_test.numpy(), plot\_type="bar", feature\_names=X.columns)

This code does the following:

1. Loads and preprocesses the data, including standardizing features.
2. Converts data to PyTorch tensors and creates data loaders.
3. Defines a PyTorch neural network model.
4. Trains the model and evaluates its performance.
5. Computes and plots SHAP values for feature importance analysis.

Please ensure you have the required libraries installed:

pip install torch torchvision shap scikit-learn pandas matplotlib seaborn scikit-optimize