```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from torch.utils.data import TensorDataset, DataLoader


# Load dataset
data = pd.read_csv("customers.csv")
data.head()


data.columns


# Drop ID column as it's not useful for classification
data = data.drop(columns=["ID"])


# Handle missing values
data.fillna({"Work_Experience": 0, "Family_Size": data["Family_Size"].median()}, inplace=True)


# Encode categorical variables
categorical_columns = ["Gender", "Ever_Married", "Graduated", "Profession", "Spending_Score", "Var_1"]
for col in categorical_columns:
    data[col] = LabelEncoder().fit_transform(data[col])


# Encode target variable
label_encoder = LabelEncoder()
data["Segmentation"] = label_encoder.fit_transform(data["Segmentation"])  # A, B, C, D -> 0, 1, 2, 3


# Split features and target
X = data.drop(columns=["Segmentation"])
y = data["Segmentation"].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Convert to 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, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)


# Create DataLoader
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_da                        uffle=True)
test_loader = DataLoader(test_data
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by NARESH R
13:58 (6 minutes ago)
executed in 57.232 s

```
# Define Neural Network(Model1)
class PeopleClassifier(nn.Module):
    def __init__(self, input_size):
        super(PeopleClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 4)  # 4 classes (A, B, C, D)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.fc3(x)   # No softmax here (CrossEntropyLoss handles it)
        return x







# Training Loop
def train_model(model, train_loader, criterion, optimizer, epochs):
    model.train()
    for epoch in range(epochs):
        for X_batch, y_batch in train_loader:
            outputs = model(X_batch)
            loss = criterion(outputs, y_batch)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if (epoch + 1) % 10 == 0:
            print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')






    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')


# Initialize model
input_size = X_train.shape[1]

model = PeopleClassifier(input_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


train_model(model, train_loader, criterion, optimizer, epochs=100)


# Evaluation
model.eval()
predictions, actuals = [], []
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        outputs = model(X_batch)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.numpy())
        actuals.extend(y_batch.numpy())


# Compute metrics
accuracy = accuracy_score(actuals, predictions)
conf_matrix = confusion_matrix(actuals, predictions)
class_report = classification_report(actuals, predictions, target_names=[str(i) for i in label_encoder.classes_])
print("Name:           ")
print("Register No:       ")
print(f'Test Accuracy: {accuracy:.2f}%')
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)


import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(conf_matrix, annot=Tru                        bels=label_encoder.classes_, yticklabels=label_encoder.classes_,fm
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()


# Prediction for a sample input
sample_input = X_test[12].clone().unsqueeze(0).detach().type(torch.float32)
with torch.no_grad():
    output = model(sample_input)
```

Run cell (Ctrl+Enter)
cell executed since last change

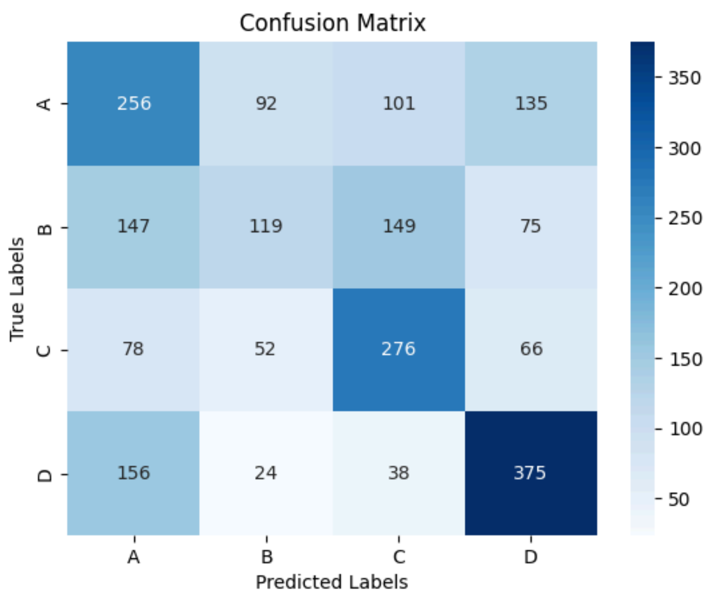
executed by NARESH R
13:58 (6 minutes ago)
executed in 57.232 s

```
        # Select the prediction for the sample (first element)
        predicted_class_index = torch.argmax(output[0]).item()
        predicted_class_label = label_encoder.inverse_transform([predicted_class_index])[0]
    print("Name:NARESH.R")
    print("Register No:212223240104")
    print(f'Predicted class for sample input: {predicted_class_label}')
    print(f'Actual class for sample input: {label_encoder.inverse_transform([y_test[12].item()])[0]}')
```

```
Epoch [10/100], Loss: 1.3714
Epoch [20/100], Loss: 1.0607
Epoch [30/100], Loss: 1.3445
Epoch [40/100], Loss: 1.1011
Epoch [50/100], Loss: 1.0033
Epoch [60/100], Loss: 1.0042
Epoch [70/100], Loss: 1.0742
Epoch [80/100], Loss: 1.0381
Epoch [90/100], Loss: 1.2266
Epoch [100/100], Loss: 1.0980
Epoch [100/100], Loss: 1.0980
Name:
Register No:
Test Accuracy: 0.48%
Confusion Matrix:
 [[256  92 101 135]
 [147 119 149  75]
 [ 78  52 276  66]
 [156  24  38 375]]
Classification Report:
               precision    recall  f1-score   support

           A       0.40      0.44      0.42       584
           B       0.41      0.24      0.31       490
           C       0.49      0.58      0.53       472
           D       0.58      0.63      0.60       593

    accuracy                           0.48      2139
   macro avg       0.47      0.47      0.47      2139
weighted avg       0.47      0.48      0.47      2139
```



```
Name:NARESH.R
Register No:212223240104
Predicted class for sample input: D
Actual class for sample input: D
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by NARESH R
13:58 (6 minutes ago)
executed in 57.232 s