Rizqy Asyraff Athallah

1103210158

Week 10

1. Load dan Preprocess Dataset

```
# Load dataset
data = pd.read_csv('/content/drive/MyDrive/StudentsPerformance.csv')

# Define target (categorical labels) and features
# Convert 'math score' to categorical labels
def categorize_math_score(score):
    if score < 50:
        return 0  # Low
    elif 50 <= score <= 75:
        return 1  # Medium
    else:
        return 2  # High

data['math_category'] = data['math score'].apply(categorize_math_score)

# Features (X) and target (y)
X = data.drop(columns=['math score', 'math_category'])
y = data['math_category']

# Preprocessing
categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(exclude=['object']).columns

# Encode categorical columns
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Perbarui ke 'sparse_output'
X_encoded = encoder.fit_transform(X[categorical_cols])


# Scale numerical columns
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X[numerical_cols])

# Combine encoded categorical and scaled numerical features
X_processed = torch.tensor(
    pd.concat(
        [pd.DataFrame(X_encoded), pd.DataFrame(X_scaled)],
        axis=1
```

```
    ).values,
    dtype=torch.float32
)

# Convert target to tensor
y_processed = torch.tensor(y.values, dtype=torch.long)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y_processed, test_size=0.2, random_state=42)

# Define DataLoader
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)
```

- **Target (math_category)**: Kolom math score diubah menjadi kategori low, medium, dan high.
- Data kategorikal diencoding menggunakan **OneHotEncoder**.
- Data numerik distandarisasi menggunakan **StandardScaler**.
- **Split Data**: Dataset dibagi menjadi training (80%) dan testing (20%) untuk melatih dan menguji model.

2. Definisi Model

```
# Define MLP model
class MLPClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLPClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.softmax(x)
        return x

# Model parameters
input_size = X_train.shape[1]  # Jumlah fitur setelah preprocessing
hidden_size = 32               # 1 hidden layer dengan 32 neuron
```

```
output_size = 3               # 3 kelas: low, medium, high

model = MLPClassifier(input_size, hidden_size, output_size)
```

- Input layer berdasarkan jumlah fitur yang tersedia setelah preprocessing.
- Hidden layer menggunakan **ReLU** sebagai fungsi aktivasi.
- Output layer menggunakan **Softmax** untuk menghasilkan probabilitas untuk setiap kelas.
- input_size: Dimensi input (jumlah fitur).
- hidden_size: Jumlah neuron di hidden layer (32).
- output_size: Jumlah kelas target (3).

3. Training Model

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()  # Loss untuk klasifikasi multi-kelas
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  # Optimizer Adam

# Training loop
num_epochs = 50  # Jumlah epoch
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        inputs, labels = batch
        outputs = model(inputs)           # Forward pass
        loss = criterion(outputs, labels)  # Hitung loss

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

    # Print loss for every 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

- **CrossEntropyLoss** digunakan untuk menghitung seberapa baik prediksi model sesuai dengan label target.
- **Adam Optimizer** dipakai untuk mempercepat konvergensi.
- Model dilatih dengan batch data menggunakan train_loader.
- Loss dicetak setiap 10 epoch untuk memantau performa.

4. Evaluasi Model

```
# Evaluation
model.eval()  # Set model ke mode evaluasi
y_pred = []
```

```
y_true = []

with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)  # Ambil kelas dengan probabilitas tertinggi
        y_pred.extend(predicted.numpy())
        y_true.extend(labels.numpy())

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

- Model tidak melakukan backpropagation saat evaluasi.
- Probabilitas dikonversi menjadi kelas dengan nilai tertinggi menggunakan torch.max.
- Metrik akurasi digunakan untuk mengevaluasi performa model.