Rizqy Asyraff Athallah

1103210158

Week 11

- **Load dan Explore Data**

```
# Menampilkan beberapa baris pertama
print("Data Overview:\n", data.head())
print("\nData Info:\n")
data.info()
```

  - Dataset heart.csv dimuat menggunakan pandas.
  - data.head() menampilkan 5 baris pertama dataset untuk memahami struktur data.
  - data.info() memberikan informasi tentang tipe data setiap kolom dan ada/tidaknya nilai kosong (missing values).

- **Prepocessing**

```
# Step 2: Preprocessing
# Memisahkan fitur dan target
X = data.drop(columns=['target'])
y = data['target']

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Membagi data ke dalam set pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)
```

  - X berisi semua kolom kecuali target (fitur), dan y berisi kolom target (label kelas).
  - StandardScaler digunakan untuk menstandarkan fitur agar memiliki rata-rata 0 dan standar deviasi 1.
  - train_test_split memisahkan data menjadi 80% untuk pelatihan dan 20% untuk pengujian.
  - Data dalam format numpy diubah menjadi tensor untuk digunakan dalam model PyTorch.

```
# Step 3: Membuat kelas himpunan data kustom
class HeartDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
```

```
        self.labels = labels

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

# Mmembuat dataset
dataset_train = HeartDataset(X_train_tensor, y_train_tensor)
dataset_test = HeartDataset(X_test_tensor, y_test_tensor)
```

- Membuat kelas dataset kustom untuk mengelola data menggunakan PyTorch.
- **__len__**: Mengembalikan jumlah data.
- **__getitem__**: Mengembalikan fitur dan label untuk indeks tertentu.
- Dataset ini digunakan dalam DataLoader untuk mempermudah proses batching.

- **Menentukan Kelas Model MLP**

```
# Step 4: Menentukan kelas model MLP
class MLPClassifier(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(MLPClassifier, self).__init__()
        layers = []
        in_features = input_size

        # Tambahkan lapisan tersembunyi
        for hidden_units in hidden_layers:
            layers.append(nn.Linear(in_features, hidden_units))
            layers.append(activation_fn)
            in_features = hidden_units

        # Tambahkan lapisan output
        layers.append(nn.Linear(in_features, 2))  # Binary classification (2 classes)
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

- hidden_layers: List yang menentukan jumlah neuron pada setiap hidden layer.
- **activation_fn**: Fungsi aktivasi yang digunakan setelah setiap layer.
- Menggunakan 2 neuron karena klasifikasi biner (CrossEntropyLoss mengharapkan output sebanyak jumlah kelas).
- **forward**: Mendefinisikan alur data melalui jaringan.

- **Menentukan Fungsi Pelatihan dan Evaluasi**

```
# Step 5: Tentukan fungsi pelatihan dan evaluasi
```

```
def train_and_evaluate_model(hidden_layers, activation_fn, epochs, learning_rate, batch_size):
    # Define model
    model = MLPClassifier(input_size=X_train.shape[1], hidden_layers=hidden_layers,
activation_fn=activation_fn)

    # Kerugian dan pengoptimal
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # Memeuat data
    train_loader = DataLoader(dataset_train, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(dataset_test, batch_size=batch_size, shuffle=False)

    # Latih
    for epoch in range(epochs):
        model.train()
        for features, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(features)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

    # Evaluasi
    model.eval()
    all_preds = []
    with torch.no_grad():
        for features, _ in test_loader:
            preds = model(features)
            all_preds.append(preds.argmax(dim=1).numpy())

    all_preds = np.concatenate(all_preds)
    accuracy = accuracy_score(y_test, all_preds)
    return accuracy
```

- **Model Initialization**: Model MLP diinisialisasi dengan jumlah neuron dan fungsi aktivasi yang diberikan.
- **Loss dan Optimizer**: CrossEntropyLoss untuk klasifikasi dan Adam untuk optimasi.

- **Menjalankan Experiment**

```
# Step 6: Menjalankan eksperimen dengan berbagai konfigurasi
activation_functions = {
    "ReLU": nn.ReLU(),
    "Sigmoid": nn.Sigmoid(),
    "Tanh": nn.Tanh(),
```

```
    "Softmax": nn.Softmax(dim=1),
    "Linear": nn.Identity()
}

hidden_layer_configs = [[4], [8, 4], [16, 8, 4]]
epochs_list = [1, 10, 25, 50, 100, 250]
learning_rates = [10, 1, 0.1, 0.01, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128, 256, 512]

results = []
for activation_name, activation_fn in activation_functions.items():
    for hidden_layers in hidden_layer_configs:
        for epochs in epochs_list:
            for lr in learning_rates:
                for batch_size in batch_sizes:
                    accuracy = train_and_evaluate_model(
                        hidden_layers=hidden_layers,
                        activation_fn=activation_fn,
                        epochs=epochs,
                        learning_rate=lr,
                        batch_size=batch_size
                    )
                    results.append({
                        "Activation": activation_name,
                        "Hidden Layers": hidden_layers,
                        "Epochs": epochs,
                        "Learning Rate": lr,
                        "Batch Size": batch_size,
                        "Accuracy": accuracy
                    })
```

- Melakukan eksperimen dengan berbagai konfigurasi fungsi aktivasi, hidden layers, jumlah epoch, learning rate, dan ukuran batch.
- Hasil akurasi dari setiap kombinasi disimpan dalam list results.

- **Menyimpan Visualisasi Hasil**

```
# Mengonversi hasil ke DataFrame dan tampilkan
results_df = pd.DataFrame(results)
print("\nExperiment Results:\n", results_df.head())

# Simpan hasil eksperimen
results_df.to_csv('/content/drive/MyDrive/Week 11/experiment_results.csv', index=False)
```

- Hasil eksperimen dikonversi menjadi DataFrame untuk dianalisis atau divisualisasikan.