Laporan Heart Dataset

Nama : Anita Firda Nuralifah

NIM : 1103213117

Library

```
[1] import pandas as pd  #untuk manipulasi dan analisis data.
    import numpy as np #untuk komputasi numerik dengan array dan matriks.
    from sklearn.model_selection import train_test_split #untuk membagi dataset menjadi data latih dan data uji.
    from sklearn.preprocessing import StandardScaler #untuk melakukan preprocessing data, seperti standarisasi fitur.
    from sklearn.metrics import accuracy_score #untuk menghitung akurasi model klasifikasi.
    import torch #untuk membangun dan melatih model neural network.
    import torch.nn as nn #Modul PyTorch yang berisi berbagai lapisan dan fungsi untuk membangun neural network.
    import torch.optim as optim #Modul PyTorch yang berisi berbagai algoritma optimasi untuk melatih model neural network.
    from torch.utils.data import DataLoader, TensorDataset #untuk membuat dan mengelola dataset, serta memuat data secara efisien selama pelatihan.
```

- import pandas as pd : untuk manipulasi dan analisis data.
- import numpy as np : untuk komputasi numerik dengan array dan matriks.
- from sklearn.model_selection import train_test_split : untuk membagi dataset menjadi data latih dan data uji.
- from sklearn.preprocessing import StandardScaler : untuk melakukan preprocessing data, seperti standarisasi fitur.
- from sklearn.metrics import accuracy_score : untuk menghitung akurasi model klasifikasi.
- import torch : untuk membangun dan melatih model neural network.
- import torch.nn as nn : Modul PyTorch yang berisi berbagai lapisan dan fungsi untuk membangun neural network.
- import torch.optim as optim : Modul PyTorch yang berisi berbagai algoritma optimasi untuk melatih model neural network.
- from torch.utils.data import DataLoader, TensorDataset : untuk membuat dan mengelola dataset, serta memuat data secara efisien selama pelatihan.

```
df = pd.read_csv('/content/heart.csv')
df.head()
#memuat dataset
```

Memuat dataset

```
[3] X = df.drop('target', axis=1).values
    y = df['target'].values
    #memisahkan fitur dan target
```

Kode tersebut digunakan untuk memisahkan data menjadi fitur (input) dan target (output/label) dari sebuah DataFrame bernama df. Baris pertama, X = df.drop('target', axis=1).values, berfungsi menghapus kolom bernama 'target' dari DataFrame untuk mendapatkan data input atau fitur, kemudian mengonversinya menjadi array NumPy yang disimpan dalam variabel X. Baris kedua, y = df['target'].values, mengambil nilai-nilai dari kolom 'target' dalam bentuk array NumPy, yang merepresentasikan data output atau label, dan menyimpannya dalam variabel y.

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
#normalisasi data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
#split data menjadi training dan testing
```

```
[5] X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train, dtype=torch.long)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_test_tensor = torch.tensor(y_test, dtype=torch.long)
    #mengonversi data ke dalam bentuk tensor
```

- Melakukan normalisasi data dan membagi dataset menjadi data pelatihan (training) dan pengujian (testing). Dengan menggunakan StandardScaler dari sklearn, data fitur (X) dinormalisasi agar memiliki distribusi standar (mean = 0 dan standar deviasi = 1). Kemudian, fungsi train_test_split digunakan untuk membagi dataset menjadi data pelatihan dan pengujian, dengan 30% data dialokasikan sebagai data pengujian (test_size=0.3) dan sisanya sebagai data pelatihan. Parameter random_state=42 digunakan untuk memastikan pembagian data bersifat deterministik.
- Mengonversi data pelatihan dan pengujian ke dalam bentuk tensor menggunakan pustaka PyTorch. Variabel X_train, y_train, X_test, dan y_test diubah menjadi tensor dengan tipe data tertentu. Fitur (X) dikonversi menjadi tensor bertipe float32, sedangkan target (y) dikonversi menjadi tensor bertipe long. Konversi ini diperlukan untuk mempersiapkan data agar dapat digunakan dalam model berbasis PyTorch.

```python
def create_mlp(input_size, hidden_layers, hidden_neurons, activation_function):
    layers = []
    layers.append(nn.Linear(input_size, hidden_neurons))
    #lapisan input ke lapisan tersembunyi pertama

    for _ in range(hidden_layers - 1):
        layers.append(nn.Linear(hidden_neurons, hidden_neurons))
    #menambahkan lapisan tersembunyi tambahan

    if activation_function == 'linear':
        activation = nn.Identity()
    elif activation_function == 'Sigmoid':
        activation = nn.Sigmoid()
    elif activation_function == 'ReLU':
        activation = nn.ReLU()
    elif activation_function == 'Softmax':
        activation = nn.Softmax(dim=1)
    elif activation_function == 'Tanh':
        activation = nn.Tanh()
    #memilih fungsi aktivasi

    layers.append(nn.Linear(hidden_neurons, 2))
    #menambahkan lapisan output
```

```python
def create_mlp(input_size, hidden_layers, hidden_neurons, activation_function):
    layers = []
    layers.append(nn.Linear(input_size, hidden_neurons))

    for _ in range(hidden_layers - 1):
        layers.append(nn.Linear(hidden_neurons, hidden_neurons))
    #menambahkan lapisan tersembunyi tambahan

    if activation_function == 'linear':
        activation = nn.Identity()
    elif activation_function == 'Sigmoid':
        activation = nn.Sigmoid()
    elif activation_function == 'ReLU':
        activation = nn.ReLU()
    elif activation_function == 'Softmax':
        activation = nn.Softmax(dim=1)
    elif activation_function == 'Tanh':
        activation = nn.Tanh()
    #memilih fungsi aktivasi

    layers.append(nn.Linear(hidden_neurons, 2))
    #menambahkan lapisan output

    model = nn.Sequential(*layers)
    return model
    #menggabungkan semua lapisan menjadi satu model

#membuat model MLP
```

Membuat model MLP

```python
def train_and_evaluate(model, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, batch_size, epochs, learning_rate):
    train_data = TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
    #dataLoader untuk pelatihan dalam batch

    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    criterion = nn.CrossEntropyLoss()
    #optimizer (Adam) dan fungsi loss (CrossEntropyLoss untuk klasifikasi)

    model.train()
    for epoch in range(epochs):
        for X_batch, y_batch in train_loader:
            optimizer.zero_grad()
            output = model(X_batch)
            loss = criterion(output, y_batch)
            loss.backward()
            optimizer.step()
    #loop pelatihan

    model.eval()
    with torch.no_grad():
    #evaluasi model setelah pelatihan
        train_output = model(X_train_tensor)
        train_pred = torch.argmax(train_output, dim=1)
        train_accuracy = accuracy_score(y_train_tensor.numpy(), train_pred.numpy())
        train_loss = criterion(train_output, y_train_tensor).item()
        #akurasi pelatihan
        test_output = model(X_test_tensor)
        test_pred = torch.argmax(test_output, dim=1)
        test_accuracy = accuracy_score(y_test_tensor.numpy(), test_pred.numpy())
        test_loss = criterion(test_output, y_test_tensor).item()
        #akurasi pengujian

    return train_accuracy, train_loss, test_accuracy, test_loss
    #mengembalikan hasil

#fungsi untuk melatih dan mengevaluasi model
```

Melatih dan mengevaluasi model

```python
results = []  # Menyimpan hasil eksperimen
best_result = None
worst_result = None
#melakukan eksperimen dengan semua kombinasi hyperparameter

for hidden_layers in hidden_layers_options:
    for hidden_neurons in hidden_neurons_options:
        for activation_function in activation_functions:
            for epochs in epochs_options:
                for learning_rate in learning_rates:
                    for batch_size in batch_sizes:
                        print(f"Melatih dengan {hidden_layers} lapisan tersembunyi, {hidden_neurons} neuron, fungsi aktivasi {activation_function}, "
                              f"{epochs} epoch, tingkat pembelajaran {learning_rate}, ukuran batch {batch_size}")
                        #mencetak konfigurasi hyperparameter yang sedang digunakan

                        model = create_mlp(X_train_tensor.shape[1], hidden_layers, hidden_neurons, activation_function)
                        #membuat model dengan kombinasi hyperparameter tertentu

                        train_acc, train_loss, test_acc, test_loss = train_and_evaluate(
                            model, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, batch_size, epochs, learning_rate
                        )
                        #melatih dan mengevaluasi model

                        results.append({
                            'Hidden Layers': hidden_layers,
                            'Hidden Neurons': hidden_neurons,
                            'Activation Function': activation_function,
                            'Epochs': epochs,
                            'Learning Rate': learning_rate,
                            'Batch Size': batch_size,
                            'Train Accuracy': train_acc,
                            'Train Loss': train_loss,
                            'Test Accuracy': test_acc,
                            'Test Loss': test_loss
                        })
                        #menyimpan hasil eksperimen
```

```python
                        print(f"Akurasi Pelatihan: {train_acc * 100:.2f}%")
                        print(f"Loss Pelatihan: {train_loss:.4f}")
                        print(f"Akurasi Pengujian: {test_acc * 100:.2f}%")
                        print(f"Loss Pengujian: {test_loss:.4f}\n")
                        #mencetak hasil akhir dari eksperimen saat ini

                        if best_result is None or test_acc > best_result['Test Accuracy']:
                            best_result = {
                                'Hyperparameters': {
                                    'Hidden Layers': hidden_layers,
                                    'Hidden Neurons': hidden_neurons,
                                    'Activation Function': activation_function,
                                    'Epochs': epochs,
                                    'Learning Rate': learning_rate,
                                    'Batch Size': batch_size
                                },
                                'Test Accuracy': test_acc
                            }
                        if worst_result is None or test_acc < worst_result['Test Accuracy']:
                            worst_result = {
                                'Hyperparameters': {
                                    'Hidden Layers': hidden_layers,
                                    'Hidden Neurons': hidden_neurons,
                                    'Activation Function': activation_function,
                                    'Epochs': epochs,
                                    'Learning Rate': learning_rate,
                                    'Batch Size': batch_size
                                },
                                'Test Accuracy': test_acc
                            }
                            #memperbarui hasil terbaik dan terburuk
```

Melakukan pengujian parameter

```
print("\nBest Hyperparameter Configuration:")
print(f"Hidden Layers: {best_result['Hyperparameters']['Hidden Layers']}")
print(f"Hidden Neurons: {best_result['Hyperparameters']['Hidden Neurons']}")
print(f"Activation Function: {best_result['Hyperparameters']['Activation Function']}")
print(f"Epochs: {best_result['Hyperparameters']['Epochs']}")
print(f"Learning Rate: {best_result['Hyperparameters']['Learning Rate']}")
print(f"Batch Size: {best_result['Hyperparameters']['Batch Size']}")
print(f"Test Accuracy: {best_result['Test Accuracy'] * 100:.2f}%")
#menampilkan best dan worst hyperparameter berdasarkan akurasi
```

```
Best Hyperparameter Configuration:
Hidden Layers: 2
Hidden Neurons: 8
Activation Function: Tanh
Epochs: 10
Learning Rate: 0.001
Batch Size: 128
Test Accuracy: 83.44%
```

Menampilkan parameter terbaik

```
[10] print("\nWorst Hyperparameter Configuration:")
print(f"Hidden Layers: {worst_result['Hyperparameters']['Hidden Layers']}")
print(f"Hidden Neurons: {worst_result['Hyperparameters']['Hidden Neurons']}")
print(f"Activation Function: {worst_result['Hyperparameters']['Activation Function']}")
print(f"Epochs: {worst_result['Hyperparameters']['Epochs']}")
print(f"Learning Rate: {worst_result['Hyperparameters']['Learning Rate']}")
print(f"Batch Size: {worst_result['Hyperparameters']['Batch Size']}")
print(f"Test Accuracy: {worst_result['Test Accuracy'] * 100:.2f}%")
#menampilkan hyperparameter terburuk berdasarkan akurasi terendah
```

```
Worst Hyperparameter Configuration:
Hidden Layers: 3
Hidden Neurons: 8
Activation Function: ReLU
Epochs: 10
Learning Rate: 0.1
Batch Size: 64
Test Accuracy: 18.18%
```

Menampilkan parameter teburuk