LAPORAN RESMI

Klasifikasi Bunga Iris dengan Multilayer Perceptron (MLP)

DOSEN PENGAMPU: Al-Ustazd Dr. Oddy Virgantara Putra, S.Kom., M.T.

Firlana Ummi Azzakiy / 442023618081

Andini Putri Rosyidi / 442023618079

Mutiara Afny Imro'atus Sholihah / 442023618080

Shafiyyah Al-Khansa / 442023618075

Rana Rohadatul Aisy / 442023618076

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS DARUSSALAM GONTOR KAMPUS PUTRI

MANTINGAN, NGAWI – INDONESIA

2025

Laporan ini menjelaskan setiap sel (bubble) dari notebook proyek klasifikasi bunga iris menggunakan model MLP (Multilayer Perceptron) dengan PyTorch, serta model pembanding Logistic Regression. Tujuan proyek ini adalah untuk mempraktikkan pembuatan dan pelatihan model klasifikasi sederhana berbasis machine learning sekaligus membandingkan performa metode neural network dengan metode konvensional.

1. Import Library

Potongan Kode:

```
import toch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_splitimport pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

Penjelasan:

Pada bagian ini dilakukan import berbagai library penting yang dibutuhkan untuk mengolah data dan membangun model machine learning. Library yang digunakan di antaranya adalah:

- pandas dan numpy untuk manipulasi data
- matplotlib.pyplot untuk visualisasi grafik
- torch dan torch.nn dari PyTorch untuk membangun arsitektur MLP
- sklearn untuk preprocessing, split data, dan evaluasi performa model

Langkah ini merupakan tahap awal yang penting untuk memastikan semua dependensi tersedia sebelum melanjutkan ke pengolahan dan pelatihan data.

2. Load Dataset’

Potongan Kode:

```
# Load dataset
df = pd.read_csv("/kaggle/input/iris-flower-dataset/IRIS.csv")
df.head()
```

Penjelasan:

Pada tahap ini, dataset bunga iris dibaca menggunakan pandas dari file CSV. Dataset ini memuat panjang dan lebar kelopak (petal) dan sepal dari tiga jenis bunga iris: Setosa, Versicolor, dan Virginica. Fungsi df.head() digunakan untuk menampilkan lima baris pertama dari data, guna melihat bentuk data sebelum diproses lebih lanjut.

Dataset ini nantinya akan digunakan sebagai dasar dalam proses pelatihan model klasifikasi.

3. Preprocessing Data

Potongan Kode:

```
# Encode label target
le = LabelEncoder()
df['species'] = le.fit_transform(df['species'])

# Fitur dan target
X = df.drop('species', axis=1)
y = df['species']

# Normalisasi
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data train-test
X_train_np, X_test_np, y_train_np, y_test_np = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Konversi ke tensor
X_train = torch.tensor(X_train_np, dtype=torch.float32)
X_test = torch.tensor(X_test_np, dtype=torch.float32)
y_train = torch.tensor(y_train_np.values, dtype=torch.long)
y_test = torch.tensor(y_test_np.values, dtype=torch.long)
```

Penjelasan:

Di tahap ini, dilakukan proses pra-pemrosesan (preprocessing) terhadap data agar siap digunakan untuk pelatihan model. Beberapa langkah penting yang dilakukan:

1. Label encoding
   Kolom target species yang semula berupa teks (Setosa, Versicolor, Virginica) dikonversi ke dalam bentuk angka 0, 1, 2 menggunakan LabelEncoder.
2. Pemilahan fitur dan target
   Data fitur (X) dipisahkan dari label target (y). Fitur adalah kolom panjang dan lebar kelopak dan sepal.
3. Normalisasi fitur

Fitur X dinormalisasi menggunakan StandardScaler, supaya nilai tiap kolom memiliki distribusi yang seragam dan tidak mendominasi satu sama lain.

4. Pembagian data
   Dataset dibagi menjadi data latih dan data uji menggunakan train_test_split. Ini penting untuk mengukur performa model terhadap data yang belum pernah dilihat.

4. Judul: MLP Model

Potongan Markdown:

```
**MLP MODEL**
```

Penjelasan:

Ini adalah sel markdown yang digunakan sebagai pemisah bagian dalam notebook. Penanda ini menunjukkan bahwa bagian selanjutnya akan membahas tentang arsitektur model MLP (Multilayer Perceptron) yang akan digunakan dalam proses klasifikasi.

Markdown seperti ini penting untuk membuat struktur notebook jadi lebih mudah dipahami dan terorganisir, khususnya ketika membagi antara bagian pra-pemrosesan, modeling, dan pelatihan.

5. Membangun Model MLP

Potongan Kode:

```
class IrisMLP(nn.Module):
    def __init__(self, input_size=4, hidden_size=10, output_size=3):
        super(IrisMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size)
        )

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

Penjelasan:

Pada bagian ini dibuat kelas IrisMLP, yaitu arsitektur model neural network menggunakan **PyTorch**. Model ini dirancang untuk klasifikasi bunga iris yang memiliki **4 fitur input** dan **3 label kelas**.

Struktur Model:

- Input layer dengan 4 neuron (karena ada 4 fitur)
- Hidden layer dengan 10 neuron dan fungsi aktivasi ReLU
- Output layer dengan 3 neuron (karena ada 3 kelas target). Model dibungkus dalam `nn.Sequential` agar lebih ringkas dan mudah dibaca.

6. Judul: Train Model

Potongan Markdown:

```
**TRAIN MODEL**
```

Penjelasan:

Ini adalah sel markdown yang menandai bahwa bagian berikutnya akan berisi kode untuk melatih model MLP.

Markdown seperti ini berfungsi sebagai pembatas antar bagian utama dalam notebook agar pembaca bisa lebih mudah memahami alur proyek: setelah model dibuat, langkah berikutnya adalah melatihnya menggunakan data training.

Markdown ini sederhana tapi penting untuk dokumentasi dan keterbacaan proyek.

7. Fungsi Pelatihan

Potongan Kode:

```
def train_model(X_train, y_train, X_test, y_test, epochs=100, lr=0.01):
    model = IrisMLP()
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    losses = []
    for epoch in range(epochs):
        model.train()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    return model, losses
```

Penjelasan:

Bagian ini membuat fungsi train_model() yang digunakan untuk melatih model MLP. Fungsi ini melakukan proses sebagai berikut:

- **Membuat objek model** dari kelas IrisMLP
- Menentukan **loss function** (CrossEntropyLoss) karena ini adalah klasifikasi multikelas
- Menentukan **optimizer** (Adam) dengan learning rate tertentu
- Melakukan training dalam loop sejumlah epoch:
  - Forward pass (prediksi output)
  - Hitung loss

- Backpropagation
- Update parameter model

Fungsi ini juga menyimpan nilai loss tiap epoch, yang nanti akan divisualisasikan untuk mengetahui progres pelatihan.

8. Menjalankan Proses Pelatihan

Potongan Kode:

```
# Training
model, losses = train_model(X_train, y_train, X_test, y_test)
```

Penjelasan:

Setelah fungsi train_model() dibuat, pada bagian ini fungsinya dipanggil untuk menjalankan proses training model. Hasil dari fungsi ini adalah:

- **model** yang telah terlatih
- **losses** yaitu list nilai loss dari setiap epoch, yang bisa divisualisasikan

9. Visualisasi Loss

Potongan Kode:

```
# Plot loss training
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss")
plt.show()
```

Penjelasan:

Setelah training, nilai loss dari tiap epoch divisualisasikan dalam bentuk grafik menggunakan matplotlib. Tujuannya untuk melihat apakah model berhasil belajar (loss menurun) atau justru stagnan. Ini penting untuk evaluasi proses pelatihan.

10. Model Pembanding (Logistic Regression)

Potongan Kode:

```
# Logistic Regression sebagai pembanding
lr_model = LogisticRegression(max_iter=200)
lr_model.fit(X_train_np, y_train_np)
lr_preds = lr_model.predict(X_test_np)
lr_acc = accuracy_score(y_test_np, lr_preds)
print("Akurasi Logistic Regression:", lr_acc)
```

Penjelasan:

Bagian ini melatih model Logistic Regression menggunakan data yang sama sebagai pembanding terhadap performa MLP.
Model ini termasuk metode konvensional yang sederhana. Hasil akurasinya dibandingkan untuk melihat apakah penggunaan neural network (MLP) memberikan keunggulan.

11. Inference (Prediksi Ulang)

Potongan Kode:

```
infer_df = pd.DataFrame(X_test_np, columns=X.columns)
infer_df['species'] = le.inverse_transform(y_test_np)

# Konversi fitur inferensi ke tensor (pakai scaler yang sama)
x_infer = torch.tensor(X_test_np, dtype=torch.float32)

# Load model untuk inference
model_inf = IrisMLP()
model_inf.load_state_dict(model.state_dict())

# Prediksi
preds = model_inf(x_infer)
pred_labels = torch.argmax(preds, dim=1)
print("Prediksi:", le.inverse_transform(pred_labels.numpy()))
```

Penjelasan:

Bagian ini mempersiapkan proses inference atau prediksi menggunakan model MLP yang sudah dilatih dan disimpan. Model diload kembali dari file dan digunakan untuk memprediksi data uji.

Langkah-langkahnya:

- Data uji dikonversi ke tensor
- Model diload menggunakan load_state_dict()
- Output prediksi dibandingkan dengan label asli

12. Menyimpan Model

Potongan Kode:

```
torch.save(model.state_dict(), "model.pth")
```

Penjelasan:

Model MLP yang sudah dilatih disimpan ke dalam file model.pth. Hal ini penting agar model bisa digunakan ulang nanti tanpa perlu retraining. File ini bisa dipanggil untuk deployment atau aplikasi lainnya.