Nama: Andi Cleopatra Maryam Jamila

Nim 1103213071

Analisis Markov model dan Hidden Markov Model Week 14

```
# Install PyTorch (skip if already installed)
!pip install torch torchvision torchaudio --quiet
```

```
# Import libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
# Load dataset
def load_data():
    url = "/content/sample_data/bank-full.csv"
    df = pd.read_csv(url, sep=';')

    # Preprocess data
    X = df.drop(columns=['y'])
    y = df['y']

    # Convert categorical to numerical
    X = pd.get_dummies(X)
    le = LabelEncoder()
    y = le.fit_transform(y)

    # Standardize features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    return train_test_split(X, y, test_size=0.2, random_state=42)
```

Analisis:

Dataset yang digunakan berasal dari data bank marketing, yang berisi informasi pelanggan dengan target klasifikasi apakah pelanggan akan berlangganan produk bank (variabel y). Data preprocessing melibatkan penghapusan kolom target dari fitur, konversi fitur kategorikal menjadi numerik menggunakan one-hot encoding, dan normalisasi menggunakan StandardScaler untuk memastikan semua fitur memiliki skala yang sebanding. Label target dikodekan menjadi nilai numerik dengan LabelEncoder. Data kemudian dibagi menjadi set pelatihan (80%) dan pengujian (20%) secara acak untuk mempersiapkan analisis lebih lanjut. Proses ini memastikan bahwa data siap digunakan untuk model pembelajaran mesin dengan input yang terstandardisasi dan dapat diinterpretasikan oleh model.

```python
# Create custom dataset class
class BankDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32).unsqueeze(1)  # Add sequence length dimension
        self.y = torch.tensor(y, dtype=torch.long)

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

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

```python
# Define RNN model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1, pooling='max'):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.pooling = pooling

        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        if self.pooling == 'max':
            out, _ = torch.max(out, 1)
        elif self.pooling == 'avg':
            out = torch.mean(out, 1)
        out = self.fc(out)
        return out
```

```python
# Define Hidden Markov Model (HMM) Class
class HMMModel(nn.Module):
    def __init__(self, n_states, n_features):
        super(HMMModel, self).__init__()
        self.n_states = n_states
        self.n_features = n_features

        self.transition = nn.Parameter(torch.randn(n_states, n_states))  # Transition probabilities
        self.emission = nn.Parameter(torch.randn(n_states, n_features))  # Emission probabilities
        self.initial = nn.Parameter(torch.randn(n_states))  # Initial state probabilities

    def forward(self, x):
        batch_size, seq_len, _ = x.size()
        log_alpha = self.initial.unsqueeze(0).expand(batch_size, -1)

        for t in range(seq_len):
            obs = x[:, t, :]
            log_alpha = torch.logsumexp(
                log_alpha.unsqueeze(2) + self.transition + obs.unsqueeze(1) @ self.emission.T, dim=1
            )

        return log_alpha
```

```python
# Train function
def train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs=25):
    train_loader, val_loader = dataloaders
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)

        scheduler.step()

        val_loss = 0.0
        val_preds = []
        val_labels = []

        model.eval()
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * inputs.size(0)

            preds = torch.argmax(outputs, dim=1)
            val_preds.extend(preds.cpu().numpy())
            val_labels.extend(labels.cpu().numpy())

    train_loss = running_loss / len(train_loader.dataset)
    val_loss = val_loss / len(val_loader.dataset)
    val_acc = accuracy_score(val_labels, val_preds)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

return model
```

Analisis:

Mencakup dua model pembelajaran: Recurrent Neural Network (RNN) dan Hidden Markov Model (HMM), dengan implementasi untuk menangani data urutan. BankDataset menambahkan dimensi sekuensial pada data untuk kompatibilitas dengan arsitektur RNN, yang dirancang untuk memproses data sekuensial dengan hidden states, menggunakan metode pooling (max atau avg) untuk menggabungkan output temporal sebelum klasifikasi. Di sisi lain, HMM menggunakan probabilitas transisi dan emisi untuk memodelkan distribusi probabilistik dari urutan observasi, menjadikannya lebih cocok untuk data temporal berbasis probabilitas. Fungsi pelatihan mencakup optimisasi dengan backpropagation, validasi model di setiap epoch, dan penggunaan scheduler untuk menyesuaikan tingkat pembelajaran. Analisis mencakup evaluasi akurasi, kerugian pelatihan, dan validasi, memungkinkan identifikasi kinerja dan generalisasi model pada data yang belum terlihat.

```python
# Main Experiment
if __name__ == "__main__":
    X_train, X_test, y_train, y_test = load_data()
    train_dataset = BankDataset(X_train, y_train)
    val_dataset = BankDataset(X_test, y_test)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

    dataloaders = (train_loader, val_loader)

    input_size = X_train.shape[1]
    output_size = len(np.unique(y_train))

    hidden_sizes = [16, 32, 64]
    poolings = ['max', 'avg']
    optimizers = {'SGD': optim.SGD, 'RMSProp': optim.RMSprop, 'Adam': optim.Adam}
    epochs_list = [5, 50, 100, 250, 350]

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    results = []

    for hidden_size in hidden_sizes:
        for pooling in poolings:
            for optimizer_name, optimizer_fn in optimizers.items():
                for num_epochs in epochs_list:
                    print(f"\nHidden Size: {hidden_size}, Pooling: {pooling}, Optimizer: {optimizer_name}, Epochs: {num_epochs}")

                    model = RNNModel(input_size, hidden_size, output_size, pooling=pooling)
                    criterion = nn.CrossEntropyLoss()
    optimizer = optimizer_fn(model.parameters(), lr=0.01)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.5)

    trained_model = train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs)

    # Evaluate on test set
    test_preds = []
    test_labels = []

    trained_model.eval()
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = trained_model(inputs)
            preds = torch.argmax(outputs, dim=1)
            test_preds.extend(preds.cpu().numpy())
            test_labels.extend(labels.cpu().numpy())

    test_acc = accuracy_score(test_labels, test_preds)
    print(f"Test Accuracy: {test_acc:.4f}")

    results.append((hidden_size, pooling, optimizer_name, num_epochs, test_acc))
```

Output:

```
Epoch 44/50, Train Loss: 0.1686, Val Loss: 0.2421, Val Acc: 0.9006
Epoch 45/50, Train Loss: 0.1673, Val Loss: 0.2349, Val Acc: 0.9029
Epoch 46/50, Train Loss: 0.1676, Val Loss: 0.2372, Val Acc: 0.9040
Epoch 47/50, Train Loss: 0.1654, Val Loss: 0.2399, Val Acc: 0.9019
Epoch 48/50, Train Loss: 0.1670, Val Loss: 0.2356, Val Acc: 0.9030
Epoch 49/50, Train Loss: 0.1662, Val Loss: 0.2395, Val Acc: 0.8995
Epoch 50/50, Train Loss: 0.1656, Val Loss: 0.2419, Val Acc: 0.9008
Test Accuracy: 0.9008

Hidden Size: 32, Pooling: max, Optimizer: Adam, Epochs: 100
Epoch 1/100, Train Loss: 0.2380, Val Loss: 0.2381, Val Acc: 0.8970
Epoch 2/100, Train Loss: 0.2167, Val Loss: 0.2205, Val Acc: 0.9034
Epoch 3/100, Train Loss: 0.2099, Val Loss: 0.2225, Val Acc: 0.8987
Epoch 4/100, Train Loss: 0.2069, Val Loss: 0.2223, Val Acc: 0.9001
Epoch 5/100, Train Loss: 0.2036, Val Loss: 0.2139, Val Acc: 0.9047
Epoch 6/100, Train Loss: 0.2010, Val Loss: 0.2164, Val Acc: 0.9028
Epoch 7/100, Train Loss: 0.1988, Val Loss: 0.2261, Val Acc: 0.8988
Epoch 8/100, Train Loss: 0.1964, Val Loss: 0.2176, Val Acc: 0.9028
Epoch 9/100, Train Loss: 0.1944, Val Loss: 0.2203, Val Acc: 0.9050
Epoch 10/100, Train Loss: 0.1944, Val Loss: 0.2183, Val Acc: 0.9036
Epoch 11/100, Train Loss: 0.1919, Val Loss: 0.2173, Val Acc: 0.9049
Epoch 12/100, Train Loss: 0.1902, Val Loss: 0.2250, Val Acc: 0.8990
Epoch 13/100, Train Loss: 0.1883, Val Loss: 0.2243, Val Acc: 0.9008
Epoch 14/100, Train Loss: 0.1869, Val Loss: 0.2196, Val Acc: 0.9042
Epoch 15/100, Train Loss: 0.1849, Val Loss: 0.2245, Val Acc: 0.9039
Epoch 16/100, Train Loss: 0.1847, Val Loss: 0.2268, Val Acc: 0.9019
Epoch 17/100, Train Loss: 0.1828, Val Loss: 0.2252, Val Acc: 0.9005
Epoch 18/100, Train Loss: 0.1828, Val Loss: 0.2295, Val Acc: 0.8974
Epoch 19/100, Train Loss: 0.1811, Val Loss: 0.2295, Val Acc: 0.9006
Epoch 20/100, Train Loss: 0.1819, Val Loss: 0.2295, Val Acc: 0.9005
```

Analisis:

Mengevaluasi kinerja model RNN pada dataset marketing bank dengan variasi parameter utama: ukuran hidden layer (16, 32, 64), jenis pooling (max atau average), optimizers (SGD, RMSProp, Adam), dan jumlah epoch (5, 50, 100, 250, 350). Setiap kombinasi parameter diuji untuk menentukan konfigurasi optimal dengan memanfaatkan CrossEntropyLoss sebagai fungsi loss dan StepLR untuk penyesuaian learning rate selama pelatihan. Proses evaluasi melibatkan perhitungan akurasi pada data uji setelah pelatihan untuk setiap konfigurasi. Eksperimen ini bertujuan untuk memahami pengaruh setiap parameter terhadap akurasi dan generalisasi model. Hasil akhirnya berupa daftar kinerja akurasi (test accuracy) yang dapat digunakan untuk menganalisis pola dan memilih konfigurasi terbaik untuk data sekuensial seperti dataset bank ini.

```python
# HMM Experiment
    print("\nRunning Hidden Markov Model (HMM) Experiment")
    hmm_model = HMMModel(n_states=4, n_features=input_size)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(hmm_model.parameters(), lr=0.01)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.5)

    trained_hmm_model = train_model(hmm_model, dataloaders, criterion, optimizer, scheduler, num_epochs=100)
```

Output:

```
Running Hidden Markov Model (HMM) Experiment
Epoch 1/100, Train Loss: 1.1313, Val Loss: 0.2644, Val Acc: 0.8985
Epoch 2/100, Train Loss: 0.2523, Val Loss: 0.2597, Val Acc: 0.8959
Epoch 3/100, Train Loss: 0.2491, Val Loss: 0.2547, Val Acc: 0.8994
Epoch 4/100, Train Loss: 0.2494, Val Loss: 0.2511, Val Acc: 0.9008
Epoch 5/100, Train Loss: 0.2485, Val Loss: 0.2542, Val Acc: 0.8969
Epoch 6/100, Train Loss: 0.2484, Val Loss: 0.2564, Val Acc: 0.8952
Epoch 7/100, Train Loss: 0.2470, Val Loss: 0.2653, Val Acc: 0.8959
Epoch 8/100, Train Loss: 0.2487, Val Loss: 0.2547, Val Acc: 0.8974
Epoch 9/100, Train Loss: 0.2497, Val Loss: 0.2522, Val Acc: 0.8977
Epoch 10/100, Train Loss: 0.2481, Val Loss: 0.2529, Val Acc: 0.8994
Epoch 11/100, Train Loss: 0.2479, Val Loss: 0.2518, Val Acc: 0.8994
Epoch 12/100, Train Loss: 0.2484, Val Loss: 0.2587, Val Acc: 0.8972
Epoch 13/100, Train Loss: 0.2480, Val Loss: 0.2552, Val Acc: 0.8965
Epoch 14/100, Train Loss: 0.2482, Val Loss: 0.2532, Val Acc: 0.8966
Epoch 15/100, Train Loss: 0.2482, Val Loss: 0.2507, Val Acc: 0.8975
Epoch 16/100, Train Loss: 0.2483, Val Loss: 0.2557, Val Acc: 0.8958
Epoch 17/100, Train Loss: 0.2468, Val Loss: 0.2592, Val Acc: 0.8988
Epoch 18/100, Train Loss: 0.2483, Val Loss: 0.2540, Val Acc: 0.8983
Epoch 19/100, Train Loss: 0.2479, Val Loss: 0.2530, Val Acc: 0.8978
Epoch 20/100, Train Loss: 0.2468, Val Loss: 0.2522, Val Acc: 0.8996
Epoch 21/100, Train Loss: 0.2494, Val Loss: 0.2521, Val Acc: 0.8997
Epoch 22/100, Train Loss: 0.2469, Val Loss: 0.2501, Val Acc: 0.8970
Epoch 23/100, Train Loss: 0.2482, Val Loss: 0.2525, Val Acc: 0.8976
Epoch 24/100, Train Loss: 0.2485, Val Loss: 0.2522, Val Acc: 0.8979
Epoch 25/100, Train Loss: 0.2480, Val Loss: 0.2491, Val Acc: 0.8983
Epoch 26/100, Train Loss: 0.2494, Val Loss: 0.2505, Val Acc: 0.8982
Epoch 27/100, Train Loss: 0.2472, Val Loss: 0.2512, Val Acc: 0.8976
Epoch 28/100, Train Loss: 0.2482, Val Loss: 0.2540, Val Acc: 0.8991
Epoch 29/100, Train Loss: 0.2492, Val Loss: 0.2463, Val Acc: 0.8997
Epoch 30/100, Train Loss: 0.2484, Val Loss: 0.2482, Val Acc: 0.9007
Epoch 31/100, Train Loss: 0.2480, Val Loss: 0.2522, Val Acc: 0.8979
Epoch 32/100, Train Loss: 0.2487, Val Loss: 0.2522, Val Acc: 0.8985
```

Analisis:

menjalankan eksperimen menggunakan model Hidden Markov Model (HMM) yang diimplementasikan dengan PyTorch. Model HMM dirancang dengan 4 state tersembunyi (n_states) dan dimensi fitur input setara dengan jumlah fitur dalam data (input_size). Fungsi loss yang digunakan adalah CrossEntropyLoss, sedangkan optimisasi dilakukan dengan Adam Optimizer untuk memastikan konvergensi yang efisien. StepLR Scheduler mengatur penurunan learning rate setiap 50 epoch untuk meningkatkan stabilitas pelatihan. Model dilatih selama 100 epoch menggunakan fungsi pelatihan yang sama dengan RNN. Eksperimen ini bertujuan untuk mengevaluasi kemampuan model HMM dalam menangkap pola urutan pada dataset bank marketing, dengan hasil yang dapat dibandingkan dengan model RNN untuk memahami perbedaan performa kedua pendekatan.