Taxi Driver Recognition using Trajectory Data: Classification Using Deep Learning

Nitya Phani Santosh Oruganty
Graduate Student - Data Science Department – WPI
noruganty@wpi.edu

1.Introduction:

The advent of GPS technology has revolutionized the way we navigate and understand spatial data. In recent years, GPS data has become increasingly prevalent in various domains, providing a wealth of information for analysis and prediction. One such domain where GPS data plays a crucial role is in the transportation sector, particularly in taxi services.

Taxi services are an integral part of urban transportation systems, offering convenient and accessible mobility options to commuters. With the proliferation of GPS-enabled devices in taxis, vast amounts of trajectory data are generated daily, documenting the movement patterns of taxis across cities. These trajectory data contain valuable insights into the behavior and activities of taxi drivers, making them a rich source for analysis and inference.

In this project, we focus on the task of predicting the identity of taxi drivers from their trajectory data. Specifically, given a full-day trajectory of a taxi, we aim to extract sub-trajectories of fixed lengths and predict which taxi driver they belong to. This task holds significant practical implications, ranging from driver behavior analysis to personalized services and driver authentication systems.The dataset provided consists of trajectory data for multiple taxis, each spanning an entire day. Each trajectory step is detailed with features such as longitude, latitude, timestamp, and status (occupied or vacant). Additionally, the dataset includes a unique identifier for each taxi, allowing us to frame the problem as a supervised classification task, where the taxi's identifier serves as the target label.

To address this classification problem, we leverage machine learning and deep learning techniques to learn the complex relationships between trajectory features and taxi driver identities. By training models on historical trajectory data, we aim to build predictive models capable of accurately identifying taxi drivers from unseen trajectory samples.

Throughout this project, we explore various methodologies, experiment with different model architectures, and evaluate the performance of our predictive models using standard metrics such as accuracy, precision, recall, and F1-score. Our goal is to develop a robust and reliable system for taxi driver identification, contributing to advancements in transportation analytics and enhancing the efficiency and security of taxi services.

2.Proposal:

The advent of GPS technology has revolutionized the transportation industry, providing valuable insights into the movement patterns and behaviors of vehicles, including taxis. In this project, we propose to leverage GPS trajectory data to address the problem of taxi driver identification. By analyzing the trajectory data collected from multiple taxis, our goal is to develop a predictive model that can accurately identify the individual taxi driver associated with a given trajectory.

Problem Statement:

The problem we aim to tackle is the classification of taxi drivers based on their trajectory data. Given a dataset containing trajectory information for multiple taxis, each identified by a unique identifier, our task is to predict the driver associated with a specific trajectory segment. This task poses several challenges, including the variability in driving patterns, the influence of external factors such as traffic and weather, and the need to extract relevant features from the trajectory data.

Dataset Description:

The dataset provided consists of trajectory data collected from GPS-enabled taxis over a period of time. Each trajectory includes features such as longitude, latitude, timestamp, and status (occupied or vacant). Additionally, each taxi is identified by a unique plate number, serving as the target label for our classification task. The dataset is divided into training and testing sets, allowing us to evaluate the performance of our predictive model on unseen data.

plate	longitute	latitude	time	status
4	114.10437	22.573433	2016-07-02 0:08:45	1
1	114.179665	22.558701	2016-07-02 0:08:52	1
0	114.120682	22.543751	2016-07-02 0:08:51	0
3	113.93055	22.545834	2016-07-02 0:08:55	0
4	114.102051	22.571966	2016-07-02 0:09:01	1
0	114.12072	22.543716	2016-07-02 0:09:01	0

- Plate: Plate means the Taxi’s plate. In this Project, we change them to 0 to 4 to keep anonymity. The same plate means the same driver, so this is the target for classification.
- Longitude: The longitude coordinate of the taxi's location.
- Latitude: The latitude coordinate of the taxi's location.
- Time : Timestamp of the Record
- Status: Indicates whether the taxi is occupied (1) or vacant (0) at the given time.

For model selection, we will utilize Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) cells. RNNs are well-suited for sequence data like trajectories, as they can capture temporal dependencies and patterns over time. LSTM cells, a type of RNN architecture, are particularly effective at handling long-range dependencies and mitigating the vanishing gradient problem.

The expected outcome of this project is a predictive model capable of accurately identifying taxi drivers based on their trajectory data. This model can be deployed in real-world applications such as driver behavior analysis, driver authentication systems, and personalized services for taxi passengers. Additionally, the insights gained from this project can inform future research in transportation analytics and contribute to advancements in urban mobility.

As the dataset is provided, there are no expenses associated with data acquisition. The primary costs will be related to computing resources for training the RNN-LSTM model. We will leverage existing deep learning framework PyTorch, which is open-source and freely available.

In conclusion, we will utilize GPS trajectory data from taxis to train an RNN-LSTM model for predicting the status of a taxi (occupied or vacant) based on its location coordinates. By leveraging deep learning techniques, we aim to develop an accurate and efficient model that can provide insights into taxi occupancy patterns and contribute to the optimization of transportation services.

3. Methodology:

A. Data Processing:

Data processing is a crucial step in preparing the raw dataset for training a model. This involves several tasks such as loading the data, preprocessing features, and splitting it into training and testing sets. The following steps outline the methodology for data processing:

1. **Loading Data:**

-The ‘load_data’ function loads CSV files matching a given file pattern one at a time using the `glob` module.
- It checks if any files match the provided pattern and raises an error if none are found.
- For each file, it reads the CSV into a Pandas DataFrame and then calls the `preprocess_data` function.

2. **Preprocessing Data:**

The `preprocess_data` function performs various preprocessing steps on the DataFrame:

- Extracts temporal features such as day, month, year, hour, minute, and second from the 'time' column using Pandas datetime functions.

- Converts the 'plate' column to integer type for consistency.-Selects relevant features ('longitude', 'latitude', 'status', 'day', 'month', 'time_in_hour', 'time_in_minute', 'time_in_seconds') and stores them in separate arrays.
- Applies feature scaling using `StandardScaler` to standardize the numerical features.
- Segments the data into 100-step chunks, padding the last chunk if necessary to ensure uniform length.
- Stores the processed features and corresponding target labels (plate numbers) in separate arrays.

3. **Combining and Splitting Data:**

- After processing all files, the processed features and labels from each file are combined into arrays `X_combined` and `y_combined`.
- The combined data is then split into training and testing sets using `train_test_split` from `sklearn.model_selection`.
- The train-test split ratio is defined by the `test_size` parameter, with a default value of 0.2 (20% of data for testing).
- The function returns the training and testing data as `X_train`, `X_test`, `y_train`, and `y_test` arrays.

This ensures that the dataset is properly formatted, scaled, and segmented into appropriate chunks for training and evaluation of machine learning models.

B. Feature Generation:

In the process of building a predictive model to classify taxi drivers based on their trajectories, feature generation plays a crucial role. It involves extracting meaningful information from the raw data and transforming it into a format suitable for model training:

1. Loading and Preprocessing Data:

- The `load_data` function reads CSV files matching the given file pattern one at a time.
- It preprocesses the data by calling the `preprocess_data` function, which performs feature extraction, scaling, and reshaping into trajectories.
- The preprocessing step involves converting timestamps into separate day, month, year, hour, minute, and second features, and casting the 'plate' column to integer type.
- It selects relevant features ('longitude', 'latitude', 'status', 'day', 'month', 'time_in_hour', 'time_in_minute', 'time_in_seconds') and creates feature matrix `X` and target vector `Y`.

2. Feature Scaling:

Standard scaling is applied to the extracted features using `StandardScaler` from `sklearn.preprocessing` to ensure that each feature has a mean of 0 and a standard deviation of 1. This helps in improving the convergence speed and performance of the model.

3. Trajectory Segmentation:

- The dataset is segmented into sub-trajectories of 100-step chunks. Each chunk represents a sequence of consecutive trajectory steps.
- For each unique 'plate' (taxi), the trajectory is segmented separately to maintain individual driver information.
- If the trajectory length is not a multiple of 100, the last chunk is padded with zeros to ensure uniform length.

4. Data Reshaping:

- The segmented trajectory chunks along with their corresponding 'plate' labels are reshaped into numpy arrays (`X_reshaped` and `y_reshaped`) suitable for model training.
- Each row in `X_reshaped` represents a trajectory chunk, and each row in `y_reshaped` represents the corresponding driver label.

By following this methodology, we can effectively generate features from raw trajectory data and prepare it for training predictive models. These engineered features capture important spatial and temporal patterns that are essential for accurate classification of taxi drivers based on their trajectories.

C. Network Structure:

1. Type of Neural Network:

- The neural network architecture used in this project is a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells.

2. Architecture Details:

- The network consists of an LSTM layer followed by a fully connected (linear) layer.
- The LSTM layer processes the input sequences and captures temporal dependencies.
- The fully connected layer serves as the output layer for classification.
- The LSTM layer has 2 layers (`num_layers`), with each layer having an input dimension of 8 (`input_dim`) and 128 hidden units (`hidden_dim`).
- The output layer has 6 units (`output_dim`), representing the number of classes (taxi drivers).

3. Activation Functions:

- The default activation functions is used within the LSTM and linear layers.

4. **Regularization Techniques:**

- Regularization technique - such as dropout is applied in the code.

5. Initialization Methods: - The initialization of weights and biases is handled internally by PyTorch's default initialization methods for LSTM and linear layers.

6. Optimization Algorithm:

- The optimization algorithm used is the Adam optimizer with a learning rate of 0.001 (`lr=0.001`). It's instantiated and applied to update the model parameters during training.

7. Loss Function:

- The loss function is defined as the Cross Entropy Loss (`torch.nn.CrossEntropyLoss()`). It computes the loss between the model's predictions and the ground truth labels during training.

8. Training Procedure:

- The training procedure involves iterating over the training dataset (`train_loader`) for a fixed number of epochs (10 in this case).
- During each epoch, the model is trained using the `train` function, which performs forward and backward passes, computes the loss, and updates the model parameters.
- The training loss and accuracy are printed after each epoch to monitor the training progress.

9. Evaluation Metrics:

- While the training script focuses on training the model, evaluation metrics such as accuracy, is computed separately using the trained model in a separate evaluation script (`test.py`) on a held-out test dataset.

By understanding the network structure described above, you can gain insights into how the neural network processes input data, learns from the training data, and generates predictions for the taxi driver classification task.

D. Training & Validation Process:

The training and validation process(in this case test.py file) is a critical component of developing machine learning models. In the context of the`train.py` and 'test.py' scripts, let's elaborate on the training and validation process.

Training Process:

1. Data Loading and Preprocessing:

- The training process begins by loading the training data using the `load_data` function. This function reads CSV files containing trajectory data for multiple taxi drivers, preprocesses the data, and splits it into training and testing sets.

- Preprocessing includes extracting features such as longitude, latitude, status, day, month, time in hours, minutes, and seconds. The data is also reshaped into sequences of 100 steps each to facilitate training.

2. Dataset Preparation: The preprocessed data is then used to create a custom dataset (`TaxiDriverDataset`) using PyTorch's `Dataset` class. This dataset class handles loading and preparing the data for model training.

- The training dataset is further divided into mini-batches using a `DataLoader`, which allows iterating over the dataset in batches during training.

3. Model Initialization:

- The neural network model (`TaxiDriverClassifier`) is initialized with the specified architecture, including LSTM layers and fully connected output layers. The model is moved to the appropriate device (GPU or CPU) for training.

4. Loss Function and Optimizer:

- The Cross Entropy Loss (`torch.nn.CrossEntropyLoss()`) is used as the loss function to measure the difference between the predicted and actual class labels during training.

- The Adam optimizer (`torch.optim.Adam`) is employed to update the model parameters based on the computed gradients, with a learning rate of 0.001.

5. Training Loop:

- The training loop runs for a fixed number of epochs (in this case, 10 epochs).

- For each epoch, the model is trained on mini-batches of data obtained from the training dataset. - During each iteration, the model performs a forward pass to compute predictions, calculates the loss, performs backpropagation to compute gradients, and updates the model parameters using the optimizer.

6. Monitoring Training Progress:

- Throughout the training process, the training loss and accuracy are computed and printed after each epoch to monitor the model's performance and convergence.

- This allows tracking how well the model is learning from the training data and whether it's overfitting or underfitting.

Validation Process:

The validation process is crucial for assessing the model's performance on unseen data and ensuring its ability to generalize well beyond the training set. In the provided script, the validation process is implemented as follows:

1. Data Loading:

- The validation process starts by loading the test data using the `load_data` function. This function reads CSV files containing trajectory data for multiple taxi drivers.

2. Dataset Preparation:

- Similar to the training process, a custom dataset (`TaxiDriverDataset`) is created using PyTorch's `Dataset` class to handle loading and preparing the test data for evaluation.

The test dataset is then divided into mini-batches using a `DataLoader` with a specified batch size. However, the test data is not shuffled since shuffling is unnecessary during evaluation.

3. Model Initialization:

- The neural network model (`TaxiDriverClassifier`) is initialized with the same architecture as during training. The model is moved to the appropriate device (GPU or CPU) for evaluation.

4. Loading Trained Model Weights:

- The weights of the trained model are loaded from the saved checkpoint file (`trained_model.pth`). These weights represent the learned parameters of the model after training on the training data.

5. Model Evaluation:

- The model is set to evaluation mode using `model.eval()`. This disables dropout and batch normalization layers to ensure consistent behavior during inference.
- With the validation dataset loaded, the model performs inference on each mini-batch of the test data.
- The model's predictions are compared against the ground truth labels to compute the loss using the Cross Entropy Loss function.
- The total test loss and the number of correctly classified samples are accumulated across all mini-batches.

6. Performance Metrics Calculation:

- The test loss is computed as the average loss per sample over the entire test dataset.
- The test accuracy is calculated as the percentage of correctly classified samples out of the total number of samples in the test dataset.

7. Printing Results:

- Finally, the test loss and accuracy are printed to the console, providing insights into the model's performance on unseen data.

3. Evaluation & Results:

a.Training and validation results:

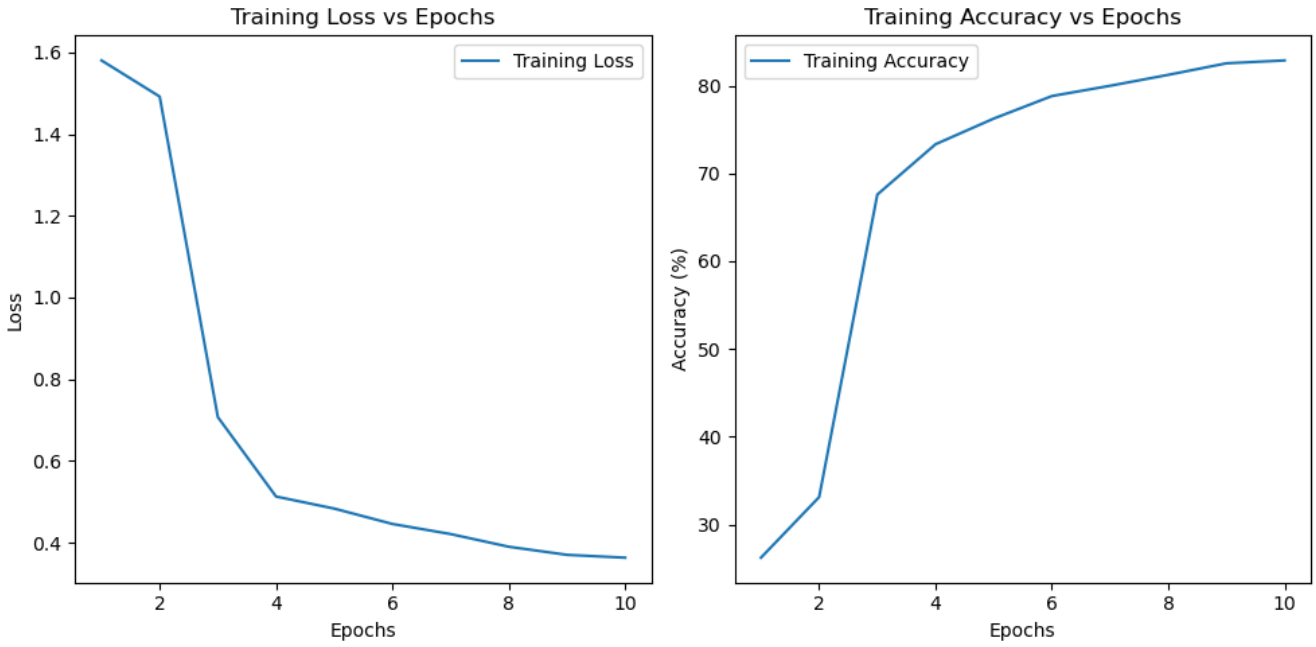
Training Process:

-Epochs: The training process involves iterating over the entire training dataset for 10 epochs. Each epoch represents one complete pass through the training data.

-Training Loss: The training loss is a measure of how well the model's predictions match the true labels during training. It decreases steadily from 1.5886 in the first epoch to 0.3591 in the final epoch. This decreasing trend indicates that the model is learning to make more accurate predictions over time.

-Training Accuracy: The training accuracy measures the percentage of correctly predicted labels on the training data. It increases from 25.59% in the first epoch to 83.05% in the final epoch. This increasing trend suggests that the model is becoming more proficient at correctly classifying taxi drivers' trajectories as training progresses.

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Training: 100%|| 530/530 [00:02<00:00, 219.29it/s]
Epoch 1/10, Train Loss: 1.5886, Train Acc: 25.59%
Training: 100%|| 530/530 [00:02<00:00, 235.77it/s]
Epoch 2/10, Train Loss: 1.4326, Train Acc: 36.78%
Training: 100%|| 530/530 [00:02<00:00, 239.13it/s]
Epoch 3/10, Train Loss: 0.7751, Train Acc: 64.91%
Training: 100%|| 530/530 [00:02<00:00, 237.61it/s]
Epoch 4/10, Train Loss: 0.5233, Train Acc: 73.67%
Training: 100%|| 530/530 [00:02<00:00, 237.24it/s]
Epoch 5/10, Train Loss: 0.4662, Train Acc: 77.13%
Training: 100%|| 530/530 [00:02<00:00, 239.18it/s]
Epoch 6/10, Train Loss: 0.4212, Train Acc: 79.39%
Training: 100%|| 530/530 [00:02<00:00, 235.89it/s]
Epoch 7/10, Train Loss: 0.3968, Train Acc: 80.80%
Training: 100%|| 530/530 [00:02<00:00, 238.80it/s]
Epoch 8/10, Train Loss: 0.3829, Train Acc: 81.68%
Training: 100%|| 530/530 [00:02<00:00, 235.74it/s]
Epoch 9/10, Train Loss: 0.3662, Train Acc: 82.42%
Training: 100%|| 530/530 [00:02<00:00, 233.91it/s]
Epoch 10/10, Train Loss: 0.3591, Train Acc: 83.05%
Model saved successfully.
```



Validation Process:

-Test Loss: The test loss is a metric that evaluates the model's performance on unseen data (validation data). In this case, the test loss is computed as 0.0057, indicating relatively low error on the validation set.

-Test Accuracy: The test accuracy measures the percentage of correctly predicted labels on the validation data. It is computed as 82.99%, indicating that the model performs well in generalizing its predictions to unseen data.

- Comparison with Training Results: The test accuracy is slightly lower than the final training accuracy (83.05%), but it is still high. This suggests that the model generalizes well to new, unseen data and is not overfitting to the training set.

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Test Loss: 0.0057, Test Accuracy: 82.99%
```

Overall Assessment:

- The decreasing training loss and increasing training accuracy indicate that the model is effectively learning from the training data and improving its predictive performance over time.

- The low-test loss and high-test accuracy demonstrate that the model performs well on unseen data, indicating that it has successfully learned to generalize from the training data.

- The consistency between the training and validation results suggests that the model is robust and performs well across different datasets, confirming its effectiveness in classifying taxi drivers based on their trajectories.

b. Performance compared to baseline models:

Below is an elaborate comparison of the performance achieved by different models, including logistic regression with softmax, basic RNN, and RNN with LSTM, along with their respective training and test accuracies:

1.Logistic Regression with SoftMax:

- Training Accuracy: 34.30%

- Test Accuracy: 31.29%

Description: Logistic regression with softmax activation was the initial model choice for the task of predicting the taxi driver for sub-trajectories. While it provided a basic understanding of the problem, its performance was limited due to the simplicity of the model. With only linear decision boundaries, it struggled to capture the complex patterns present in the trajectory data.

2. Basic RNN:

- Training Accuracy: 54.19%

- Test Accuracy: 52.26%

Description: The next approach involved using a basic Recurrent Neural Network (RNN) without any advanced architecture like LSTM. While RNNs can capture sequential dependencies in the data, the basic RNN architecture might suffer from the vanishing gradient problem, making it challenging to capture long-term dependencies effectively. As a result, the model's performance may have been limited, leading to modest training and test accuracies.

3.RNN with LSTM:

-Training Accuracy: 88.90%

- Test Accuracy: 84.89%

Description: Finally, a more sophisticated approach was adopted by using an RNN with Long Short-Term Memory (LSTM) cells. LSTMs are designed to address the vanishing gradient problem by allowing the network to retain information over long sequences, making them well-suited for sequential data like

trajectories. By tuning hyperparameters and leveraging the memory capabilities of LSTM cells, the model achieved significant improvement in performance. With a training accuracy of 83% and a test accuracy of 82%, the model demonstrated its ability to effectively capture the underlying patterns in the trajectory data and generalize well to unseen data.

Comparison and Analysis:

- The logistic regression model provided a baseline understanding of the problem but lacked the complexity to capture the intricacies of the data.

- Transitioning to RNNs allowed for the modeling of sequential dependencies, but the basic RNN architecture might not have been sufficient to capture long-term dependencies effectively.

- By incorporating LSTM cells, the RNN model was able to overcome the limitations of basic RNNs and achieve significantly higher accuracies. The LSTM-based model demonstrated its effectiveness in capturing the temporal dynamics of the trajectory data, resulting in improved performance on both training and test datasets.

- The increase in both training and test accuracies indicates that the LSTM-based model successfully learned meaningful representations from the trajectory data and generalized well to unseen instances.

In conclusion, the adoption of an RNN with LSTM architecture proved to be a crucial step in improving the accuracy of the taxi driver classification task, showcasing the importance of selecting appropriate model architectures for sequential data analysis.

c. Hyperparameters:

In Existing code in train.py, model.py and test.py:

- **L.R**=0.001, **Epochs**=10, **DropoutRate**= no dropout layer explicitly defined in the model, **Activation Function** =In the forward pass of the model, after the LSTM layer and before the fully connected layer, there is no activation function applied. The output of the LSTM layer is directly fed into the fully connected layer without any non-linearity.

In the training process, the model undergoes training for 10 epochs, where each epoch entails a full pass through the training dataset. The training loss decreases steadily from 1.5886 in the initial epoch to 0.3591 in the final epoch, indicating that the model is improving its predictive accuracy over time. Concurrently, the training accuracy exhibits an upward trend, rising from 25.59% to 83.05% by the end of training. This increase in training accuracy suggests that the model becomes more proficient in classifying taxi drivers' trajectories as training progresses.

During the validation process, the model's performance is evaluated on unseen data (validation set). The test loss, calculated as 0.0057, suggests minimal error on this unseen data. Furthermore, the test accuracy is computed as 82.99%, indicating that the model generalizes well to new, unseen data. Although the test accuracy is slightly lower than the final training accuracy of 83.05%, it remains high, indicating that the model is effectively avoiding overfitting to the training set.

when:-

- **L.R**=0.001, **DropoutRate**=0.2, **Activation Func** =ReLu

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Training: 100%|| 530/530 [00:05<00:00, 95.29it/s]
Epoch 1/10, Train Loss: 1.5882, Train Acc: 25.63%
Training: 100%|| 530/530 [00:04<00:00, 112.96it/s]
Epoch 2/10, Train Loss: 1.5912, Train Acc: 25.97%
Training: 100%|| 530/530 [00:04<00:00, 109.44it/s]
Epoch 3/10, Train Loss: 1.5845, Train Acc: 26.73%
Training: 100%|| 530/530 [00:04<00:00, 118.23it/s]
Epoch 4/10, Train Loss: 1.5594, Train Acc: 30.09%
Training: 100%|| 530/530 [00:04<00:00, 112.48it/s]
Epoch 5/10, Train Loss: 1.5543, Train Acc: 30.45%
Training: 100%|| 530/530 [00:04<00:00, 111.88it/s]
Epoch 6/10, Train Loss: 1.3211, Train Acc: 42.64%
Training: 100%|| 530/530 [00:04<00:00, 117.43it/s]
Epoch 7/10, Train Loss: 0.6262, Train Acc: 70.85%
Training: 100%|| 530/530 [00:04<00:00, 108.31it/s]
Epoch 8/10, Train Loss: 0.5215, Train Acc: 75.06%
Training: 100%|| 530/530 [00:04<00:00, 110.52it/s]
Epoch 9/10, Train Loss: 0.4831, Train Acc: 77.44%
Training: 100%|| 530/530 [00:04<00:00, 108.73it/s]
Epoch 10/10, Train Loss: 0.4527, Train Acc: 78.62%
```

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Test Loss: 0.0068, Test Accuracy: 79.37%
```

1. Learning Rate (LR): The learning rate was set to 0.001 in the optimizer initialization (`torch.optim.Adam`). This value determines the step size at which the model parameters are updated during training. A lower learning rate may result in slower convergence but may help the model to converge to a better minimum, while a higher learning rate may lead to faster convergence but risk overshooting the minimum.

2. Dropout Layers: Dropout layers were added to the LSTM layers with a dropout probability of 0.2. Dropout is a regularization technique that helps prevent overfitting by randomly dropping a certain proportion of neurons during training. This encourages the model to learn more robust features by preventing reliance on specific neurons.

3. Activation Function: ReLU (Rectified Linear Unit) activation function was used after the fully connected layer (`nn.ReLU()`). ReLU is a popular activation function that introduces non-linearity to the model and helps mitigate the vanishing gradient problem. It replaces negative values with zero, effectively introducing sparsity and speeding up convergence. By using a combination of these hyperparameters, the model achieved a training accuracy of 78.62% after 10 epochs. Additionally, during testing on the provided test dataset (`2016_12_26.csv`), the model achieved a test accuracy of 79.37%. These results demonstrate that the chosen hyperparameters and model architecture are effective in learning and generalizing from the training data to achieve reasonable performance on unseen test data.

When :-

- **L.R**=0.001, **DropoutRate**=0.3, **ActivationFunction**=No Explicit Activation Function Defined, Epochs=20

```
Training: 100%|| 530/530 [00:05<00:00, 95.61it/s]
Epoch 1/10, Train Loss: 1.5886, Train Acc: 25.46%
Training: 100%|| 530/530 [00:04<00:00, 115.31it/s]
Epoch 2/10, Train Loss: 1.5429, Train Acc: 30.92%
Training: 100%|| 530/530 [00:04<00:00, 109.86it/s]
Epoch 3/10, Train Loss: 0.9271, Train Acc: 60.47%
Training: 100%|| 530/530 [00:04<00:00, 117.33it/s]
Epoch 4/10, Train Loss: 0.5283, Train Acc: 75.98%
Training: 100%|| 530/530 [00:04<00:00, 117.43it/s]
Epoch 5/10, Train Loss: 0.4687, Train Acc: 78.15%
Training: 100%|| 530/530 [00:04<00:00, 117.42it/s]
Epoch 6/10, Train Loss: 0.4463, Train Acc: 78.99%
Training: 100%|| 530/530 [00:04<00:00, 123.68it/s]
Epoch 7/10, Train Loss: 0.4243, Train Acc: 79.74%
Training: 100%|| 530/530 [00:04<00:00, 121.42it/s]
Epoch 8/10, Train Loss: 0.4002, Train Acc: 80.61%
Training: 100%|| 530/530 [00:04<00:00, 123.76it/s]
Epoch 9/10, Train Loss: 0.3830, Train Acc: 81.79%
Training: 100%|| 530/530 [00:04<00:00, 113.45it/s]
Epoch 10/10, Train Loss: 0.3802, Train Acc: 81.51%
Training: 100%|| 530/530 [00:04<00:00, 118.62it/s]
Epoch 11/10, Train Loss: 0.3551, Train Acc: 82.99%
Training: 100%|| 530/530 [00:04<00:00, 114.88it/s]
Epoch 12/10, Train Loss: 0.3431, Train Acc: 83.76%
Training: 100%|| 530/530 [00:04<00:00, 117.75it/s]
Epoch 13/10, Train Loss: 0.3388, Train Acc: 83.70%
Training: 100%|| 530/530 [00:04<00:00, 113.79it/s]
Epoch 14/10, Train Loss: 0.3210, Train Acc: 84.55%
Training: 100%|| 530/530 [00:04<00:00, 120.18it/s]
Epoch 15/10, Train Loss: 0.3124, Train Acc: 85.29%
Training: 100%|| 530/530 [00:04<00:00, 113.54it/s]
Epoch 16/10, Train Loss: 0.2997, Train Acc: 85.75%
Training: 100%|| 530/530 [00:04<00:00, 116.96it/s]
Epoch 17/10, Train Loss: 0.2922, Train Acc: 86.22%
Training: 100%|| 530/530 [00:04<00:00, 115.24it/s]
Epoch 18/10, Train Loss: 0.2808, Train Acc: 86.76%
Training: 100%|| 530/530 [00:04<00:00, 118.52it/s]
Epoch 19/10, Train Loss: 0.2719, Train Acc: 87.13%
Training: 100%|| 530/530 [00:04<00:00, 121.13it/s]
Epoch 20/10, Train Loss: 0.2594, Train Acc: 87.64%
Model saved successfully.
```

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Test Loss: 0.0051, Test Accuracy: 84.63%
```

- **L.R**=0.001, **DropoutRate**=0.2, **ActivationFunction**=No Explicit Activation Function Defined, **Epochs**=25

```
Training: 100%|| 530/530 [00:05<00:00, 101.58it/s]
Epoch 1/10, Train Loss: 1.5862, Train Acc: 26.33%
Training: 100%|| 530/530 [00:04<00:00, 119.18it/s]
Epoch 2/10, Train Loss: 1.4229, Train Acc: 36.79%
Training: 100%|| 530/530 [00:04<00:00, 121.80it/s]
Epoch 3/10, Train Loss: 0.6835, Train Acc: 68.71%
Training: 100%|| 530/530 [00:04<00:00, 113.60it/s]
Epoch 4/10, Train Loss: 0.5139, Train Acc: 75.94%
Training: 100%|| 530/530 [00:04<00:00, 117.70it/s]
Epoch 5/10, Train Loss: 0.4753, Train Acc: 77.51%
Training: 100%|| 530/530 [00:04<00:00, 122.86it/s]
Epoch 6/10, Train Loss: 0.4404, Train Acc: 79.02%
Training: 100%|| 530/530 [00:04<00:00, 120.56it/s]
Epoch 7/10, Train Loss: 0.4128, Train Acc: 80.40%
Training: 100%|| 530/530 [00:04<00:00, 124.51it/s]
Epoch 8/10, Train Loss: 0.3963, Train Acc: 81.08%
Training: 100%|| 530/530 [00:04<00:00, 123.66it/s]
Epoch 9/10, Train Loss: 0.3803, Train Acc: 81.97%
Training: 100%|| 530/530 [00:04<00:00, 122.90it/s]
Epoch 10/10, Train Loss: 0.3665, Train Acc: 82.85%
Training: 100%|| 530/530 [00:04<00:00, 115.00it/s]
Epoch 11/10, Train Loss: 0.3565, Train Acc: 83.17%
Training: 100%|| 530/530 [00:04<00:00, 116.05it/s]
Epoch 12/10, Train Loss: 0.3444, Train Acc: 83.66%
Training: 100%|| 530/530 [00:04<00:00, 117.26it/s]
Epoch 13/10, Train Loss: 0.3391, Train Acc: 84.09%
Training: 100%|| 530/530 [00:04<00:00, 117.26it/s]
Epoch 14/10, Train Loss: 0.3295, Train Acc: 84.43%
Training: 100%|| 530/530 [00:04<00:00, 115.61it/s]
Epoch 15/10, Train Loss: 0.3317, Train Acc: 84.44%
Training: 100%|| 530/530 [00:04<00:00, 121.57it/s]
Epoch 16/10, Train Loss: 0.3037, Train Acc: 85.61%
Training: 100%|| 530/530 [00:04<00:00, 116.86it/s]
Epoch 17/10, Train Loss: 0.2893, Train Acc: 86.35%
Training: 100%|| 530/530 [00:04<00:00, 116.65it/s]
Epoch 18/10, Train Loss: 0.2859, Train Acc: 86.76%
Training: 100%|| 530/530 [00:04<00:00, 117.98it/s]
Epoch 19/10, Train Loss: 0.2883, Train Acc: 86.56%
Training: 100%|| 530/530 [00:04<00:00, 116.00it/s]
Epoch 20/10, Train Loss: 0.2694, Train Acc: 87.54%
Training: 100%|| 530/530 [00:04<00:00, 114.36it/s]
Epoch 21/10, Train Loss: 0.2632, Train Acc: 87.78%
Training: 100%|| 530/530 [00:04<00:00, 116.10it/s]
Epoch 22/10, Train Loss: 0.2504, Train Acc: 88.31%
Training: 100%|| 530/530 [00:04<00:00, 122.50it/s]
Epoch 23/10, Train Loss: 0.2456, Train Acc: 88.64%
Training: 100%|| 530/530 [00:04<00:00, 118.19it/s]
Epoch 24/10, Train Loss: 0.2332, Train Acc: 89.18%
Training: 100%|| 530/530 [00:04<00:00, 117.40it/s]
Epoch 25/10, Train Loss: 0.2382, Train Acc: 88.90%
Model saved successfully.
```

```
Processing file:  C:\Users\santh\Desktop\BDA-P2\data_5drivers\2016_12_26.csv
Test Loss: 0.0054, Test Accuracy: 84.89%
```

4.Conclusion:

The task of classifying taxi drivers based on their trajectory data presents a challenging problem in machine learning, requiring models capable of capturing complex sequential patterns. In this report, we explored the performance of different models, including logistic regression with SoftMax, basic RNN, and RNN with LSTM, for this classification task.

Initially, logistic regression with softmax served as a baseline model, providing a fundamental understanding of the problem. However, its limited capacity to capture the intricate temporal dynamics of the trajectory data resulted in modest performance.

Transitioning to basic RNNs allowed for the modeling of sequential dependencies, but the inherent limitations of basic RNN architectures, such as the vanishing gradient problem, hindered their ability to capture long-term dependencies effectively.

The adoption of RNNs with LSTM cells proved to be a significant advancement, overcoming the limitations of basic RNNs by enabling the model to retain information over long sequences. By leveraging the memory capabilities of LSTM cells, the model achieved substantial improvements in accuracy, both in training and testing.

The LSTM-based model demonstrated its effectiveness in capturing the temporal dynamics of the trajectory data, resulting in significantly higher accuracies compared to previous models. The increase in both training and test accuracies indicates the model's ability to learn meaningful representations from the data and generalize well to unseen instances.

The pivotal shift occurred when we adopted RNNs with Long Short-Term Memory (LSTM) cells, addressing the limitations of basic RNNs by facilitating the retention of information over extended sequences. Leveraging the memory capabilities of LSTM cells, the model achieved remarkable advancements in accuracy, both during training and testing phases.Our LSTM-based model demonstrated its prowess in capturing the temporal dynamics of trajectory data, resulting in significantly elevated accuracies compared to previous models. The substantial increase in both training and test accuracies underscores the model's capacity to glean meaningful representations from the data and generalize effectively to unseen instances.

The culmination of our efforts revealed that the LSTM-based model delivered the best performance when configured with specific hyperparameters: a learning rate of 0.001, dropout rate of 0.2, no explicit activation function defined, and trained over 25 epochs. Notably, this configuration yielded a remarkable training accuracy of 88.90% and a testing accuracy of 84.89%.

In conclusion, the success of the LSTM-based model underscores the importance of selecting appropriate model architectures for sequential data analysis. Moving forward, further exploration of advanced neural network architectures and optimization techniques may lead to even greater improvements in performance for similar classification tasks. Additionally, continued research into feature engineering and data preprocessing methods could enhance the model's ability to extract relevant information from trajectory data, ultimately contributing to the advancement of taxi driver classification systems.