The University of Jordan

King Abdullah II School for Information Technology

Machine Learning and Neural Networks Course

Fall Semester, 2023/2024

Done By:

Anas Moosa 0212567

Khaled Shrouf 0215106

Salem Nsour 0213371

Table of Contents

Classification

Description of the Dataset:

Origin and Nature:
The dataset is a comprehensive collection of 2,000 mobile phone specifications, designed for a classification task. This dataset can be instrumental in understanding the factors that influence mobile phone pricing.

Size and Structure:
Total Entries: 2,000.

Features: 21 features, including both technical specifications and features relevant to consumer preferences.

Technical Specifications: Includes battery_power, clock_speed, mobile_wt (weight), n_cores (number of processor cores), px_height (pixel resolution height), px_width (pixel resolution width), ram (random access memory), and talk_time.

Consumer-Oriented Features: Includes blue (Bluetooth capability), dual_sim, fc (front camera megapixels), four_g, int_memory (internal memory), m_dep (mobile depth), pc (primary camera megapixels), sc_h (screen height), sc_w (screen width), three_g, touch_screen, and wifi.

Target Variable:
price_range, indicating the price category of each mobile device,

Into classes (0, 1, 2, 3).

Data Types and Statistics:

The dataset consists mainly of integer and floating-point values.

Battery Power: Varies significantly, reflecting the diverse range of mobile phones.

RAM: Shows a wide range from 256 to 3998, indicating a mix of low-end to high-end mobile phones.

Screen Dimensions: sc_h and sc_w suggests a variety of screen sizes, catering to different user preferences.

Camera Quality: Represented by fc and pc, indicating the range of camera capabilities in the dataset.

Data Analysis:

We used the correlation heat map and the importance of the features plot to analyze the data and we found the following. (see figure 1)

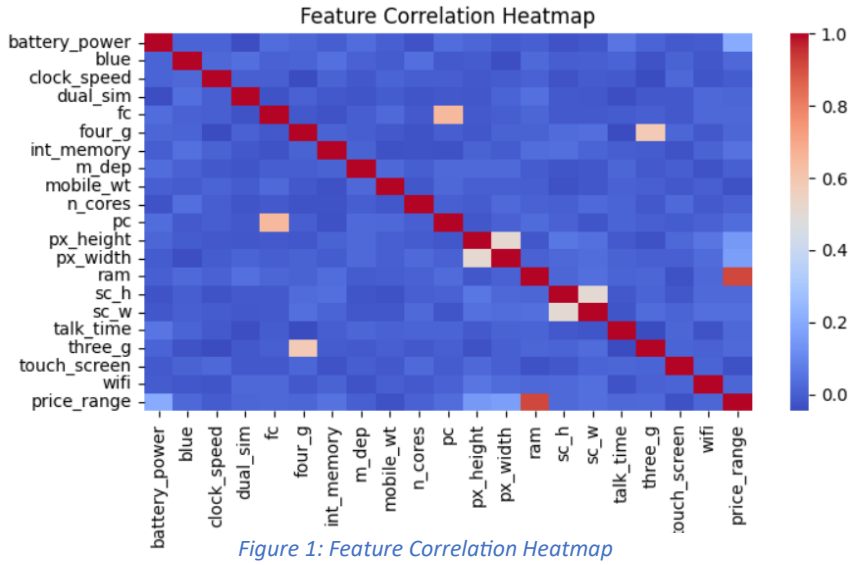

Figure 1: Feature Correlation Heatmap

The correlation heatmap can reveal potential multicollinearity issues or redundant features, which could be addressed in preprocessing steps.

The feature importance's plot assists in identifying the most significant predictors for the classification task, potentially guiding further data collection or feature engineering efforts. (see figure 2)

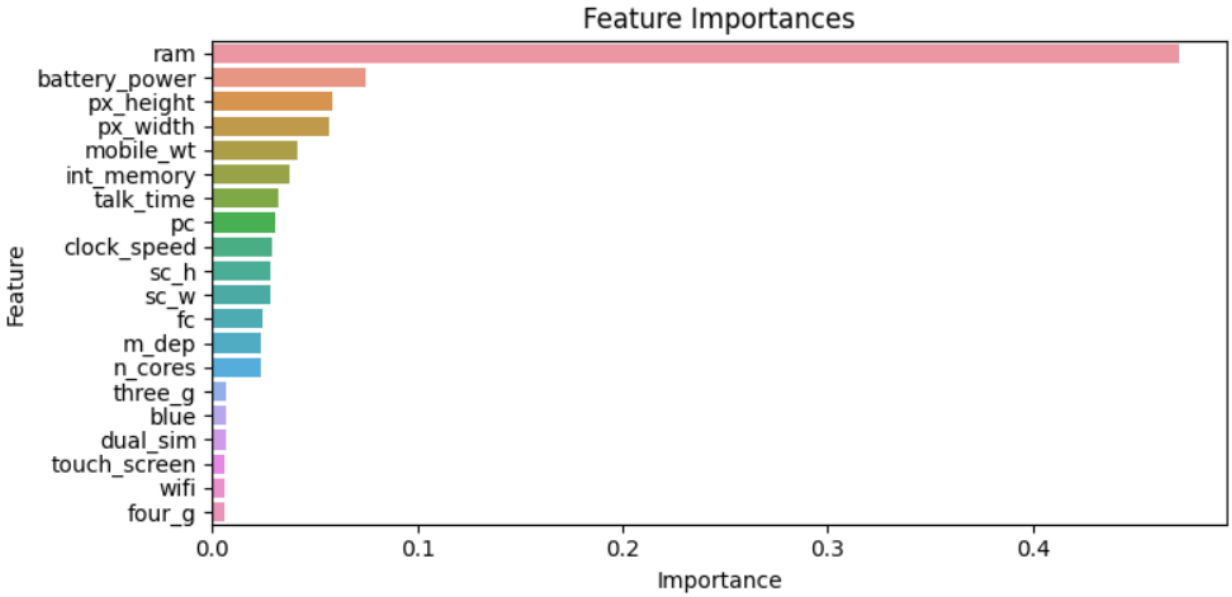

Figure 2 : Feature Importances

Preprocessing Considerations:

Missing Values: we didn't find any missing values on our data.

Dimensionality reduction: we found that 6 out of the 20 features are unimportant so we decided to drop those 6 features (3g, blue, dual_sim, touch_sceen, wifi, 4g).

PCA Analysis: Applied Principal Component Analysis (PCA) to reduce the dataset to two principal components.

Data Splitting: we split the dataset into 80% for training, and 20% for testing.

<u>Dataset Use Cases:</u>

This dataset is ideal for building models to predict price ranges based on mobile specifications, aiding in market analysis, consumer behavior studies, and competitive analysis in the mobile industry.

Description of the used models:

1. Multi-Layer Perceptron (MLP):
   A Multi-Layer Perceptron is a class of feedforward artificial neural network. It consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node, or neuron, uses a nonlinear activation function, except for input nodes. MLP utilizes a supervised learning technique called backpropagation for training, making it capable of learning complex, non-linear mappings from inputs to outputs, suitable for complex classification tasks.

2. Support Vector Machine (SVM):
   Support Vector Machine is a powerful, widely-used learning algorithm for classification and regression. It works by finding the hyperplane that best divides a dataset into classes. The strength of SVM lies in its use of kernels, which allow the algorithm to operate in a high-dimensional space, and its effectiveness in high-dimensional spaces. SVM is especially useful when the dataset has clear margin of separation and when the number of dimensions exceeds the number of samples.

3. Random Forest:
   Random Forest is an ensemble learning method, particularly effective for classification tasks. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. It improves upon the simplicity of decision trees by adding randomness in the tree building process, which helps in reducing overfitting and improving model robustness.

4. Gradient Boosting:
   Gradient Boosting is a machine learning technique used for both classification and regression problems. It builds the model in a stage-wise fashion like other boosting methods, but it generalizes them by allowing optimization of an arbitrary differentiable loss function. In gradient boosting, trees are built sequentially, with each tree being trained to correct

the mistakes of the previous ones. It's known for its effectiveness, especially with large datasets and complex decision boundaries.

Experiment Setup and Results:

1) MLP model:

Experiment Setup:

- Hyperparameter Tuning:
  - Implemented using GridSearchCV from Scikit-learn.
  - Explored a range of hyperparameters:
    - hidden_layer_sizes: Tested different architectures, including [100], [50, 50], and [100, 50, 25] neurons in the layers.
    - activation: Evaluated with 'relu' and 'tanh' functions.
    - solver: Both 'adam' and 'sgd' solvers were assessed.
    - learning_rate_init: Experimented with learning rates 0.01 and 0.001.
    - alpha: Regularization terms 0.0001, 0.001, and 0.01 were tested.
    - batch_size: Sizes 64, 128, and 'auto' were included in the tuning.
    - early_stopping: Evaluated the effect of both enabling and disabling early stopping.
    - validation_fraction: Tested with 0.1 and 0.2 of the training data as validation set.
  - The model was trained with a maximum of 1500 iterations over a 5-fold cross-validation setup.

Model Training:

- The best parameters identified from GridSearchCV were used to train the final MLP model.

```
Best MLP Hyperparameters: {'activation': 'relu', 'alpha':
0.0001, 'batch_size': 64, 'early_stopping': False,
'hidden_layer_sizes': (100, 50, 25),
'learning_rate_init': 0.01, 'solver': 'adam',
'validation_fraction': 0.1}
```

- A random state of 42 was set for reproducibility, with a maximum iteration limit of 1000 for the final model training.

Results:

- Accuracy and Classification Report:
  - The MLP model achieved an accuracy of 74.5%

- Loss Curve:
  - A plot of the loss curve over iterations was generated, showing the model's convergence behavior during training. (see figure 3)

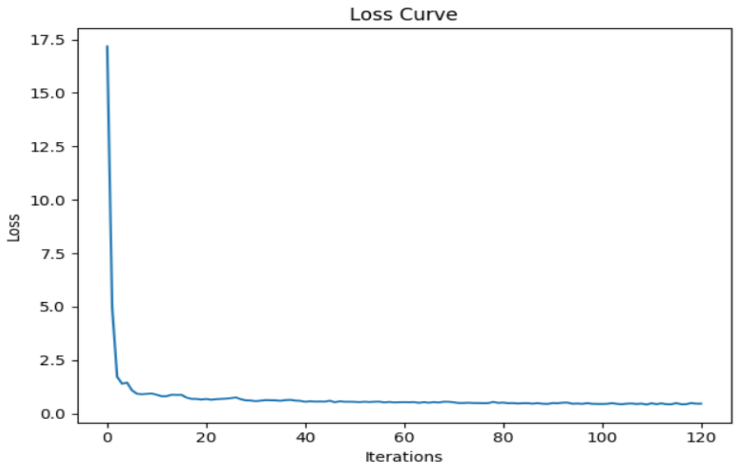

Figure 3 : Loss Curve

- Confusion Matrix:
  - The confusion matrix visually represented the model's performance, highlighting the correct and incorrect predictions across different classes. (see figure 4)

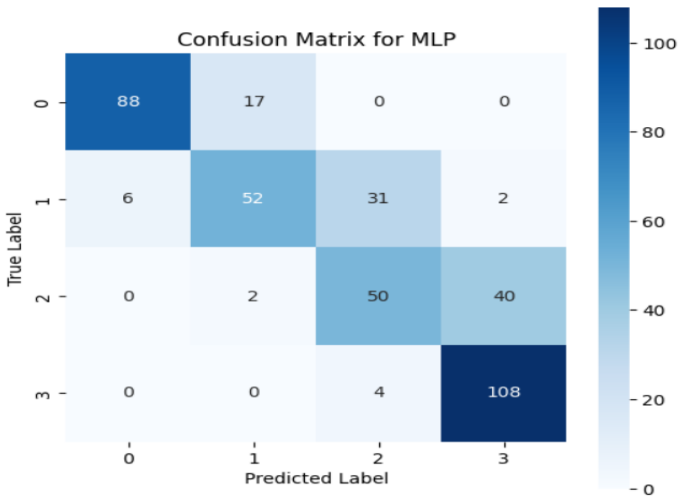

Figure 4 : Confusion Matrix For MLP

2) SVM model:

- **Experiment Setup**

- Hyperparameter Tuning:
  - Conducted using Optuna, a hyperparameter optimization framework.
  - Explored a range of hyperparameters for the SVM:
    - C: Regularization parameter, varied from 0.1 to 10.
    - gamma: Kernel coefficient, tested with 'scale' and 'auto'.
    - kernel: Types of kernels evaluated included 'linear', 'poly', 'rbf', and 'sigmoid'.
  - The model aimed to maximize accuracy, with the tuning process running for 50 trials.

Model Training:

- The best parameters obtained from Optuna were used to retrain the final SVM model.

```
Best hyperparameters for SVC: {'C': 7.86231872191541,
'gamma': 'auto', 'kernel': 'linear'}
```

- Set with a random state of 42 for reproducibility

Results:

- Accuracy:
  - The optimized SVM model achieved an accuracy of 98%.
- Confusion Matrix:
  - Visual representation of the model's performance, highlighting the true positives, true negatives, false positives, and false negatives. (see figure 5)

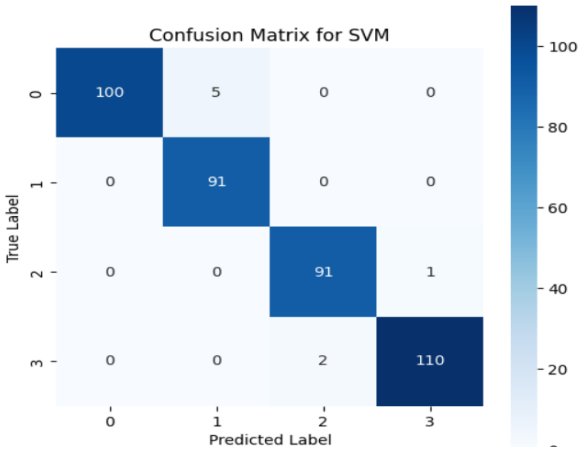

Figure 5 : Confusion Matrix For SVM

- ROC Curve and AUC:
  - Generated a Receiver Operating Characteristic (ROC) curve, showcasing the trade-off between sensitivity and specificity.
  - The Area Under the Curve (AUC) was 1.00, indicating the model's effectiveness in distinguishing between classes. (see figure 6)

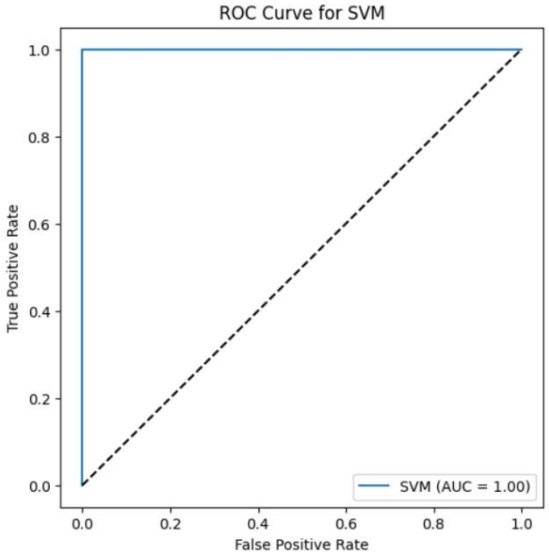

Figure 6 : ROC Curve For SVM

- PCA Analysis:
  - Applied Principal Component Analysis (PCA) to reduce the dataset to two principal components for visualization.
  - The PCA-transformed data was visualized, showing the separation of classes and the decision boundary of the SVM model. (see figure 7)

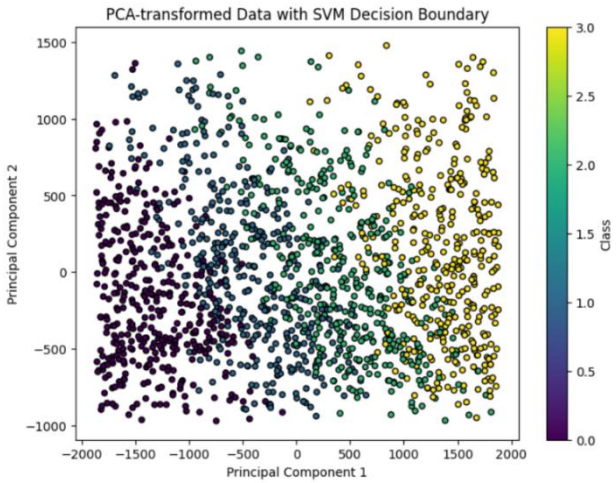

Figure 7 : PCA-SVM Decision Boundary

3) Random Forest Model:

- **Experiment Setup**

- Hyperparameter Tuning:
  - Conducted using GridSearchCV from Scikit-learn
  - Explored a range of hyperparameters for the Random Forest:
    - n_estimators: Number of trees in the forest, tested with values 50, 100, and 150.
    - max_features: Maximum number of features considered for splitting a node, tested with 'auto', 'sqrt', and 0.5.
    - max_depth: Maximum depth of the tree, with values 5, 10, and 15.
    - min_samples_split: Minimum number of samples required to split a node, tested with 4, 6, and 8.
    - min_samples_leaf: Minimum number of samples required at a leaf node, tested with 3, 4, and 5.
  - The tuning process used a 5-fold cross-validation setup.

- **Model Training:**
  - The best parameters identified from GridSearchCV were used to train the final Random Forest model.
  - The optimized Random Forest classifier was then fitted to the training data.

Results:

- Accuracy:
  - The optimized RF model achieved an accuracy of 91.5%.

- Tree Visualization:
    - A single tree from the Random Forest was visualized, offering insights into the decision-making process of the model. (see figure 8)

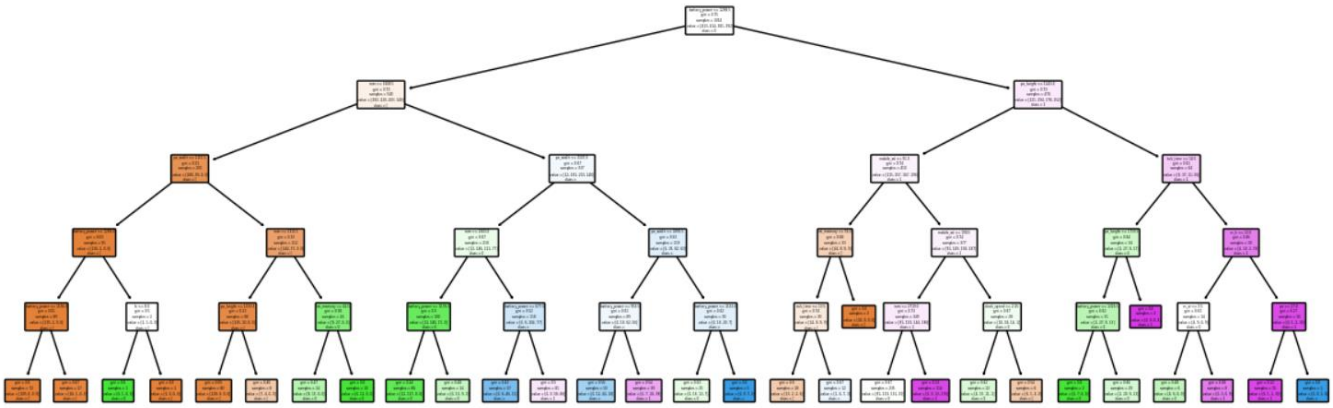

Figure 8 : Random Forest Tree

- Feature Importance:
    - The top 10 most important features were identified and visualized in a bar chart, highlighting their relative importance in the model. (see figure 9)

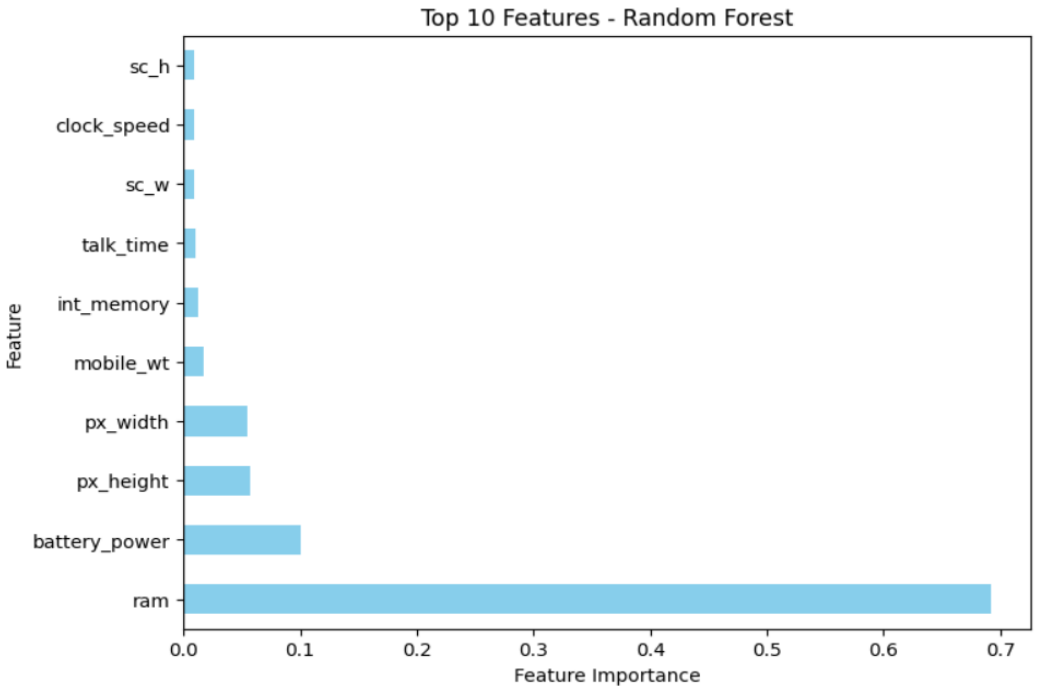

Figure 9 : Top 10 Features For RF

4) Gradient Boosting Model:

- **Experiment Setup**

- Hyperparameter Tuning:
  - Conducted using RandomizedSearchCV from Scikit-learn
  - Explored a range of hyperparameters for Gradient Boosting:
    - n_estimators: Number of boosting stages to be run, tested with values 60, 80, and 100.
    - learning rate: Rate of contribution of each tree, with values 0.01, 0.02, and 0.05.
    - max_depth: Maximum depth of the individual estimators, tested with 2 and 3.
    - min_samples_split: Minimum number of samples required to split a node, tested with 6, 8, and 10.
    - min_samples_leaf: Minimum number of samples required at a leaf node, tested with 4, 6, and 8.
    - subsample: Fraction of samples used for fitting the individual base learners, tested with 0.6, 0.7, and 0.8.
    - max_features: Maximum number of features considered for splitting a node, tested with 'auto' and 'sqrt'.
  - The tuning process used a 5-fold cross-validation setup and was iterated for 100 trials.

- **Model Training**
  - The best parameters identified from the random search were used to train the final Gradient Boosting model.

  - The optimized Gradient Boosting classifier was then fitted to the training data.

- **Results:**
  - Accuracy and Classification Report:
  - The optimized GB model achieved an accuracy of 89%.

- Learning Curve:
    - A learning curve was plotted to evaluate the model's performance over different training set sizes, indicating how well the model learns as more data is provided. (see figure 10)

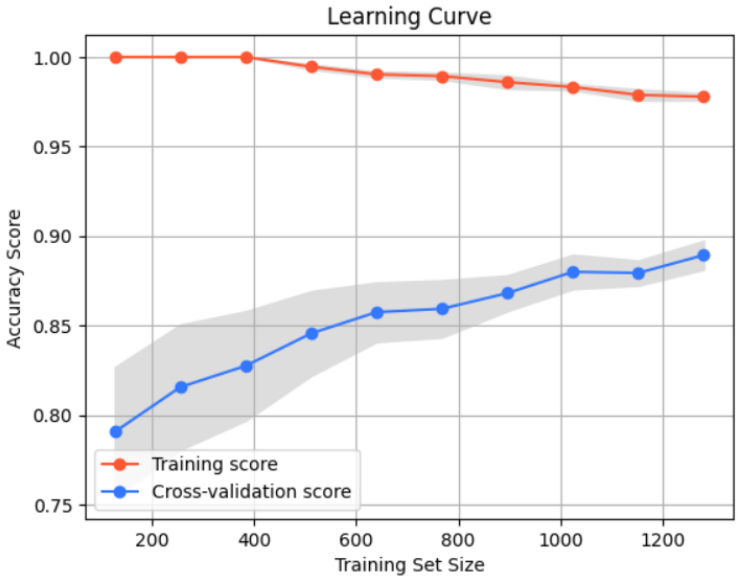

Figure 10 : Learning Curve

- Feature Importance:
    - The top 10 most important features were identified and visualized in a bar chart, highlighting their relative importance in the model's decision-making process. (see figure 11)

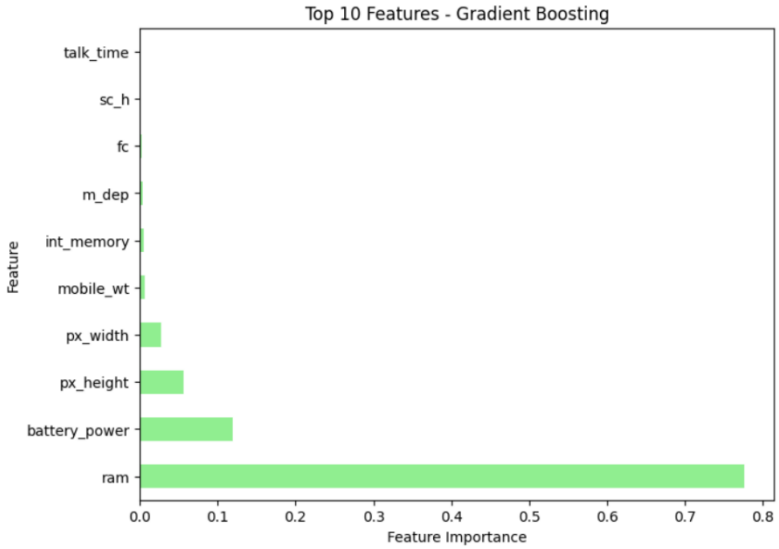

Figure 11 : Top 10 Features For GB

Comparative Analysis of Classifier Performances:

A comprehensive performance comparison of the four classifiers - MLP, SVM, RF (Random Forest), and GB (Gradient Boosting) - is presented below. The table summarizes key performance metrics for each model, including accuracy, precision, recall, and F1-score. These metrics collectively provide a holistic view of each model's performance.

- Comparative classifier performances Table: (see table 1)

Table 1 : Accuracy, Precision, Recall and F1-score of Models

Classifier	Accuracy	Precision	Recall	F1-score
MLP	0.765	0.7663	0.765	0.7615
SVM	0.9675	0.9807	0.98	0.98
RF	0.91	0.9116	0.91	0.9104
GB	0.89	0.892	0.89	0.8906

Summary:

In this study, four different machine learning models were evaluated for their effectiveness in a classification task. These models included a Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), Random Forest (RF), and Gradient Boosting (GB). Each model was carefully tuned for optimal performance using techniques like Grid Search and Randomized Search, and their performance was evaluated based on accuracy, precision, recall, and F1-score.

MLP: Exhibited moderate performance with an accuracy of 0.765. While it was the least accurate among the four models, its relatively balanced precision and recall indicate its potential in scenarios where model interpretability and simplicity are prioritized.

SVM: Emerged as the top-performing model with an impressive accuracy of 0.9675. Its high precision and recall suggest that it is highly capable of handling complex patterns in the data, making it suitable for tasks requiring high reliability and accuracy.

RF (Random Forest): Demonstrated strong performance with an accuracy of 0.91. The model's ability to balance bias and variance, along with its high feature

importance interpretability, makes it a robust choice for a variety of classification tasks.

GB (Gradient boosting): Showed solid performance with an accuracy of 0.89. Known for its effectiveness in handling varied data distributions, its performance indicates its applicability in scenarios where models need to adapt to changes in data over time.

In conclusion, while the SVM model exhibited superior performance in this specific context, each model has its unique strengths that could make it more suitable for different types of classification tasks. The choice of model should be aligned with the specific requirements of the task, including complexity, interpretability, and the nature of the dataset.

Time Series

Description of the Dataset:

Description:
This dataset provides a detailed account of vehicle traffic, quantified on an hourly basis. It is structured into two primary columns:

Hour: This column signifies the hour of the day, ranging from 1 to 1689, corresponding to a 24-hour time format along estimated 70 days. Each entry in this column is an integer representing a specific hour during which vehicle data was recorded.

Vehicles: The 'Vehicles' column enumerates the number of vehicles observed or counted at the corresponding hour. These values are integral, denoting the total vehicle count per hour.

Initial Observations:
A preliminary examination of the dataset reveals data such as 699 vehicles recorded in the first hour, 346 vehicles in the second hour, and so forth. This pattern indicates that the dataset is likely utilized for monitoring vehicular flow in a designated area over different hours of the day.

Preprocessing Considerations:

Normalization Technique: The data is normalized using the Min-Max Scaler, a common preprocessing technique in machine learning. This method scales the 'Vehicles' column, transforming the data so that it falls within a given range, typically 0 to 1. This process is essential for neural network models, as it ensures that all input features have the same scale, preventing any one feature from dominating the learning process.

Data Splitting: The normalized data is split into training and testing sets. 80% of the data is used for training, and the remaining 20% for testing. This split is crucial for evaluating the model's performance on unseen data.

Function to Create Dataset: A custom function, create_dataset, is defined to prepare the dataset for the RNN. This function takes the dataset and a look_back parameter, which specifies the number of previous time steps to use as input variables to predict the next time period.

Generating Training and Testing Data: The create_dataset function is utilized to generate training and testing datasets for the RNN, incorporating the look_back parameter.

Reshaping Input: The RNN requires the input to be in the form of [samples, time steps, features]. This reshaping aligns with the structure of an RNN, which is designed to understand the temporal dynamics of data. The X_train and X_test datasets are reshaped accordingly to fit this requirement. This step is critical for enabling the RNN to effectively learn from the sequential or time-based patterns in the data.

Description of the used models:

The Echo State Network (ESN) represents a unique approach within the realm of Recurrent Neural Networks (RNNs). Its core feature is a large, randomly generated reservoir of neurons with fixed weights. Unlike traditional neural networks where most weights are subject to training, in an ESN, only the output layer is trainable. This reservoir acts as a dynamic temporal processing space, transforming input data into a higher-dimensional space. The linear regression applied at the output layer then learns from this transformed data. The key advantage of ESNs lies in their training efficiency - since the bulk of the network remains unchanged during training, the computational load is significantly reduced. This makes ESNs particularly suitable for applications in time-series prediction, pattern recognition, and dynamic systems modeling, where they provide a more manageable alternative to more complex RNN structures.

Long Short-Term Memory networks (LSTMs) are an advanced type of RNN specifically designed to address the vanishing gradient problem commonly encountered in traditional RNNs. The structure of an LSTM is composed of various memory blocks, each containing a set of gates: input, output, and forget gates. These gates are critical in managing the flow of information. They determine what information is retained or discarded, allowing the network to maintain a longer memory. This capability is particularly beneficial for tasks involving sequential data with long-range dependencies, such as language modeling and time-series analysis. LSTMs have gained popularity in various applications, ranging from natural language processing to complex sequence prediction tasks, due to their ability to capture long-term dependencies in data effectively.

Bidirectional Long Short-Term Memory (BI-LSTM) networks extend the concept of LSTMs by introducing a two-way flow of information. Unlike standard LSTMs that process data in a single direction (forward), BI-LSTMs have two separate layers for processing the input sequence. One-layer processes the sequence from start to end, while the other processes it from end to start. This bidirectional processing allows the network to capture context from both past and future states, providing

a more comprehensive understanding of the sequence. This attribute makes BI-LSTMs exceptionally well-suited for applications where the context of the entire sequence is crucial, such as in sophisticated language modeling, speech recognition, and text generation. The enhanced context awareness of BI-LSTMs often results in improved performance over traditional LSTMs, especially in tasks requiring nuanced understanding of sequential data.

Echo State Network (ESN) Model:

EchoStateNetwork Class Definition:

- The EchoStateNetwork class is defined with an initialization method __init__ that sets up the network's structure. Key parameters include input_size, reservoir_size, spectral_radius, and sparsity.
- input_weights are randomly initialized and connect the input to the reservoir.
- reservoir_weights are also initialized randomly but are then sparsely connected and scaled to achieve the desired spectral radius, a critical factor in the stability and dynamics of the reservoir.

Reservoir State Update:

- The _reservoir_state_update private method updates the reservoir's state. It uses the tanh activation function, incorporating the effects of both the current state and the input data.

Model Training (fit method):

- The fit method initializes the reservoir states and updates them for each time step in the training data (X).
- A Ridge Regression (from sklearn.linear_model) is then used to train the readout layer. This linear model is preferred for its stability and ability to handle collinearity.

Prediction (predict method):

The predict method uses the trained ESN to make predictions on new data. It follows a similar procedure as fit, updating the reservoir states and then using the trained readout weights to generate predictions.

Instantiation and Application:

- An instance of EchoStateNetwork is created with specified input_size and reservoir_size.
- The ESN is trained using X_train and y_train.
- Predictions are then made on both the training and testing data.

Model Evaluation:

- The performance of the model is evaluated using the mean squared error (MSE) metric from sklearn.metrics. The MSE is computed for both the training and testing datasets, providing an indication of the model's accuracy.

Making Predictions on New Data:

- The model makes predictions on the last few entries of the test data (new_input_data).
- The predictions are then inverse transformed using the previously defined MinMaxScaler (scaler) to bring them back to the original scale.

Output:

- The final output includes the MSE values for both training and test sets, showing the model's performance, and a flattened array of predictions on the new input data, representing the model's forecast in the original scale of the data.

```
(0.03486517463912167, 0.03513322970557431)
array([ 193.55924964,  189.63611615,  440.64406888, 1529.16525209,
       4843.0005764 ])
```

Visualization:

Training and Testing Data Error Plot:

- This plot visualizes the errors (residuals) for both training and testing data.
- The left subplot represents the error over time for the training data, showing how much the model's predictions deviate from the actual values during the training phase.
- The right subplot shows the error for the testing data, indicating the model's prediction accuracy on new, unseen data. (see figure 12)

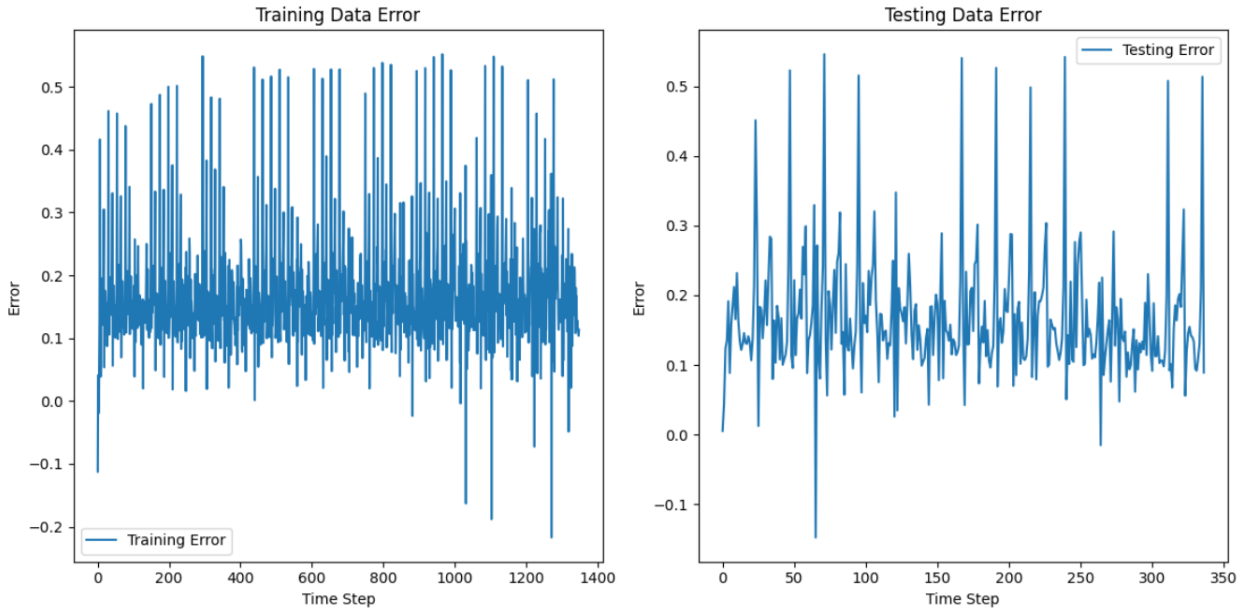

Figure 12: Training & Testing data error

Actual vs. Predicted Values Plot:

- This plot compares the actual values of the test dataset with the values predicted by the model.
- It provides a visual representation of how closely the predicted values align with the true values.
- This plot is crucial for assessing the model's predictive performance, highlighting areas where the model performs well and areas where it may have discrepancies. (see figure 13)

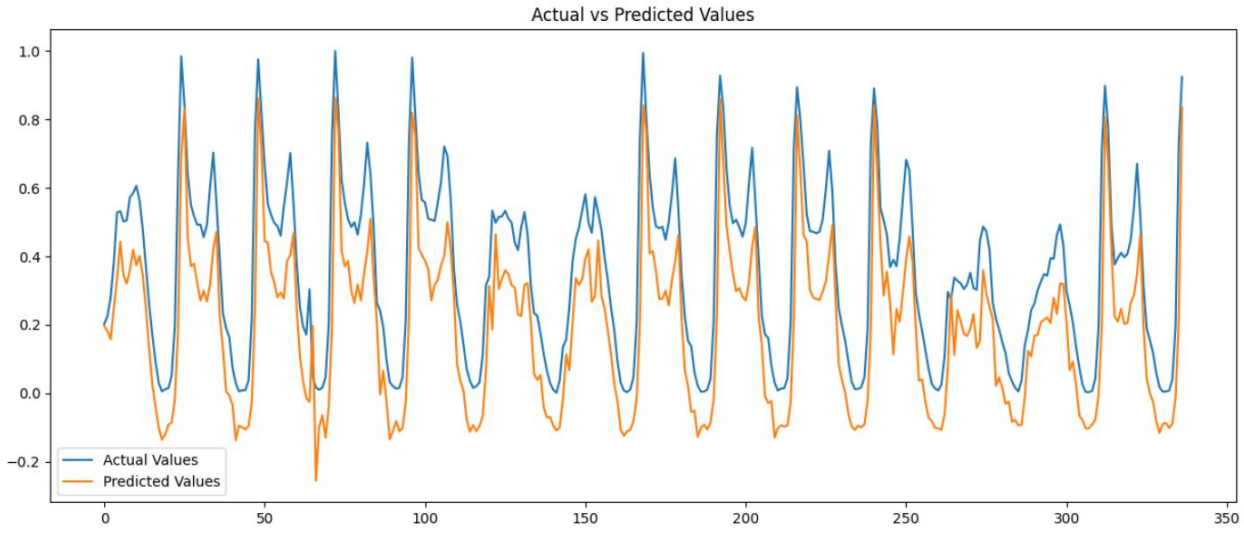

Figure 13: Actual vs Predicted Values

Residuals Plot:

- The residuals plot shows the difference between the actual values and the predicted values (residuals) over time for the test data.
- This plot is useful for identifying patterns in the residuals, such as periods of higher or lower prediction errors, which can indicate model biases or areas where the model may be improved.(see figure 14)

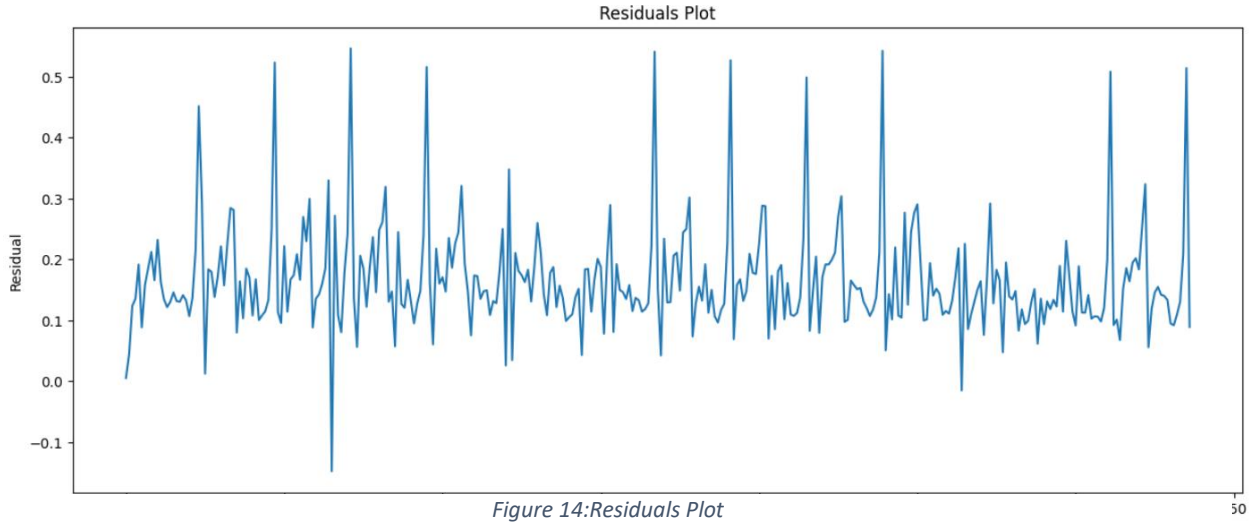

Figure 14:Residuals Plot

Distribution of Predicted vs. Actual Values:

- This plot uses kernel density estimation (KDE) to visualize the distribution of both the actual and predicted values.
- It gives an overview of how the predicted values' distribution compares with the actual values' distribution, highlighting whether the model is over or underestimating. (see figure 15)

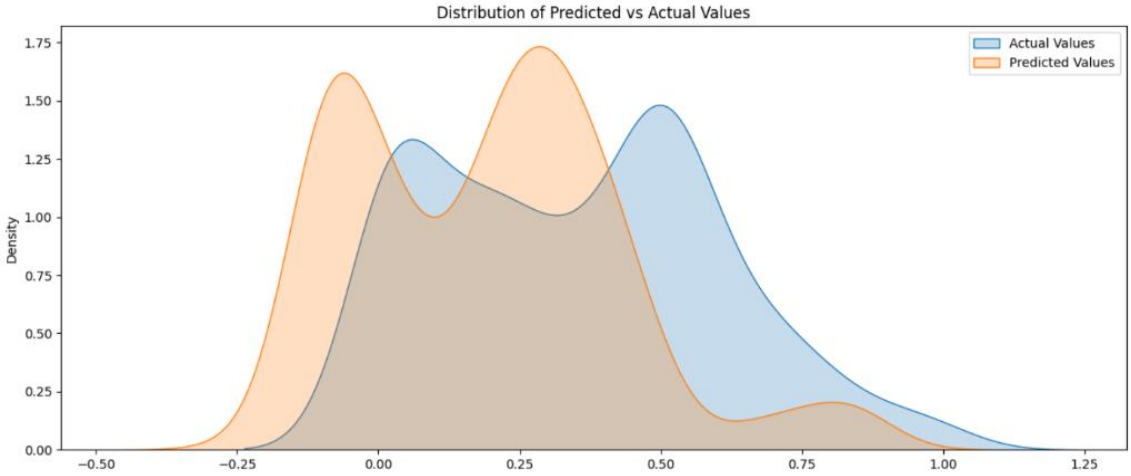

Figure 15:Distribution of Predicted vs Actual Values

Error Distribution Plot:

- This histogram, enhanced with a kernel density estimate (KDE), shows the distribution of the residuals (errors).
- It is crucial for understanding the nature of the errors made by the model. For instance, a normal distribution centered around zero would indicate good model performance
- The plot can reveal biases in the model's predictions, such as a tendency to consistently overpredict or underpredict values. (see figure 16)

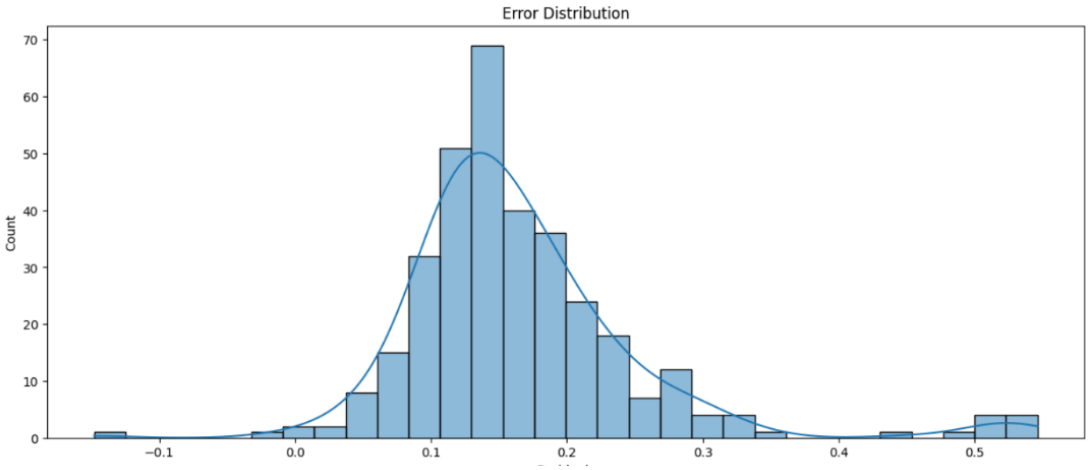

Figure 16:Error Distribution Plot

Long Short-Term Memory (LSTM) Model:

Model Building:

- The LSTM model is built using a sequential model architecture, a standard approach in Keras for creating models in a step-by-step fashion.
- The first layer is an LSTM layer with 50 units. The return_sequences=True parameter is set, indicating that the output for each input time step is required as input to the next layer.
- The second LSTM layer also has 50 units but does not return sequences, meaning it only returns the output for the last time step in the input sequence, suitable for making a prediction.
- A Dense layer with a single unit is added as the output layer. It is responsible for producing a single continuous output, which is typical for regression problems like time-series prediction.

Model Compilation:

- The model is compiled with the Adam optimizer, a popular choice for deep learning applications. The learning rate is set to 0.01, which determines the step size at each iteration while moving toward a minimum of the loss function.

- The loss function used is 'mean_squared_error', which is appropriate for regression problems and works by calculating the average of the squares of the differences between the predicted and actual values.

Model Training:

- The model is trained on the training dataset (X_train, y_train) for 20 epochs, with a batch size of 150.
- validation_data is set to the testing dataset (X_test, y_test), allowing the model to evaluate its performance on unseen data after each epoch.
- The verbose=1 setting enables visual feedback in the console during the training process.
- shuffle=False is specified, which is often appropriate for time-series data where the order of the data points is important.

Model Evaluation:

- The training and validation loss (mean squared error) for each epoch are captured in lstm_train_loss and lstm_test_loss respectively. These metrics are crucial for understanding how the model's performance evolves over the training process.

Additional LSTM Model:

- Another LSTM model is defined with similar architecture but uses the 'relu' activation function in the LSTM layer, and is compiled with the default settings for the Adam optimizer.

Visualizations:

Training and Testing over epochs:

- This plot shows the loss (mean squared error) for both training and validation (testing) datasets over the course of 20 training epochs.
- The training loss curve (labeled as 'Training Loss') and the validation loss curve (labeled as 'Validation Loss') are plotted to visualize how the model's performance evolves during training.(see figure 17)

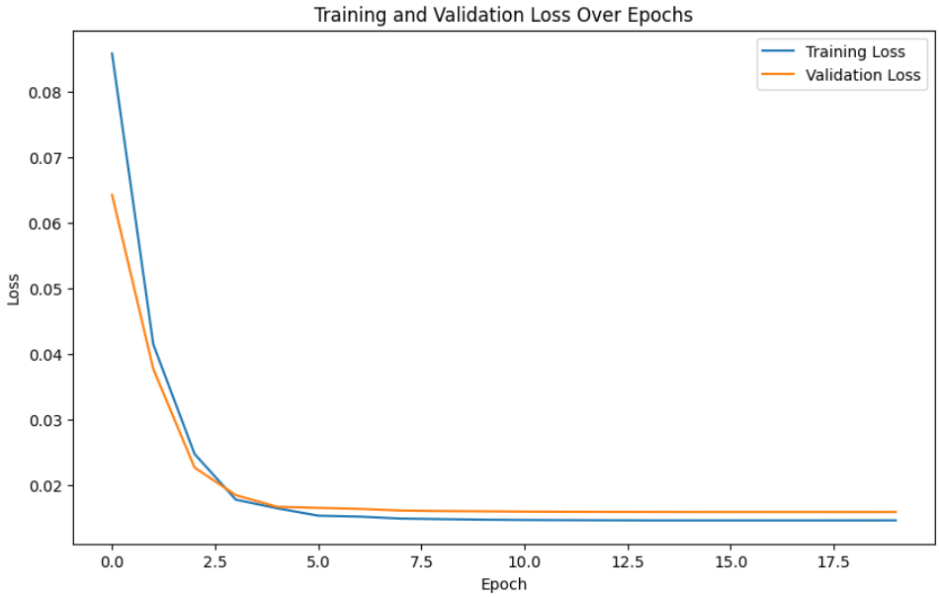

Figure 17:Training and Testing over epochs

Actual vs. Predicted Values Plot:

(see figure 18)

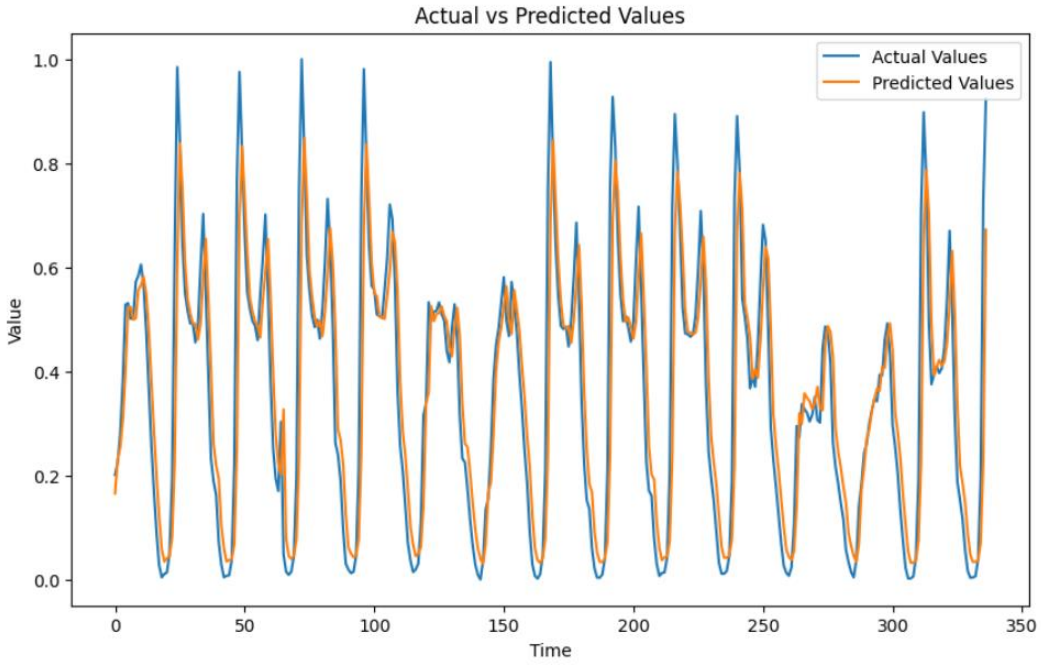

Figure 18: Actual vs. Predicted Values For LSTM

Residuals Plot:
(see figure 19)

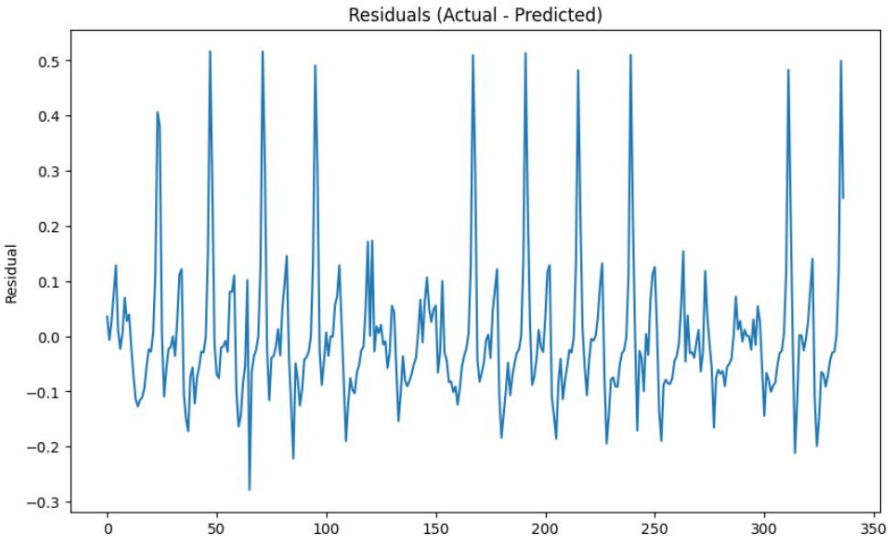

Figure 19: Residuals Plot For LSTM

Distribution of Residuals:

- This histogram shows the distribution of the residuals.
- The number of bins is set to 30 to provide a detailed view of the residuals' distribution.(see figure 20)

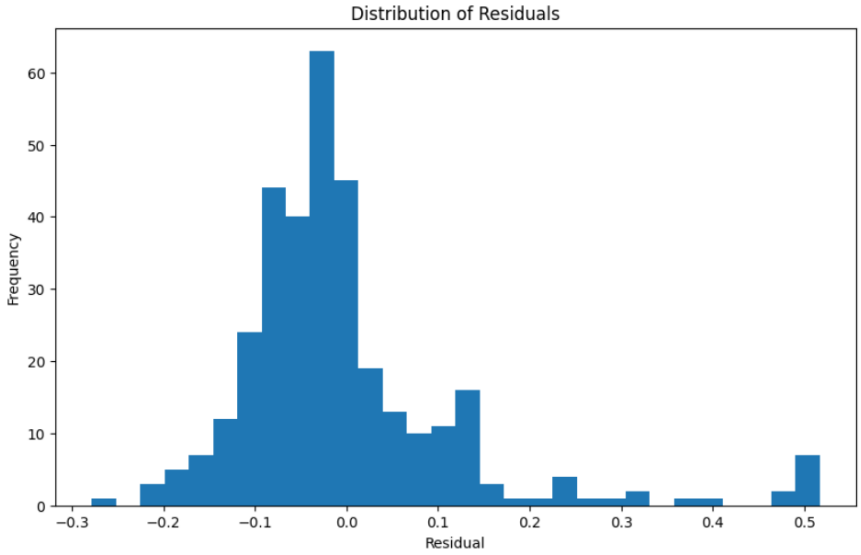

Figure 20:Distribution of Residuals

Bidirectional Long Short-Term Memory (BI-LSTM) Model:

Model Building:

- The BI-LSTM model is constructed using a sequential architecture, which allows for layer-by-layer model assembly.
- The first layer is a Bidirectional LSTM layer with 50 units. The Bidirectional wrapper allows the LSTM to process the input sequence in both forward and backward directions, providing a more comprehensive understanding of the sequence context.
- The return_sequences=True parameter in the first BI-LSTM layer ensures that the output for each input time step is passed to the next layer, essential for stacking LSTM layers.
- A second Bidirectional LSTM layer, also with 50 units, is added. This layer does not return sequences and is geared towards preparing the output for the final prediction.
- The model concludes with a Dense layer of a single unit, serving as the output layer for producing continuous numerical predictions.

Model Compilation:

- The BI-LSTM model is compiled using the Adam optimizer with a learning rate of 0.01, facilitating efficient training of the model.
- The loss function is set to 'mean_squared_error', appropriate for regression tasks like time-series forecasting, focusing on minimizing the average of the squared differences between predicted and actual values.

Model Training:

- The model is trained on the X_train and y_train datasets for 40 epochs with a batch size of 175.
- Validation is performed using the X_test and y_test datasets, allowing for performance assessment on unseen data after each epoch.

- The verbose=1 setting provides visual feedback during training, while shuffle=False is important for time-series data where the sequence of data points matters.

Model Evaluation:

- The training and validation loss for each epoch are stored in bi_lstm_train_loss and bi_lstm_test_loss, respectively. These metrics are crucial for monitoring the model's learning progress and for identifying issues like overfitting.
- The model's performance is further evaluated using the mean squared error (MSE) metric, calculated using the y_test dataset and the predictions from the BI-LSTM model (y_pred).
- The residuals, calculated as the difference between the actual and predicted values, offer additional insight into the model's prediction accuracy.

Visualizations:

Training and Testing over epochs:

(see figure 21)

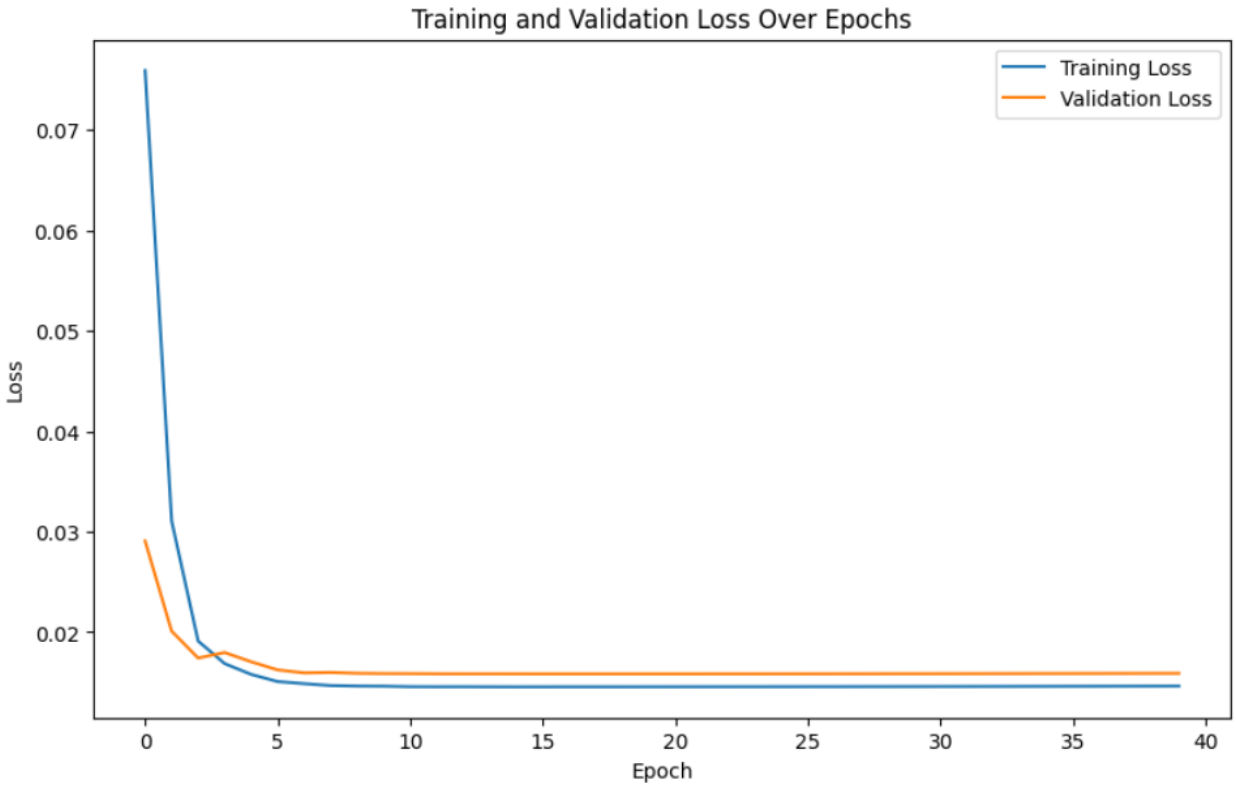

Figure 21:Training and Testing over epochs For BI-LSTM

Actual vs. Predicted Values Plot:

(see figure 22)

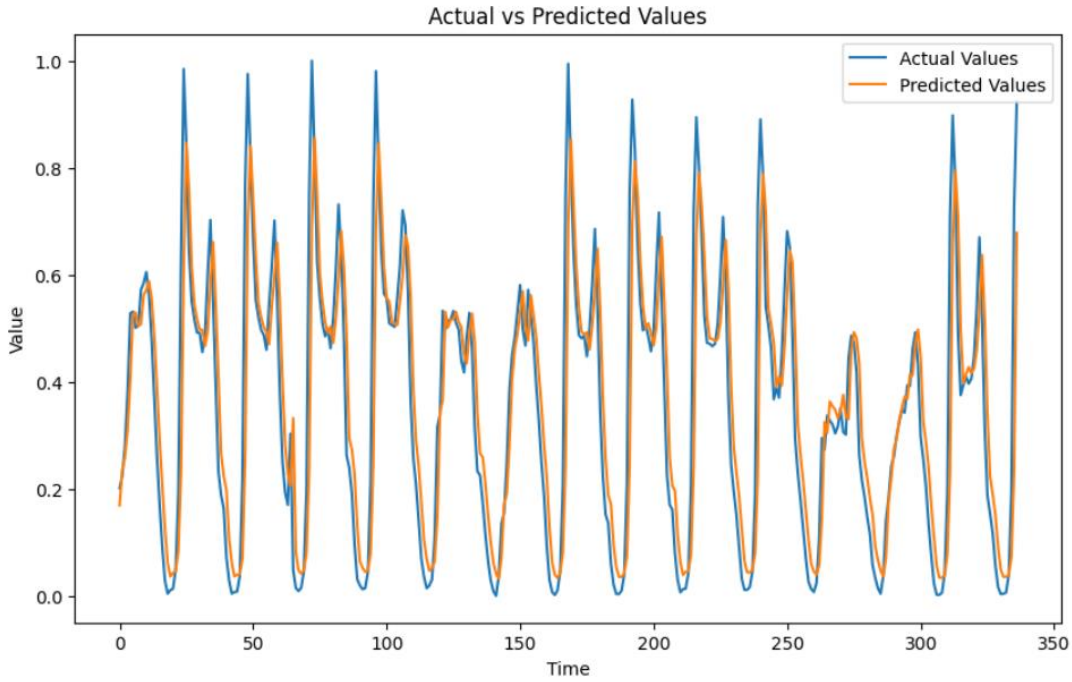

Figure 22:Actual vs Predicted Values For BI-LSTM

Residuals Plot - Bi-LSTM Model:

- The residuals plot visualizes the difference between the actual and predicted values, shown as residuals, for each data point in the test set. (see figure 23)

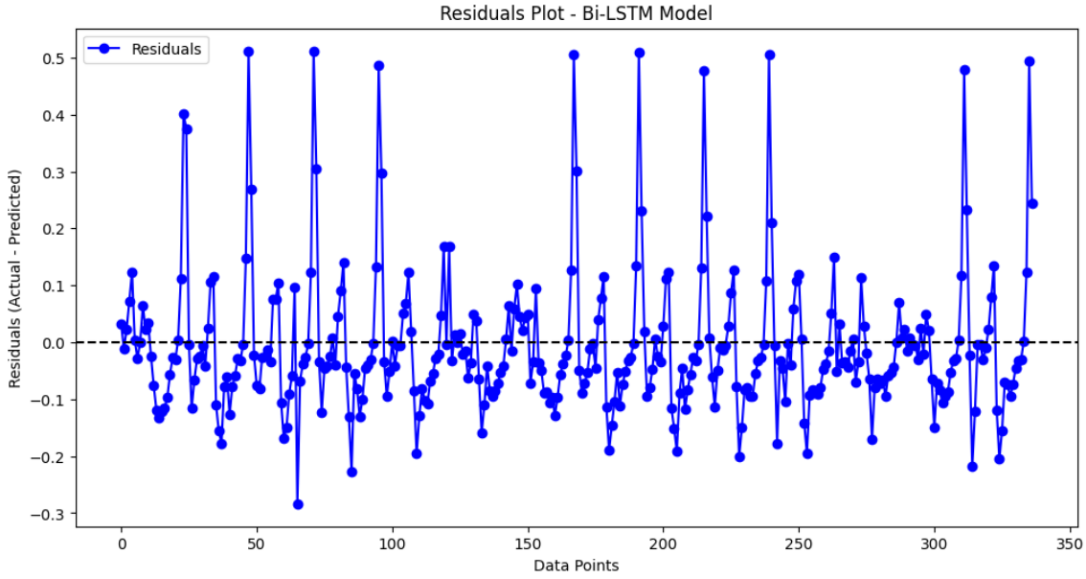

Figure 23:Residuals Plot For Bi-LSTM

Table of Figures