LABORATORY REPORT

Application Development Lab (CS33002)

B.Tech Program in ECSc

Submitted By

Name: Shriya Shukla

Roll No: 2230201

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Experiment Number	3
Experiment Title	To perform stock price prediction using Linear Regression and LSTM model
Date of Experiment	21/01/2025
Date of Submission	27/01/2025

Objective:- To perform stock price prediction using Linear Regression and LSTM model

Procedure:-

1. Collect historical stock price data.
2. Preprocess the data for analysis (missing data, scaling, splitting into train/test).
3. Implement Linear Regression to predict future stock prices.
4. Design and train an LSTM model for time-series prediction.
5. Compare the accuracy of both models.
6. Create a Flask backend for model predictions.
7. Build a frontend to visualize predictions using charts and graphs.

Code:-

- **MODEL TRAINING CODE:**

```
import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
import joblib

# Download historical stock data
ticker = "AAPL"   # Example: Apple Inc.
data = yf.download(ticker, start="2015-01-01", end="2023-12-31")
data.to_csv('stock_data.csv')   # Save the data to a CSV file
```

```
print(data.head())

# Load the data
data = pd.read_csv('stock_data.csv')

# Inspect the dataset
print("First few rows of data:")
print(data.head())
print("\nData types:")
print(data.dtypes)
print("\nMissing values:")
print(data.isnull().sum())

# Convert relevant columns to numeric
numeric_columns = ['Close', 'High', 'Low', 'Open', 'Volume']  # Adjust based on your dataset
for col in numeric_columns:
    data[col] = pd.to_numeric(data[col], errors='coerce')  # Convert to numeric, invalid values become NaN

# Drop rows with missing or invalid data
data = data.dropna()

# Verify cleaning
print("\nAfter cleaning:")
print(data.isnull().sum())
print(data.head())

# Scaling 'Close' prices for LSTM
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data[['Close']])  # Scale 'Close' prices

# Save the scaler for future use (for consistent scaling)
joblib.dump(scaler, 'scaler.pkl')  # Save the scaler

# Train/test split (80% train, 20% test)
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]

# Print shapes of train/test sets
print("\nTrain data shape:", train_data.shape)
```

```
    print("Test data shape:", test_data.shape)

    # Prepare features and labels for Linear Regression
    X = data.index.values.reshape(-1, 1)   # Date as feature
    y = data['Close'].values   # Closing prices as label

    # Train/test split for Linear Regression
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Train Linear Regression
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)

    # Predictions
    y_pred = lr_model.predict(X_test)

    # Evaluate the model
    print("MSE (Linear Regression):", mean_squared_error(y_test,
y_pred))
    print("R² Score (Linear Regression):", r2_score(y_test, y_pred))

    # Prepare data for LSTM
    X_train_lstm, y_train_lstm = [], []
    for i in range(60, len(train_data)):
        X_train_lstm.append(train_data[i-60:i])
        y_train_lstm.append(train_data[i])
    X_train_lstm, y_train_lstm = np.array(X_train_lstm),
np.array(y_train_lstm)

    # Design LSTM Model
    lstm_model = Sequential()
    lstm_model.add(LSTM(50, return_sequences=True,
input_shape=(X_train_lstm.shape[1], 1)))
    lstm_model.add(LSTM(50))
    lstm_model.add(Dense(1))
    lstm_model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the LSTM model
    lstm_model.fit(X_train_lstm, y_train_lstm, epochs=20,
batch_size=32)
```

```
# Predictions using LSTM (here you'll need to prepare test data for LSTM as well)
lstm_predictions = lstm_model.predict(X_train_lstm)  # Example: Use the training set for prediction (you can replace with test set)

# Compare MSE for Linear Regression and LSTM
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred))
print("LSTM MSE:", mean_squared_error(y_train_lstm, lstm_predictions))

# Save the Linear Regression model
joblib.dump(lr_model, 'linear_regression_model.pkl')

# Save the LSTM model
lstm_model.save('lstm_model.keras')

# Plot Linear Regression predictions
original_predictions_lr = scaler.inverse_transform(y_pred.reshape(-1, 1))
plt.figure(figsize=(10, 6))
plt.plot(y_test, label="Actual Prices", color="blue")
plt.plot(original_predictions_lr, label="Linear Regression Predictions", color="green")
plt.title("Stock Price Prediction with Linear Regression")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.show()

# Plot LSTM predictions (inverse scaling required for LSTM)
original_predictions_lstm = scaler.inverse_transform(lstm_predictions.reshape(-1, 1))
plt.figure(figsize=(10, 6))
plt.plot(y_train_lstm, label="Actual Prices", color="blue")
plt.plot(original_predictions_lstm, label="LSTM Predictions", color="red")
plt.title("Stock Price Prediction with LSTM")
plt.ylabel("Price")
plt.legend()
plt.show()
```

- **FLASK CODE**:

```
from flask import Flask, request, render_template, jsonify
import yfinance as yf
from datetime import datetime
import joblib
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model  # Import correct load function for Keras model

# Load pre-trained models
lr_model = joblib.load('linear_regression_model.pkl')  # Linear Regression model
lstm_model = load_model('lstm_model.keras')  # Correct way to load LSTM model
scaler = joblib.load('scaler.pkl')  # Scaler for inverse scaling

# Initialize Flask app
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Get user input from the form
    stock_symbol = request.form.get('stock_symbol')  # Stock symbol (e.g., 'AAPL')
    input_date = request.form.get('date')  # Date in YYYY-MM-DD format

    # Convert the input date to a datetime object
    try:
        input_date = datetime.strptime(input_date, '%Y-%m-%d')
    except ValueError:
        return jsonify({"error": "Invalid date format. Use YYYY-MM-DD."})

    # Ensure the date is not in the future
    if input_date <= datetime.today():
            return jsonify({"error": "Please enter a future date for
prediction."})

        # Fetch historical stock data for the symbol
        data = yf.download(stock_symbol, start="2015-01-01",
end=datetime.today().strftime('%Y-%m-%d'))

        if data.empty:
            return jsonify({"error": f"No data found for the stock symbol:
{stock_symbol}."})

        # Prepare data for prediction (Linear Regression and LSTM
models)
        X = np.array(range(len(data))).reshape(-1, 1)  # Using the date
index as the feature (you can change this)
        y = data['Close'].values  # Closing prices as the target variable

        # Linear Regression prediction for the next day (example)
        lr_prediction = lr_model.predict(np.array([[len(data)]]))  #
Predicting for the next day
        lr_prediction = scaler.inverse_transform(lr_prediction.reshape(-
1, 1))[0][0]  # Inverse scaling

        # Prepare data for LSTM model (using the last 60 days as input to
predict the next day)
        last_60_days = data['Close'].values[-60:]  # Last 60 days of
closing prices
        last_60_days_scaled = scaler.transform(last_60_days.reshape(-1,
1))  # Scale it

        X_input = last_60_days_scaled.reshape((1, 60, 1))  # Reshaped
for LSTM input
        lstm_prediction = lstm_model.predict(X_input)  # Predict the
next day
        lstm_prediction =
scaler.inverse_transform(lstm_prediction.reshape(-1, 1))[0][0]  # Inverse
scaling

        # Convert predictions to standard Python float (to make them
serializable)
        return jsonify({
            'linear_regression_prediction': float(lr_prediction),  #
Convert to float
```

```
            'lstm_prediction': float(lstm_prediction),   # Convert to float
            'prediction_date': input_date.strftime('%Y-%m-%d')
        })

if __name__ == '__main__':
    app.run(debug=True)
```

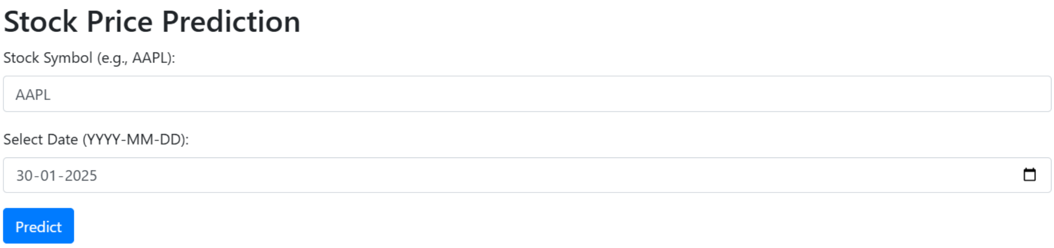
- **HTML CODE**:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stock Price Prediction</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.2/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <h2>Stock Price Prediction</h2>
        <form action="/predict" method="POST">
            <div class="form-group">
                <label for="stock_symbol">Stock Symbol (e.g., AAPL):</label>
                <input type="text" class="form-control" id="stock_symbol" name="stock_symbol" required>
            </div>
            <div class="form-group">
                <label for="date">Select Date (YYYY-MM-DD):</label>
                <input type="date" class="form-control" id="date" name="date" required>
            </div>
            <button type="submit" class="btn btn-primary">Predict</button>
        </form>

        <div id="prediction-results" class="mt-4">
```

```
                <!-- Predictions will be displayed here after the form is
submitted -->
            </div>
        </div>

        <script>
            // Handle form submission asynchronously (AJAX)
            document.querySelector('form').addEventListener('submit',
function(event) {
                event.preventDefault();

                // Get the form data
                const formData = new FormData(event.target);

                // Send AJAX request to Flask server
                fetch('/predict', {
                    method: 'POST',
                    body: formData
                })
                .then(response => response.json())
                .then(data => {
                    if (data.error) {
                        document.getElementById('prediction-
results').innerHTML = <div class="alert alert-
danger">${data.error}</div>;
                    } else {
                        document.getElementById('prediction-
results').innerHTML = `
                            <h4>Prediction Results for
${data.prediction_date}:</h4>
                            <p><strong>Linear Regression
Prediction:</strong>
$${data.linear_regression_prediction.toFixed(2)}</p>
                            <p><strong>LSTM Prediction:</strong>
$${data.lstm_prediction.toFixed(2)}</p>
                        `;
                    }
                })
```

```
                    .catch(error => {
                        document.getElementById('prediction-results').innerHTML = <div class="alert alert-danger">An error occurred while predicting. Please try again.</div>;
                    });
                });
            </script>
        </body>
        </html>
```

Results/Output:



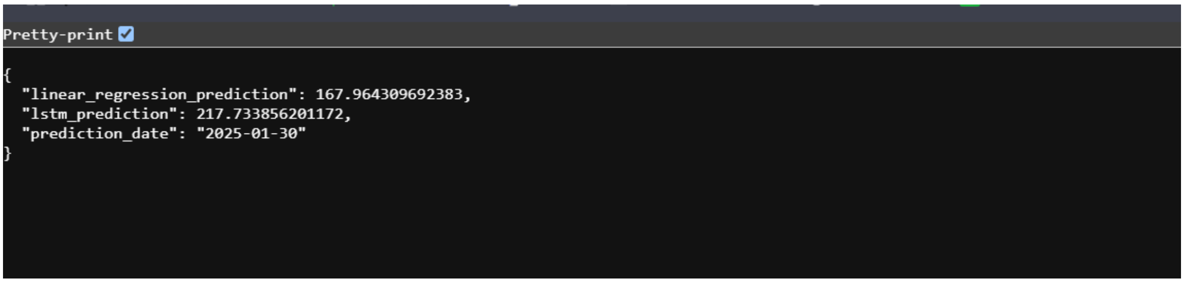
Stock Price Prediction

Stock Symbol (e.g., AAPL):

AAPL

Select Date (YYYY-MM-DD):

30-01-2025



```
{
  "linear_regression_prediction": 167.964309692383,
  "lstm_prediction": 217.733856201172,
  "prediction_date": "2025-01-30"
}
```

Remarks:-

From this experiment I learned how to use machine learning models to To perform stock price prediction using Linear Regression and LSTM models.

Signature of the Student

Somo Prasad Pattnaik

____

Signature of the Lab Coordinator

Bhargav Appasani

____