Sprint 5

Day 9 Task 1:

Sprints for AI Use Case For Demand Forecasting Using POS Transaction Data

Team Members:

Tufail Irshad

Amjad Ali Malik

Zameer Ahmad Mir

Asif Ahmad Najar

Irfan Ahmad Mutoo

Sprint 5 Day 9 Task 1

AI_Forecasted_results

Introduction

This report provides a comprehensive analysis of the performance and optimization of various machine learning models applied to a dataset. The primary goal is to evaluate and compare the models based on their predictive accuracy and error metrics on both training and test datasets. This helps in identifying the most suitable model for making accurate predictions on new data.

Data Preparation

Encoding and Scaling

Before training the models, it was essential to preprocess the data. The preprocessing steps included encoding categorical variables and scaling numerical variables:

- **Categorical Encoding:**
  - Categorical variables such as 'Item Purchased', 'Transaction Mode', and 'Transaction Currency' were encoded using one-hot encoding. This converts categorical variables into a binary format that machine learning algorithms can process.
  - The OneHotEncoder from scikit-learn was used for this purpose.
- **Numerical Scaling:**
  - Numerical variables such as 'Unit Price', 'Quantity', 'Tax', 'Transaction Amount', and 'Total Amount In INR' were standardized to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model training process.
  - The StandardScaler from scikit-learn was used for scaling.

Train-Test Split

The dataset was split into training and test sets using an 80-20 split. The training set (80% of the data) was used to train the models, while the test set (20% of the data) was used to evaluate the models' performance on unseen data. This helps in assessing the models' ability to generalize to new, unseen data.

```
from sklearn.model_selection import train_test_split

df_preprocessed = pd.read_csv('preprocessed_data.csv')

# Split the data into features (X) and target (y)
X = df_preprocessed.drop(['Transaction Amount','Date of Transaction'],
axis=1)
y = df_preprocessed['Transaction Amount']
```

- **Shapes of the datasets:**
  - **X_train shape**: The number of samples and features in the training set.

    ```
    print("Shape of X_train",X_train.shape)
    Shape of X_train (7984, 15)
    ```
  - **X_test shape**: The number of samples and features in the test set.

    ```
    print("Shape of X_test",X_test.shape)
    Shape of X_test (1997, 15)
    ```
  - **y_train shape**: The number of target values in the training set.

    ```
    print("Shape of y_train",y_train.shape)
    Shape of y_train (7984,)
    ```
  - **y_test shape**: The number of target values in the test set.

    ```
    print("Shape of y_test",y_test.shape)
    Shape of y_test (1997,)
    ```

Models Evaluated

The following machine learning models were evaluated in this study:

- **Linear Regression:**

A linear approach to modeling the relationship between the dependent variable and one or more independent variables.

- **Decision Tree Regressor:**

A non-linear model that splits the data into subsets based on feature values, creating a tree-like structure.

- **Random Forest Regressor:**

  An ensemble learning method that builds multiple decision trees and merges their predictions to improve accuracy and control overfitting.

- **Gradient Boosting Regressor:**

  An ensemble technique that builds trees sequentially, each one correcting the errors of the previous one.

- **Support Vector Regressor (SVR):**

  A regression model that uses support vector machines to perform regression analysis, particularly effective in high-dimensional spaces.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
import numpy as np
models = {
```

```
        "Linear Regression": LinearRegression(),
        "Decision Tree": DecisionTreeRegressor(),
        "Random Forest": RandomForestRegressor(),
        "Gradient Boosting": GradientBoostingRegressor(),
        "Support Vector Regressor": SVR(),
    }
results = {}
```

Model Training and Evaluation

- **Cross-Validation**

Cross-validation was employed to ensure the robustness of the model evaluation. For each model, cross-validation was performed on the training set, and the average performance across folds was considered. This process involves:

- Splitting the training data into several subsets (folds).
- Training the model on some folds and validating it on the remaining fold.
- Repeating this process multiple times to ensure the model's performance is consistent.

```
#cross validation
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
mean_cv_score = np.mean(cv_scores)
print(mean_cv_score)
```

```
-5.411724740640088e-09
```

Performance Metrics

The models were evaluated based on the following metrics:

- **Mean Squared Error (MSE):** Measures the average of the squares of the errors between predicted and actual values. It penalizes larger errors more than smaller ones.

- **Root Mean Squared Error (RMSE):** The square root of the MSE, representing the average magnitude of the errors in the same units as the target variable.
- **R-squared (R^2):** Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. Higher values indicate a better fit.

```
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    results[model_name] = {'MSE': mse, 'RMSE': rmse, 'R-squared':
r2}

# Display the results
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"  MSE: {metrics['MSE']}")
    print(f"  RMSE: {metrics['RMSE']}")
    print(f"  R-squared: {metrics['R-squared']}")
    print()
```

```
Model: Linear Regression
  MSE: 4.944737115705312e-09
  RMSE: 7.031882476055265e-05
  R-squared: 0.9999999947713677

Model: Decision Tree
  MSE: 1.1656322364089945e-06
  RMSE: 0.0010796444953821579
  R-squared: 0.9999987674446075

Model: Random Forest
  MSE: 4.711648768581656e-07
  RMSE: 0.000686414508047554
  R-squared: 0.999999501783846

Model: Gradient Boosting
  MSE: 5.773522195070386e-05
  RMSE: 0.007598369690315407
  R-squared: 0.9999389499904586

Model: Support Vector Regressor
  MSE: 0.0030952711624387176
  RMSE: 0.05563516120618972
  R-squared: 0.9967270181422121
```

Best Model Selection

The best model was selected based on the Root Mean Squared Error (RMSE). The model with the lowest RMSE was chosen as the best performing model:

```
# Select the best model based on RMSE (or any other preferred metric)
best_model_name = min(results, key=lambda x: results[x]['RMSE'])
best_model = models[best_model_name]

print(f"Best Model: {best_model_name}")
```

```
Best Model: Linear Regression
```

Forecasting Future Transactions:

Using the selected best model, the script generates forecasted transaction amounts for future dates. It does this by providing the model with features corresponding to those future dates (such as day of the week and month). The model then predicts the transaction amounts for those dates.

```
# Forecast demand for new data using the best model
new_dates = pd.date_range(start='2024-04-17', periods=100, freq='D')
new_features = pd.DataFrame({
    'DayOfWeek': new_dates.dayofweek,
    'Month': new_dates.month
})
new_predictions = best_model.predict(new_features)

forecast_df = pd.DataFrame({
    'Date': new_dates,
    'Predicted Transaction Amount': new_predictions
})
# Write the forecasted results to a new CSV file
forecast_df.to_csv('forecasted_results.csv', index=False)
print("Forecasted results saved to 'forecasted_results.csv'")
```

Date	Predicted Transaction Amount
17/04/2024	494.3628997
18/04/2024	493.6556228
19/04/2024	492.9483459
20/04/2024	492.241069
21/04/2024	491.5337921
22/04/2024	495.7774534

Actual Values and Predicted Values

Actual values in machine learning represent the true outcomes or observations of the target variable in a dataset. These values are known and serve as the ground truth against which the model's predictions are compared. For example, in a regression task predicting house prices, the actual values would be the real sale prices of the houses in the dataset.

Actual Amount	Predicted Amount
479.92	495.7775
625.62	499.4513
654.3	499.5945
430.15	499.0275
828.57	497.4727

Conclusion

This report provides a detailed comparison of various machine learning models in terms of their predictive accuracy and error metrics. The use of cross-validation ensures that the results are robust and reliable. Based on the evaluation, the best model was selected for further use. The selected model demonstrated the lowest RMSE, indicating its superior performance in making accurate predictions.

Gantt Chart

Sprints	Days Worked on Each Sprint	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total Man-Days
Synthetic Data Generation	2	X	X													2
Data Cleaning	2			X	X											2
Feature Engineering	1					X										1
Model Training							X	X								2
Model Application and Iteration									X							1
Minimization of Bias and Overfitting										X						1
Test Models on Unseen Data or Validation Set											X	X				2
Final Model Selection and Conclusion													X			1
Documentation																
Total Days		X	X	X	X	X	X	X	X	X	X	X				14