Telecom Churn Prediction

Infosys Springboard

Agenda

1. About telecom churn prediction
2. Our Mission and Vision
3. Our Goals
4. Our Milestones
5. Challenges
6. Data Cleaning & Preprocessing
7. ML Model Building
8. Hyperparameter Tuning
9. SVM & Confusion Matrix
10. Conclusion

Github:

https://github.com/springboardmentor113/Batch2_Churn_Modeling_On_Telecom_Data/blob/main/Rohan_kumar_singh/Churn_model-checkpoint.ipynb

About Telecom Churn Prediction

- "Telecom Churn Prediction" aims to empower telecom companies with actionable insights and tools to reduce customer churn, improve retention, and enhance overall business performance in the highly competitive telecommunications market.
- By identifying patterns and predicting which customers are likely to leave, companies can proactively address issues, tailor marketing strategies, and provide targeted offers to retain valuable customers.

Our Mission And Vision

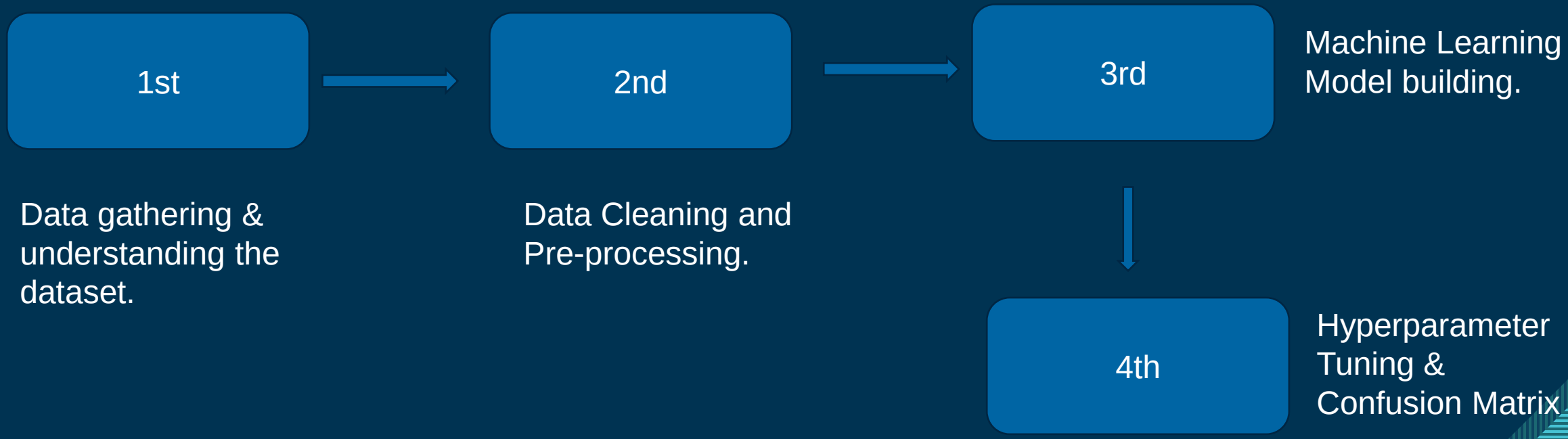
Mission

- To leverage advanced data analytics and machine learning techniques to accurately predict customer churn

Vision

- Enabling telecom companies to proactively manage customer relationships. Through continuous innovation and improvement, the model aspires to create a customer-centric approach in the telecommunications industry, where personalized services and preventative measures lead to long-term customer loyalty and competitive advantage.

Our Milestones



Our Goals

Minimize Churn Rate

Maximise Customer Retention

Optimize Business Performance

Data Cleaning & Pre Processing

- **Convert datatypes of variables which are misclassified:** Ensures accurate analysis and efficient processing.
- **Removing duplicate records:** Ensures accuracy, maintains integrity, and optimizes efficiency in data management and analysis.

3. Remove Duplicate Records

```
[10]: df = df.drop_duplicates()
      print("Dimensions after removing duplicates: ", df.shape)
```

```
Dimensions after removing duplicates:  (25000, 111)
```

- **Removing unique value variables:** Necessary to avoid redundancy, improve efficiency, and ensure meaningful analysis.

4. Remove Unique Value Variables

Removes columns that have only unique value as they do not contribute to the model.

```
[11]: unique_counts = df.nunique()
      df = df.loc[:, unique_counts != 1]
      print("Dimensions after removing unique value variables: ", df.shape)
```

```
Dimensions after removing unique value variables:  (25000, 111)
```

Data Cleaning & Pre Processing

- **Removing zero variance variables:** Necessary to eliminate redundant information, improve model stability, and simplify interpretation.

5. Remove Zero Variance Variables

Eliminates columns with no variability (constant features) which do not help in modeling.

```
]: zero_variance_columns = [col for col in df.columns if df[col].var() == 0]
df = df.drop(zero_variance_columns, axis=1)
print("Dimensions after removing zero variance variables: ", df.shape)
```

```
Dimensions after removing zero variance variables:  (25000, 111)
```

- **Outlier treatment:** Necessary to maintain data integrity, improve model performance, and ensure robust and interpretable analyses

OUTLIER TREATMENT

Boxplot:

It shows the minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. Outliers are often identified as data points that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR, where IQR is the interquartile range (Q3 - Q1).

Capping and Flooring:

Capping and flooring involves setting a cap (upper limit) and floor (lower limit) for data values. Outliers are capped at the maximum threshold or floored at the minimum threshold to limit the effect of extreme values.

Standardization (3 Sigma Approach)

Data points are considered outliers if they lie more than three standard deviations (σ) away from the mean (μ).

```
: z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))
df = df[(z_scores < 3).all(axis=1)]
print("Dimensions after removing outliers using 3 sigma: ", df.shape)
```

```
Dimensions after removing outliers using 3 sigma:  (17624, 111)
```

Data Cleaning & Pre Processing

- . **Missing value treatment:** Essential for maintaining data integrity, improving model performance, and ensuring accurate and statistically valid analyses.
- **Removing highly correlated variables:** Necessary to reduce redundancy, improve model stability, and enhance efficiency in model training and interpretation.

Removing Highly Correlated Variables

Eliminates one of each pair of variables that have a high correlation to avoid multicollinearity.

```
]: # Calculate the absolute value of the correlation matrix
corr_matrix = df.corr().abs()
# Create an upper triangle matrix of the correlation matrix to avoid duplicate pairs
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
# Drop the columns identified as highly correlated from the DataFrame
df = df.drop(to_drop, axis=1)
print("Dimensions after removing highly correlated variables: ", df.shape)
```

```
Dimensions after removing highly correlated variables:  (17624, 81)
```

- **Multicollinearity (VIF > 5):** Crucial to ensure accurate, stable, and interpretable models.

```
In [16]: # Function to calculate Variance Inflation Factor (VIF)
def calculate_vif(df):
    # Create a DataFrame to store VIF values
    vif = pd.DataFrame()
    # Assign column names to the DataFrame
    vif["features"] = df.columns
    # Calculate VIF for each feature and store in the DataFrame
    vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]
    return vif

# Calculate VIF for the initial DataFrame
vif_data = calculate_vif(df)

# Loop to remove features with VIF > 5
while vif_data['VIF'].max() > 5:
    # Identify the feature with the highest VIF
    feature_to_drop = vif_data.loc[vif_data['VIF'].idxmax(), 'features']
    # Drop the feature with the highest VIF from the DataFrame
    df = df.drop(columns=[feature_to_drop])
    # Recalculate VIF for the updated DataFrame
    vif_data = calculate_vif(df)

# Print the dimensions of the DataFrame after removing multicollinear variables
print("Dimensions after removing multicollinear variables: ", df.shape)
```

```
Dimensions after removing multicollinear variables:  (17624, 26)
```

Machine Learning Model Building

- Logistic Regression
- Decision Tree
- Random Forest

Hyperparameter Tuning

Hyperparameter Tuning is the process of finding the best settings for a machine learning model to improve its performance.

This involves adjusting parameters that control the learning process, like the number of trees in a Random Forest or the learning rate in a neural network. By testing different combinations of these parameters, we identify the ones that yield the best results for our specific data. This process helps in optimizing the model to achieve higher accuracy and better generalization to new data.

Confusion Matrix

A confusion matrix is a table that evaluates a classification model's performance by showing the counts of true positives, true negatives, false positives, and false negatives.

It helps derive metrics like accuracy, precision, recall, and F1 score for a more detailed performance assessment. This table provides insights into the types of errors made by the model and can help identify areas for improvement.

Components :

True Positives (TP): The number of instances correctly predicted as positive.

True Negatives (TN): The number of instances correctly predicted as negative.

False Positives (FP): The number of instances incorrectly predicted as positive (Type I error).

False Negatives (FN): The number of instances incorrectly predicted as negative (Type II error).

Logistic Regression

Logistic regression is a type of supervised machine learning algorithm used for binary classification problems, where the target variable is a binary outcome (0 or 1). It's an extension of linear regression, but instead of predicting a continuous value, it predicts the probability of the binary outcome.

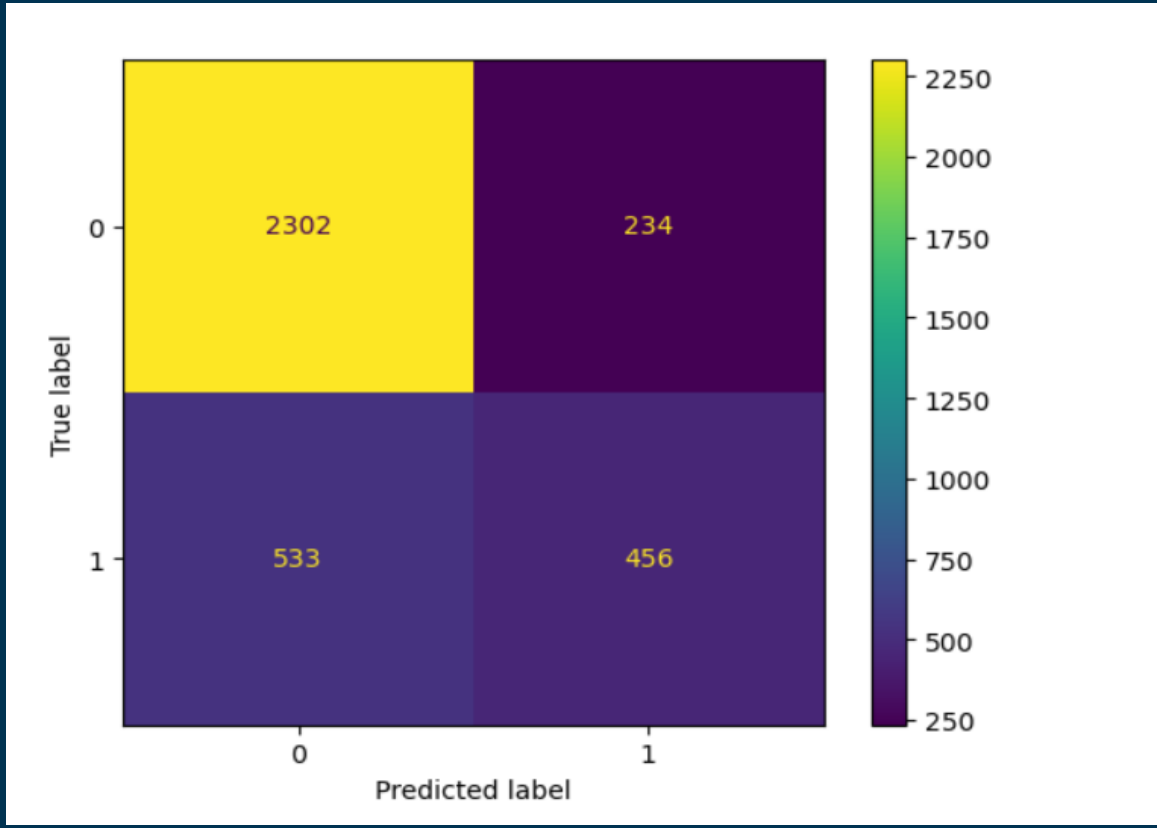
Logistic regression is crucial for predicting binary outcomes, offering interpretable results, probabilistic predictions, efficiency, and serving as a foundation for more complex models.

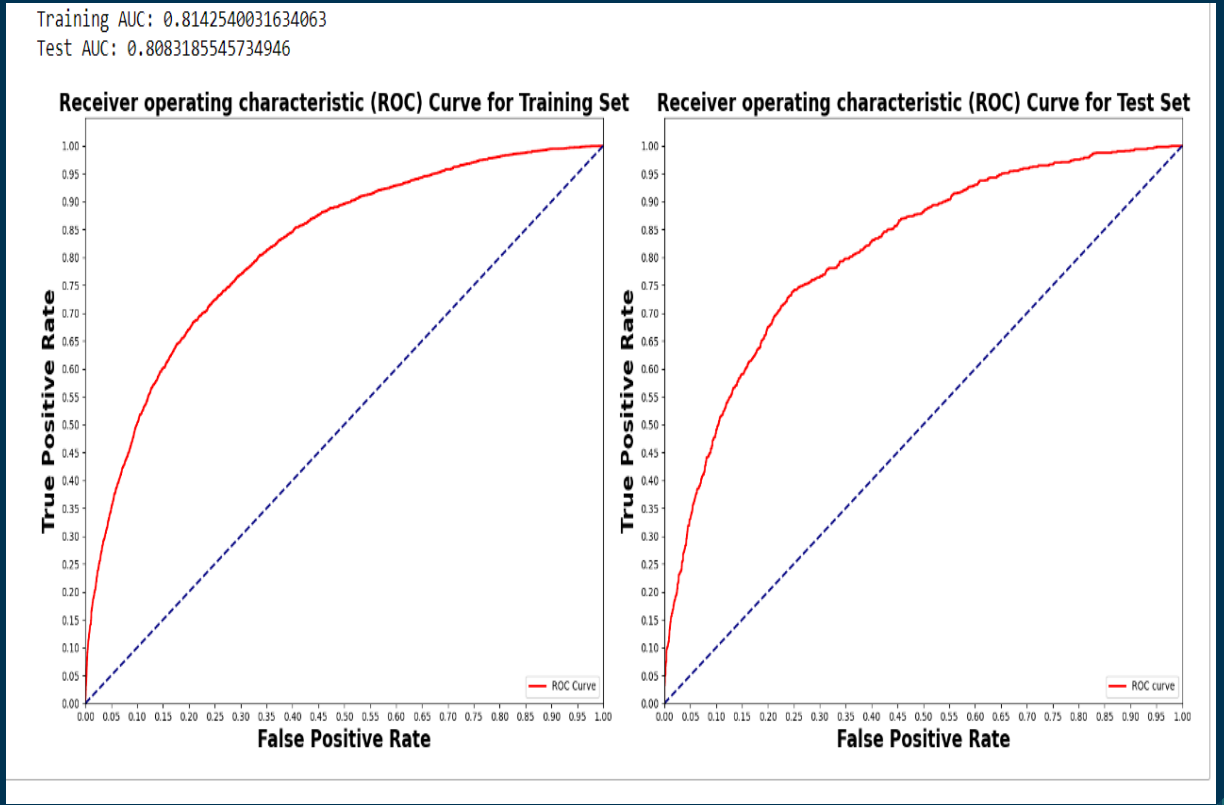
CODE SNIPPET

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the logistic regression model
logreg.fit(X_train, y_train)

# Generate predictions
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)

# Print the metrics
print_metrics(y_train, y_hat_train, y_test, y_hat_test)
```

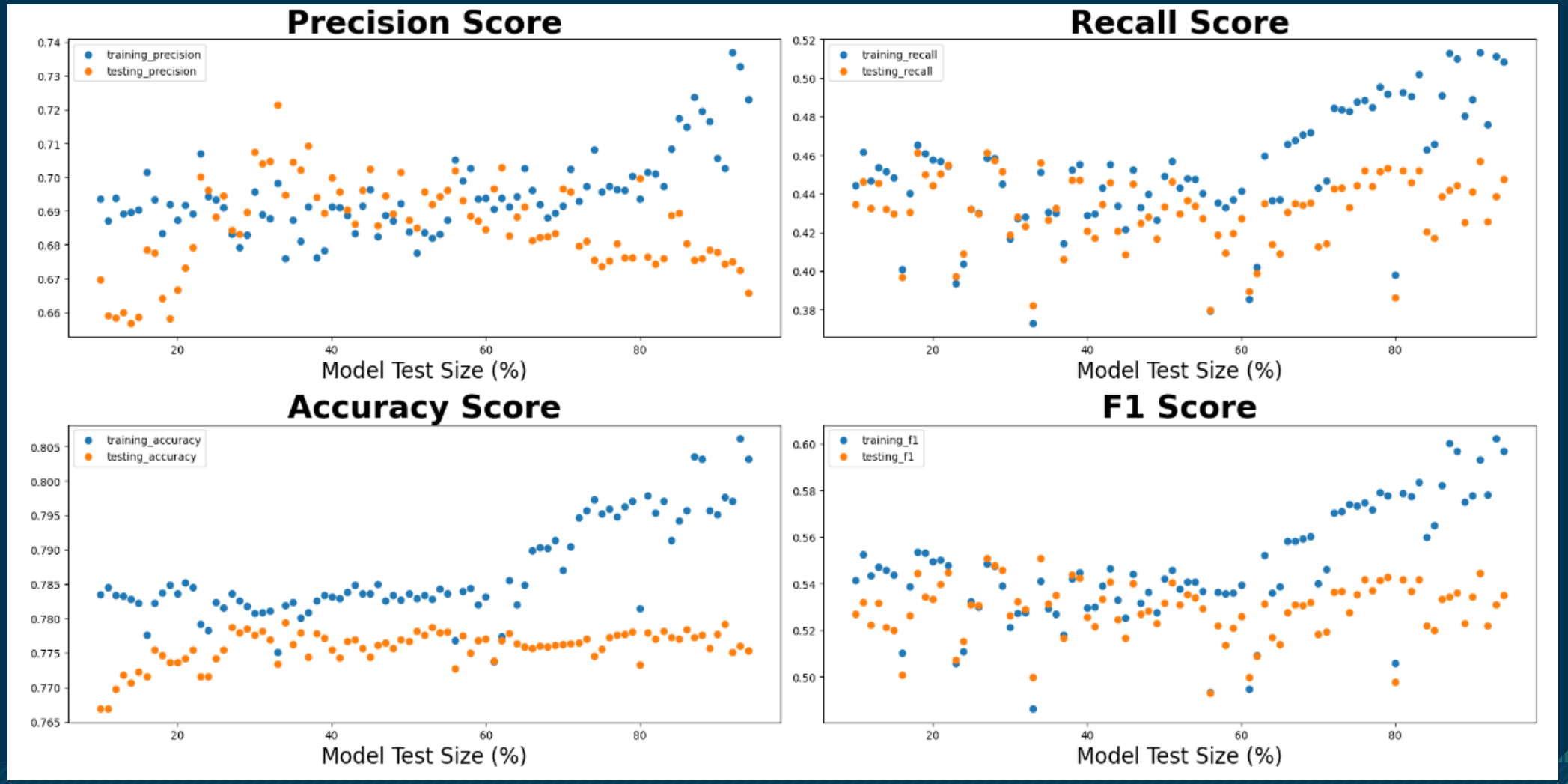
```
Classification Report for Training Data:
              precision    recall  f1-score   support

         0.0       0.80      0.92      0.86      9997
         1.0       0.69      0.45      0.55      4102

    accuracy                           0.78     14099
   macro avg       0.75      0.68      0.70     14099
weighted avg       0.77      0.78      0.77     14099


Classification Report for Testing Data:
              precision    recall  f1-score   support

         0.0       0.81      0.91      0.86      2536
         1.0       0.66      0.46      0.54       989

    accuracy                           0.78      3525
   macro avg       0.74      0.68      0.70      3525
```

Confusion Matrix and ROC curve





Metrics across different train-test splits



Decision Tree

Decision trees in machine learning are a supervised learning algorithm that enables developers to analyze the possible consequences of a decision and predict outcomes for future data. A decision tree is a tree-like model that starts at the root and branches out to demonstrate various outcomes.

Decision trees are crucial for their simplicity, ability to handle non-linear relationships, feature importance identification, and robustness to outliers, applicable to both classification and regression tasks.

CODE SNIPPET

```
# Assuming df is your preprocessed DataFrame
# Define the target variable and features
y = df['target'].values
X = df.drop('target', axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Instantiate a Decision Tree model
dtree = DecisionTreeClassifier(random_state=42)
```

```
Best parameters found:  {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
Classification Report for Training Data:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.89      9997
         1.0       0.78      0.58      0.67      4102

    accuracy                           0.83     14099
   macro avg       0.81      0.76      0.78     14099
weighted avg       0.83      0.83      0.82     14099


Classification Report for Testing Data:
              precision    recall  f1-score   support

         0.0       0.80      0.88      0.84      2536
         1.0       0.59      0.44      0.50       989

    accuracy                           0.76      3525
   macro avg       0.70      0.66      0.67      3525
weighted avg       0.74      0.76      0.75      3525
```

Random Forest

Random Forest is a supervised learning algorithm that combines multiple decision trees to improve the accuracy and robustness of predictions. It is a popular ensemble learning method widely used in classification and regression tasks.

Random Forests enhance accuracy and robustness through ensemble learning, resist overfitting, highlight feature importance, handle missing values, scale well with large datasets, and are versatile for both classification and regression.

CODE SNIPPET

```python
# Assuming df is your preprocessed DataFrame
# Define the target variable and features
y = df['target'].values
X = df.drop('target', axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Instantiate a Random Forest model
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

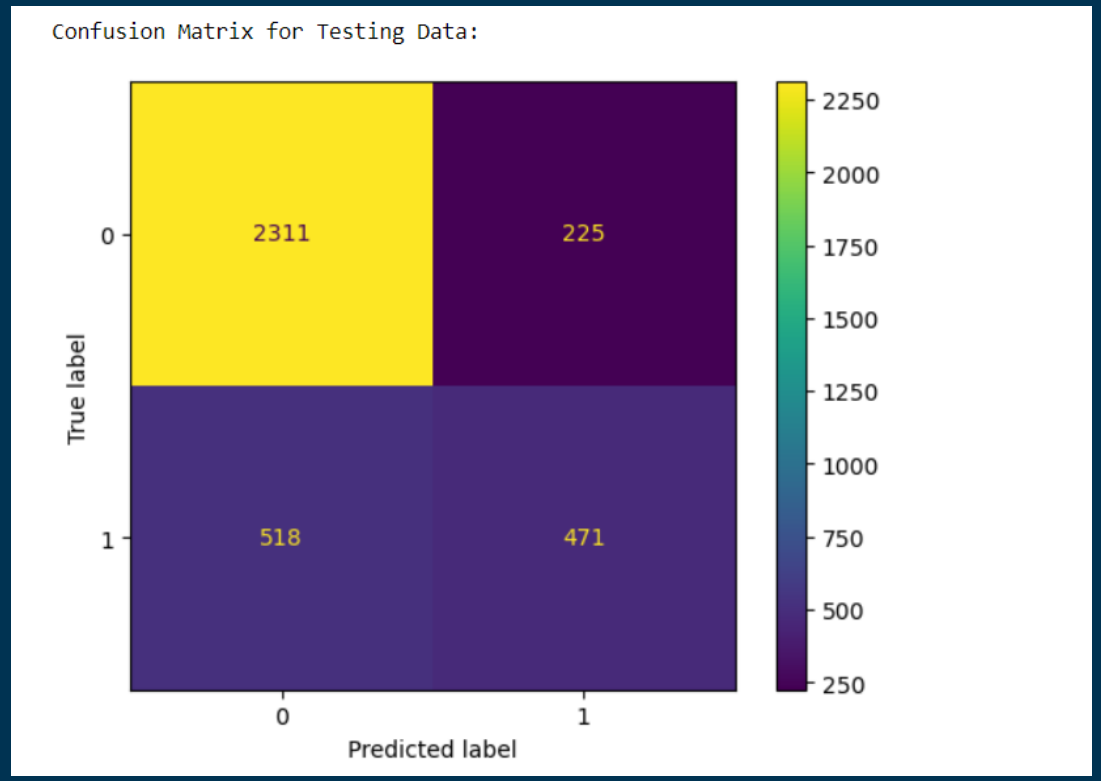
```
Classification Report for Training Data (Random Forest):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      9997
         1.0       1.00      1.00      1.00      4102

    accuracy                           1.00     14099
   macro avg       1.00      1.00      1.00     14099
weighted avg       1.00      1.00      1.00     14099

Classification Report for Testing Data (Random Forest):
              precision    recall  f1-score   support

         0.0       0.82      0.90      0.86      2536
         1.0       0.66      0.48      0.55       989

    accuracy                           0.78      3525
   macro avg       0.74      0.69      0.70      3525
weighted avg       0.77      0.78      0.77      3525

Random Forest Cross-validation scores: [0.77390071 0.7787234  0.79120567 0.78638298 0.77412032]
Random Forest Mean cross-validation score: 0.7808666167556211
```

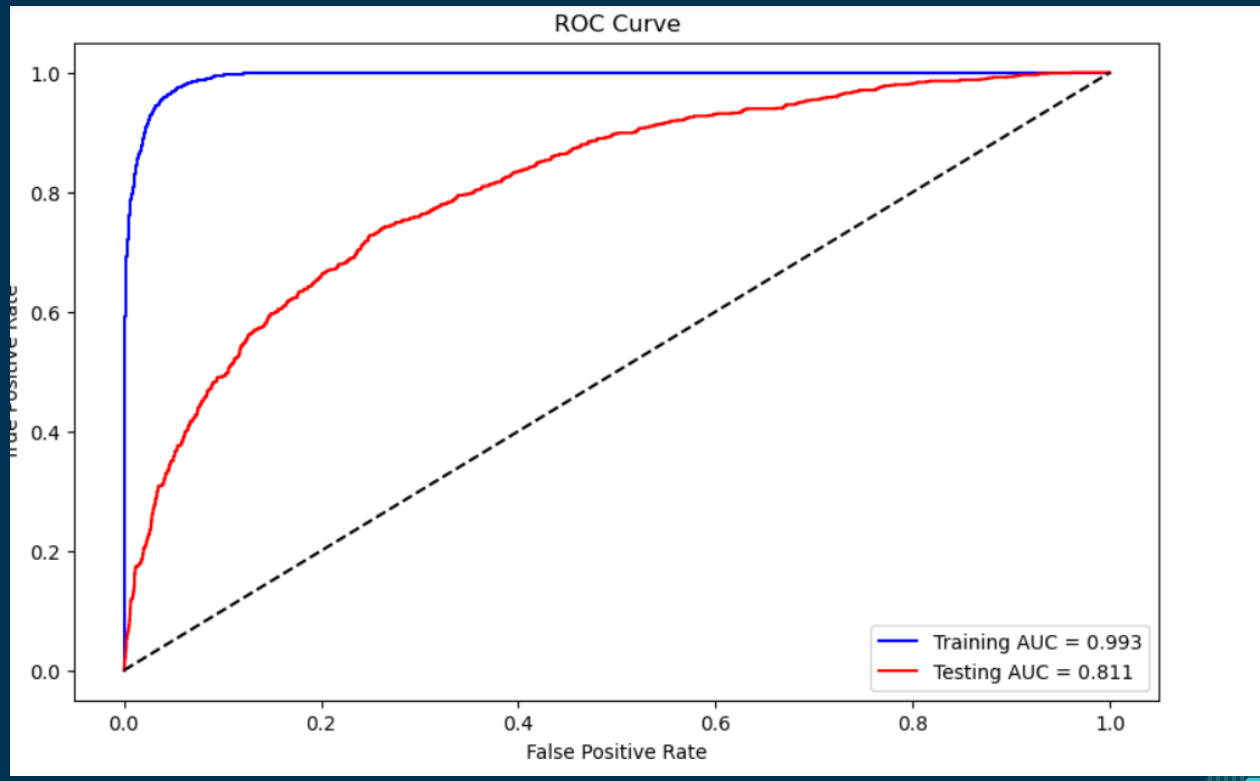
After hyperparameter tuning

```python
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Define a simpler parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100],  #  number of trees
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 4],
    'criterion': ['gini', 'entropy']
}

# Perform grid search with cross-validation
grid = GridSearchCV(rf, param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)

# Get the best parameters from grid search
best_params = grid.best_params_
print("Best parameters found: ", best_params)

# Train the Random Forest model with the best parameters
rf = RandomForestClassifier(**best_params, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

#n_jobs=allows the computation to run in parallel using all available processors, speeding up the process.

# Generate predictions
y_hat_train = rf.predict(X_train)
y_hat_test = rf.predict(X_test)
```

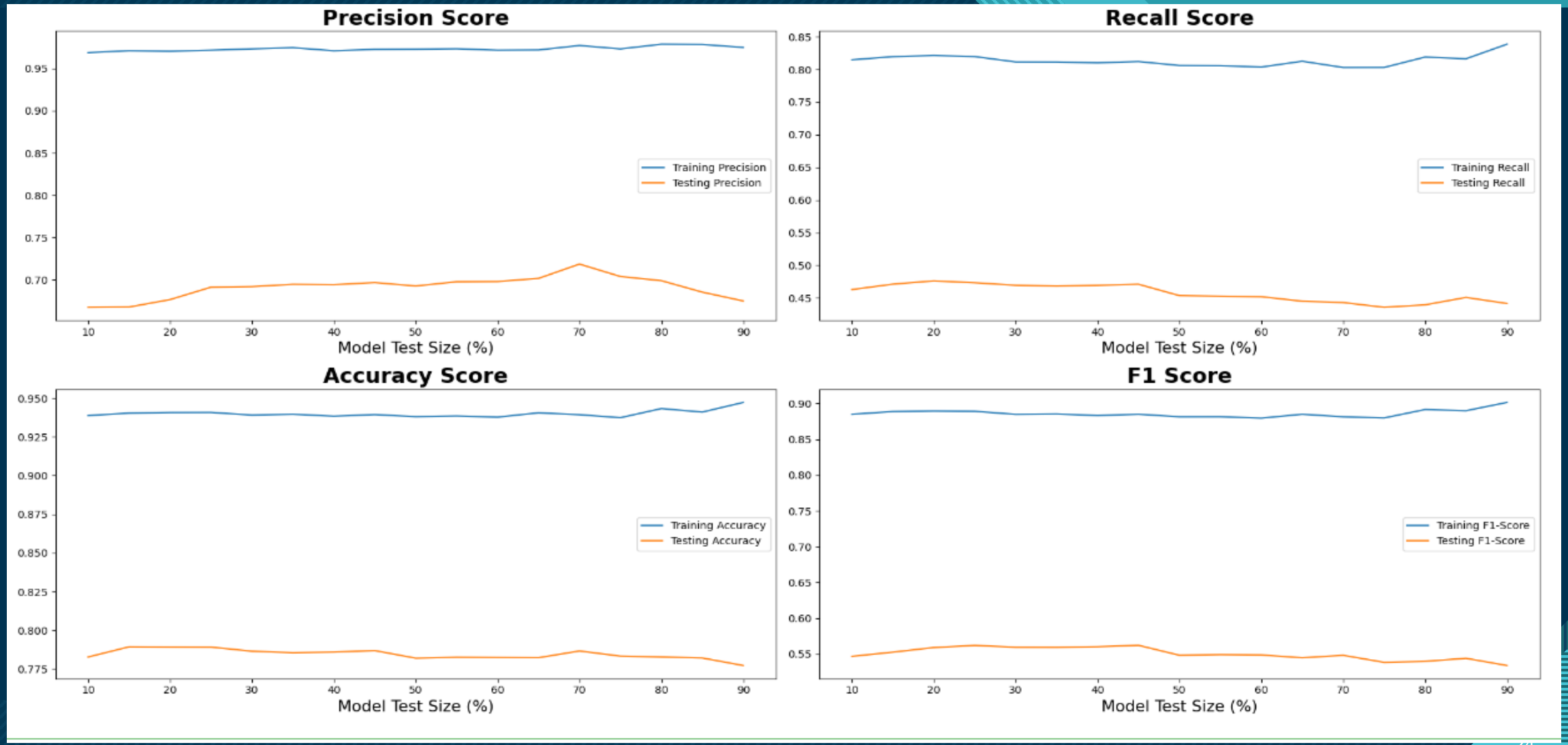
```
Best parameters found:  {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimato
rs': 100}
Classification Report for Training Data:
              precision    recall  f1-score   support

         0.0       0.93      0.99      0.96      9997
         1.0       0.97      0.82      0.89      4102

    accuracy                           0.94     14099
   macro avg       0.95      0.91      0.92     14099
weighted avg       0.94      0.94      0.94     14099


Classification Report for Testing Data:
              precision    recall  f1-score   support

         0.0       0.82      0.91      0.86      2536
         1.0       0.68      0.48      0.56       989

    accuracy                           0.79      3525
   macro avg       0.75      0.69      0.71      3525
weighted avg       0.78      0.79      0.78      3525
```

Confusion Matrix and ROC Curve





Metrics across different train-test splits



Why Random Forest ?

1. **Accuracy and Robustness**: By combining multiple decision trees, Random Forest improves the accuracy and robustness of predictions compared to a single decision tree.
2. **Resistant to Overfitting**: The ensemble approach helps mitigate the risk of overfitting, making the model generalize better to unseen data.
3. **Feature Importance**: It provides insights into the importance of each feature, aiding in feature selection and understanding the model.
4. **Handles Missing Values**: Random Forest can handle missing values in the dataset effectively.
5. **Scalability**: It scales well with large datasets and can handle high-dimensional data efficiently.
6. **Versatility**: Suitable for both classification and regression tasks, making it a versatile tool in various applications.

Metrics Difference :

Parameter	Logistic Regression	Decision Tree	Random Forest
Training Accuracy	0.78	0.83	0.94
Testing Accuracy	0.78	0.76	0.79
Precision	(0) 0.81 (1) 0.66	(0) 0.80 (1) 0.55	(0) 0.82 (1) 0.68
Recall	(0) 0.91 (1) 0.46	(0) 0.88 (1) 0.44	(0) 0.91 (1) 0.48

Challenges

- **Data Quality and Integration:** Telecom companies often have vast amounts of data stored across multiple systems, including customer demographics, call logs, usage patterns, and billing information. Integrating and cleansing this data for analysis can be challenging, and poor data quality can lead to inaccurate predictions.

- **Imbalanced Data:** Telecom datasets are often imbalanced, with a small percentage of customers churning compared to those who remain. Imbalanced data can bias predictive models and lead to inaccurate churn predictions.

- **Dynamic Customer Behavior:** Customer behavior and preferences evolve over time, making it challenging to build accurate predictive models that adapt to changing patterns and trends.

- **Scalability:** Telecom companies serve large customer bases, and churn prediction systems must be scalable to handle massive volumes of data and real-time predictions.

- **Real-Time Prediction:** Implementing real-time churn prediction systems that can provide timely interventions to prevent churn is challenging due to the need for fast data processing and decision-making.

Conclusion :

- Improved Customer Retention: By accurately predicting which customers are likely to churn, telecom companies can implement targeted strategies to retain valuable customers, ensuring their satisfaction and loyalty.
- Cost Reduction: Preventing churn is more cost-effective than acquiring new customers. Churn prediction allows for efficient resource allocation and reduces costs associated with customer attrition.
- Enhanced Customer Experience: Understanding customer behavior and preferences enables telecom providers to offer personalized experiences, address concerns proactively, and significantly improve overall satisfaction.
- Increased Revenue: Retaining existing customers and maximizing their lifetime value leads to steady revenue streams and sustainable business growth.
- Competitive Advantage: Effective churn prediction and management provide telecom companies with a competitive edge through superior service, tailored offers, and proactive retention initiatives.

Thank You