Predictive Analytics

“Predictive Analytics to Increase Telecom Customer Retention”

Seneca Polytechnic

Course code:SEA600

Aeraf Mohammed Khan , Madhur Saluja, Tasbi Tasbi

Supervised by: Vida Movahedi

Contents

SEA600- Assignment 1 2

1. **Introduction** 3
2. **Data Exploration and Preprocessing** 4
3. **DataVisualization** 5
4. **Baseline Model** 9
   - **4.1 Before Smote** 9
   - **4.2 After Smote** 10
5. **Baseline Model Training** 12
6. **Advance Models** 13
   - 6.1 AUROC graph 15
   - 6.2 Precision-Recall graph 16
7. **Feature engineering** 17
   - 7.1 feature creation 17
   - 7.2 feature encoder 18
   - 7.3 feature scaling 19
8. **Hyperparameter Tuning** 21
9. **Testing on test Data** 23

INTRODUCTION

Predicting Customer Churn in Telecom: A Machine Learning Approach

The objective of our project is to train a model that can predict whether customers of the telco company will churn or not. This is crucial for businesses as it helps in identifying and retaining at-risk customers, ultimately reducing customer turnover. In this report, we outline our approach to transforming this objective into a machine learning (ML) problem, focusing on the implementation constraints and considerations that arise during the process.

Our approach involves training several models, including Logistic Regression, Decision Tree, and Random Forest, to compare their effectiveness in predicting customer churn. The dataset used for this analysis has been split as 80% training and 20% testing to ensure a robust evaluation of the models' performance. This report outlines the steps taken to transform our objective into a machine-learning problem, focusing on the implementation constraints and considerations encountered during the process. We discuss the binary classification nature of the problem, the data preprocessing steps undertaken, and the feature engineering techniques employed to enhance model performance.

We discuss the binary classification nature of the problem, the data preprocessing steps undertaken, and the feature engineering techniques employed to enhance model performance. As we delve into the models' training and evaluation, we emphasize the importance of precision, recall, F1 score, and AUROC metrics in assessing their predictive capabilities. Our analysis also extends to advanced models, exploring the potential of Random Forest and other sophisticated algorithms in improving churn prediction accuracy.

By comparing simple baseline methods with advanced models, and applying feature engineering and hyperparameter tuning, we aim to identify the most suitable model for deployment. This document presents our methodologies, analyses, and findings, offering insights into the predictive capabilities of each model

Data Exploration and Preprocessing

Data preparation is a crucial first step in the data analysis and machine learning pipeline, aimed at transforming raw data into a clean, accurate format suitable for insightful analysis. This process involves several key tasks such as cleaning, which removes or corrects inaccurate, incomplete, or irrelevant parts of the data. we have done the data preprocessing, we imported the raw data set here, dropped some columns and altered some data types for further useage

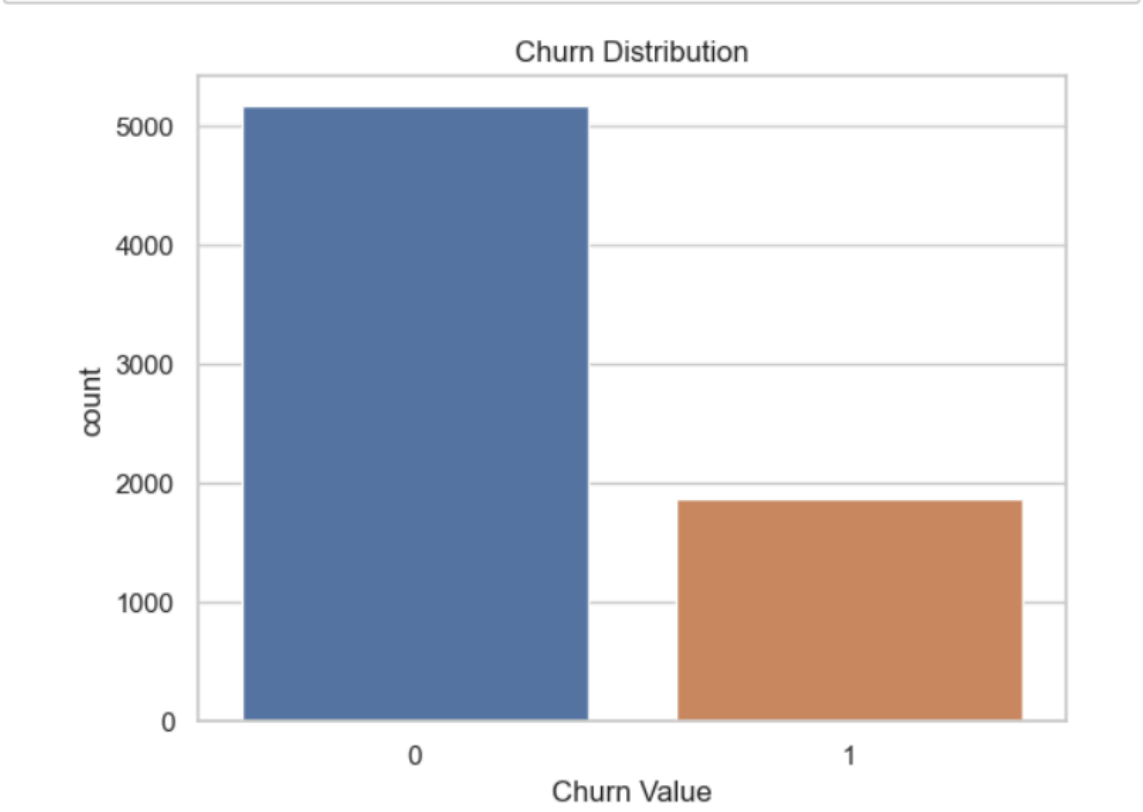
We dropped some columns which we decided should not be used and removed location details to avoid models getting biased for the people of one region. However, we didn't had the data to represent the whole population. But we decided to still remove the location columns. count and customerID couldn't be used by a machine learning model. Lasly all the other Target variable related columns were dropped as well. Initially, The only data type we changed from the actual dataset is the total charges data type from object to float for better understanding . Furthermore, we have attached the features and target variable table (min, max, mean, std. dev.) in detail in the data visualization document.

#	Column	Non-Null Count	Dtype
0	Gender	7043 non-null	object
1	Senior Citizen	7043 non-null	object
2	Partner	7043 non-null	object
3	Tenure Months	7043 non-null	int64
4	Phone Service	7043 non-null	object
5	Multiple Lines	7043 non-null	object
6	Internet Service	7043 non-null	object
7	Online Security	7043 non-null	object
8	Online Backup	7043 non-null	object
9	Device Protection	7043 non-null	object
10	Tech Support	7043 non-null	object
11	Streaming TV	7043 non-null	object
12	Streaming Movies	7043 non-null	object
13	Contract	7043 non-null	object
14	Paperless Billing	7043 non-null	object
15	Payment Method	7043 non-null	object
16	Monthly Charges	7043 non-null	float64
17	Total Charges	7032 non-null	float64
18	Churn Value	7043 non-null	int64

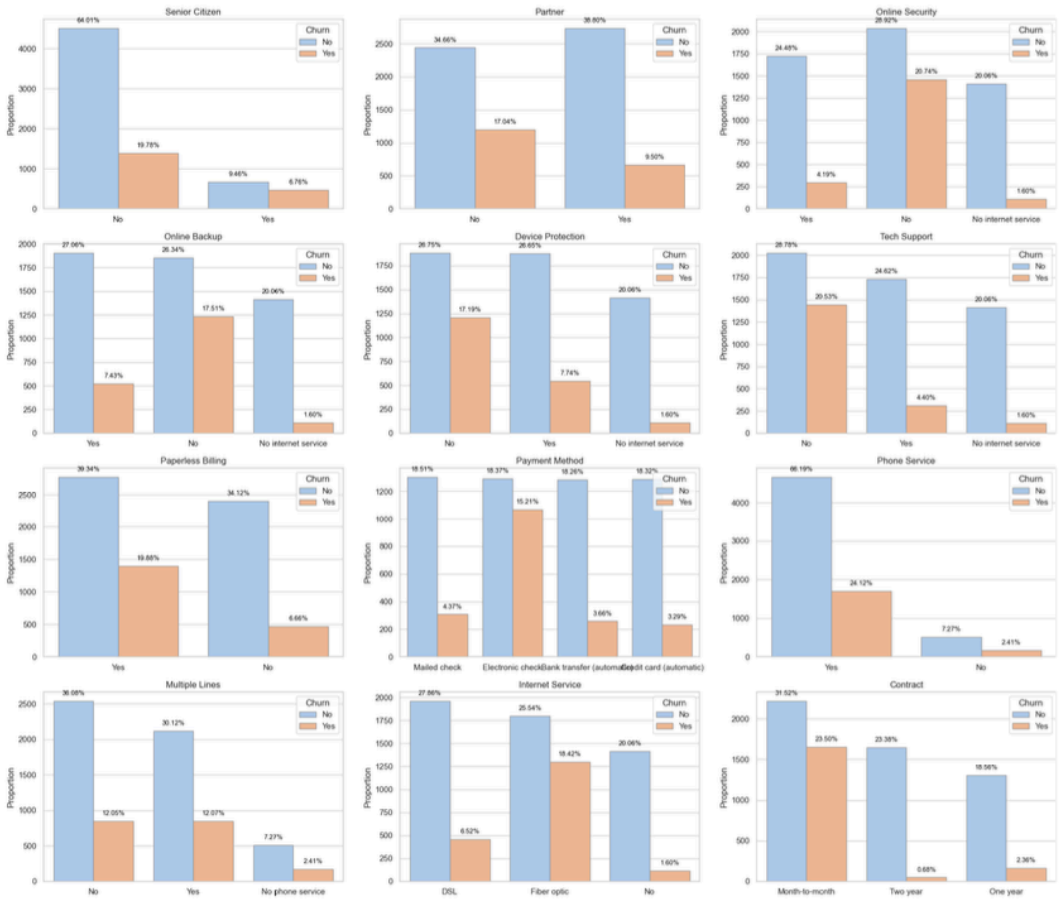
The features are all the customer-related information, such as gender, senior citizen status, partner, etc., that you will utilize to estimate the probability of churn. 'Churn Value,' the target variable, indicates if the client has churned or not.

Data Visualization

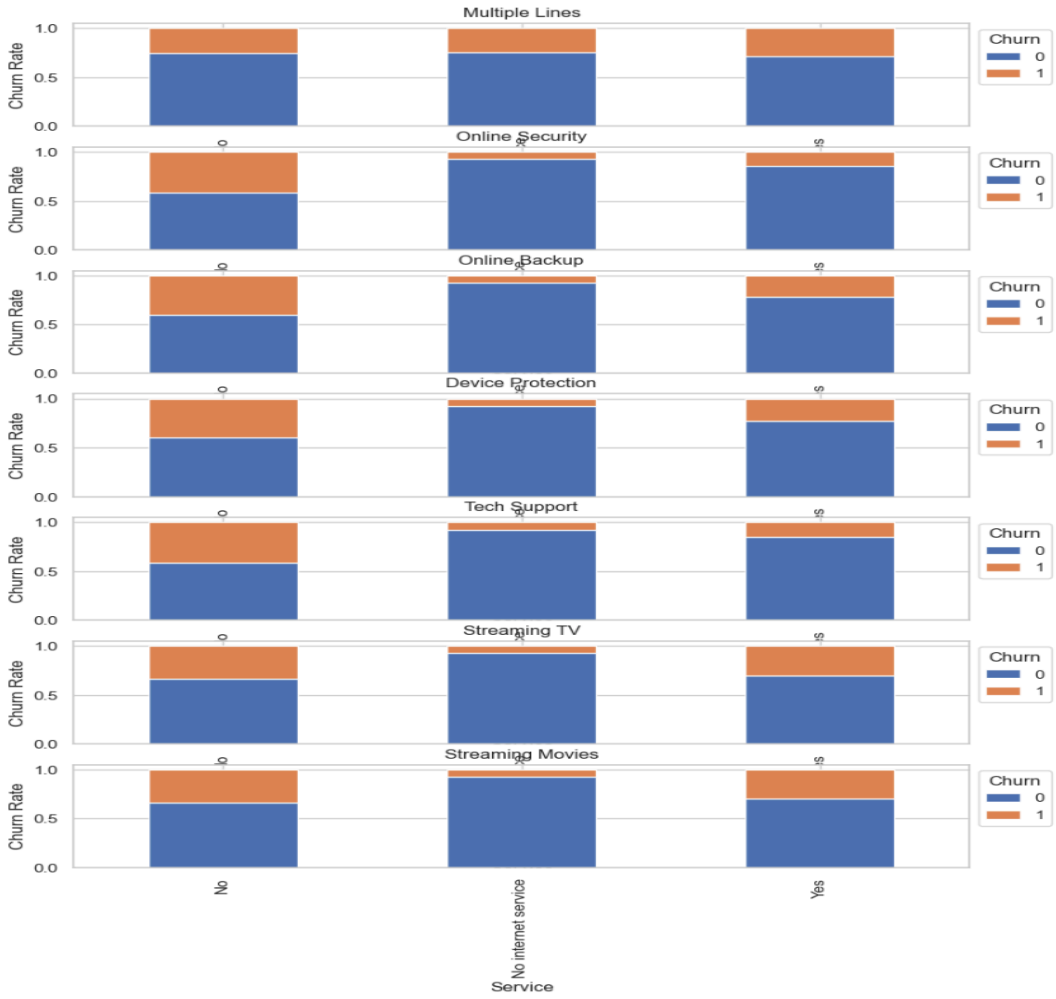
In data visualization, first we loaded the dataset and used '.head()', '.info()' and '.describe()' to get an initial understanding of the data structure, types and potential categorical variables. We identified and listed unique categories within each feature for better understanding. Then we used ' .describe()' to obtain the statistical summary for numerical insights, which will help in highlighting initial patterns or anomalies.



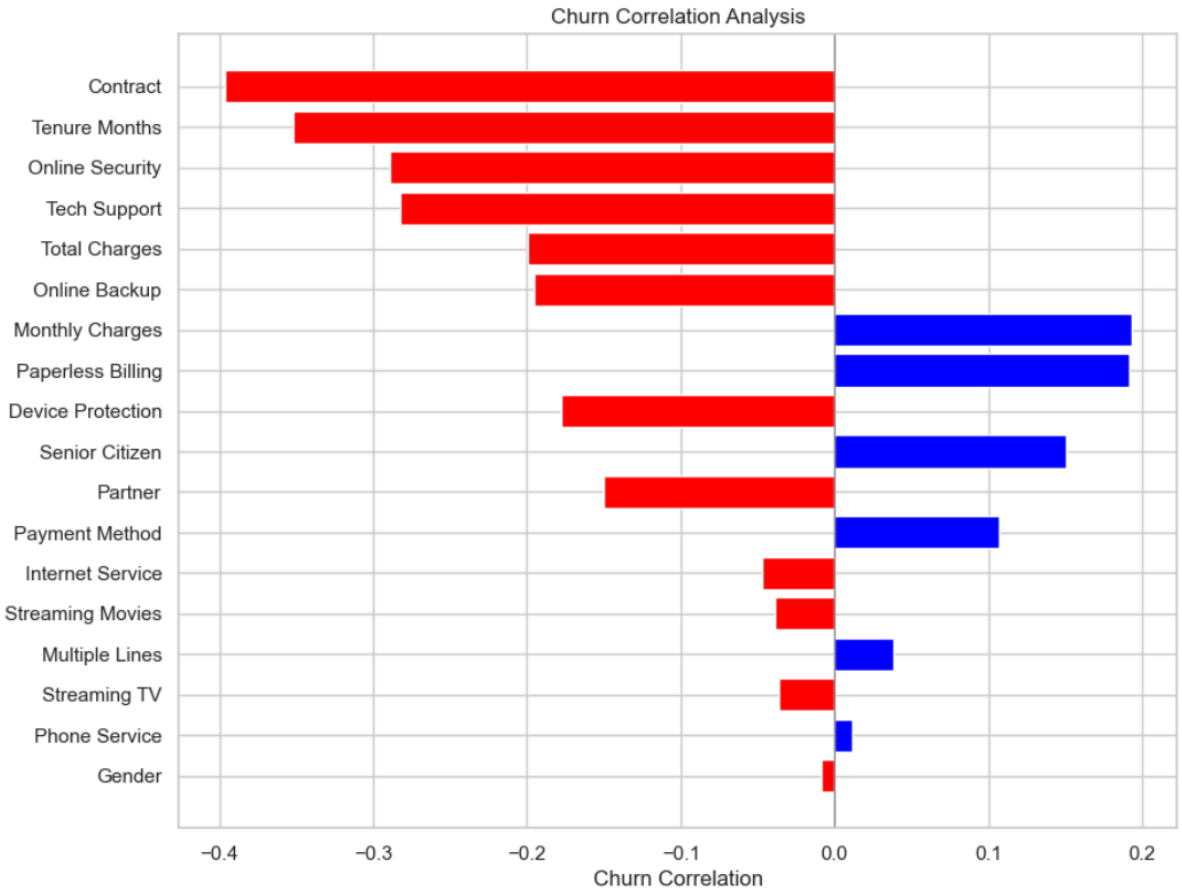
This bar chart illustrates the distribution of churn. The significant imbalance between the retained and churned customer is crucial to consider when selecting performance metrics for model evalua- tion. Models like Random Forest Classifier and Decision Tree are insensitive to such imbalances. We can use techniques like SMOTE for data balancing before training more sensitive models such as Logistic Regression or SVM.



The churn rate of customers is influenced by various factors, including age, gender, and payment method. Almost half od the senior citizen can be seen to be churned, suggesting the need for more effort to retain senior citizen by providing more comfort. Customers with partners are slightly less likely to churn, suggesting services that appeal to couples might keep them loyal. Online security services significantly decrease churn, highlighting the importance of security features. Tech support also significantly reduces churn, underlining the importance of customer service. Payment methods also impact churn rates, especially electronic check, they seem to be causing comparatively large effect on customer churn. The set of plots above give tons of such insights, to make it more easily readable we plotted a churn correlation plot in the end which shows how much a particular column as a whole effects churn.



The plot shows above have some of its label hidden, The ones in the middle are “No phone service” 8 and “No Internet service”. The columns which are plotted had 3 categories in them, multiple lines had “No phone service” value in it if the Phone service column had “No” in that row. As for the rest of the services they had a categorical value “No internet service” in the row where theInternet Service column had the value “No”. Here we just plotted all those categories and the effect they had on churn to decide if any can be further worked upon during feature engineering to optime our models performance.



The horizontal bar chart presented, shows the correlation of various features with Churn Value. Features with positive correlations are colored blue, they suggest a positive correlation and a tendency to increase churn possibility (simply leading to customer leaving Telco company). On the other hand, features with negative correlations are colored red and are suggesting a potential retention factors (it can be noted that most o fthe columns are leading to retention and thus our data set is imbalanced). The graph is sorted to prioritize features with stronger correlations on the top and weaker on the bottom. This helps in feature selection and provids insights into areas that may require more focused customer retention strategies to further decrease customer churn.

Baseline Model

As a basic point of comparison for assessing the effectiveness of increasingly complex predictive models, we present a baseline model in this work.In order to evaluate how data imbalance affects model performance, we use the Synthetic Minority Over-sampling Technique (SMOTE) to generate a dataset that is more evenly distributed. To understand how data balance affects prediction, we examine the baseline model's performance both before and after applying SMOTE.

Before SMOTE,

Upon analyzing the performance of both Logistic Regression and Decision Tree classifiers, we observe the following before applying SMOTE:

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       845
           1       0.64      0.56      0.60       282

    accuracy                           0.81      1127
   macro avg       0.75      0.73      0.74      1127
weighted avg       0.81      0.81      0.81      1127


Decision Tree
              precision    recall  f1-score   support

           0       0.85      0.82      0.84       845
           1       0.51      0.55      0.53       282

    accuracy                           0.76      1127
   macro avg       0.68      0.69      0.68      1127
weighted avg       0.76      0.76      0.76      1127
```

- Precision: When it comes to predicting a positive or negative class, logistic regression is more dependable because its precision is higher for both classes. In situations when the cost of false positives is significant, this is essential.
- Recall is another area where Logistic Regression performs better than the Decision Tree model. For the majority class (class 0), recall is much greater, and for the minority class (class 1), it is modestly higher. This implies that Logistic Regression has a higher ability to find all examples that are pertinent for both groups.

- The F1-Score measures the balance between recall and precision by combining the two into a single score. For both classes, Logistic Regression yields a higher F1-score, suggesting a better overall balance between recall and precision.
- Support: Since the two models were evaluated using the same dataset, their support metrics are the same. This has no bearing on the comparison of model performance.

After Smote

After applying smote, we analyzed the performance of both Logistic Regression and Decision Tree classifiers, we observed the following:

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.88      0.78      0.82       828
           1       0.53      0.71      0.61       299

    accuracy                           0.76      1127
   macro avg       0.71      0.74      0.72      1127
weighted avg       0.79      0.76      0.77      1127


Decision Tree:
              precision    recall  f1-score   support

           0       0.84      0.81      0.82       828
           1       0.52      0.57      0.55       299

    accuracy                           0.75      1127
   macro avg       0.68      0.69      0.69      1127
weighted avg       0.76      0.75      0.75      1127
```

- Precision: The better accuracy of the Logistic Regression model for both the positive (class 1) and negative (class 0) classes indicates that it is more dependable in making predictions. This feature is especially crucial in situations when there is a significant risk of a false positive, or the incorrect identification of an occurrence as positive when it is not.

- Recall: For both classes, Logistic Regression outperforms Decision Trees in recall. It shows a markedly higher recall for the majority class, meaning that 91% of real negative situations are properly identified. Even in the minority class, where fewer occurrences frequently provide challenges to precision, Logistic Regression performs somewhat

better. This implies that Logistic Regression is a better method for locating and accurately categorizing all relevant examples.

- F1-Score: For all classes, Logistic Regression yields a higher F1-score, which is a statistic that combines precision and recall. The higher F1-scores show a better trade-off between recall and precision, suggesting that Logistic Regression is more efficient overall.

- Support: Since all models are tested using the same dataset, the support metric measures how many instances of each class there are in reality. As a result, support has little bearing on how well models perform in comparison.

The Logistic Regression model showed improved prediction performance by following the use of smote. The model demonstrated a significant improvement in recall for the minority class (1), which is essential for a balanced evaluation of an unbalanced dataset, while maintaining high precision, lowering the possibility of false positives. The model's improved precision and recall balance is further supported by the higher F1-scores for both classes following smote. These enhancements show that the Logistic Regression model performed better when smote was used to rectify the class imbalance, which makes it the better option for our dataset.

Baseline Model Training

This comparison sheds light on the value and efficacy of SMOTE in improving model performance. We will use SMOTE for the dataset

.

Decision trees and logistic regression were selected as baseline models due to their unique benefits. Because of its effectiveness and ability to provide a probabilistic interpretation, logistic regression is preferred when dealing with binary outcomes such as churn prediction. When there is a linear relationship between the independent variables and the dependent variable's log odds, the linear model performs well.

On the other hand, decision trees are non-linear models that do not require any modification in order to represent intricate relationships between features. Additionally, they are intuitively clear, and the ability to view the decision-making process aids in understanding how the model makes decisions.

In actuality, logistic regression and decision trees can beat other basic models like Naive Bayes, despite the fact that they are rapid and efficient in some situations and frequently presume feature independence. Furthermore, models such as k-Nearest Neighbours (k-NN) are less ideal as straightforward, fast-running baselines because they require data scaling and can be computationally demanding. Therefore, in the first stage, decision trees and logistic regression offer a nice mix of computational efficiency, interpretability, and complexity.

In conclusion, the Logistic Regression model performs better than the Decision Tree on every metric that was assessed. the Logistic Regression model outclasses the Decision Tree in all significant metrics. Logistic Regression demonstrates higher precision (0.88 for class 0 and 0.53 for class 1), compared to the Decision Tree (0.84 for class 0 and 0.52 for class 1). In terms of recall, Logistic Regression again leads (0.78 for class 0 and 0.71 for class 1), which is particularly notable against the Decision Tree's performance (0.81 for class 0 and 0.57 for class 1). F1-scores, which combine precision and recall into one metric, are also higher with Logistic Regression, scoring 0.82 for class 0 and 0.61 for class 1, versus the Decision Tree's 0.82 for class 0 and 0.55 for class 1. Moreover, Logistic Regression achieves an overall accuracy of 0.76, surpassing the Decision Tree's 0.75.These results unequivocally show that the Logistic Regression model, which produces more accurate and dependable predictions, is a superior fit for the dataset and the problem of customer churn prediction.

We also take into account the support for each class, which shows how many real examples there are of each class in the dataset. The class distribution has probably been modified by SMOTE, but even with this modification, the Logistic Regression model seems to be able to handle the synthetic minority class better than the Decision Tree model.

Milestone -1

Advanced Models

In our study, we trained two advanced models, Random Forest and SVM, alongside Logistic Regression and Decision Tree to enhance predictive accuracy. To address potential imbalances in our training dataset, we employed the Synthetic Minority Over-sampling Technique (SMOTE) to assess whether balancing leads to improved performance compared to the baseline models. Additionally, we experimented with the Naive Bayes model for its simplicity, computational efficiency, and probabilistic predictions, which are valuable for decision-making processes. The performance metrics of all trained models were generated using the classification_report() function to facilitate a comprehensive comparison. Our findings suggest that balanced training data generally yield better performance metrics than the imbalanced training dataset used for the baseline models. This led to the strategic decision to balance the training dataset after feature engineering, which was creatively executed to enhance performance metrics further.

```
data shapes:
X_train shape: (5634, 18)
X_test_final shape: (1409, 18)
y_train shape: (5634,)
y_test_final shape: (1409,)
X_train_val shape: (4507, 18)
X_test_validation shape: (1127, 18)
y_train_val shape: (4507,)
y_test_validation shape: (1127,)
```

Similar to how we had done for the baseline model, we split our data into 3 parts. First we divided the whole data set into 80/20, kept the 20 percent data aside for final testing. The 80 percent data was then used to get another 20% data from within it as validation set.

The only difference is that after getting the train, test and validation set, we balanced the train data before giving it to our model.

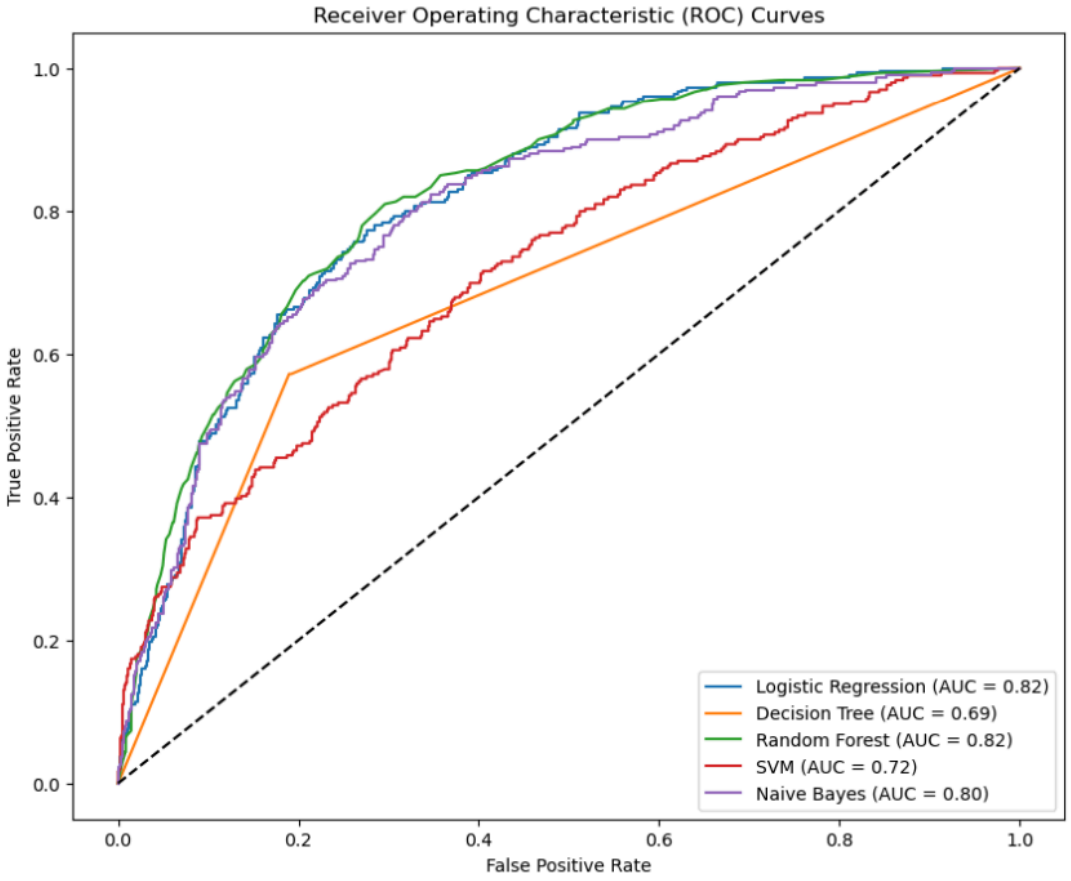
```
Random Forest:
              precision    recall  f1-score   support

           0       0.85      0.85      0.85       828
           1       0.58      0.59      0.58       299

    accuracy                           0.78      1127
   macro avg       0.72      0.72      0.72      1127
weighted avg       0.78      0.78      0.78      1127


SVM:
              precision    recall  f1-score   support

           0       0.82      0.73      0.77       828
           1       0.43      0.57      0.49       299

    accuracy                           0.69      1127
   macro avg       0.63      0.65      0.63      1127
weighted avg       0.72      0.69      0.70      1127
```

we have found out that SMOTE is giving us a boost in recall with a high precision for class 0, we decided to use the balanced train data to train random forest and SVM.We can note that the weighted average recall of random forest is the best so far for all the models.

As said above, we also tried using Naive bayes but the reports were not that great so we dropped the idea of using it.

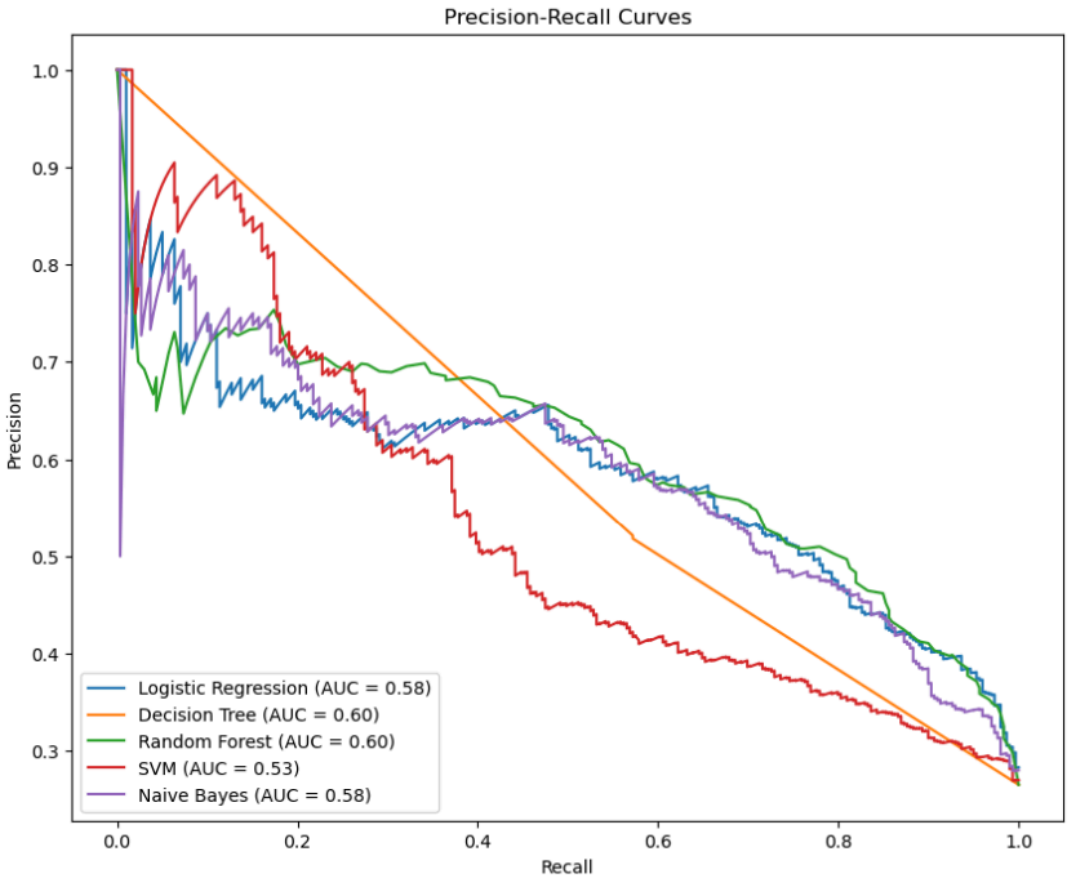
```
Naive Bayes:
              precision    recall  f1-score   support

           0       0.88      0.75      0.81       828
           1       0.51      0.71      0.59       299

    accuracy                           0.74      1127
   macro avg       0.69      0.73      0.70      1127
weighted avg       0.78      0.74      0.75      1127
```

AUROC Graph



So, the above graph is AUROC graph for all the model that we decided and AUC, or the Area Under the ROC Curve, offers a unified way to assess a classifier's overall effectiveness and makes comparing models possible.The performance of different models is shown by the ROC curve in our graph. The best AUC scores, which indicate good discriminating abilities, are displayed by Random Forest and Logistic Regression. The lowest AUC is shown by the decision tree, indicating a restricted capacity for classification. Fair predictive performance is indicated by the modest AUC values of SVM and naive bayes

Precision - Recall graph



Based on the area under the curve (AUC) measure, the Precision-Recall graph shows that Random Forest and Decision Tree models perform somewhat better than Logistic Regression and Naive Bayes, while SVM trails behind. When assessing the practicality of these models for circumstances where real validation is crucial, it is important to consider the trade-off involved in detecting positive instances in the context of all positive predictions made.

Feature Engineering

While doing the feature engineering first we created a new notebook took the database which we got after cleaning in the initial stages and then on the whole dataset(including training and testing set) we did :

-feature creation:

[5]:
```
# creating a new column 'Streaming' instead of these 2 columns: 'Streaming TV',␣
 ↪'Streaming Movies'
X['Streaming'] = X['Streaming TV'] + X['Streaming Movies']

# Replace values with new labels based on whether the customer has no, one, or␣
 ↪both Streaming Services
X['Streaming'] = X['Streaming'].replace({'NoNo': 'No Streaming',
                                         'YesNo': 'Only TV',
                                         'NoYes': 'Only Movies',
                                         'YesYes': 'Both',
                                         'No internet serviceNo internet␣
 ↪service': 'No Streaming'})

# Drop the original "StreamingTV" and "StreamingMovies" features
X = X.drop(['Streaming TV', 'Streaming Movies'], axis=1)

X.head(31)
```

We combined related functions into more composite ones, such as combining "Phone Service" and "Multiple Lines" into a "Lines" feature, and "Streaming TV" and "Streaming Movies" into a single "Streaming" feature. These metrics seek to simplify the data while encapsulating its substance.

Feature encoder

```
[7]: # One hot Encoding the categorical Values
from sklearn.preprocessing import OneHotEncoder

columns_to_encode = ['Senior Citizen', 'Partner', 'Online Security',
                     'Online Backup', 'Device Protection', 'Tech Support',␣
 ↪'Contract',
                     'Paperless Billing', 'Payment Method', 'Streaming',␣
 ↪'Lines']


# Create a OneHotEncoder object
ohe = OneHotEncoder()

# Fit and transform the columns using the OneHotEncoder
encoded_columns = ohe.fit_transform(X[columns_to_encode])

# Create a new DataFrame with the encoded columns
encoded_df = pd.DataFrame(encoded_columns.toarray(), columns=ohe.
 ↪get_feature_names_out(columns_to_encode))

# Drop the original columns from the original DataFrame
```

```
X.drop(columns_to_encode, axis=1, inplace=True)

# Concatenate the original DataFrame with the new encoded DataFrame
X = pd.concat([X, encoded_df], axis=1)
```

Using one-hot encoding, we overcame the problem of categorical variables during our data preprocessing stage. This strategy is important because, while machine learning algorithms usually demand numerical input, improperly encoded categorical variables might cause issues. We converted these category variables into a binary matrix using the OneHotEncoder module from the sklearn.preprocessing library. By guaranteeing that every category is represented by a distinct set of zeros and ones, this encoding technique makes it possible for machine learning algorithms to interpret and make use of the data without imposing arbitrary ordinality.

- feature scaling

```
[9]: # Scaling the columns with numeric data
     from sklearn.preprocessing import MinMaxScaler

     scaler = MinMaxScaler()

     X[['Monthly Charges', 'Total Charges', 'Tenure Months']] = scaler.
      ↪fit_transform(X[['Monthly Charges', 'Total Charges', 'Tenure Months']])

     X.head()
```

Since numerical features can vary widely in magnitude, unscaled numerical inputs can skew the model's attention, with larger-scale features unjustifiably dominating the learning process. To get around this, we normalized our numerical input into a consistent range, usually [0, 1], using the MinMaxScaler from the same sklearn.preprocessing module. By guaranteeing that every feature contributes equally to the final prediction, this normalization helps gradient descent convergence and enhances both the training and model performance.

After doing the feature engineering we again split the dataset and then again trained the models to see compare which one is best so far. and after. We have attached the results down:

```
SVM:
              precision    recall  f1-score   support

                                                  22

--------------------------------------------------------

           0       0.83      0.77      0.80       843
           1       0.78      0.84      0.81       823

    accuracy                           0.80      1666
   macro avg       0.81      0.81      0.80      1666
weighted avg       0.81      0.80      0.80      1666
```

Logistic Regression

	precision	recall	f1-score	support
0	0.81	0.74	0.78	843
1	0.76	0.82	0.79	823
accuracy			0.78	1666
macro avg	0.78	0.78	0.78	1666
weighted avg	0.78	0.78	0.78	1666

Decision Tree

	precision	recall	f1-score	support
0	0.80	0.80	0.80	843
1	0.80	0.79	0.79	823
accuracy			0.80	1666
macro avg	0.80	0.80	0.80	1666
weighted avg	0.80	0.80	0.80	1666

Random Forest:

	precision	recall	f1-score	support
0	0.86	0.84	0.85	843
1	0.84	0.86	0.85	823
accuracy			0.85	1666
macro avg	0.85	0.85	0.85	1666
weighted avg	0.85	0.85	0.85	1666

When compared the results , Random Forest came to be the best one so next we will do the hyperparameter tuning with Random Forest

Hyperparameter Tuning

To maximize the performance of our machine learning models, we performed hyperparameter tuning for our project. In order to find the combination that yields the greatest outcomes based on a predetermined scoring criteria, this procedure entails methodically adjusting a model's parameters.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [200, 250, 300],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [2, 10],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 6],
    'bootstrap': [True, False],
    'criterion': ['entropy', 'gini']

}

# Initialize the classifier
rf = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,␣
 ↪n_jobs=-1, verbose=2)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print(best_params)
print(best_estimator)
```

GridSearchCV was used to perform hyperparameter tuning in order to find our RandomForestClassifier's optimal settings. A variety of 'n_estimators','max_features','max_depth','min_samples_split','min_samples_leaf', 'bootstrap', and 'criterion' variables were included in our search. The goal of this exhaustive search was to identify the setup that optimizes model performance. We determined the ideal parameter set and model instance after fitting GridSearchCV to our training set, confirming that our classifier is well-suited for the job at hand. This methodological step is essential to improving prediction accuracy and reliability of the model.

```
: from sklearn.ensemble import RandomForestClassifier

  # Use the best hyperparameters from GridSearchCV
  # best_params = {'criterion': 'gini', 'bootstrap': True, 'max_depth': 25,␣
   ↪'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2,␣
   ↪'n_estimators': 275}
  best_params = {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 10,␣
   ↪'max_features': 'auto', 'min_samples_leaf': 6, 'min_samples_split': 2,␣
   ↪'n_estimators': 200}


  random_forest_best = RandomForestClassifier(**best_params, random_state=42)

  # Fit the model on the entire training set
  random_forest_best.fit(X_train_val, y_train_val)

  # Now you can use rf_best to make predictions and evaluate the model
  y_pred_rd = random_forest_best.predict(X_test_validation)

  print((classification_report(y_test_validation, y_pred_rd)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.78      0.81       843
           1       0.79      0.86      0.82       823

    accuracy                           0.82      1666
   macro avg       0.82      0.82      0.82      1666
weighted avg       0.82      0.82      0.82      1666
```

We thoroughly investigated the optimal hyperparameters for our RandomForestClassifier using GridSearchCV, varying variables such as the number of trees, tree depth, and the minimum amount of samples needed to split a node. The method yielded improved model precision, recall, and f1-scores, indicating a notable increase over our baseline model. These ideal parameters were then determined. The efficacy of hyperparameter optimization in predictive modeling is emphasized by this careful adjustment.

Final Testing on Test Data

	precision	recall	f1-score	support
0	0.89	0.76	0.82	1009
1	0.56	0.77	0.65	400
accuracy			0.76	1409
macro avg	0.72	0.76	0.73	1409
weighted avg	0.80	0.76	0.77	1409

In conclusion, The model that was trained using the optimized hyperparameters exhibited better performance metrics in comparison to the baseline, proving that fine-tuning hyperparameters can increase the model's predictive accuracy. For this we majorly used the recall and to be specific the 1 value of recall because it is the true positive and true positive is something that is really important for us in the model as we are predicting churn. Some of our parameter value did came a little low as compared to baseline but still overall performance was good as it shows that the tuning and feature engineering was done adequately. Below we have also provided the result of baseline on test set for the comparison.

Logistic Regression

	precision	recall	f1-score	support
0	0.88	0.77	0.82	1035
1	0.53	0.72	0.61	374
accuracy			0.76	1409
macro avg	0.71	0.74	0.72	1409
weighted avg	0.79	0.76	0.77	1409

Decision Tree

	precision	recall	f1-score	support
0	0.83	0.78	0.81	1035
1	0.48	0.55	0.51	374
accuracy			0.72	1409
macro avg	0.65	0.67	0.66	1409
weighted avg	0.74	0.72	0.73	1409

Random Forest:

	precision	recall	f1-score	support
0	0.85	0.86	0.85	1035
1	0.59	0.59	0.59	374
accuracy			0.78	1409
macro avg	0.72	0.72	0.72	1409

SVM:

	precision	recall	f1-score	support
0	0.83	0.72	0.77	1035
1	0.44	0.61	0.51	374
accuracy			0.69	1409
macro avg	0.64	0.66	0.64	1409
weighted avg	0.73	0.69	0.70	1409