CSIT332
Fundamentals of Machine Learning

PROJECT REPORT

Predicting Customer Churn in a Telecommunications Company

December 10, 2024

Aadit Shah
Computer Science
FLAME University
aadit.shah@flame.edu.in

1 Introduction

1.1 Background

Customer churn is a problem faced by telecommunications companies. When customers leave, it can be costly for these companies due to revenue loss and the expenses incurred in acquiring new customers. By accurately predicting customer churn, companies can implement strategies to retain customers.

1.2 Problem Statement

The focus of this project is to predict customer churn in a telecommunications company. Understanding which customers are at high risk of churning allows the company to take measures to retain them.

1.3 Objective

The objective of this project is to develop a predictive model for customer churn using machine learning techniques. The steps involved will include data preprocessing, feature selection, model training, hyperparameter tuning, and evaluation. We will experiment with different machine learning algorithms to determine which method provides the best results based on evaluation metrics.

1.4 Data

The dataset used for this analysis is the **Telco Customer Churn Dataset**. The dataset provides valuable insights into customer demographics, subscription information, and service usage. By predicting which customers are likely to churn, the company can take proactive measures to retain them, ultimately reducing churn rates and increasing customer satisfaction.

1.4.1 Dataset Summary

- **Size**: 7,043 rows and 21 columns.
- **Target Variable**: `Churn` (Yes/No)
- **Features**: The dataset includes the following key attributes:
  - **Customer Information**: `customerID`, `gender`, `SeniorCitizen`, `Partner`, `Dependents`, `tenure`
  - **Service Information**: `PhoneService`, `MultipleLines`, `InternetService`, `OnlineSecurity`, `OnlineBackup`, `DeviceProtection`, `TechSupport`, `StreamingTV`, `StreamingMovies`

- **Account Details**: `Contract`, `PaperlessBilling`, `PaymentMethod`, `MonthlyCharges`, `TotalCharges`

2 Data Analysis and Preprocessing

2.1 Exploratory Data Analysis (EDA)

To gain insights into the dataset, we initially explore its structure, check for missing values, and identify patterns. This ensures that the data is ready for model training.

- **Inspecting the Data:** The dataset is loaded using `pd.read_csv()` to read the customer churn data into a DataFrame.
  - Use `data.head()` to preview the first few rows and understand the variables.
- **Observations:**
  - The dataset contains several categorical variables that may require transformation or encoding.
  - The `TotalCharges` column is stored as an `object` type despite containing numeric values. This suggests potential non-numeric entries.
- **Summary of the Dataset:** Use `data.info()` to examine the structure of the dataset, including column names and data types.
  - Most columns are of type `object`.
  - The `TotalCharges` column requires type conversion for analysis.
- **Handling Missing and Inconsistent Values:**
  - Check for missing values using `data.isna().sum()`.
    - **Observation:** No missing values were found in the dataset.
  - Check for empty strings using `data.eq(' ').sum()`.
    - **Observation:** The `TotalCharges` column has 11 empty entries. These rows will be filled with the median value.

2.2 Data Cleaning

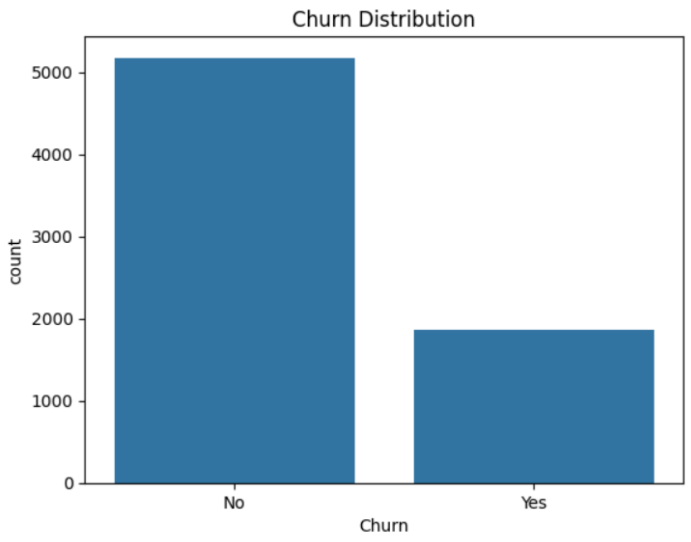
The cleaning process focuses on ensuring the data is free of errors, inconsistencies, and irrelevant features.

- **Converting Data Types:** Convert `TotalCharges` from `object` to `float` using `pd.to_numeric()`, coercing invalid entries to `NaN`, and replacing them with the median value.

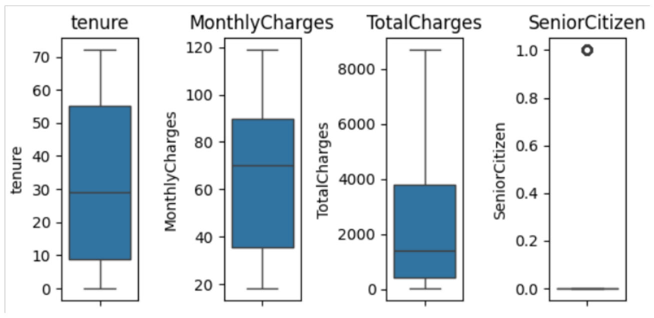
- **Removing Irrelevant Features:** Drop the `customerID` column as it does not contribute to prediction using `data.drop()`.
- **Identifying and Removing Duplicates:** Check for duplicates using `data.duplicated().sum()` and remove them to avoid biased model performance.
- **Handling Categorical and Numerical Columns:** Separate categorical and numerical columns using `data.select_dtypes()` for preprocessing.

2.3 Feature Engineering

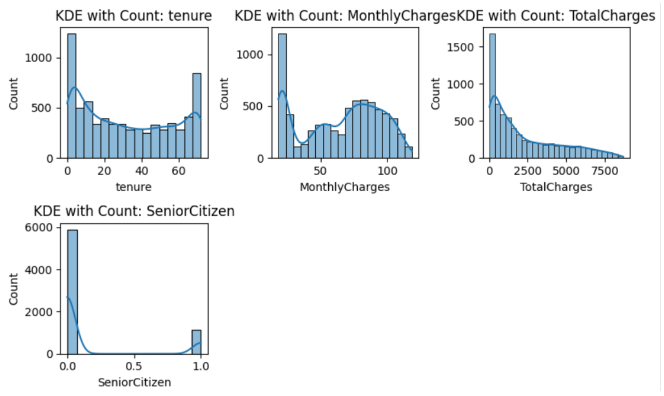
- **Visualizing Distributions:** Use various plots to understand feature distributions and detect anomalies.
  - **Churn Distribution:** A count plot shows the target variable's imbalance, with 5163 customers labeled as `No` and 1869 as `Yes`.



  - **Boxplots for Outliers:**
    - **Tenure:** It is consistent and the distribution does not have extreme values.
    - **MonthlyCharges:** Even this is a uniform distribution and not much deviations.
    - **TotalCharges:** We did not identify any extreme values or outliers detected.

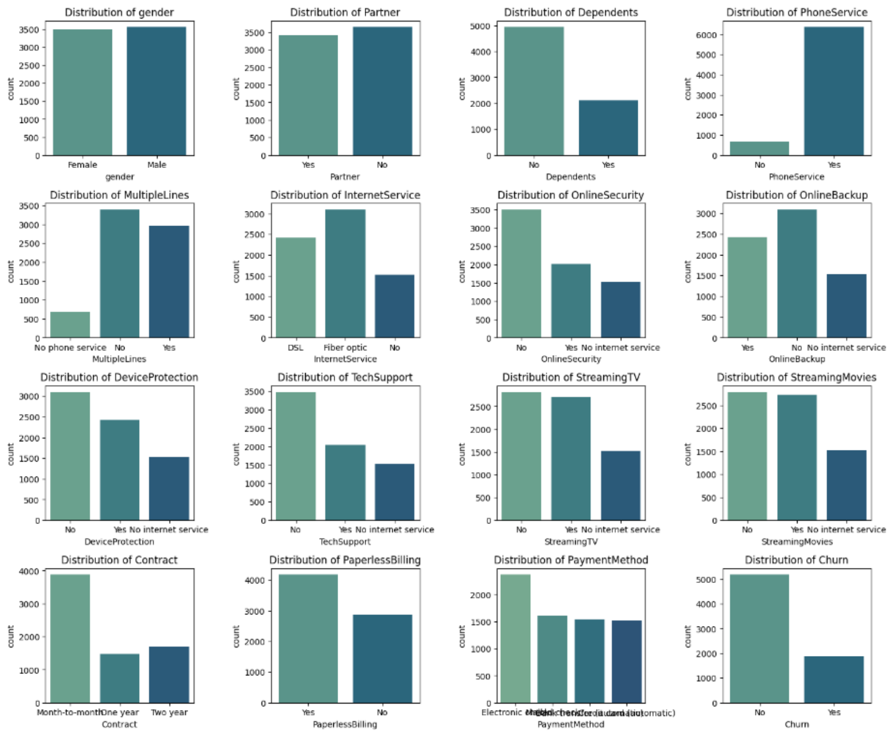


- **KDE Plots for Numerical Variables:** This gives a representation of data distribution.
  - `Tenure` shows a balanced distribution very short or very long subscription periods.
  - `MonthlyCharges` are mostly balanced, but we can see that many customers in the histogram have peaks at lower values.
  - The histogram for `TotalCharges` is positively skewed and concentrated towards lower values. Customers appear to have lower total charges, which can be seen through the peak around the lower values.



- **Count Plots for Categorical Variables:** Help identify overrepresented or underrepresented categories.
  - We see that gender and partner statuses are distributed evenly, with balanced male and female users being represented.
  - Around 30% of customers have dependents, seeing this we aren't sure if it impacts churn rates.

* A strong preference for paperless billing is evident, with a near-equal distribution of customers opting for or against it.

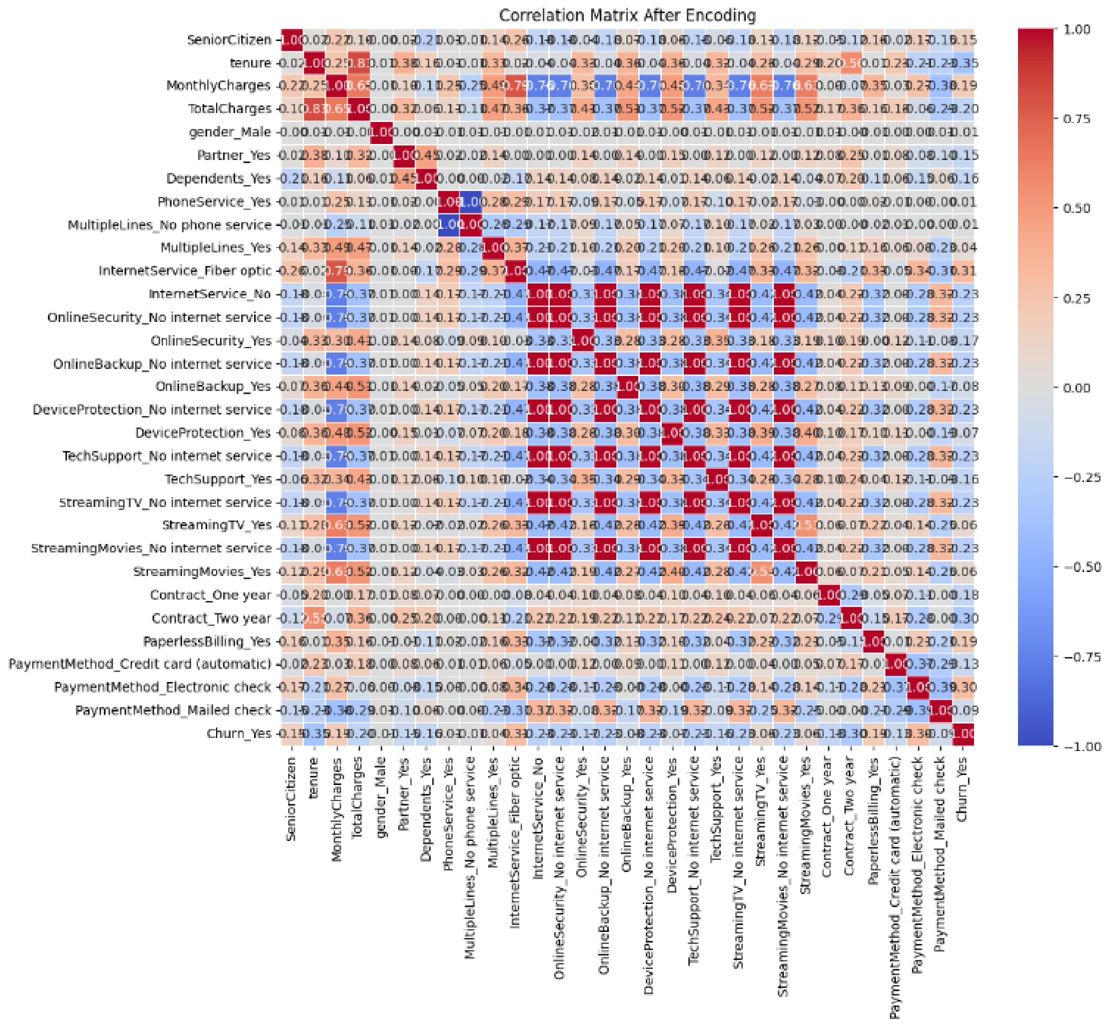


- **One-Hot Encoding:** Convert categorical variables to numerical representations using one-hot encoding to avoid ordinal relationships.
  - **Impact on the Model:** After one-hot encoding, the dataset might have more features, especially for high-cardinality categorical variables. While this can lead to a sparse matrix, it ensures that the model treats each category equally and prevents any biased assumptions about their relationships.
- **Feature Scaling:** Use `StandardScaler` to scale numerical features like `MonthlyCharges` and `TotalCharges` to a mean of 0 and a standard deviation of 1.
  - **Scaling** ensures that numerical features are on a comparable scale, which helps prevent models from giving undue importance to features with larger ranges.

2.4 Feature Selection

- **Correlation Analysis:** A correlation matrix and heatmap are used to identify and visualize relationships between numerical features, helping to reduce multicollinearity. Features with correlations above 0.9 are dropped.

- **Dropped Features:**
  - `MultipleLines_No phone service`
  - `OnlineSecurity_No internet service`
  - `OnlineBackup_No internet service`
  - `DeviceProtection_No internet service`
  - `TechSupport_No internet service`
  - `StreamingTV_No internet service`
  - `StreamingMovies_No internet service`



2.5 Final Preprocessing

- **Train-Test Split:** Divide the data into training and testing sets using `train_test_split()` with an 80-20 split.

3 Model Selection and Implementation

3.1 Model Architecture of Decision Tree Classifier

The Decision Tree Classifier was chosen for its simplicity and interpretability. Its hierarchical structure makes it easy for decision-making and useful for classification problems like customer churn prediction. It can efficiently handle numerical and categorical data and models non-linear decision boundaries.

3.1.1 Hyperparameter Tuning

I did not conduct extensive hyperparameter tuning as I initially wanted to get an idea of the results I was getting. I kept the scoring as f1 as that seemed to be the best metric for an imbalanced dataset. I used Grid Search with 5-fold cross-validation and trained three parameter combinations:

- **Criterion:** Tested `gini` and `entropy` splitting rules.
- **Max Depth:** Varying tree depth prevented overfitting while capturing sufficient complexity.
- **Min Samples Split:** Adjusted to ensure nodes split only when containing enough samples, controlling tree growth.
- **Class Weight:** Setting to `balanced` accounted for dataset imbalance.

The optimal configuration was:

```
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 2,
                    'class_weight': 'balanced'}
```

3.1.2 Training and Validation

The classifier was trained with the optimal hyperparameters identified during the grid search. Validation metrics, focusing on F1-score, prioritized identifying churned customers for proactive retention.

3.1.3 Performance Metrics

The Decision Tree achieved:

- **Accuracy:** 0.73
- **Precision:** 0.49
- **Recall:** 0.80
- **F1-Score:** 0.61
- **AUC-ROC:** 0.78

Analysis: High recall (0.80) reflected effective identification of churned customers, while lower precision (0.49) indicated a higher false-positive rate. This was a positive sign due to its identification and I thought that this could be used to implement random forests. While providing good interpretability and recall, the Decision Tree's lower precision highlighted potential for improvement.

3.2 Model Architecture of Random Forest Classifier

The Random Forest Classifier, an ensemble method combining multiple decision trees, was chosen for its generalizability. Since decision trees did a good job in identifying those that churn, I thought that Random Forests would only be better since it combines multiple decision trees.

3.2.1 Hyperparameter Tuning

Again, I put the scoring as f1 and conducted Grid Search t focused on tree depth, sample splits, and feature selection:

- **Number of Estimators (`n_estimators`):** Tested 25 and 50 trees.
- **Maximum Depth (`max_depth`):** Included 3, 5, 10, and no restriction to evaluate complexity.
- **Minimum Samples Split (`min_samples_split`):** Varied from 2 to 5.
- **Minimum Samples per Leaf (`min_samples_leaf`):** Tested 1, 2, and 4.
- **Class Weight (`class_weight`):** Set to `balanced`.

Optimal configuration:

```
{'n_estimators': 25, 'max_depth': 5, 'min_samples_split': 2,
    'min_samples_leaf': 2, 'class_weight': 'balanced'}
```

3.2.2 Training and Validation

Cross-validation further reinforced the reliability, as the selected parameters yielded stable f1 scores and recall metrics across multiple iterations. The balanced class weight helped mitigate any potential bias towards the majority class

3.2.3 Performance Metrics

The Random Forest achieved:

- **Accuracy:** 0.76
- **Precision:** 0.53
- **Recall:** 0.83
- **F1-Score:** 0.65
- **AUC-ROC:** 0.86

Analysis: High recall demonstrated effective churn detection, with slightly lower precision (0.53) indicating acceptable trade-offs in this business context. Compared to Decision Tree, Random Forest improved recall and AUC-ROC, reflecting better generalization.

3.3 Model Architecture of Neural Network (MLP Classifier)

The Neural Network was chosen for its capability to model complex, non-linear relationships.

3.3.1 Hyperparameter Tuning

Grid Search with 5-fold cross-validation explored:

- **Hidden Layer Sizes:** Tested architectures with 3–5 layers.
- **Activation Functions:** `tanh` and `relu`.
- **Solver:** `adam` and `sgd`.
- **Max Iterations:** 200, 300, and 500.

Optimal configuration:

```
{'hidden_layer_sizes': (100, 50, 25), 'activation': 'tanh', 'solver':
                    'sgd', 'max_iter': 300}
```

3.3.2 Training and Validation

The neural network was trained on the churn dataset using optimal parameters. 5 fold cross validation was used and we saw a slight increase from 3 fold to 5 fold.

3.3.3 Performance Metrics

The Neural Network achieved:

- **Accuracy:** 0.81
- **Precision:** 0.65
- **Recall:** 0.60
- **F1-Score:** 0.62
- **AUC-ROC:** 0.86

Analysis: Higher accuracy and AUC-ROC compared to Decision Tree demonstrated better generalization. However, slightly lower recall (0.60) suggested the need for improvement. The Neural Network outperformed simpler models in overall accuracy but lagged in recall.

3.4 Model Architecture of Logistic Regression

Logistic Regression was chosen as a simple and interpretable model for the binary classification task.

3.4.1 Hyperparameter Tuning

A grid search was conducted to identify the best combination of hyperparameters for Logistic Regression:

- **Regularization Strength (‘C‘):** The regularization parameter ‘C‘ controls the trade-off between fitting the data and regularizing the model. We tested values of 1, 10, 100, and 200.
- **Maximum Iterations (‘max_iter‘):** The maximum number of iterations for optimization was varied between 200, 300, and 500 to ensure convergence during training.
- **L1 Ratio (‘l1_ratio‘):** Tested values 0.0 (L2 regularization) and 0.02 to explore the effect of a small L1 regularization component on feature selection.
- **Class Weights (‘class_weight‘):** The ‘balanced‘ option was used to adjust weights inversely proportional to class frequencies, addressing class imbalance.

The optimal hyperparameters found were:

```
{'C': 100, 'class_weight': 'balanced', 'l1_ratio': 0.0, 'max_iter': 300}
```

3.4.2 Training and Validation

The logistic regression model was trained using the optimal hyperparameters identified by the grid search. The class balancing was particularly important due to the imbalanced nature of the dataset, with significantly more non-churned customers than churned ones.

3.4.3 Performance Metrics

The Logistic Regression model achieved the following results:

- **Accuracy:** 0.75
- **AUC-ROC:** 0.86
- **F1-Score:** 0.64
- **Precision:** 0.52
- **Recall:** 0.83

Analysis: The model displayed good recall (0.83), indicating a strong ability to identify churned customers. However, its precision (0.52) was relatively low, suggesting a higher false positive rate. Despite this, the model performed well overall, with a balanced AUC-ROC score of 0.86.

3.5 Model Architecture of Support Vector Machine (SVM)

Support Vector Machine (SVM) was selected for its ability to handle high-dimensional feature spaces.

3.5.1 Hyperparameter Tuning

A grid search was conducted to identify the best combination of hyperparameters for SVM:

- **Regularization Strength (‘C‘):** Tested values of 0.1, 1, and 10 to explore the trade-off between margin maximization and misclassification.
- **Kernel:** Experimented with ‘rbf‘, ‘poly‘, and ‘linear‘ kernels to determine the best fit for the dataset.
- **Gamma (‘gamma‘):** Used the ‘scale‘ option for automatic scaling of gamma based on the input features.
- **Class Weights (‘class_weight‘):** The ‘balanced‘ option adjusted the weights inversely proportional to class frequencies to mitigate the impact of class imbalance.

The optimal hyperparameters identified were:

```
{'C': 10, 'class_weight': 'balanced', 'gamma': 'scale', 'kernel': 'rbf'}
```

3.5.2 Training and Validation

The SVM model was trained using the best hyperparameters found through grid search. The inclusion of the radial basis function (RBF) kernel enabled the model to capture non-linear relationships in the data, while class balancing addressed the dataset's inherent imbalance.

3.5.3 Performance Metrics

The SVM model achieved the following results:

- **Accuracy:** 0.75
- **AUC-ROC:** 0.86
- **F1-Score:** 0.64
- **Precision:** 0.52
- **Recall:** 0.84

Analysis: The model demonstrated excellent recall (0.84), indicating a strong ability to identify churned customers. However, the precision (0.52) was lower, reflecting a higher false positive rate. The balanced AUC-ROC score of 0.86 highlights the model's overall performance in distinguishing between classes.

3.6 Unsupervised Model - Model Architecture of K-Means Clustering

K-Means clustering, an unsupervised learning algorithm, was used to segment the data into distinct groups based on feature similarity. This approach helps identify patterns in customer churn behavior.

3.6.1 Hyperparameter Tuning

- **Optimal Number of Clusters (k):** Determined using:
  - **Elbow Method:** Plotted inertia for k values from 2 to 9. The elbow point suggested $k = 2$.
  - **Silhouette Analysis:** Silhouette scores were computed for the same k range, with the highest score of 0.66 at $k = 2$.

3.6.2 Training and Validation

- The K-Means model was trained on the dataset excluding the target variable (`Churn_Yes`).
- Clusters were assigned to the dataset as a new feature, `KMeans_Cluster`.

- **Validation:** Cluster quality was evaluated using inertia and silhouette score:
  - **Inertia:** 887,908.22.
  - **Silhouette Score:** 0.66.

3.6.3 Performance Metrics

The clustering results were analyzed using the following metrics:

3.6.4 Cluster vs. Churn Analysis

A cross-tabulation of clusters and `Churn_Yes` revealed:

- **Cluster 0:** 87.80% non-churn, 12.20% churn.
- **Cluster 1:** 61.94% non-churn, 38.06% churn.

4 Results and Discussion

4.1 Model Performance

The performance of various models was evaluated using multiple metrics. I mainly looked at F1 score but also wanted to ensure that the recall score was not very low. Since recall gives us the true positive rate we want a high value when we're identifying the churn. We want to ensure that we identify the maximum number of customers that will churn so that we can take measures to retain them. Having a high F1 score ensures that so that we do not have many false negatives. Although there is not much harm in marketing to someone who is continuing it will add to costs. Thus, keeping F1 as the main metric, we ensured that recall score was also high. The results of the experiments under the best configurations are presented in Table 1

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	73.0%	49.0%	80.0%	61.0%
Support Vector Machine (SVM)	75.0%	52.0%	84.0%	64.0%
Random Forest	**76.0%**	**53.0%**	**83.0%**	**65.0%**
Neural Network	81.0%	65.0%	60.0%	62.0%

Table 1: Comparison of different models

4.2 Comparative Study

Among all models, two models stood out based on their performance: Support Vector Machine (SVM) and Random Forest. Both models underwent extensive hyperparameter tuning to optimize their performance. Despite efforts to improve the F1 score further, Random Forest consistently provided the best results with an F1 score of 65.0%.

SVM also performed well, achieving an F1 score of 64.0% and a slightly higher recall. However, the Random Forest model outperformed SVM and other models in terms of F1 but fell slightly behind in recall, which is a crucial metric for this customer churn dataset.

The confusion matrices for SVM and Random Forest are as follows:

$$\text{SVM Confusion Matrix: } \begin{bmatrix} 741 & 295 \\ 59 & 314 \end{bmatrix}$$

$$\text{Random Forest Confusion Matrix: } \begin{bmatrix} 764 & 272 \\ 62 & 311 \end{bmatrix}$$

The classification reports for both models are provided below:

Table 2: Random Forest Classification Report

Class	**Precision**	**Recall**	**F1-Score**	**Support**
False	0.92	0.74	0.82	1036
True	0.53	0.83	0.65	373
Accuracy	0.76(1409)			
Macro Avg	0.73	0.79	0.74	1409
Weighted Avg	0.82	0.76	0.78	1409

Table 3: SVM Classification Report

Class	**Precision**	**Recall**	**F1-Score**	**Support**
False	0.93	0.72	0.81	1036
True	0.52	0.84	0.64	373
Accuracy	0.75(1409)			
Macro Avg	0.72	0.78	0.72	1409
Weighted Avg	0.82	0.75	0.76	1409

4.3 Discussion

The Random Forest model was selected as the final model due to its high F1 score and recall. These metrics were critical because identifying churned customers accurately was more important than minimizing false positives. Despite extensive hyperparameter tuning across all models, including adjusting the number of trees in Random Forest and kernel functions in SVM, the F1 score was not increasing this highlighted the complexity of the dataset.

Key insights:

- Random Forest achieved a balance between precision and F1 score, making it well-suited for churn prediction.
- SVM, while performing slightly lower, still demonstrated great results, especially on smaller datasets.
- Models such as Logistic Regression and Neural Networks lagged behind due to their inability to capture complex patterns in the data effectively.

4.4 Limitations and Future Work

The primary limitation of this analysis was the inability to further improve the F1 score despite experimenting with various models and hyperparameters. Although a score of 65% is good ideally a value greater than 70 is something I would be happy with This could be due to:

- Limited feature engineering: Additional engineered features could enhance model performance. With my current knowledge, I tried different feature engineering techniques which did not work maybe some complex ones would have.
- Imbalanced dataset: This dataset had a great imbalance and despite trying techniques such as SMOTE, the values of the metrics were not increasing. Maybe some other technique to balance the data could improve the metrics.

Future work will focus on:

- Exploring advanced techniques like feature selection and dimensionality reduction.
- Identifying a technique that solves the problem of the complexity and imbalance in the dataset.

5 Conclusion

In this project, we developed a predictive model for customer churn in a telecommunications company using various machine learning algorithms. The primary goal was to identify customers at high risk of churning, enabling the company to implement effective retention strategies.

The Random Forest model emerged as the most effective solution, achieving an F1 score of 65.0% and a recall of 83.0%. This model's balance between precision and recall makes it particularly suitable for this application, where minimizing false negatives is crucial.

In summary, this project contributes valuable insights into customer churn prediction, which can assist telecommunications companies in making informed decisions and enhancing customer satisfaction.