VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MACHINE LEARNING (CO3117)

COMPREHENSIVE MACHINE LEARNING MODELS:

IMPLEMENTATION AND COMPARISON

Advisor:	Nguyễn An Khương	
Students:	Nguyễn Quang Duy	2252120
	Văn Duy Anh	2252045
	Nguyễn Đoàn Hải Băng	2252078
	Trần Vũ Hảo	2052978
	Đào Tiến Tuấn	1953069

HO CHI MINH CITY, MARCH 2025

Members Contribution

No	Fullname	Student ID	Tasks assigned	Contributions
1	Nguyen Quang Duy	2252120	Neural networks model	100%
2	Van Duy Anh	2252045	Naive Bayes model	100%
3	Nguyen Doan Hai Bang	2252078	Genetic algorithm	100%
4	Tran Vu Hao	2052978	Decision trees model	100%
5	Dao Tien Tuan	1953069	Graphical model	100%

Contents

1 Introduction

1.1 Background

Customer churn prediction is a crucial task in understanding why customers discontinue their subscriptions and how businesses can mitigate these losses. In the telecommunications industry, customer churn directly impacts revenue and customer retention strategies. By analyzing customer behavior, service usage patterns, and contractual details, businesses can take proactive measures to enhance customer experience and reduce churn rates. This project focuses on building and comparing multiple machine learning models, including decision trees, neural networks, naive Bayes, genetic algorithms, and graphical models, to predict customer churn in the telecom sector. The performance of these models is evaluated using precision, recall, and F1-score to determine their effectiveness in identifying potential churners.

1.2 Goal

The primary goal of this project is to implement and compare various machine learning models to understand their strengths and weaknesses in predicting customer churn. Rather than focusing on achieving state-of-the-art performance, this study emphasizes model engineering, implementation, and comparative analysis. By evaluating different algorithms, we aim to gain hands-on experience and insights into their practical applications in churn prediction.

1.3 Setup

The project is implemented using PyTorch for model development and training. The Telco Customer Churn dataset is preprocessed to handle missing values, encode categorical features, and normalize numerical data. Various machine learning models are then trained and evaluated using standard classification metrics. The experimental setup includes computational resource considerations, hyperparameter tuning, and performance analysis across different models.

2 Data Overview

2.1 Dataset Description

The Telco Customer Churn dataset contains information about telecommunications customers, with the primary objective of predicting customer churn. Churn refers to customers discontinuing their service with the company. By analyzing customer attributes, including demographics, service subscriptions, and billing details, businesses can identify patterns that may indicate a higher likelihood of churn.

This dataset is widely used for customer retention analysis, offering valuable insights into customer behavior and helping organizations develop data-driven strategies to improve customer loyalty. The dataset was originally provided by IBM and serves as a benchmark dataset for churn prediction in the telecom industry.

2.2 Features

The dataset consists of 7,043 records, each representing a unique customer, and includes 21 attributes that capture various aspects of a customer's profile and service history. These attributes can be categorized into different groups:

- **Customer Information:** These attributes provide demographic details about the customer, such as:
  - customerID: A unique identifier assigned to each customer.
  - gender: The gender of the customer.
  - SeniorCitizen: A binary indicator (0 or 1) representing whether the customer is a senior citizen.
  - Partner: Whether the customer has a partner.
  - Dependents: Whether the customer has dependents.
- **Subscription Details:** These attributes describe the customer's subscription type and contract status:
  - tenure: The number of months a customer has stayed with the company.
  - Contract: The contract type (Month-to-month, One year, Two year).
  - PaperlessBilling: Indicates whether the customer uses paperless billing.
  - PaymentMethod: The customer's preferred payment method (Electronic check, Mailed check, Bank transfer, Credit card).
- **Service Usage:** These attributes indicate the services subscribed to by the customer:
  - PhoneService: Whether the customer has a phone service.
  - MultipleLines: Whether the customer has multiple lines.
  - InternetService: The type of internet service (DSL, Fiber optic, No service).
  - OnlineSecurity: Whether the customer has online security services.
  - OnlineBackup: Whether the customer has an online backup service.
  - DeviceProtection: Whether the customer has device protection.

  - TechSupport: Whether the customer has technical support.
  - StreamingTV: Whether the customer has a streaming TV service.
  - StreamingMovies: Whether the customer has a streaming movie service.

- **Billing Information:** These attributes provide financial details of the customer's subscription:
  - MonthlyCharges: The amount charged to the customer monthly.
  - TotalCharges: The total amount charged to the customer over their tenure.

- **Target Variable:**
  - Churn: The dependent variable indicates whether the customer has churned (Yes/No).

2.3 Data Characteristics

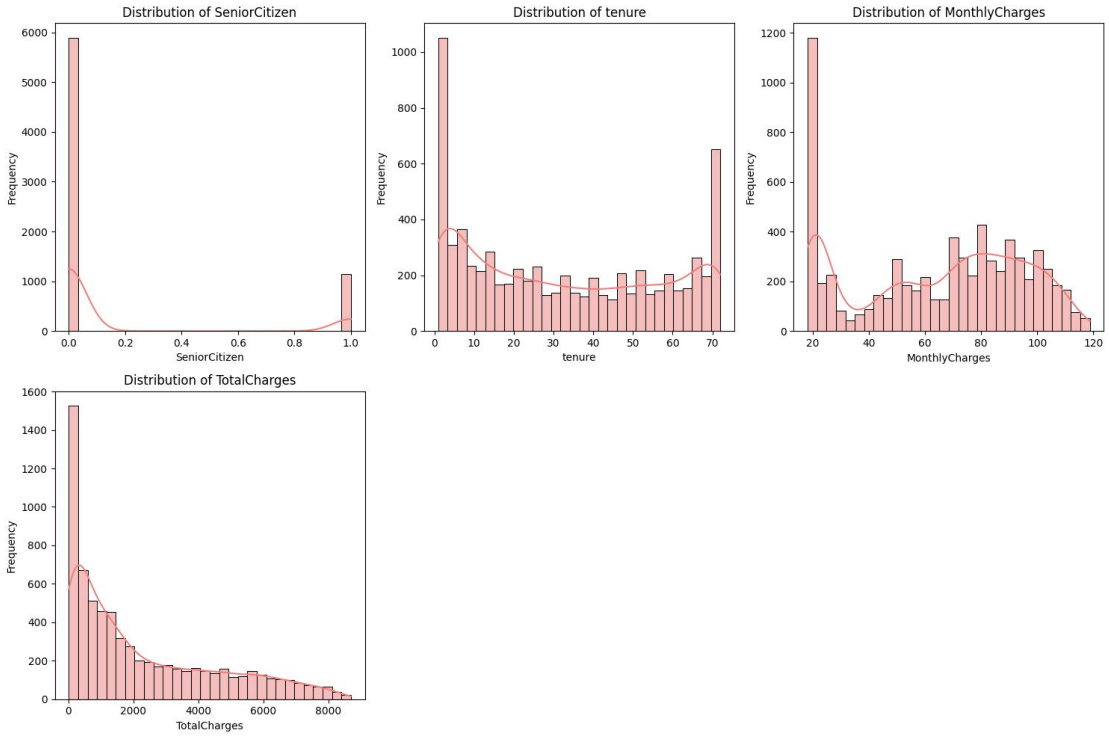

Figure 1: Distribution of Key Features in the Telco Customer Churn Dataset

The histograms provide a comprehensive look at the distribution of key features in the Telco Customer Churn dataset. The SeniorCitizen variable is highly imbalanced, with most customers being non-senior citizens. This indicates that the dataset has a skewed demographic, with fewer senior customers. The tenure distribution shows a wide range of values, with notable peaks at specific points, suggesting that many customers have been with the company for just a few years. The MonthlyCharges distribution is bimodal, with peaks at around 20 and 80, likely reflecting two distinct customer groups: one with lower-cost subscriptions and another with higher-cost

plans. Lastly, the TotalCharges distribution is right-skewed, with most customers having lower total charges, but a small group of long-term or high-value customers having significantly higher charges. These insights are valuable for preprocessing and feature engineering in predictive models for customer churn.

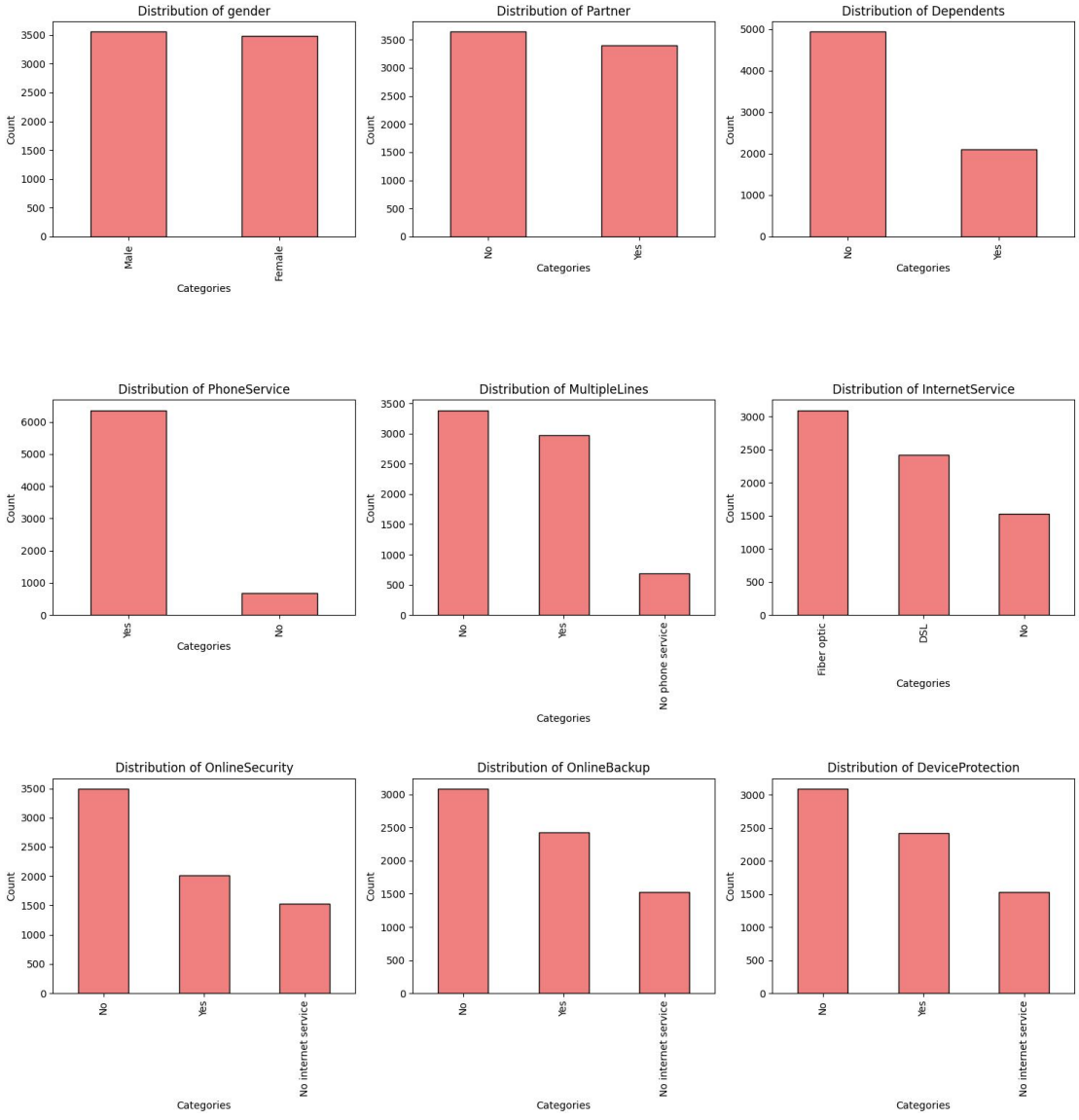

Figure 2: Distribution of Categorical Features in the Telco Customer Churn Dataset

The bar charts give an overview of the categorical features in the dataset. The gender distribution is nearly balanced between male and female customers. However, a large proportion of customers do not have partners or dependents, as seen in the Partner and Dependents distributions. Regarding service features, most customers have PhoneService, but fewer have MultipleLines. The majority of customers use fiber optic or DSL for internet, while a smaller proportion do not have

internet service at all.

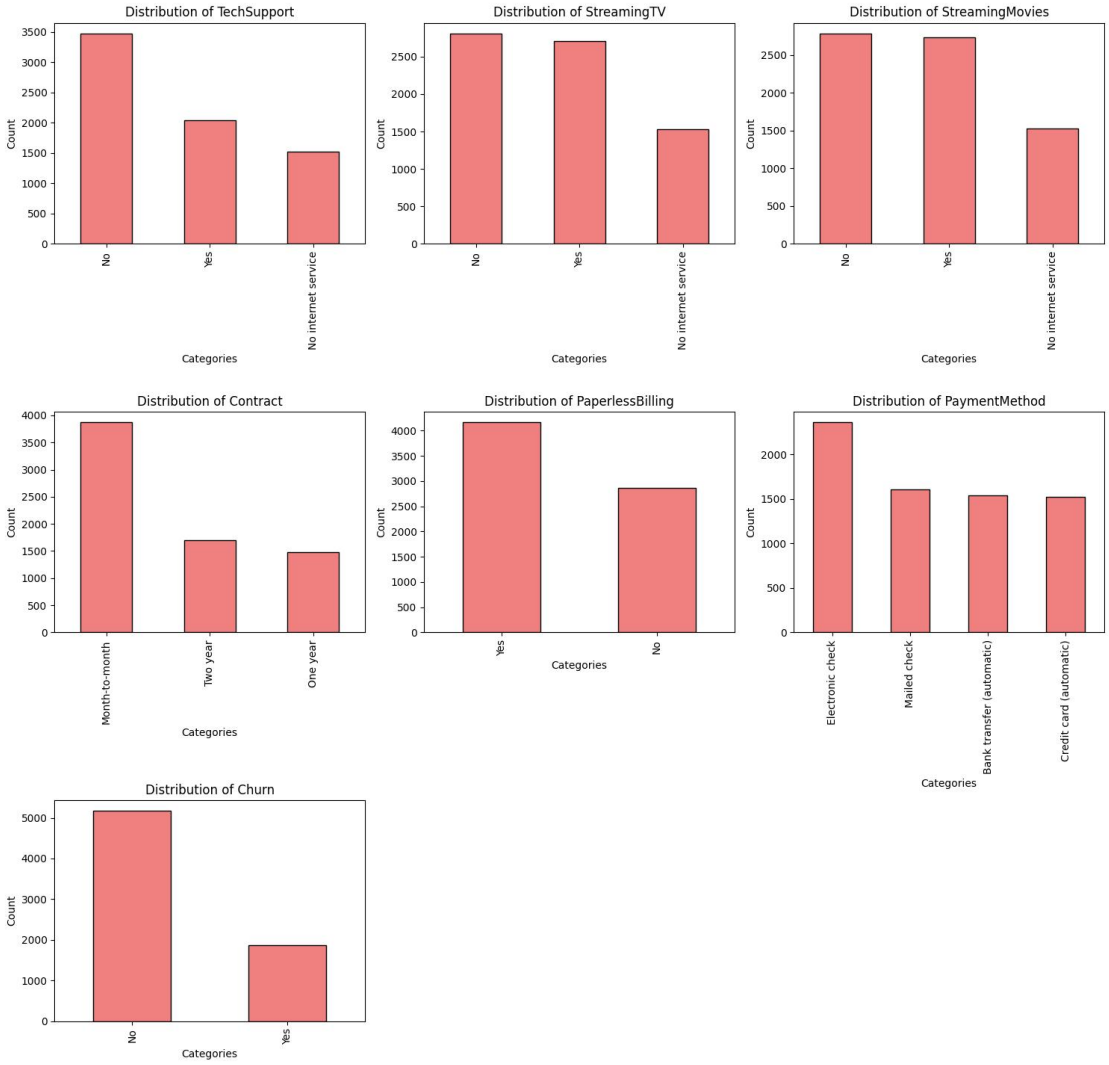

Figure 3: Distribution of Categorical Features in the Telco Customer Churn Dataset

Additionally, many customers do not use add-on services like OnlineSecurity, OnlineBackup, DeviceProtection, or Streaming services, which may suggest limited engagement with premium features. Most customers are on month-to-month contracts, and a large proportion use paperless billing. As for payment methods, electronic checks are the most common, followed by mailed checks and bank transfers. Finally, the Churn distribution shows that most customers stay with the company, with a smaller number deciding to leave. These observations can help inform future analyses and model development aimed at predicting customer churn.

The chart showing the relationship between churn and tenure reveals that churn is highest in the first few months of a customer's subscription, with many customers leaving early. As the

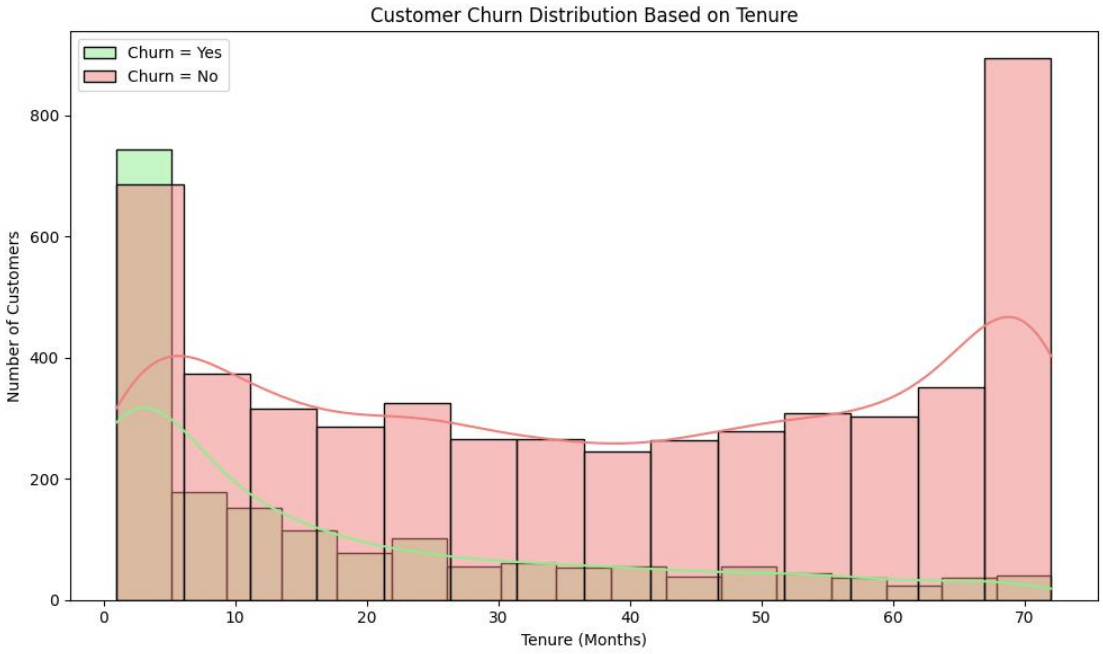

Figure 4: Customer Churn Distribution Based on Tenure

tenure increases, the number of churned customers drops significantly, indicating that long-term customers are far less likely to cancel their services. This suggests that the company has a higher retention rate among long-term customers, possibly due to factors like customer satisfaction or loyalty programs. The trend underscores the importance of focusing on customer retention in the early stages of subscription, as customers who stay longer are more likely to remain with the company.

In the chart illustrating the relationship between churn and monthly charges, we see that customers with lower monthly charges (around 20) tend to stay with the service, as indicated by the higher bar and lower churn rate in this range. Conversely, as monthly charges increase, the likelihood of churn also rises. This suggests that customers with higher monthly charges may be more sensitive to the perceived value of the service or may find better alternatives from competitors, leading to a higher churn rate among this group. The trend highlights the importance of carefully managing pricing plans, particularly for customers with higher charges, to reduce the risk of churn.

2.4 Data Preprocessing

Data preprocessing is a crucial step in preparing the dataset for machine learning models. It ensures that all features are formatted correctly, handles any missing values, and encodes categorical variables properly. The preprocessing steps performed in this study are described below.

We begin by storing the data in a pandas DataFrame, which allows for efficient manipulation and analysis. The dataset is loaded using the read_csv function.

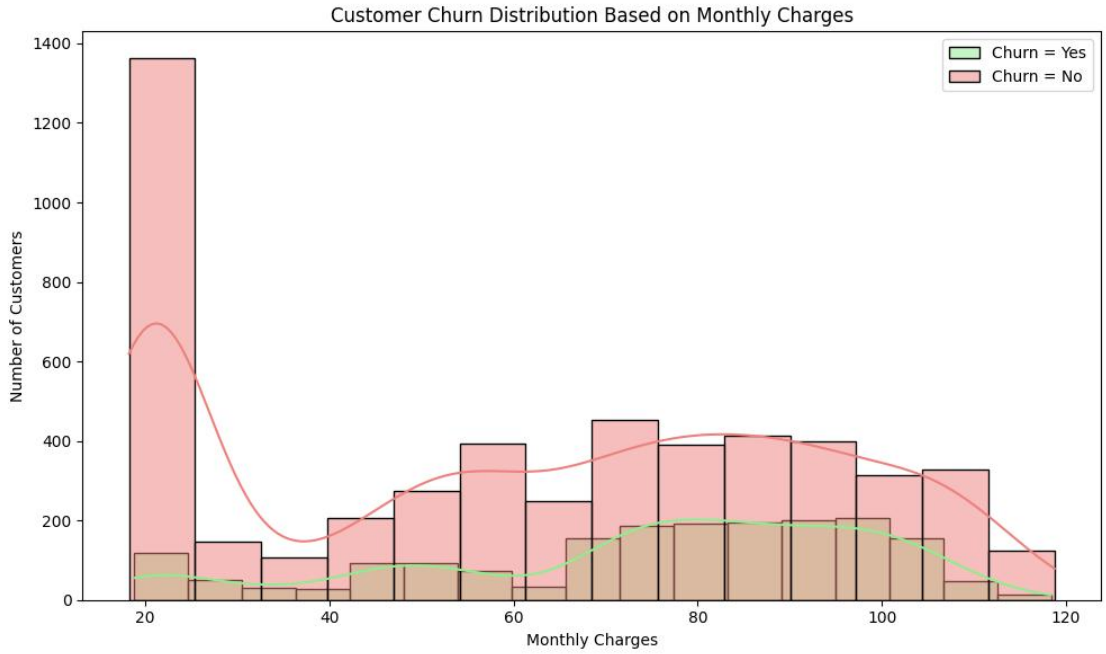

Figure 5: Customer Churn Distribution Based on Monthly Charges

2.4.1 Data Cleaning

The first step in cleaning the data is to remove the customerID column. This column only serves as an identifier and does not contribute to the predictive analysis, so removing it helps simplify the dataset and ensures that irrelevant information doesn't negatively impact the model's performance.

Next, we address an issue with the TotalCharges column. While this column should be numerical, it is stored as an object (string). Upon inspection, we found that some values in this column were blank spaces instead of numerical data. Since there were only 11 missing values out of 7,043 records, we decided to remove these rows rather than attempt to impute the missing values, which helps maintain the integrity of the data.

2.4.2 Handling Missing Values

The dataset was checked for missing values using the df.isnull().sum() method, and the results confirmed that all columns have zero missing values. This means the dataset is complete and free of missing data, which is essential for building accurate and reliable machine learning models.

2.4.3 Feature Engineering

Feature engineering is done to enhance the predictive power of the models. We begin by creating a new feature, tenure group, which categorizes customers based on their tenure duration, helping the model understand different customer segments.

We also examine categorical columns and their unique values. Many columns, such as Partner and Dependents, contain binary values like Yes and No, so we convert these into numerical representations (1 for Yes and 0 for No) to make them compatible with machine learning models. Additionally, some variables contain values like "No internet service" or "No phone service," which we standardize to just "No" for consistency across these columns.

For the gender variable, we map "Female" to 1 and "Male" to 0. Boolean columns are also converted into integers, where True becomes 1 and False becomes 0. We apply this transformation to all boolean columns in the dataset to simplify processing while preserving interpretability.

For categorical variables with more than two unique values (e.g., InternetService, Contract, and PaymentMethod), we use One-Hot Encoding. This technique converts each categorical feature into multiple binary columns, ensuring that no unintended ordinal relationships are introduced, which makes the data ready for machine learning models.

2.4.4 Feature Scaling

Since machine learning models can be sensitive to features with different scales, we apply Min-Max Scaling to normalize the numerical features. Columns such as tenure, MonthlyCharges, and TotalCharges have values that span a wide range, so scaling these features ensures they all contribute proportionally to the model's learning process. Min-Max Scaling transforms these values to a range between 0 and 1.

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.         0.46478873 0.01408451 0.61971831 0.09859155 0.29577465
 0.12676056 0.38028169 0.85915493 0.16901408 0.21126761 0.8028169
 0.67605634 0.33802817 0.95774648 0.71830986 0.98591549 0.28169014
 0.15492958 0.4084507  0.64788732 1.         0.22535211 0.36619718
 0.05633803 0.63380282 0.14084507 0.97183099 0.87323944 0.5915493
 0.1971831  0.83098592 0.23943662 0.91549296 0.11267606 0.02816901
 0.42253521 0.69014085 0.88732394 0.77464789 0.08450704 0.57746479
 0.47887324 0.66197183 0.3943662  0.90140845 0.52112676 0.94366197
 0.43661972 0.76056338 0.50704225 0.49295775 0.56338028 0.07042254
 0.04225352 0.45070423 0.92957746 0.30985915 0.78873239 0.84507042
 0.18309859 0.26760563 0.73239437 0.54929577 0.81690141 0.32394366
 0.6056338  0.25352113 0.74647887 0.70422535 0.35211268 0.53521127]
PhoneService: [0 1]
MultipleLines: [0 1]
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
...
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 6: Feature Scaling Applied to the Dataset

Finally, after completing the preprocessing steps, we validate the transformed dataset by checking the unique values in the categorical columns. This final check ensures that all transformations have been applied correctly and that the dataset is ready for training without any inconsistencies.

3 Methodology

3.1 Decision Trees

3.2 Neural Networks

Neural networks are a type of machine learning model inspired by how the human brain works. They consist of layers of interconnected nodes, or artificial neurons, that process data, learn patterns, and make predictions. Neural networks are especially good at identifying complex relationships in data because they can model non-linear dependencies. This ability makes them ideal for tasks like image recognition, language processing, and, in our case, predicting customer churn.

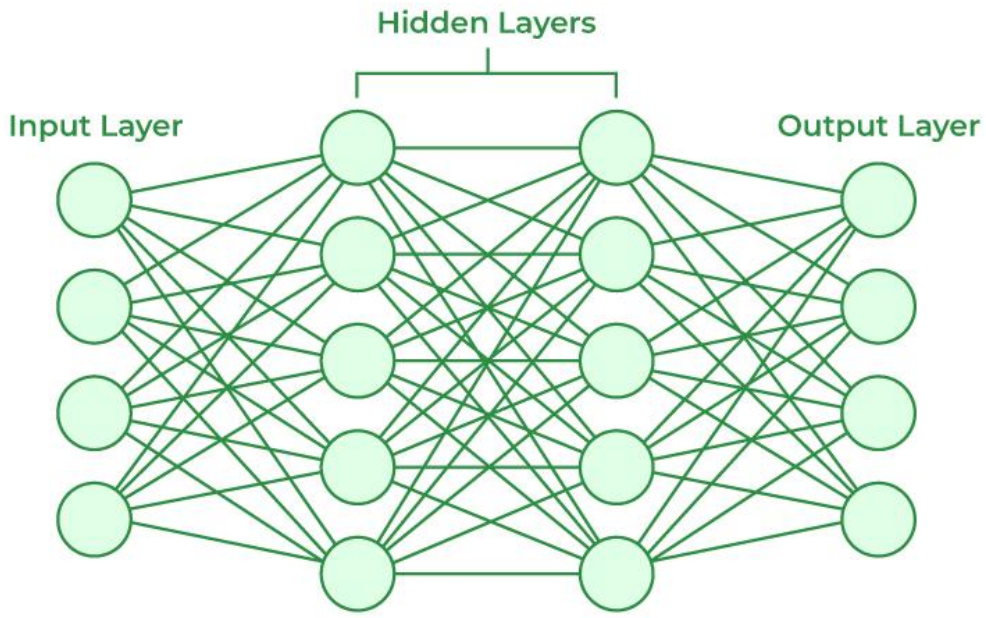

Figure 7: Neural Network Architecture

The core unit of a neural network is the artificial neuron. Each neuron takes input values, computes a weighted sum, passes the result through an activation function, and produces an output. These neurons are organized into layers: the input layer takes in the raw data, hidden layers process the data by applying multiple transformations, and the output layer gives the final prediction. The network learns by adjusting its weights in a process called backpropagation, where the model tries to minimize prediction errors. This is done using optimization algorithms like stochastic gradient descent (SGD) or Adam.

For customer churn prediction, neural networks are particularly useful because they can capture complex patterns in customer behavior, contract details, and service usage that might not be easily identified by simpler models. The network uses activation functions like ReLU (Rectified Linear Unit) in the hidden layers to introduce non-linearity and sigmoid in the output layer to

produce probabilities representing the likelihood of customer churn. The loss function typically used for training is binary cross-entropy, which measures how far the predicted values are from the actual values and helps guide the model's learning.

To prevent overfitting and improve the model's ability to generalize to new data, regularization techniques such as dropout and weight decay are often used. Additionally, hyperparameter tuning plays a critical role in optimizing the model's performance. This includes selecting the right number of hidden layers, the number of neurons in each layer, the learning rate, and the batch size. By training on historical customer data and evaluating the model's performance using metrics like precision, recall, F1-score, and AUC-ROC, the neural network can effectively predict the likelihood of a customer churning, which provides valuable insights for business decision-making.

3.3 Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem, which calculates the probability of a class given a set of features. The "naive" assumption is that all features are independent of each other given the class label. This assumption simplifies the computation significantly, making Naive Bayes a highly efficient algorithm for large datasets. In the context of customer churn prediction, each customer attribute — such as contract type, monthly charges, or tenure — is treated as an independent variable contributing to the likelihood of churn.

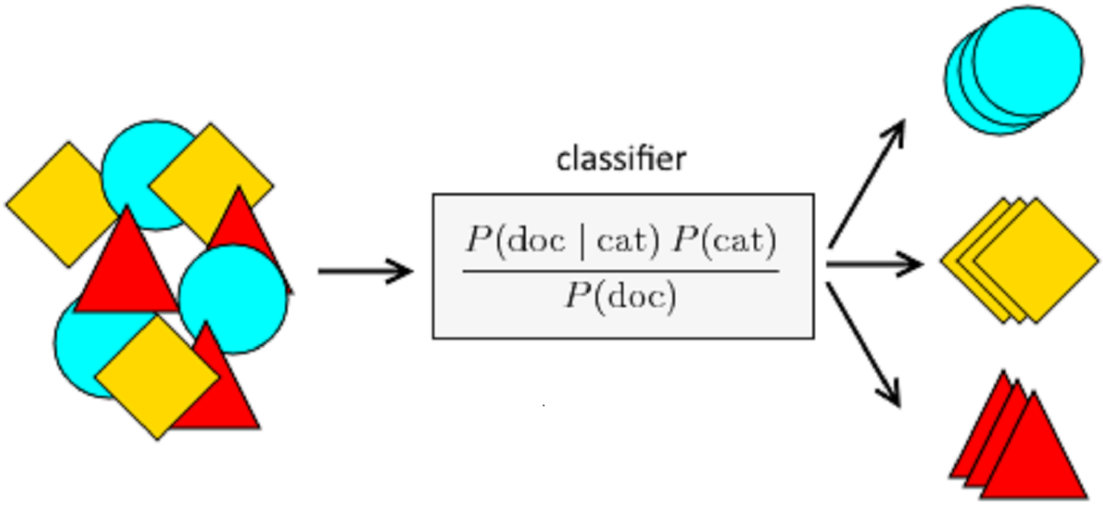

Figure 8: Naive Bayes Algorithm

The training process involves estimating the prior probabilities of each class (churn vs. no churn) and the conditional probabilities of each feature given the class. During prediction, Naive Bayes calculates the posterior probability for each class by combining these estimates and assigns the class with the highest probability to each instance. The model's simplicity makes it robust to small datasets and less prone to overfitting, although its performance can be affected if the independence assumption is strongly violated.

For this study, the Categorical Naive Bayes variant was employed due to the predominantly categorical nature of the Telco dataset. The model was tuned to optimize the smoothing parameter, which handles zero probabilities for unseen feature-class combinations. Despite its simplicity, Naive Bayes provided quick predictions and valuable insights into the relationships between individual features and the likelihood of churn.

4 Decision Trees

5 Neural Networks

5.1 Model Implementation

In this project, we implemented the neural network model using PyTorch's nn.Module. The NeuralNetworkModel class is built to predict customer churn from the Telco dataset. The model starts with an input layer that takes in 26 features (which corresponds to the number of features in the dataset) and maps them to a specified number of hidden neurons, with the default being 15 neurons. This number can be adjusted depending on the model's complexity. The model also allows for flexibility in the number of hidden layers, which can be defined by the num_hidden_layers parameter. Each hidden layer consists of a fully connected layer followed by a ReLU activation function, which adds non-linearity to the network and allows it to learn more complex relationships. To prevent overfitting, we include a dropout layer after each hidden layer, with a 50% probability by default. This means that during training, half of the neurons are randomly set to zero, forcing the model to generalize better.

The output layer consists of a single neuron that is connected to the last hidden layer. This output is passed through a Sigmoid activation function, which compresses the result into a value between 0 and 1. This output represents the probability of a customer churning. The forward method defines how the data flows through the network: it starts at the input layer, passes through the hidden layers with ReLU activations, and ends at the output layer with the Sigmoid function applied. This design enables the model to capture complex patterns in customer behavior while controlling for overfitting using dropout. The model is structured as a sequential model, making it easy to define and implement multiple layers in a straightforward manner.

5.2 Training Implementation

The train function is responsible for training the neural network model. It starts by initializing the model using the NeuralNetworkModel class, setting parameters like the input size, hidden layer size, number of hidden layers, and dropout rate. The function then creates data loaders for both the training and validation datasets, allowing the data to be processed in batches, which helps manage memory usage and speeds up training.

For the loss function, Binary Cross-Entropy Loss (BCELoss) is used, as it is well-suited for binary classification problems, such as predicting whether a customer will churn. The Adam optimizer is chosen because it efficiently adjusts the learning rate based on gradients, making the training process faster and more stable.

The training loop runs for a set number of epochs, with each epoch representing one full pass through the training data. During each epoch, the model is put in training mode, and the optimizer gradients are reset to zero before processing each batch. The input data is passed through the model, and the loss is calculated by comparing the model's predictions with the actual labels. The loss is then backpropagated through the network to compute gradients, and the optimizer updates the model's weights accordingly.

At the end of each epoch, the average training loss is calculated and saved for later analysis. Afterward, the model is evaluated on the validation dataset, using the same process but in evaluation mode (with no gradient calculation). The validation loss is also calculated and tracked. Every 10 epochs, the training and validation losses are printed to help monitor the model's

progress.

Once all epochs are completed, the function returns the trained model along with the lists of training and validation losses. These losses can be used to evaluate the model's performance and see how it improves over time. Overall, this implementation provides a structured approach to training and evaluating the neural network while keeping track of its progress at each step.

5.3 Model Tuning Implementation

The tune_hyperparameters function is used to optimize the hyperparameters of the neural network model with the help of Optuna, a framework for hyperparameter optimization. It accepts several inputs: the training function (model_train_fn), training and validation data (X_train, y_train, X_val, y_val), the number of trials for optimization (n_trials), the optimization direction (either minimize or maximize), and the search space for the hyperparameters.

The function starts by setting up an Optuna study, specifying the direction of optimization and using a TPESampler to guide the search. Then, the optimization process begins, and for each trial, it calls the model_train_fn function, passing the training and validation data, along with the current trial's hyperparameters. Once all the trials are complete, the function logs the best set of hyperparameters found during the search process. It then returns the Optuna study object, which holds details about the entire optimization process. This approach allows for efficient hyperparameter tuning, helping to find the best configuration for the neural network to improve its performance.

5.4 Training Process

The training process begins by training the neural network model using manually selected hyperparameters. In this initial step, we set key hyperparameters such as the hidden layer size, number of hidden layers, dropout rate, learning rate, weight decay, batch size, and the number of epochs. These parameters are passed into the train_and_save function, which trains the model according to the given settings and saves both the model and training results.

After completing the initial training, we move on to fine-tuning the model by optimizing its hyperparameters. We define a search space that includes a range of values for hyperparameters such as learning rate, batch size, hidden layer size, number of hidden layers, dropout rate, epochs, and weight decay. We use the tune_and_save function to run 50 trials, with the goal of minimizing the validation loss and finding the best combination of hyperparameters. After the tuning process is finished, the best hyperparameters are selected.

Finally, with the optimal hyperparameters identified, we re-train the model using the train_study_and_save function, ensuring that the model is trained with the most effective settings. Once training is complete, the model and its results are saved for further evaluation and deployment. This process ensures that the model is not only trained with appropriate hyperparameters but also optimized for better performance.

5.5 Manual Training Results and Analysis

In this step, the neural network model is trained with manually selected hyperparameters. The training runs for 50 epochs, with key hyperparameters such as a hidden layer size of 15 neurons,

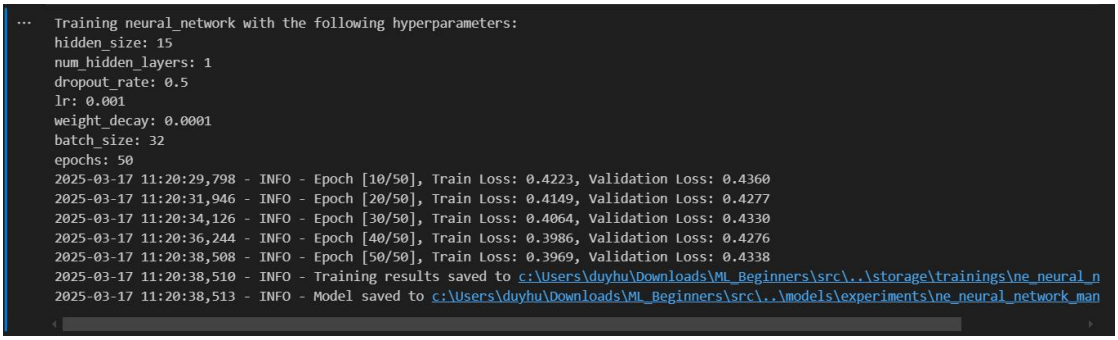

Figure 9: Training Log for Manual Hyperparameters

one hidden layer, dropout rate of 0.5, learning rate of 0.001, weight decay of 0.0001, batch size of 32, and a total of 50 epochs. The training and validation losses show that the model improves over time, but the validation loss remains relatively stable by the end of the training, indicating that there may still be room for improvement, particularly in generalizing to unseen data.

5.5.1 Evaluation and Cross-Validation

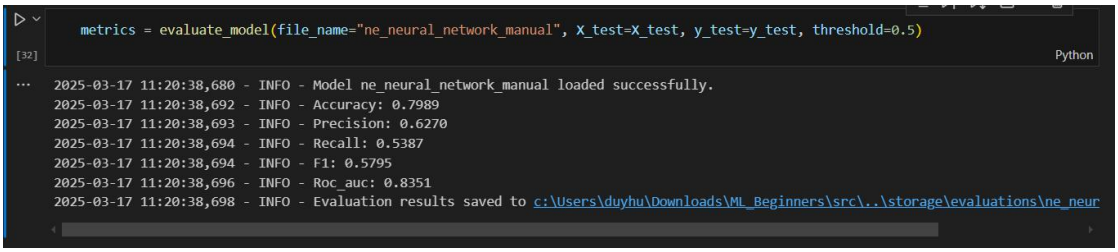

Figure 10: Model Evaluation and Cross-Validation Results for Manual Hyperparameters

The model evaluation shows decent performance with an accuracy of 79.89%, meaning the model correctly predicted churn about 80% of the time. The precision score of 0.6270 suggests that when the model predicts churn, it is accurate 63% of the time, but the recall of 0.5387 indicates it misses around 46% of the actual churn cases. This demonstrates that the model is conservative in predicting churn and misses a significant portion of churners. The F1 score of 0.5795 reflects a balance between precision and recall, but the model could improve in identifying more churn cases. The ROC AUC score of 0.8351 suggests that the model can effectively distinguish between churn and non-churn customers, but there's still room for improvement.

When evaluated using cross-validation, the model shows similar metrics, with a slight improvement in precision (0.6628) and accuracy (0.7996). However, the recall remains low at 0.5015, highlighting that the model is still missing a considerable number of churn cases. The F1 score of 0.5707 and ROC AUC score of 0.8378 further confirm that while the model performs well, improving recall remains a key area for development to capture more churn cases.

Figure 11: Training and Validation Loss Curves for Manual Hyperparameters

5.5.2 Loss Curves

The loss curve shows a sharp initial drop in both training loss and validation loss, indicating that the model is quickly learning. However, the training loss decreases faster than the validation loss, suggesting that the model fits the training data well but struggles to generalize to unseen data. As training continues, both losses stabilize, but the training loss remains lower than the validation loss, which is a sign of potential overfitting. This could be improved with techniques like regularization or early stopping to help the model generalize better to new data.

5.5.3 Confusion Matrix

The confusion matrix reveals that the model performs well in predicting non-churning customers, correctly predicting 919 true negatives and 195 true positives. However, the model makes 116 false positives and 167 false negatives, meaning it incorrectly predicts churn for some customers who do not churn and misses many actual churn cases. To improve the model, focusing on reducing false negatives is essential to ensure better accuracy in predicting churn.

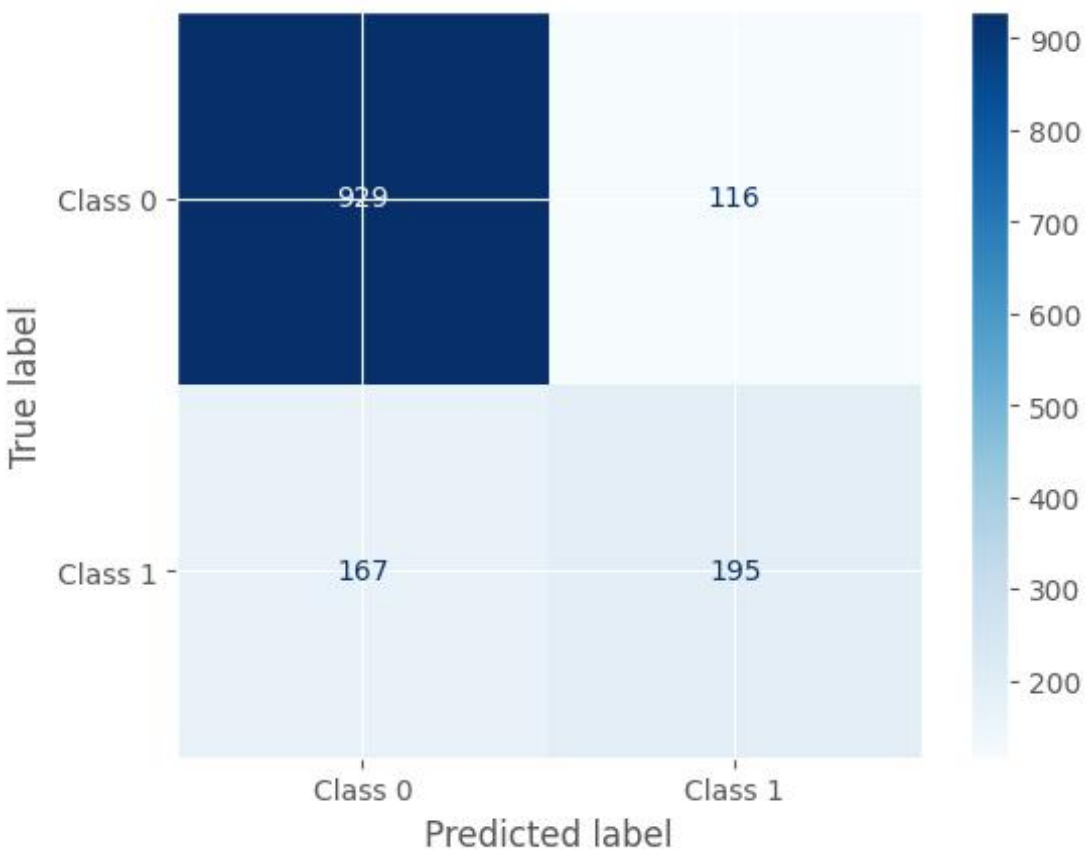

Figure 12: Confusion Matrix for Manual Hyperparameters

5.5.4 ROC Curve and AUC Score

The ROC curve indicates that the model is effective at distinguishing between churned and non-churned customers, with an AUC score of 0.84. This is a strong result, suggesting the model can differentiate well between the two classes. The optimal threshold for predicting churn is set at 0.37, which helps balance sensitivity (recall) and specificity (precision). The curve's distance from the diagonal line shows that the model performs much better than random guessing, confirming that the model is effectively identifying churn cases.

5.5.5 Precision-Recall Curve

The Precision-Recall curve shows that as recall increases, precision decreases, indicating that the model is trading off precision for recall to identify more churn cases. This is a typical trade-off, where trying to identify more churned customers increases false positives, which lowers precision. The AUC of 0.64 reflects a moderate ability to balance precision and recall, but there's still room for improvement. Reducing false positives and improving recall would help the model perform better in capturing churn cases.

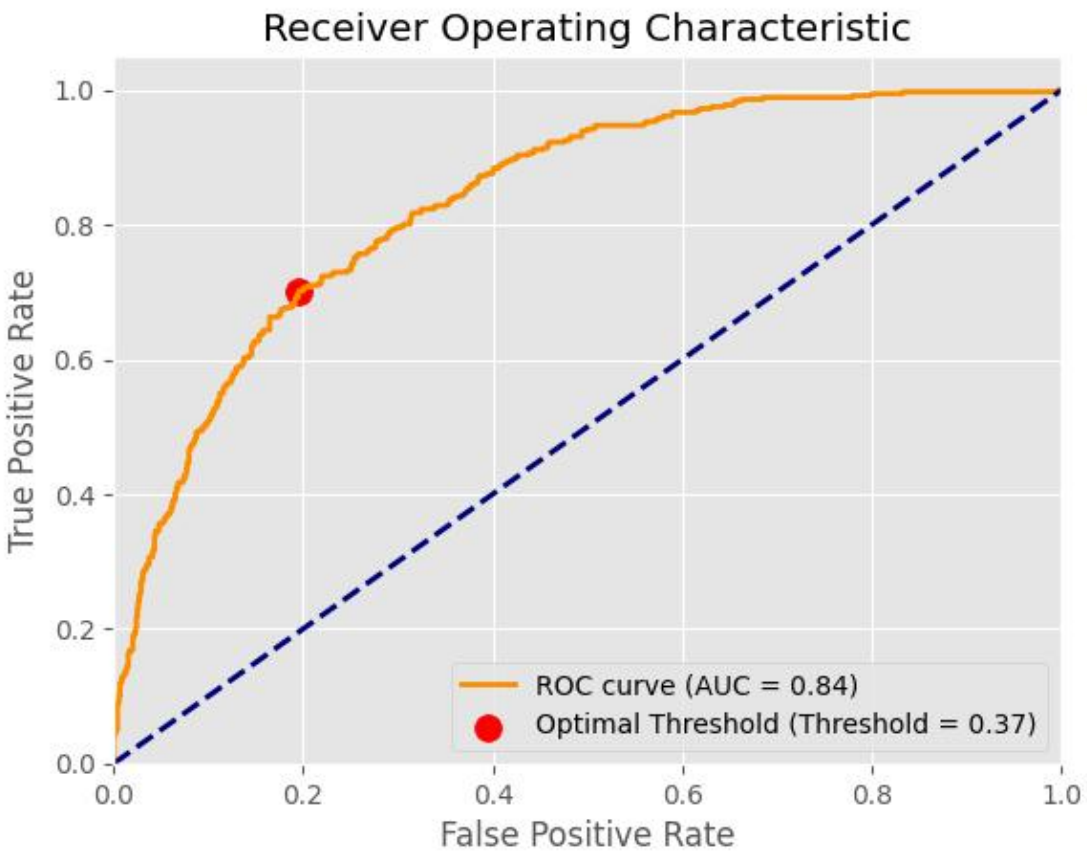

Figure 13: ROC Curve for Manual Hyperparameters

5.6 Model Tuning Results and Analysis

The hyperparameter tuning for the neural network model was conducted using Optuna to find the best combination of hyperparameters. After defining a search space for hyperparameters such as learning rate, batch size, hidden layer size, number of hidden layers, dropout rate, epochs, and weight decay, Optuna ran 50 trials to minimize the validation loss. The best-performing hyperparameters were a learning rate of 0.00062, a batch size of 94, 94 hidden neurons, 1 hidden layer, a dropout rate of 0.316, 23 epochs, and a weight decay of 0.000898. These optimized settings are expected to improve the model's ability to predict churn more accurately.

5.6.1 Optimization History Plot

The optimization history plot shows how the tuning process unfolded across 50 trials. Early in the process, there is a sharp drop in the objective value, indicating that initial hyperparameter choices made significant improvements to the model. After this initial drop, the objective value stabilizes, and later trials show minimal changes, suggesting the optimization process quickly converged to a good set of hyperparameters. The red line representing the best objective value remains stable after the initial improvement, confirming that the best hyperparameters were found early in the process. This plot demonstrates that the tuning process was both efficient

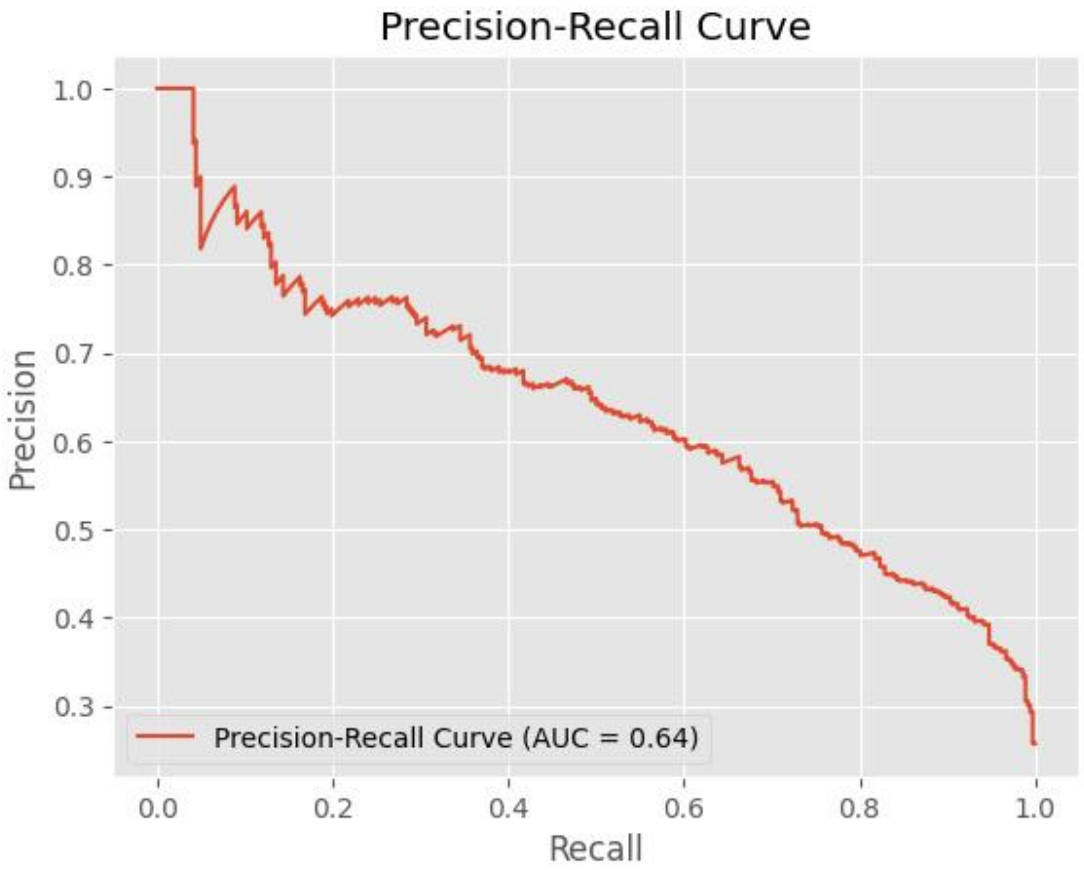

Figure 14: Precision-Recall Curve for Manual Hyperparameters

and effective.

5.6.2 Slice Plot

The slice plot illustrates the impact of different hyperparameters on the model's performance. The batch size does not seem to have a strong influence, as the performance is spread evenly across the range. The dropout rate shows some fluctuations, but no clear pattern emerges, suggesting it doesn't significantly improve model performance. The number of epochs is more impactful, with moderate numbers of epochs leading to better results. The hidden size has a noticeable effect on performance, especially for values between 50 and 100, while the learning rate also plays a significant role, with lower rates yielding better results. The number of hidden layers appears to have a minimal effect, and weight decay shows little variation, indicating it has a smaller impact on the overall model performance. The most influential hyperparameters for improving model performance were learning rate and hidden size, while batch size and dropout rate had a lesser effect.

```python
# Set the random seed
set_seed(DEFAULT_SEED)

# Define the hyperparameter search space
search_space = {
    "lr": [1e-5, 1e-2],
    "batch_size": [16, 128],
    "hidden_size": [16, 128],
    "num_hidden_layers": [1, 3],
    "dropout_rate": [0.2, 0.5],
    "epochs": [10, 100],
    "weight_decay": [1e-5, 1e-3],
}

# hyperparameter tuning and save the best study
tune_and_save(
        model_type="neural_network",
        model_name="neural_network_study",
        X_train=X_train,
        y_train=y_train,
        X_val=X_val,
        y_val=y_val,
        n_trials=50,
        direction="minimize",
        search_space=search_space
    )
```

[35] Python

Figure 15: Model Tuning Using Optuna

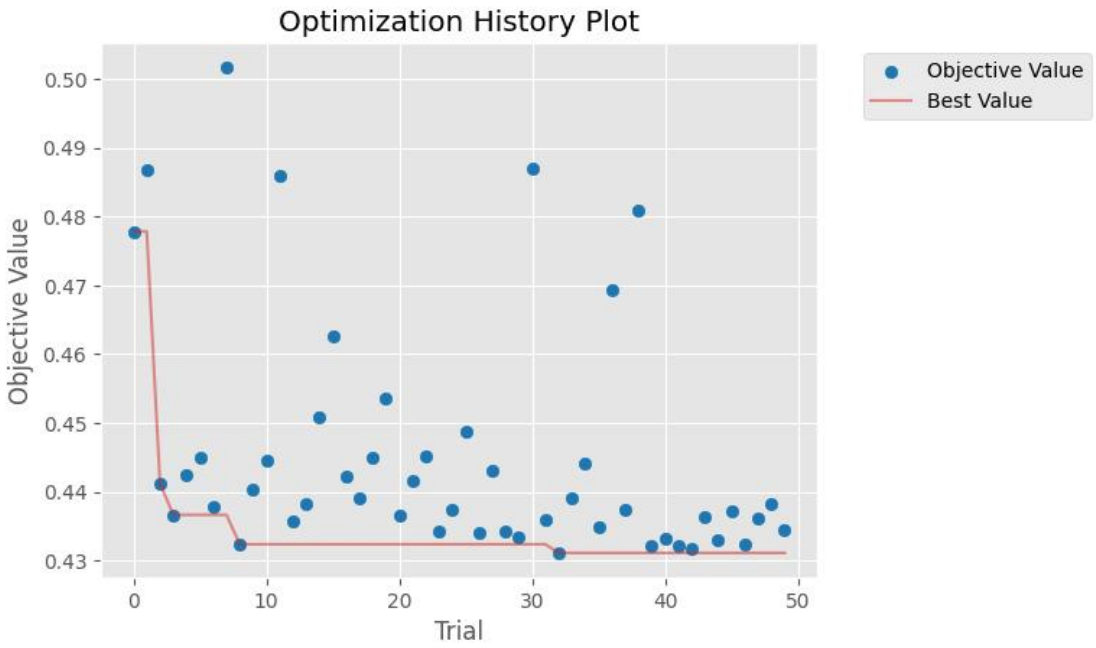

Figure 16: Optimization History Plot

5.6.3 Hyperparameter Importances

The hyperparameter importance plot shows that epochs have the highest importance score (0.39), suggesting that the number of epochs is crucial for the model's ability to learn and generalize effectively. Weight decay (0.28) also plays a significant role in preventing overfitting, while batch

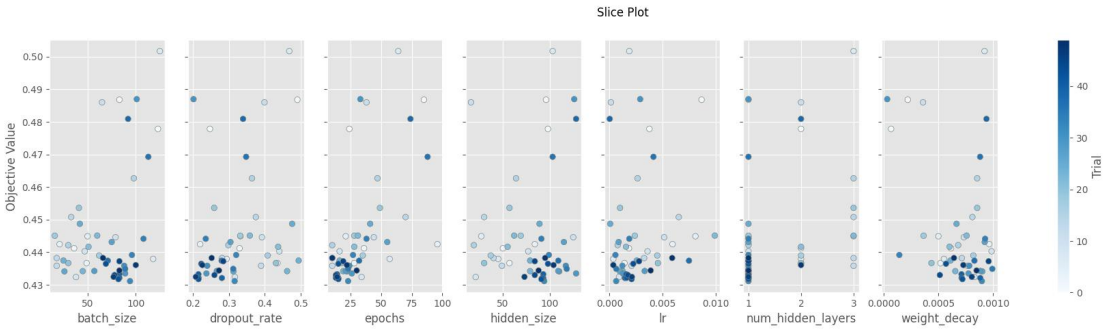

Figure 17: Slice Plot

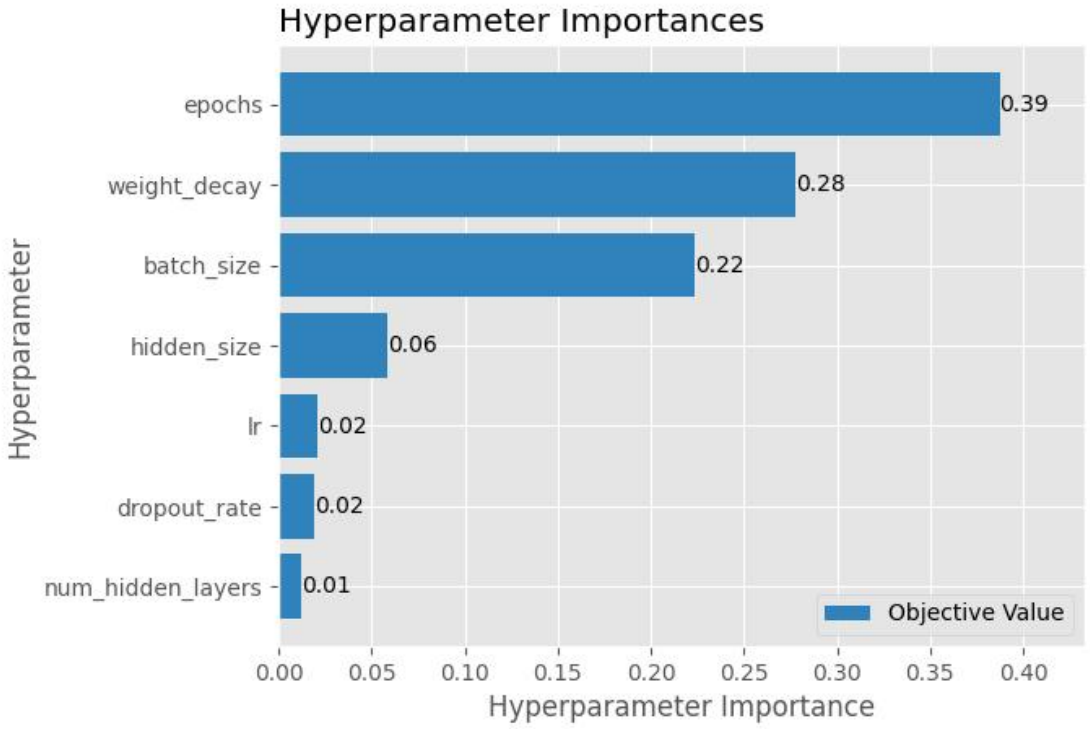

Figure 18: Hyperparameter Importances Plot

size (0.22) impacts the model's training efficiency and generalization ability. Hidden size has a lower importance score (0.06), showing that the number of neurons in the hidden layers is less significant compared to other hyperparameters. Both learning rate and dropout rate have low importance scores (0.02), suggesting that adjusting these parameters does not lead to substantial improvements in performance. Finally, the number of hidden layers (0.01) has the least impact, indicating that the depth of the model does not significantly affect performance. Overall, the plot suggests that tuning epochs, weight decay, and batch size would yield the most significant improvements in model performance, while other parameters have a smaller impact.

5.7 Optimal Training Results and Analysis

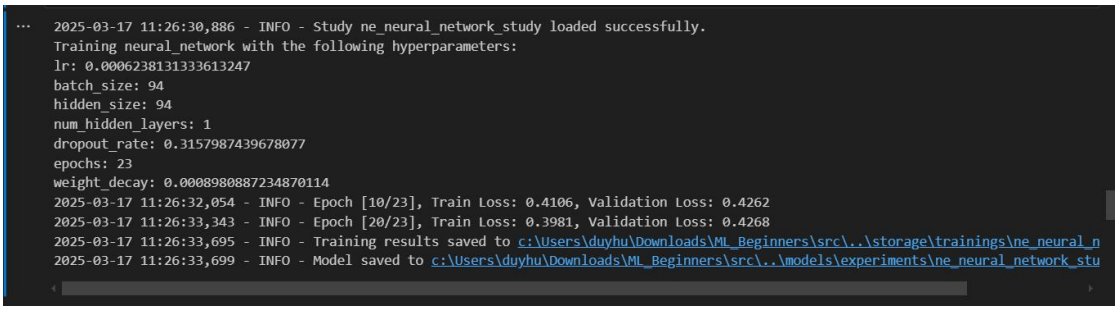

Figure 19: Training Log for Optimal Hyperparameters

The model was trained using the best hyperparameters identified through Optuna. After 10 epochs, the training loss was 0.4106, and the validation loss was 0.4262. By epoch 20, the training loss decreased to 0.3981, while the validation loss remained nearly constant at 0.4268. This suggests that the model is learning over time, but the validation loss isn't improving much, indicating that the model might be nearing convergence. After completing the training, the model and its results were saved for further evaluation.

5.7.1 Evaluation and Cross-Validation

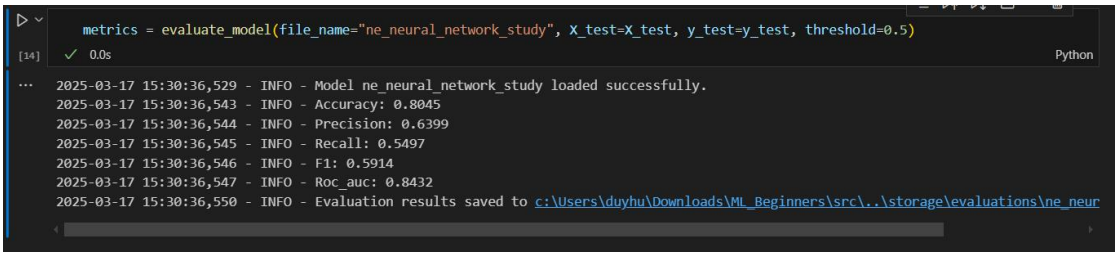

Figure 20: Model Evaluation and Cross-Validation Results for Optimal Hyperparameters

When evaluated on the test dataset, the model performed well with an accuracy of 80.45%. Precision was 0.6399, indicating the model correctly predicts churn 64% of the time when it makes a churn prediction. The recall was 0.5497, meaning the model identifies about 55% of actual churn cases but misses a considerable portion. The F1 score was 0.5914, balancing precision and recall, suggesting reasonable performance but with room for improvement, particularly in recall. The ROC AUC score of 0.8432 shows that the model is good at distinguishing between churned and non-churned customers. These metrics indicate solid performance, though improving recall would capture more churn cases.

Cross-validation results show an average accuracy of 0.8008, closely matching the test set evaluation. Precision improved slightly to 0.6584, indicating the model is more reliable when predicting churn across multiple folds. However, recall still remains at 0.5190, meaning the model still misses many churned customers. The F1 score was 0.5795, showing a moderate trade-off between precision and recall. The ROC AUC was 0.8426, confirming that the model remains strong in

distinguishing churn from non-churn cases. These results suggest the model performs well but still needs further tuning, especially to improve recall.

5.7.2 Loss Curves

Figure 21: Training and Validation Loss Curves for Optimal Hyperparameters

The loss curve shows the training loss (blue line) and validation loss (orange line) over 23 epochs. Initially, both losses decrease sharply, indicating that the model is learning quickly and fitting the training data well. However, the validation loss remains higher than the training loss, suggesting the model is not generalizing well to unseen data. As training continues, the losses level off, but the gap between the training loss and validation loss indicates potential overfitting. The fluctuating validation loss points to difficulty in generalizing. This suggests that regularization techniques or early stopping might be needed to improve the model's generalization.

5.7.3 Confusion Matrix

The confusion matrix shows that the model predicted 919 non-churning customers (True Negatives) and 195 churned customers (True Positives) correctly. However, it made 116 false positive errors, where it predicted churn when the customer did not churn, and 167 false negatives, where it failed to predict churn for customers who actually churned. While the model does a good job

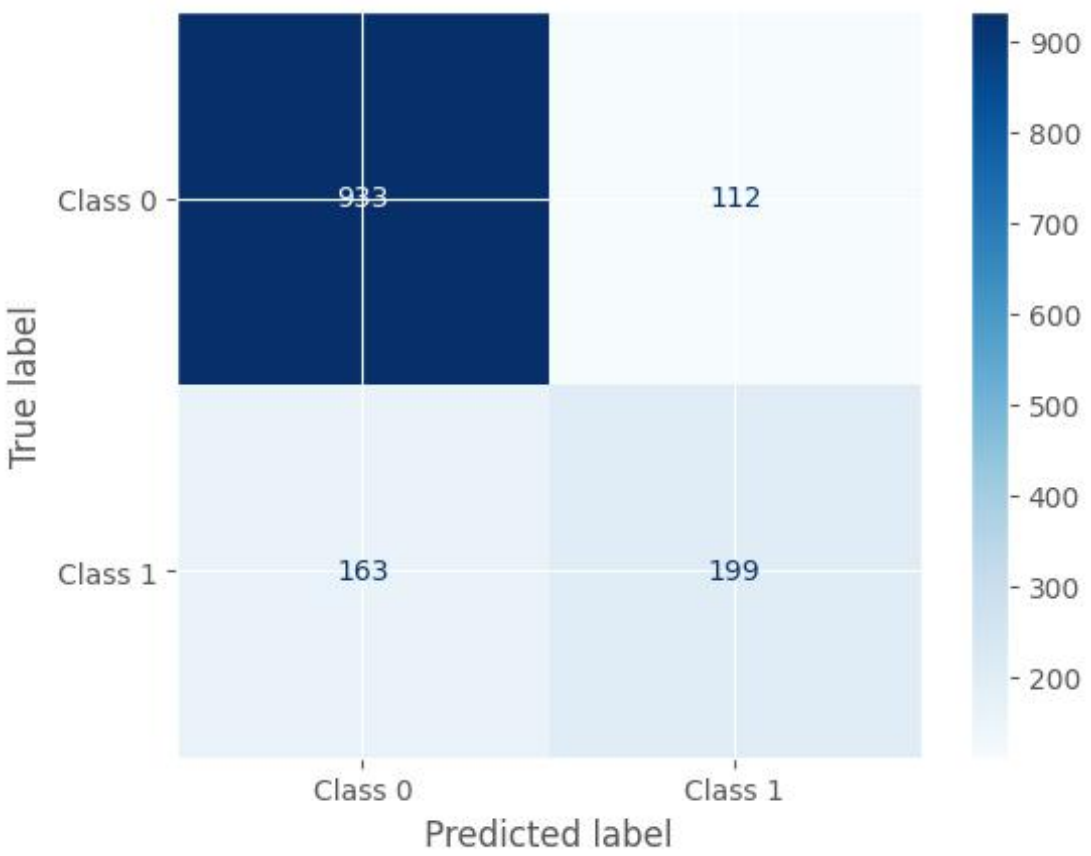

Figure 22: Confusion Matrix for Optimal Hyperparameters

identifying non-churning customers, it misses many churn cases. Reducing these false negatives is essential to improving the model's accuracy in predicting churn.

5.7.4 ROC Curve and AUC Score

The ROC curve shows the model's ability to distinguish between churned and non-churned customers. The AUC of 0.84 indicates the model is effective at differentiating between the two classes. The optimal threshold of 0.36, marked by the red dot on the curve, helps balance recall and precision. The curve is far from the diagonal line (representing random guessing), showing that the model is much better than random at identifying churn. The ROC curve confirms that the model is effective at minimizing errors while predicting churn.

5.7.5 Precision-Recall Curve

The Precision-Recall curve evaluates the model's performance with respect to churn cases (the positive class). The AUC of 0.64 shows that the model balances precision and recall moderately well. As recall increases (the model tries to capture more churn cases), precision drops significantly, which is typical when trying to capture as many churn cases as possible but at the cost

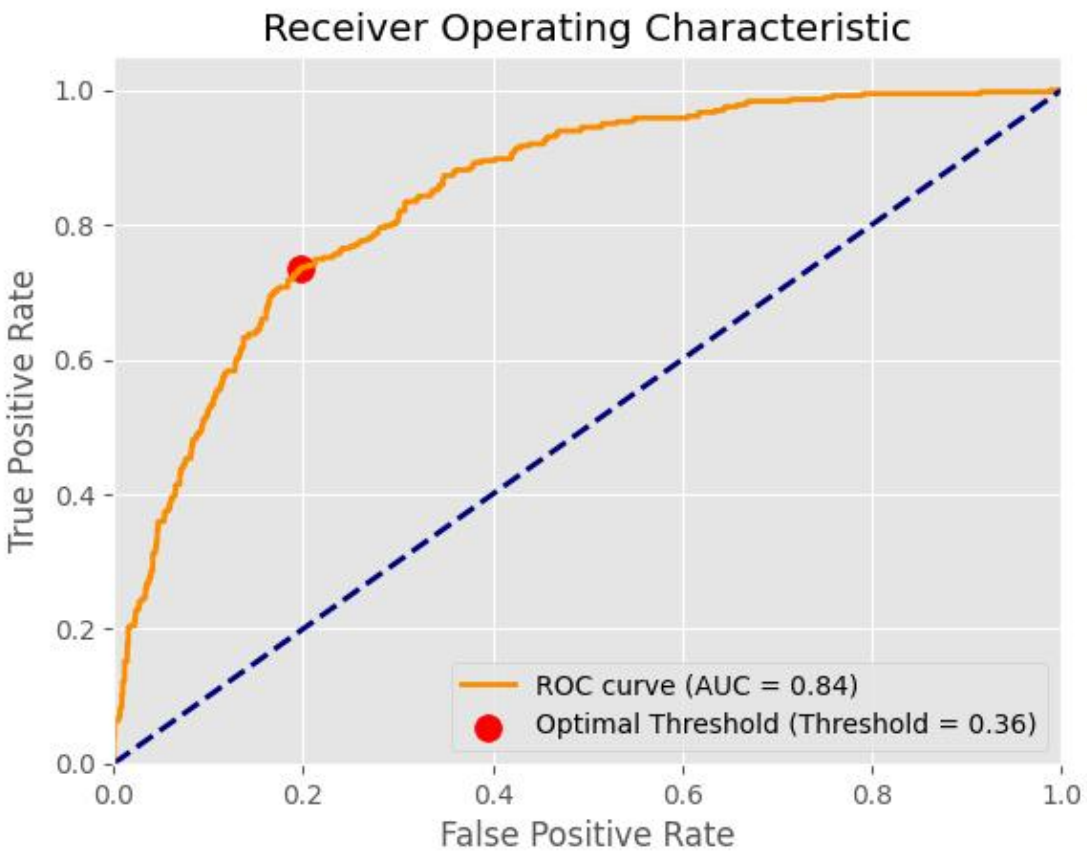

Figure 23: ROC Curve for Optimal Hyperparameters

of predicting more false positives. The trade-off between precision and recall is evident, suggesting that improving recall while reducing false positives could lead to better model performance. Overall, while the model is somewhat effective at churn prediction, there's room to improve, especially in minimizing false positives and increasing recall.

5.8 Models Comparison

When comparing the performance of the models trained with manual versus optimal hyperparameters, we can see that the optimal hyperparameters led to slight improvements in most performance metrics. The accuracy increased marginally from 79.89% to 80.45%, indicating a small but positive effect on the overall prediction accuracy. Precision also saw a modest boost, rising from 0.6270 to 0.6399, meaning the model became slightly more reliable in predicting churn. While recall showed a small improvement, moving from 0.5387 to 0.5497, it still remains relatively low, signaling a key area for further optimization. The F1 score also improved from 0.5795 to 0.5914, showing a better balance between precision and recall. The ROC AUC score increased slightly from 0.8351 to 0.8432, indicating the optimal hyperparameters helped the model better distinguish between churned and non-churned customers. Cross-validation results confirmed these improvements, with the model trained with optimal hyperparameters perform-

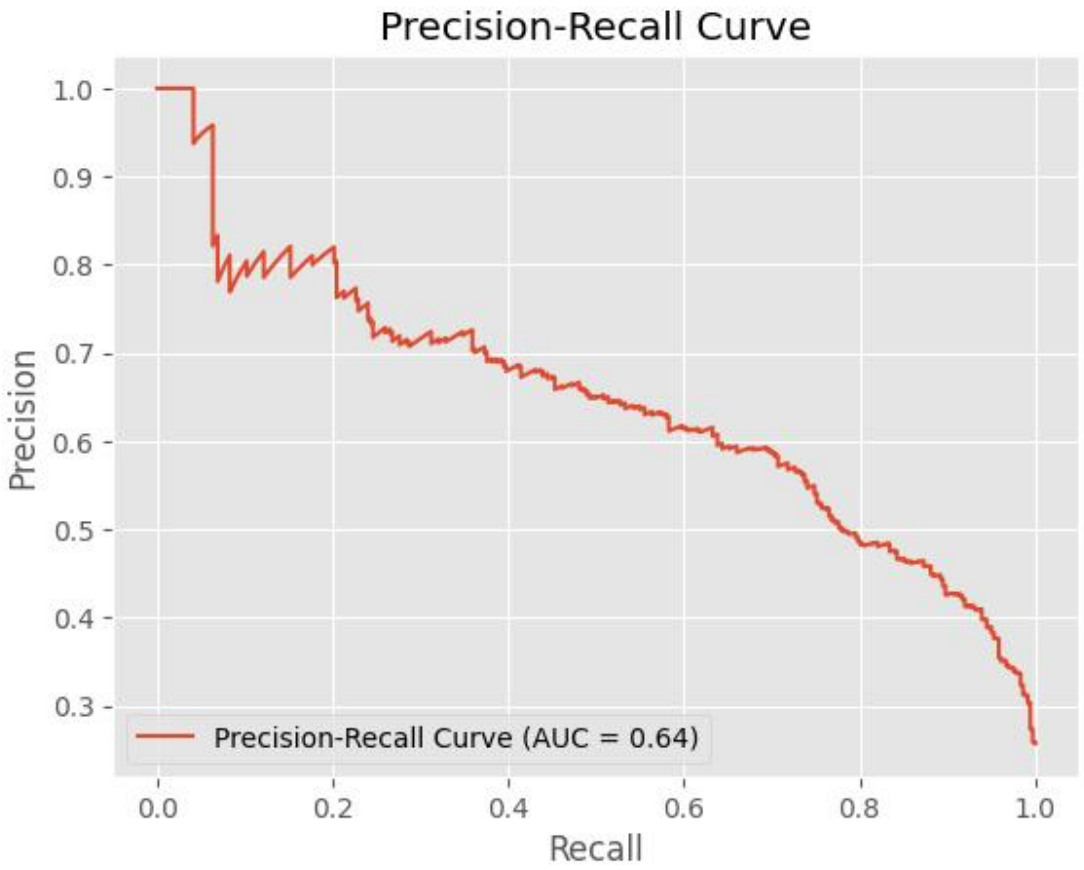

Figure 24: Precision-Recall Curve for Optimal Hyperparameters

ing slightly better in terms of accuracy, precision, recall, F1 score, and ROC AUC. Despite the improvements, recall continues to be a challenge, and the models, while strong, could still be further fine-tuned to capture more churned customers.

6 Naive Bayes

6.1 Model Implementation

The Naive Bayes model implemented for predicting customer churn in the Telco dataset is a mixed model designed to handle both categorical and numeric features. The model begins with parameter validation, ensuring proper handling of Laplace smoothing for categorical features and variance smoothing for numeric features. During the fitting process, the dataset is split by class labels to calculate class priors and extract categorical and numeric feature indices. For categorical features, probabilities are calculated using Laplace smoothing, and for numeric features, the model computes class-wise means and variances, adding a small variance boost to prevent numerical instability. In the prediction phase, the model computes likelihoods for categorical features based on precomputed probabilities and applies the Gaussian probability density function for numeric features. To ensure stability, log-likelihoods are used, combining the contributions from both feature types and class priors to make predictions. The implementation integrates key machine learning practices like input validation and model persistence while leveraging the simplicity and efficiency of the Naive Bayes approach for churn prediction.

6.2 Training Implementation

The training implementation for the Naive Bayes model follows a structured process to handle both categorical and numeric features effectively. The first step is input validation, ensuring the provided data matches expectations. The model verifies the dimensions of the feature matrix and checks that the categorical feature mask aligns with the number of features. It also calculates the class priors by counting occurrences of each class in the target variable, which helps in determining the baseline probability for each class before considering the feature values.

For categorical features, the model computes conditional probabilities using a frequency-based approach with Laplace smoothing to prevent zero probabilities when unseen categories appear in the prediction phase. It creates a nested dictionary structure where each class maps to its respective categorical features and their value counts. For numeric features, the model calculates the mean and variance for each feature per class, ensuring stability by adding a small variance boost proportional to the maximum variance across features. This variance smoothing prevents numerical instability when feature variances are very small. These computations ensure the model captures the underlying data distribution effectively, preparing it to make robust predictions across mixed data types.

6.3 Model Tuning Implementation

The model tuning process for the Naive Bayes classifier involves using GridSearchCV to optimize the hyperparameters, ensuring the model achieves the best performance across multiple metrics. In this implementation, the laplace smoothing parameter is the focus of the tuning process, which addresses the issue of zero probabilities for unseen categorical values by adding a constant value to all feature counts. A wide range of values is tested, starting from 0.5 and gradually increasing to 200. This comprehensive range allows the model to explore subtle adjustments as well as more aggressive smoothing, helping to identify the optimal balance between bias and variance. Additionally, a Pipeline is used to streamline the process, ensuring the Naive Bayes model is properly integrated into the tuning workflow and preventing data leakage by ensuring transformations happen inside the cross-validation loop.

The grid search is performed using k-fold cross-validation, where the dataset is split into k folds, ensuring the model is trained and validated on different subsets of data to prevent overfitting and assess generalizability. The evaluation is conducted across multiple metrics, including accuracy, F1 macro, precision macro, and recall macro, providing a well-rounded assessment of the model's performance. The refit parameter is set to optimize for accuracy, meaning the final model will be selected based on the best accuracy score. This approach ensures that the model tuning process is thorough, identifying the laplace smoothing value that leads to the most accurate predictions while balancing other important performance metrics. By combining cross-validation, grid search, and multiple evaluation metrics, the model tuning process aims to create a robust classifier that generalizes well to unseen data.

6.4 Training Process

The training process for the Naive Bayes model begins by preparing the data and ensuring that it aligns with the expected input format. The fit method is responsible for learning the parameters required for making predictions. First, the training data is validated to ensure its integrity, and the categorical feature mask is checked to match the number of features in the dataset. This mask is crucial, as it determines which features should be treated as categorical and which should be treated as numeric. The model then identifies the indices of categorical and numeric features, which allows it to handle these two types of data differently. Afterward, the training labels are analyzed to determine the unique classes present in the dataset, and the prior probabilities for each class are calculated by dividing the count of each class by the total number of samples. These priors represent the likelihood of encountering each class in the absence of any other information.

For categorical features, the model calculates the probability distribution of each feature value within each class using Laplace smoothing. This technique prevents zero probabilities when certain feature values are absent from the training set for a given class. The smoothing parameter ensures that every feature value has a small probability, even if unseen in the training data. The probabilities are stored in a nested dictionary structure, where each class maps to another dictionary containing feature indices, which in turn map to dictionaries of feature values and their associated probabilities. This structure makes it efficient for the model to retrieve probabilities during the prediction phase. The numeric features are handled differently, assuming a normal (Gaussian) distribution. For each numeric feature, the model computes the mean and variance per class, adding a variance smoothing factor to avoid numerical instabilities caused by very small variances.

Finally, once the categorical probabilities and numeric feature statistics have been computed, the model saves these parameters for later use in prediction. The training process ensures that the model has learned a set of class priors, conditional probabilities for categorical features, and Gaussian distributions for numeric features. These learned parameters represent the probability distributions needed to classify new data points accurately. Additionally, the model records the total number of features and marks itself as fitted, ensuring that predictions can only be made after the training process has been completed successfully. By carefully separating the handling of categorical and numeric features, the model captures the distinct statistical properties of each type, leading to a more accurate and reliable Naive Bayes classifier tailored to mixed datasets

6.5 Manual Training Results and Analysis

For our Naive Bayes model, we manually selected the Laplace smoothing parameter to be 5, which plays a crucial role in handling zero-frequency problems. In traditional Naive Bayes, if a particular feature value does not appear in the training set for a given class, the model assigns a probability of zero to any instance containing that feature, which can severely affect predictions. Laplace smoothing addresses this by adding a constant value (in this case, 5) to the numerator of the probability calculation for each feature-class combination, ensuring no probability is ever zero. Choosing a higher smoothing value reduces the model's sensitivity to rare events and prevents overfitting, especially when dealing with categorical data. Setting the smoothing parameter to 5 strikes a balance between ensuring unseen feature values have a reasonable probability and avoiding excessive smoothing that could blur meaningful patterns. This choice reflects careful consideration of the model's ability to generalize better across unseen data while remaining responsive to the underlying distribution in the training set.

6.5.1 Evaluation and Cross Validation

```
print_results_table([scores_nb_own], headings=["our_nb"])
✓ 0.0s
```

	our_nb
avg. accuracy:	0.7691
avg. f1_macro:	0.7306
avg. precision_macro:	0.7203
avg. recall_macro:	0.7576
avg. fitting time:	0.4040
avg. score time:	0.9634

Figure 25: Model Evaluation and Cross-Validation Results of Naive Bayes Model

The results of the model training indicate a solid performance with an average accuracy of 0.7691, meaning the model correctly classifies about 76.91% of instances in the training set on average across cross-validation folds. Accuracy alone doesn't paint the full picture, so additional metrics offer deeper insights into the model's effectiveness. The F1 macro score of 0.7306 balances precision and recall across all classes, making it particularly useful when dealing with imbalanced datasets. A macro-averaged F1 score treats each class equally, ensuring that smaller or less frequent classes contribute equally to the overall performance evaluation.

Furthermore, the model's precision macro of 0.7203 suggests that when the model predicts a positive instance for any class, it is correct 72.03% of the time on average. Precision is crucial in scenarios where minimizing false positives is important. On the other hand, the recall macro of 0.7576 shows that the model correctly identifies about 75.76% of all true positive instances across classes, indicating its ability to capture most relevant instances without overlooking too many. The training process was also efficient, with an average fitting time of 0.4040 seconds and an average scoring time of 0.9634 seconds. These results highlight the model's robustness with the chosen Laplace smoothing parameter, providing a well-balanced performance in terms of accuracy, precision, recall, and computational efficiency. The combination of these metrics

suggests that the model generalizes well across different folds of the dataset, making it a reliable choice for the given task.

6.5.2 Confusion Matrix

The confusion matrix shows the performance of the Naive Bayes model with a manually chosen Laplace smoothing parameter of 5. The matrix indicates that the model correctly predicted 2,397 positive instances (True Positives) and 983 negative instances (True Negatives). However, there were 687 instances where the model incorrectly classified negative cases as positive (False Positives) and 152 instances where it failed to detect positive cases, misclassifying them as negative (False Negatives). The model performs well in identifying positive cases, as shown by the high number of True Positives, but the relatively higher number of False Positives suggests that the model leans slightly towards predicting positives. This may be influenced by the chosen smoothing value, which can affect probability estimates and decision boundaries.

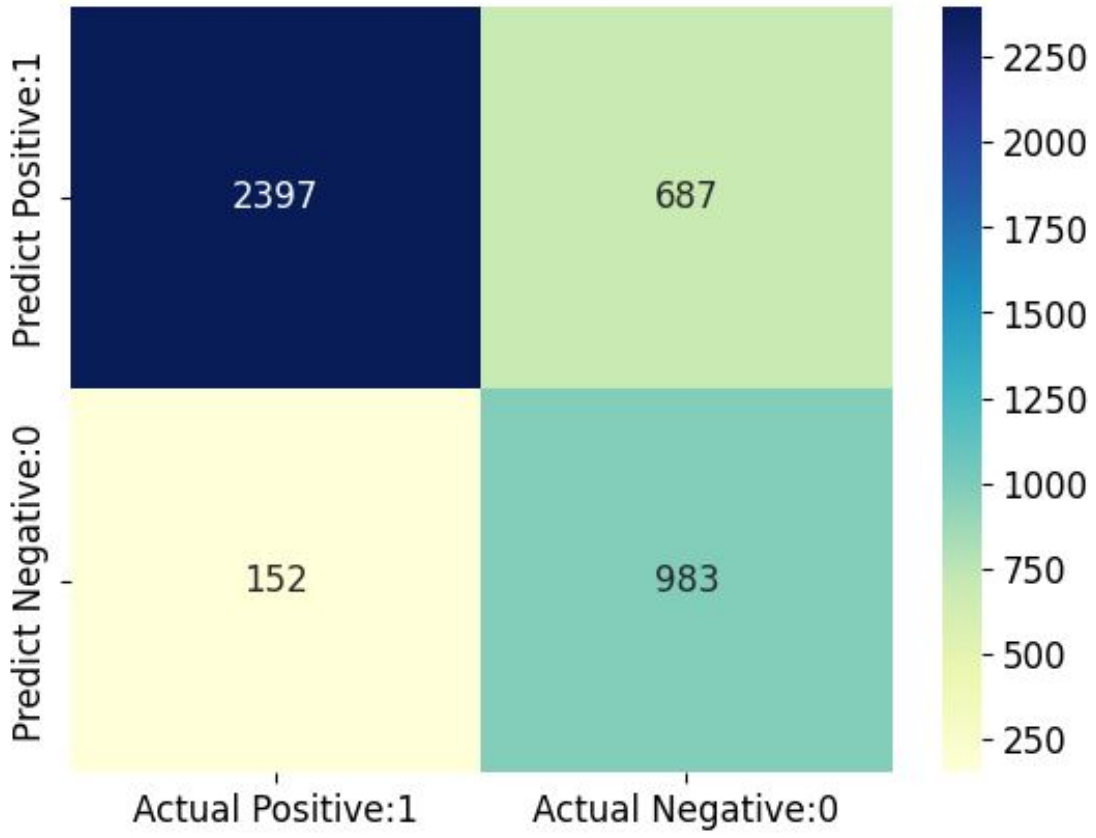

Figure 26: Confusion Matrix for Naive Bayes Model

6.5.3 ROC Curve and AUC Score

The Area Under the Curve (AUC) score of 0.827 indicates that the Naive Bayes model, with a Laplace smoothing value of 5, demonstrates strong discriminative ability between positive and

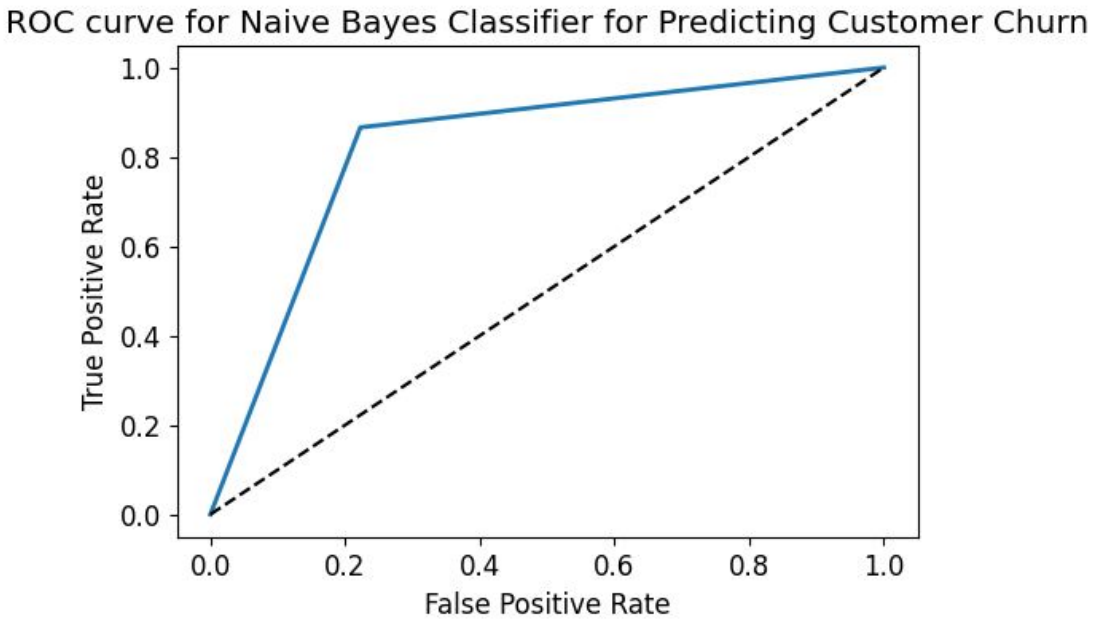

Figure 27: ROC Curve for Naive Bayes Model

```
ROC_AUC = roc_auc_score(y_train, predict)
print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

✓ 0.0s

```
ROC AUC : 0.8217
```

Figure 28: AUC Score for Naive Bayes Model

negative classes. An AUC score ranges from 0 to 1, where 0.5 implies random guessing, and 1.0 represents perfect classification. A score of 0.827 shows that the model can correctly distinguish between classes about 82.7% of the time, suggesting a well-performing model with good balance between sensitivity and specificity. While the model performs reliably, there is still room for optimization to push the score closer to 1.0, potentially by fine-tuning the smoothing parameter or exploring other feature engineering techniques.

6.5.4 Precision-Recall Curve

The Precision-Recall (PR) curve illustrates the trade-off between precision and recall for the Naive Bayes model with Laplace smoothing set to 5. From the plot, we can see that as recall increases, precision steadily declines. Initially, at very low recall, precision is close to 1.0, indicating that the model makes very few false positives when predicting the positive class. However, as recall rises towards 1.0, precision drops significantly, reflecting that the model captures more true positives at the cost of increasing false positives. This behavior is typical for models where higher recall comes with a trade-off in precision. Analyzing this curve is essential to understanding model performance in imbalanced datasets or scenarios where prioritizing precision or recall

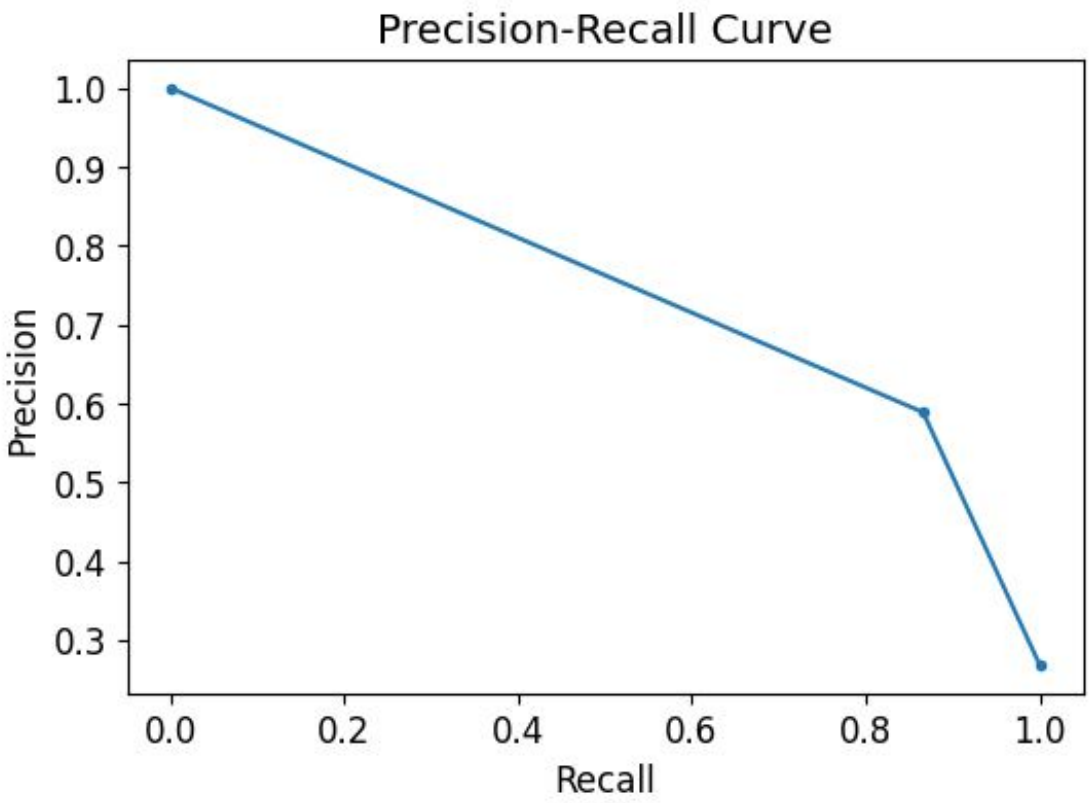

Figure 29: Precision-Recall Curve for Naive Bayes Model

impacts the task differently. The gradual slope indicates a relatively stable trade-off, suggesting that the model maintains reasonable precision as recall improves.

6.6 Model Optimization with Hyperparameter Tuning

In this model training process, hyperparameter tuning was conducted using GridSearchCV to identify the optimal value for Laplace smoothing in the Naive Bayes model. Laplace smoothing, also known as additive smoothing, addresses the problem of zero probabilities in Naive Bayes classification, which occurs when the model encounters categorical features in the test set that were not present in the training set. Without smoothing, unseen categories result in a probability of zero, which effectively nullifies the entire likelihood for that class. By adding a constant value to each feature count, Laplace smoothing prevents zero probabilities, ensuring the model can still make meaningful predictions for previously unseen feature values. This technique is especially useful in datasets with high cardinality or sparse categorical features, where some combinations of features and classes may not appear during training.

In the provided code, a Pipeline was created to streamline the training process by combining data preprocessing and model training into a single workflow. The Naive Bayes model was wrapped in this pipeline with a hyperparameter called laplace smoothing, which determines the amount of smoothing applied. The categorical feature mask was set to True for all features, indicating that

```
mask = [True]*X_train.shape[1]
pipe = Pipeline([
    ('m', NaiveBayesModel(categorical_feature_mask=mask))
])
parameters = {
    "m__laplace_smoothing": [0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 8, 9, 10, 15, 20, 25, 50, 100, 200]
}
grid = GridSearchCV(pipe, parameters,cv=kfold, scoring=["accuracy", "f1_macro", "precision_macro", "recall_macro"], refit="accuracy")
grid.fit(X_train, y_train)
✓ 3m 6.2s
```

Figure 30: Model Tuning using GridSearchCV

every feature in the dataset is treated as categorical. The GridSearchCV function systematically searched through a range of possible values for laplace smoothing — from 0.5 to 200 — ensuring the model explored both low and high smoothing values. This comprehensive search aimed to balance the trade-off between bias and variance in the model's predictions.

To ensure robust evaluation, k-fold cross-validation was used, where the training set was split into multiple folds. In each iteration, the model was trained on k-1 folds and validated on the remaining fold, repeating the process k times. This not only prevented the model from memorizing the training data but also provided a more reliable estimate of its performance. The scoring was conducted across multiple metrics like Accuracy, F1-macro, Precision-macro and Recall-macro.

The refit="accuracy" argument ensured that after exploring all hyperparameter combinations, the final model would be trained using the hyperparameters that maximized accuracy. This allowed the model to optimize its overall correctness while also providing insights into other performance aspects through the additional scoring metrics.

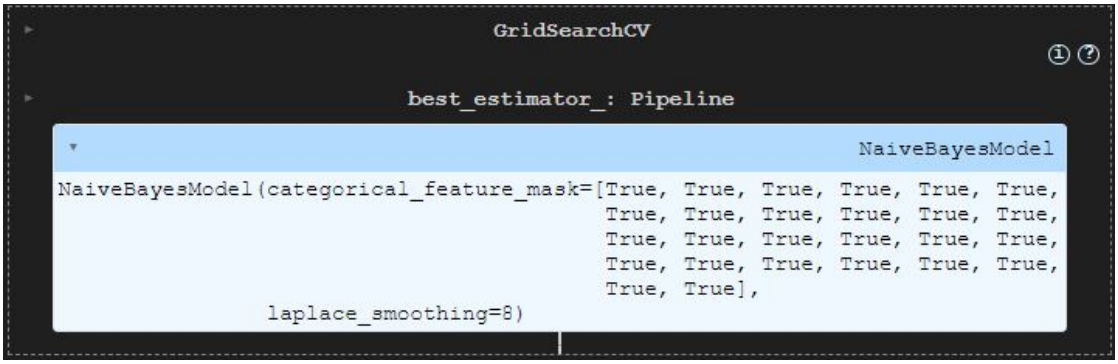

Figure 31: Model Tuning Result

After completing the hyperparameter search, the optimal value for Laplace smoothing was found to be 8. This value indicates that adding a smoothing factor of 8 resulted in the best accuracy across the cross-validation folds. The choice of this value suggests a balance between under-smoothing and over-smoothing: If the smoothing value is too low, the model becomes overly confident when encountering unseen categories, as it assigns zero probability to unobserved features, leading to poor generalization. On the other hand, if the smoothing value is too high, the model excessively dilutes the influence of observed data, resulting in overly flat probability distributions and making predictions less sensitive to meaningful patterns in the data.

By selecting 8 as the optimal smoothing value, the model effectively mitigated these risks. It

applied enough smoothing to handle unseen feature values without overshadowing the contributions of frequently occurring features. This value likely reflects the dataset's characteristics — possibly a moderate number of categorical features with occasional rare values — making a mid-range smoothing value the best choice. Additionally, the model benefited from a slight bias towards stability by avoiding extreme sensitivity to rare events.

The hyperparameter tuning process played a crucial role in enhancing the model's performance. Through cross-validation, the GridSearchCV procedure ensured that the selected smoothing value generalized well across different data splits, preventing the model from overfitting to specific patterns in the training set. The fact that 8 emerged as the optimal value demonstrates that moderate smoothing worked best for this dataset, ensuring a balance between generalization and predictive accuracy.

In summary, the use of GridSearchCV for hyperparameter tuning allowed for a thorough exploration of smoothing values, ultimately enhancing the Naive Bayes model's robustness. The selected smoothing value of 8 suggests that the model benefited from a moderate degree of smoothing, effectively handling unseen features while preserving meaningful patterns in the data. This tuning process not only optimized accuracy but also ensured the model was evaluated across multiple metrics, providing a well-rounded assessment of its performance. The result is a Naive Bayes classifier capable of making stable, reliable predictions even in the presence of previously unseen data points.

6.7 Optimal Training Results and Analysis

6.7.1 Evaluation and Cross Validation

```
print_results_table([scores_nb_own], headings=["our_nb"])
✓ 0.0s

                        our_nb
avg. accuracy:          0.7924
avg. f1_macro:          0.7224
avg. precision_macro:   0.7360
avg. recall_macro:      0.7139
avg. fitting time:      0.3296
avg. score time:        0.7202
```

Figure 32: Model Evaluation and Cross Validation after Tuning

The results after hyperparameter tuning with Laplace smoothing = 8 show that the Naive Bayes model achieved an average accuracy of 0.7924, meaning the model correctly classified about 79.24% of the instances across the cross-validation folds. This indicates that the model has learned meaningful patterns from the data and generalizes well to unseen examples. The average f1 macro score of 0.7224 reflects a balance between precision and recall across all classes, which is particularly important in cases of class imbalance, ensuring that both false positives and false negatives are accounted for.

The average precision macro of 0.7360 highlights the model's ability to avoid false positives, meaning when the model predicts a class, it is correct about 73.60% of the time on average

across classes. Similarly, the average recall macro of 0.7139 shows the model's ability to correctly identify positive instances, capturing about 71.39% of all actual positives across classes. The fitting and scoring times, averaging 0.3296 seconds and 0.7202 seconds respectively, demonstrate that the model is computationally efficient. Overall, these results indicate that the tuned Naive Bayes model effectively balances accuracy and generalization, making it well-suited for handling categorical data with a moderate degree of smoothing.

6.7.2 Confusion Matrix

The confusion matrix after tuning reveals the performance of the Naive Bayes model with Laplace smoothing set to 8. The model correctly predicted 2,513 positive instances (True Positives) and 821 negative instances (True Negatives). However, it also misclassified 571 negative instances as positive (False Positives) and 314 positive instances as negative (False Negatives). This indicates that the model performs well at identifying positive cases, with fewer false negatives, though there are still a notable number of false positives. The balance between true positives and true negatives suggests the model is making reasonably accurate predictions overall, aligning with the earlier reported accuracy and F1 score.

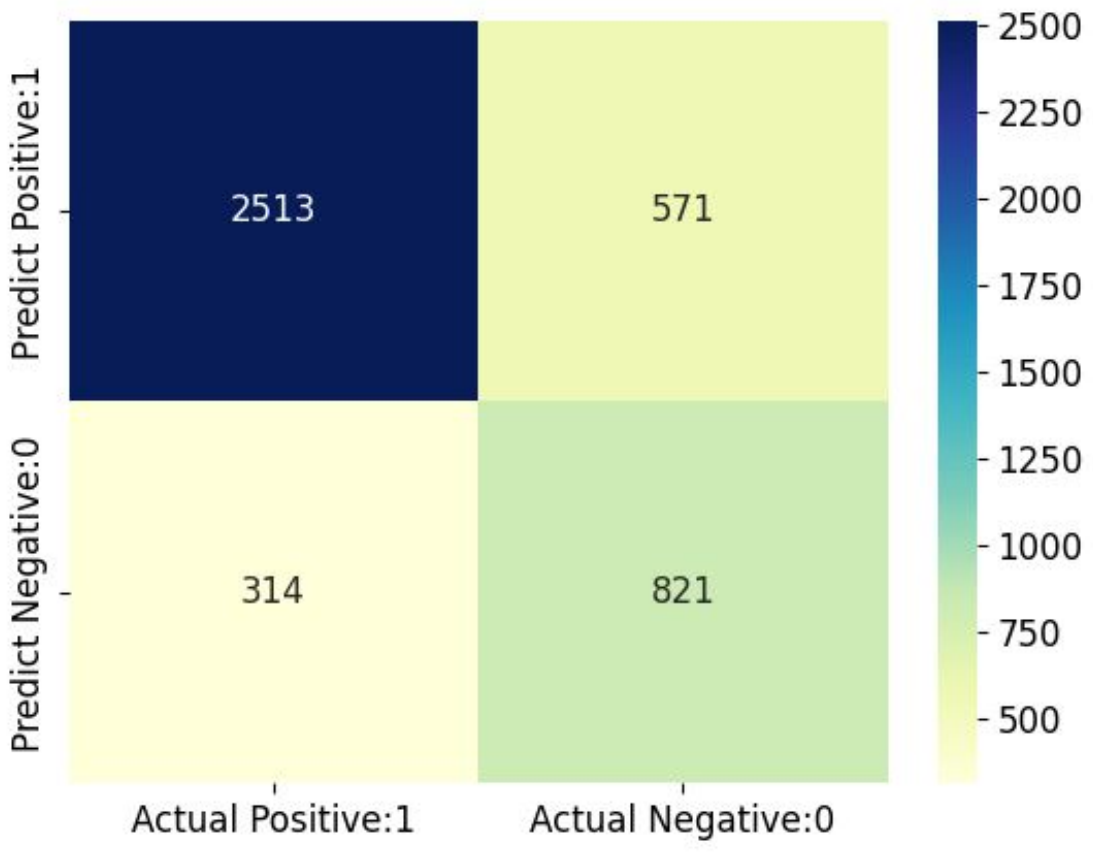

Figure 33: Confusion Matrix after Tuning

6.7.3 ROC Curve and AUC Score

The ROC curve for the Naive Bayes Classifier after tuning shows a solid balance between true positive and false positive rates, with an AUC score of 0.8325. This indicates that the model has a good ability to distinguish between customers who are likely to churn and those who are not. An AUC of 0.8325 means that if we randomly select one positive instance and one negative instance, there is an 83.25% chance that the model will correctly rank them. The curve's rise towards the top-left corner reflects strong performance, suggesting that the tuning process, particularly optimizing the Laplace smoothing parameter, has enhanced the model's predictive accuracy.

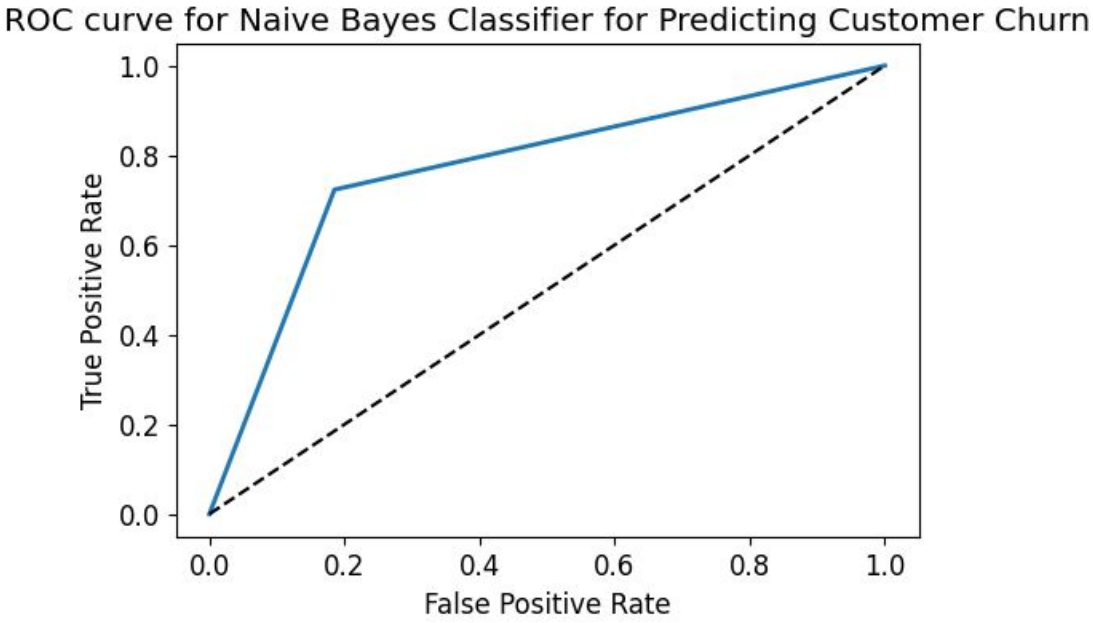

Figure 34: ROC Curve after Tuning

6.7.4 Precision-Recall Curve

The Precision-Recall (PR) curve after tuning illustrates the trade-off between precision and recall for the Naive Bayes classifier. As recall increases, precision steadily decreases, which is a common pattern in models that aim to balance false positives and false negatives. Initially, when recall is low, precision is very high, meaning the model is confident in its positive predictions. However, as recall increases, the model captures more true positives but at the cost of increased false positives, resulting in lower precision. This behavior reflects the model's capacity to detect churned customers while trying to maintain accuracy in its predictions, showing that tuning has helped in achieving a more balanced performance.

6.8 Model Comparison

In this section, we will compare our model after Tuning with two others model that already implemented in the library. The performance comparison between the tuned Naive Bayes model and the two other implementations — Categorical Naive Bayes (CategoricalNB) and Random

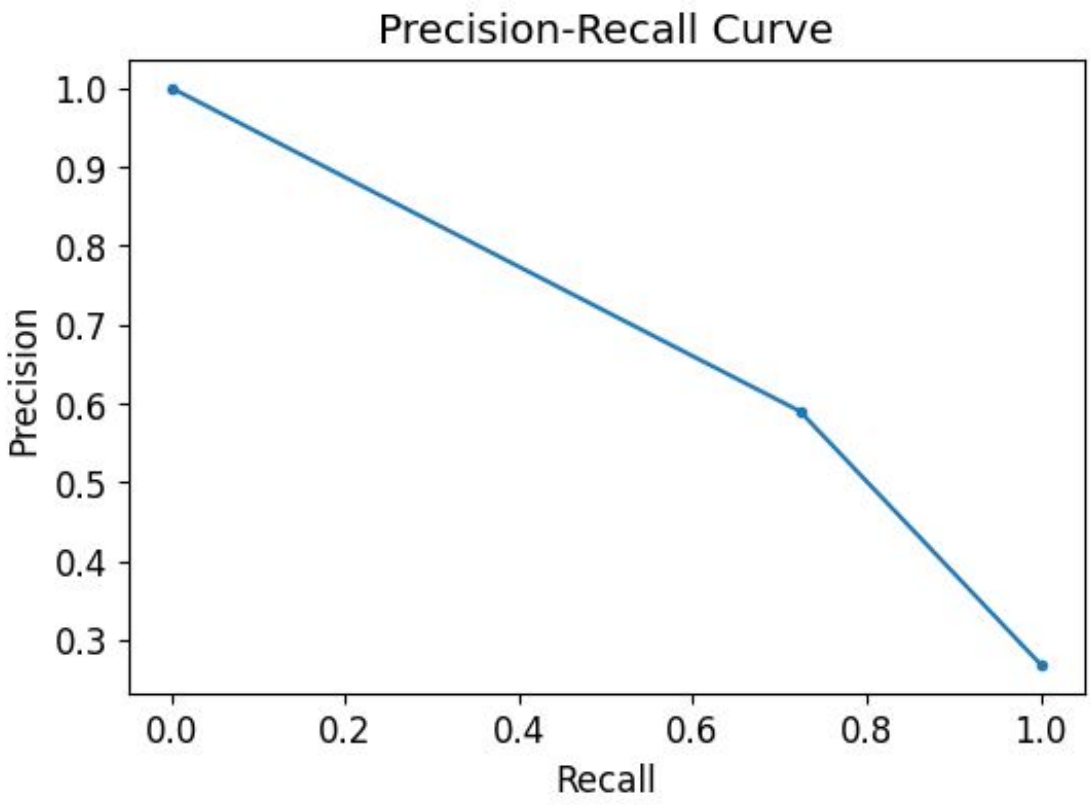

Figure 35: Precision-Recall Curve after Tuning

Forest Classifier — reveals some insightful observations regarding accuracy, macro F1-score, precision, recall, and computational efficiency.

```
pipeline_nb = Pipeline([
    ('m', CategoricalNB(min_categories=X_train.nunique(), alpha=50))
])
scores_nb = cross_validate(pipeline_nb, X_train, y_train, error_score="raise", scoring=["accuracy","f1_macro","precision_macro","recall_macro"])
✓ 0.2s
```

Figure 36: CategoricalNB Implementation

```
pipeline_rf = Pipeline([
    ('m', RandomForestClassifier(random_state=DEFAULT_SEED))
])
scores_rf = cross_validate(pipeline_rf, X_train, y_train,error_score="raise", scoring=["accuracy","f1_macro","precision_macro","recall_macro"])
✓ 4.1s
```

Figure 37: Random Forest Classifier Implementation

The tuned Naive Bayes model achieves an average accuracy of 0.7924, outperforming both the CategoricalNB model with 0.7608 and the Random Forest Classifier with 0.7943 by a small margin. This indicates that the tuned Naive Bayes model effectively balances correctly predicting

```
print_results_table([scores_nb_own, scores_nb, scores_rf], headings=["our_nb", "sklearn_nb", "sklearn_rf"])
✓ 0.0s
```

	our_nb	sklearn_nb	sklearn_rf
avg. accuracy:	0.7924	0.7608	0.7943
avg. f1_macro:	0.7224	0.7221	0.7183
avg. precision_macro:	0.7360	0.7124	0.7424
avg. recall_macro:	0.7139	0.7482	0.7045
avg. fitting time:	0.3719	0.0188	0.7854
avg. score time:	0.8153	0.0278	0.0400

Figure 38: Comparison between 3 models

both positive and negative cases, making it a strong contender in terms of overall prediction performance.

When analyzing the macro F1-score, the tuned Naive Bayes and CategoricalNB show similar results at 0.7224 and 0.7221, respectively, while the Random Forest Classifier falls slightly behind at 0.7183. The F1-score is particularly important for imbalanced datasets as it balances precision and recall, suggesting that the tuned Naive Bayes model effectively handles both false positives and false negatives while maintaining a balanced classification performance across classes.

Precision and recall show contrasting strengths between these models. The tuned Naive Bayes has the highest precision at 0.7360, meaning its predictions for the positive class are more reliable compared to CategoricalNB at 0.7124 and Random Forest at 0.7424. In contrast, the CategoricalNB model achieves the highest recall at 0.7482, indicating that it captures more true positives, albeit at the cost of slightly lower precision. The Random Forest Classifier lags behind in recall at 0.7045, suggesting it misses more positive cases than the other two models.

Finally, comparing computational efficiency reveals notable differences. The tuned Naive Bayes model's fitting time averages at 0.3719 seconds, significantly longer than the CategoricalNB's 0.0188 seconds but faster than Random Forest's 0.7854 seconds. Similarly, the tuned Naive Bayes model has a longer scoring time of 0.8153 seconds compared to CategoricalNB's 0.0278 and Random Forest's 0.0400 seconds. This indicates that while the tuned Naive Bayes achieves better predictive performance, it comes at the cost of increased computational complexity. Overall, the tuned Naive Bayes model demonstrates the best balance between performance metrics, while the CategoricalNB offers speed and higher recall, and the Random Forest Classifier stands out for precision but with higher computational demands.

7 Genetic Algorithms

8 Graphical Models

9 Conclusion

10 References