

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 Bằng	2252078
	Trần Vũ Hào	2052978
	Đào Tiến Tuấn	1953069

HO CHI MINH CITY, MARCH 2025



Contents

1	Introduction	3
1.1	Background	3
1.2	Goal	3
1.3	Setup	3
2	Data Overview	4
2.1	Dataset Description	4
2.2	Features	4
2.3	Data Characteristics	5
2.4	Data Preprocessing	8
2.4.1	Data Cleaning	9
2.4.2	Handling Missing Values	9
2.4.3	Feature Engineering	9
2.4.4	Feature Scaling	10
3	Methodology	11
3.1	Decision Trees	11
3.2	Neural Networks	11
4	Decision Trees	13
5	Neural Networks	14
5.1	Model Implementation	14
5.2	Training Implementation	14
5.3	Model Tuning Implementation	15
5.4	Training Process	15
5.5	Manual Training Results and Analysis	15
5.5.1	Evaluation and Cross-Validation	16
5.5.2	Loss Curves	17
5.5.3	Confusion Matrix	17
5.5.4	ROC Curve and AUC Score	18
5.5.5	Precision-Recall Curve	18
5.6	Model Tuning Results and Analysis	19
5.6.1	Optimization History Plot	19
5.6.2	Slice Plot	20
5.6.3	Hyperparameter Importances	21
5.7	Optimal Training Results and Analysis	23
5.7.1	Evaluation and Cross-Validation	23
5.7.2	Loss Curves	24
5.7.3	Confusion Matrix	24
5.7.4	ROC Curve and AUC Score	25
5.7.5	Precision-Recall Curve	25
5.8	Models Comparison	26
6	Naive Bayes	28
7	Genetic Algorithms	29



8	Graphical Models	30
9	Conclusion	31
10	References	32



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

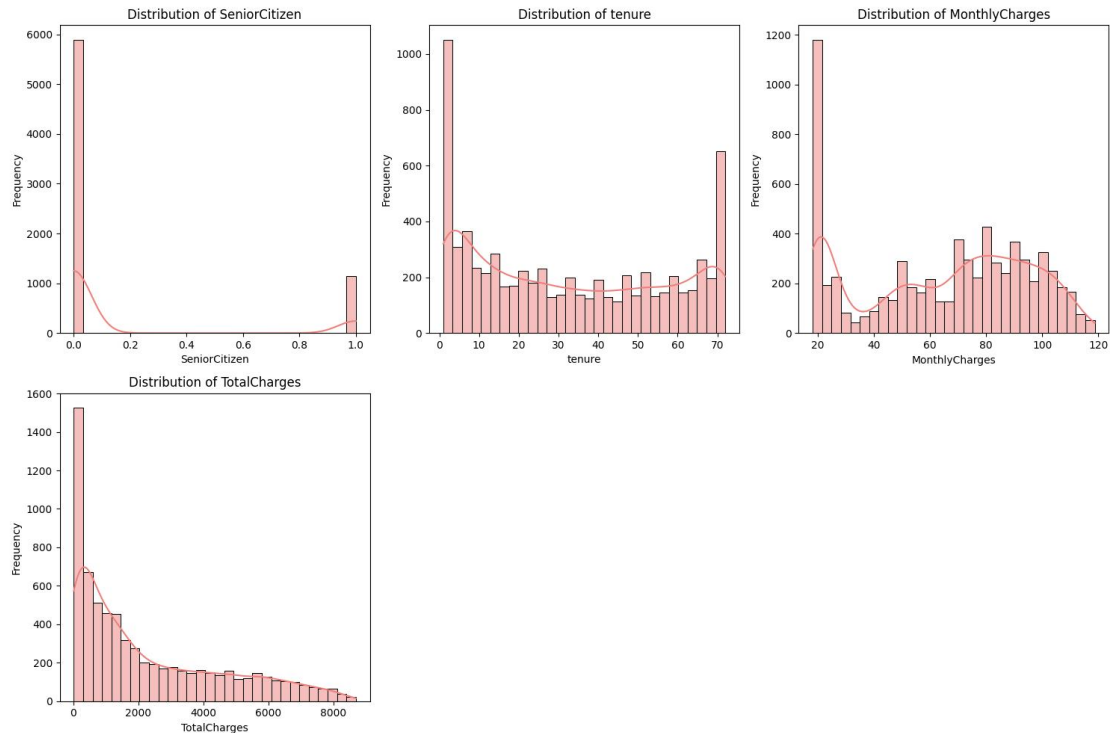


Figure 1: Distribution of Key Features in the Telco Customer Churn Dataset

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.

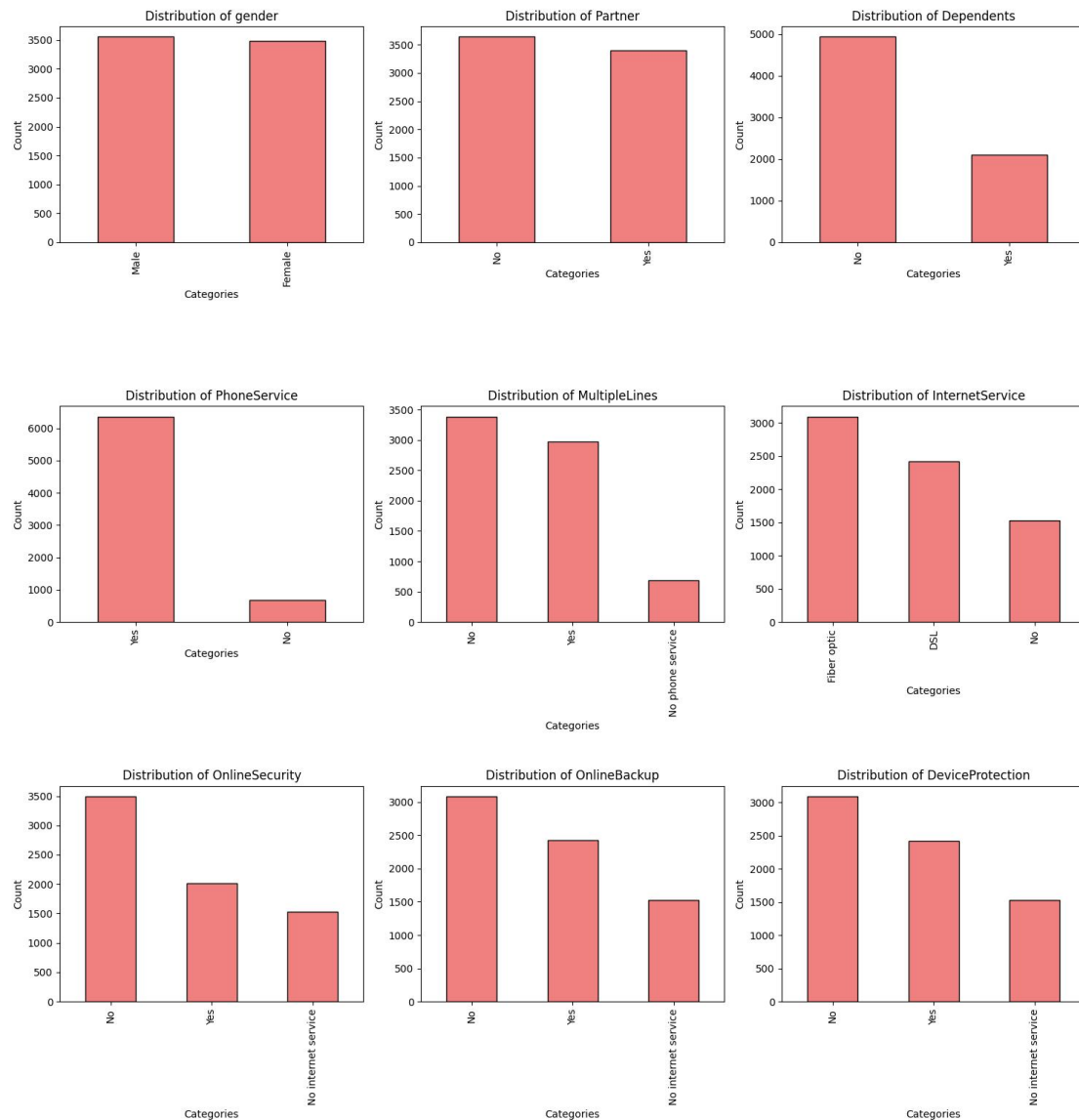


Figure 2: Distribution of Categorical Features in the Telco Customer Churn Dataset

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.

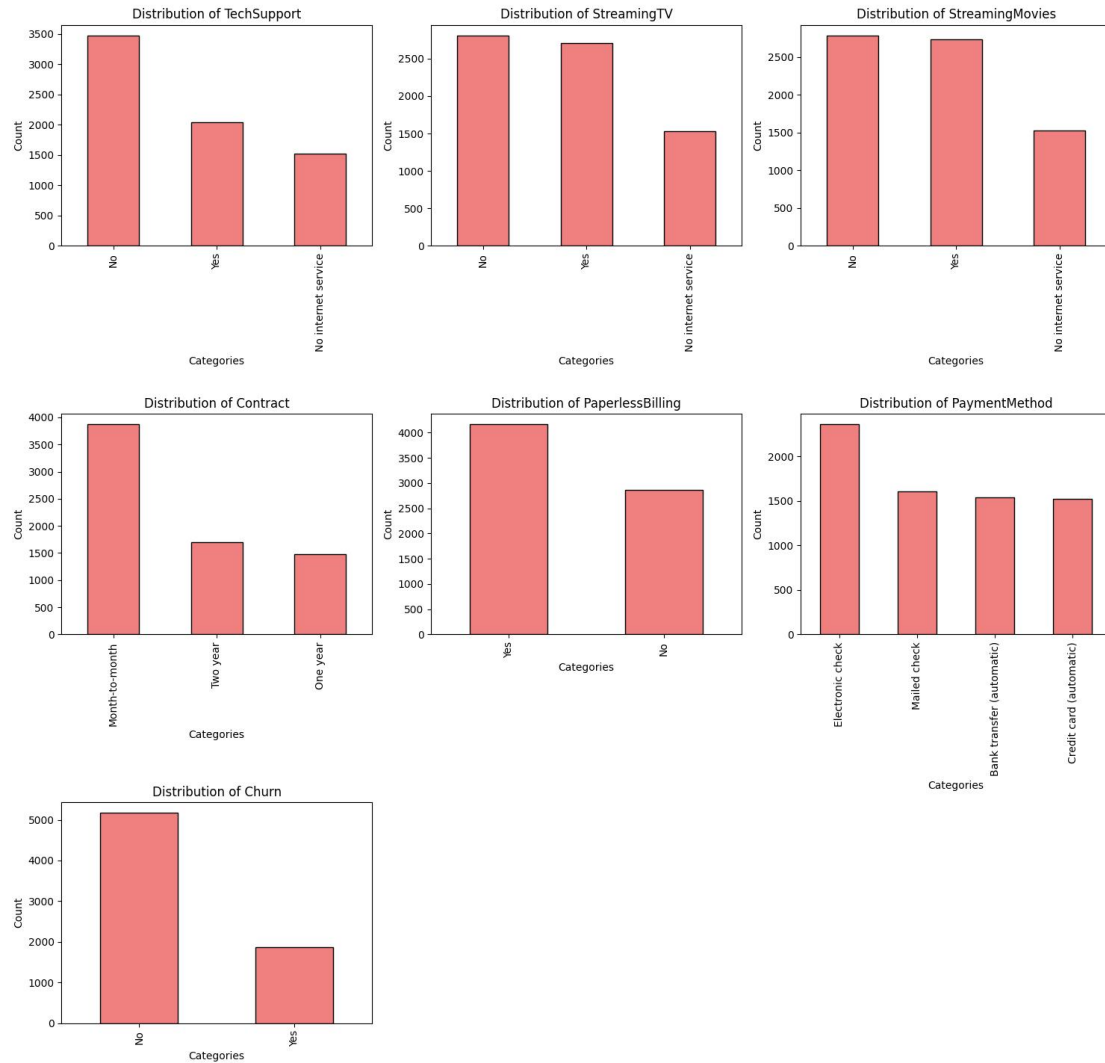


Figure 3: Distribution of Categorical Features in the Telco Customer Churn Dataset

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

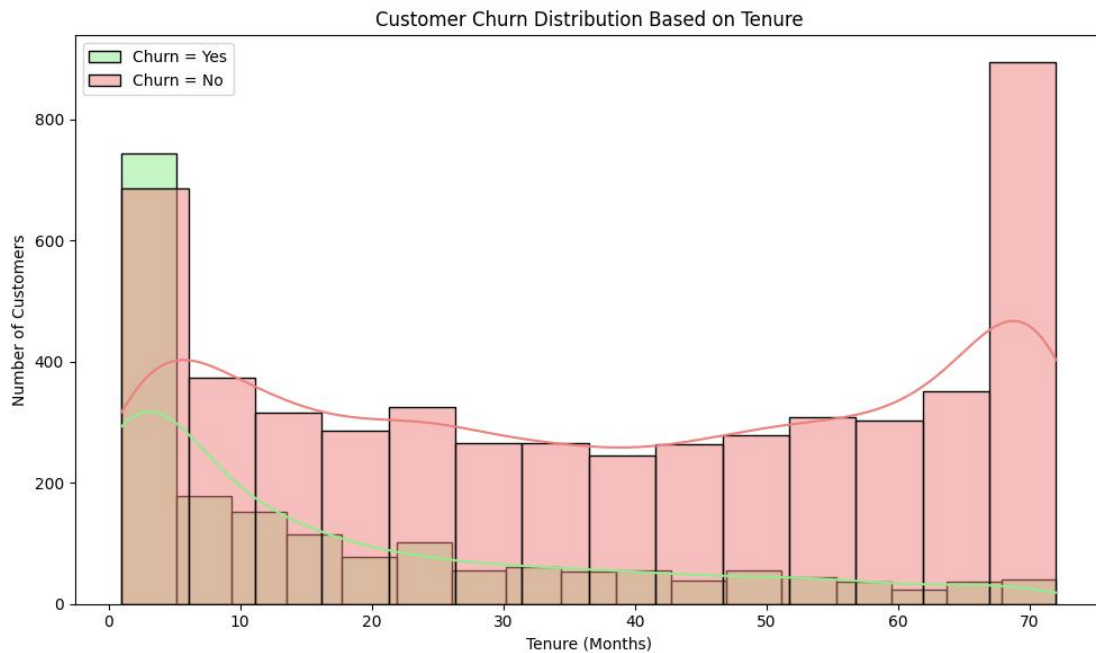


Figure 4: Customer Churn Distribution Based on Tenure

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.

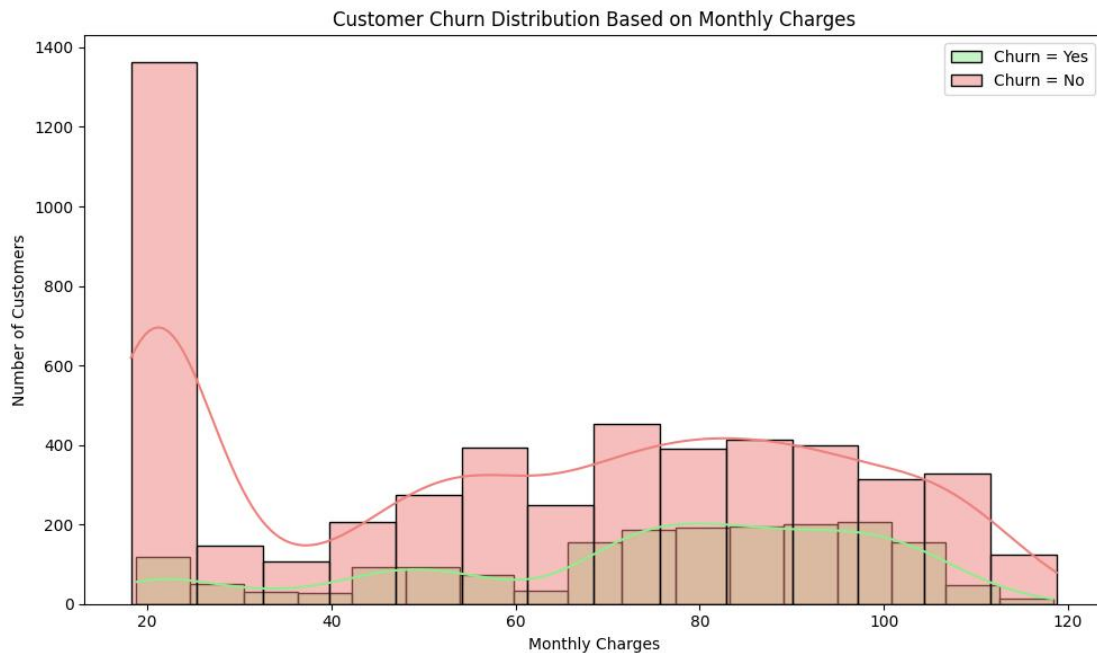


Figure 5: Customer Churn Distribution Based on Monthly Charges

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.

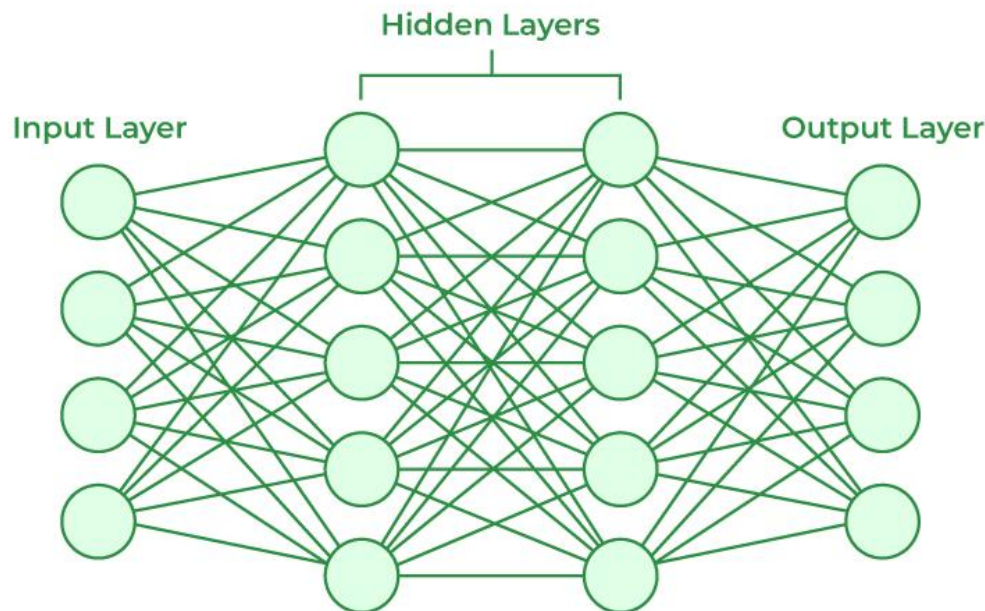


Figure 7: Neural Network Architecture

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.



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,


```
... Training neural network with the following hyperparameters:
hidden_size: 15
num_hidden_layers: 1
dropout_rate: 0.5
lr: 0.001
weight_decay: 0.0001
batch_size: 32
epochs: 50
2025-03-17 11:20:29,798 - INFO - Epoch [10/50], Train Loss: 0.4223, Validation Loss: 0.4360
2025-03-17 11:20:31,946 - INFO - Epoch [20/50], Train Loss: 0.4149, Validation Loss: 0.4277
2025-03-17 11:20:34,126 - INFO - Epoch [30/50], Train Loss: 0.4064, Validation Loss: 0.4330
2025-03-17 11:20:36,244 - INFO - Epoch [40/50], Train Loss: 0.3986, Validation Loss: 0.4276
2025-03-17 11:20:38,508 - INFO - Epoch [50/50], Train Loss: 0.3969, Validation Loss: 0.4338
2025-03-17 11:20:38,510 - INFO - Training results saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\storage\trainings\ne_neural_n
2025-03-17 11:20:38,513 - INFO - Model saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\models\experiments\ne_neural_network_man
```

Figure 8: 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

```
metrics = evaluate_model(file_name="ne_neural_network_manual", X_test=X_test, y_test=y_test, threshold=0.5)
[32]
... 2025-03-17 11:20:38,680 - INFO - Model ne_neural_network_manual loaded successfully.
2025-03-17 11:20:38,692 - INFO - Accuracy: 0.7989
2025-03-17 11:20:38,693 - INFO - Precision: 0.6270
2025-03-17 11:20:38,694 - INFO - Recall: 0.5387
2025-03-17 11:20:38,694 - INFO - F1: 0.5795
2025-03-17 11:20:38,696 - INFO - Roc_auc: 0.8351
2025-03-17 11:20:38,698 - INFO - Evaluation results saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\storage\evaluations\ne_neur
```

Figure 9: 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.

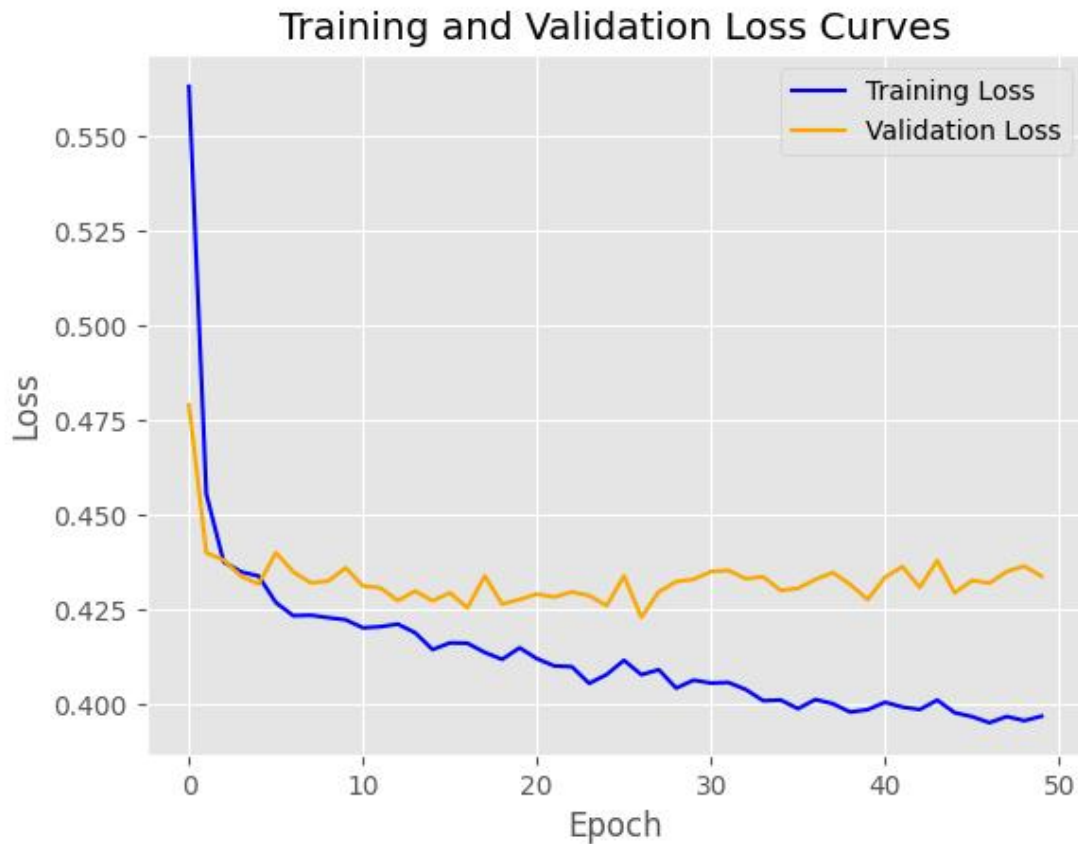


Figure 10: Training and Validation Loss Curves for Manual Hyperparameters

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.

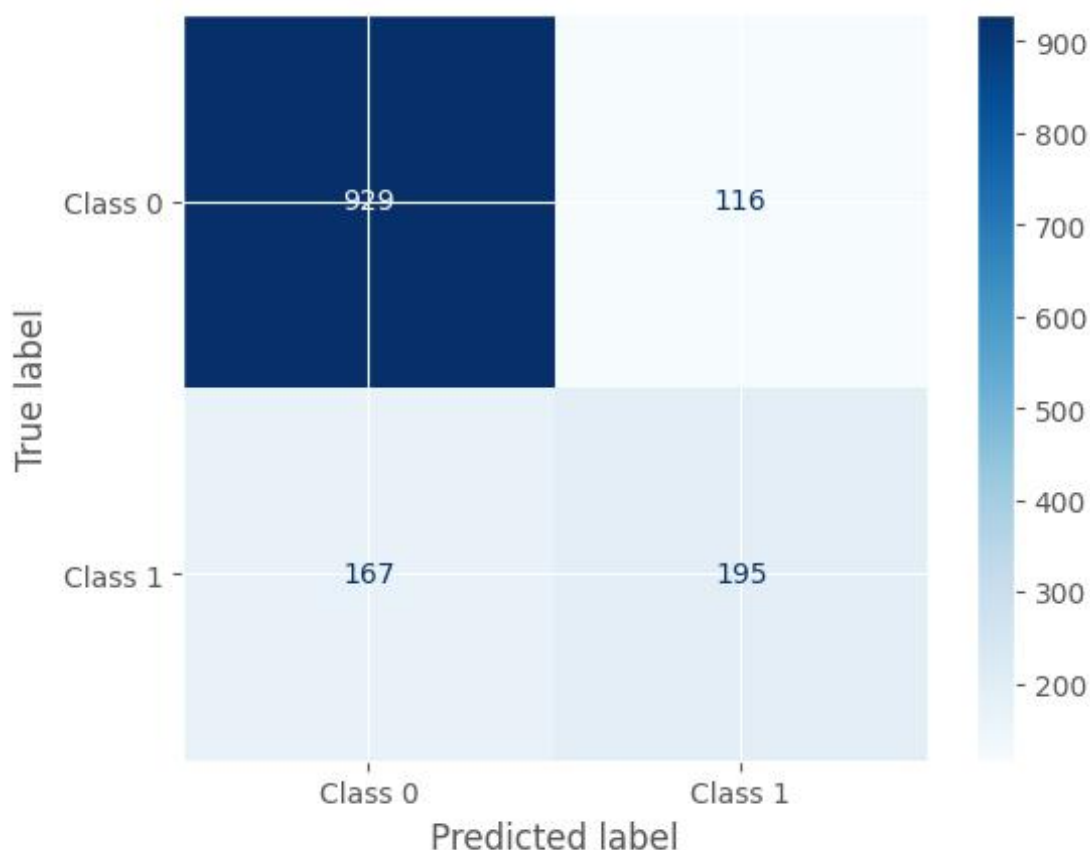


Figure 11: Confusion Matrix for Manual Hyperparameters

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.

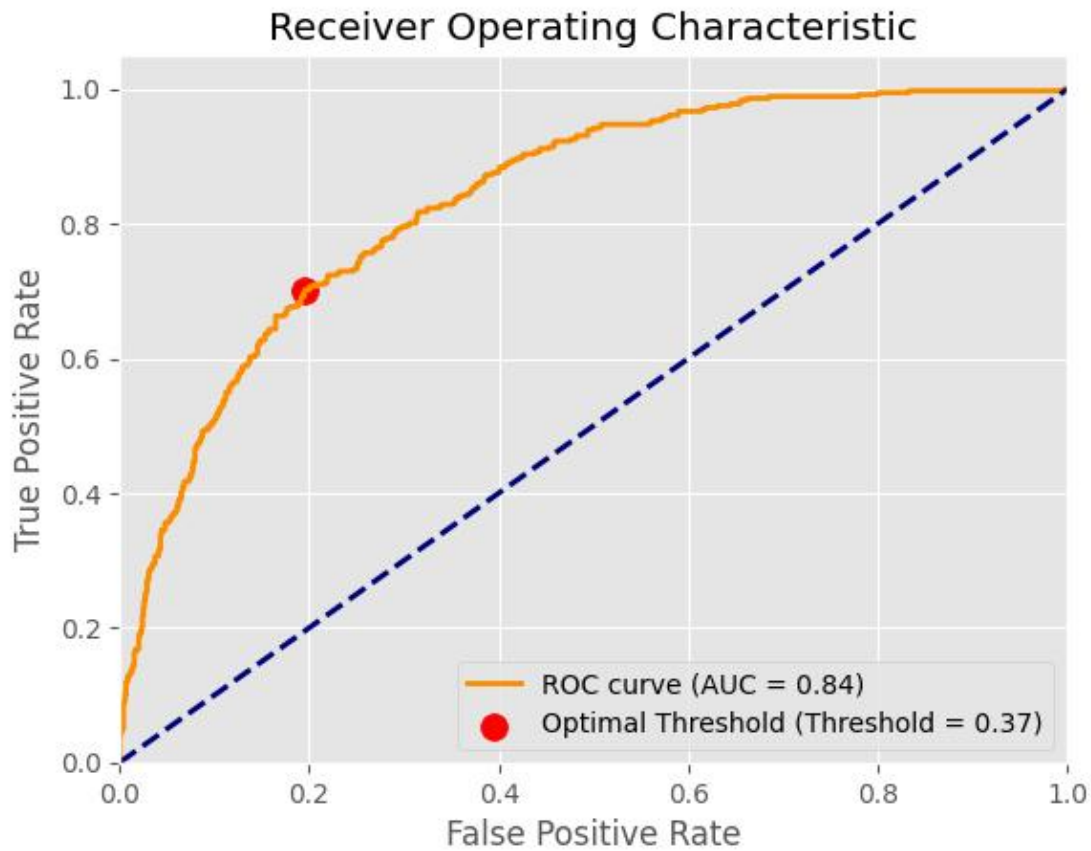


Figure 12: ROC Curve for Manual Hyperparameters

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

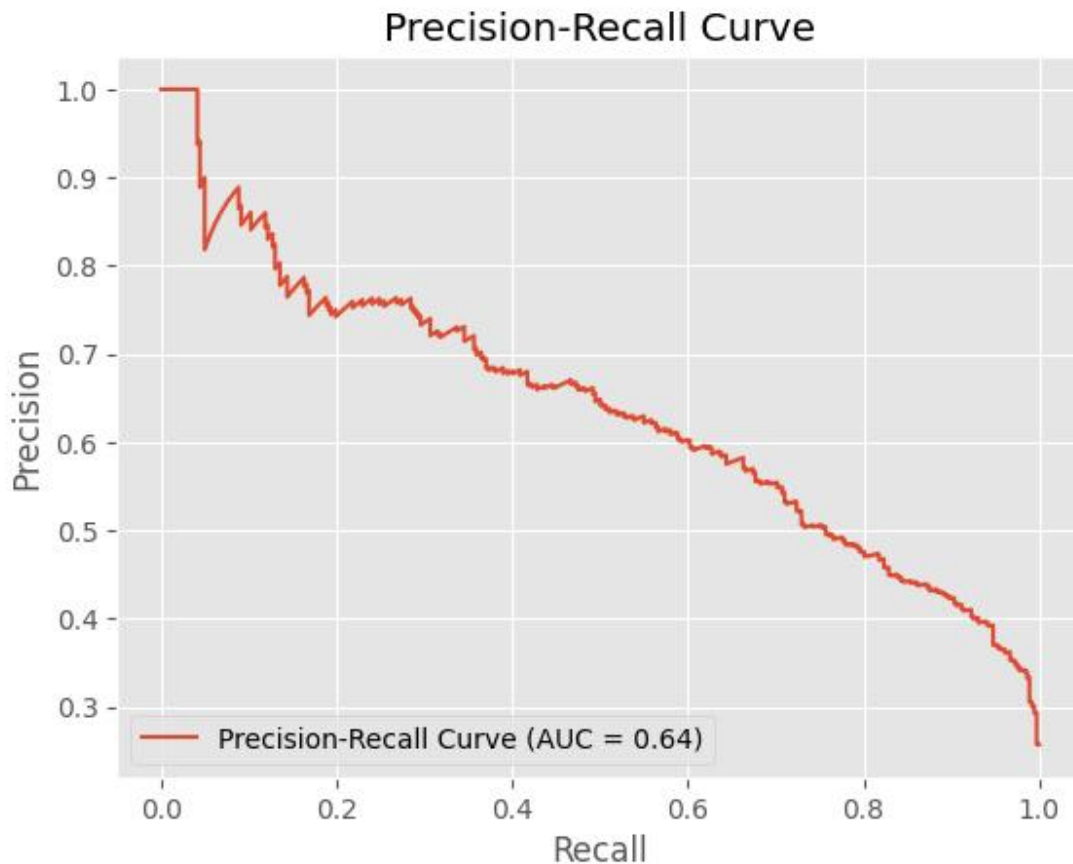


Figure 13: Precision-Recall Curve for Manual Hyperparameters

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.

```
# 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
)
```

Figure 14: Model Tuning Using Optuna

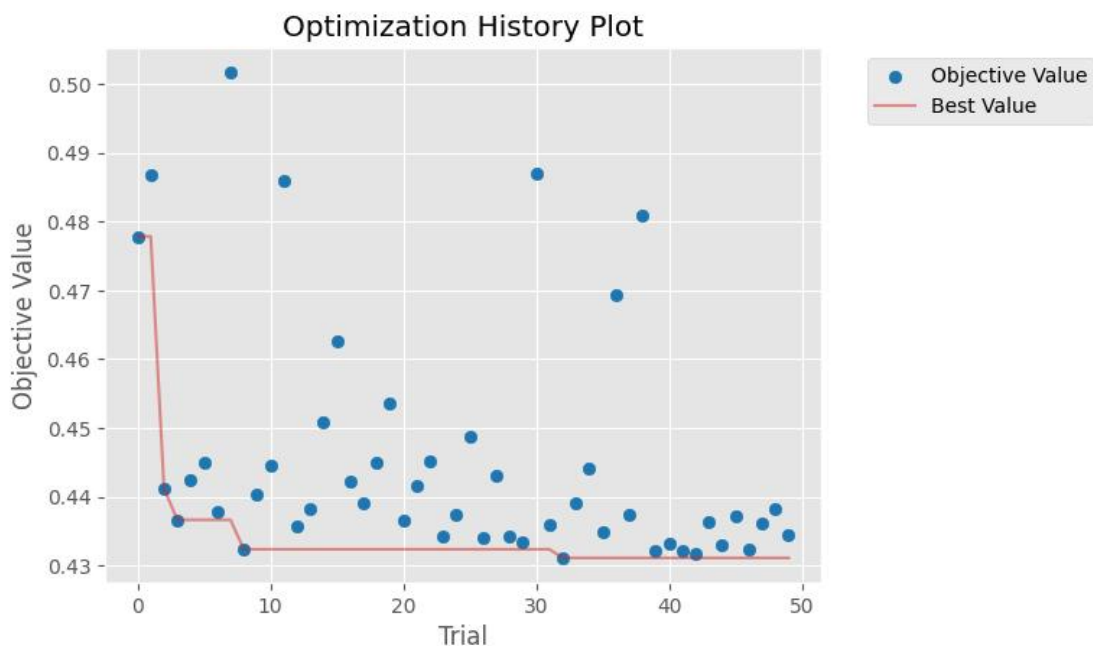


Figure 15: Optimization History Plot

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

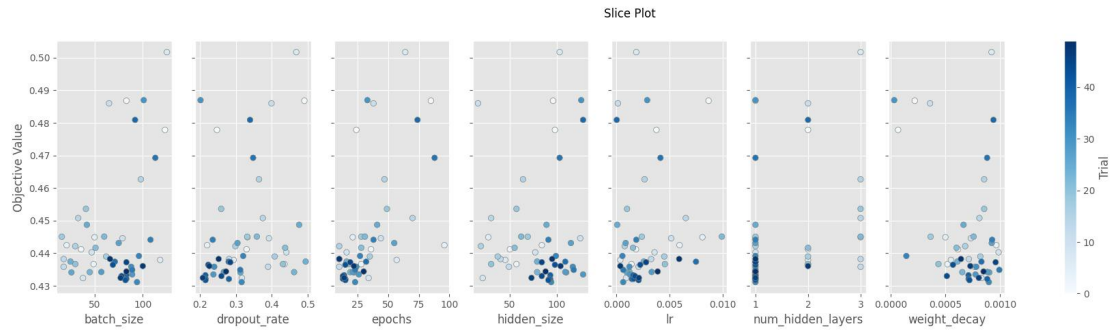


Figure 16: Slice Plot

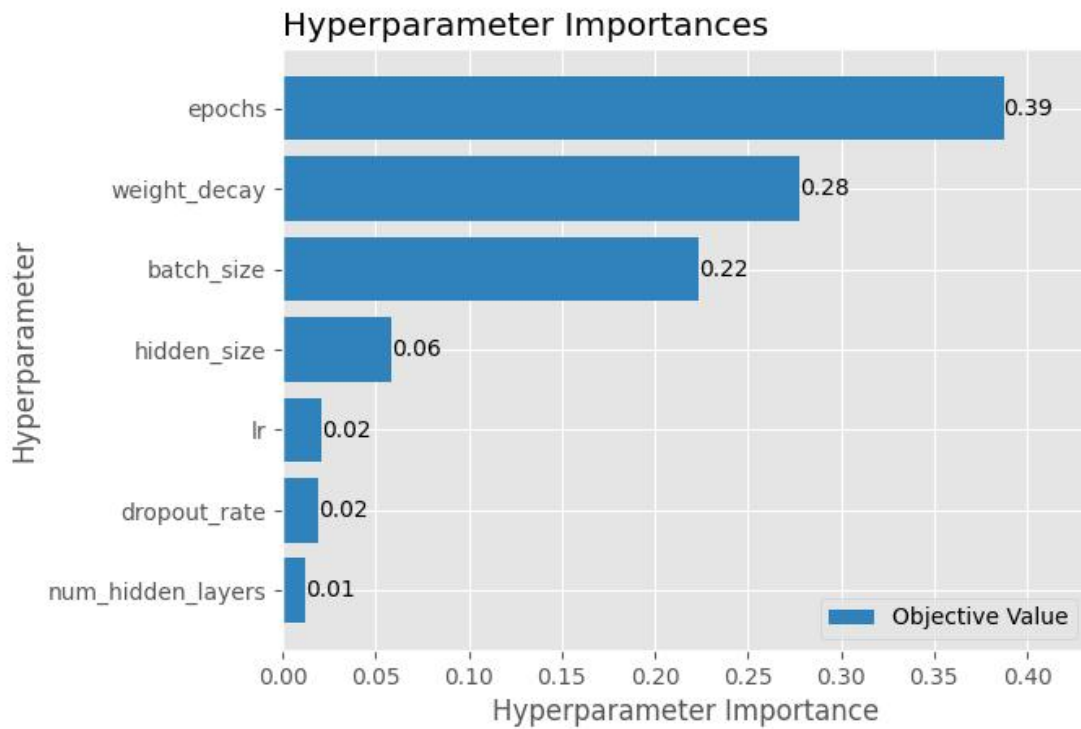


Figure 17: Hyperparameter Importances Plot

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

```
... 2025-03-17 11:26:30,886 - INFO - Study ne_neural_network_study loaded successfully.
Training neural_network with the following hyperparameters:
lr: 0.00062381313333613247
batch_size: 94
hidden_size: 94
num_hidden_layers: 1
dropout_rate: 0.3157987439678077
epochs: 23
weight_decay: 0.0008980887234870114
2025-03-17 11:26:32,054 - INFO - Epoch [10/23], Train Loss: 0.4106, Validation Loss: 0.4262
2025-03-17 11:26:33,343 - INFO - Epoch [20/23], Train Loss: 0.3981, Validation Loss: 0.4268
2025-03-17 11:26:33,695 - INFO - Training results saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\storage\trainings\ne_neural_n
2025-03-17 11:26:33,699 - INFO - Model saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\models\experiments\ne_neural_network_stu
```

Figure 18: 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

```
metrics = evaluate_model(file_name="ne_neural_network_study", X_test=X_test, y_test=y_test, threshold=0.5)
[14] ✓ 0.0s Python
... 2025-03-17 15:30:36,529 - INFO - Model ne_neural_network_study loaded successfully.
2025-03-17 15:30:36,543 - INFO - Accuracy: 0.8045
2025-03-17 15:30:36,544 - INFO - Precision: 0.6399
2025-03-17 15:30:36,545 - INFO - Recall: 0.5497
2025-03-17 15:30:36,546 - INFO - F1: 0.5914
2025-03-17 15:30:36,547 - INFO - Roc_auc: 0.8432
2025-03-17 15:30:36,550 - INFO - Evaluation results saved to c:\Users\duyhu\Downloads\ML_Beginners\src\..\storage\evaluations\ne_neur
```

Figure 19: 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

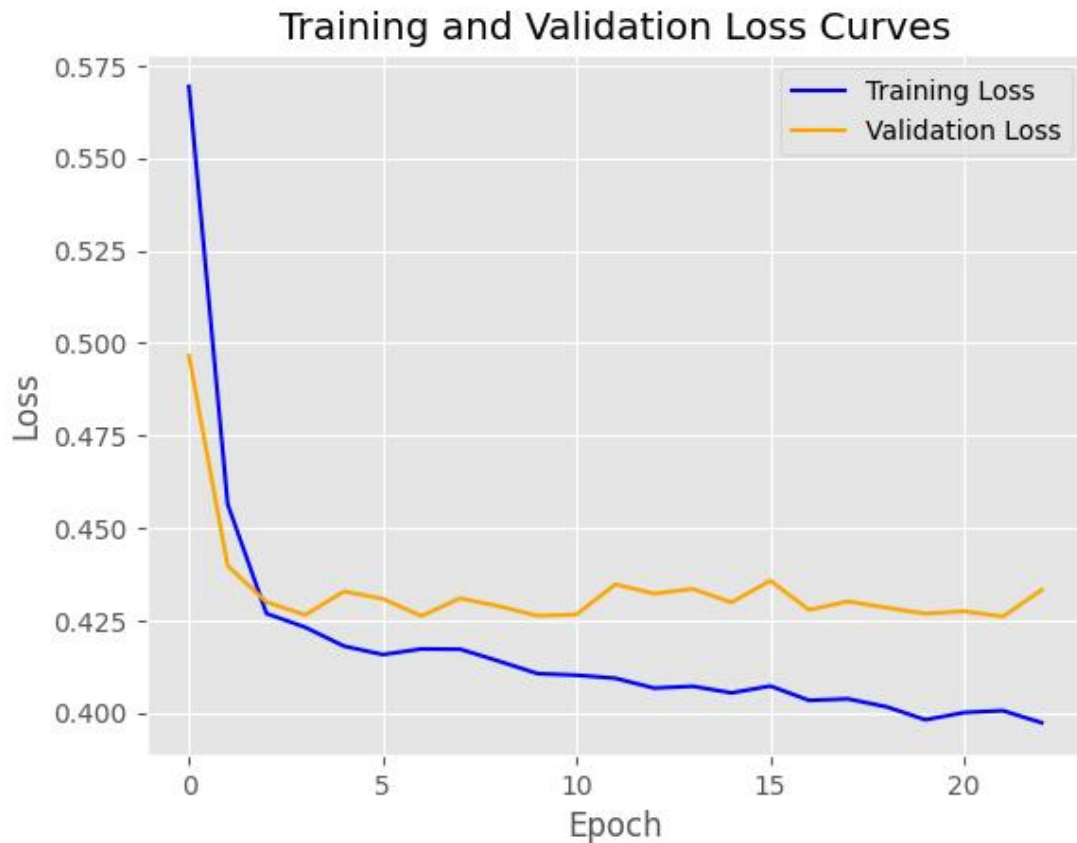


Figure 20: Training and Validation Loss Curves for Optimal Hyperparameters

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

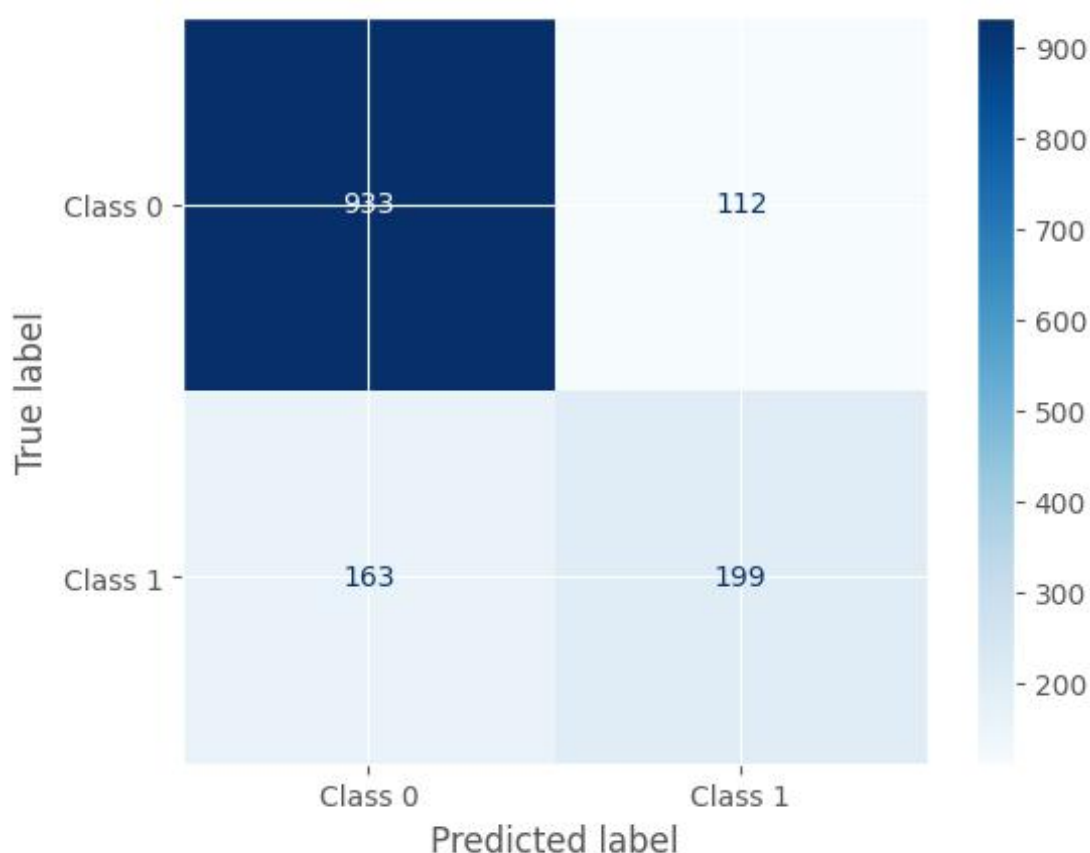


Figure 21: Confusion Matrix for Optimal Hyperparameters

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

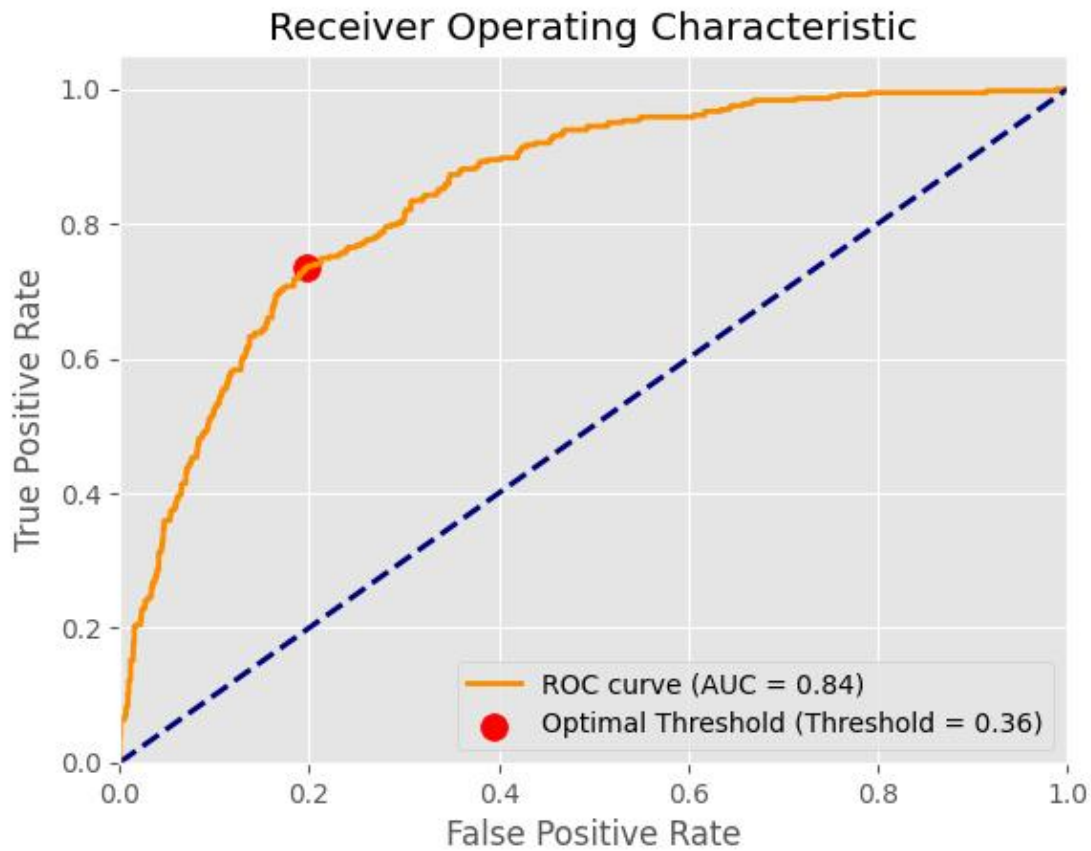


Figure 22: ROC Curve for Optimal Hyperparameters

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-

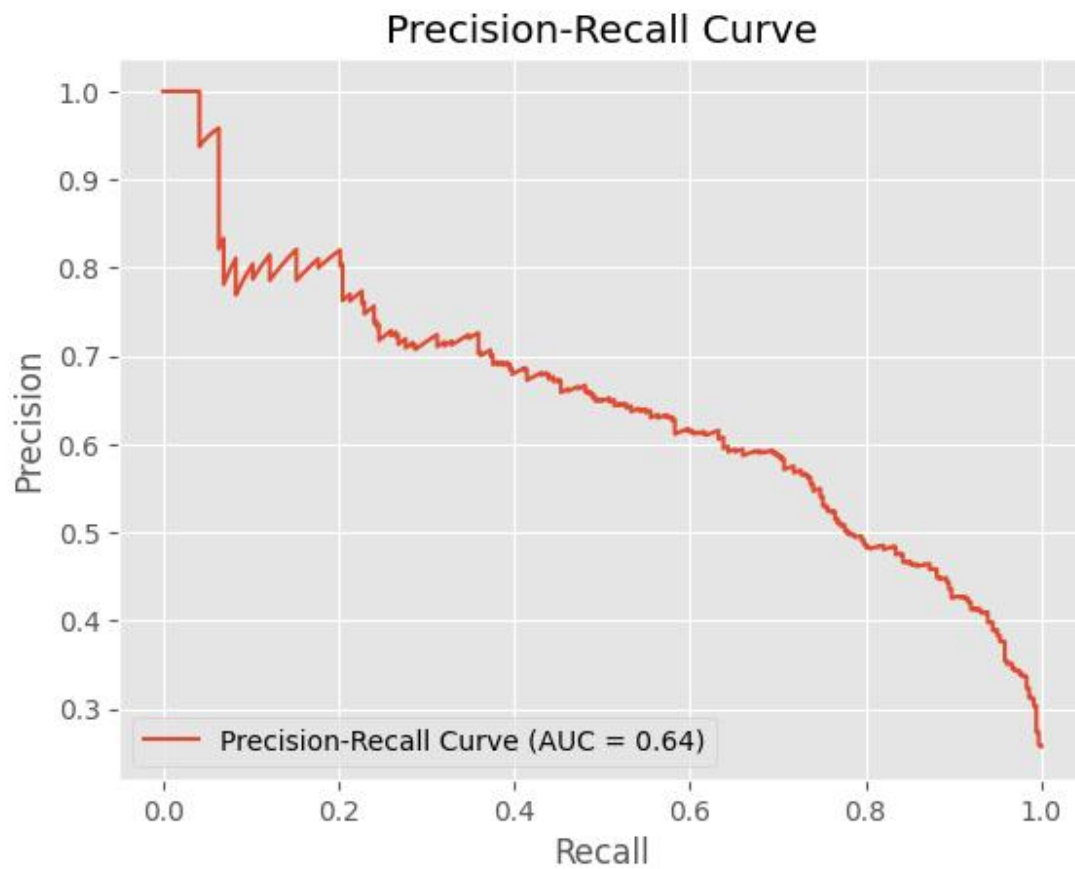


Figure 23: Precision-Recall Curve for Optimal Hyperparameters

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



7 Genetic Algorithms



8 Graphical Models



9 Conclusion



10 References