UNIVERSITY OF CALIFORNIA

Los Angeles

Income Prediction Using Machine Learning Techniques

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics & Data Science

by

Kahyun Jo

2024

ABSTRACT OF THE THESIS

Income Prediction Using Machine Learning Techniques

by

Kahyun Jo
Master of Applied Statistics & Data Science
University of California, Los Angeles, 2024
Professor Frederic R. Paik Schoenberg, Chair

This thesis presents a comprehensive study on predicting income levels, specifically predicting whether individuals earn more than $50,000 per year, with advanced machine learning techniques, using various demographic predictor variables such as capital gain, education level, relationship, occupation, and capital loss. The prediction of income levels is crucial for elucidating economic disparities and informing policy decisions. Utilizing the Adult Income dataset from the UCI Machine Learning Repository, which comprises demographic and socio-economic variables, the research entails a thorough evaluation of each model's performance. The methodology involves a preprocessing stage to ensure data quality, followed by the application of various machine learning algorithms including, but not limited to, Logistic Regression, k-Nearest Neighbors, Decision Trees, Random Forests, Support Vector Machines, and Neural Networks. A significant focus is placed on systematic hyper-parameter tuning to fine-tune models, particularly with the complex structures of Neural Networks and Random Forests. The findings indicate that Random Forest models exhibit superior performance in income prediction tasks across most metrics, including accuracy, sensitivity, precision, specificity, F1 score, AUC, and RMSE. The Baseline Random Forest achieves the best accuracy (86.410%), specificity (88.600%), and RMSE (0.315), suggesting strong overall performance and well-calibrated probabilities. The Tuned Random Forest achieves the highest AUC (94.964%) and F1 score (82.057%), indicating strong overall performance and an effective balance between precision and recall.

The thesis of Kahyun Jo is approved.

University of California, Los Angeles

2024

To my mother ...

who—among so many other things—

opened my eyes to the vast world

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Income inequality is a pressing issue that adversely affects economies and societies worldwide, often linked to social and political concerns, as societies with greater income disparities tend to experience more social problems, such as lower life expectancy, obesity, and mental illness [WP09]. Therefore, accurate income prediction is pivotal in economic and political systems, guiding optimal resource allocation and political decisions to lead to positive social outcomes. Income prediction can help identify populations at risk, prioritize interventions, and ensure that resources are allocated efficiently.

The objectives of this thesis are to evaluate the performance of machine learning models in predicting income levels of individuals from diverse socio-economic backgrounds, utilizing key predictors such as capital gain, education level, relationship, occupation, and capital loss. This paper will focus on commonly used metrics such as accuracy, sensitivity, specificity, precision, F1 score, AUC, and RMSE to assess the performance of each model. The methodology involves preprocessing the data to ensure quality, followed by applying and conducting a thorough analysis of various machine learning algorithms, including Logistic Regression, k-Nearest Neighbors, Decision Trees, Random Forests, Support Vector Machines, and Neural Networks. Special attention is given to hyperparameter tuning, particularly for complex models such as Neural Networks and Random Forests, to optimize their performance. In addition to quantitative metrics, this paper incorporates a comparative analysis of binned residual plots for the two best-performing models— the baseline and tuned Random Forest models. This comparison, presented in the conclusion, helps to validate the superior model performance and assess how well the predictions are align with the actual outcomes, particularly across different segments of the data. Enhanced modeling accuracy and predictive insights from this research could improve socio-economic policies and decision-making, further reducing the impact of income inequality.

The remainder of this thesis is structured as follows: Chapter 2 provides data preparation and EDA. Chapter 3 describes model training and evaluation of various machine learning models. Chapter 4 summarizes the key findings and their implications, highlighting the effective use of machine learning in predicting income levels.

CHAPTER 2

Data Preparation and EDA

The Adult Income dataset, derived from the 1994 US Census database, comprises 32,561 entries across 15 features. These features, which include demographic and socio-economic data, are summarized in Table 2.1.

Feature	**Type**	**Description**
age	Numerical	Age of the individual.
workclass	Categorical	Employment classification of the individual.
fnlwgt	Numerical	Statistical weight assigned to the individual.
education	Categorical	Highest education level attained by the individual.
education_num	Numerical	Number of educational years completed by the individual.
marital_status	Categorical	Legal marital status of the individual.
occupation	Categorical	Type of occupation of the individual.
relationship	Categorical	Individual's relationship status within a family.
race	Categorical	Race of the individual.
sex	Binary	Gender of the individual.
capital_gain	Numerical	Total amount of capital gains for the individual, reflecting income from sources other than salary/wages.
capital_loss	Numerical	Total amount of capital losses for the individual, reflecting losses from investments or other financial sources.
hours_per_week	Numerical	Number of hours the individual worked per week.
native_country	Categorical	Country of origin of the individual.
income	Binary	Income level of the individual, whether it is $\leq 50K$ or $> 50K$.

Table 2.1: Description of Variables in the Adult Income Dataset

2.1 Data Preparation

It is observed that the dataset does not contain any NA values. However, the features *workclass*, *occupation*, and *native_country* contain entries with question marks, which are indicative of missing data. These are imputed as an 'Unknown' category within their respective features.

```
# Imputing missing values with 'Unknown'
df$workclass[df$workclass == '?'] <- 'Unknown'
df$occupation[df$occupation == '?'] <- 'Unknown'
df$native_country[df$native_country == '?'] <- 'Unknown'
```

Furthermore, sparse categories within these features are combined to reduce the granularity and potentially enhance the model's ability to generalize from the data.

```
# Combining similar sparse categories for 'workclass'
df <- df %>% mutate(workclass = case_when(
  workclass %in% c('Federal-gov', 'Local-gov', 'State-gov') ~ 'Gov',
  workclass %in% c('Self-emp-inc', 'Self-emp-not-inc') ~ 'Self',
  workclass %in% c('Never-worked', 'Without-pay', 'Unknown') ~ 'Other/Unknown',
  TRUE ~ workclass))
```

Similar principles are applied to other categorical variables to improve model performance by combining sparse or similar categories. For the *occupation* feature, categories related to clerical roles, manual work, service, and professional occupations are combined into broader groups. This aggregation is intended to simplify the model's understanding of the data by reducing the complexity of the occupational classifications. The *education* feature is consolidated by grouping various levels of schooling into categories such as 'Elem' for elementary levels, 'HS-grad' for high school and equivalent levels, and 'Assoc' for different forms of associate degrees. The *marital_status* categories are combined into 'Married' for all types of marriages, and 'Previously-Married' for categories that indicate a past marriage. Similarly, the *relationship* variable is simplified by combining categories that represent individuals not part of a traditional family unit into 'Not-in-family'. The *native_country* variable, with 29,170 entries from the United States and 3,391 from 40 other countries, is simplified to 'US' and 'Non-US.' to prevent potential overfitting, assuming significant economic disparities between domestic and international data that could influence income predictions.

2.2 Exploratory Data Analysis (EDA)

The *income* target variable exhibits a class imbalance, with a distribution of 76% for $\leq 50K$ and 24% for $> 50K$. Such imbalances often present a challenge, as the minority class is underrepresented during the model fitting process, potentially leading to biased models that do not accurately capture the underrepresented class signal. Data-level methods aim to alleviate the impact of class imbalance by methods such as resampling, while algorithm-level strategies modify classifiers to improve minority class prediction [ASR13]. This paper employs both data-level resampling techniques, such as Logistic Regression with Oversampling, and algorithm-level methods, including Random Forest and Support Vector Machine, to enhance the robustness of the study.

Table 2.2 presents the distribution of demographic features in the dataset. The proximity of the median and mean ages suggests a symmetrical age distribution. The *education_num* variable, denoting years of education, averages to approximately 10 years across the sample. The *fnlwgt* variable, reflecting sample weights, shows a right-skewed distribution with its mean exceeding the median, indicating the presence of outliers. The *capital_gain* and *capital_loss* variables are usually zero for most individuals, indicating that the majority do not engage significantly in investment activities beyond earning their regular wages or salaries. Although the skewed distribution of these features could impact certain types of analysis, they are retained in our study without transformation as the primary focus of this thesis is on predictive modeling, where these features may still provide valuable signals for the algorithms employed. The *hours_per_week* feature centers around the standard 40-hour workweek, as evidenced by both the mean and median. These distributions are visually confirmed by

Figure 2.1, which shows symmetry in the features *age* and *education_num*, and significant skewness in the *capital_gain* and *capital_loss* distributions.

Feature	**Min**	**Median**	**Mean**	**Max**
age	17.00	37.00	38.58	90.00
fnlwgt	12,285	178,356	189,778	1,484,705
education_num	1.00	10.00	10.08	16.00
hours_per_work	1.00	40.00	40.44	99.00
capital_gain	0.00	0.00	1,078.00	99,999.00
capital_loss	0.00	0.00	87.30	4,356.00

Table 2.2: Summary statistics of numerical features in the dataset

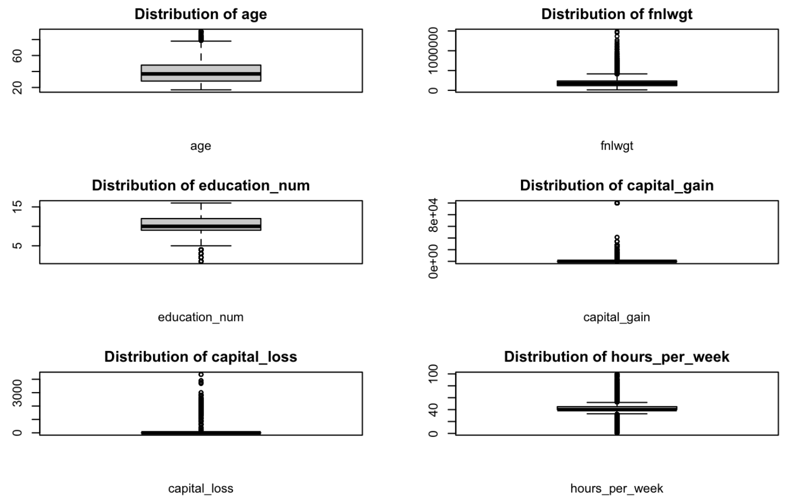


Figure 2.1: Box and Whisker Plots for numerical features

After data preprocessing, here is a summarized overview of the categorical variables' information:

- *workclass:* The majority of individuals are employed in the private sector (70%), followed by significant proportions in government (13%) and self-employment (11%). A smaller number fall into the category of 'Other/Unknown (6%)'.

- *education:* The most common education level is 'HS-grad' (47%), followed by 'Some-college' (24%), 'Bachelors' (13%), 'Assoc' (7%), 'Elem' (5%), 'Masters' (3%), 'Prof-school' (0.6%), and 'Doctorate' (0.4%).

- *marital_status:* The most common marital status is 'Married' (47%), followed by 'Never-married' (33%). Other categories such as 'Divorced' (14%), 'Separated' (3%), and 'Widowed' (3%) are also present but less frequent.

- *occupation:* The most frequent occupations are classified as 'White-Collar' (27%) and 'Blue-Collar' (24%), followed by 'Manual-Labor' (18%), 'Professional' (15%), 'Sales' (11%), and Other/Unknown (6%).

- *relaitonship:* The most common relationship status is 'Husband' (41%), followed by 'Not-in-family' (26%) which may reflect the demographic composition of the dataset. A smaller number fall into the categories of 'Own-child' (16%), 'Unmarried' (11%), 'Wife' (5%), and 'Other-relative' (3%).

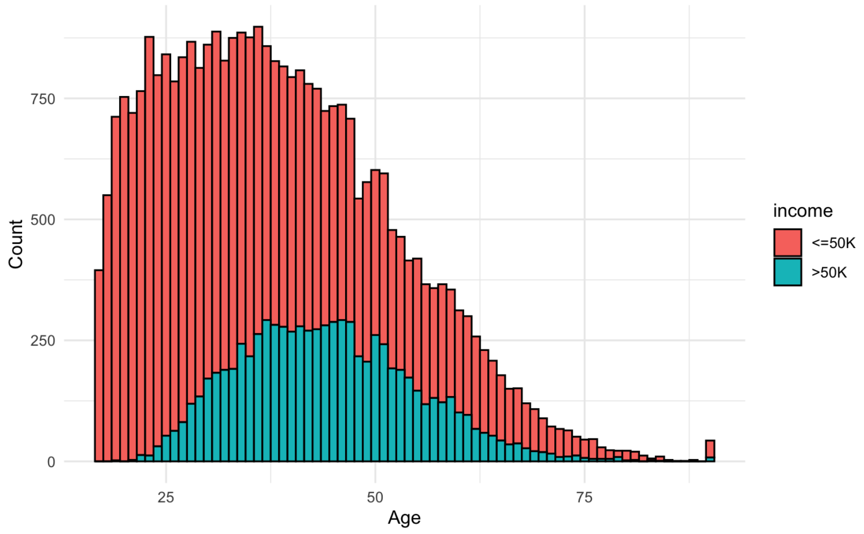
- *race:* The most common races are 'White' (85%) and 'Black' (10%), followed by 'Asian-Pac-Islander' (3%), 'Amer-Indian-Eskimo' (1%), and 'Other' (1%).

- *sex:* The majority of the dataset consists of males (67%), while females make up 33%.

- *native_country:* Most individuals are from the US (90%), while 10% are from non-US countries.

- *income:* The income distribution shows that 76% of individuals earn $\leq 50K$, while 24% earn $> 50K$.

Our bivariate analysis, depicted in Figure 2.2, explores patterns between the *income* and other features. The histogram at the top illustrates the distribution of income across different ages, while the bar chart at the bottom breaks down income levels by education. Income increases with age up to a certain point before declining, which is common as individuals enter retirement. Also, there is a clear positive relationship between the level of education and income, with those with higher educational attainment, like Bachelor's degrees or higher, possibly earning more.



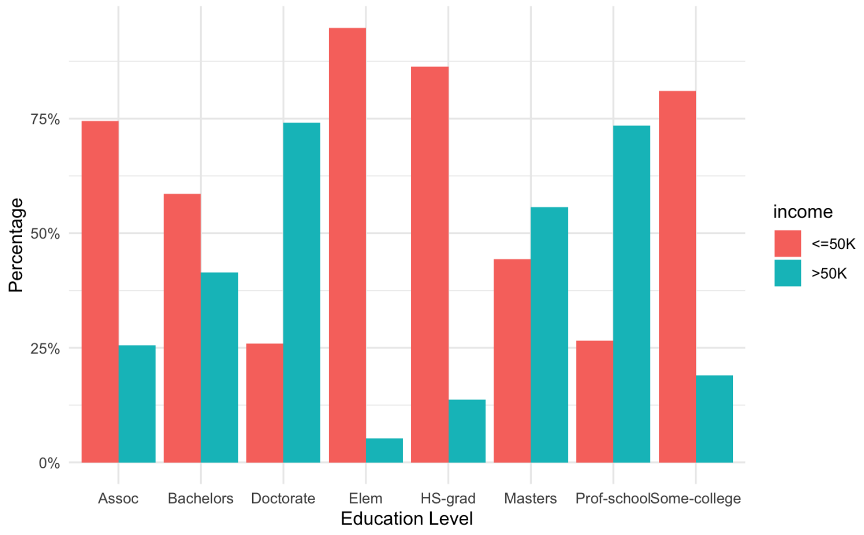


Figure 2.2: *income* by *age* (top) and *income* by *education* (bottom)

Figure 2.3 presents the correlation plot for all the variables. Notably, the *education_num* variable shows a high correlation with the *education* variable. It suggests that they are conveying similar information, possibly in a numeric and a categorical format. To prevent multicollinearity, which can adversely affect the performance of predictive models, the *education_num* variable is removed from our subsequent analyses. The features *sex* and *relationship* are also highly correlated, however, both of them are retained in our analysis. The *relationship* feature includes categories such as 'Husband' which inherently correspond to the 'Male' category in the *sex* variable. Given that 'Husband' is the most frequent category within relationship, constituting 41% of the sample where males represent 67%, a correlation arises naturally. Despite this correlation, the decision to retain both variables is justified as each offers unique predictive capabilities. The *sex* feature provides a broad demographic classification, while the *relationship* feature captures more specific household dynamics, both of which are relevant for the predictive objectives of this study.

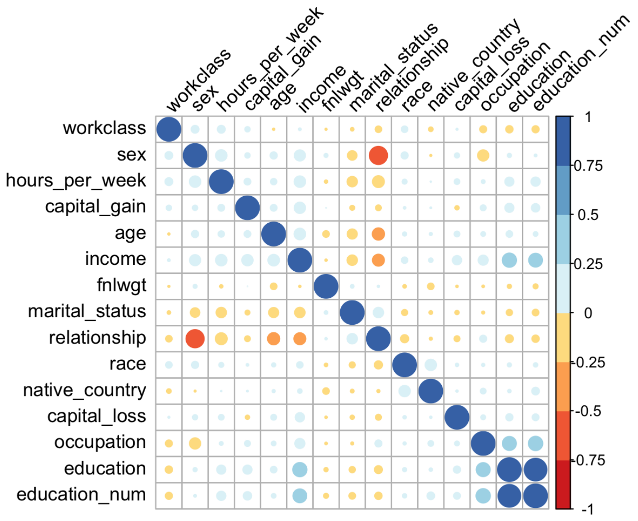


Figure 2.3: Correlation plot

This chapter has demonstrated the steps taken to preprocess and explore the Adult Income dataset. The EDA has provided valuable insights, particularly highlighting the distributions and imbalances inherent within our features, as exemplified by the detailed visualizations in Figures 2.1 and 2.2. The decision to remove the *education_num* feature is based on its high correlation with the *education* feature, simplifying our model without losing valuable information. Despite the correlation between the features *sex* and *relationship*, we preserve both for their distinct contributions to the model's predictive power. With these steps, we ensure clean data and robust analysis, laying the foundation for machine learning techniques in the subsequent stages of this study.

CHAPTER 3

Methodology

During the model fitting process, the dataset is partitioned into a training set and a testing set using a 80-20 split, ensuring that 80% of the data is used for model training and the remaining 20% for validation.

```
set.seed(123)
split <- createDataPartition(df$income, p = 0.8, list = FALSE)
```

After the model is trained, it is then applied to the testing set to generate predictions. The predicted probabilities are converted into binary class predictions, where a probability greater than 0.5 indicates an income above $50K$ (denoted as '1'), and below or equal to 0.5 indicates an income of $50K$ or less (denoted as '0'). To evaluate the performance of each model, several performance metrics are used, as outlined in Table 3.1. Accuracy, sensitivity, specificity, precision, and F1 score are obtained from a confusion matrix. RMSE evaluates the difference between values predicted by a model and the values actually observed, where P_j is the predicted value of instance j, A_j is actual value of instance j, and n is the total number of instances. Unlike other metrics, AUC assesses a classifier's ability to differentiate classes. S_p is the sum of the ranks for the positive instances, n_p is the number of positive instances, and n_p is the number of negative instances. The AUC is known to be a superior metric for evaluating the classifier performance and selecting an optimal solution during the classification training process [Pre81].

Metric	Formula	Description
Accuracy	$\frac{tp+tn}{tp+fp+tn+fn}$	Correct predictions ratio
Sensitivity	$\frac{tp}{tp+fn}$	True positive rate
Specificity	$\frac{tn}{tn+fp}$	True negative rate
Precision	$\frac{tp}{tp+fp}$	Accuracy of positive predictions
F1 Score	$\frac{2 \cdot p \cdot r}{p+r}$	Balance of precision and recall
RMSE	$\sqrt{\frac{1}{n}\sum(P_j - A_j)^2}$	Prediction error magnitude
AUC	$\frac{S_p - n_p(n_n+1)/2}{n_p n_n}$	Class separation ability

Table 3.1: Performance Metrics for Classification

3.1 Logistic Regression

In logistic regression, we model the probability that a given input belongs to a certain class—specifically, whether an individual's income exceeds $50K$. This probability is obtained using the sigmoid function, which maps any real number to a value between 0 and 1. The logistic regression model generates raw predictions, known as logits. These logits are the inputs to the sigmoid function. The logit (θ) is a linear combination of the input features $X_1, X_2, \ldots, X_n$, which represent the features used to predict income level in our case. Each feature is multiplied by its respective learned coefficient $\beta_1, \beta_2, \ldots, \beta_n$, with the addition of a bias term β_0. This logit represents the log-odds of the predicted probability ($\hat{p}$), and is calculated as follows:

$$\theta = \log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n \tag{3.1}$$

To convert the logits to a probability, we apply the sigmoid function:

$$\hat{p} = \frac{1}{1+e^{-\theta}} \tag{3.2}$$

Upon fitting the logistic regression model to the training data, it is then employed to make predictions on the testing set. The model's performance is evaluated using various metrics, revealing an accuracy of 84.623%, a sensitivity of 74.120%, and a specificity of

87.197%. In Figure 3.1, the area under the ROC curve is 90.496%, which indicates that the model has a high probability of correctly distinguishing between individuals who earn more and less than $50K$. An RMSE of 0.323 suggests that the model's probability predictions for an individual's income category deviate from the actual observed outcomes by about 32.3%. This provides an indication of the average magnitude of prediction errors. However, the precision and F1 score are comparatively lower, at 58.591% and 65.447% respectively. These figures suggest that while the model is generally robust in identifying true positives and true negatives, there is room for improvement in its precision and the balance between precision and recall, as indicated by the F1 score.

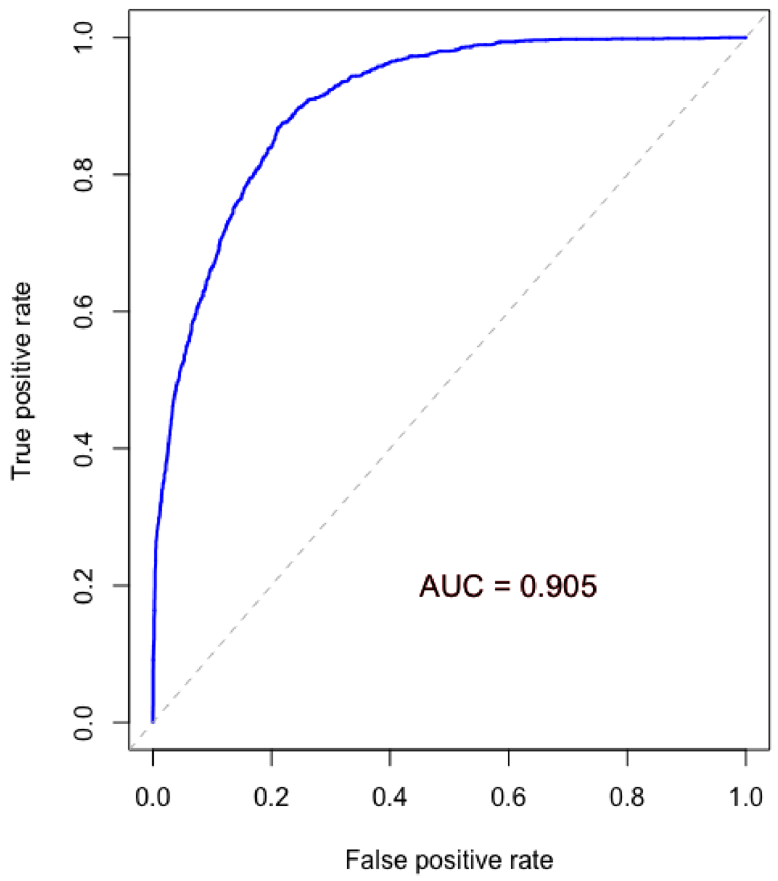


Figure 3.1: ROC Curve for Logistic Regression

3.2 Logistic Regression with Oversampling

In this section, we explore the application of oversampling techniques to address the class imbalance evident in the target variable of our dataset, where 76% of individuals have an income of $\leq 50K$ and 24% have an income $> 50K$. Initially, we utilize the `upSample` function from the `caret` package to address the class imbalance in our dataset. However, over-sampling with replacement tends to duplicate the existing minority samples without introducing new information. As more instances of the minority class are replicated, the model tries to identify more and more specific regions, which in turn increases the risk of overfitting [CBH02]. To mitigate this, we transition to using `ovun.sample` from the `ROSE` package, which generates synthetic samples by introducing random noise into the process of creating new data points. Unlike oversampling with replacement, the synthetic examples encourage the classifier to define broader and less specific decision regions. As a result, the model generalizes better, learning regions that adapt to variations in the minority class [CBH02].

Following the challenges associated with class imbalance and the overview of the oversampling techniques, we proceed to apply these methods in our analysis. We employee the `upSample` function from the `caret` package to balance the dataset by duplicating the samples of the minority class. This adjustment aims to improve the logistic regression model's performance by providing a more balanced dataset. The results from the logistic regression model trained on the upsampled data shows a notable impact across various metrics. There is a slight improvement in accuracy, up from 84.623% to 85.227%, suggesting that the model is correctly predicting a higher proportion of the outcomes when trained on the balanced dataset (Table 3.2). The model's sensitivity decreases from 74.120% to 73.264%, suggesting that the model becomes slightly less adept at identifying individuals earning more than 50K. There is an improvement in specificity from 87.197% to 88.078%, indicating that the model trained on the upsampled data is slightly better at identifying those who earn 50K or less. These comparisons reveal that while upsampling has improved certain aspects of the model's performance, particularly specificity, it has also introduced slight decreases in other areas, such as sensitivity and the F1 score. The AUC has stayed consistent, affirming the model's strong discriminative power even after addressing class imbalance.

Metric	Baseline	Upsampled	ROSE-Augmented
Accuracy	84.623%	85.227%	80.988%
Sensitivity	74.120%	73.264%	58.010%
Precision	58.591%	59.417%	85.043%
Specificity	87.197%	88.078%	94.155%
F1 Score	65.447%	65.618%	68.972%
AUC	90.496%	90.294%	90.490%
RMSE	0.323	0.320	0.359

Table 3.2: Comparison of Performance Metrics Across Logistic Regression Models

We then utilize the `ovun.sample` function from the `ROSE` package to our dataset, which originally comprises 24,720 instances with income $\leq 50K$ and 7,841 instances with income $> 50K$. This function rebalances the dataset to consist of 19,826 instances for the $\leq 50K$ class and 19,863 for the $> 50K$ class, equalizing the presence of both income groups in the training set. The code snippet below illustrates the use of `ovun.sample` to achieve this balance, followed by fitting a logistic regression model on the newly balanced data:

```
over <-  ovun.sample(income~., data = training_set, method = "over")$data
overlogit <- glm(income ~., data = over, family = binomial('logit'))
```

As shown in Table 3.2, the ROSE-augmented logistic regression model demonstrates an interesting trade-off between the metrics. The ROSE-augmented model's accuracy (80.988%) is lower than both the regular (84.623%) and upsampled models (85.227%). This could suggest that while the synthetic data generation adds diversity, it may introduce complexities that slightly reduce overall prediction correctness. The sensitivity sees a significant drop in the ROSE-augmented model (58.010%) compared to both the regular (74.120%) and upsampled models (73.264%). There is a substantial increase in precision for the ROSE-augmented model (85.043%) over the regular (58.591%) and upsampled models (59.417%).

This suggests that when the model does predict the positive class, it is more likely to be correct after applying the ROSE method. There is a marked improvement in precision and specificity, which could be valuable in certain applications where the cost of false positives is high. However, this comes at the expense of lower overall accuracy and sensitivity. The stability of the AUC across models suggests that the discriminative ability of the model remains strong, despite variations in other metrics.

3.3 L1 and L2 Regularization

In this section, we explore L1 and L2 regularization techniques to enhance the predictive performance and robustness of our models. The primary motivation behind fitting these models to our data arises from the need to address issues of overfitting and to improve model interpretability in the presence of multicollinearity among predictors.

Let $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ represent the vector of features for the i-th observation. Let $\hat{\beta} = (\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_p)$ represent the vector of estimated coefficients for all p features. The lasso loss function is given by:

$$L_{\text{Lasso}}(\beta) = \underbrace{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2}_{\text{Loss}} + \lambda \underbrace{\sum_{j=1}^{p} |\beta_j|}_{\text{Penalty}} \tag{3.3}$$

The first part of the lasso objective function is the RSS, which measures the fit of the model to the data by calculating the sum of the squares of the residuals. The second part of the objective function is the penalty term, which adds the absolute values of the coefficients multiplied by the tuning parameter λ. This penalty term balances the trade-off between fitting the data well and keeping the model simple with fewer predictors, driving some of them to be exactly zero. The lasso methodology involves estimating the prediction error and identifying the optimal shrinkage degree, with the tuning parameter playing a crucial role in modulating the regularization intensity [Tib96].

In our analysis, we apply the lasso regression model to our dataset. To determine the optimal regularization parameter, which balances the trade-off between complexity and fit of the model, we utilize a 10-fold cross-validation approach. The R code snippet for this procedure is as follows:

```
x <- model.matrix(income ~ . - 1, data = training_set)
y <- training_set$income
cv_lasso <- cv.glmnet(x, y, family = "binomial", alpha = 1,
                      standardize = TRUE, nfolds = 10)
best_lambda <- cv_lasso$lambda.min
```

The `best_lambda` parameter is chosen as the value that minimizes the cross-validation error, which in this case is 0.00017. The corresponding plot, Figure 3.2, illustrates the relationship between different values of lambda and their associated cross-validation errors. The red dots indicate the mean error for each lambda, with bars representing one standard error. The value of `best_lambda` coincides with the minimum point on the curve, suggesting it as the appropriate level of model complexity for our dataset.

Using the `best_lambda`, the lasso regression is executed. This process results in the exclusion of the `Gov` category from the *workcalss* variable in the model. The removal of `Gov` category suggests that this level does not significantly contribute to the predictive performance when other variables are considered. The performance of the lasso regression model is very similar to that of the baseline logistic regression model across all metrics (accuracy: 84.674%). There are marginal differences, with slight improvements in accuracy, sensitivity, and specificity, and a marginal decrease in precision for the lasso model. Considering that the lasso regression model achieves a performance comparable to that of the regular logistic regression while simultaneously reducing the feature set by excluding the `Gov` category, one might conclude that the lasso model is more efficient.

Similarly, the ridge regression analysis is conducted to examine its predictive performance. The objective function for ridge regression, which includes a penalty proportional to the

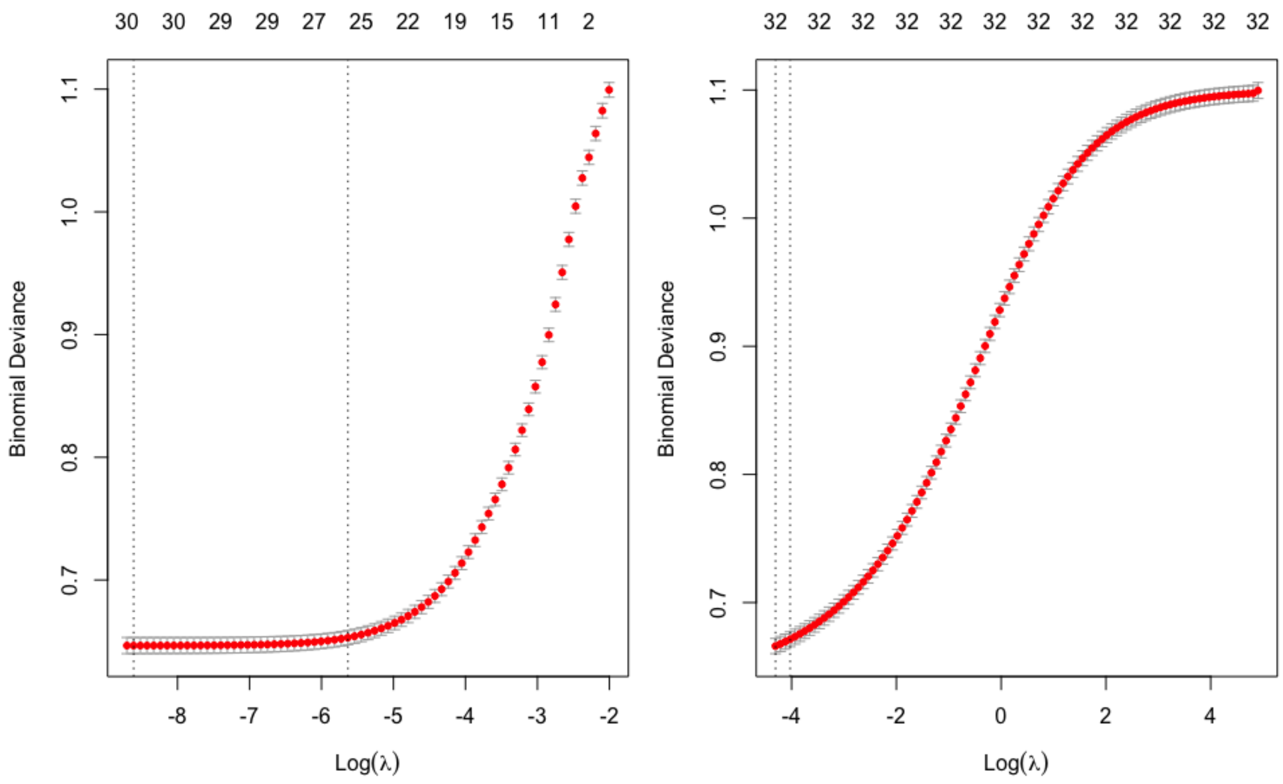


Figure 3.2: Cross-Validation Plots for Lasso (left) and Ridge (right)

square of the magnitude of the coefficients, is expressed as:

$$L_{\text{Ridge}}(\beta) = \underbrace{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2}_{\text{Loss}} + \underbrace{\lambda \sum_{j=1}^{p}\beta_j^2}_{\text{Penalty}} \tag{3.4}$$

The penalty term is controlled by the parameter λ, where larger values of λ result in greater shrinkage of the coefficients towards zero. The optimal value of λ, which we refer to as `best_lambda`, is determined using 10-fold cross-validation via the `cv.glmnet` function. The `best_lambda` for our ridge model is 0.01347416. Upon fitting the model with this optimal lambda, it demonstrates an accuracy of 84.122%. The performance metrics, detailed in Table 3.3, reveal that the ridge regression model, while performing slightly below the lasso model in terms of accuracy and precision, still maintains a strong level of predictive performance.

Metric	**Lasso**	**Ridge**
Accuracy	84.674%	84.122%
Sensitivity	74.330%	74.788%
Precision	58.530%	54.449%
Specificity	87.750%	86.183%
F1 Score	65.491%	63.019%
AUC	90.504%	90.122%
RMSE	0.323	0.329

Table 3.3: Comparison of Performance Metrics Between Lasso and Ridge Regression Models

The slight decrease in performance relative to lasso regression can be attributed to the fundamental differences in how ridge regression manages regularization. Unlike lasso, ridge regression does not reduce coefficients to absolute zero but rather shrinks them, allowing for the retention of all variables in the model. This characteristic implies that ridge regression is less effective at feature selection, which can be both a strength and a limitation [Tib96].

3.4 Naive Bayes

A classifier is a function f that maps input feature vectors $x \in X$ to output class labels $y \in \{0, 1\}$, where X is the feature space. In our case, the class labels represent binary income levels: 0 for income $\leq 50k$ and 1 for income $> 50k$. Naive Bayes operates under the generative modeling approach, where the class-conditional probabilities $p(x \mid y)$ for each class y, along with the class priors $p(y)$, are learned. The Naive Bayes classifier then applies Bayes' rule to compute the posterior probability [Mur06]:

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)} = \frac{p(x \mid y)p(y)}{\sum_{y'} p(x \mid y')p(y')} \tag{3.5}$$

This classifier assumes that all features x_i in the feature vector x are conditionally independent given the class label y. Therefore, the class-conditional probability simplifies to:

$$p(x \mid y) = \prod_{i=1}^{n} p(x_i \mid y) \tag{3.6}$$

Based on this assumption, the classification decision is made by selecting the class y that maximizes the posterior probability:

$$\hat{y} = \arg\max_{y} \left(p(y) \prod_{i=1}^{n} p(x_i \mid y) \right) \tag{3.7}$$

Naive Bayes simplifies the computational process by eliminating the need to estimate covariances among features. However, this classifiers can show decreased performance when there is significant correlation among predictor variables [BCR21]. As observed in our EDA and Figure 2.3, there are high correlations amongst certain variables, such as *sex* and *relationship*, which have been retained for their distinct contextual contributions despite this interdependence.

When we apply the Naive Bayes model to our data, the impact of these correlations manifests in the model's performance metrics. The model yields an accuracy of approximately 80.759%, sensitivity of 70.092%, and an F1 score of 50.668%. Although these figures indicate a reasonably good predictive ability, the precision of the model stands at around 39.673%, suggesting a considerable number of false positives within the predictions. This is because Naive Bayes assumes conditional independence between features, and the presence of correlation violates this assumption [BCR21]. Despite these challenges, the RMSE of 0.399 and AUC of 86.378% indicate the model's decent discrimination capacity.

In contrast, the logistic regression we fitted presents with an accuracy of 84.623%, and notably higher precision at 58.591%, alongside an F1 score of 65.447%. These metrics show that logistic regression may have an advantage over Naive Bayes in scenarios with notable feature interrelations. The observed discrepancy between the performance of the Naive Bayes and logistic regression models underscores the importance of considering feature correlation in model selection. While Naive Bayes offers a straightforward, computationally efficient approach, the correlation of predictor variables in the UCI Adult dataset suggests that models capable of handling feature covariance might provide more robust predictions.

3.5 KNN

The K-Nearest Neighbors (KNN) classification method is based on the idea that the closest data points to a given target x provide useful label information. In KNN, the class label is determined by the majority label among the K-nearest data points in the feature space. A key aspect of this methodology involves the selection of K, the number of neighbors considered, which directly impacts classification performance [Kra13]. Additionally, when applying KNN to datasets where features are scaled differently, the disparity in value ranges can disproportionately influence those attributes with smaller scales. To mitigate this issue, data normalization is essential. Techniques such as Min-Max normalization and Z-score normalization standardize attribute scales, ensuring no single feature disproportionately influences the classification outcomes [HWR21]. For this study, Min-Max normalization is applied to the features of our dataset as follows:

$$X_{\text{new}} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{3.8}$$

Categorical variables, such as *sex* and *race*, are encoded numerically. The dataset is subsequently divided into subsets for numerical and categorical variables, which are processed accordingly. The `preProcess` function from the `caret` package is utilized to normalize the numerical data. Training and test datasets are created, ensuring balanced class representation. Both sets are normalized to align feature scales. The KNN model's hyperparameter K is optimized using grid search, with cross-validation to evaluate accuracy. The optimal number of neighbors is determined to be 33, as indicated in Figure 3.3, where the model's accuracy peaks, confirming the effectiveness of the chosen K.

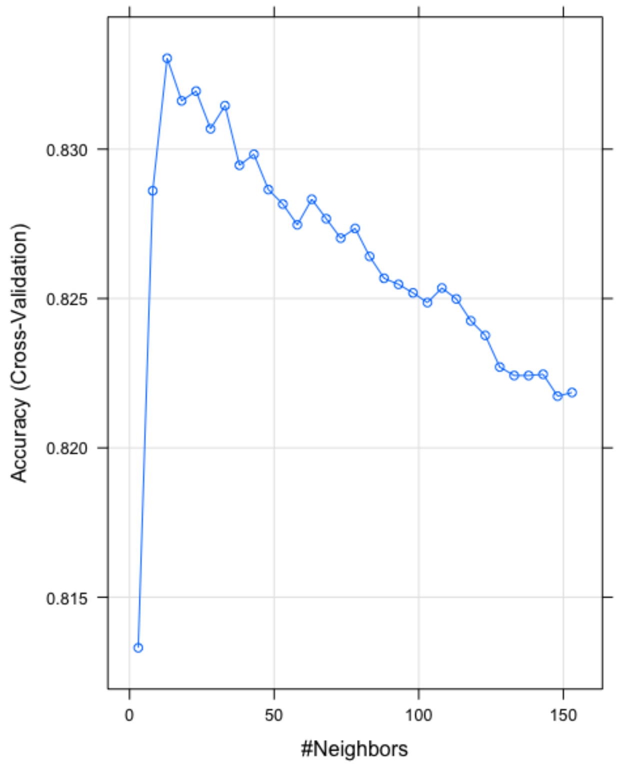


Figure 3.3: Accuracy of the KNN model for varying k

In comparing the performance of the optimized KNN model with the baseline logistic regression model, there are notable differences in their metrics. The logistic regression exhibits superior performance in almost all aspects. It achieves a higher accuracy at 84.623% compared to 83.108% for KNN, indicating a slightly better overall rate of correctly classified instances. Similarly, the sensitivity is improved in the logistic regression model, which is at 74.120% over the KNN's 70.988%, suggesting that logistic regression is more effective at identifying true positives. Precision stands at 58.591% for logistic regression, outperforming the KNN's 54.141%. The specificity of logistic regression, indicating its ability to identify true negatives, also exceeds that of the KNN model, with a value of 87.197% against 85.941%. The F1 score, which balances the trade-off between precision and sensitivity, is higher for the logistic regression at 65.447%, compared to the KNN's 61.430%. Furthermore, logistic regression yields a lower RMSE value of 0.323, suggesting that the predicted probabilities of the positive class are, on average, closer to the true outcomes than those of the KNN model, which has an RMSE of 0.337. Lastly, the area under the ROC curve (AUC) for logistic regression is 90.496%, indicating a better model performance than the KNN's AUC of 88.580%. Overall, while the KNN model demonstrates solid performance, especially considering its non-parametric nature, the regular logistic regression model appears to be more predictive for this particular dataset.

3.6 Decision Trees

3.6.1 Constructing a Baseline Decision Tree

A decision tree utilizes predictor variables to determine the class labels of a target variable. The classification begins at the root node and recursively continues down to the leaf nodes, where the final class labels are assigned. At each node, a split condition determines whether the input value should proceed to the left or right subtree. The objective of these conditions is to create as homogenous subsets as possible. However, achieving completely homogenous subsets is challenging with real-world data, which typically contains some degree of class mixing. Thus, the aim at each node is to choose a split condition that most effectively segregates the dataset into homogeneous subsets. This selection is guided by a "goodness of split" criterion, based on the concept of impurity, which quantitatively evaluates how well a condition splits the data. The criterion for measuring impurity includes several indices such as the Gini index, Information Gain, Gain Ratio, and Misclassification Rate [Tan20]. In this paper, we use the Gini Index as a measure of impurity for determining the splits within the decision tree. The Gini Index is calculated using the formula:

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} p_i^2 \tag{3.9}$$

where p_i represents the proportion of the class i in a given node. This index measures the frequency at which any element of the dataset will be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset, providing a scale of impurity at each node of the decision tree. After fitting the decision tree to our dataset, we observe the following structure and detailed insights in the resulting decision tree (Figure 3.4) :

- The initial split in the decision tree is based on *capital_gain*, dividing the dataset into groups with the values less than or equal to 5,119 and those with values greater. This split indicates that *capital_gain* is a significant predictor in our model.
- Various other attributes influence the path taken through the tree, including *marital_status*, *education*, and *occupation*.
- Additional conditions that appear in lower nodes of the tree include *capital_loss* and *age*, suggesting these factors also play critical roles in the classification process.

Each leaf node in the tree represents a terminal classification outcome, displaying the class proportions, which reveal the likelihood of an individual belonging to a specific class based on the pathways leading to that node. For instance, if an individual's *capital_gain* is less than or equal to 5,119, and they have never been married, they would follow the left branches down the tree to a particular leaf node. In the context of the target variable—where 0 represents an income of $50K$ or less, and 1 represents an income above $50K$—the 0 at the final node indicates the predicted class for the income level. The figure 32% suggests that this node accounts for 32% of the samples, while the value 0.04 indicates that 4% of the samples in this node are classified as having an income above $50K$. Given that the final class prediction at this node is 0, it implies that individuals in this node are predominantly predicted to have an income of $50K$ or less. The decision tree's major performance metrics are as follows: accuracy of 84.177%, sensitivity of 71.104%, precision of 61.438%, and specificity of 87.762%. These results will later be compared with those from pruned and tuned versions of the tree in Table 3.4.

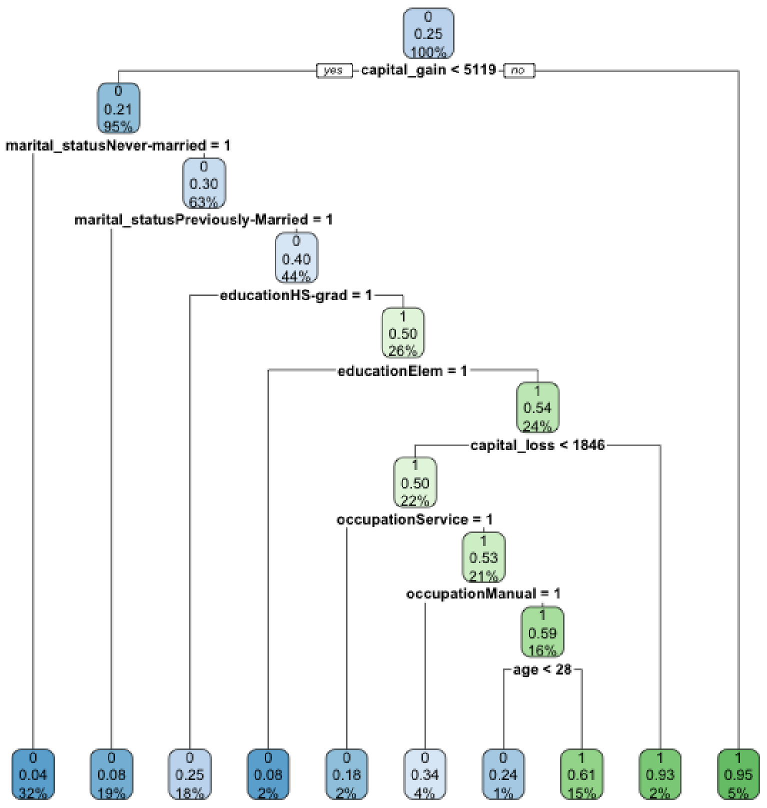


Figure 3.4: Decision tree demonstrating the classification process

3.6.2 Optimizing with Tree Pruning

Tree pruning involves removing parts of the tree that do not contribute significantly to its predictive accuracy. Typically, decision trees are constructed by initially growing a comprehensive tree and subsequently pruning it. Pruning is essential because a fully developed tree can overfit the data, resulting in complex, unnecessary structures that do not aid in classifying instances. Pruning aims to mitigate overfitting by reducing the tree to a version that yields the lowest error rate on new, unseen data, with a preference for simpler, smaller trees [Bre17]. In this study, we apply cost-complexity pruning as described by Breiman et al. [Bre17], which is designed to balance the trade-off between the tree's size and its prediction accuracy to prevent overfitting while enhancing interpretability. The cost-complexity pruning method entails a two-stage process where a sequence of increasingly simplified trees, from T_0 to T_k is created. T_0 is the initial, unpruned tree, and each subsequent tree is formed by replacing one or more of subtrees with leaves, continuing until T_k becomes a single leaf. Each iteration assesses the pruned tree's performance based on a cost-complexity metric defined as:

$$R(T) = E + \alpha L(T) \tag{3.10}$$

where R(T) represents the cost-complexity, E is the error count from the training set, L(T) is the count of leaves in the tree, and α is a regularization parameter that influences the severity of penalties imposed for tree complexity. The objective is to minimize this penalized function, effectively balancing the accuracy of the tree against its simplicity. We follow the same algorithm to optimize the tree with cost-complexity pruning method. First, we grow a large, unpruned tree to determine the complexity parameter that minimizes the cross-validated error. Once the optimal cost complexity parameter is identified, the tree is pruned to reduce overfitting while maintaining predictive accuracy. For our dataset, the minimum cost-complexity parameter (`min_CP`) is determined to be 0.000723. This parameter's effectiveness in optimizing tree complexity is demonstrated in Figure 3.5, which highlights the relationship between tree complexity and cross-validated error, underlining the optimal complexity trade-offs. The following R code illustrates the process of identifying the minimum cost-complexity parameter, pruning the decision tree, and predicting using the pruned model:

```
min_CP <- large_tree$cptable[which.min(large_tree$cptable[,"xerror"]), "CP"]
pruned_model <- prune(large_tree, min_CP)
pruned_predictions <- predict(pruned_model, testing_set, type = "class")
```

Following the optimization of the decision tree using the cost-complexity pruning method, it is important to compare the performance of the original unpruned tree and the pruned tree. The unpruned tree, while more complex and potentially better fitting the training data, tends to overfit, leading to poorer performance on unseen data. In contrast, the pruned tree, optimized with a `min_CP` of 0.000723, demonstrates better generalization across various metrics. In Table 3,4, the pruned tree not only shows improved accuracy but also exhibits higher sensitivity and specificity, indicating a more balanced detection capability for both classes. The reduction in RMSE and the increase in AUC further corroborate the enhanced predictive power and error reduction in the pruned model.

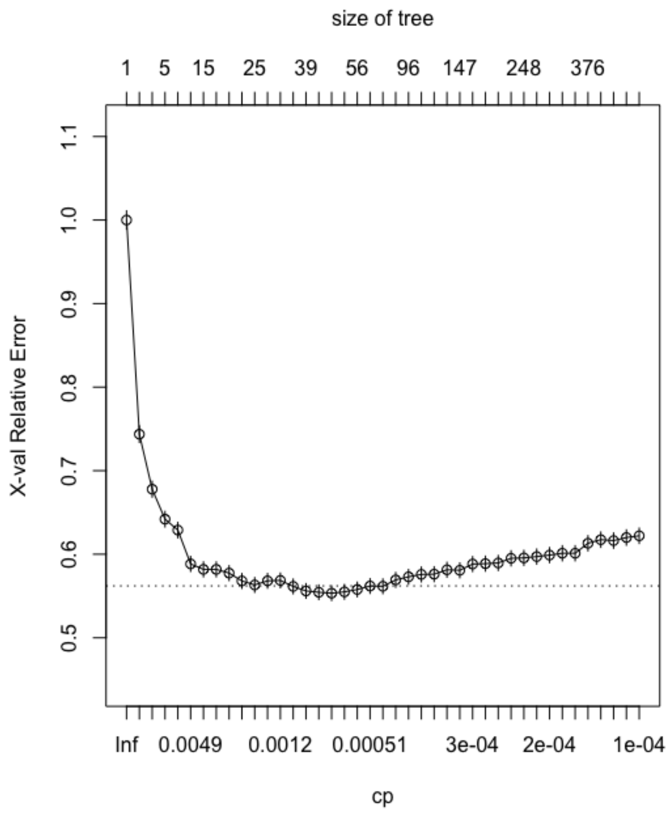


Figure 3.5: Decision Tree Error vs. Complexity Parameter

3.6.3 Enhancing Predictive Power through Tree Tuning

Hyper-parameter optimization is crucial in building effective machine learning models, as it significantly influences model performance. Traditional methods like grid search are often computationally expensive, particularly with models having multiple hyper-parameters. Recent advancements in hyper-parameter optimization emphasize the efficiency of random search methods, especially when paired with parallel computing strategies [BBB11]. This section discusses the implementation of a parallelized random search approach to optimize the hyper-parameters of a decision tree classifier. The R code snippet below demonstrates the application of a parallelized random search method using the `mlr` package for a classification task. The target is to optimize the decision tree model's performance by tuning various hyper-parameters such as `minsplit`, `minbucket`, `cp`, and `maxdepth`.

```
treeTask <- makeClassifTask(data = df, target = "income")
tree <- makeLearner("classif.rpart")
treeParamSpace <- makeParamSet(
  makeIntegerParam("minsplit", lower = 10, upper = 50),
    makeIntegerParam("minbucket", lower = 5, upper = 20),
  makeNumericParam("cp", lower = 0.0002, upper = 0.001),
  makeIntegerParam("maxdepth", lower = 5, upper = 20))
randSearch <- makeTuneControlRandom(maxit = 200)
cvForTuning <- makeResampleDesc("CV", iters = 5)
parallelStartSocket(cpus = detectCores())
tunedTreePars <- tuneParams(tree, task = treeTask,
                            resampling = cvForTuning,
                            par.set = treeParamSpace,
                            control = randSearch)
parallelStop()
> tunedTreePars
Tune result:
Op. pars: minsplit=31; minbucket=5; cp=0.000254; maxdepth=9
mmce.test.mean=0.1372807
```

Random search is employed for optimization, configured to perform 200 iterations. Cross-validation with five iterations is used to evaluate model performance. To enhance computational efficiency, the `parallelStartSocket` function activates parallel processing using all available CPU cores, allowing simultaneous evaluations across different sets of hyper-parameters. This parallel execution not only shortens the optimization timeline but also broadens the exploration of the parameter space, increasing the probability of identifying optimal configurations. Upon completion of the hyper-parameter tuning process, the best parameter combination identified is a minimum split of 31, minimum bucket size of 5, complexity parameter of 0.000254, and a maximum depth of 9. This configuration results in a mean misclassification error (mmce) on the test dataset of 0.1372807, showcasing the effectiveness of the tuned parameters in enhancing the decision tree model's performance.

Metric	**Baseline Tree**	**Pruned Tree**	**Tuned Tree**
Accuracy	83.323%	86.271%	86.210%
Sensitivity	70.060%	79.335%	77.950%
Precision	57.417%	60.507%	62.052%
Specificity	86.714%	87.893%	88.247%
F1 Score	63.111%	68.654%	69.098%
AUC	86.380%	89.065%	89.324%
RMSE	0.342	0.319	0.317

Table 3.4: Comparison of Performance Metrics For Regular, Pruned, and Tuned Trees

Table 3.4 demonstrates that the hyper-parameter tuning process successfully optimized the decision tree model, yielding significant improvements in several key performance metrics when compared with the baseline (untuned) and pruned tree models. The comparison of some key performance metrics is summarized as follows:

- The tuned tree achieves an accuracy of 86.210%, which is a marginal improvement over the regular tree's accuracy of 83.323% and slightly below the pruned tree's accuracy of 86.271%.

- Sensitivity shows a notable improvement in the tuned tree at 77.950%, compared to 70.060% in the regular tree and 79.335% in the pruned tree, indicating better performance in identifying true positive rates.

- The AUC, which measures the ability of the model to avoid false classification, increases from 86.380% in the baseline tree to 89.324% in the tuned tree, surpassing the pruned tree's 89.065%.

These improvements demonstrate that the tuned tree, configured via parallelized random search, outperforms the regular tree in nearly all metrics and exhibits comparable or slightly better performance than the pruned tree. The slight variances in performance between the tuned and pruned trees suggest that while pruning effectively reduces overfitting and complexity, tuning specific hyper-parameters can provide a more balanced approach across various measures of performance.

3.7 Random Forest

3.7.1 Baseline Random Forest

The Random Forest algorithm is an ensemble learning method for classification (and regression) that constructs a multitude of decision trees at training time and outputs the mode of the classes (for classification) or the mean prediction (for regression) of the individual trees [Bre01]. Random Forest constructs an ensemble of decision trees, each generated from bootstrap samples of the training dataset. These samples are created through random selection with replacement, ensuring diversity among the trees in the forest. After constructing the forest, predictions for classification tasks, such as ours, are determined by majority voting among the outputs of all the trees. The Random Forest model is as follows:

$$\mathrm{RF} = \frac{1}{N}\sum_{i=1}^{N} h(x, \theta_i) \tag{3.11}$$

where N is the number of trees. $h(x, \theta_i)$ represents the prediction of the i-th tree and x is the feature vector. The θ_i are independently sampled vectors defining the structure and splits of each tree based on the training data [Bre01]. This highlights the ensemble nature of the model, where multiple decision trees vote to determine the final class output. In our study, we utilize the `randomForest` package to build a baseline Random Forest model. The model is configured to build 1000 trees, a choice intended to leverage the strengths of multiple trees in making robust predictions. The performance metrics of this baseline model are summarized in Table 3.5, which provides a comparison with the tuned Random Forest model discussed later in this section. The Random Forest model outperforms the baseline logistic regression model in several key metrics, including accuracy, sensitivity, precision, specificity, F1 score, and RMSE. However, the AUC metric, which measures the overall ability of the model to discriminate between positive and negative classes, is similar for both models. This suggests that while the Random Forest model provides a slight edge in classification performance.

3.7.2 Random Forest with Tuned Parameters

Hyperparameter tuning in Random Forest algorithms is crucial as it seeks an optimal balance between model accuracy and computational efficiency. According to Probst et al., the following hyperparameters are key to enhancing Random Forest model performance [PWB19]:

- `mtry`: Controls the number of features considered for each split, balancing between tree diversity and accuracy.
- `nodesize`: Determines the minimum samples a node must contain before it can split, influencing tree depth and complexity.
- `ntree`: Represents the number of trees in the forest. More trees generally improve model performance by reducing variance, but benefits diminish with very high numbers.
- `maxnodes`: Limits the maximum number of terminal nodes, controlling tree growth and complexity to prevent overfitting in complex datasets.

To identify the optimal combination of hyperparameters, a parallelized random search is employed, similar to the tree tuning approach previously described. Upon concluding the hyperparameter tuning process, the most effective parameter combination identified is `mtry` of 6, `nodesize` of 33, `ntree` of 286, `maxnodes` of 20. This configuration results in a mean misclassification error (mmce) of 0.1879804. As summarized in Table 3.5, the comparison highlights the effects of tuning on various key performance indicators. The table reflects the trade-offs and enhancements resulted from the tuning. Notably, while accuracy and

Metric	**Baseline RF**	**Tuned RF**
Accuracy	86.410%	80.876%
Sensitivity	77.786%	77.283%
Precision	63.412%	87.460%
Specificity	88.600%	85.558%
F1 Score	69.867%	82.057%
AUC	90.920%	94.864%
RMSE	0.315	0.360

Table 3.5: Comparison of Performance Metrics For Baseline and Tuned Random Forest Models

specificity show slight declines, precision and F1 score improve significantly, indicating enhanced positive predictive value and a better balance between precision and recall. Figure 3.6 demonstrates that the AUC also shows a notable improvement, underscoring an enhanced ability of the model to distinguish between the classes effectively. The tuned model not only outperforms the baseline model in terms of AUC but also demonstrates a consistently higher true positive rate across almost all thresholds of false positive rate. These results substantiate the impact of systematic hyperparameter tuning in optimizing model performance, particularly in complex datasets where default parameters do not suffice.

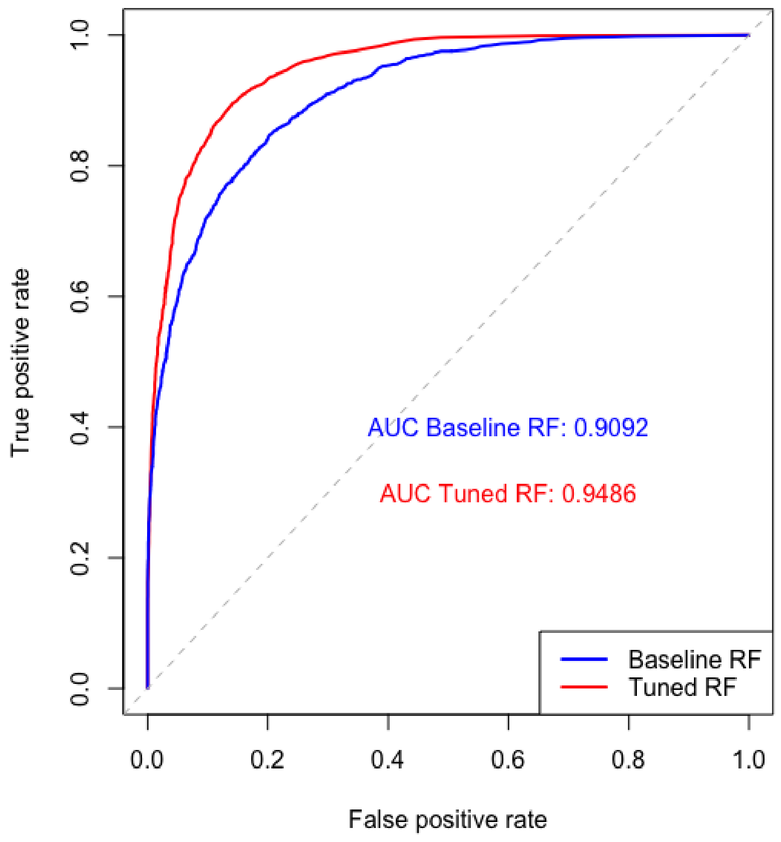


Figure 3.6: ROC comparison between Baseline RF and Tuned RF

3.8 Support Vector Machine

Support Vector Machines (SVMs) represent a sophisticated computational approach designed for the classification and regression of complex datasets. SVMs operate by transforming input data vectors non-linearly into a high-dimensional feature space. Within this space, the algorithm constructs a linear decision surface or hyperplane optimized to segregate two or more classes of data with the maximum possible margin. The theoretical framework discussed here is founded on the principles of Reproducing Kernel Hilbert Spaces (RKHS) as outlined in seminal works by Aronszajn (1950) [Aro50], Girosi (1997) [OFG97], Heckman (1997) [HIT97], and Wahba (1990) [Wah90]. These studies propose that an inner product in feature space corresponds to an equivalent kernel in the input space, given by [Gun98]:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \tag{3.12}$$

This efficiency in separation is attributed to "support vectors," which are the nearest data points to the hyperplane, defining its position and orientation. These vectors are not only help in maximizing the margin and ensure the model captures essential patterns for robust generalization [VGS96]. Figure 3.7 visually supports these concepts. The left plot shows

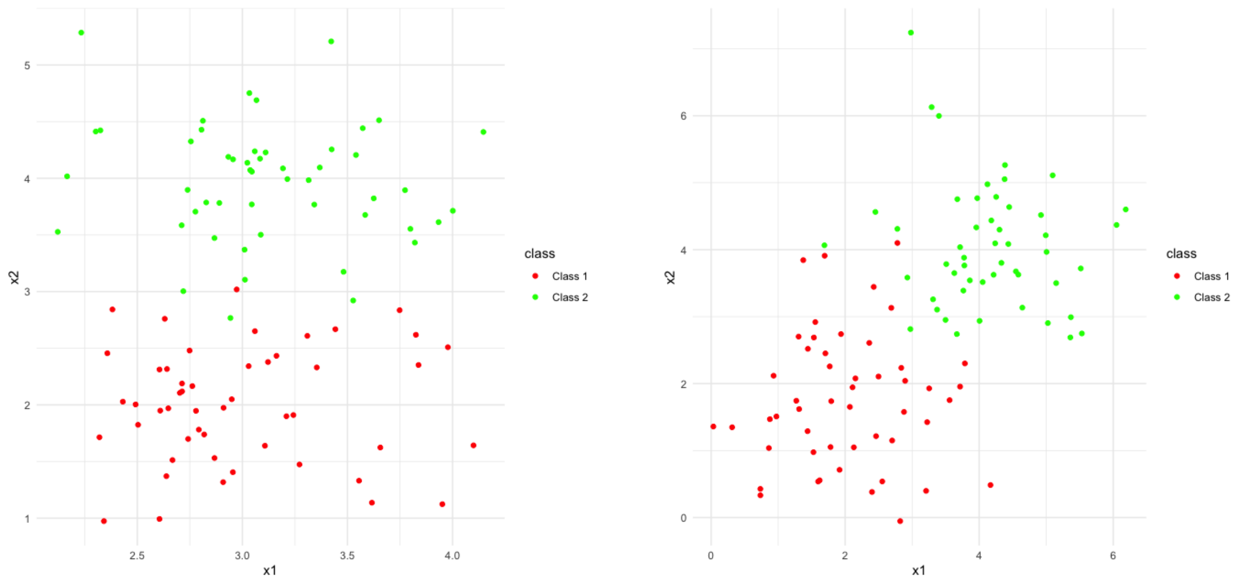


Figure 3.7: Linearly Separable (left) vs. Not Linearly Separable (right)

two distinct groups where a line could feasibly separate the lower cluster of red points from the upper cluster of green points. Conversely, the right plot shows some overlap between the green and red points, especially around the central region where the green and red points are close together. If there is indeed overlap, as it seems, this plot would not be linearly separable. This overlap indicates a lack of linear separability, highlighting the necessity for more sophisticated approaches such as kernel functions. The core innovation of the SVM lies in its use of kernel functions, which facilitate the high-dimensional transformation of data. Table 3.6 lists common kernel functions. The linear kernel is the simplest type of kernel function, which is appropriate when the data is linearly separable. The polynomial kernel allows for learning non-linear models by raising the linear kernel to the power of d degree of the polynomial. This can capture more complex interactions between features. The RBF kernel measures the exponential decay of the Euclidean distance between two data points. The parameter γ controls the width of the kernel and hence the decision boundary. The RBF kernel is highly versatile and can model complex non-linear relationships.

Kernel Functions	
Linear	$K(x_i, x_j) = x_i \cdot x_j$
Polynomial	$K(x_i, x_j) = (1 + x_i \cdot x_j)^d$
Radial Basis Function	$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$

Table 3.6: Comparison of Kernel Functions

In this paper, we explore the application of Support Vector Machines (SVMs) to our dataset, implementing two distinct SVM models to predict income levels. The goal is to compare a baseline SVM model with a tuned SVM model, using different R packages and tuning strategies to optimize performance.The initial SVM model is constructed using the `kernlab` package. This model utilizes the Radial Basis Function (RBF) kernel, configured with automatically tuned kernel parameters. Specific details of the model setup include:

- Kernel: RBF ("rbfdot"), with parameters set to "automatic" for optimal adaptation to data.
- Regularization: The cost parameter C is set to 1, balancing the trade-off between achieving a low error on the training data and minimizing the model complexity.
- Scaling: Input features are scaled to enhance model performance, crucial for kernel-based SVM.
- Cross-Validation: Employee 5-fold cross-validation within the training process to ensure robustness and reduce overfitting.

For the second SVM model, we utilize the `e1071` package in R, focusing on optimizing the cost and gamma parameters through a systematic tuning process with the `caret` package. The process involves key steps. First, a tuning grid for the gamma and cost parameters is defined to cover a range of possible values. This grid is used to systematically explore different combinations of these parameters to find the optimal settings for the RBF kernel. Next, the `tune.svm` function from the `e1071` package is employed to perform the model tuning. This function explores the defined grid of gamma and cost parameters to identify the combination that resulted in the best model performance. The optimal combination is a cost of 11.31371 and a gamma of 0.0078125. Figure 3.8 illustrates that the results of this tuning process, where the red dot represents the best combination found.

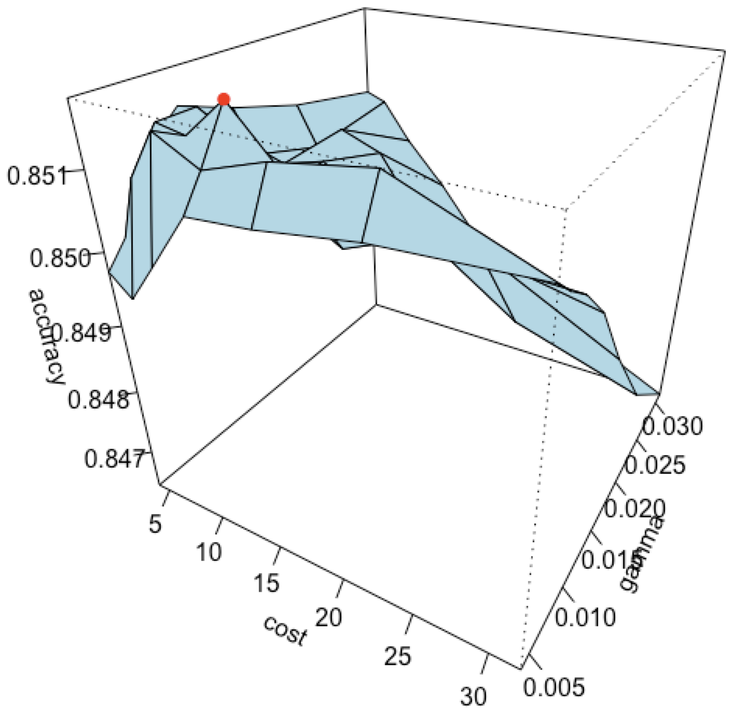


Figure 3.8: Performance of SVM Model with Different Hyperparameters

The hyperparameter tuning process for the SVM model generates mixed results compared to the baseline model, as shown in Table 3.7. The tuned model demonstrates a slight improvement in accuracy (84.843% vs. 84.747%) and sensitivity (76.314% vs. 75.330%), while the AUC increased marginally to (89.804% vs. 89.256%). Conversely, other metrics such as precision, specificity, and F1 score show slight decreases. These results indicate that the baseline SVM model is already close to optimal, and further tuning provides minimal improvements. The slight decreases in precision and specificity highlight the trade-offs inherent in model tuning.

Metric	Baseline SVM	Tuned SVM
Accuracy	84.747%	84.843%
Sensitivity	75.330%	76.314%
Precision	58.865%	56.551%
Specificity	87.061%	86.768%
F1 Score	66.088%	64.963%
AUC	89.256%	89.804%
RMSE	0.334	0.332

Table 3.7: Comparison of Performance Metrics For Baseline and Tuned SVM Models

3.9 Neural Network

Neural Networks are computational models inspired by the architecture of biological neurons, particularly those in the human brain [ZHS09]. They consist of interconnected nodes or neurons, each representing a specific function, referred to as the activation function. The connections between neurons are weighted, representing the strength of the signal passed through the connection [WF18]. In a neural network, information is processed in layers. The process begins with the input layer, which receives the initial data. This data is then passed through one or more hidden layers, where the network performs intermediate computations and feature extraction. Each hidden layer typically applies a nonlinear activation function, which allows the network to learn complex patterns. One common activation function is the sigmoid function, defined as:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{3.13}$$

As demonstrated in Figure 3.9, the sigmoid function maps all the range of x into $[0, 1]$. Finally, the data reaches the output layer, where the network produces its final prediction.

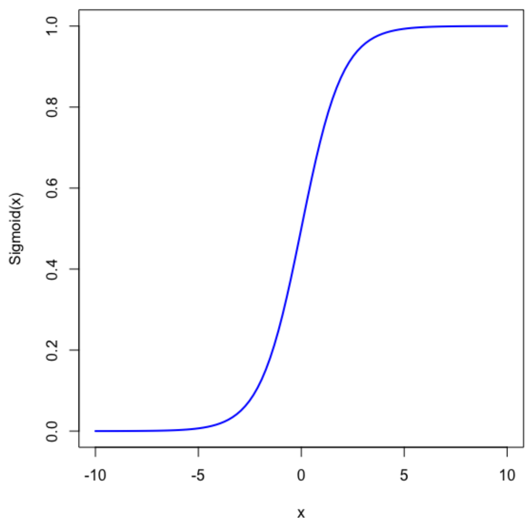


Figure 3.9: Sigmoid Function

3.9.1 Baseline Neural Network

In our analysis, we construct a one-hidden-layer neural network model, featuring a single hidden layer between the input and output layers. 32 input nodes are obtained through the conversion of categorical variables into numeric ones using `dummyVars`. Within the hidden layer, we utilize 20 neurons to facilitate the extraction of meaningful features from the input data. The general structure of a neural network is illustrated in Figure 3.10, showing how inputs are processed in layers. During the training process, we use a maximum of 10,000 iterations to refine the model parameters and minimize the error. To prevent overfitting, a weight decay of 0.001 is incorporated, following the recommendation by Dreiseitl et al. [DO02], who highlights that neural network models offer greater flexibility compared to logistic regression but are more prone to overfitting. The size of the network can be controlled by reducing the number of variables and hidden neurons, or by pruning the network after training. Alternatively, smoothness in the model output can be enforced through regularization, known as weight decay in the context of neural networks. This regularization technique penalizes larger weights in the network, encouraging the maintenance of smaller weights and

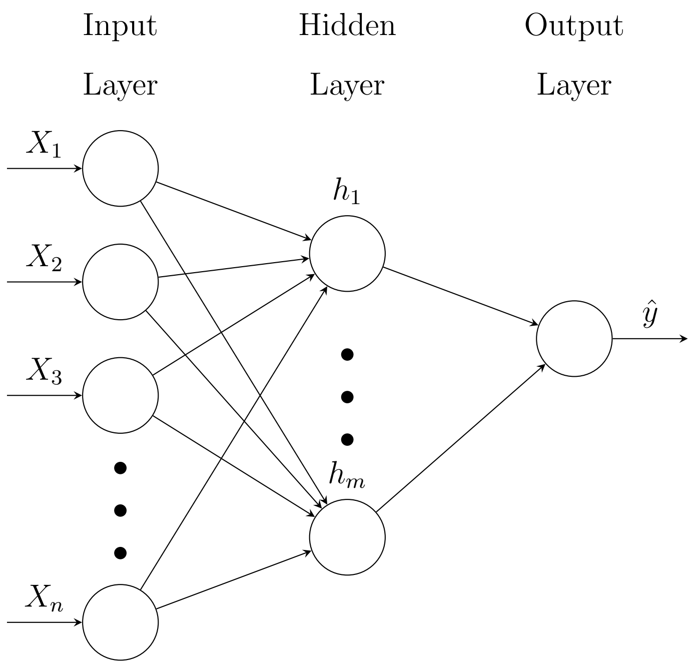


Figure 3.10: Neural Network Diagram

a simpler model structure. Given an input vector $(x_1, \ldots, x_{32})$ the output of the neural network with a single hidden layer of 20 neurons can be represented as follows:

$$z_i = \sum_{j=1}^{32} w_{ij}^{(1)} x_j + b_i^{(1)} \tag{3.14}$$

$$h_i = \sigma(z_i) \tag{3.15}$$

$$y_k = \sum_{i=1}^{20} w_{ki}^{(2)} h_i + b_k^{(2)} \tag{3.16}$$

where:

- x_j denotes the input nodes obtained through the conversion of categorical variables into numeric ones using `dummyVars`.
- h_i represents the activation of the i-th neuron in the hidden layer.
- $\hat{y}$ represents the predicted output of the network.

- $\sigma(\cdot)$ is the sigmoid function, defined as $\sigma(z) = \frac{1}{1+\exp(-z)}$.

Before fitting the neural network model to the training data, categorical variables are converted into numeric ones, as illustrated in Table 3.8. For example, the *marital_status* feature is encoded with "Married" serving as the base category and "Never-married" represented as $[1, 0, 0, 0]$. This type of encoding prevents multicollinearity issues while including $n - 1$ dummy variables for n different levels.

Marital Status	**Never-married**	**Divorced**	**Separated**	**Widowed**
Married	0	0	0	0
Never-married	1	0	0	0
Divorced	0	1	0	0
Separated	0	0	1	0
Widowed	0	0	0	1

Table 3.8: Example of Dummy Variables for *marital_status*

The performance of the baseline neural network model is similar to the baseline logistic regression across most metrics, with the logistic regression achieving slightly higher accuracy (84.623% vs. 84.459%) and AUC (90.296% vs. 90.198%), along with better sensitivity (74.120% vs. 73.095%). The neural network model has slightly higher precision (59.271% vs. 58.591%) and specificity (87.327% vs. 87.197%). Overall, both models exhibit balanced and comparable performance.

3.9.2 Tuned Neural Network

To further optimize the neural network model, we conduct a grid search over different parameter combinations, including different sizes of the hidden layer and different weight decay values. The hyperparameter grid includes sizes 10,20,30, and 50 for the hidden layer, and decay values 0.0001, 0.001, 0.01, and 0.1. The grid search identifies the optimal model configuration with a hidden layer size of 30 and a decay value of 0.1, achieving a performance of 85.197%. Figure 3.11 illustrates the performance of the neural network across different sizes and decay values, confirming that the most effective combination is a hidden layer size of 30 and a decay value of 0.1.

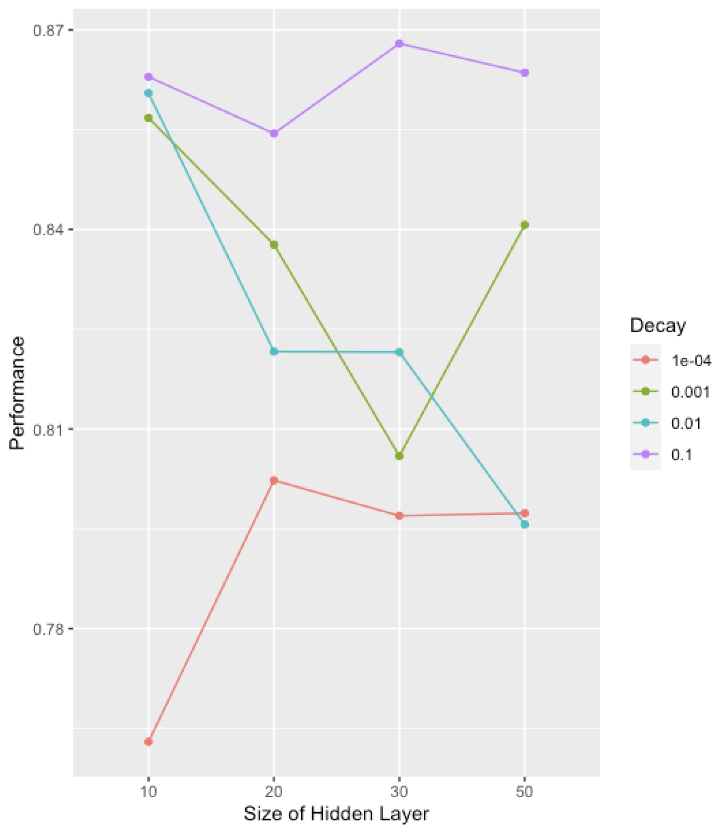


Figure 3.11: Performance of Models by Size and Decay

When comparing the baseline neural network model to the tuned neural network model, the latter performs better across all metrics, as shown in Table 3.9. The tuned neural network achieves higher accuracy (85.197% vs. 84.459%), sensitivity (74.258% vs. 73.095%), precision (61.867% vs. 59.271%), specificity (88.052% vs. 87.327%), F1 score (67.498% vs. 65.461%), and AUC (90.863% vs. 90.198%), while also having a lower RMSE (0.318 vs. 0.324). These consistent improvements across all key metrics demonstrate that the grid search successfully identifies a more optimal neural network configuration, leading to a model that not only performs better in terms of accuracy and balanced performance but also offers better discriminatory power and predictive accuracy.

Metric	**Baseline NN**	**Tuned NN**
Accuracy	84.459%	85.197%
Sensitivity	73.095%	74.258%
Precision	59.271%	61.867%
Specificity	87.327%	88.052%
F1 Score	65.461%	67.498%
AUC	90.198%	90.863%
RMSE	0.318	0.324

Table 3.9: Comparison of Performance Metrics For Baseline and Tuned Neural Network Models

CHAPTER 4

Conclusion

Many classifiers use accuracy as a metric to identify the optimal solution during classification training [HS15]. Accuracy measures the classifier based on overall correctness, which refers to the total number of instances correctly predicted when tested with unseen data. Table 4.1 shows that the baseline random forest model results in the highest accuracy of 94.410%. Table 4.1 demonstrates that the best model by each metric is:

- Accuracy: Baseline Random Forest - 86.410%
- Sensitivity: Pruned Tree - 79.335%
- Precision: Tuned Random Forest - 87.460%
- Specificity: Baseline Random Forest - 88.600%
- F1 Score: Tuned Random Forest - 82.057%
- AUC: Tuned Random Forest - 94.964%
- RMSE: Baseline Random Forest - 0.315

However, in several studies [RP06] [Wil00], it has been shown that the simplicity of accuracy can lead to suboptimal solutions, especially when dealing with imbalanced class distributions. The UCI Adult dataset is imbalanced, with 76% of instances having an income $\leq 50K$ and 24% having an income $> 50K$. To determine the best model for our dataset, which exhibits an imbalance in income levels, it's essential to consider multiple metrics due to the various aspects of performance each metric highlights. For example, precision is important when the cost of false positives is high. which might be relevant if focusing on those with incomes

$> 50K$. In contrast, specialty might serve the best when it is important to correctly identify individuals whose income does not exceed 50K. The AUC measures the classifier's ability to distinguish between positive and negative classes. A higher AUC value indicates better performance. The RMSE measures the average deviation between the predicted probabilities and the actual outcomes. For binary classification problems, this metric provides insight into how well-calibrated the probability estimates are. While accuracy measures overall correctness, the RMSE provides additional information about the model's prediction reliability.

Model	Acc.	Sens.	Prec.	Spec.	F1	AUC	RMSE
Logistic - B	84.623%	74.120%	58.591%	87.197%	65.447%	90.496%	0.323
Logistic - U	85.227%	73.264%	59.417%	88.078%	65.618%	90.294%	0.320
Logistic - R	80.988%	58.010%	85.043%	94.155%	68.972%	90.490%	0.359
Lasso	84.674%	74.330%	58.5330%	87.750%	65.491%	90.504%	0.323
Ridge	84.122%	74.788%	54.449%	86.183%	63.019%	90.122%	0.329
NB	80.759%	70.092%	39.673%	82.509%	50.668%	86.378%	0.399
KNN	83.108%	70.988%	54.141%	85.941%	61.430%	88.580%	0.337
Tree - B	83.323%	70.060%	57.417%	86.714%	63.111%	86.380%	0.342
Tree - P	86.271%	79.335%	60.507%	87.893%	68.654%	89.065%	0.319
Tree - T	86.210%	77.950%	62.052%	88.247%	69.098%	89.324%	0.317
RF - B	86.410%	77.786%	63.412%	88.600%	69.867%	90.920%	0.315
RF - T	80.876%	77.283%	87.460%	85.558%	82.057%	94.964%	0.360
SVM - B	84.747%	75.330%	58.865%	87.061%	66.088%	89.256%	0.334
SVM - T	84.843%	76.314%	56.551%	86.768%	64.963%	89.804%	0.332
NN - B	84.459%	73.095%	59.271%	87.327%	65.461%	90.198%	0.318
NN - T	85.197%	74.258%	61.867%	88.052%	67.498%	90.863%	0.324

Table 4.1: Model Performance Metrics. The model name abbreviations are as follows: B = Baseline, P = Pruned, T = Tuned, U = Upsampled, R = ROSE- Augmented.

The Random Forest models, particularly the baseline and tuned models, perform well across most metrics. The tuned Random Forest has the highest AUC and F1 Score, indicating strong overall performance and effective balance between precision and recall. The baseline Random Forest has the best accuracy, specificity, and RMSE, suggesting strong general performance and well-calibrated probabilities. Considering the overall performance, the tuned Random Forest can be considered the best model as it excels in critical metrics such as AUC and F1 Score, which are especially valuable when dealing with imbalanced datasets. Plots of residual-versus-fitted values are less effective for models with binary outcomes due

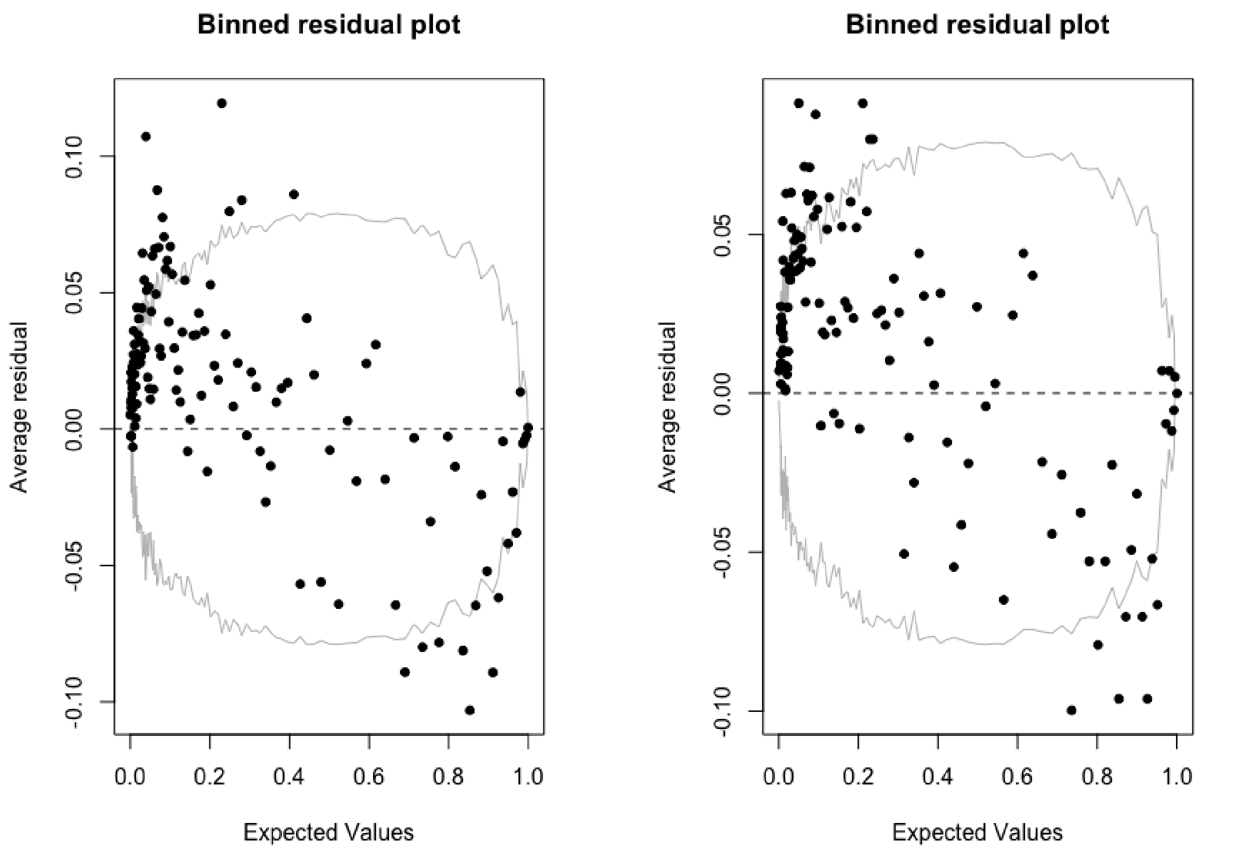


Figure 4.1: Comparison of Binned Residual Plots for Baseline (left) and Tuned (right) Random Forest Models

to the discrete nature of the residuals [Kas15]. Instead, binned residual plots, as suggested by Gelman and Hill (2007) [GH06], are advantageous for evaluating the overall fit of models for binary outcomes and considering the effect of continuous variables. These plots involve grouping observations into bins of equal size. For each bin, the 95% confidence interval is set at $(\frac{-2\sqrt{p(1-p)}}{n}, \frac{2\sqrt{p(1-p)}}{n})$, calculated using the standard deviation of the residuals in each bin. The comparative binned residual plots for the baseline and tuned Random Forest models are illustrated in Figure 4.1. The left plot shows slightly dispersed residuals around zero, with notable deviations, particularly in the middle and higher ranges of expected values. In contrast, the right plot shows a more centered clustering of residuals around zero, with deviations from zero are smaller compared to the left plot, further confirming the superior performance of the tuned Random Forest model. Moreover, the variable importance plot in

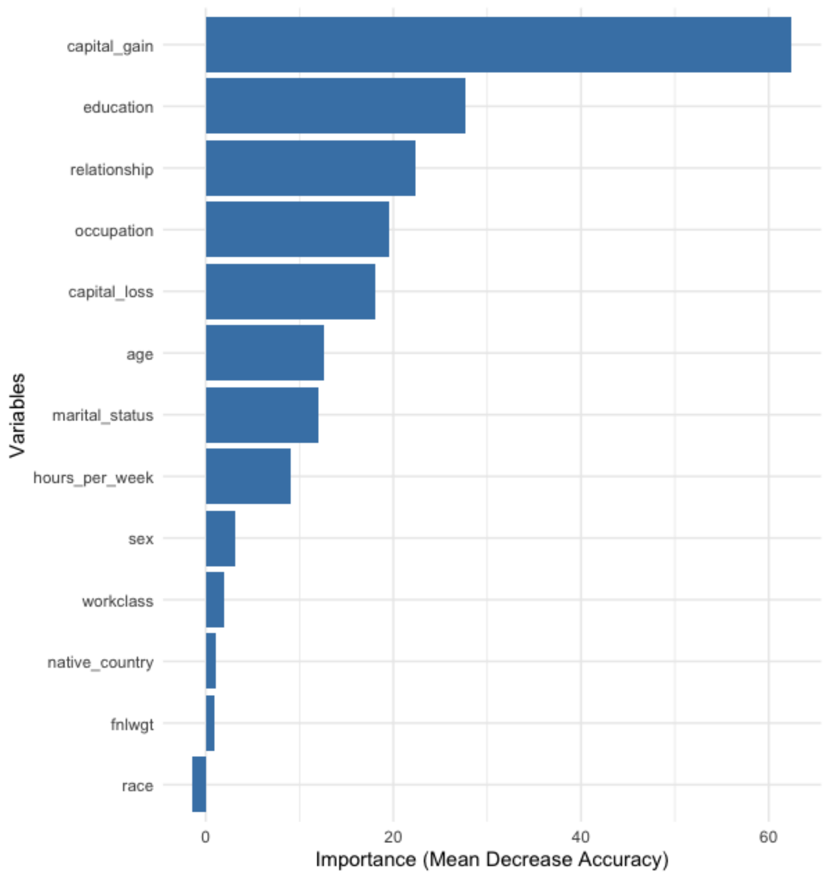


Figure 4.2: Variable Importance Plot for Indicating the Mean Decrease in Accuracy for Each Predictor

Figure 4.2 provides insights into which variables significantly impact the model's accuracy. Key variables such as *capital_gain*, *education*, *relationship*, *occupation*, and *capital_loss* are identified as the most influential in predicting income levels. For example, the *capital_gain* variable demonstrates the highest importance, suggesting that it is the most significant predictor of income levels in the model. Changes or variations in this variable are likely to have a substantial impact on the model's prediction accuracy. Understanding the influence of these predictors helps in refining model features and guiding further data collection and feature engineering.

This study underscores the effectiveness of advanced machine learning techniques, specifically Random Forest, in income prediction based on socio-economic data. The ability of the Random Forest model to handle high-dimensional data and exhibit the importance of various predictors is crucial for developing targeted policies and interventions. Future research could build upon these findings by exploring more complex interactions among these variables or by applying similar models to different demographic or geographic datasets to validate the findings.

REFERENCES

[Aro50] Nachman Aronszajn. "Theory of reproducing kernels." *Transactions of the American mathematical society*, **68**(3):337–404, 1950.

[ASR13] Aida Ali, Siti Mariyam Shamsuddin, and Anca L Ralescu. "Classification with class imbalance problem." *Int. J. Advance Soft Compu. Appl*, **5**(3):176–204, 2013.

[BBB11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization." *Advances in neural information processing systems*, **24**, 2011.

[BCR21] Rafael Blanquero, Emilio Carrizosa, Pepa Ramírez-Cobo, and M Remedios Sillero-Denamiel. "Variable selection for Naïve Bayes classification." *Computers & Operations Research*, **135**:105456, 2021.

[Bre01] Leo Breiman. "Random forests." *Machine learning*, **45**:5–32, 2001.

[Bre17] Leo Breiman. *Classification and regression trees.* Routledge, 2017.

[CBH02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." *Journal of artificial intelligence research*, **16**:321–357, 2002.

[DO02] Stephan Dreiseitl and Lucila Ohno-Machado. "Logistic regression and artificial neural network classification models: a methodology review." *Journal of biomedical informatics*, **35**(5-6):352–359, 2002.

[GH06] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models.* Cambridge university press, 2006.

[Gun98] Steve R Gunn et al. "Support vector machines for classification and regression." *ISIS technical report*, **14**(1):5–16, 1998.

[HIT97] James J Heckman, Hidehiko Ichimura, and Petra E Todd. "Matching as an econometric evaluation estimator: Evidence from evaluating a job training programme." *The review of economic studies*, **64**(4):605–654, 1997.

[HS15] Mohammad Hossin and Md Nasir Sulaiman. "A review on evaluation metrics for data classification evaluations." *International journal of data mining & knowledge management process*, **5**(2):1, 2015.

[HWR21] Henderi Henderi, Tri Wahyuningsih, and Efana Rahwanto. "Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer." *International Journal of Informatics and Information Systems*, **4**(1):13–20, 2021.

[Kas15] Jessica Kasza. "Stata tip 125: Binned residual plots for assessing the fit of regression models for binary outcomes." *The Stata Journal*, **15**(2):599–604, 2015.

[Kra13] Oliver Kramer et al. *Dimensionality reduction with unsupervised nearest neighbors*, volume 51. Springer, 2013.

[Mur06] Kevin P Murphy et al. "Naive bayes classifiers." *University of British Columbia*, **18**(60):1–8, 2006.

[OFG97] Edgar Osuna, Robert Freund, and Federico Girosi. "An improved training algorithm for support vector machines." In *Neural networks for signal processing VII. Proceedings of the 1997 IEEE signal processing society workshop*, pp. 276–285. IEEE, 1997.

[Pre81] Daryl Pregibon. "Logistic regression diagnostics." *The annals of statistics*, **9**(4):705–724, 1981.

[PWB19] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. "Hyperparameters and tuning strategies for random forest." *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, **9**(3):e1301, 2019.

[RP06] Romesh Ranawana and Vasile Palade. "Optimized precision-a new measure for classifier performance evaluation." In *2006 IEEE international conference on evolutionary computation*, pp. 2254–2261. IEEE, 2006.

[Tan20] Suryakanthi Tangirala. "Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm." *International Journal of Advanced Computer Science and Applications*, **11**(2):612–619, 2020.

[Tib96] Robert Tibshirani. "Regression shrinkage and selection via the lasso." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **58**(1):267–288, 1996.

[VGS96] Vladimir Vapnik, Steven Golowich, and Alex Smola. "Support vector method for function approximation, regression estimation and signal processing." *Advances in neural information processing systems*, **9**, 1996.

[Wah90] Grace Wahba. *Spline models for observational data.* SIAM, 1990.

[WF18] Yu-chen Wu and Jun-wen Feng. "Development and application of artificial neural network." *Wireless Personal Communications*, **102**:1645–1656, 2018.

[Wil00] Stewart W Wilson. "Mining oblique data with XCS." In *International Workshop on Learning Classifier Systems*, pp. 158–174. Springer, 2000.

[WP09] Richard G Wilkinson and Kate E Pickett. "Income inequality and social dysfunction." *Annual review of sociology*, **35**:493–511, 2009.

[ZHS09] Jinming Zou, Yi Han, and Sung-Sau So. "Overview of artificial neural networks." *Artificial neural networks: methods and applications*, pp. 14–22, 2009.