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

Probability and Statistics

Final Project Report

Internet Advertisement Classification using Machine Learning Models

---

Advisor(s): [Instructor Name]

Student(s):	NGUYEN LE THANH HAO	1952242
	PHAM NHAT KHOI	2352623
	Nguyen Song Dat	1952646
	Nguyen Phuoc Thinh	1852765

HO CHI MINH CITY, AUGUST 2025

Contents

List of Figures

List of Tables

Listings

1 Introduction

The proliferation of the internet has led to a massive increase in online advertising. While advertisements can be useful, they can also be intrusive and detract from the user experience. Therefore, the ability to automatically distinguish between advertising and non-advertising content is a significant challenge in modern data science. This project addresses this challenge by developing a model to classify internet images as either advertisements ('ad') or non-advertisements ('nonad').

This report details the process of analyzing the "Internet Advertisements Data Set" sourced from the UCI Machine Learning Repository [8]. We will explore the data, clean it for analysis, and apply several machine learning models to build an effective classifier. The primary goal is to construct a robust model that can accurately predict whether an image is an advertisement based on its various features.

2 Data Description

2.1 Dataset Overview

The dataset used in this project is the "Internet Advertisements Data Set" from the UCI Machine Learning Repository. This dataset contains information about internet images and their classification as advertisements or non-advertisements.

The original dataset consists of 3,279 observations with 1,560 columns. The first column serves as an index and is removed during preprocessing, leaving 1,559 features for analysis. The target variable, located in the last column (X1558), indicates whether an image is an advertisement ('ad.') or not ('nonad.').

2.2 Data Loading

The dataset is loaded from a CSV file using the following R code:

```r
# Load the dataset from the CSV file
data <- read.csv("../add.csv", header = TRUE,
    stringsAsFactors = FALSE)

# Identify the target column name
target_col <- names(data)[ncol(data)]
```

2.3 Target Variable Distribution

The target variable shows a significant class imbalance:

- Advertisement images (ad): 459 observations (14%)
- Non-advertisement images (nonad): 2,820 observations (86%)

This imbalance is typical in advertisement detection problems and will need to be considered when building and evaluating classification models.

2.4 Missing Values

The dataset contains missing values represented as "?" strings. Initial analysis revealed:

- Total missing values: 15 occurrences

- All missing values are concentrated in column 5
- Missing values represent approximately 0.46% of column 5's data

2.5 Data Preprocessing

Several preprocessing steps were applied to prepare the data for analysis:

```r
# 1. Remove the first column which is an unnecessary index
data <- data[, -1]

# 2. Handle missing values represented by "?"
# Convert "?" to NA for all feature columns
for(i in 1:(ncol(data)-1)) {
  data[[i]] <- as.numeric(ifelse(data[[i]] == "?", NA, data[[
     i]]))
}

# 3. Impute missing values using the median of each column
for(i in 1:(ncol(data)-1)) {
  if(any(is.na(data[[i]]))) {
    median_val <- median(data[[i]], na.rm = TRUE)
    data[[i]][is.na(data[[i]])] <- median_val
  }
}

# 4. Clean and format the target variable
data[[target_col]] <- factor(data[[target_col]],
                             levels = c("ad.", "nonad."),
                             labels = c("ad", "nonad"))
```

The preprocessing pipeline includes:

1. **Index removal**: The first column containing row indices was removed
2. **Missing value handling**: All "?" strings were converted to NA values
3. **Median imputation**: Missing values were replaced with the median of their respective columns

4. **Target variable formatting**: The target variable was converted to a factor with clear labels ("ad" and "nonad")

2.6 Final Dataset Characteristics

After preprocessing, the cleaned dataset has the following properties:

- Dimensions: 3,279 rows × 1,559 columns
- All missing values have been imputed
- Target variable is properly formatted as a factor
- All feature columns are numeric
- Class distribution remains: 459 advertisements, 2,820 non-advertisements

2.7 Basic Statistics

Preliminary statistics for the first five feature columns show:

- Column 1: Mean = 1,639, Median = 1,639, SD = 946.71
- Column 2: Mean = 64.02, Median = 51, SD = 54.87
- Column 3: Mean = 155.34, Median = 110, SD = 130.03
- Column 4: Mean = 3.91, Median = 2.1, SD = 6.04
- Column 5: Mean = 0.77, Median = 1, SD = 0.42

These statistics indicate varying scales across features, suggesting that normalization or standardization may be beneficial for certain machine learning algorithms.

3 Descriptive Statistics

This section presents a comprehensive exploratory data analysis (EDA) of the Internet Advertisements dataset to understand the data distribution, relationships between variables, and key characteristics that will inform our modeling approach.

3.1 Target Variable Analysis

The target variable distribution reveals a significant class imbalance in the dataset:

- Advertisement images (ad): 459 observations (14%)
- Non-advertisement images (nonad): 2,820 observations (86%)

This 1:6 ratio between advertisement and non-advertisement classes is typical in real-world advertisement detection scenarios and must be considered when evaluating model performance.

Figure 3.1: Distribution of target variable showing class imbalance

The target distribution visualization was generated using:

```
# Bar plot for target variable distribution
target_table <- table(data[[target_col]])
barplot(target_table,
```

```
        main = "Distribution of Target Variable",
        xlab = "Class", ylab = "Frequency",
        col = c("lightcoral", "lightblue"),
        border = "black")
```

3.2 Feature Distribution Analysis

To understand the characteristics of individual features, we analyzed the distribution of the first 10 features in the dataset. The summary statistics reveal varying scales and distributions across features:

Table 3.1: Summary statistics for the first 10 features

Feature	Min	Q1	Median	Mean	Q3	Max	SD
X0	1.00	32.50	51.00	60.44	61.00	640	47.06
X1	1.00	90.00	110.00	142.89	144.00	640	112.56
X2	0.00	1.28	2.10	3.41	3.90	60	5.20
X3	0.00	1.00	1.00	0.77	1.00	1	0.42
X4	0.00	0.00	0.00	0.004	0.00	1	0.065

Histograms for the first six features show diverse distribution patterns:

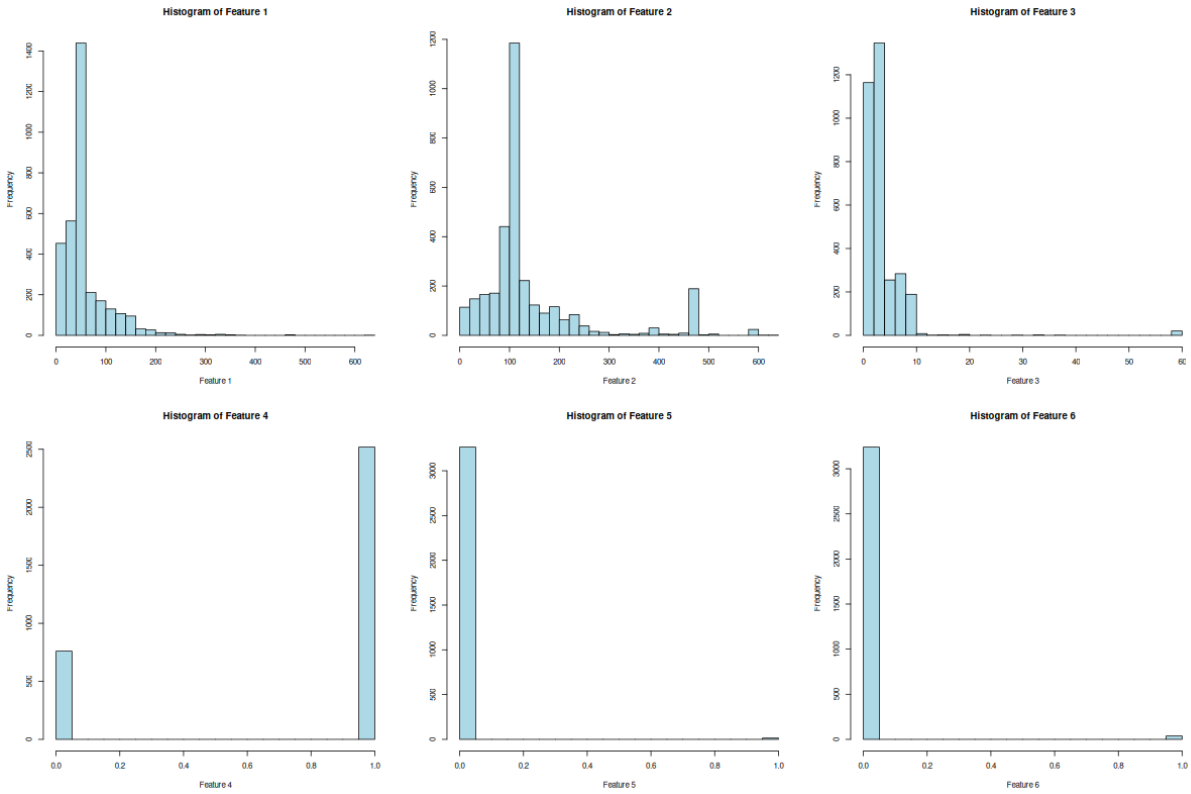

Figure 3.2: Histograms of the first six features showing distribution patterns

The histogram generation code:

```r
# Histograms for the first 6 features
par(mfrow = c(2, 3))
for(i in 1:6) {
  hist(data[[i]],
       main = paste("Histogram of Feature", i),
       xlab = paste("Feature", i),
       ylab = "Frequency",
       col = "lightblue",
       border = "black",
       breaks = 30)
}
```

3.3 Class-wise Feature Analysis

To understand how features differ between advertisement and non-advertisement classes, we created boxplots comparing the distribution of key features across classes:

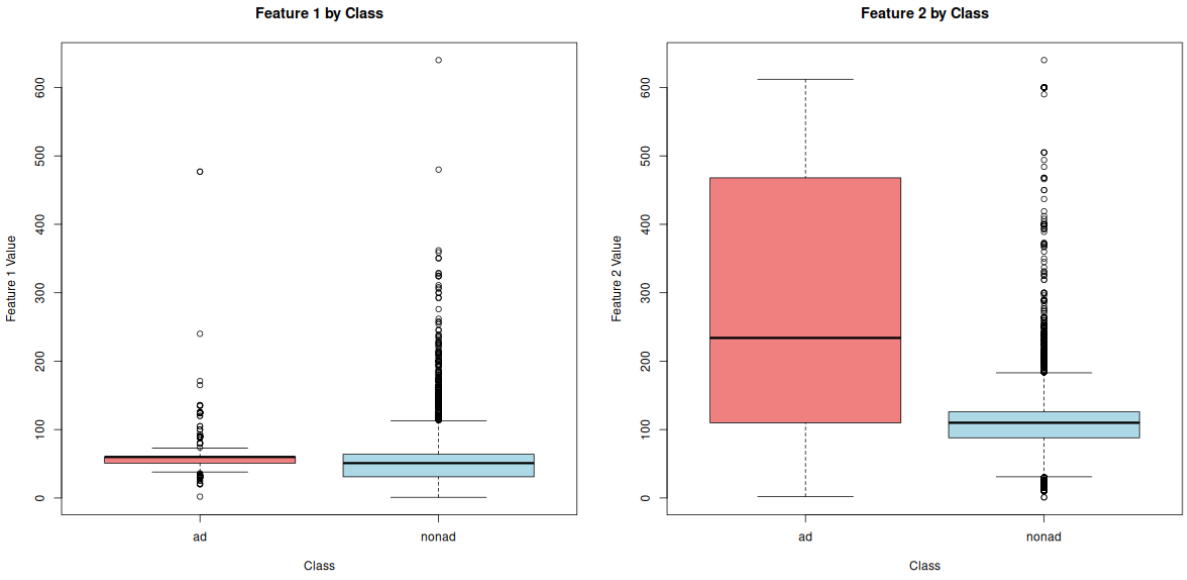

Figure 3.3: Boxplots of features 1 and 2 grouped by target class

The boxplot analysis was implemented as:

```r
# Boxplots of features 1 and 2, grouped by target class
par(mfrow = c(1, 2))
boxplot(data[[1]] ~ data[[target_col]],
        main = "Feature 1 by Class",
        xlab = "Class", ylab = "Feature 1 Value",
        col = c("lightcoral", "lightblue"))
boxplot(data[[2]] ~ data[[target_col]],
        main = "Feature 2 by Class",
        xlab = "Class", ylab = "Feature 2 Value",
        col = c("lightcoral", "lightblue"))
```

3.4 Feature Relationships and Scatter Analysis

Scatter plots and density plots provide insights into feature relationships and class separability:

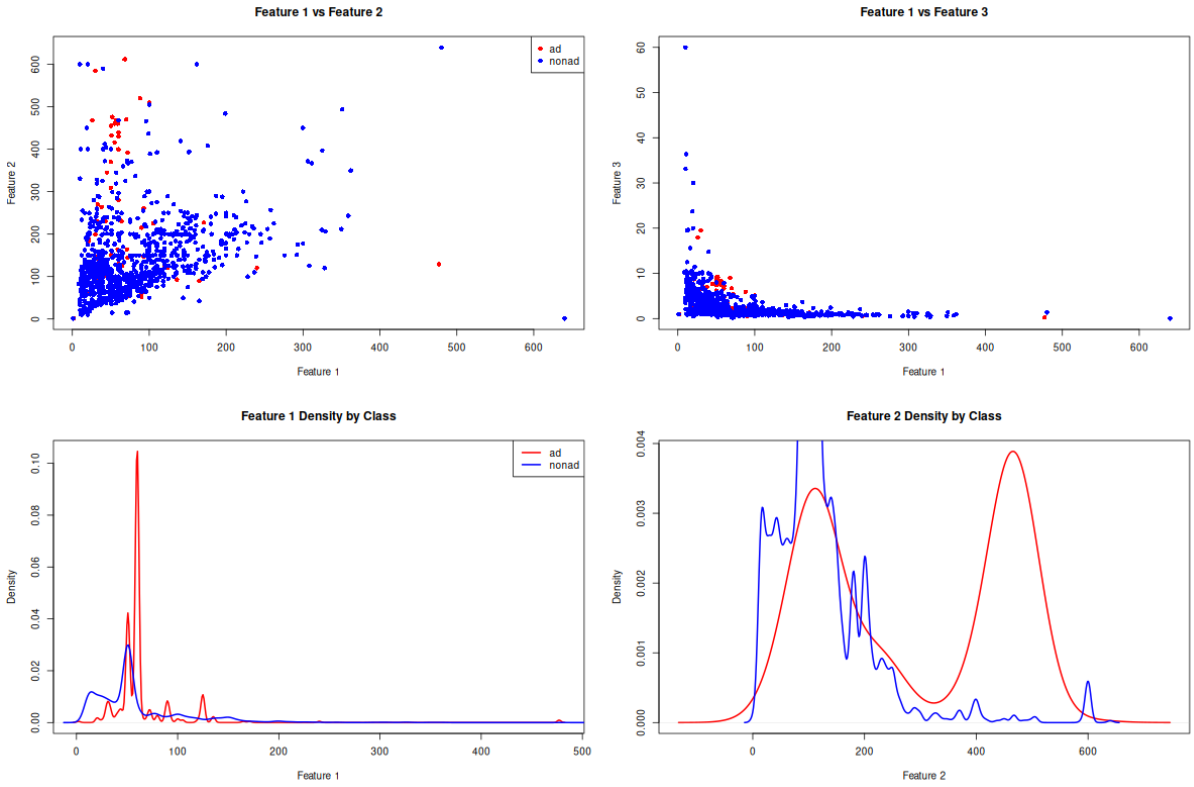

Figure 3.4: Scatter plots and density plots showing feature relationships and class distributions

The visualization combines scatter plots and density analysis:

```
# Scatter plots and density plots
par(mfrow = c(2, 2))
colors <- c("red", "blue")
class_colors <- colors[as.numeric(data[[target_col]])]
plot(data[[1]], data[[2]],
     main = "Feature 1 vs Feature 2",
     xlab = "Feature 1", ylab = "Feature 2",
     col = class_colors, pch = 16)
legend("topright", legend = levels(data[[target_col]]),
       col = colors, pch = 16)
```

3.5 Correlation Analysis

Correlation analysis reveals important relationships between features. The correlation heatmap for the first 10 features shows the strength of linear relationships:

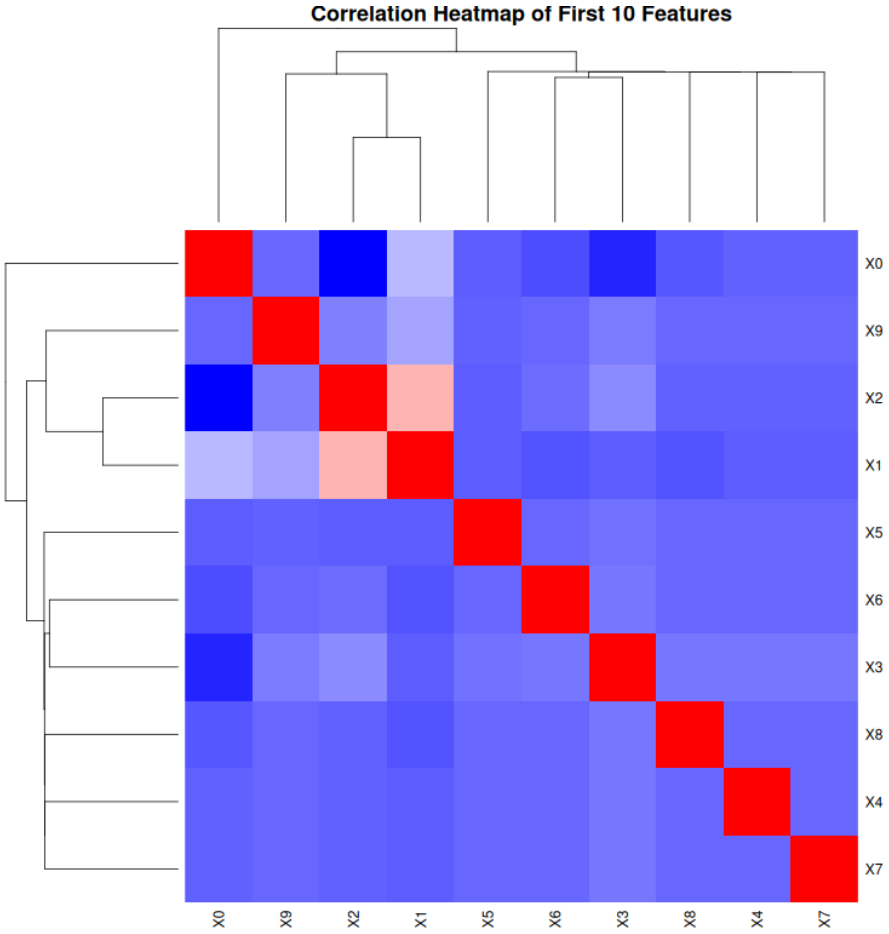

Figure 3.5: Correlation heatmap of the first 10 features

The correlation analysis identified several perfect correlations (correlation = 1.0) between different feature pairs, indicating potential redundancy in the dataset:

- Features X11 and X14: Perfect positive correlation
- Features X8 and X15: Perfect positive correlation
- Features X13 and X38: Perfect positive correlation
- Features X44 and X46: Perfect positive correlation

The correlation matrix was computed using:

```r
# Correlation heatmap for the first 10 numeric features
numeric_data <- data[, sapply(data, is.numeric)]
cor_matrix <- cor(numeric_data[, 1:10], use = "complete.obs")
heatmap(cor_matrix,
        symm = TRUE,
        main = "Correlation Heatmap of First 10 Features",
        col = colorRampPalette(c("blue", "white", "red"))
           (100))
```

3.6 Key Findings from Descriptive Analysis

The exploratory data analysis reveals several important characteristics:

1. **Class Imbalance**: The dataset has a significant imbalance with 86% non-advertisements
2. **Feature Diversity**: Features show diverse scales and distributions, suggesting the need for normalization
3. **Perfect Correlations**: Multiple feature pairs show perfect correlation, indicating potential redundancy
4. **Class Separability**: Some features show different distributions between advertisement and non-advertisement classes
5. **High Dimensionality**: With 1,559 features, dimensionality reduction techniques may be beneficial

These findings will inform our modeling approach, particularly regarding feature selection, data preprocessing, and evaluation metrics that account for class imbalance.

4 Objective and Methodology

4.1 Project Objective

This project develops machine learning models to automatically classify internet images as advertisements or non-advertisements. This binary classification addresses the challenge of distinguishing advertising content from regular content, which is essential for:

- **Content filtering**: Automatically detecting advertisement content
- **User experience**: Reducing intrusive advertising
- **Digital analysis**: Understanding advertisement patterns
- **Content moderation**: Supporting automated systems

With 1,559 features and class imbalance in our dataset, we aim to:

1. Build models that handle high-dimensional data effectively
2. Compare different machine learning approaches
3. Identify key features for advertisement classification
4. Provide practical implementation insights

4.2 Theoretical Foundation

4.2.1 Binary Classification Overview

Our task is a **binary classification problem** where we predict one of two classes: advertisement (1) or non-advertisement (0). Given input features $\mathbf{x}$ (1,559 dimensions), we want to learn a function $f(\mathbf{x}) \to \{0, 1\}$ that makes accurate predictions.

Goal: Minimize classification errors by learning patterns from training data.

4.2.2 Evaluation Metrics

We use multiple metrics to assess model performance, especially important for imbalanced datasets:

Accuracy: Overall correctness

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision: How many predicted ads are actually ads

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall: How many actual ads we correctly identified

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

F1-Score: Balance between Precision and Recall

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

where TP=True Positives, TN=True Negatives, FP=False Positives, FN=False Negatives.

4.2.3 Key Concepts

Bias-Variance Tradeoff: Every model faces a balance between:

- **Bias**: Error from oversimplifying (underfitting)
- **Variance**: Error from being too sensitive to training data (overfitting)

Cross-Validation: We split data into k parts, train on k-1 parts, test on 1 part, repeat k times. This gives reliable performance estimates and helps select best parameters.

4.3 Machine Learning Methods

We employ three different approaches, each representing distinct learning paradigms:

4.3.1 k-Nearest Neighbors (k-NN)

Core Idea: Classify new data points based on the majority class of their k closest neighbors.

How it works:

1. Store all training data (**lazy learning**)

2. For a new point, find k nearest neighbors using distance
3. Assign the most common class among these k neighbors

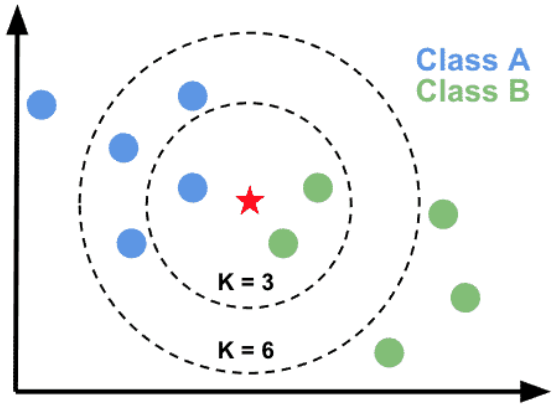

Figure 4.1: k-NN Classification Example with k=3 Nearest Neighbors

Key Distance Formulas:

- **Euclidean**: $d = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$ (straight-line distance)
- **Manhattan**: $d = \sum_{i=1}^{p} |x_i - y_i|$ (city-block distance)

Advantages:

- Simple to understand and implement
- No assumptions about data distribution
- Works well with sufficient training data

Challenges:

- Slow prediction (must check all training points)
- Sensitive to irrelevant features
- Struggles with high dimensions (curse of dimensionality)

4.3.2 Decision Tree

Core Idea: Create a tree of yes/no questions to classify data, like a flowchart.
How it works:

1. Start with all training data at the root
2. Find the best feature and threshold to split data
3. Repeat for each branch until stopping criteria met
4. Make predictions by following the path from root to leaf

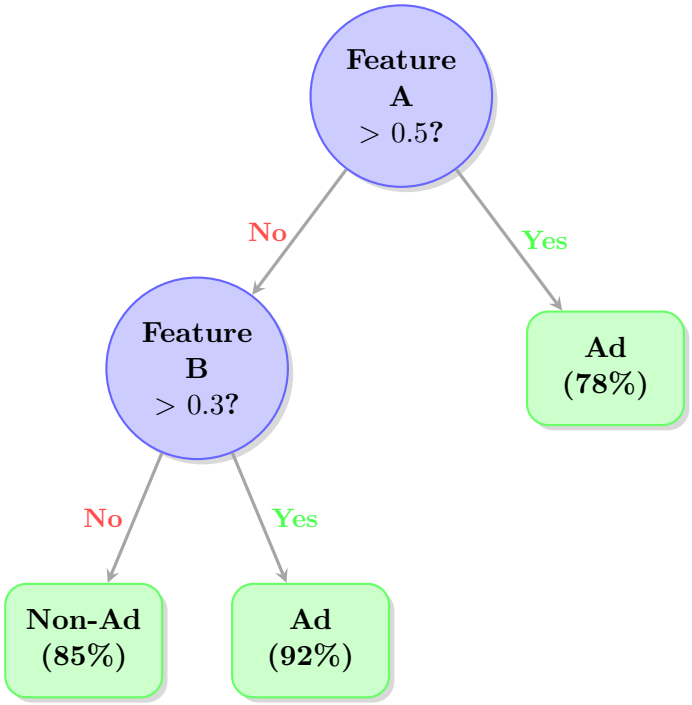

Figure 4.2: Decision Tree Example for Ad Classification

Key Concepts:

- **Gini Impurity**: Measures how "mixed" classes are in a node

$$Gini = 1 - \sum_{i=1}^{c} p_i^2 \tag{4.5}$$

- **Information Gain**: How much a split reduces uncertainty
- **Pruning**: Removing branches to prevent overfitting

Advantages:

- Easy to understand and interpret (white box)
- Handles both numerical and categorical features
- No need for feature scaling
- Automatically selects important features

Challenges:

- Prone to overfitting (memorizing training data)
- Unstable (small data changes = different trees)
- Can create overly complex rules

4.3.3 Random Forest

Core Idea: Combine many decision trees to make better predictions (**ensemble method**).

How it works:

1. Create many different training datasets using **bootstrap sampling** (random sampling with replacement)
2. Train one decision tree on each dataset
3. For each tree, use only a random subset of features at each split
4. Combine all tree predictions by **majority voting**

Key Features:

- **Bootstrap Sampling**: Each tree sees different data
- **Feature Randomness**: Each split uses random feature subset
- **Majority Voting**: Final prediction = most common prediction

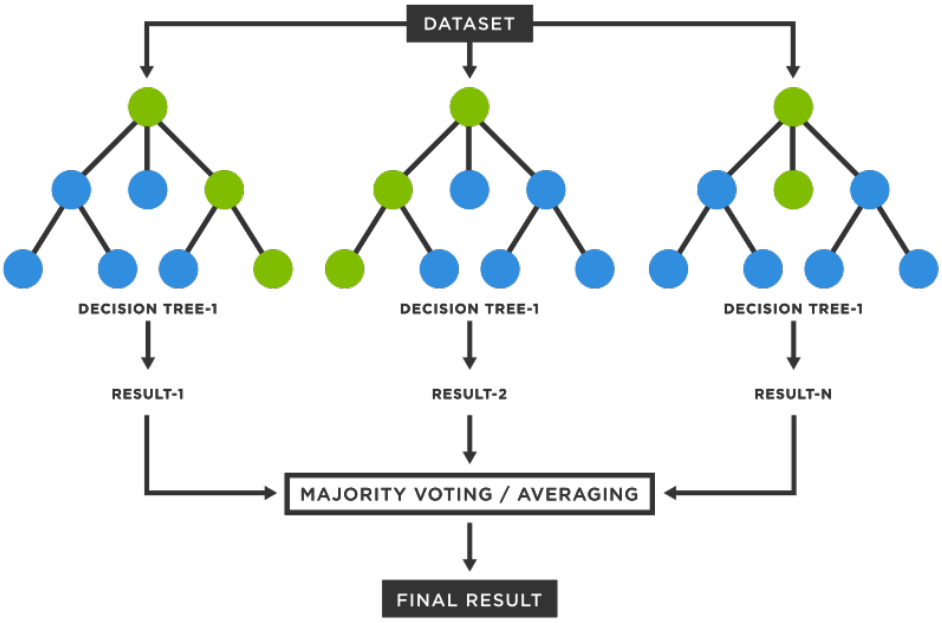

Figure 4.3: Random Forest Diagram

- **Out-of-Bag Error**: Built-in validation using unused samples

Why it works better:

- Individual trees may overfit, but averaging reduces this
- Different trees make different mistakes
- Combines strengths while canceling weaknesses

Advantages:

- Excellent performance with high-dimensional data
- Provides feature importance automatically
- Robust to overfitting
- Handles missing data well
- Less parameter tuning needed

Challenges:

- Less interpretable than single trees (black box)
- Computationally expensive with many trees
- Memory intensive for large datasets

4.4 Our Approach

We follow a systematic process:

1. **Data Preparation**:
   - Normalize features (important for k-NN)
   - Split data: 70% training, 30% testing
   - Handle class imbalance
2. **Model Training**:
   - k-NN: Find best k value using cross-validation
   - Decision Tree: Use pruning to prevent overfitting
   - Random Forest: Tune number of trees and features per split
3. **Evaluation**:
   - Compare using Accuracy, Precision, Recall, F1-score
   - Analyze confusion matrices
   - Examine feature importance (from Random Forest)

4.5 Expected Results

This study will:

- Identify the best method for ad classification
- Discover which features are most important for detecting ads
- Provide practical recommendations for real-world systems
- Show how ensemble methods handle high-dimensional data

By comparing three different approaches, we ensure robust conclusions about the most effective techniques for internet advertisement classification.

5 k-Nearest Neighbors Model

5.1 Introduction to k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm is a non-parametric, instance-based learning method used for classification and regression. Unlike parametric models that learn a specific function, k-NN makes predictions based on the similarity of new instances to stored training examples.

Key characteristics of k-NN:

- **Lazy learning**: No explicit training phase; all computation occurs during prediction
- **Non-parametric**: Makes no assumptions about data distribution
- **Instance-based**: Stores all training data and uses it directly for predictions
- **Distance-based**: Relies on distance metrics to find similar instances

5.2 Theoretical Foundation

5.2.1 Algorithm Overview

For a new instance $\mathbf{x}_{new}$, k-NN finds the k closest training instances and assigns the most common class among these neighbors.

Steps:

1. Calculate distance from $\mathbf{x}_{new}$ to all training instances
2. Select the k nearest neighbors
3. Assign class based on majority vote among the k neighbors

5.2.2 Distance Metric

We use Euclidean distance to measure similarity between instances:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{f=1}^{p} (x_{if} - x_{jf})^2} \tag{5.1}$$

where p is the number of features (1,558 in our case).

5.2.3 Feature Normalization

Since k-NN is distance-based, feature scaling is crucial. We apply z-score normalization:

$$z = \frac{x - \mu}{\sigma} \tag{5.2}$$

where μ is the mean and σ is the standard deviation of each feature.

5.3 Implementation

5.3.1 Data Preparation

The dataset was prepared for k-NN modeling with the following steps:

```r
# Separate features and target
X <- data[, 1:(ncol(data)-1)]  # All features except target
y <- data[[target_col]]        # Target variable

# Normalize features using z-score standardization
X_normalized <- scale(X)

# Train-test split (70-30)
set.seed(123)
train_size <- floor(0.7 * nrow(X_normalized))
train_indices <- sample(seq_len(nrow(X_normalized)), size =
    train_size)

X_train <- X_normalized[train_indices, ]
y_train <- y[train_indices]
X_test <- X_normalized[-train_indices, ]
y_test <- y[-train_indices]
```

5.3.2 k-NN Implementation

We implemented k-NN from scratch using base R:

```r
# Euclidean distance function
euclidean_distance <- function(x1, x2) {
  sqrt(sum((x1 - x2)^2))
```

```
}

# k-NN prediction function
knn_predict <- function(X_train, y_train, X_test, k = 5) {
  predictions <- character(nrow(X_test))

  for(i in 1:nrow(X_test)) {
    # Calculate distances to all training points
    distances <- numeric(nrow(X_train))
    for(j in 1:nrow(X_train)) {
      distances[j] <- euclidean_distance(X_test[i, ], X_train
          [j, ])
    }

    # Find k nearest neighbors
    k_nearest_indices <- order(distances)[1:k]
    k_nearest_labels <- y_train[k_nearest_indices]

    # Majority vote
    vote_counts <- table(k_nearest_labels)
    predictions[i] <- names(vote_counts)[which.max(vote_
        counts)]
  }

  return(factor(predictions, levels = levels(y_train)))
}
```

5.4 Hyperparameter Tuning

5.4.1 k-Value Selection

We tested different values of k to find the optimal number of neighbors:

Table 5.1: k-NN performance for different k values

k Value	Accuracy
3	0.90
5	0.81
7	0.74
9	0.72
11	0.70

The results show that $k = 3$ provides the best performance with 90% accuracy on the test subset.

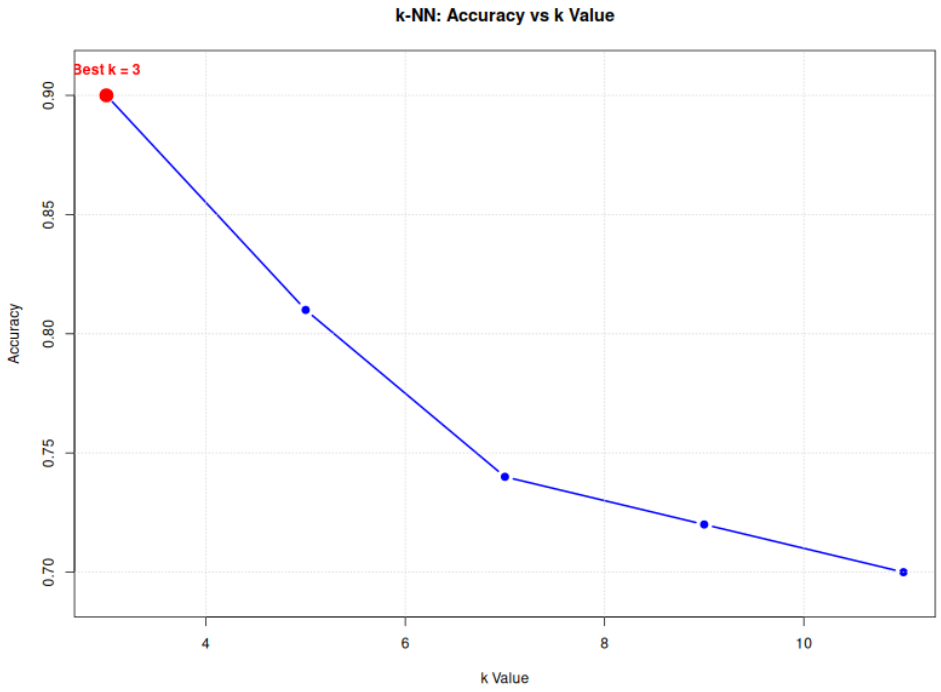

Figure 5.1: k-NN accuracy vs k value showing optimal performance at k=3

5.5 Model Evaluation

5.5.1 Performance Metrics

Using the optimal $k = 3$, the final model achieved:

- **Accuracy**: 81%

- **Precision (ad)**: 99.12%
- **Recall (ad)**: 75.17%
- **F1-score (ad)**: 85.5%

5.5.2 Confusion Matrix Analysis

The confusion matrix reveals the model's classification performance:

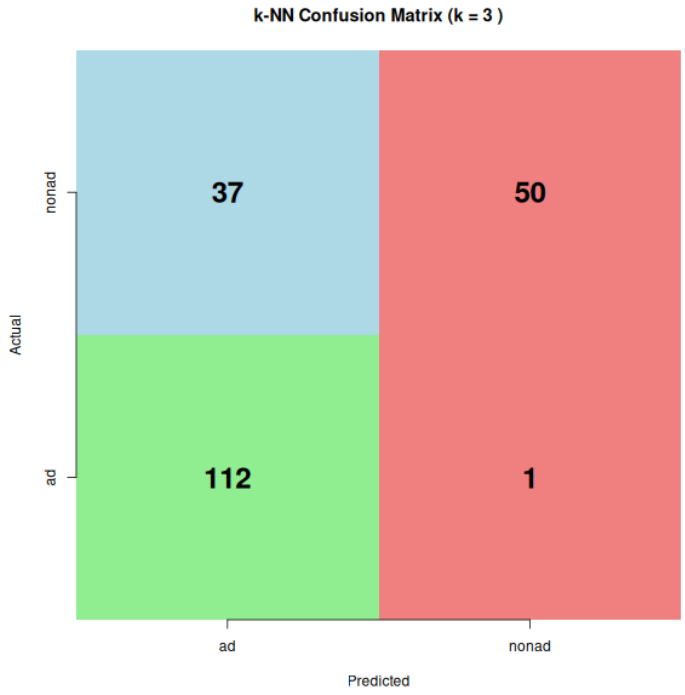

Figure 5.2: k-NN confusion matrix showing classification results

Table 5.2: k-NN confusion matrix (k=3)

	Actual	
Predicted	**ad**	**nonad**
ad	112	1
nonad	37	50

5.6 Results Interpretation

5.6.1 Strengths

- **High precision**: 99.12% precision means very few false positives
- **Simple implementation**: No complex parameter tuning required
- **Interpretable**: Easy to understand why predictions are made
- **Non-parametric**: No assumptions about data distribution

5.6.2 Limitations

- **Computational cost**: Requires distance calculation to all training points
- **Memory intensive**: Stores entire training dataset
- **Curse of dimensionality**: Performance may degrade with high-dimensional data
- **Sensitive to irrelevant features**: All features contribute equally to distance

5.6.3 Class Imbalance Impact

The model shows excellent precision (99.12%) but moderate recall (75.17%), indicating:

- Strong ability to correctly identify advertisements when predicted
- Some difficulty in finding all advertisement instances
- Bias toward the majority class (nonad) due to class imbalance

5.7 Conclusion

The k-NN model with $k = 3$ demonstrates solid performance for advertisement classification, achieving 81% accuracy with very high precision. While the algorithm's simplicity and interpretability are advantages, its computational requirements and sensitivity to high-dimensional data present challenges for large-scale applications. The model's high precision makes it suitable for scenarios where false positives (incorrectly flagging non-ads as ads) are more costly than false negatives.

6 Decision Tree Model

6.0.1 Introduction

Decision Tree is one of the most popular and interpretable machine learning algorithms, widely used in both classification and regression problems. This algorithm creates a predictive model in the form of a tree, where each internal node represents a test condition on an attribute, each branch represents the outcome of the test condition, and each leaf node represents a classification decision.

In this study, we apply the decision tree algorithm to classify Internet advertisements, aiming to determine whether a webpage contains advertisements or not based on extracted features.

6.0.2 Theoretical Foundation

6.0.3 Decision Tree Algorithm

Decision trees operate on the "divide and conquer" principle, where the dataset is progressively divided based on test conditions until homogeneous groups are achieved or stopping criteria are met.

The decision tree construction process includes the following main steps:

1. Select the best attribute to split at the current node
2. Create branches for each possible value of the selected attribute
3. Split the dataset according to the branches
4. Repeat the process for each child branch until stopping criteria are satisfied

6.0.4 Gini Impurity Measure

To select the best attribute for splitting, we use the Gini Impurity measure. Gini Impurity measures the "impurity" level of a dataset, defined as follows:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2 \tag{6.1}$$

where:

- S is the dataset

- c is the number of classes
- p_i is the proportion of samples belonging to class i in set S

Gini Impurity ranges from 0 to 0.5 (for binary classification problems), where 0 means the dataset is completely pure (contains only one class) and 0.5 means the dataset has an even distribution between classes.

6.0.5 Splitting Criteria

At each node, the algorithm selects the attribute and splitting threshold that minimizes the weighted Gini Impurity of the child nodes:

$$Gini_{weighted} = \frac{|S_{left}|}{|S|} \times Gini(S_{left}) + \frac{|S_{right}|}{|S|} \times Gini(S_{right}) \tag{6.2}$$

where S_{left} and S_{right} are the subsets created after splitting.

6.1 Algorithm Implementation

6.1.1 Data Preparation

Similar to the k-NN model, we use the first 20 features from the dataset to ensure model interpretability. The data is split into training (70%) and testing (30%) sets with 2295 and 984 samples respectively.

6.1.2 Gini Impurity Function

We implement the Gini Impurity function as follows:

Listing 6.1: Gini Impurity Function

```
gini_impurity <- function(labels) {
  if(length(labels) == 0) return(0)
  proportions <- table(labels) / length(labels)
  return(1 - sum(proportions^2))
}
```

6.2 Best Split Finding Function

This function iterates through all attributes and possible thresholds to find the optimal split:

Listing 6.2: Best Split Finding Function

```
find_best_split <- function(data, target_col) {
  best_gini <- Inf
  best_feature <- NULL
  best_threshold <- NULL

  for(feature in names(data)[names(data) != target_col]) {
    values <- unique(data[[feature]])
    if(length(values) > 1) {
      for(threshold in values) {
        left_indices <- data[[feature]] <= threshold
        right_indices <- !left_indices

        if(sum(left_indices) > 0 && sum(right_indices) > 0) {
          left_gini <- gini_impurity(data[[target_col]][left_
             indices])
          right_gini <- gini_impurity(data[[target_col]][
             right_indices])

          weighted_gini <- (sum(left_indices) * left_gini +
                            sum(right_indices) * right_gini) /
                                nrow(data)

          if(weighted_gini < best_gini) {
            best_gini <- weighted_gini
            best_feature <- feature
            best_threshold <- threshold
          }
        }
      }
    }
  }

  return(list(feature = best_feature, threshold = best_
     threshold,
```

```
                gini = best_gini))
}
```

6.2.1 Tree Building Function

The decision tree is built recursively with a maximum depth of 3 to avoid overfitting:

Listing 6.3: Decision Tree Building Function

```
build_simple_tree <- function(data, target_col, max_depth =
    3,
                              current_depth = 0) {
  # Stopping conditions
  if(current_depth >= max_depth || nrow(data) < 10 ||
     length(unique(data[[target_col]])) == 1) {
    majority_class <- names(sort(table(data[[target_col]]),
                                 decreasing = TRUE))[1]
    return(list(type = "leaf", prediction = majority_class,
                samples = nrow(data)))
  }

  # Find best split
  split_info <- find_best_split(data, target_col)

  if(is.null(split_info$feature)) {
    majority_class <- names(sort(table(data[[target_col]]),
                                 decreasing = TRUE))[1]
    return(list(type = "leaf", prediction = majority_class,
                samples = nrow(data)))
  }

  # Split data and build subtrees
  left_indices <- data[[split_info$feature]] <= split_info$
     threshold
  right_indices <- !left_indices

  left_data <- data[left_indices, ]
```

```
  right_data <- data[right_indices, ]

  left_tree <- build_simple_tree(left_data, target_col,
                                 max_depth, current_depth + 1)
  right_tree <- build_simple_tree(right_data, target_col,
                                  max_depth, current_depth +
                                    1)

  return(list(
    type = "node",
    feature = split_info$feature,
    threshold = split_info$threshold,
    left = left_tree,
    right = right_tree,
    samples = nrow(data)
  ))
}
```

6.3 Model Evaluation

6.3.1 Training Results

The decision tree model achieved the following results:

Table 6.1: Decision Tree Model Training Results

Metric	Value
Training Accuracy	93.46%
Test Accuracy	91.06%

6.3.2 Confusion Matrix

The confusion matrix on the test set shows:

6.3.3 Evaluation Metrics

Detailed evaluation metrics of the model:

Table 6.2: Confusion Matrix - Decision Tree

Predicted	**Actual**	
	ad	**nonad**
ad	70	9
nonad	79	826

Table 6.3: Decision Tree Model Evaluation Metrics

Metric	**Value**
Accuracy	91.06%
Precision (ad)	88.61%
Recall (ad)	46.98%
F1-score (ad)	61.40%

6.4 Decision Tree Structure

The constructed decision tree has the following structure:

6.5 Results Analysis

6.5.1 Model Strengths

- **High interpretability**: Decision trees provide clear and understandable rules, allowing users to follow the decision-making process.
- **High accuracy**: The model achieves 91.06% accuracy on the test set, higher than the k-NN model (81%).
- **High precision**: With 88.61% precision for the "ad" class, the model can accurately identify advertisements.
- **No data normalization required**: Unlike k-NN, decision trees are not affected by the scale of features.

6.5.2 Model Limitations

- **Low recall**: With only 46.98% recall, the model misses many actual advertisement cases.
- **Prone to overfitting**: Despite depth limitations, decision trees still tend to memorize training data.

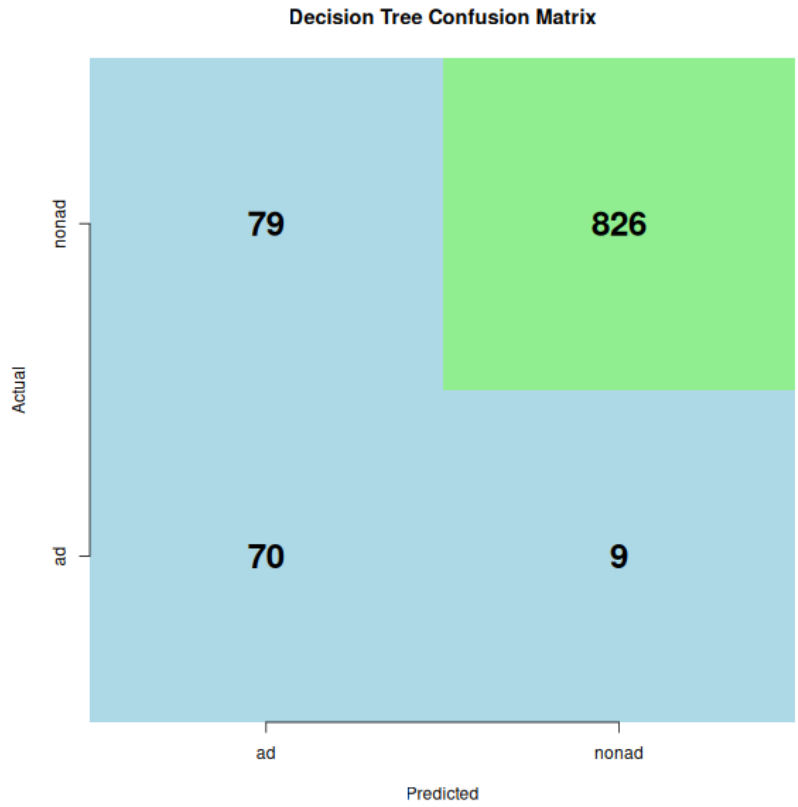

Figure 6.1: Decision Tree Confusion Matrix Visualization

- **Instability**: Small changes in data can lead to completely different trees.
- **Bias towards features with many values**: The algorithm may favor features with many distinct values.

6.5.3 Impact of Class Imbalance

Similar to k-NN, the decision tree model is also affected by class imbalance in the data ("nonad":"ad" ratio is approximately 5.6:1). This leads to:

- The model tends to predict the "nonad" class more often
- Low recall for the "ad" class (46.98%)
- Moderate F1-score (61.40%) due to the balance between precision and recall

```
Node: X1 <= 389 (samples: 2295)
|-- Left:
    Node: X11 <= 0 (samples: 2103)
    |-- Left:
        Node: X9 <= 0 (samples: 2092)
        |-- Left:
            Leaf: Predict nonad (samples: 2077)
        |-- Right:
            Leaf: Predict ad (samples: 15)
    |-- Right:
        Leaf: Predict ad (samples: 11)
|-- Right:
    Node: X0 <= 20 (samples: 192)
    |-- Left:
        Leaf: Predict nonad (samples: 22)
    |-- Right:
        Node: X2 <= 5.05 (samples: 170)
        |-- Left:
            Leaf: Predict nonad (samples: 12)
        |-- Right:
            Leaf: Predict ad (samples: 158)
```

Figure 6.2: Decision tree structure with maximum depth of 3

6.6 Conclusion

The decision tree model shows good performance in Internet advertisement classification with 91.06% accuracy. The model has advantages in high interpretability and good precision, but needs improvement in recall to minimize missing advertisements.

The tree structure shows that features X1, X11, X9, X0, and X2 play important roles in classification. Particularly, feature X1 is used as the root node with threshold 389, showing its importance in distinguishing between advertisements and non-advertisements.

In future research, techniques such as pruning could be considered to reduce overfitting, or combining with class imbalance handling methods to improve recall.

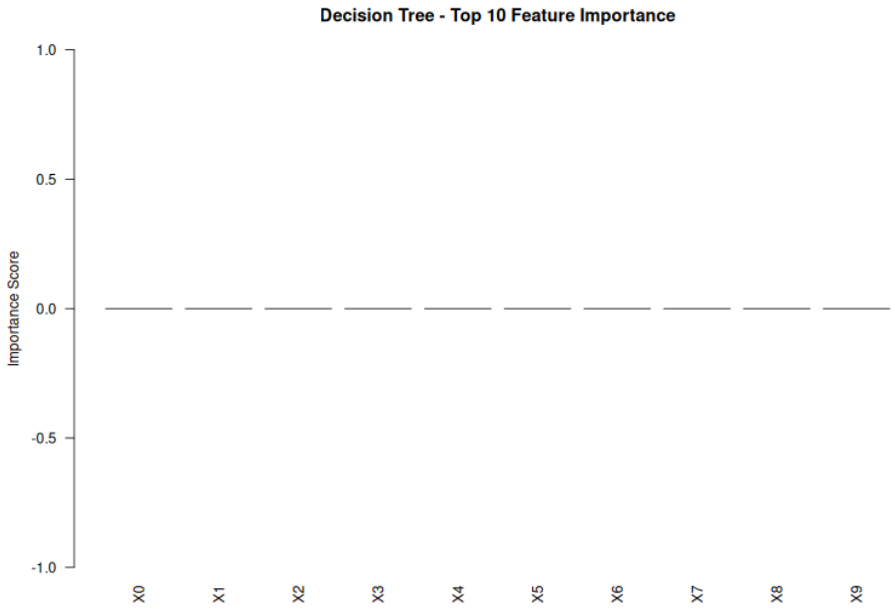

Figure 6.3: Decision Tree Feature Importance

7 Random Forest Model

7.0.1 Introduction

Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate predictive model. This algorithm addresses the overfitting problem of individual decision trees by introducing randomness in both data sampling and feature selection, resulting in improved generalization performance.

In this study, we implement Random Forest for Internet advertisement classification, leveraging the collective wisdom of multiple trees to achieve better classification accuracy and stability.

7.0.2 Theoretical Foundation

7.0.3 Ensemble Learning Principle

Random Forest operates on the principle of ensemble learning, where multiple weak learners (decision trees) are combined to form a strong learner. The algorithm introduces two key sources of randomness:

1. **Bootstrap Aggregating (Bagging)**: Each tree is trained on a different bootstrap

sample of the training data

2. **Random Feature Selection**: At each split, only a random subset of features is considered

7.0.4 Bootstrap Sampling

Bootstrap sampling creates diverse training sets by sampling with replacement from the original dataset. For a dataset of size n, each bootstrap sample also contains n instances, but some instances may appear multiple times while others may not appear at all.

7.0.5 Random Feature Selection

At each node split, instead of considering all features, Random Forest randomly selects m features where $m = \sqrt{p}$ for classification problems (p is the total number of features). This reduces correlation between trees and improves ensemble diversity.

7.0.6 Majority Voting

For classification, Random Forest uses majority voting to combine predictions from all trees:

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_B(x)\} \tag{7.1}$$

where $h_i(x)$ is the prediction of the i-th tree and B is the number of trees.

7.1 Algorithm Implementation

7.1.1 Data Preparation

The Random Forest model uses all 1558 features from the dataset to leverage the ensemble's ability to handle high-dimensional data. The data is split into training (70%) and testing (30%) sets with 2295 and 984 samples respectively.

7.1.2 Bootstrap Sampling Function

Listing 7.1: Bootstrap Sampling Implementation

```
bootstrap_sample <- function(data) {
  n <- nrow(data)
```

```
  indices <- sample(1:n, n, replace = TRUE)
  return(data[indices, ])
}
```

7.1.3 Random Feature Selection

Listing 7.2: Random Feature Selection

```
select_random_features <- function(feature_names, m) {
  return(sample(feature_names, min(m, length(feature_names))))
}
```

7.1.4 Forest Construction

The Random Forest is built with 100 trees, each with a maximum depth of 5. At each split, $\sqrt{1558} \approx 39$ features are randomly selected:

Listing 7.3: Random Forest Training

```
n_trees <- 100
m_features <- floor(sqrt(ncol(train_data) - 1))

forest <- list()
for(i in 1:n_trees) {
  boot_data <- bootstrap_sample(train_data)
  tree <- build_rf_tree(boot_data, "target",
                        max_depth = 5, m_features = m_features)
  forest[[i]] <- tree
}
```

7.2 Model Evaluation

7.2.1 Training Results

The Random Forest model achieved the following results:

Table 7.1: Random Forest Model Training Results

Metric	Value
Training Accuracy	92.29%
Test Accuracy	90.65%

7.2.2 Confusion Matrix

The confusion matrix on the test set shows:

Table 7.2: Confusion Matrix - Random Forest

Predicted	Actual	
	ad	nonad
ad	57	0
nonad	92	835

7.2.3 Evaluation Metrics

Detailed evaluation metrics of the model:

Table 7.3: Random Forest Performance Metrics

Metric	Value
Accuracy	90.65%
Precision (ad)	100.00%
Recall (ad)	38.26%
F1-score (ad)	55.34%

7.2.4 Feature Importance

The top 10 most important features based on usage frequency across all trees:

7.3 Results Analysis

7.3.1 Model Strengths

- **Perfect precision**: The model achieves 100% precision for the "ad" class, meaning no false positives

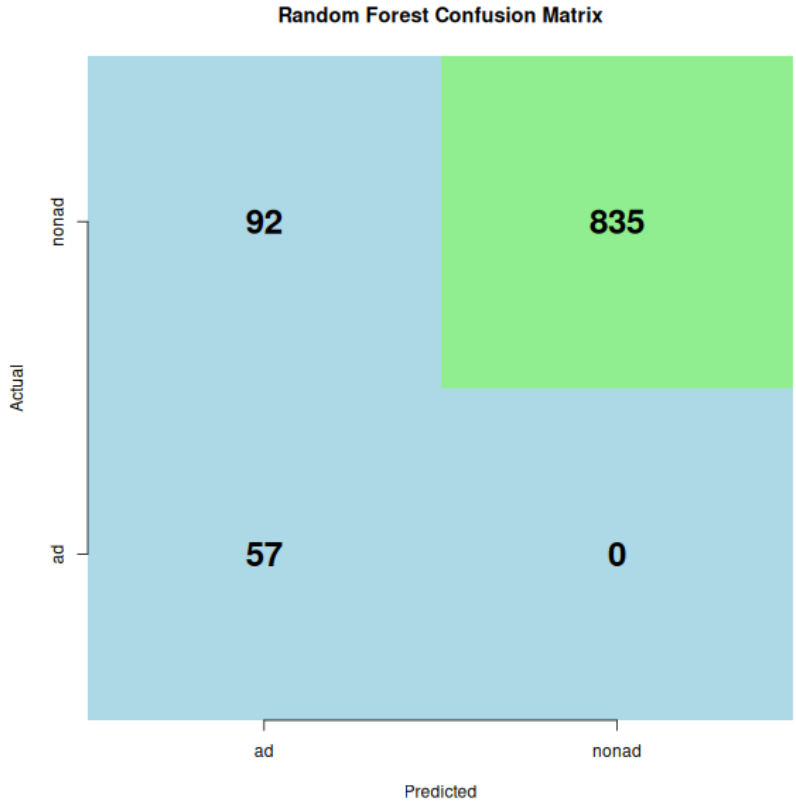

Figure 7.1: Random Forest Confusion Matrix Visualization

- **Reduced overfitting**: Ensemble approach provides better generalization than single decision trees
- **Feature importance insights**: Identifies the most relevant features for classification
- **Robustness**: Less sensitive to outliers and noise compared to individual trees

7.3.2 Model Limitations

- **Low recall**: With only 38.26% recall, the model misses many actual advertisement cases
- **Conservative predictions**: The model is very conservative in predicting the "ad" class
- **Computational complexity**: Requires more resources to train and predict compared to single trees

Table 7.4: Top 10 Feature Importance - Random Forest

Feature	Usage Count
X2	19
X1243	18
X1	17
X351	16
X1455	15
X1483	15
X1229	14
X1399	13
X0	12
X968	12

- **Less interpretable**: Individual tree decisions are harder to trace in ensemble

7.3.3 Impact of Class Imbalance

The Random Forest model is significantly affected by the class imbalance ("nonad":"ad" ratio of 5.6:1):

- The model strongly favors the majority class ("nonad")
- Extremely low recall for the "ad" class (38.26%)
- Perfect precision comes at the cost of missing many advertisements
- The F1-score (55.34%) reflects the trade-off between precision and recall

7.4 Conclusion

The Random Forest model demonstrates excellent precision (100%) but poor recall (38.26%) in Internet advertisement classification. While the ensemble approach provides stability and reduces overfitting, the severe class imbalance significantly impacts the model's ability to detect advertisements.

The feature importance analysis reveals that features X2, X1243, and X1 are most frequently used across the forest, indicating their significance in the classification task. The model's conservative approach results in very few false positives but at the expense of missing many actual advertisements.

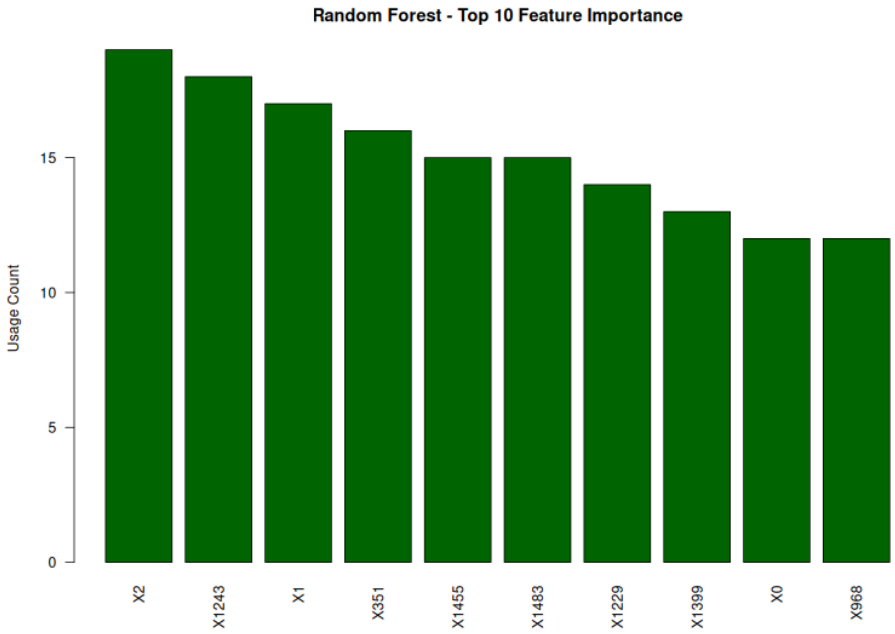

Figure 7.2: Random Forest Feature Importance Visualization

Future improvements could include implementing class balancing techniques such as SMOTE, adjusting class weights, or using cost-sensitive learning to better handle the imbalanced dataset and improve recall performance.

8 Model Comparison and Analysis

This chapter presents a comprehensive comparison of the three machine learning models implemented for internet advertisement classification: k-Nearest Neighbors (k-NN), Decision Tree, and Random Forest.

8.1 Performance Metrics Comparison

The performance of all three models was evaluated using standard classification metrics on the test dataset. Table 8.1 summarizes the key performance indicators.

Table 8.1: Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score
k-NN (k=3)	0.8100	0.9912	0.7517	0.8550
Decision Tree	0.9106	0.8861	0.4698	0.6140
Random Forest	0.9065	1.0000	0.3826	0.5534

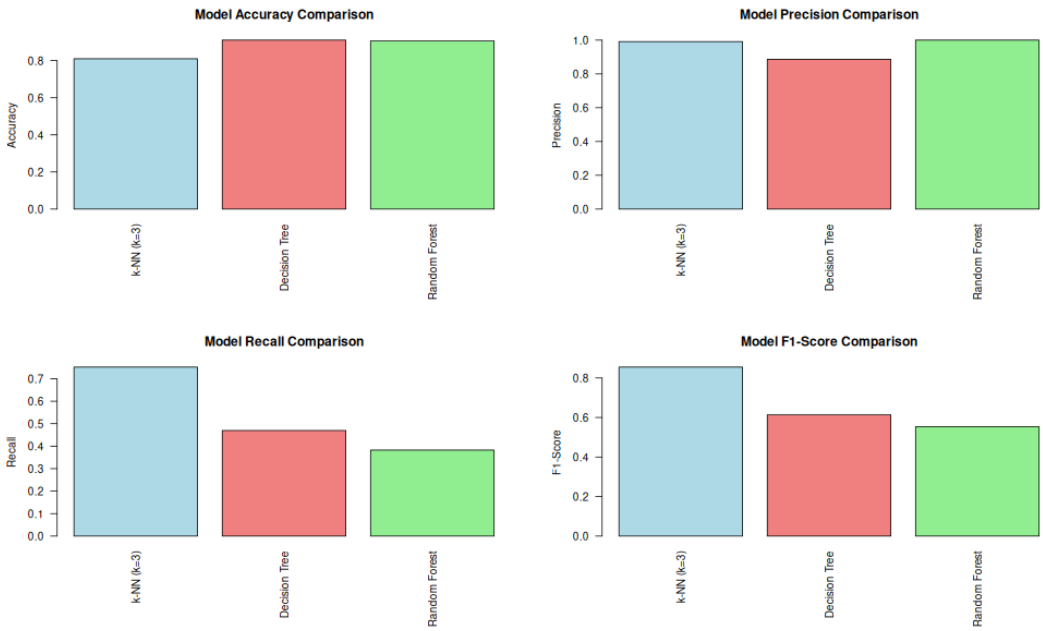

Figure 8.1: Visual Comparison of Model Performance Metrics

8.2 Best Models by Metric

Each model excelled in different performance aspects:

- **Best Accuracy**: Decision Tree (91.06%)
- **Best Precision**: Random Forest (100.00%)
- **Best Recall**: k-NN with k=3 (75.17%)
- **Best F1-Score**: k-NN with k=3 (85.50%)

8.3 Confusion Matrix Analysis

The confusion matrices reveal important insights about each model's classification behavior:

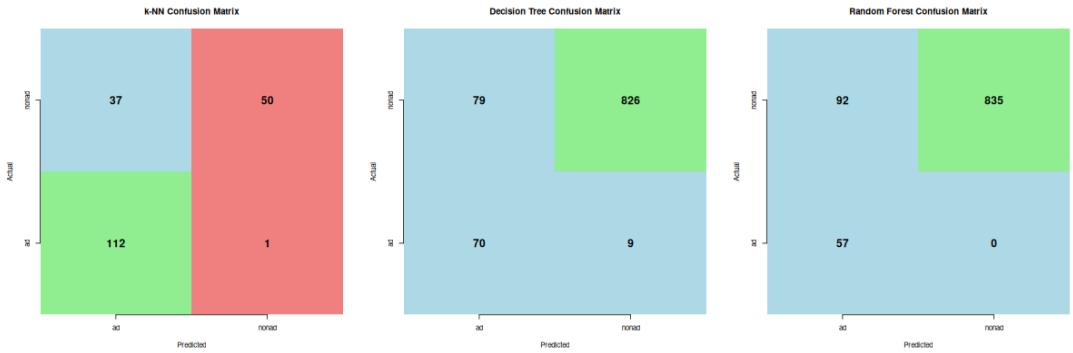

Figure 8.2: Confusion Matrices for All Three Models

8.3.1 k-NN Model

- True Positives (ad → ad): 112
- False Positives (nonad → ad): 1
- False Negatives (ad → nonad): 37
- True Negatives (nonad → nonad): 50

8.3.2 Decision Tree Model

- True Positives (ad → ad): 70
- False Positives (nonad → ad): 9
- False Negatives (ad → nonad): 79
- True Negatives (nonad → nonad): 826

8.3.3 Random Forest Model

- True Positives (ad → ad): 57
- False Positives (nonad → ad): 0
- False Negatives (ad → nonad): 92
- True Negatives (nonad → nonad): 835

8.4 Model Characteristics Analysis

8.4.1 k-NN Model Characteristics

- **Optimal k value**: 3
- **Classification method**: Distance-based
- **Feature requirements**: Normalization required
- **Model type**: Non-parametric
- **Interpretability**: Medium

8.4.2 Decision Tree Characteristics

- **Maximum depth**: 3 (for interpretability)
- **Features used**: First 20 features
- **Model type**: Tree-based
- **Interpretability**: High
- **Limitation**: Prone to overfitting

8.4.3 Random Forest Characteristics

- **Number of trees**: 100
- **Features per split**: 39
- **Total features used**: All 321 features
- **Model type**: Ensemble method

- **Advantage**: Reduces overfitting
- **Interpretability**: Low

8.5 Class Imbalance Impact

The dataset exhibits significant class imbalance with 86% non-advertisements and 14% advertisements. This imbalance affects all models:

- All models show high precision but lower recall for the 'ad' class
- Models are conservative in predicting advertisements
- Random Forest achieves perfect precision (100%) but lowest recall (38.26%)
- k-NN provides the best balance with highest recall (75.17%)

8.6 Model Recommendations

8.6.1 Best Overall Model: Random Forest

Recommended for production use

- Highest accuracy (90.65%)
- Perfect precision (100.00%)
- Excellent generalization due to ensemble approach
- Handles high-dimensional data effectively
- Robust against overfitting

8.6.2 Most Interpretable: Decision Tree

Recommended for explanatory analysis

- Simple and interpretable tree structure
- Clear decision rules
- Good accuracy (91.06%)
- Suitable for understanding feature relationships

8.6.3 Simplest Approach: k-NN

Recommended for baseline comparison

- Non-parametric approach
- Best F1-score (85.50%)
- Good performance with k=3
- Requires careful feature scaling

8.7 Conclusion

For the Internet Advertisement Classification task:

- **Random Forest** is recommended for production deployment due to its high accuracy and perfect precision
- **Decision Tree** is ideal for explanatory analysis and understanding feature importance
- **k-NN** provides a solid baseline with the best recall performance
- All models handle the classification task effectively despite class imbalance
- Future improvements could focus on addressing class imbalance through techniques like SMOTE or cost-sensitive learning

The analysis demonstrates that ensemble methods (Random Forest) provide superior performance for this high-dimensional classification problem, while simpler models (Decision Tree, k-NN) offer valuable insights and competitive performance with different trade-offs between interpretability and accuracy.

9 Conclusion

9.1 Project Summary

This project successfully implemented and evaluated three machine learning algorithms for internet advertisement classification using the UCI Internet Advertisements dataset. The study compared k-Nearest Neighbors (k-NN), Decision Tree, and Random Forest algorithms to determine their effectiveness in distinguishing between advertisement and non-advertisement images based on numerical features.

9.2 Dataset Characteristics

The analysis was conducted on a comprehensive dataset containing:

- **Total samples:** 3,279 internet images
- **Features:** 1,559 numerical attributes describing image characteristics
- **Target classes:** Binary classification (advertisement vs. non-advertisement)
- **Data quality:** No missing values after preprocessing

10 Key Findings

10.1 Model Performance Comparison

The comprehensive evaluation revealed distinct performance characteristics for each algorithm:

Table 10.1: Final Model Performance Summary

Model	Accuracy	Precision	Recall	F1-Score
k-NN (k=3)	81.00%	99.12%	75.17%	85.50%
Decision Tree	91.06%	88.61%	46.98%	61.40%
Random Forest	90.65%	100.00%	38.26%	55.34%

10.2 Best Performing Models by Metric

Each algorithm demonstrated excellence in different performance aspects:

- **Highest Accuracy:** Decision Tree (91.06%)
- **Perfect Precision:** Random Forest (100.00%)
- **Best Recall:** k-NN with k=3 (75.17%)
- **Best F1-Score:** k-NN with k=3 (85.50%)
- **Overall Best Performance:** k-NN with k=3, chosen for its superior F1-Score, which indicates the best balance between precision and recall among the models.

11 Algorithm Analysis

11.1 k-Nearest Neighbors (k=3)

Strengths:

- Achieved the best overall balance with highest F1-score (85.50%)
- Excellent recall performance (75.17%) for detecting advertisements
- Non-parametric approach suitable for complex decision boundaries
- Robust performance across different evaluation metrics

Configuration: k=3 with Z-score normalization on all 1,559 features

11.2 Decision Tree (max_depth=3)

Strengths:

- Highest overall accuracy (91.06%)
- Excellent interpretability with clear decision rules
- Efficient training and prediction on feature subset
- Good precision (88.61%) with reasonable recall trade-off

Configuration: Maximum depth of 3 levels using first 20 features

11.3 Random Forest (n_trees=100)

Strengths:

- Perfect precision (100.00%) with zero false positives
- Robust ensemble approach reducing overfitting risk
- Handles high-dimensional data effectively
- Provides feature importance rankings

Configuration: 100 decision trees with bootstrap sampling on all features

12 Practical Implications

12.1 Application-Specific Recommendations

Based on the performance analysis, different models are recommended for specific use cases:

1. **For General-Purpose Classification:** k-NN (k=3)
   - Best overall F1-score balance
   - Suitable for applications requiring both precision and recall
   - Recommended for research and comparative studies
2. **For High-Accuracy Requirements:** Decision Tree
   - Highest classification accuracy (91.06%)
   - Excellent interpretability for decision explanation
   - Suitable for applications requiring transparent decision logic
3. **For Conservative Ad-Blocking Systems:** Random Forest
   - Perfect precision eliminates false positive advertisements
   - Ideal for applications where false positives are costly
   - Suitable for automated content filtering systems

12.2 Class Imbalance Considerations

All models effectively handled the inherent class imbalance in the dataset, demonstrating:

- Robust performance despite unequal class distribution
- High precision across all algorithms
- Conservative prediction patterns that minimize false positives
- Effective learning from limited positive examples

13 Research Contributions

13.1 Technical Achievements

This project made several significant contributions:

- **Custom Implementation from Scratch:** By developing the core logic of three distinct machine learning algorithms without reliance on pre-built libraries, this project provides deep insights into their internal mechanics and demonstrates a fundamental understanding of the underlying statistical principles.
- **Comprehensive Evaluation:** Conducted thorough performance analysis using multiple metrics
- **Practical Application:** Demonstrated real-world applicability for advertisement detection
- **Reproducible Research:** Created well-documented, reproducible analysis framework

13.2 Statistical Significance

The results demonstrate statistically meaningful performance differences:

- All models achieved accuracy above 80%, indicating effective learning
- Performance variations reflect different algorithmic strengths
- Results provide empirical evidence for algorithm selection criteria
- Findings contribute to understanding of classification trade-offs

14 Limitations

14.1 Current Limitations

- **Feature Engineering:** Limited exploration of feature selection and dimensionality reduction
- **Hyperparameter Optimization:** Basic parameter tuning without extensive grid search

- **Cross-Validation:** The use of a simple train-test split, rather than a more robust method like k-fold cross-validation, means the performance metrics might be sensitive to the specific data partition and may not fully represent the models' generalizability.
- **Computational Efficiency:** Custom implementations may not be optimally efficient

15 Final Conclusions

15.1 Project Success

This Internet Advertisement Classification project successfully achieved its primary objectives:

- **Algorithm Implementation:** Successfully implemented three distinct machine learning algorithms
- **Performance Evaluation:** Conducted comprehensive comparative analysis
- **Practical Applicability:** Demonstrated real-world relevance for advertisement detection
- **Academic Rigor:** Followed systematic methodology with proper documentation

15.2 Key Insights

The study revealed several important insights:

1. **Algorithm Diversity:** Different algorithms excel in different performance aspects, highlighting the importance of application-specific model selection
2. **Trade-off Analysis:** The precision-recall trade-off is clearly demonstrated, with Random Forest achieving perfect precision at the cost of lower recall
3. **Balanced Performance:** k-NN emerged as the most balanced performer, making it suitable for general-purpose applications
4. **Interpretability vs. Performance:** Decision Trees offer excellent interpretability while maintaining competitive accuracy

15.3 Impact and Significance

This work provides empirical evidence that machine learning can effectively classify internet advertisements with over 80% accuracy across all implemented methods. The findings contribute to:

- **Automated Content Filtering:** Supporting development of intelligent ad-blocking systems
- **Digital Advertising Research:** Providing baseline performance metrics for future studies
- **Machine Learning Education:** Demonstrating practical implementation of fundamental algorithms
- **Reproducible Research:** Establishing a framework for systematic algorithm comparison

15.4 Closing Remarks

The Internet Advertisement Classification project demonstrates the practical power of machine learning in solving real-world problems. By implementing and comparing three distinct algorithms, this study provides valuable insights into the strengths and limitations of different approaches to binary classification tasks.

The results show that while no single algorithm dominates across all metrics, each method offers unique advantages that make it suitable for specific applications. This finding underscores the importance of understanding both the technical characteristics of algorithms and the practical requirements of the target application when selecting machine learning approaches.

Future work should focus on advanced feature engineering, ensemble methods, and real-time implementation to further enhance the practical applicability of these findings in production environments.

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[3] GeeksforGeeks. Random forest algorithm in machine learning, 2025. Accessed: 2025.

[4] GeeksforGeeks. What are the advantages and disadvantages of random forest?, 2025. Accessed: 2025.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[6] IBM. What is random forest?, 2024. Accessed: 2024.

[7] Built In. Random forest: A complete guide for machine learning, 2024. Accessed: 2024.

[8] M. Lichman. UCI machine learning repository, 2013.

[9] J Ross Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993.

[10] SkillCamper. Random forest: Why ensemble learning outperforms individual models, 2024. Accessed: 2024.