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 Random Forest

Advisor(s):	[Instructor Name]	
Student(s):	NGUYỄN LÊ THANH HÀO	1952242
	PHẠM NHẬT KHÔI	2352623
	Nguyễn Song Đạt	1952646
	Nguyễn Phước Thịnh	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 $\times$ 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.

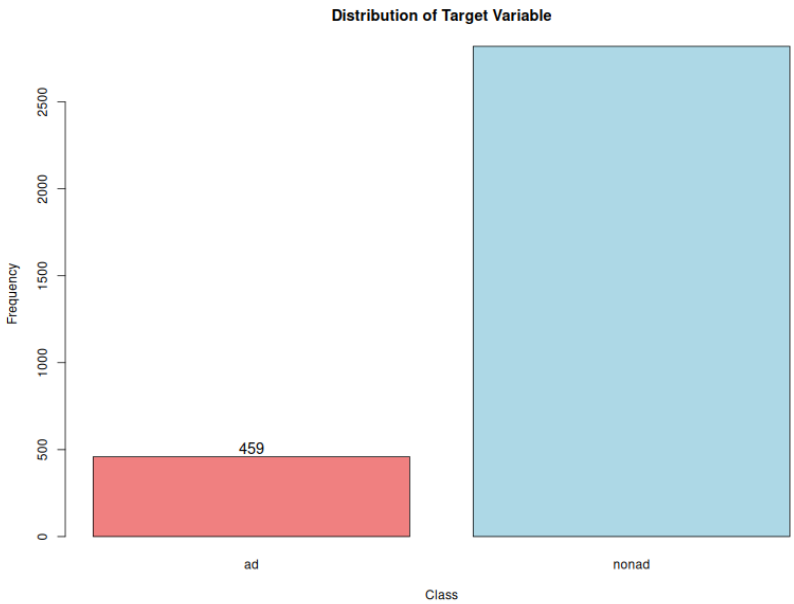

Figure 3.1: Distribution of target variable showing class imbalance

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:

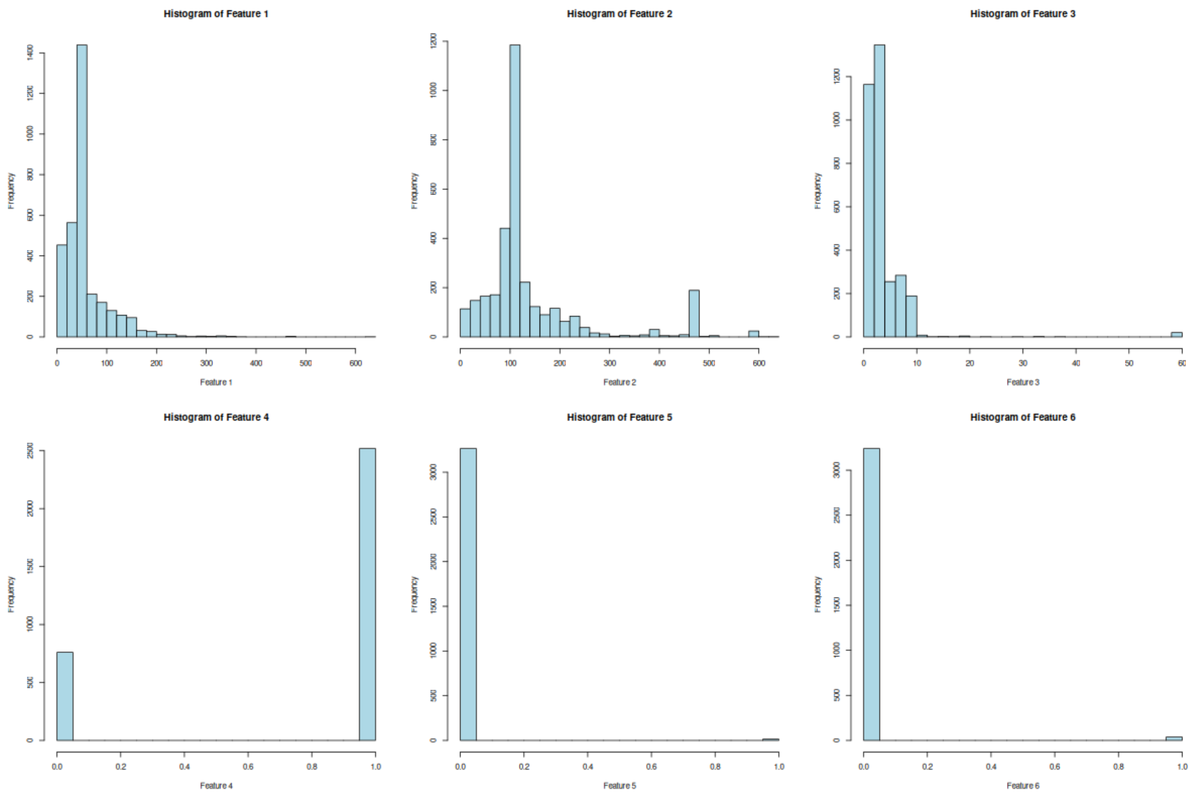

Figure 3.2: Histograms of the first six features showing distribution patterns

The histogram generation code:

```
# 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:

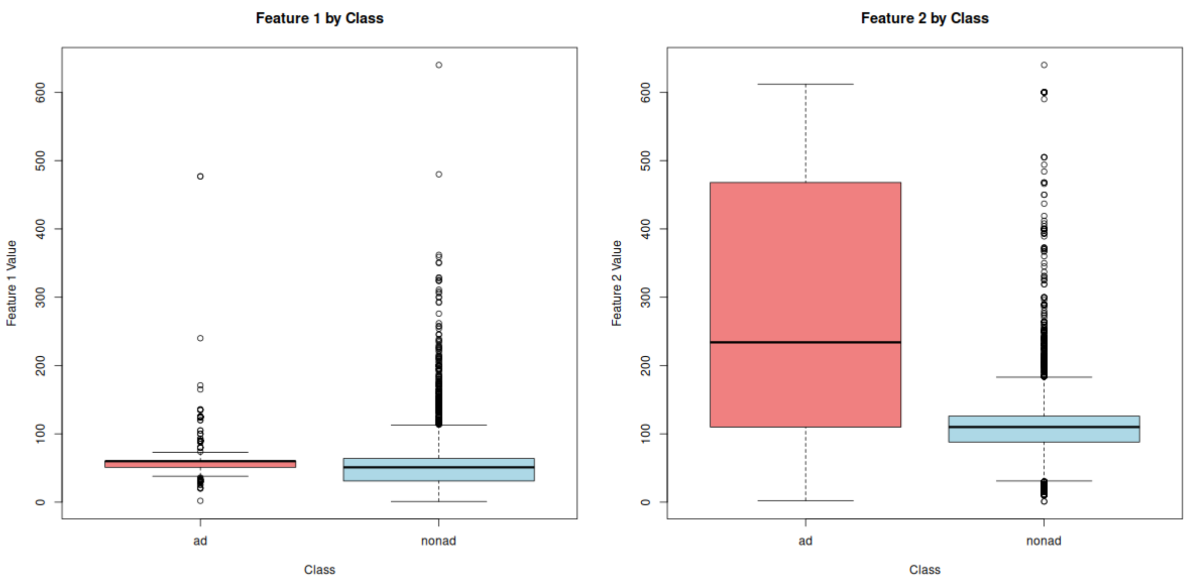

Figure 3.3: Boxplots of features 1 and 2 grouped by target class

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:

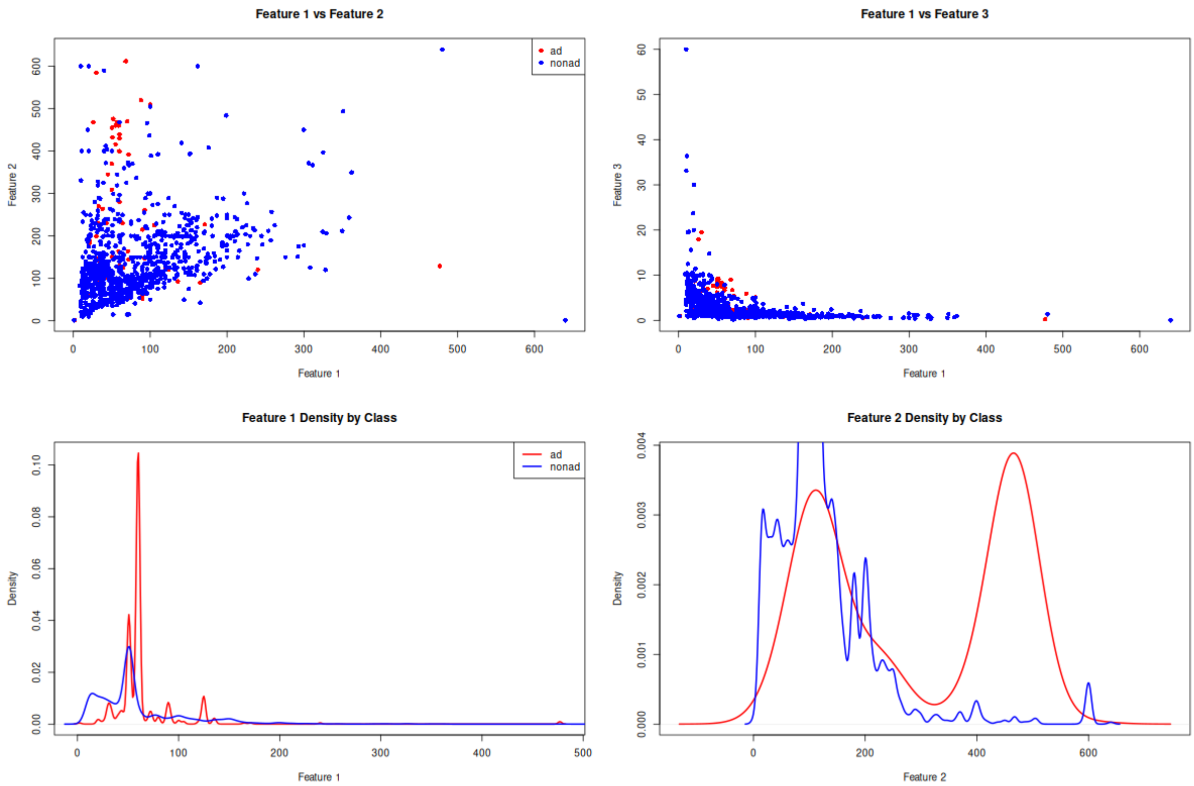

Figure 3.4: Scatter plots and density plots showing feature relationships and class distributions

The visualization combines scatter plots and density analysis:

```r
# 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:

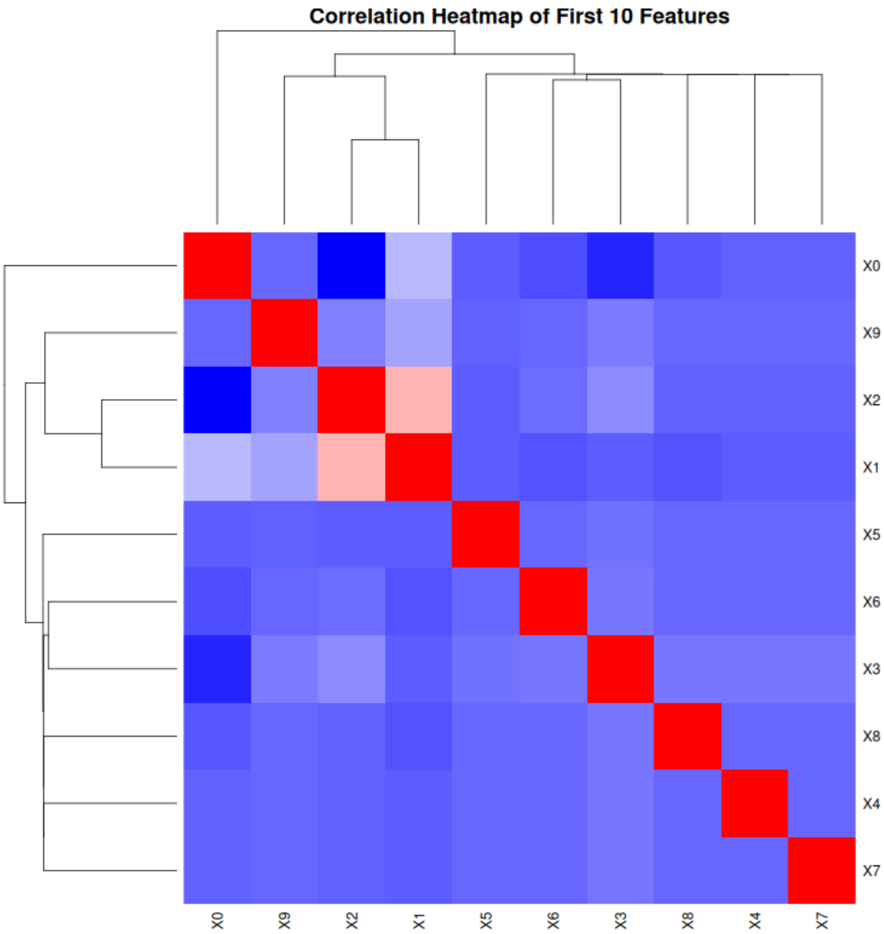

Figure 3.5: Correlation heatmap of the first 10 features

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:

```
# 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 $f(\mathbf{x}) \rightarrow \{0, 1\}$ 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

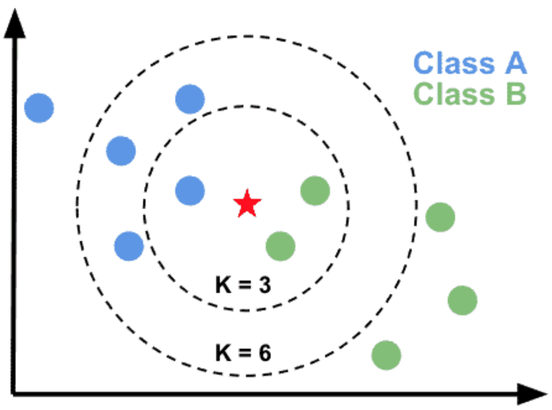

Figure 4.1: k-NN Classification Example with k=3 Nearest Neighbors

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

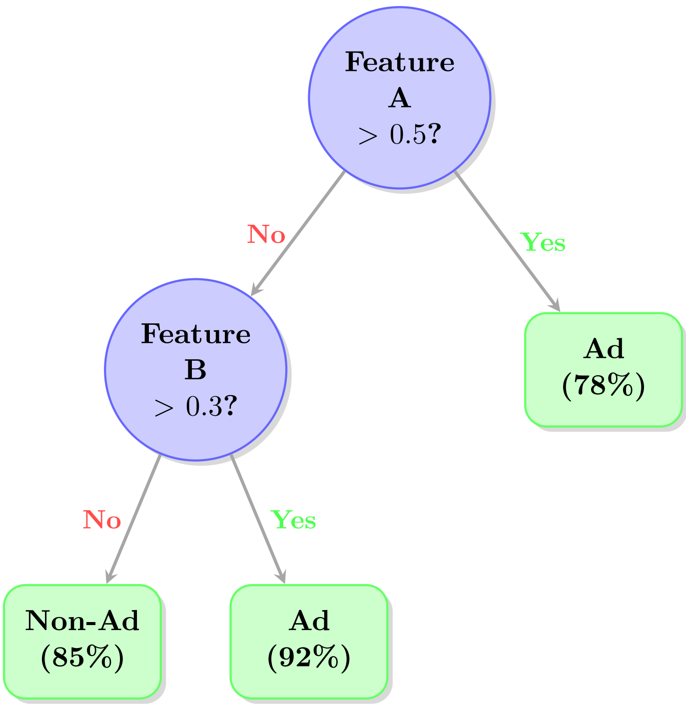

Figure 4.2: Decision Tree Example for Ad Classification

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

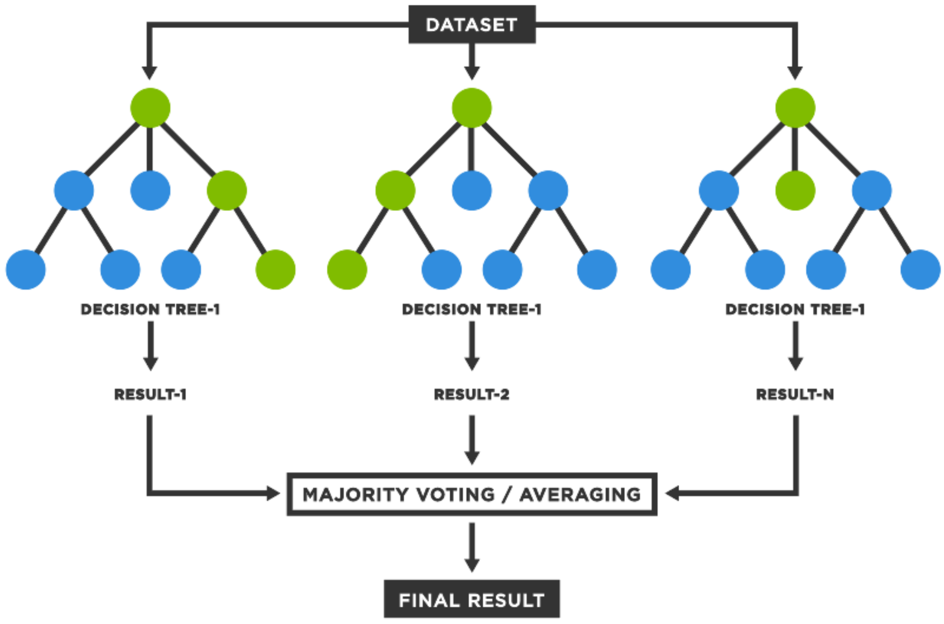

Figure 4.3: Random Forest: Ensemble of Decision Trees for Ad Classification

- **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.

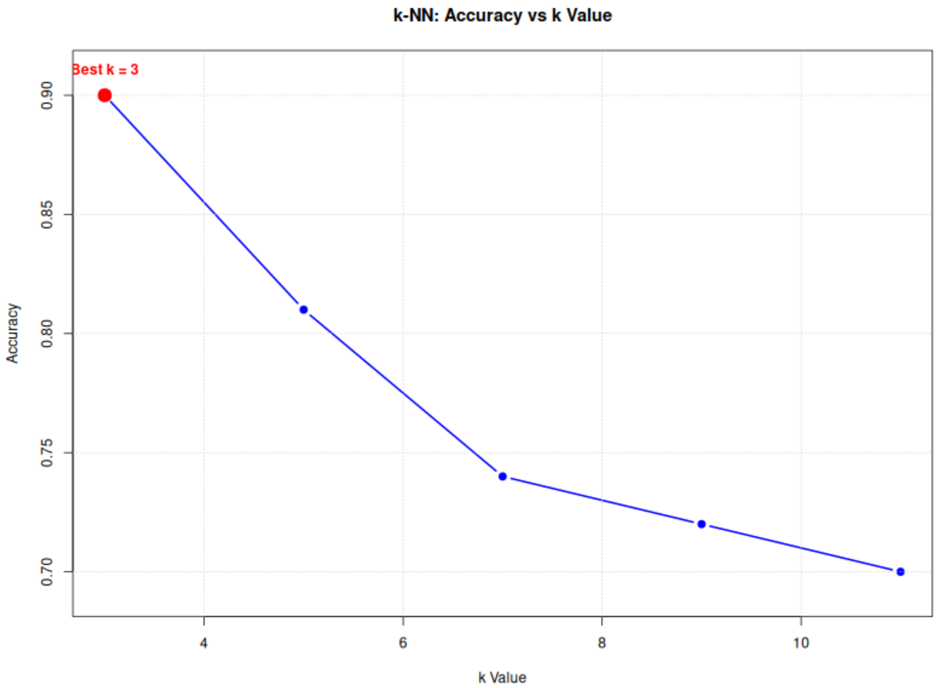

Figure 5.1: k-NN accuracy vs k value showing optimal performance at k=3

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:

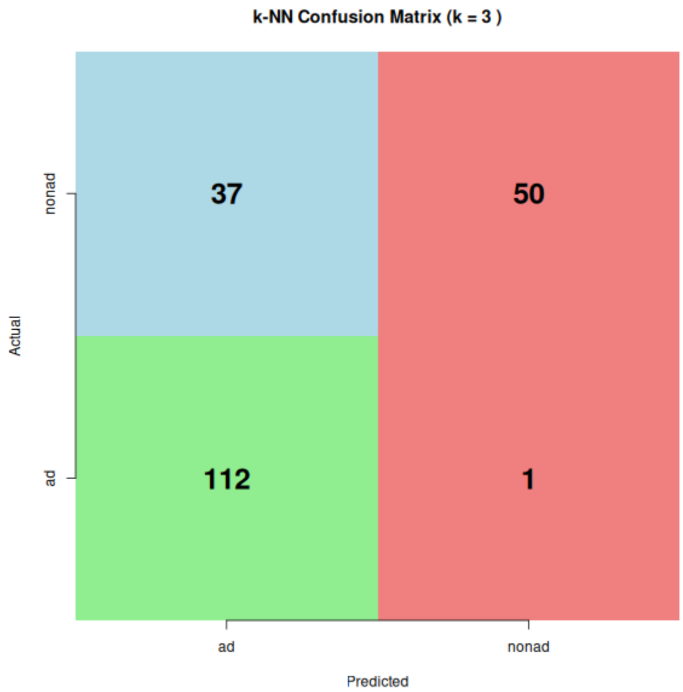

Figure 5.2: k-NN confusion matrix showing classification results

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.

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.