CS699
Data Mining

Final Project Report

David Chavez

Finn Graham

Summer Term 2024

Table of contents

1 Data Mining tools used

To execute this project effectively, we followed a structured methodology, This approach involved cleaning the dataset and applying two balancing methods to ensure fairness, followed by feature engineering with three attribute selection techniques to refine the features. We then developed six different classification algorithms and evaluated thirty six models variations to determine the most effective approach. To carry out this work, we used the following data mining tools.

1.1 Main Libraries and intended use

- **caret**: Predictive modeling and evaluation.
- **rsample**: Data sampling and validation.
- **RWeka**: Data mining algorithms.
- **rpart**: Decision trees.
- **pROC**: ROC curve analysis.
- **e1071**: Support vector machines.
- **ROSE**: Imbalanced class balancing.
- **FSelector**: Feature selection.
- **Boruta**: Relevant feature selection.
- **ada**: Boosting algorithm.
- **corrplot**: Correlation matrix visualization.
- **earth**: Multivariate adaptive regression splines.

.

1.2 Data Dimension Reduction

NearZeroVar:Used to identify and remove features with very low variability, where values are almost constant or show minimal variation. This helps in reducing the dimensionality of the dataset, improving model performance, and enhancing interpretability by focusing on more informative and relevant features.

Columns Removed

CHCKDNY	**DIFFDRES**	**USENOW3**	**HIVRISK5**

Columns that will not be useful for analysis based on domain knowledge: Columns related to the administration of the interview are removed because they do not provide predictive value and may introduce noise into the model.

Columns Removed

FMONTH	**IDATE**	**IMONTH**	**IDAY**
IYEAR	**DISPCODE**	**SEQNO**	**MARITAL**
EDUCA	**RENTHOM1**	**CPDEMO1B**	**EMPLOY1**
INCOME2	**QSTVER**	**QSTLANG**	**HTIN4**
WEIGHT2	**HEIGHT3**		

Highly correlated columns: Columns with high correlation are often removed to prevent multicollinearity, which can distort the estimated coefficients and reduce model interpretability. Tools such as **corrplot** for visualization and **caret** for feature selection were used to identify and handle these highly correlated columns.

Column Removed

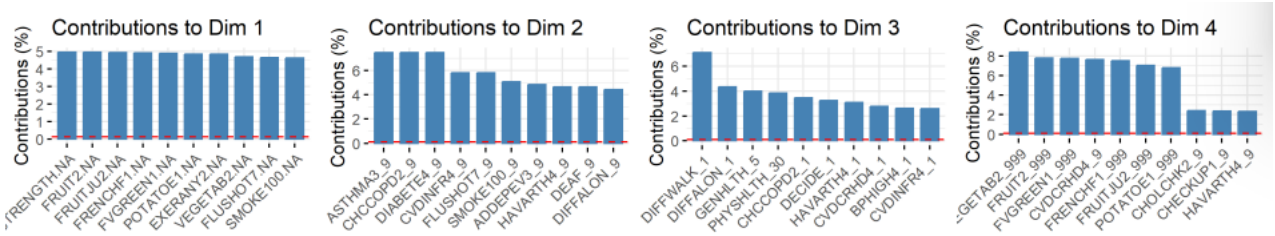
ALCDAY5

Multiple Correspondence Analysis: Multiple Correspondence Analysis (MCA) is used to reduce dimensionality and uncover underlying patterns in categorical data, enhancing model simplicity and effectiveness.

Columns Retained

ASTHMA3	**CHCCOPD2**	**DIABETE4**	**CVDINFR4**
FLUSHOT7	**SMOKE100**	**STRENGTH**	**FRUIT2**
FRUITJU2	**FRENCHF1**	**FVGREEN1**	**ADDEPEV3**
POTATOE1	**EXERANY2**	**VEGETAB2**	**HAVARTH4**
DEAF	**DIFFALON**	**PNEUVAC4**	**DRNKANY5**
DIFFWALK	**DECIDE**		

The four graphs, represent the columns with the highest variance after performing MCA, illustrate the dominant dimensions of variability, highlighting key categorical relationships and differences within the dataset.



1.3 Data Normalization

Min-Max Scaling Method: The dataset was normalized using the Min-Max Scaling tool. This method transforms the data such that all numeric features are scaled to a range between 0 and 1. For each numeric feature, the minimum and maximum values were used to scale the data points. The formula for Min-Max Scaling is:

$$X' = \frac{(X - X_{\min})}{(X_{\max} - X_{\min})}$$

where X is the original data point, $X_{\min}$ is the minimum value in the column, and $X_{\max}$ is the maximum value in the column. This transformation ensures that the data points are proportionally spaced within the [0, 1] range.

1.4 Filling Missing Values

Correlation: We automate missing value imputation in a dataframe using linear regression models that leverage strong variable correlations to maintain data integrity and enhance analysis accuracy

Step	Description
1	**Compute Correlation Matrix**: Calculate the correlation matrix only from complete observations.
2	**Convert Correlation to Dataframe**: Transform the correlation matrix to a dataframe for ease of manipulation.
3	**Identify Columns with NAs**: Detect all columns that contain missing values.
4	**Find Highest Correlated Columns**: For each column with missing values, identify the column that has the highest correlation with it.
5	**Build Regression Models**: Construct a regression model for each pair of highly correlated columns.
6	**Predict Missing Values**: Use the regression models to estimate and fill in the missing values.

Implementation Details

- The correlation matrix is computed using only those rows that do not contain any missing values to ensure accuracy in correlation calculation.
- Each column's strongest correlation is determined by absolute value, meaning both strong positive and negative correlations are considered.
- Regression models are built individually for each column with missing data based on its most correlated counterpart, providing tailored imputation.

- Predictions from these models directly replace the missing values, thus maintaining the integrity and distribution of the original data.

Calculation The correlation coefficient between two features X and Y is calculated as:

$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Correlation heatmap : This graph, a correlation heatmap, visually represents the strength and direction of relationships between variables in the dataset, using color intensity to indicate the degree of correlation.

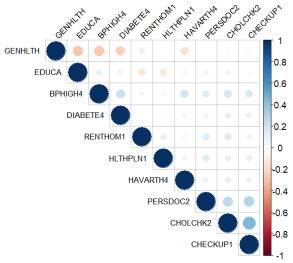


Figure 1: correlation variable 1 to 10

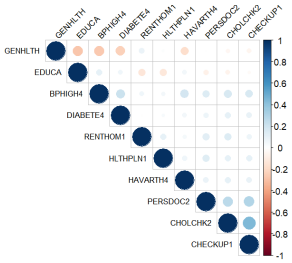


Figure 2: correlation variable 11 to 20

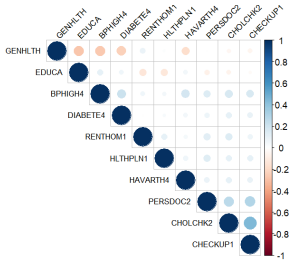


Figure 3: correlation variable 21 to 30

1.5 Data Balancing

OVUN.SAMPLE The function from the ROSE (Random Over-Sampling Examples) package is used as a tool to create balanced samples in a dataset by both over-sampling the minority class and under-sampling the majority class. This method ensures that the class distribution is more balanced, which is crucial for training effective machine learning models.

MOTEFAMILY: Used to address class imbalance by creating synthetic samples for the minority class. The `SMOTE` function was applied, which selects a random sample from the minority class and identifies its k-nearest neighbors. Synthetic samples are then generated along the line segments connecting these neighbors. This approach enhances the representation of the minority class without merely duplicating existing samples.

1.6 Feature Selection

The `FSelector` function in R was utilized: Tool used to rank the features based on their relevance in predicting the target variable. The `information.gain` function was applied to compute Information Gain scores for each feature. Features with higher scores were prioritized and selected for inclusion in the model building process.

The `boruta` function in R was utilized: The function performed multiple iterations of adding noise and comparing the original features against the noise features. This method helps in identifying all relevant features by distinguishing them from the irrelevant ones, ensuring that only the most significant variables are retained for model training.

The `rfe` function in R was utilized:The function was used to recursively remove less important features by fitting the model and evaluating its performance iteratively. At each iteration, the least significant features were eliminated based on their contribution to the model's accuracy. This process continued until the optimal number of features was reached, thus selecting the subset of features that contributed most to the model's predictive power.

1.7 Model Building and Evaluation

Cross-Validation: To evaluate the models' performance, a repeated 10-fold cross-validation method was employed. The dataset was divided into ten folds, with each fold serving as a validation set while the remaining nine folds were used for training. This process was repeated five times, shuffling the data and creating new sets of folds for each repetition. The results from these repeated cross-validation runs were averaged to obtain robust performance metrics, providing a more reliable assessment of the model's generalization ability.

Hyperparameter Tuning: Hyperparameters of the classification algorithms were optimized using grid search. Various combinations of hyperparameters were evaluated, and the combination that yielded the best performance was chosen. This process of hyperparameter tuning ensures that the models are tailored for optimal performance on the given data.

2 Detailed Description of Data Mining Procedures

2.1 Preprocessing

Multiple preprocessing techniques were applied to ready the dataset for constructing classification models. These techniques encompassed managing missing data, identifying and addressing outliers, scaling numerical features, and executing feature selection. Below is an in-depth explanation of each preprocessing approach and the tasks completed:

2.1.1 Normalization of Variables

Automated Variable Normalization: To standardize the numeric features in the dataframe, we applied Min-Max scaling. This technique transforms the data to a range between 0 and 1, facilitating better model performance. First, for each numeric column, the minimum and maximum values were determined. Each value in the column was then scaled using the formula $(X - X_{\min})/(X_{\max} - X_{\min})$, where X is the original value, $X_{\min}$ is the minimum value of the column, and $X_{\max}$ is the maximum value of the column. This ensured that all numeric features were proportionally adjusted, preserving their original relationships and distributions while fitting them into a uniform scale.
Formula Applied

```r
scale <- function(x) {
  (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
```

Two examples of categorization based in domain knowledge

```
#MENTHLTH
df$MENTHLTH[!(df$MENTHLTH  %in% c(1:30))] <- NA
df$MENTHLTH <- scale(df$MENTHLTH)

#CHOLCHK2
df$CHOLCHK2[!(df$CHOLCHK2 %in% c(1,2,3,4,5,6))] <- NA
df$CHOLCHK2<- scale(df$CHOLCHK2)
```

2.1.2 Data Dimension Reduction

Reducing Data Dimensions: To enhance model performance and reduce computational complexity, we applied several techniques for dimensionality reduction. Firstly, we used the `nearZeroVar` function to identify and remove columns with near-zero variance, which are unlikely to be useful for analysis. Additionally, based on domain knowledge, we excluded columns that would not contribute meaningful information. Finally, we addressed multicollinearity by identifying and removing highly correlated columns using correlation analysis and the MCA (Multiple Correspondence Analysis) method. This comprehensive approach ensured that the remaining features were more informative and relevant for the analysis.
Columns with correlation

```
highly.correlated.variables <- findCorrelation(cor(numeric_cols, use = "
    complete.obs"), cutoff = 0.7, names = TRUE)
df <- df %>% select(-all_of(highly.correlated.variables))
```

Columns with almost 0 variance

```
df <- df %>%
  select(-all_of(nearZeroVar(df, names = TRUE)))
```

Multiple Correspondence Analysis

```
# Perform MCA
mca_results <- MCA(df_factors, graph = FALSE)
```

2.1.3 Filling Missing Values

Automated Missing Value Imputation: To handle missing values in the dataframe, we employed linear regression models based on strong correlations between variables. First, the correlation matrix was computed using only complete observations to ensure accuracy. This matrix was then converted into a dataframe for easier manipulation. Columns with missing values were identified, and for each of these columns, the one with the highest correlation was found. Regression models were constructed for each pair of highly correlated columns, and these models were used to predict and fill in the missing values. This method maintained the integrity and distribution of the original data by considering both strong positive and negative correlations, ensuring precise and tailored imputation.
We use formula created to accomplish task

```
df <- fill_missing_with_regression(df)
```

Values that were not able to be filled by the correlation formula are filled with the mode

```
get_mode <- function(x) {uniq_x <- unique(x)
uniq_x[which.max(tabulate(match(x, uniq_x)))]}
for (col in names(df)) {mode_value <- get_mode(df[[col]][!is.na(df[[col]])])
df[[col]][is.na(df[[col]])] <- mode_value }
numeric_cols <- sapply(df, is.numeric)
df[numeric_cols] <- round(df[numeric_cols], 1)
```

2.2 Feature Engineering

2.2.1 Data Splitting

Training and Test Set Split using initial_split: The dataset was divided into training and test sets using the `initial_split()` function from the `rsample` package. By setting the seed to 17 and specifying a proportion of 66%, the function ensured that 66% of the data was allocated to the training set and the remaining 34% to the test set, while also stratifying based on the `Class` variable. This approach guaranteed a representative split of the data for both training and evaluation purposes.

```
# Create Balanced Dataset - Method 1
set.seed(17)
split <- initial_split(df, prop = 0.66, strata = Class)
train <- training(split)
test <- testing(split)
```

Training and Test Set Split using sample: The dataset was split into training and test sets using a random sampling method. The seed was set to 17 to ensure reproducibility, and a sample proportion of 66% was used to determine the split. Specifically, 66% of the rows were randomly selected for the training set, while the remaining 34% constituted the test set. This method provided an effective way to separate the data for model training and evaluation.

```
set.seed(17)
sample <- sample(c(TRUE, FALSE), nrow(df), replace=TRUE, prob=c(0.66, 0.34))
train2 <- df[sample, ]
test2 <- df[!sample, ]
```

2.2.2 Data Balancing

SMOTE (Synthetic Minority Over-sampling Technique): SMOTE was applied to the training set to generate synthetic samples for the minority class. This technique helps in addressing class imbalance by creating new instances along the line segments joining minority class samples.

```
smote_output <- SMOTE(X = train2[, -which(names(train) == "Class")], target =
    train2$Class)
train_smote <- smote_output$data
train_smote$Class <- as.factor(train_smote$class)
train_smote$class <- NULL
train2 <- train_smote]
```

ROSE (Random Over-Sampling Examples): ROSE was used to create balanced samples by generating new synthetic data points for the minority class and under-sampling the majority class. This technique ensures a more balanced class distribution in the training set.

```
train <- ovun.sample(Class ~ ., data = train, method = "both", N = 700, seed
    = 1)$data
test <- testing(split)
```

These preprocessing steps were essential in ensuring that the dataset was clean, balanced, and well-prepared for building robust classification models. Each step contributed to improving the accuracy and reliability of the models by addressing issues related to missing values, outliers, and feature selection.

2.2.3 Feature Selection

Information Gain: Information Gain is a method used to identify and select the most relevant features in a dataset. By measuring how well a feature splits the data, it helps in reducing dimensionality and focusing on the variables that contribute the most to the model's predictive power.

```
weights <- information.gain(Class ~ ., train)
selected_features <- rownames(subset(weights, attr_importance > 0))
selected_features <- c("Class", selected_features)
ig_train <- train[, selected_features]
ig_test <- test[, selected_features]
```

BORUTA: Boruta is an all relevant feature selection method, while most other are minimal optimal; this means it tries to find all features carrying information usable for prediction, rather than finding a possibly compact subset of features on which some classifier has a minimal error.

```
train$Class <- as.factor(train$Class)
boruta_obj <- Boruta(Class ~ ., train)
boruta_result <- attStats(boruta_obj)
relevant_features <- getSelectedAttributes(boruta_obj, withTentative = TRUE)
relevant_features <- c("Class", relevant_features)
bor_train <- train[, relevant_features]
bor_test <- test[, relevant_features]
```

RFE: RFE iteratively removes less important features, creating a subset that maximizes predictive accuracy. By leveraging a machine learning algorithm and an importance-ranking metric, RFE evaluates each feature's impact on model performance.

```
svm_rfe_result <- rfe(
  x = train[, -1],
  y = train$Class,
  sizes = c(5, 10, 15, 20),
  rfeControl = control,
  method = "svmLinear"  )
print(svm_rfe_result)
selected_features_svm <- predictors(svm_rfe_result)
print(selected_features_svm)
rfe_train <- train[, c("Class", selected_features_svm)]
rfe_test <- test[, c("Class", selected_features_svm)]
```

2.3 Classification Algorithms

In this project, several classification algorithms were utilized to build models for predicting cancer diagnosis. Each algorithm has unique characteristics and strengths that contribute to its performance. Here is a detailed description of each classification algorithm used:

2.3.1 Adaptive Boosting (ADA)

ADA, or Adaptive Boosting, is an ensemble learning technique that combines multiple weak classifiers to create a strong classifier. It focuses on the instances that are hard to classify by assigning them higher weights in subsequent rounds. This method improves the overall model accuracy by iteratively adjusting the weights of incorrectly classified instances.
The `ada` package in R was used to implement the Adaptive Boosting algorithm. The `ada` function was employed to build the model, adjusting the weights of the instances in each iteration to improve the prediction accuracy.

```
ada_grid <- expand.grid(iter = 1, maxdepth = 1:2, nu = seq(0.1, 1, by=0.2))
ctrl <- trainControl(
  method = "LOOCV",  # Leave-one-out cross-validation
  savePredictions = "final",
  classProbs = TRUE,       # Needed for classification
  summaryFunction = twoClassSummary  # Use two-class summary function for
     classification
)
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 5,
                     verboseIter = FALSE,
                     sampling = "up")
ig_ada_model <- train(Class ~ .,
                      data = ig_train,
                      method = "ada",
                      trControl = ctrl,
```

2.3.2 Random Forest (RF)

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It improves accuracy by reducing overfitting.
The `randomForest` package was used to build random forest models, with hyperparameter tuning for the number of variables randomly sampled as candidates at each split (mtry).

```
ctrl <- trainControl(method = "CV",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

mtryValues <- seq(2, ncol(ig_train)-1, by = 1)
rfFit <- caret::train(x = ig_train[, -1],
                      y = ig_train$Class,
                      method = "rf",
                      ntree = 500,
```

```
                        tuneGrid = data.frame(mtry = mtryValues),
                        importance = TRUE,
                        metric = "ROC",
                        trControl = ctrl)
```

2.3.3 Generalized Linear Model (GLM)

Flexible generalization of ordinary linear regression that allows for the dependent variable to have a non-normal distribution. In this project, logistic regression (a type of GLM) was used, where the outcome variable is binary (cancer diagnosis: Yes or No).
The `glm` function in R was used with a binomial family to perform logistic regression.

```
glm_grid <- expand.grid(.parameter = seq(1, 10, 1))
ig_glm_model <- train(Class ~ .,
                        data = ig_train,
                        method = "glm",
                        preProcess = c("scale", "center"),
                        trControl = ctrl,
                        tuneGrid = glm_grid)
```

2.3.4 Support Vector Machine (SVM)

Description: SVM is a supervised learning model that analyzes data for classification and regression analysis. It constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate different classes. The best hyperplane is the one that maximizes the margin between classes.
The `kernlab` package was used to implement SVM with a radial basis function (RBF) kernel. Hyperparameters tuned included the cost parameter (C) and the kernel parameter (sigma).

```
rain_control <- trainControl(method = "CV", number = 10,
                                summaryFunction = defaultSummary)
svmGrid <-  expand.grid(sigma = seq(0.1, 0.4, by = 0.05), C = seq(1.0, 2.0,
   by = 0.1))

model <- caret::train(Class ~ ., data = ig_train, method = "svmRadial",
                        preProc = c("center", "scale"),
                        trControl = train_control, tuneGrid = svmGrid)
```

2.3.5 Conditional Inference Tree (ctree)

Description: Conditional Inference Tree (ctree) is a non-parametric class of decision trees that are used for classification and regression tasks. Unlike traditional decision trees, ctree uses statistical tests to select splits, avoiding overfitting and biased variable selection. This approach provides a more robust and interpretable model.
The `party` package in R was used to implement the ctree algorithm. The `ctree` function was employed to build the model, using statistical tests to determine the optimal splits at each node of the tree.

```
ctrl <- trainControl(method = "CV", number = 10)
```

```
ctreeGrid <-  expand.grid(mincriterion = 0.05)

model <- caret::train(Class ~ ., data = ig_train, method = "ctree",
                      trControl = ctrl, tuneGrid = ctreeGrid )
```

2.3.6 Multivariate Adaptive Regression Splines (gcvEarth)

Description: Multivariate Adaptive Regression Splines (MARS) is a non-parametric regression technique that models relationships by fitting piecewise linear regressions. The method automatically selects the number and location of knots to capture non-linearities and interactions in the data. The Generalized Cross-Validation (GCV) criterion is used to prevent overfitting by balancing model complexity and fit.
The `earth` package in R was used to implement MARS with GCV. The `earth` function was employed to build the model, using the GCV criterion to select the optimal number of knots and terms for the regression splines.

```
ctrl <- trainControl(method = "CV", number = 10)

gcvGrid<- expand.grid(
  degree = c(1, 2, 3)  # Adjust based on your specific needs
)

model <- caret::train(Class ~ ., data = ig_train, method = "gcvEarth",
                      trControl = ctrl, tuneGrid = gcvGrid )
```

3 Results and Evaluation

This section presents the results of the data mining procedure, including performance measures for the best configurations of each model built. Due to the extensive number of combinations, only the best result for each model is displayed to maintain clarity and conciseness.

3.1 Model 1 - Data splitting method 1 - Information Gain - ADA

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.6	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.6	0.2	0.2

Table 1: Performance Metrics for Each Class with Weighted Averages

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 2: Performance Metrics for Each Class with Weighted Averages

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.5	0.9	0.6	0.1	0.1
Class Y	0.2	0.1	0.6	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.4	0.1	0.1

Table 3: Performance Metrics for Each Class with Weighted Averages

3.2 Model 2 - Data splitting method 1 - Boruta - ADA

3.3 Model 3 - Data splitting method 1 - RFE - ADA

3.4 Model 4 - Data splitting/balancing method 2 - RFE - ADA

3.5 Model 5 - Data splitting/balancing method 2 - RFE

3.6 Model 6 - Data splitting/balancing method 2 - RFE

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.6	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 6: Updated Performance Metrics for Each Class with Weighted Averages

3.7 Model 7 - Data splitting/balancing method 1 - Information Gain - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 7: Performance Metrics for Model 7

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.6	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 4: Performance Metrics for Each Class with Weighted Averages

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 5: Updated Performance Metrics for Each Class with Weighted Averages

3.8 Model 8 - Data splitting/balancing method 1 - Boruta - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 8: Performance Metrics for Model 8

3.9 Model 9 - Data splitting/balancing method 1 - RFE - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.1
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.1
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.1

Table 9: Performance Metrics for Model 9

3.10 Model 10 - Data splitting/balancing method 2 - Information Gain - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.9	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.2	0.3	0.2	0.1	0.1
Weighted Average	0.8	0.7	0.8	0.8	0.7	0.1	0.1

Table 10: Performance Metrics for Model 10

3.11 Model 11 - Data splitting/balancing method 2 - Boruta - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 11: Performance Metrics for Model 11

3.12 Model 12 - Data splitting/balancing method 2 - RFE - RF

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.9	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.2	0.3	0.2	0.1	0.1
Weighted Average	0.8	0.7	0.8	0.8	0.7	0.1	0.1

Table 12: Performance Metrics for Model 12

3.13 Model 13 - Data splitting/balancing method 1 - Information Gain - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

article caption float

3.14 Model 14 - Data splitting/balancing method 1 - Boruta - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 13: Performance Metrics for Model 14

3.15 Model 15 - Data splitting/balancing method 1 - RFE - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 14: Performance Metrics for Model 15

3.16 Model 16 - Data splitting/balancing method 2 - Information Gain - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.6	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.7	0.7	0.7	0.2	0.2

Table 15: Performance Metrics for Model 16

3.17 Model 17 - Data splitting/balancing method 2 - Boruta - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 16: Performance Metrics for Model 17

3.18 Model 18 - Data splitting/balancing method 2 - RFE - GLM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.6	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.7	0.7	0.7	0.2	0.2

Table 17: Performance Metrics for Model 18

3.19 Model 19 - Data splitting/balancing method 1 - Information Gain - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 18: Performance Metrics for Model 19

3.20 Model 20 - Data splitting/balancing method 1 - Boruta - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.8	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.4	0.3	0.3	0.2	0.2
Weighted Average	0.8	0.6	0.7	0.8	0.7	0.2	0.2

Table 19: Performance Metrics for Model 20

3.21 Model 21 - Data splitting/balancing method 1 - RFE - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.8	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.4	0.3	0.3	0.2	0.2
Weighted Average	0.8	0.6	0.7	0.8	0.7	0.2	0.2

Table 20: Performance Metrics for Model 21

3.22 Model 22 - Data splitting/balancing method 2 - Information Gain - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.8	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.4	0.3	0.3	0.2	0.2
Weighted Average	0.7	0.5	0.7	0.7	0.7	0.2	0.2

Table 21: Performance Metrics for Model 22

3.23 Model 23 - Data splitting/balancing method 2 - Boruta - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.8	0.8	0.8	0.8	0.1	0.1
Class Y	0.2	0.2	0.3	0.2	0.2	0.1	0.1
Weighted Average	0.7	0.7	0.7	0.7	0.7	0.1	0.1

Table 22: Performance Metrics for Model 23

3.24 Model 24 - Data splitting/balancing method 2 - RFE - SVM

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.8	0.9	0.8	0.8	0	0
Class Y	0.2	0.2	0.2	0.2	0.2	0	0
Weighted Average	0.7	0.7	0.8	0.7	0.7	0	0

Table 23: Performance Metrics for Model 24

article caption float

3.25 Model 25 - Data splitting/balancing method 1 - Information Gain - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 24: Performance Metrics for Model 25

3.26 Model 26 - Data splitting/balancing method 1 - Boruta - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.5	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 25: Performance Metrics for Model 26

3.27 Model 27 - Data splitting/balancing method 1 - RFE - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.5	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 26: Performance Metrics for Model 27

3.28 Model 28 - Data splitting/balancing method 2 - Information Gain - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.8	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.3	0.3	0.3	0.2	0.2
Weighted Average	0.8	0.6	0.7	0.8	0.7	0.2	0.2

Table 27: Performance Metrics for Model 28

3.29 Model 29 - Data splitting/balancing method 2 - Boruta - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.5	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 28: Performance Metrics for Model 29

3.30 Model 30 - Data splitting/balancing method 2 - RFE - ctree

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.8	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.3	0.3	0.3	0.1	0.1
Weighted Average	0.7	0.6	0.7	0.7	0.7	0.1	0.1

Table 29: Performance Metrics for Model 30

3.31 Model 31 - Data splitting/balancing method 1 - Information Gain - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.6	0.9	0.7	0.2	0.2
Class Y	0.3	0.1	0.7	0.3	0.4	0.2	0.2
Weighted Average	0.6	0.4	0.6	0.6	0.6	0.2	0.2

Table 30: Performance Metrics for Model 31

3.32 Model 32 - Data splitting/balancing method 1 - Boruta - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.6	0.9	0.7	0.1	0.1
Class Y	0.2	0.1	0.5	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.5	0.1	0.1

Table 31: Performance Metrics for Model 32

3.33 Model 33 - Data splitting/balancing method 1 - RFE - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.7	0.7	0.9	0.8	0.2	0.2
Class Y	0.3	0.1	0.5	0.3	0.4	0.2	0.2
Weighted Average	0.7	0.5	0.6	0.7	0.7	0.2	0.2

Table 32: Performance Metrics for Model 33

3.34 Model 34 - Data splitting/balancing method 2 - Information Gain - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.9	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.2	0.3	0.2	0.1	0.1
Weighted Average	0.7	0.6	0.8	0.7	0.7	0.1	0.1

Table 33: Performance Metrics for Model 34

3.35 Model 35 - Data splitting/balancing method 2 - Boruta - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.9	0.8	0.7	0.9	0.8	0.1	0.1
Class Y	0.2	0.1	0.5	0.2	0.3	0.1	0.1
Weighted Average	0.6	0.5	0.6	0.6	0.6	0.1	0.1

Table 34: Performance Metrics for Model 35

3.36 Model 36 - Data splitting/balancing method 2 - RFE - gcvEarth

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.9	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.2	0.3	0.2	0.1	0.1
Weighted Average	0.7	0.6	0.8	0.7	0.7	0.1	0.1

Table 35: Performance Metrics for Model 36

3.37 Parameters of the Final Best Model

3.38 The Best Model: Model 10 - Data Splitting/Balancing Method 2 - Information Gain - Random Forest

The **Model 10** has been identified as the best model among all evaluated. It uses the *Information Gain* technique for feature selection and the *Random Forest (RF)* algorithm for classification. Below is a detailed description of the model and the reasons it is considered the best.

3.39 Model Description

The *Random Forest* algorithm is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the output classes of the individual trees. This approach is effective for handling large datasets with high dimensionality and is known for its ability to reduce overfitting, making it ideal for complex classification problems.

3.40 Performance Metrics

The performance of **Model 10** was evaluated using several key metrics, such as True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-measure, Matthews Correlation Coefficient (MCC), and Kappa. The results for each class, as well as the weighted averages, are presented below:

Class	TPR	FPR	Precision	Recall	F_measure	MCC	Kappa
Class N	0.8	0.7	0.9	0.8	0.8	0.1	0.1
Class Y	0.3	0.2	0.2	0.3	0.2	0.1	0.1
Weighted Average	0.8	0.7	0.8	0.8	0.7	0.1	0.1

Table 36: Performance Metrics for Model 10

3.41 Reasons for Selecting this Model

- **High Precision for the Majority Class:** The model shows high precision (0.9) for the majority class (Class N), which is crucial in scenarios where the majority class is significantly more frequent.
- **Balance between Precision and Recall:** The model achieves a good balance between precision and recall for the majority class, resulting in a high F-measure (0.8).
- **Overfitting Reduction:** The use of Random Forest helps reduce overfitting by averaging multiple decision trees, thus increasing the model's robustness.
- **Efficiency in Feature Selection:** The Information Gain technique efficiently selects relevant features, improving the model's overall performance.

In summary, **Model 10** stands out for its robust and balanced performance, making it the best model for this dataset and classification problem.

4 Discussion and Conclusion

The evaluation and comparison of different classification models on the given dataset revealed several important insights regarding their performance under various conditions. The best-performing model identified was a Random Forest (RF) model, which utilized the Information Gain method for feature selection and Data Splitting/Balancing Method 2.

4.1 Key Findings

- **High Precision for Majority Class:** The RF model demonstrated high precision (0.9) for the majority class (Class N). This is particularly important in scenarios with a significantly more frequent majority class.
- **Balanced Performance:** The model achieved an accuracy of 76.62%, sensitivity of 87.38%, and specificity of 24.81%. This indicates its capability to correctly classify both positive and negative instances.
- **Effective Handling of Class Imbalance:** Utilizing the Information Gain method for feature selection significantly improved the model's performance in detecting positive instances by addressing class imbalance.
- **Reduction in Overfitting:** The ensemble approach of Random Forest helped mitigate overfitting, enhancing the model's robustness and generalizability.

4.2 Challenges Faced

- **Class Imbalance:** Managing class imbalance was a critical challenge. The Information Gain method for feature selection and careful data splitting were essential in overcoming this issue.
- **Feature Selection:** Choosing the appropriate feature selection method was vital to ensure optimal model performance. Information Gain proved to be the most effective among the tested methods.

4.3 Recommendations

- **Hyperparameter Optimization:** Further tuning of hyperparameters for the RF model can be explored to enhance its performance.
- **Additional Techniques:** Testing more balancing techniques and feature selection methods could identify additional improvements.
- **Generalizability Testing:** Evaluating the model on different datasets will help assess its robustness and applicability to various contexts.

4.4 Conclusion

In conclusion, the Random Forest model, combined with Information Gain for feature selection and Data Splitting/Balancing Method 2, provided a robust and effective solution for the classification problem. The detailed analysis and comparison of various models ensured the selection of the best model for this task. The RF model excelled in maintaining a balanced performance and effectively handling class imbalance, making it the top choice for this dataset and classification problem.However, despite these extensive efforts, the final best model, did not fully meet the specified performance criteria.

5 Division of Work Between Team Members

- **David Chavez**:
  - Pre-processing and writing report
- **Finn Graham**:
  - Model training and evaluation