Abstract

This report investigates key risk factors that contribute to whether an individual is diagnosed with diabetes. Several classification methods were applied to identify which model best predicts diabetes diagnosis based on a variety of predictors. The models explored include logistic regression, linear discriminant analysis (LDA), naive Bayes, k-nearest neighbors (KNN), classification trees, random forests, and gradient boosting. Each method was evaluated based on its classification accuracy and testing error rates.

Initially, models were trained on a subset of the data and evaluated using testing data to compare performance. Among all models, the boosting model demonstrated the lowest testing error, suggesting it was the most accurate at classifying individuals as diabetic or non-diabetic when tested on unseen data. However, when cross validation techniques were applied to provide a more robust estimate of model performance, a classification tree and boosting yielded the lowest cross validated error rate. This indicates that while boosting performed best on the test set, a classification tree may offer better generalization when considering variability across different data partitions.

The analysis highlights the importance of model selection and validation when working with medical data, where accurate classification can be critical. The results suggest that although advanced methods like boosting can offer high accuracy, simpler models such as decision trees may perform more consistently when evaluated through cross-validation.

Introduction

Diabetes is a chronic disease that affects millions of individuals worldwide and poses serious health risks if left unmanaged. Early identification of individuals at risk of diabetes is critical for timely intervention and treatment. However, classifying whether an individual is diabetic based on various risk factors presents significant challenges, especially when dealing with large and complex datasets. The goal of this report is to analyze key factors that contribute to diabetes diagnosis and to apply various data mining and classification techniques to build models capable of accurately predicting diabetes outcomes.

A major challenge in this analysis lies in selecting appropriate classification methods that can handle potential issues such as class imbalance, variable correlations, and overfitting. Since medical datasets often contain overlapping features and noise, choosing a model that balances accuracy and generalizability is essential. To address these challenges, a variety of classification algorithms were implemented, including logistic regression, linear discriminant analysis (LDA), naive Bayes, k-nearest neighbors (KNN), classification trees,

random forests, and gradient boosting. These models were evaluated using both a testing dataset and cross-validation to ensure robust performance assessment.

Through this study, important insights were gained regarding the performance of different classifiers on the diabetes dataset. Notably, boosting provided the lowest testing error rate when evaluated on unseen data, while both boosting and a classification tree achieved the lowest error under cross validation, suggesting its potential for better generalization.

This report is organized as follows: First, the dataset and preprocessing steps are described. Next, the various classification methods used are outlined. Finally, a comparison and discussion of the models' performances are provided, followed by concluding remarks and lessons that were learned during this process.

Problem Statement/Data Sources

The goal of this study is to develop and compare classification models that can accurately predict whether an individual has been diagnosed with diabetes based on several health-related risk factors. Accurately identifying individuals at risk for diabetes is essential for early intervention and treatment, potentially improving health outcomes and reducing long-term healthcare costs. However, predicting diabetes status can be challenging due to overlapping characteristics between individuals with and without the disease, as well as potential noise in the data.

The data used for this analysis comes from the "diabetes_dataset.csv" file, a publicly available dataset on Kaggle compiled to study factors contributing to diabetes. The dataset contains 9,538 observations and 17 variables, including demographic, medical, and lifestyle-related attributes. The primary outcome variable is Outcome, a binary variable indicating whether an individual is diabetic (1) or not diabetic (0).

The variables included in the dataset are:

- **Age**: Age of the individual
- **Pregnancies**: Number of pregnancies an individual had
- **BMI**: Body Mass Index, a measure of body fat based on height and weight (kg/m^2)
- **Glucose**: Blood glucose concentration (mg/dL), a key diabetes indicator
- **BloodPressure**: Systolic blood pressure (mmHg), higher levels may indicate hypertension
- **HbA1c**: Hemoglobin A1c level (%), representing average blood sugar over months

- **LDL (Low-Density Lipoprotein)**: "Bad" cholesterol level (mg/dL)
- **HDL (High-Density Lipoprotein)**: "Good" cholesterol level (mg/dL)
- **Triglycerides**: Fat levels in the blood (mg/dL), high values increase diabetes risk
- **WaistCircumference**: Waist measurement (cm), an indicator of central obesity
- **HipCircumference**: Hip measurement (cm), used to calculate WHR
- **WHR (Waist-to-Hip Ratio)**: Waist circumference divided by hip circumference
- **FamilyHistory**: Indicates if the individual has a family history of diabetes (1 = Yes, 0 = No)
- **DietType**: Dietary habits (0 = Unbalanced, 1 = Balanced, 2 = Vegan/Vegetarian)
- **Hypertension**: Presence of high blood pressure (1 = Yes, 0 = No)
- **MedicationUse**: Indicates if the individual is taking medication (1 = Yes, 0 = No)
- **Outcome**: Diabetes diagnosis result (1 = Diabetes, 0 = No Diabetes)

A brief exploratory data analysis (EDA), using the DataExplorer package in R (refer to appendix 1), revealed that some variables are continuous (e.g., BMI, HbA1c), while others are categorical (e.g., Hypertension). There is also a moderate correlation between certain variables, such as MedicationUse and Age, which may introduce challenges like multicollinearity when building predictive models. When looking at the correlation matrix, it was found that one of the variables, FamilyHistory, was very positively correlated with the response variable Outcome (refer to appendix 2). Due to this large correlation value, most of the models weighted this too heavily and produced some error values of 0 which did not make sense. So, for these models the FamilyHistory column was removed, and all the other columns remained the same.

Understanding and addressing these data characteristics are essential steps toward developing accurate and reliable classification models. This study aims to explore various classification approaches to overcome these challenges and identify the most effective model for predicting diabetes outcomes.

Proposed Methodology

To predict whether an individual has diabetes, I explored and compared several different classification methods. Each method was chosen for its unique approach to learning

patterns from the data, and I used cross validation on the training set to tune the models and avoid overfitting. The kind of cross validation used was k-folds cross validation. This kind of cross validation splits the training data set into x number of buckets, in this case it was 10 buckets, trains the model on 9 of them and then validates that model on the remaining 1. This process is then repeated 9 times, with each bucket being used one time as the validator (refer to appendix 3). Below, I describe the reasoning behind selecting and tuning each model.

Logistic Regression with Stepwise AIC was used as one of the models because it is a well-known method for predicting binary outcomes, such as determining whether an individual has diabetes. To select the best combination of variables, I applied stepwise selection based on the Akaike Information Criterion (AIC). AIC helps balance model fit and simplicity by penalizing unnecessary variables, ensuring that the final model is both effective and not overly complex. After building the model using stepwise AIC, I further validated the logistic regression model's performance through cross validation. Cross validation allowed me to assess how well the model would generalize to new data and helped prevent overfitting. Unlike the stepwise AIC model, the cross validated model kept all factors in the equation. By building a logistic regression model using both stepwise AIC and cross validation, I aimed to create a model that was both accurate and reliable when applied to unseen cases.

Linear Discriminant Analysis (LDA) was used as another linear method for classification. LDA works by finding a linear combination of features that best separates the two groups — individuals with and without diabetes. Like logistic regression, LDA is easy to interpret and can be effective when the relationship between variables and the outcome is mostly linear. To ensure the model's reliability and prevent overfitting, I applied cross validation on the training data. By cross validating LDA, I gained a better understanding of its expected accuracy and robustness when applied to new individuals outside of the training set.

Naive Bayes was selected because it is a simple yet powerful method that relies on probabilities. It assumes that all predictor variables are independent, which may not hold in real life but can still produce strong results. Cross validation helped ensure that Naive Bayes was properly tuned to work well on this dataset.

K-Nearest Neighbors (KNN) is a non-parametric method (does not assume a distribution of the data) that classifies an individual based on the majority vote of their "neighbors" — other individuals with similar characteristics. I tested different values of k (from 1 to 19 – odd numbers only) using cross validation to find the best number of neighbors for the most accurate predictions. Odd numbers were used so that there could be no ties in number of neighbors – for example if a new data point is introduced and if k = 4, if the 4 nearest points

were 2 data points with diabetes and 2 without, then the model would pretty much be guessing what to classify to new data point as (refer to appendix 4).

Decision Trees were included because they are easy to understand and interpret. A decision tree works like a flowchart, splitting the data based on variables that best separate individuals with and without diabetes. I used cross validation to build and prune the tree — a technique to remove unnecessary splits and prevent overfitting.

Random Forest is a method that builds many decision trees and averages their results to make predictions. This helps reduce the risk of overfitting that a single tree might have. Cross validation was used to ensure that the random forest model used the right number of trees and was not overly complex.

Boosting is another technique that builds decision trees sequentially, with each tree focusing on correcting the errors made by previous trees. For boosting, I chose originally chose 5,000 trees, an interaction rate of 4 for each tree, and a learning rate of 0.01. The optimal number of iterations to run this was found to be 351 times, which was found from cross validation and was then used to find the error values.

Through this variety of models, I aimed to find the approach that would best predict diabetes, and cross validation allowed me to tune each method carefully using only the training data.

Analysis and Results

Below is a summary of the performance of the various models – they were evaluated in terms of training error, testing error, and cross validation testing error (yellow means smallest error for that column and red means largest):

Model	Training Error	Testing Error	Cross Validation Testing Error
Boosting	0.2790593	0.2907058	0.2910552
Logistic Regression	0.3169563	0.3301887	0.3242488
LDA	0.3117136	0.3232006	0.3232006
Naïve Bayes	0.3076693	0.3158630	0.3158630
Decision Tree	0.2787597	0.2910552	0.2910552
Random Forest	0.0000000	0.2966457	0.2935010
KNN – k = 1	0.0000000	0.4371069	0.4153646
KNN – k = 3	0.2032654	0.3972746	0.3827164
KNN – k = 5	0.2532954	0.3752621	0.3650383
KNN – k = 7	0.2672259	0.3626834	0.3521611
KNN – k = 9	0.2786099	0.3508036	0.3406281
KNN – k = 11	0.2875974	0.3431167	0.3323861

KNN – k = 13	0.2901438	0.3298393	0.3217456
KNN – k = 15	0.2886459	0.3305381	0.3221991
KNN – k = 17	0.2899940	0.3284416	0.3205500
KNN – k = 19	0.2925404	0.3266946	0.3181560

Boosting, using the gbm package in R, performed reasonably well, with a training error of 0.2786, a testing error of 0.2907 (the lowest), and a cross-validation error of 0.2911 (tied for lowest). It showed good performance on the training, testing, and cross validated data. Boosting found that the Glucose was by far the most important variable in classifying if someone has diabetes or not (refer to appendix 5).

Logistic regression, using the step and glm functions in R, had a higher testing error (0.3302) compared to other models, suggesting it did not perform as well on unseen data. Its cross-validation error (0.3242) was relatively close to the testing error, which may indicate that logistic regression did not overfit the training data significantly. The logistic regression model found Glucose to be the most significant risk factor in classification, with a couple others being close to a significance value of 0.05 (refer to appendix 6).

LDA, using the MASS package in R, performed similarly to logistic regression with a testing error of 0.3232 and a cross-validation error that was identical (0.3232). It didn't show much overfitting, but its performance was not as strong as some other models like boosting or a decision tree. LDA indicated that Hypertension is most strongly associated with being in the 1 (diabetic) class and WHR is most strongly associated with being in the 0 (not diabetic) class (refer to appendix 7).

Naive Bayes, using the e1071 package in R, had similar results to LDA with a testing error of 0.3159 and cross validation error of 0.3159. This consistency indicates a relatively stable performance, although it wasn't the best-performing model.

The decision tree model, using the rpart and rpart.plot packages in R, had a training error of 0.2788 and a testing error of 0.2911, similar to boosting. The cross validation error was the same as the testing error and was tied for the lowest cross validation error across all models, showing a good generalization to unseen data. The decision tree for finding both the normal testing error and cross validation testing error resulted in the same split – to split the data into 2 different sections based on if an individual's Glucose level was less than 140 or not (refer to appendix 8).

Random Forest, using the randomForest package in R, showed a perfect training error of 0.0000, which may indicate overfitting. However, it performed quite well on the testing data

with an error of 0.2966 and a cross-validation error of 0.2935. The random forest model appeared to generalize well to unseen data despite the perfect training accuracy.

KNN, using the class package in R, with k = 1 had a training error of 0.0000, suggesting it perfectly fit the training data. However, the testing error was much higher at 0.4371, indicating severe overfitting. The cross validation error (0.4154) confirmed this.

As k increased, the testing and cross validation errors decreased, indicating that higher values of k help generalize the model better. The model with k = 3 performed reasonably well with a testing error of 0.3973 and a cross validation error of 0.3827. As k increased, the errors continued to decrease, with the best performance at k = 19 (testing error of 0.3267, cross validation error of 0.3182).

All cross validations on these models were done using the caret package in R which is used for training, cross validation, and model tuning.

Conclusions

In this assignment, several data mining methods were applied and compared to classify individuals on the diabetes data set. The goal was to assess the performance of various linear and nonlinear classifiers using training, testing, and cross validation error rates as metrics.

Among all models tested, a decision tree achieved the one of the lowest testing errors and tied for the lowest cross validation error, indicating its ability to generalize well on unseen data. For this data set specifically, a decision tree would be a good model to use because it allows for easy understanding and visualization (refer to appendix 8). The trees show that if an individual's Glucose level is less than 140 for both tests then they will likely be classified as not diabetic. This would allow for results to be explained easily to patients. An issue that arises when using a decision tree is that it is difficult for a single tree to capture all the complexities in the data, which is where more complex models like random forest and boosting come into play.

Both random forest and boosting performed similar to the single decision tree in terms of testing and cross validation errors, so picking any of them will likely lead to similar results. For this data set I would lean toward using boosting rather than a random forest due to its explainability. Although the random forest performed about the same on the testing and cross validated data as the boosting the issue that comes with random forest is that it is very hard to explain its results. A random forest cannot produce a tree that shows where it says to split the data set up. Due to it being an aggregate of many trees, it is impossible to do so. On the other hand, boosting can inform a user on which variables the model focuses on as it learns (refer to appendix 5). For this data set, I would start out with a single decision

tree to better understand the data and then use boosting for the performance measure due to it being a more complex model which produces almost the same results and the simple one.

To further this decision, in the future I would find the variance values (how much variability in a model's output is in mulitple runs of it) of each model and compare them. This would lead to a sounder conclusion. For example, the decision tree produces the lowest testing and cross validation errors, but if it has a high variance then it cannot be as trusted as the error values make it seem due to it being not consistent in its outputs.

With the very correlated to the response variable FamilyHistory being taken out of the data set for model building, most models deemed Glucose level to be the most significant factor leading to classifying if someone is diabetic or not.

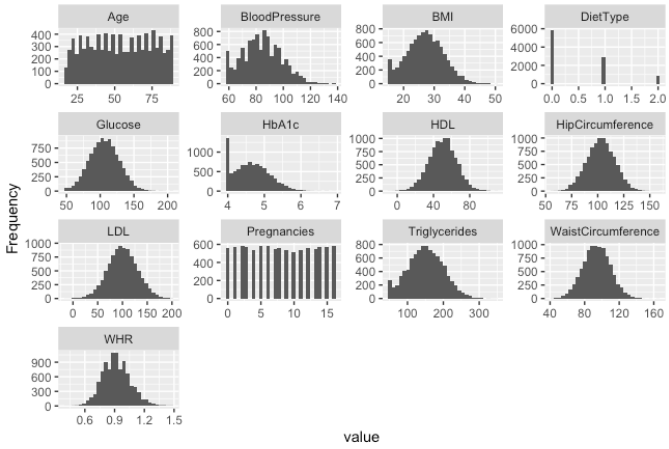
Overall, this comparative study highlights that while methods like random forest and boosting provide low error**,** simpler models like decision trees strike a useful balance between performance and explainability — a crucial factor when applying data mining techniques to sensitive domains like healthcare.

Lessons Learned:

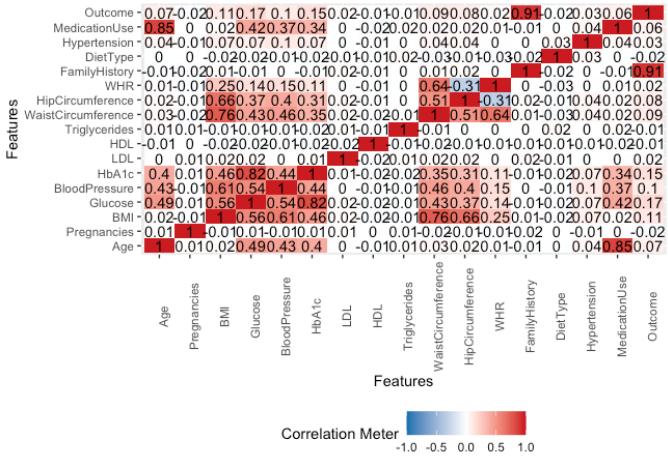
- Data Preprocessing is Critical: Ensuring clean, well-prepared data was a vital first step. Handling missing values, proper feature scaling (especially for KNN), and understanding feature relationships were essential for better model performance.
- Model Interpretability vs. Accuracy: I learned that high-performing models (like Random Forest) are not always the most interpretable. This trade-off is crucial to consider, especially in fields like healthcare where decisions need to be explained.
- Tuning Hyperparameters Makes a Difference: For methods like Boosting and KNN, hyperparameter tuning significantly affected performance, emphasizing the need for careful model selection and tuning rather than relying on defaults.
- This project deepened my appreciation for the balance between theoretical knowledge and practical implementation. While some methods are straightforward in theory, applying them to real-world data poses unique challenges.
- I found visualizations (e.g., for Decision Trees) very useful to interpret and communicate results to non-technical audiences.

Appendix

1. Histogram for each variable in the data set whose value is not binary (0,1). All continuous variables are normally distributed for the most part.

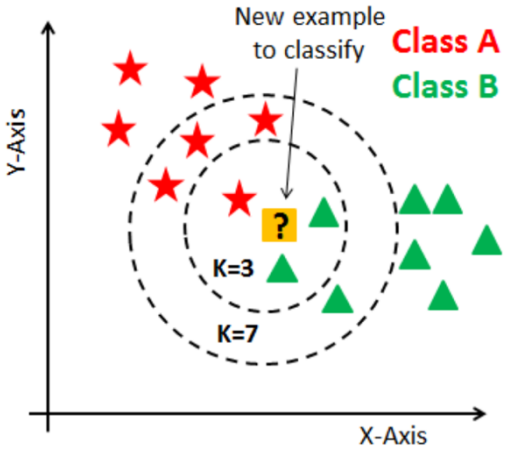


2. Correlation matrix of the data set. FamilyHistory was taken out due to its strong positive correlation with the response variable. MedicaitonUse and Age, Gluscoe and HbA1c, and WaistCircumference and BMI are all farily positively correlated.

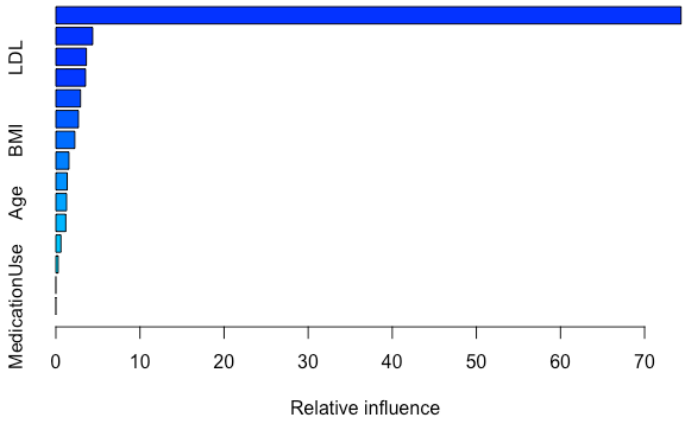


3. Example of how k-folds cross validation works.

4. KNN example as an image – if k = 3 then the point will be Class B, if k = 7 then the point will be Class A.



5. Boosting results.



Variable	Relative Influence
Glucose	75.04580544
LDL	3.73617714
HipCircumference	3.50220291
Triglycerides	3.46005046
HDL	2.81535777
WaistCircumference	2.48710552
BMI	2.25328939
WHR	1.87135156
BloodPressure	1.52373451
Age	1.43003182
Pregnancies	1.09479114
HbA1c	0.59814971
DietType	0.15814392
MedicationUse	0.02380872
Hypertension	0.00000000

6. Using stepwise regression, it was found that the variables of Age, Pregnancies, Glucose, HbA1c, LDL, WaistCircumference, WHR, and Hypertension were deemed best to build a linear regression model. Below are the coefficients of each variable and their significance.

Variable	Estimate	Significance (Pr(>|t|))
Intercept	-0.1209558	0.1115
Age	-0.0004858	0.1378
Pregnancies	-0.0022174	0.0564

Glucose	0.0027554	7.16e-08
HbA1c	0.0407979	0.0513
LDL	0.0001910	0.0938
WaistCircumference	0.0008089	0.1400
WHR	-0.0925248	0.0888
Hypertension	0.3493717	0.0680

Using cross validation, all variables were kept, and it resulted in a slightly lower error value. Below are the coefficients of each variable and their significance.

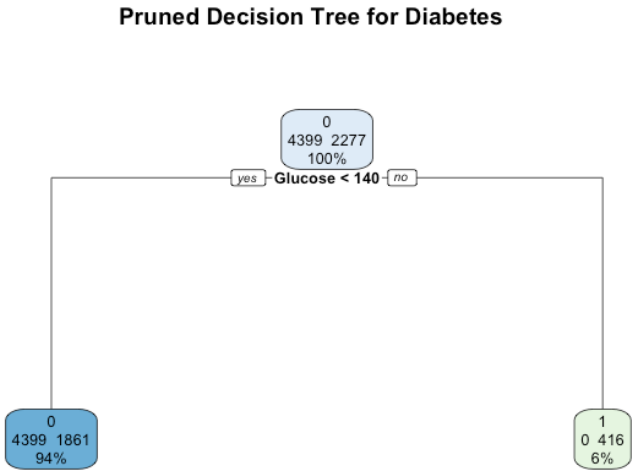
Variable	Estimate	Significance (Pr(>|z|))
Intercept	-2.0626069	0.1172
Age	-0.0035032	0.1931
Pregnancies	-0.0101772	0.0559
BMI	-0.0100272	0.2874
Glucose	0.0136145	3.58e-08
BloodPressure	0.0014586	0.5943
HbA1c	0.1631843	0.0861
LDL	0.0014588	0.0963
HDL	-0.0009880	0.5658
Triglycerides	-0.0007229	0.1809
WaistCircumference	0.0111954	0.4019
HipCircumference	-0.0047869	0.7011
WHR	-1.0425412	0.4339
DietType	-0.0101979	0.7956
Hypertension	1.7508358	0.1170
MedicationUse	0.0271100	0.7906

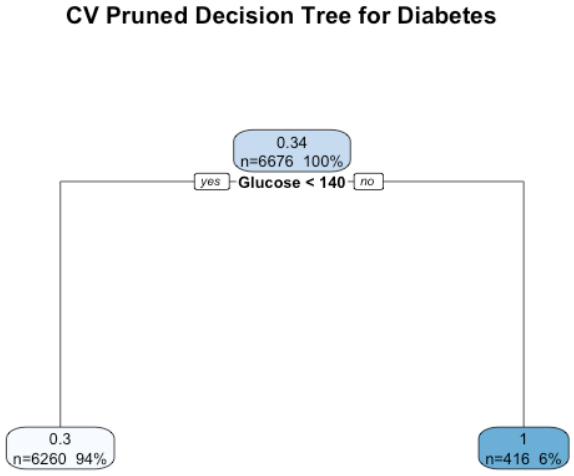
7. LDA results.

Variable	Coefficient
Age	-0.009816188
Pregnancies	-0.027828587
BMI	-0.027642003
Glucose	0.036544805
BloodPressure	0.004218621
HbA1c	0.501585903
LDL	0.003996694
HDL	-0.002721475
Triglycerides	-0.001982542
WaistCircumference	0.031426114

Hipcircumference	-0.013667189
WHR	-2.896506376
DietType	-0.025825328
Hypertension	4.467119164
MedicationUse	0.084013091

8. Decision trees for normal model and cross validated model.





9. Full R code.

```
### (A). Data Preparation

## Read Data

spam <- read.table(file= "diabetes_dataset.csv", sep = ",", header = TRUE)


dim(spam)   # 9538 17



#EDA

library(DataExplorer)

plot_histogram(spam)

plot_correlation(spam)


## Split to training and testing subset

set.seed(123)

spam$FamilyHistory <- NULL

flag <- sort(sample(9538,2862, replace = FALSE))

spamtrain <- spam[-flag,]

spamtest <- spam[flag,]

## Extra the true response value for training and testing data

y1    <- spamtrain$Outcome;

y2    <- spamtest$Outcome;



## (B) Boosting

## You need to first install this R package before using it
```

```
library(gbm)

#
gbm.spam1 <- gbm(Outcome ~ .,data=spamtrain,
        distribution = 'bernoulli',
        n.trees = 5000,
        shrinkage = 0.01,
        interaction.depth = 4,
        cv.folds = 10)

## Model Inspection
## Find the estimated optimal number of iterations
perf_gbm1 = gbm.perf(gbm.spam1, method="cv")
perf_gbm1

pred1gbm <- predict(gbm.spam1, newdata = spamtrain, n.trees = perf_gbm1, type = "response")

## summary model
## Which variances are important
summary(gbm.spam1)



## Make Prediction
## use "predict" to find the training or testing error

## Training error
y1hat <- ifelse(pred1gbm < 0.5, 0, 1)
sum(y1hat != y1)/length(y1)
```

```
## Testing Error
y2hat <- ifelse(predict(gbm.spam1, newdata = spamtest, n.trees=perf_gbm1, type="response") < 0.5, 0, 1)
mean(y2hat != y2)


## A comparison with other methods
## Testing errors of several algorithms on the spam dataset:
#A. Logistic regression: 0.1022135
modA <- step(glm(Outcome ~ ., data = spamtrain));

summary(modA)
# Training Error for Logistic Regression
y1hatA <- ifelse(predict(modA, spamtrain[,-16], type="response") < 0.5, 0, 1)
sum(y1hatA != y1) / length(y1)


y2hatA <- ifelse(predict(modA, spamtest[,-16], type="response" ) < 0.5, 0, 1)
sum(y2hatA != y2)/length(y2)

#B.Linear Discriminant Analysis : 0.1041667
library(MASS)
modB <- lda(spamtrain[,1:15], spamtrain[,16])
summary(modB)
print(modB)
# Training Error for LDA
y1hatB <- predict(modB, spamtrain[, -16])$class
mean(y1hatB != y1)
```

```
y2hatB <- predict(modB, spamtest[,-16])$class
mean( y2hatB  != y2)

## C. Naive Bayes (with full X). Testing error = 0.313151
library(e1071)
modC <- naiveBayes(as.factor(Outcome) ~. , data = spamtrain)
print(modC)

# Training Error for Naive Bayes
y1hatC <- predict(modC, newdata = spamtrain)
mean(y1hatC != y1)

y2hatC <- predict(modC, newdata = spamtest)
mean( y2hatC != y2)

## D. KNN
library(class)
# Extract features and target variable from training data
train_features <- spamtrain[, -which(names(spamtrain) == "Outcome")]
train_target <- spamtrain$Outcome

# Extract features from test data
# Separate features and target in the test data
test_features <- spamtest[, -which(names(spamtest) == "Outcome")]  # Remove target from test data
test_target <- spamtest$Outcome  # Target variable in the test data
```

```
# Initialize a vector to store error rates for different k values
k_values <- 1:20

error_rates_train <- numeric(length(k_values))
error_rates <- numeric(length(k_values))

for (k in k_values) {
  # Apply KNN model with current k
  predictions_train <- knn(train = train_features, test = train_features, cl = train_target, k = k)

  # Calculate the training error rate for the current k
  error_rate_train <- mean(predictions_train != train_target)

  # Store the training error rate for the current k
  error_rates_train[k] <- error_rate_train
}

# Display the training error rates for each k value
data.frame(k = k_values, training_error_rate = error_rates_train)



# Loop over each k value and calculate the error rate
for (k in k_values) {
  # Apply KNN model with current k
  predictions <- knn(train = train_features, test = test_features, cl = train_target, k = k)

  # Calculate the error rate for the current k
  error_rate <- mean(predictions != test_target)
```

```
  # Store the error rate for the current k
  error_rates[k] <- error_rate
}

# Display the error rates for each k value
data.frame(k = k_values, error_rate = error_rates)

#E: a single Tree: 0.1015625
library(rpart)
library(rpart.plot)

modE0 <- rpart(Outcome ~ .,data=spamtrain, method="class",
               parms=list(split="gini"))
opt <- which.min(modE0$cptable[, "xerror"]);
cp1 <- modE0$cptable[opt, "CP"];
modE <- prune(modE0,cp=cp1);
rpart.plot(modE, type=2, extra=101, fallen.leaves=TRUE, cex=0.8, main="Pruned Decision Tree for
Diabetes")
# Training Error for Decision Tree
y1hatE <- predict(modE, spamtrain[,-16], type="class")
mean(y1hatE != y1)

y2hatE <-  predict(modE, spamtest[,-16],type="class")
mean(y2hatE != y2)

#F: Random Forest: 0.04166667
library(randomForest)
modF <- randomForest(as.factor(Outcome) ~., data=spamtrain,
```

```
            importance=TRUE)

y1hatF <- predict(modF, spamtrain, type='class')
mean(y1hatF != y1)

y2hatF = predict(modF, spamtest, type='class')
mean(y2hatF != y2)




library(caret)
# (B) Set up cross-validation control
train_control <- trainControl(method = "cv", number = 10)

# (C) Models with Cross-validation

# 1. Boosting
spamtrain$Outcome <- as.factor(spamtrain$Outcome)
spamtest$Outcome <- as.factor(spamtest$Outcome)

tune_grid <- expand.grid(
  n.trees = 351,                # Number of trees to use (same as original)
  shrinkage = 0.01,             # Learning rate (same as original)
  interaction.depth = 4,   # Tree depth (same as original)
  n.minobsinnode = 10
)

# Boosting model with cross-validation
```

```
gbm_model <- train(Outcome ~ ., data = spamtrain,
                   method = "gbm",
                   trControl = train_control,
                   verbose = FALSE,
                   distribution = "bernoulli",   # Bernoulli for binary classification
                   tuneGrid = tune_grid)        # Use the defined tuning grid

# Optionally, print a summary of the model (best performance, optimal parameters)
summary(gbm_model)

# 2. Logistic Regression
logit_model <- train(Outcome ~ ., data = spamtrain,
                     method = "glm",
                     trControl = train_control)
summary(logit_model)

# 3. Linear Discriminant Analysis (LDA)
lda_model <- train(Outcome ~ ., data = spamtrain,
                   method = "lda",
                   trControl = train_control)
summary(lda_model)

# 4. Naive Bayes
nb_model <- train(Outcome ~ ., data = spamtrain,
                  method = "naive_bayes",
                  trControl = train_control)
summary(nb_model)
```

```
# 5. KNN

# Define the range of k values to test

k_values <- 1:20


# Initialize a vector to store error rates for each k

cv_error_rates <- numeric(length(k_values))


# Loop over each k value to perform cross-validation

for (k in k_values) {

 # Train the KNN model for each k with 10-fold cross-validation

 knn_model <- train(Outcome ~ .,

                    data = spamtrain,

                    method = "knn",

                    tuneGrid = data.frame(k = k),

                    trControl = train_control)


 # Extract the cross-validated error rate for the current k

 cv_error_rates[k] <- min(knn_model$results$Accuracy) # Accuracy is used here, or you can use
ErrorRate = 1 - Accuracy


}


# Display the cross-validation errors for each k

cv_error_rates_df <- data.frame(k = k_values, cv_error_rate = 1 - cv_error_rates)

print(cv_error_rates_df)


# 6. Decision Tree (rpart)


# Train the decision tree model
```

```
# (2) Train the Model with Cross-Validation
# We're specifying the method as "rpart" (decision tree) and the formula `Outcome ~ .`
# We'll use the trainControl for cross-validation and search over a range of cp values.
tune_grid <- expand.grid(cp = seq(0.001, 0.1, by = 0.001))  # Search for a good CP value

tree_model_cv <- train(Outcome ~ ., data = spamtrain,
                       method = "rpart",
                       trControl = train_control,   # Cross-validation control
                       tuneGrid = tune_grid,       # CP tuning grid
                       parms = list(split = "gini"))  # Split criterion: "gini"

# (3) Inspect the Cross-Validation Results
# Best complexity parameter (cp) and model performance
print(tree_model_cv)

# (4) Plot the model's performance
# Visualize the effect of different complexity parameters on the performance
plot(tree_model_cv)

# (5) Use the best model from cross-validation for predictions
best_cp <- tree_model_cv$bestTune$cp  # Get the best CP value from cross-validation
best_tree_model <- tree_model_cv$finalModel  # The final decision tree model based on best CP

rpart.plot(best_tree_model, type = 2, extra = 101, fallen.leaves = TRUE, cex = 0.8, main = "CV Pruned
Decision Tree for Diabetes")

# 7. Random Forest
```

```
rf_model <- train(Outcome ~ ., data = spamtrain,
                  method = "rf",
                  trControl = train_control)




# (E) Final Model Evaluation on Test Set
# Use the best-performing model (based on cross-validation results) to make predictions on the test set.

rf_predictions <- predict(rf_model, newdata = spamtest)
rf_testing_error <- mean(rf_predictions != y2)
print(paste("Random Forest Testing Error:", rf_testing_error))

# Similarly, for other models:
# For Boosting (gbm)
gbm_predictions <- predict(gbm_model, newdata = spamtest)
gbm_testing_error <- mean(gbm_predictions != y2)
print(paste("Boosting Testing Error:", gbm_testing_error))

# For Logistic Regression (logit_model)
logit_predictions <- predict(logit_model, newdata = spamtest[,-16])
logit_testing_error <- mean(logit_predictions != y2)
print(paste("Logistic Regression Testing Error:", logit_testing_error))

# For Linear Discriminant Analysis (lda_model)
lda_predictions <- predict(lda_model, newdata = spamtest[,-16])
lda_testing_error <- mean(lda_predictions != y2)
```

```
print(paste("LDA Testing Error:", lda_testing_error))


# For Naive Bayes (nb_model)
nb_predictions <- predict(nb_model, newdata = spamtest[,-16])
nb_testing_error <- mean(nb_predictions != y2)
print(paste("Naive Bayes Testing Error:", nb_testing_error))

# For KNN (knn_model)
knn_predictions <- predict(knn_model, newdata = spamtest[,-16])
knn_testing_error <- mean(knn_predictions != y2)
print(paste("KNN Testing Error:", knn_testing_error))



y2hatE <- predict(best_tree_model, spamtest[,-16], type = "class")
testing_error <- mean(y2hatE != y2)  # Calculate testing error
print(paste("Testing Error Rate:", testing_error))
```