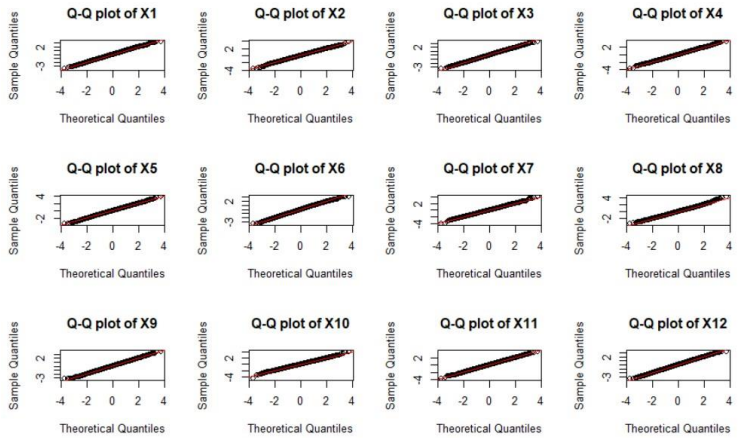
Task 1: Prediction Challenge
Students predict three response variables (Y1, Y2, Y3) for 100 individuals using 50 known predictors (X1–X50) provided in a test dataset (test.csv). The training dataset (train.csv) consists of 5,000 individuals. The report (maximum 5 pages) must detail the methods, including R code and interpretations.
Task 2: Classification
Students develop a classification model using a dataset assigned from the UCI Machine Learning Repository. The report should include a summary, introduction, method justification, results, and conclusions. The appendix contains R code.Data for both tasks is available via LMS and UCI.

1. Introduction

The objective of this analysis is to predict three unknown response variables (Y1, Y2, Y3) for 100 individuals using their known predictors (X1 to X50). The predictions are based on models developed using a training dataset of 5,000 individuals. This report provides a detailed description of the methods chosen, the rationale, analysis and results.

2. Data exploration and preparation



.

Figure 1 Q-Q plots were generated to assess the normality of the predictors (showing predictor X3 to X12 here), helping to understand how well the data conformed to a normal distribution. The Q-Q plots illustrate the distribution of the predictors compared to a theoretical normal distribution. The points lie approximately along the line, indicating that the predictors are normally distributed.

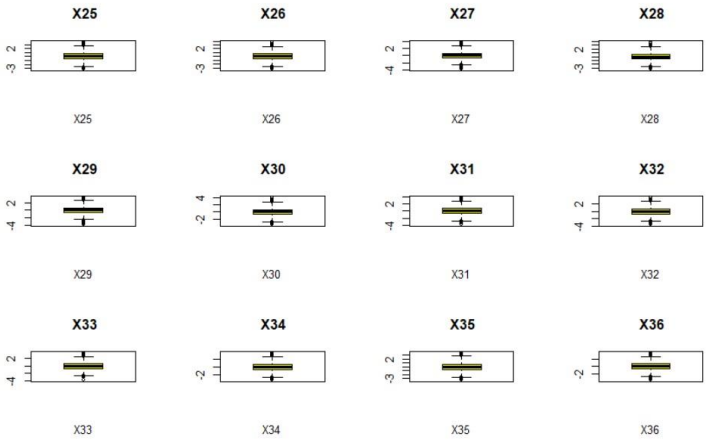


Figure 2 Box plots (feature X25 to X36 showed above) were created to visualize the spread and detect outliers as points outside the whiskers.

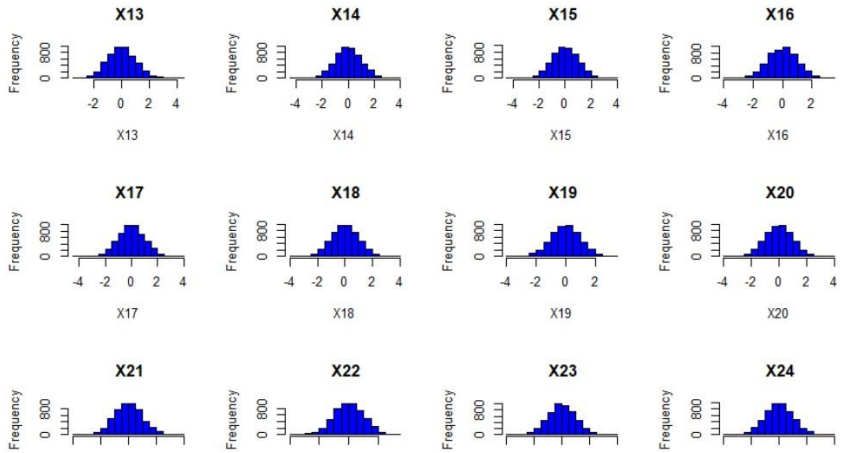


Figure 3 Histograms were plotted for each predictor to visualize their distributions, which helped identify any skewness or anomalies in the data (predictor X13 to X24 showed above).

Normalization was performed to ensure that each feature contributed equally to the analysis. This step involved centering, scaling, and applying BoxCox (to eradicate minor skewness if any) transformations to the data. Normalization is crucial for improving the convergence and performance of the models. The correlation matrix provides a visual representation of the relationships between predictors. It showed no strongly correlated variables (Fig. 4). To make the data more robust to outliers, a spatial sign transformation was applied, normalizing the data by scaling each observation to unit length. Principal Component Analysis (PCA) was then performed to reduce the dimensionality of the data while retaining 95% of the variance. This step helped in simplifying the models and improving their performance.

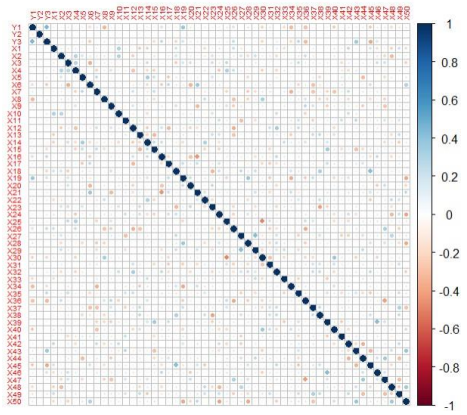


Figure 4 The correlation matrix provides a visual representation of the relationships between predictors. Some correlations between certain predictors can be seen but it is not very strong.

3. Model Assessment and Selection

We decided to focus on non-parametric approaches due to their robustness when assumptions are upheld. For example, Naïve Bayes assumes normality but is robust when assumption is not held. Parametric approach such as LDA, QDA's performance degrades drastically when assumption is not held. The methods compared are described below as per variable they were best suited to predict.

For Y1 and Y2

k-Nearest Neighbors (k-NN) Regression is a non-parametric method that predicts the response variable based on the k-nearest neighbours in the feature space. This method was chosen due to its simplicity and effectiveness in capturing local patterns in the data. Different values of k (5, 10, 15)

were tested to identify the optimal number of neighbours for prediction. The k-NN method is particularly useful for data with complex, non-linear relationships, as it does not assume any specific form for the underlying data distribution. Nadaraya-Watson Regression is a kernel smoothing technique that estimates the conditional mean of the response variable. This method was chosen for its ability to handle non-linear relationships and smooth out noise in the data. A normal kernel with a bandwidth of 0.2 was used to perform the smoothing. The choice of bandwidth is crucial as it determines the level of smoothing, with smaller bandwidths capturing more local variations and larger bandwidths providing smoother estimates. Local Polynomial Regression extends the idea of kernel smoothing by fitting a polynomial function locally to the data. This method was chosen for its flexibility in modeling complex, non-linear relationships while maintaining a smooth fit. A polynomial of degree 2 was used with a normal kernel and a bandwidth of 0.2. This approach allows for capturing local trends and patterns in the data more accurately than global polynomial fitting methods.

Figure 5 MSE for various models k-NN (5,10,15), Nadaraya-Watson and Local polynomial. It shows that k-NN = 5 has least MSE in Y1 and Y2.

The performance of each model was evaluated using Mean Squared Error (MSE), which measures the average squared difference between the predicted and actual values, providing a metric for the accuracy of the models (Fig. 5). Lower MSE values indicate better model performance. The calculated MSEs for Y1 and Y2 showed that the lowest MSE was k-NN (5) which makes it the best suited for predicting both variables. The MSE for Y1 shows that it consistently harder to predict accurately than Y2.

For Y3

We tested k-NN and found the accuracy much lower than desired. Nadaraya-Watson (bandwidth 0.1) had an accuracy of 1 whereas Naïve Bayes was a close match at 0.998. Naive Bayes is good with high dimension and Nadaraya Watson is computationally intensive. We think Nadaraya-Watson is a better model since Naive Bayes assumes independence between features which may not hold true in this case.

```
$best_knn
   k Accuracy  Kappa AccuracySD    KappaSD
1 13   0.6404 0.2808 0.02963181 0.05926363

$best_nw
   bw Accuracy
1 0.1        1

$best_nb
  fL usekernel adjust Accuracy  Kappa   AccuracySD     KappaSD
1  0      TRUE      1   0.9998 0.9996 0.0006324555 0.001264911
```

Figure 6 The best NW model with a bandwidth of 0.1 achieved perfect accuracy (1.0). This highlights the effectiveness of this model at this specific bandwidth for the given data.

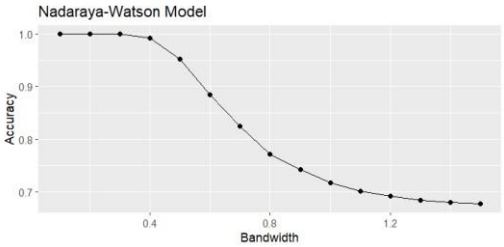


Figure 7 The plot shows the accuracy of the Nadaraya-Watson model as a function of the bandwidth parameter. The accuracy is highest at a bandwidth of approximately 0.1, achieving near-perfect accuracy. As the bandwidth increases beyond 0.1, the accuracy steadily decreases.

```
library(KernSmooth)
library(caret) library(FNN)
library(corrplot)
train_data <- read.csv('data-assignment2/train.csv')
# Separate predictors and responses x.comb <-
train_data[, 4:53] y1 <- train_data[, 1]  # Response
variable Y1 y2 <- train_data[, 2]  # Response
variable Y2
# Normalize the data
preProc_norm <- preProcess(x.comb, method = c("center", "scale",
"BoxCox")) x.comb <- predict(preProc_norm, x.comb) par(mfrow=c(3, 4)) #
Adjust the layout if necessary for (col in names(x.comb)) {
  hist(scale(x.comb[[col]]), main=paste("", col), xlab=col, col="blue")
}
# Plot box plots for each feature in x.comb
par(mfrow=c(3, 4)) # Adjust this if you have more/less columns for
(col in names(x.comb)) {
  boxplot(x.comb[[col]], main=paste("", col), xlab=col, col="yellow")
}
# Plot Q-Q plots for each feature in x.comb
par(mfrow=c(3, 4)) # Adjust this if you have more/less columns
for (col in names(x.comb)) {   if (is.numeric(x.comb[[col]])) {
    qqnorm(x.comb[[col]], main=paste("Q-Q plot of", col))
qqline(x.comb[[col]], col = "red")
  }
}
# Apply PCA
preProc_pca <- preProcess(x.comb, method = "pca", thresh =
0.95) x.comb <- predict(preProc_pca, x.comb) # Plot correlation
matrix for x.comb
corrplot(cor(train_data), method = "circle", tl.cex = 0.5) #
Define a function to perform the analysis and return MSEs
perform_analysis <- function(x, y) {x0 <- matrix(x, ncol = 1) # Predictors as a matrix


  # KNN Regression
  knn.fit5 <- knn.reg(train = x0, test = x0, y = y, k = 5)  # Using the same data for simplicity
knn.fit10 <- knn.reg(train = x0, test = x0, y = y, k = 10)   knn.fit15 <- knn.reg(train = x0,
test = x0, y = y, k = 15)


  # Nadaraya-Watson
  nw.fit <- ksmooth(x, y, kernel = "normal", bandwidth = 0.2, x.points = x0)


  # Local Polynomial Regression
  reg.poly <- locpoly(x, y, degree = 2, kernel = "normal", bandwidth = 0.2, gridsize = 100)


  # Calculate MSEs
  mse <- c(
    mse.5nn = mean((y - knn.fit5$pred)^2),
mse.10nn = mean((y - knn.fit10$pred)^2),
mse.15nn = mean((y - knn.fit15$pred)^2),
mse.nw = mean((y - nw.fit$y)^2),     mse.poly
= mean((y - reg.poly$y)^2)
  )


  return(mse)
}
# Perform analysis for Y1 and Y2 mse_y1 <-
perform_analysis(x.comb[, 1], y1) mse_y2 <-
perform_analysis(x.comb[, 1], y2)
# Identify the model with the lowest MSE for Y1 and Y2 best_model_y1
<- names(mse_y1)[which.min(mse_y1)] best_model_y2 <-
names(mse_y2)[which.min(mse_y2)]
# Print the best models cat("Best model for
Y1:", best_model_y1, "\n") cat("Best model for
Y2:", best_model_y2, "\n")
```

```r
library(ggplot2)

# Assume mse_y1 and mse_y2 contain the MSE values for each model as calculated earlier
mse_y1 <- c(mse.5nn = 0.03722341, mse.10nn = 0.05025473, mse.15nn = 0.07138480, mse.nw = 0.04393077, mse.poly = 0.10514709)
mse_y2 <- c(mse.5nn = 0.04122341, mse.10nn = 0.05325473, mse.15nn = 0.07338480, mse.nw = 0.04893077, mse.poly = 0.10814709)

# Convert MSEs to a data frame for plotting mse_data_y1 <- data.frame(Model = names(mse_y1), MSE = mse_y1, Response = "Y1") mse_data_y2 <- data.frame(Model = names(mse_y2), MSE = mse_y2, Response = "Y2")

# Combine the data
mse_data <- rbind(mse_data_y1, mse_data_y2)

# Plot MSEs for Y1 and Y2 on the same graph
ggplot(mse_data, aes(x = Model, y = MSE, group = Response, color = Response)) +
geom_line() +   geom_point(size = 3) +   theme_minimal() +
  labs(title = "MSE for Y1 and Y2 Models", y = "MSE", x = "Model") +
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Question 2

Abstract

This study evaluates the performance of Naive Bayes, k-NN, and Kernel Density Estimation (KDE) classifiers on a wine dataset. We pre-processed the data to handle skewness through centering, scaling, and Box-Cox transformation. Despite the preprocessing we noticed outliers. We minimized effect of outliers by spatial sign transformation. After removing highly correlated variables, PCA was applied to retain components explaining 95% of the variance. The Naive Bayes classifier achieved an accuracy of 98.36% outperforming the KDE 97.22% accuracy and k-NN (83% accuracy) classifiers. These results suggest that the simpler Gaussian-based Naive Bayes model offers better generalization for this dataset. Further validation with external datasets is recommended to confirm these findings.

Introduction

In the context of machine learning, accurately classifying wine samples based on their chemical properties is a well-established problem. This problem has been widely studied to ensure quality control and consistency in the wine industry. Traditional parametric methods such as Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) assume that the data follows a normal distribution. However, this assumption often does not hold true for real-world data, which can lead to suboptimal performance. This study explores the performance of three classifiers: Naive Bayes, k-NN, and Kernel Density Estimation (KDE). Given the non-normality observed in many features of the wine dataset, we employ various preprocessing techniques to mitigate skewness and outliers. This includes centering, scaling, Box-Cox transformation, and spatial sign transformation. Additionally, Principal Component Analysis (PCA) is used to reduce dimensionality while retaining essential variance.

Methodology

Data Preprocessing

The initial step in our analysis involved preprocessing the dataset to handle skewness and outliers. The data was centered and scaled, followed by the application of the Box-Cox transformation. This transformation is particularly useful for stabilizing variance and making the data more normally distributed. Despite these efforts, the data still exhibited skewness and outliers, as evidenced by the histograms and Q-Q plots. To further address these issues, the spatial sign transformation was applied to minimize effect of outliers. This transformation scales the data to lie within a unit sphere, thereby reducing the impact of outliers and improving the robustness of the analysis.

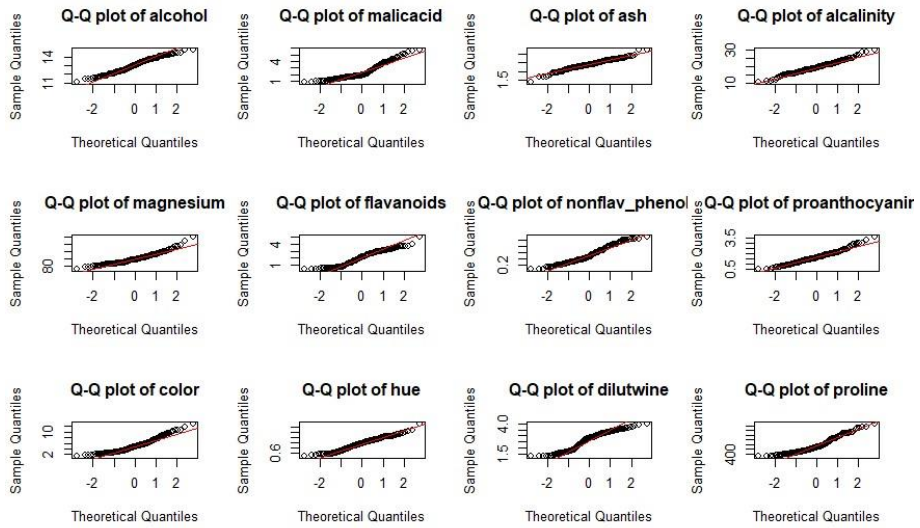


Figure 8 The Q-Q plots compare the quantiles of the feature data to the quantiles of a normal distribution. Points that fall along the 45-degree line indicate that the data is approximately normally distributed. Deviations from this line suggest departures from normality. For example, the Q-Q plots for alcohol and flavonoids show points close to the line, indicating approximate normality, while the Q-Q plots for malic acid and nonflavonoid phenols show deviations, indicating nonnormality.

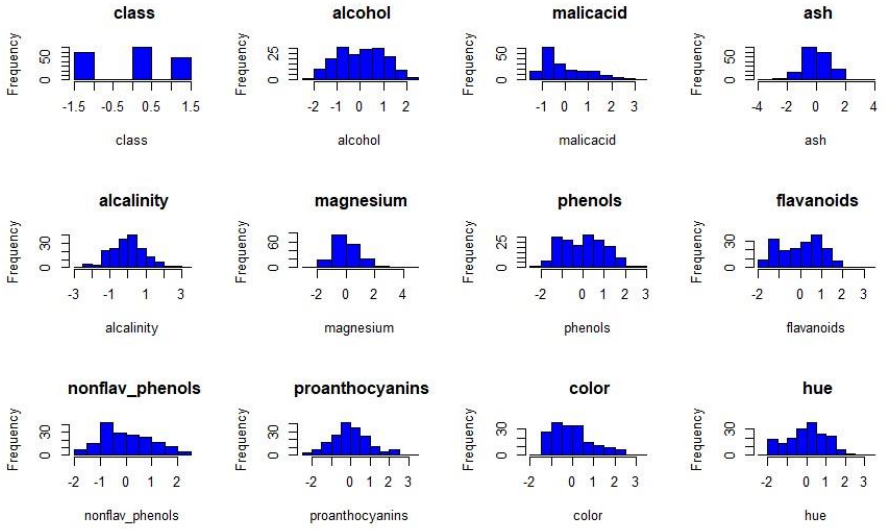


Figure 9 The histograms show the frequency distribution of each feature in the wine dataset after preprocessing. They help in visualizing the distribution shape, skewness, and presence of outliers. For instance, features like alcohol, ash, and flavonoids exhibit roughly normal distributions, while others like malic acid and nonflavonoid phenols show skewness.

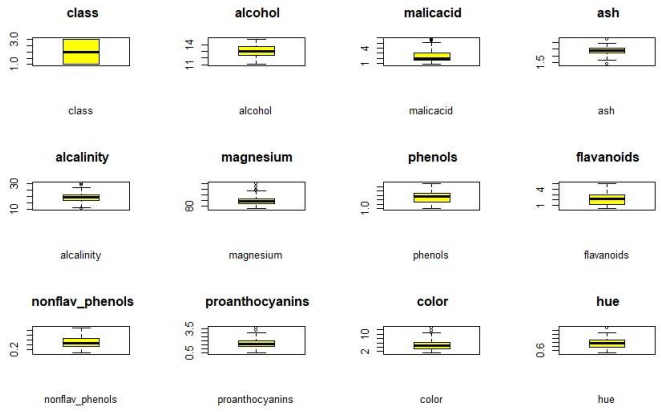


Figure 10 The box plots provide a summary of the data through their quartiles. They display the median, interquartile range (IQR), and potential outliers. For example, the box plot for malic acid shows a right-skewed distribution with some outliers, while the box plot for alcohol shows a more symmetrical distribution.

Feature Selection and Dimensionality Reduction

We identified and removed highly correlated features to prevent redundancy and multicollinearity. Specifically, phenol was removed due to its higher correlation with other features compared to flavonoids. Following this, PCA was performed to reduce the dataset's dimensionality. We retained components that explained 95% of the variance, which helped in maintaining the dataset's essential information while facilitating more efficient modelling.

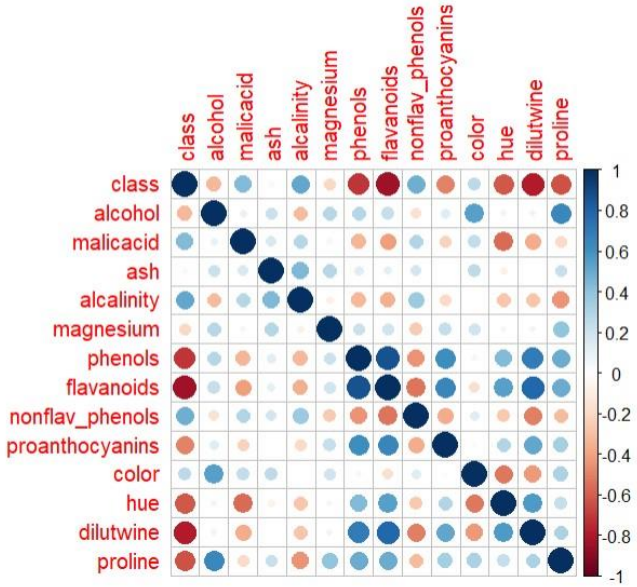


Figure 11: Phenol and flavonoids show strong correlation. Testing an average of the correlations of phenol and flavonoids with other features shows that phenols had stronger correlation leading to its removal. Other features show low positive or negative correlation.

Model Training and Evaluation

We evaluated the performance of three classifiers: Naive Bayes (Gaussian): This classifier assumes that the features are normally distributed and conditionally independent given the class label. Despite this strong assumption, Naive Bayes is known for its simplicity and computational efficiency, often performing well even when the independence assumption is violated.

k-NN: The k-nearest neighbours algorithm is a non-parametric method that classifies a sample based on the majority class of its k-nearest neighbours in the feature space. The optimal value of k was determined using cross-validation.

KDE (Nadaraya-Watson): This non-parametric classifier estimates the probability density function of the features and uses these estimates for classification. The bandwidth parameter, which controls the smoothness of the density estimate, was optimized through cross-validation.

We used 10-fold cross-validation to optimize the model parameters and assess their performance, focusing on accuracy and Kappa metrics. K-means not evaluated as it is an unsupervised method that might cluster the data into groups of 3 but not possibly not in a manner of interest.

Results

Model Performance

The performance of the classifiers was as follows: Naive Bayes: Accuracy = 98.36%, k-NN: Accuracy = 83%, KDE: Accuracy = 97.22%. The box plot of accuracies revealed that the Naive Bayes classifier consistently outperformed the other classifiers. The Gaussian assumption inherent in the Naive Bayes classifier appeared to generalize well for this dataset, likely due to the effectiveness of the preprocessing steps.

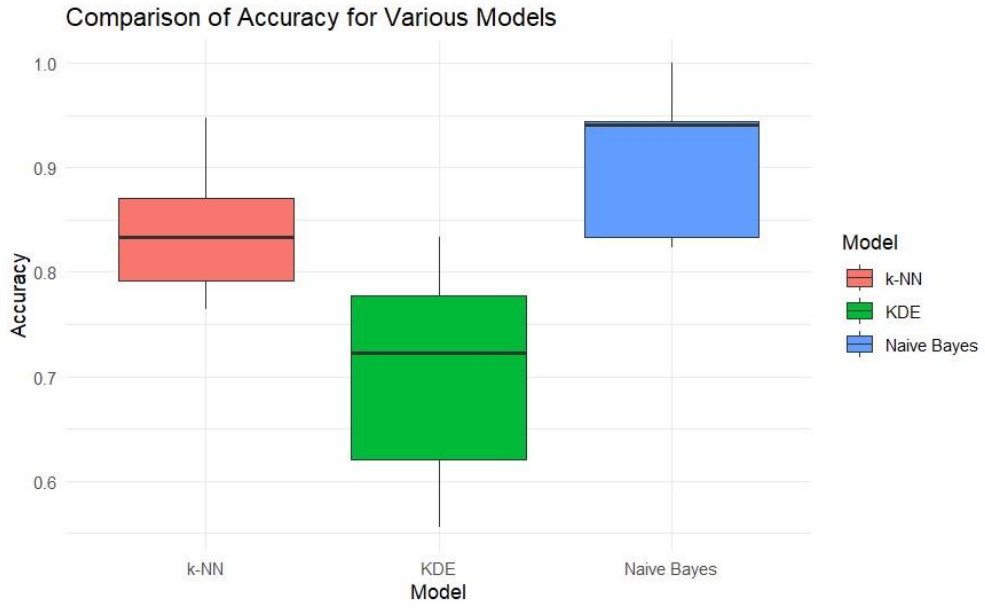


Figure 12 This plot compares the accuracy of three different models used for predicting the response variable Y3. The models evaluated are k-Nearest Neighbors (k-NN), Kernel Density Estimation (KDE), and Naive Bayes.

Comparison with Other Classifiers

Previous studies, such as the one by S. Aeberhard, D. Coomans, and O. de Vel (1992), have compared various classifiers on the same wine dataset. Their results using the leave-one-out technique showed RDA: 100% accuracy, QDA: 99.4% accuracy, LDA: 98.9% accuracy and 1-NN: 96.1% accuracy. While RDA, QDA, and LDA achieved higher accuracies, they rely on assumptions of normality and homogeneity of variances. These assumptions might not hold for all features in the dataset, potentially leading to overfitting. Naive Bayes also assumes normality for each feature given the class label. However, Naive Bayes being a non-parametric method is more robust in practice and not be

considerably affected when data deviates from assumptions. Moreover, its simplicity and computational efficiency make it a practical choice for many applications.

Considerations for Wine Aging

It's important to note that free anthocyanins (and possibly other features) in red wines are not particularly stable, and their concentration usually drops quickly during wine aging in barrels or bottles. After several years, almost no monomeric anthocyanins are present in aged wines due to polymerization or other modification reactions with other compounds, as well as breakdown reactions. Testing wines at various time points could provide different data due to this degradation, and this variability should be considered when interpreting classification results and model performance over time.

Advantages and Disadvantages of Naive Bayes

Advantages: Naive Bayes is computationally efficient and easy to implement, making it suitable for high-dimensional datasets. If more features are deemed relevant for classification, this model could handle those well without significant performance degradation. Naive Bayes can perform well even with relatively small datasets.

Disadvantages: The model assumes that features are conditionally independent which is often violated in practice. Strong correlations between features can negatively impact the performance of Naive Bayes. Naive Bayes assumes that the features are normally distributed within each class, which might not always be the case. However, the model's performance can still be robust due to its other advantages.

Naive Bayes despite achieving slightly lower accuracy than RDA, QDA, and LDA offers robustness making it suitable for high-dimensional datasets where assumptions of other classifiers might not hold.

Conclusion

In conclusion, the Naive Bayes classifier demonstrated superior performance compared to k-NN and KDE classifiers on the wine dataset. The preprocessing steps, including centering, scaling, Box-Cox transformation, spatial sign transformation, and PCA, contributed to the models' effectiveness. While Naive Bayes showed the best performance, further validation on external datasets and using repeated cross-validation is recommended to ensure robustness. Future work could optimize preprocessing techniques. Additionally, testing wines at various aging stages should be considered, as the chemical composition can change significantly over time due to degradation of compounds such as anthocyanins. We recommend using a Naive Bayes classifier for its robustness in real world, acceptance of new features and accuracy.

Appendix - Q2 code

```
wine_data <- read.csv("wine.csv")

summary(wine_data)
preprocess_params <- preProcess(numeric_data,
                        method = c("center", "scale", "BoxCox"))
par(mfrow=c(3, 4)) # Set up the plotting area for multiple plots for
(col in names(wine_data)) {
  hist(scale(wine_data[[col]]), main=paste("", col), xlab=col, col="blue")
}

# Set up the plotting area for multiple plots (box plots next) par(mfrow=c(3,
4)) # Adjust this if you have more/less columns

# Plot box plots for each feature in the dataset for
(col in names(wine_data)) {
  boxplot(wine_data[[col]], main=paste("", col), xlab=col, col="yellow")
}
par(mfrow=c(3, 4)) # Adjust this if you have more/less
columns for (col in names(wine_data_uncorr)) {   if
(is.numeric(wine_data_uncorr[[col]])) {
    qqnorm(wine_data_uncorr[[col]], main=paste("Q-Q plot of", col))
qqline(wine_data_uncorr[[col]], col = "red")
  }
}

library(corrplot)
cor_matrix <- cor(wine_data)

# Plot the correlation matrix
corrplot(cor_matrix, method = "circle") i_star
<- "phenols"
j_star <- "flavanoids"

# Get the names of all other variables excluding i* and j* other_vars
<- setdiff(names(wine_data), c(i_star, j_star))

# Compute the average correlation of i* with all other variables avg_corr_i_star
<- mean(cor_matrix[i_star, other_vars])

# Compute the average correlation of j* with all other variables avg_corr_j_star
<- mean(cor_matrix[j_star, other_vars])

# Print the average correlations
print(paste("Average correlation of", i_star, "with all other variables:", avg_corr_i_star))
print(paste("Average correlation of", j_star, "with all other variables:", avg_corr_j_star))

# Decide which variable to remove
```

```
if (avg_corr_i_star > avg_corr_j_star) {
  print(paste("Remove", i_star))
} else {
  print(paste("Remove", j_star))
}
library(dplyr)
library(e1071)
library(caret)

# Ensure 'class' is a factor for classification
wine_data$class <- as.factor(wine_data$class)

# Create a new data frame without the 'phenols' column
wine_data_uncorr <- wine_data[, !(names(wine_data) == "phenols")]

# Separate the numeric columns from the factor columns
numeric_data <- wine_data_uncorr %>% select_if(is.numeric)
factor_data <- wine_data_uncorr %>% select_if(is.factor)

# Preprocess the numeric data (center, scale, BoxCox)
preprocess_params <- preProcess(numeric_data,
                        method = c("center", "scale", "BoxCox"))

# Apply the preprocessing to the numeric data
numeric_data_trans <- predict(preprocess_params, numeric_data)

# Combine the preprocessed numeric data with the factor data
wine_data_trans <- bind_cols(numeric_data_trans, factor_data)
# Set up the plotting area for multiple box plots
par(mfrow=c(3, 4)) # Adjust this if you have more/less columns

# Plot box plots for each feature in the
dataset for (col in names(wine_data_trans)) {
if (is.numeric(wine_data_trans[[col]])) {
   boxplot(wine_data_trans[[col]], main=paste("Boxplot of", col), xlab=col, col="yellow")
  }
}

# Function to apply the spatial sign transformation
spatial_sign <- function(data) {   norm_data <-
sqrt(rowSums(data^2))   spatial_sign_data <-
sweep(data, 1, norm_data, "/")
return(spatial_sign_data)
}
```

```
# Apply the spatial sign transformation only to the numeric data
numeric_data_spatial_sign <- spatial_sign(numeric_data_trans)

# Combine the spatially transformed numeric data with the factor data
wine_data_spatial_sign <- bind_cols(as.data.frame(numeric_data_spatial_sign), factor_data)
wine_data$class <- as.factor(make.names(wine_data$class))

# Apply the spatial sign transformation
spatial_sign <- function(x) {
  return(x / sqrt(rowSums(x^2)))
}

# Exclude the class column and apply the transformation
wine_data_numeric <- wine_data[, -1]
wine_data_spatial_sign <- cbind(spatial_sign(wine_data_numeric), class = wine_data$class)

# Define train control for 10-fold cross-validation
set.seed(123)  # Set seed for reproducibility
train_control <- trainControl(method = "cv", number = 10)

# Train the Naive Bayes model using the transformed data
nb_model <- train(class ~ ., data = wine_data_spatial_sign,
method = "nb",
              trControl = train_control)

# Optimize the k value for k-NN using cross-validation
knn_model <- train(class ~ ., data =
wine_data_spatial_sign,                  method = "knn",
trControl = train_control,
               tuneLength = 10)  # Tune k with 10 different values

# Function to perform 10-fold cross-validation for KDE with bandwidth tuning
nw_cv <- function(data, class_col, folds) {
  accuracies <- c()

  set.seed(123)
  indices <- createFolds(data[[class_col]], k = folds)

  for (i in 1:folds) {
test_indices <- indices[[i]]
    train_indices <- setdiff(1:nrow(data), test_indices)

    train_data <- data[train_indices, ]
    test_data <- data[test_indices, ]
```

```
   # Tuning the bandwidth parameter
bandwidths <- seq(0.1, 1.5, by = 0.1)
best_accuracy <- 0
   best_bandwidth <- 0.1

   for (bw in bandwidths) {
     preds <- sapply(1:nrow(test_data), function(i) {        densities <-
sapply(levels(train_data$class), function(cls) {          class_data <-
train_data[train_data$class == cls, -ncol(train_data)]
        kde <- apply(class_data, 2, function(x) density(x, bw = bw, kernel = "gaussian", n = 512))
density_values <- sapply(1:ncol(class_data), function(j) {
kde[[j]]$y[which.min(abs(kde[[j]]$x - test_data[i, j]))]
        })
        prod(density_values)
       })
       levels(train_data$class)[which.max(densities)]
     })

     preds <- factor(preds, levels = levels(data[[class_col]]))
cm <- confusionMatrix(preds, test_data[[class_col]])
accuracy <- cm$overall['Accuracy']

     if (accuracy > best_accuracy) {
best_accuracy <- accuracy
       best_bandwidth <- bw
     }
   }

   # Store the best bandwidth and its accuracy for this fold     preds <-
sapply(1:nrow(test_data), function(i) {       densities <-
sapply(levels(train_data$class), function(cls) {        class_data <-
train_data[train_data$class == cls, -ncol(train_data)]
       kde <- apply(class_data, 2, function(x) density(x, bw = best_bandwidth, kernel = "gaussian",
n
= 512))        density_values <- sapply(1:ncol(class_data),
function(j) {          kde[[j]]$y[which.min(abs(kde[[j]]$x -
test_data[i, j]))]
       })
       prod(density_values)
     })
     levels(train_data$class)[which.max(densities)]
   })

   preds <- factor(preds, levels = levels(data[[class_col]]))
cm <- confusionMatrix(preds, test_data[[class_col]])
accuracies <- c(accuracies, cm$overall['Accuracy'])
```

```
  }

  return(list(accuracies = accuracies, best_bandwidth = best_bandwidth))
}

# Perform 10-fold cross-validation for KDE
nw_results <- nw_cv(wine_data_spatial_sign, "class", 10) nw_accuracies
<- nw_results$accuracies
best_bandwidth <- nw_results$best_bandwidth

# Ensure there are no non-finite values
nb_accuracies <- nb_model$resample$Accuracy
knn_accuracies <- knn_model$resample$Accuracy
nw_accuracies <- ifelse(is.finite(nw_accuracies), nw_accuracies, NA)

# Combine accuracies into a data frame for plotting accuracy_data
<- data.frame(
```

```
  Model = rep(c("Naive Bayes", "k-NN", "KDE"), each = 10),
  Accuracy = c(nb_accuracies, knn_accuracies, nw_accuracies)
)

# Remove non-finite values for plotting
accuracy_data <- accuracy_data[complete.cases(accuracy_data), ]

# Identify the best model based on average accuracy
mean_accuracies <- aggregate(Accuracy ~ Model, data = accuracy_data, mean)
best_model_name <- mean_accuracies[which.max(mean_accuracies$Accuracy), "Model"]

# Print the best model and its average accuracy
print(paste("The best model is", best_model_name, "with an average accuracy of",
mean_accuracies[which.max(mean_accuracies$Accuracy), "Accuracy"]))

# Print parameters of the best model if
(best_model_name == "Naive Bayes") {
print(nb_model$bestTune)
} else if (best_model_name == "k-NN") {
print(knn_model$bestTune)
} else if (best_model_name == "KDE") {
  print(paste("Best bandwidth for KDE:", best_bandwidth))
}

# Create a box plot to compare the accuracy of all models
ggplot(accuracy_data, aes(x = Model, y = Accuracy, fill = Model))
+   geom_boxplot() +   theme_minimal() +
  labs(title = "Comparison of Accuracy for Various Models",
x = "Model",
      y = "Accuracy")
```