Wine Cultivar Classification

Agassi Boonprasert, Eric Vargas

MATH 478: Statistical Methods in Data Science

Dr. Guo Wenge

May 5, 2026

Introduction

Wine authentication and cultivar identification are important problems in food quality control. Instead of simply relying on expensive sensory evaluation by experts, chemical analysis is often a better, cheaper alternative. This study finds what statistical learning method best identifies the cultivar origin of a wine sample given only its chemical composition.

The data used in this study comes from the UCI Machine Learning Repository and consists of 178 wine samples originating from three grape cultivars grown in the same region of Italy. Each sample is described by 13 physicochemical measurements, including alcohol content, flavanoids, proline, and color intensity. Since the true cultivar label is known for every record, this is a supervised multiclass classification problem with three classes. Four statistical learning methods were and are described in the Methods section, followed by Results and DIscussion.

Methods

Preprocessing

The dataset was downloaded directly from the UCI Machine Learning Repository and required minimal preprocessing. Column names were assigned to match the 13 physicochemical measurements from the 'names' file provided in the download. The class label was converted to a categorical variable with three levels (1, 2, 3). As no missing values were found in any column, no imputation was necessary.

The data was split into a training set (80%) and a test set (20%) using a random method, meaning that there was a uniform chance that any sample could appear in the dataset. This resulted in 142 observations in the training set and 36 observations in the test set. The predictors were standardized using the mean and standard deviation of the training set, which was then applied to the test set. This ensures that no data leakage into the test set occurs. The standardized variables were used for all statistical methods, regardless of if they are sensitive to scale. This helps keep our code clean and consistent.

Exploratory Data Analysis

Before fitting any models, exploratory data analysis (EDA) was done to better understand the structure of the data and help guide modeling choices. First, the class distribution was examined and found to be fairly balanced across the three grape cultivars, with 59, 71, and 48 observations in classes 1, 2, and 3, respectively (Figure 1).

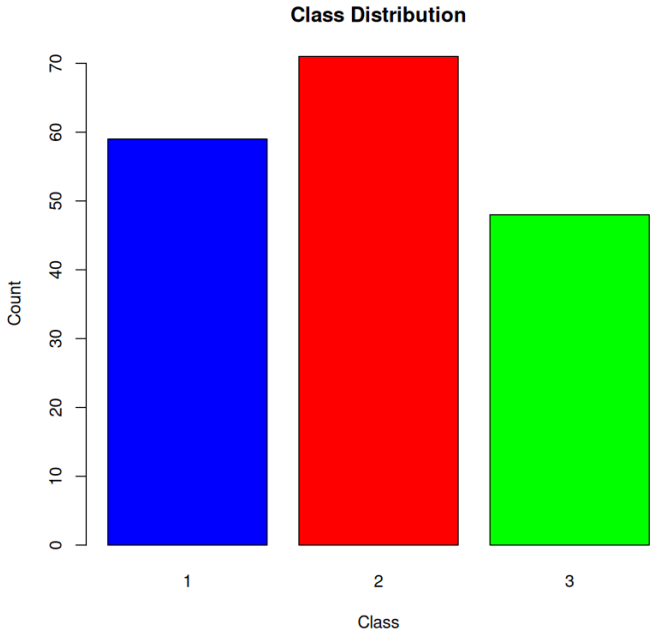

Figure 1: Class Distribution of Cultivators

A pairwise correlation matrix was computed across all 13 predictors (Figure 2). Several strong correlations were observed. For example, Flavanoids and Total Phenols contained a correlation coefficient of 0.86, and Flavanoids and OD280/OD315 had a correlation of 0.79. OD280/OD315 is a chemical metric used to measure the concentration and purity of proteins in diluted wine samples. This shows that some multicollinearity occurs. However, because prediction is the main focus, rather than inference, we chose to keep all variables.

Two pairwise scatterplot matrices across a set of six predictors and a set of seven predictors were observed to gain a general understanding of the split between classes (Figures 3 and 4). From these, we can see that the classes are reasonably separable, though not perfectly in all predictor pairs. This suggests that we should test both linear and nonlinear methods instead of just one or the other.

	Alcohol	Malic.Acid	Ash	Alcalinity.Ash	Magnesium	Total.Phenols	Flavanoids
Alcohol	1.00	0.09	0.21	-0.31	0.27	0.29	0.24
Malic.Acid	0.09	1.00	0.16	0.29	-0.05	-0.34	-0.41
Ash	0.21	0.16	1.00	0.44	0.29	0.13	0.12
Alcalinity.Ash	-0.31	0.29	0.44	1.00	-0.08	-0.32	-0.35
Magnesium	0.27	-0.05	0.29	-0.08	1.00	0.21	0.20
Total.Phenols	0.29	-0.34	0.13	-0.32	0.21	1.00	0.86
Flavanoids	0.24	-0.41	0.12	-0.35	0.20	0.86	1.00
Nonflavanoid.Phenols	-0.16	0.29	0.19	0.36	-0.26	-0.45	-0.54
Proanthocyanins	0.14	-0.22	0.01	-0.20	0.24	0.61	0.65
Color.Intensity	0.55	0.25	0.26	0.02	0.20	-0.06	-0.17
Hue	-0.07	-0.56	-0.07	-0.27	0.06	0.43	0.54
OD280.OD315	0.07	-0.37	0.00	-0.28	0.07	0.70	0.79
Proline	0.64	-0.19	0.22	-0.44	0.39	0.50	0.49

	Nonflavanoid.Phenols	Proanthocyanins	Color.Intensity	Hue	OD280.OD315	Proline
Alcohol	-0.16	0.14	0.55	-0.07	0.07	0.64
Malic.Acid	0.29	-0.22	0.25	-0.56	-0.37	-0.19
Ash	0.19	0.01	0.26	-0.07	0.00	0.22
Alcalinity.Ash	0.36	-0.20	0.02	-0.27	-0.28	-0.44
Magnesium	-0.26	0.24	0.20	0.06	0.07	0.39
Total.Phenols	-0.45	0.61	-0.06	0.43	0.70	0.50
Flavanoids	-0.54	0.65	-0.17	0.54	0.79	0.49
Nonflavanoid.Phenols	1.00	-0.37	0.14	-0.26	-0.50	-0.31
Proanthocyanins	-0.37	1.00	-0.03	0.30	0.52	0.33
Color.Intensity	0.14	-0.03	1.00	-0.52	-0.43	0.32
Hue	-0.26	0.30	-0.52	1.00	0.57	0.24
OD280.OD315	-0.50	0.52	-0.43	0.57	1.00	0.31
Proline	-0.31	0.33	0.32	0.24	0.31	1.00

Figure 2: Scatterplot Matrix of the first six predictors

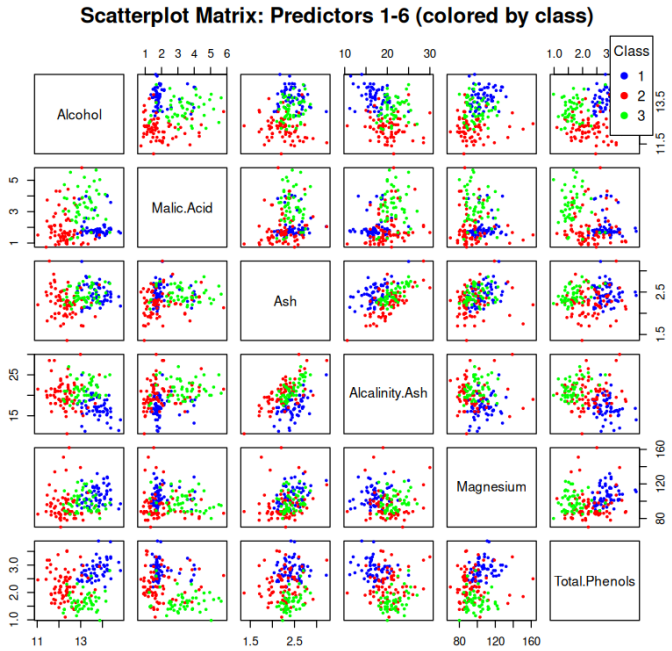

Figure 3: Scatterplot Matrix of the first six predictors

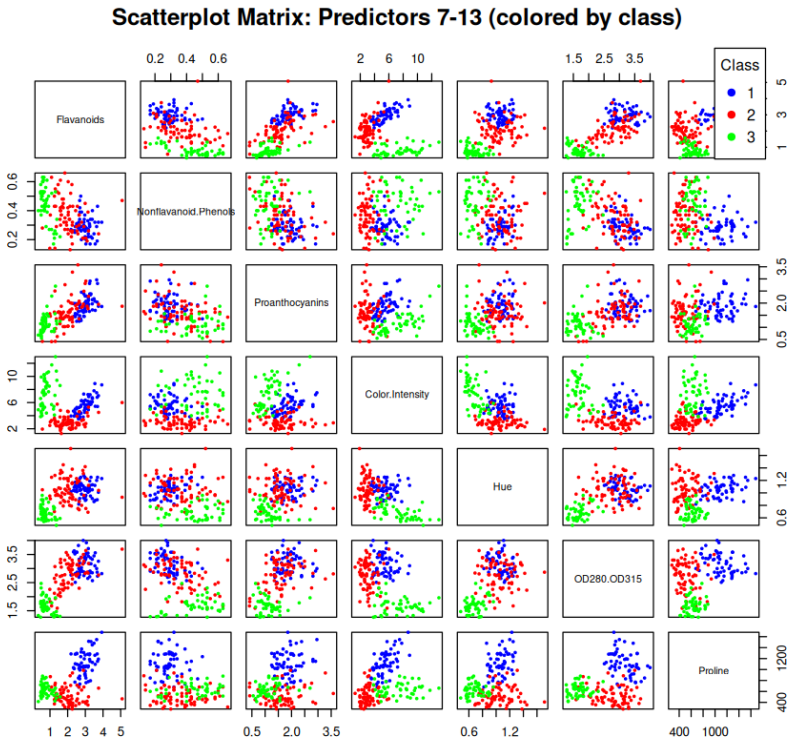

Figure 4: Scatterplot Matrix of the remaining seven predictors

Boxplots were generated for all 13 predictors, colored by class (Figure 5), showing that features such as Flavanoids, Proline, Color Intensity, and OD280/OD315 showed the strongest separation across the three cultivars, while predictors such as Ash and Magnesium showed to be less important, indicated by the considerable overlap between classes.

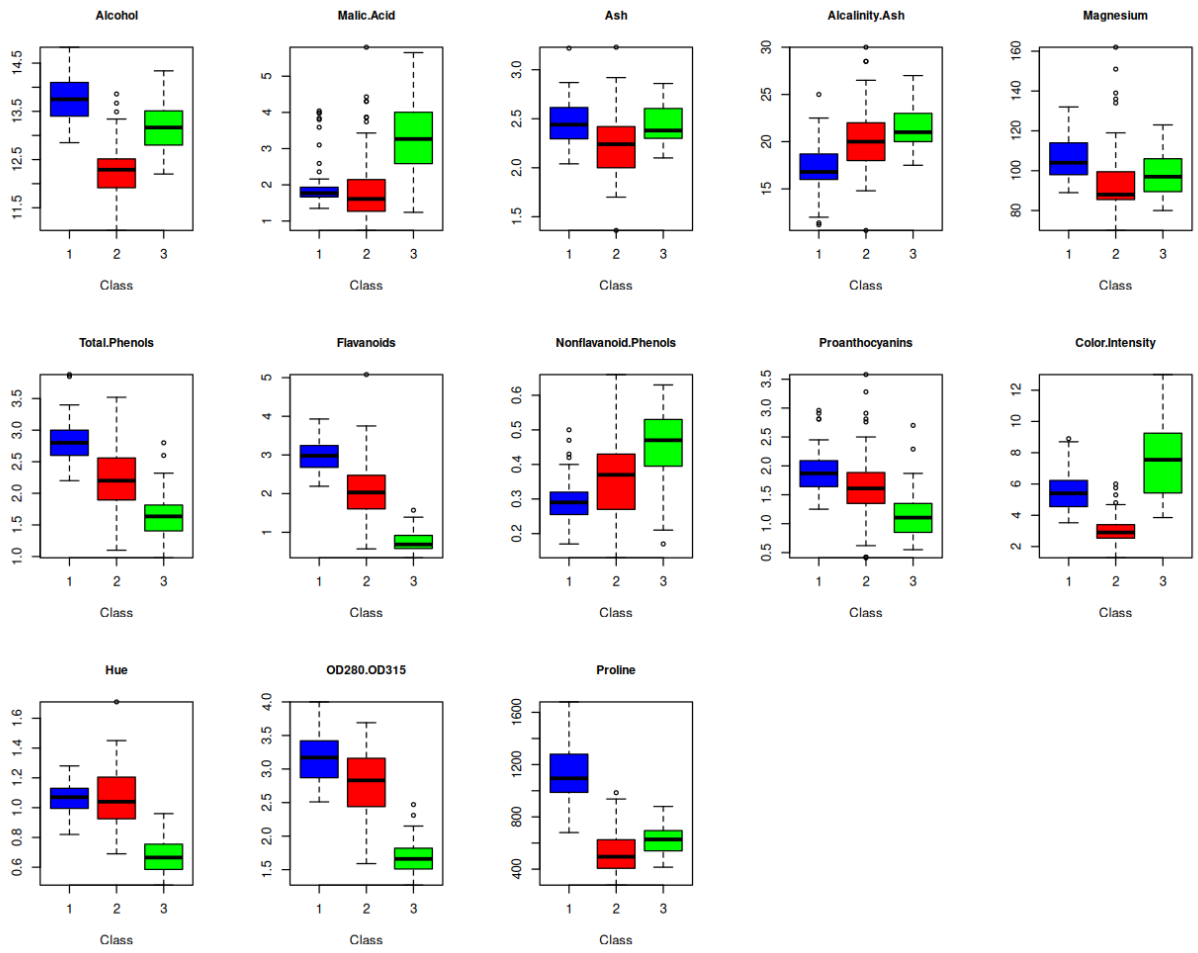

Figure 5: Boxplots of all predictors

Statistical Learning Methods

Given the results from EDA, we decided to use four different methods: Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machines, and Random Forests. Three out of the four models were tuned using 10-fold cross-validation on the training set, and final performance was evaluated on the held-out test set. We decided on 10-fold cross-validation over 3-fold or 5-fold because the small size of the dataset allowed us to use a higher amount of folds without a significant penalty to computation time. This will help in finding a low-bias estimate of model performance.

Linear Discriminant Analysis (LDA) finds linear combinations of the predictors that best separate the classes. Since the problem is multiclass classification, and fairly linear separable (noted from EDA), we deemed it a solid starting point. Also, LDA had no tuning parameters, meaning it serves a good baseline to compare the other methods against.

K-Nearest Neighbors (KNN) classifies a new observation by majority vote from the k nearest points in the predictor space based on euclidean distance. Unlike LDA, KNN makes no assumption about the shape of the decision boundary, which makes it a useful method here. The number of neighbors k was selected by cross-validation over the values k = 1, 3, 5, 7, and 9.

Support Vector Machine (SVM) with a radial basis kernel finds a decision boundary that maximizes the margins between classes. The radial kernel allows the SVM to fit nonlinear decision boundaries that LDA cannot, which is useful given that not all predictors showed clean linear separation in the EDA. Furthermore, SVM provides a good alternative to KNN because with 13 predictors, KNN may struggle with classifying due to the curse of dimensionality, which SVM proves to be more robust to. The cost parameter, C, and the kernel width parameter gamma were both selected by cross validation. The values of C included 0.1, 1, 10, and 100, while the values of gamma included 0.01, 0.1, 0.5, 1.

Random Forest builds a large number of decision trees on random subsamples of the training data, each using a random subset of predictor variables at each split, and combines their predictions by majority vote. Additionally, it handles multicollinearity well, which is evident from the correlation matrix. The number of predictors considered at each split (mtry) was tuned by cross-validation over the values 2, 4, 7, 10, 13, while the number of trees was 500. Lastly, Random Forest provides variable importances, which will help us with interpreting what predictors are most significant in predicting wine cultivars.

Results

The test set accuracies for all four models are summarized in Figure 6. LDA, SVM, and Random Forest all achieved perfect accuracy on the test set, while KNN achieved an accuracy of 0.9167, misclassifying 3 of the 36 test observations.

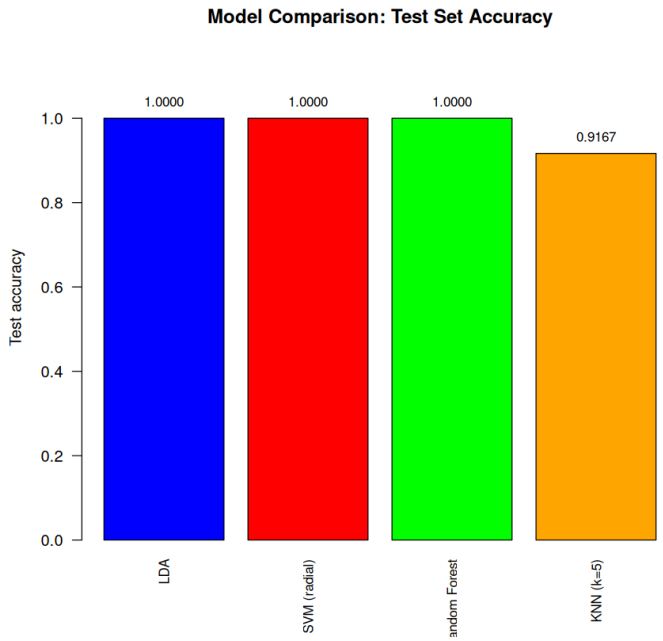

Figure 6: Comparison between LDA, SVM, RF, and KNN

As a result, the confusion matrices for LDA, SVM, and Random Forest were identical, with 14, 15, and 7 correct predictions for classes 1, 2, and 3, respectively. KNN's 3 misclassifications all came from class 2 observations being predicted as class 1 (Figures 7 and 8).

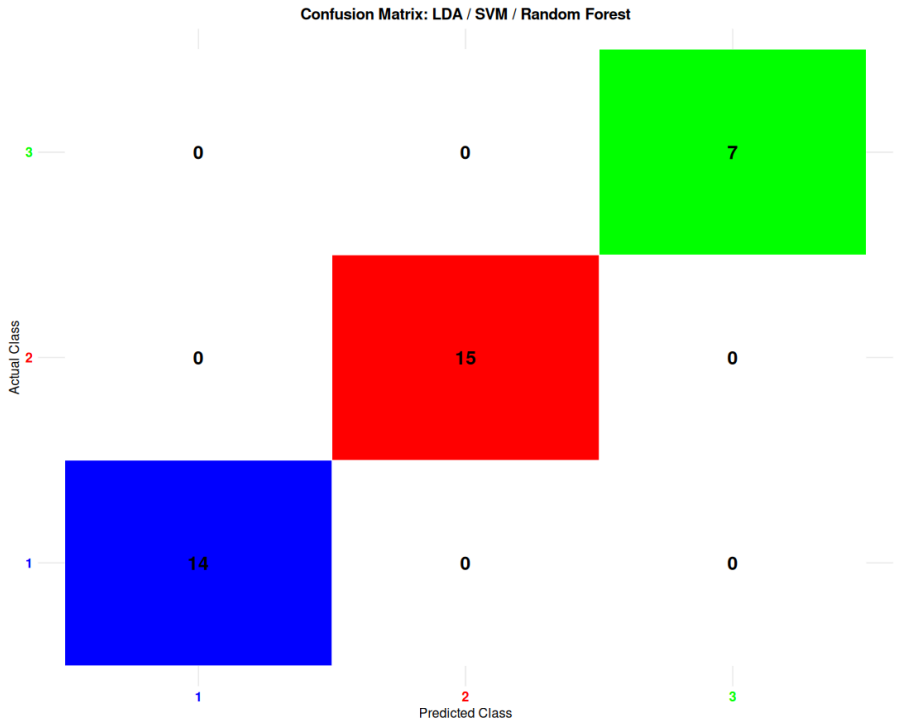

Figure 8: LDA, SVM, and Random Forest Confusion Matrix

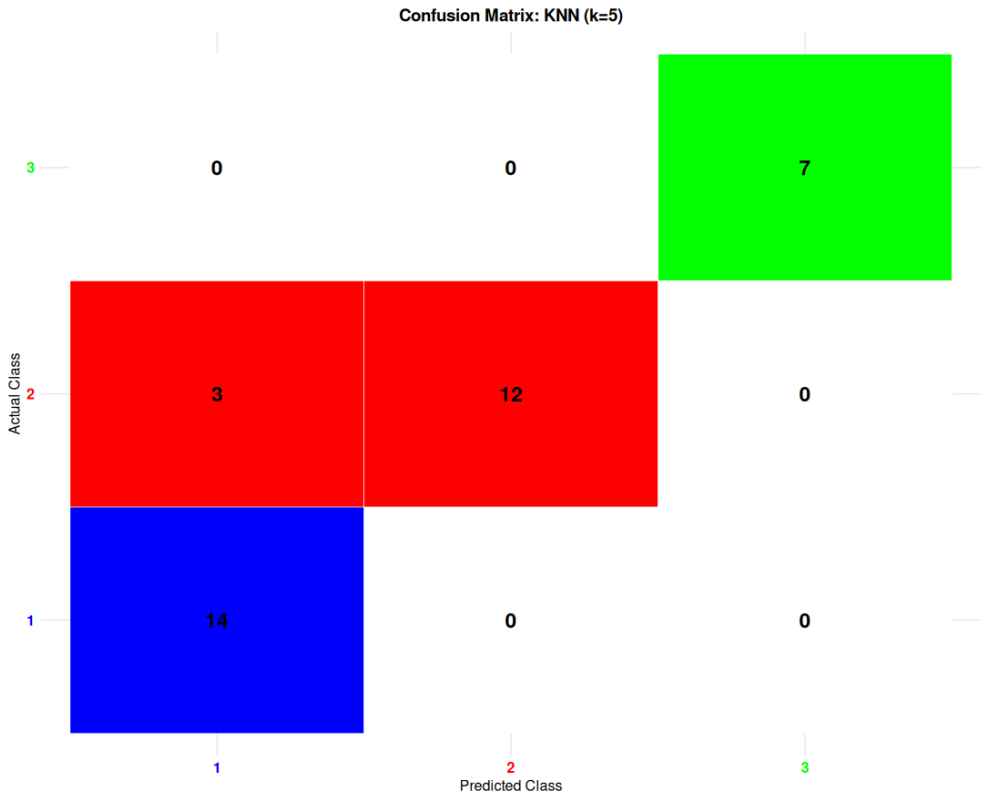

Figure 9: KNN Confusion Matrix

The LDA training projection plot (Figure 9) shows the three cultivar classes separating cleanly into distinct clusters, with no meaningful overlap. For KNN, the CV error plot for KNN (Figure 10) showed that k values of 5 and 7 produced the lowest cross-validation error at roughly 0.035. The model with a k value of 5 was chosen as it was the smallest k to achieve the minimum error. For SVM, the best tuning parameters selected by 10-fold cross-validation were a cost value of 1 and a gamma value of 0.1, with a best CV error of 0.0281. For Random Forest, the best number of predictors considered at each split was 2. The variable importance plot (Figure 11) showed that Flavanoids, Color Intensity, Proline, Alcohol, and OD280/OD315 were the five most important predictors by both Mean Decrease in Accuracy and Mean Decrease in Gini.

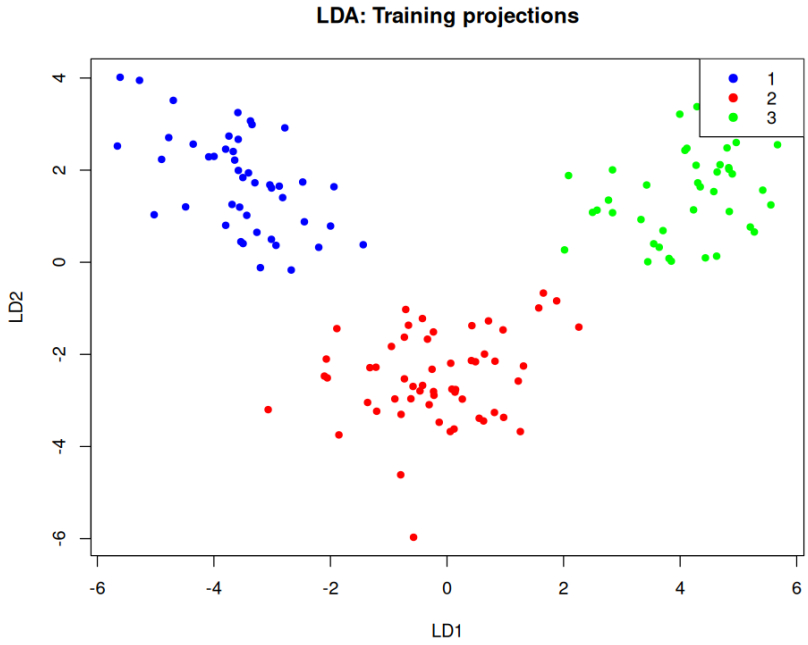

Figure 9: Scatterplot of LD1 and LD2 projections

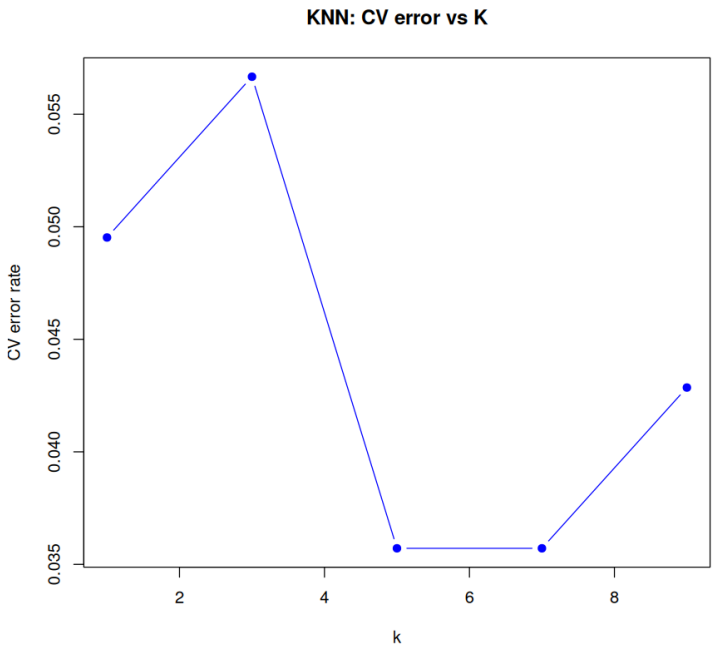

Figure 10: Validation error v. K (KNN)

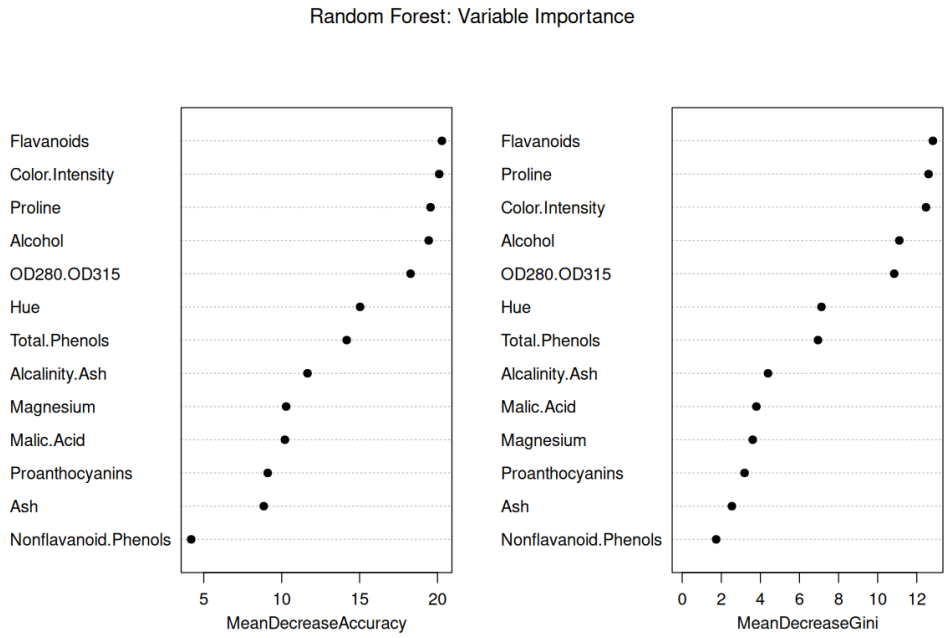

Figure 11: Variable Importance in Random Forest

Discussion

The strong performance of LDA, SVM, and Random Forest is not entirely surprising given what the EDA showed us.

LDA was perfect, despite being the simplest of the four methods. This aligns with what the scatterplot matrices indicated, namely that the classes are mostly linearly separable, precisely the environment that LDA should be well suited to work in.

SVM using a radial kernel also attained optimal accuracy. The optimal tuning parameters allowed for a moderate level of flexibility. The fact that SVM was able to match LDA implies that the decision boundary does not need to be too complex to be able to achieve perfect classification on unseen data.

Random Forest's results matched the other best methods as well. Its variable importance plot offers some useful additional information. Flavanoids is the most important predictor by both measures, followed closely by Color Intensity, Proline, Alcohol and OD280/OD315. These results are directly correlated to what the boxplots indicated during the EDA, where these same predictors had the most consistent separation across the classes. Conversely, predictors such as Nonflavaniod Phenols and Ash, were found to be the least significant predictors in both importance measures hence, they do not significantly contribute to the classification.

The only model that failed to reach the highest accuracy was the KNN. KNN is a strictly

local method, meaning it makes predictions based only on the nearest neighbors in the training set, so in parts of predictor space where there is a slight overlap between class 1 and class 2 it can be problematic. This is an established weakness of KNN in comparison to other techniques such as LDA and SVM, which make use of global data about the structure of classes.

A primary limitation of our study is that our test set only contains 36 observations. Therefore, a high accuracy on a relatively small test set should be taken with caution since it does not necessarily imply that these models will generalize to new wine samples in practice. A bigger or repeated assessment would provide more credible estimates of actual performance. Additionally, the data was taken from cultivars in Italy which may not be representative of the vast majority of other cultivars there could be. Further research could be done through data collection on most samples throughout Italy for these specific cultivars.

Appendix (R Code)

```
# wine cultivar classification using chemical measurements

# libraries
# install.packages(c("MASS", "class", "e1071", "randomForest"))
library(MASS)
library(class)
library(e1071)
library(randomForest)

# for reproducibility
set.seed(1)

# LOAD DATA
# research question:
# Which statistical learning method best classifies wine
cultivars (1, 2, 3)
# using physicochemical predictors in the UCI Wine dataset?
#
# source: UCI ML repository (wine dataset)
# 178 observations, 13 predictors

df <- read.csv("wine.data", header = FALSE)

# from names file
colnames(df) <- c(
  "Class",
  "Alcohol", "Malic.Acid", "Ash", "Alcalinity.Ash",
  "Magnesium", "Total.Phenols", "Flavanoids",
"Nonflavanoid.Phenols",
  "Proanthocyanins", "Color.Intensity", "Hue", "OD280.OD315",
"Proline"
)

# response
df$Class <- as.factor(df$Class)


# EDA
cat("Dimensions:", dim(df), "\n")
head(df)
```

```
summary(df)

cat("\nClass distribution:\n")
print(table(df$Class))

cat("\nMissing values per column:\n")
print(colSums(is.na(df)))
# No missing values in this dataset

# check class balance
barplot(
  table(df$Class),
  col  = c("blue", "red", "green"),
  main = "Class Distribution",
  xlab = "Class",
  ylab = "Count"
)

# correlation matrix
corr_mat <- cor(df[, -1])
print(round(corr_mat, 2))

# scatterplot matrices (split into two halves for visibility)
class_cols <- c("blue", "red", "green")[as.integer(df$Class)]

# first half of predictors
pairs(
  df[, 2:7],
  col  = class_cols, pch = 19, cex = 0.4,
  main = "Scatterplot Matrix: Predictors 1-6 (colored by class)"
)
legend("topright", legend = levels(df$Class),
       col = c("blue", "red", "green"), pch = 19, title =
"Class", cex = 0.8)

# second half of predictors
pairs(
  df[, 8:14],
  col  = class_cols, pch = 19, cex = 0.4,
  main = "Scatterplot Matrix: Predictors 7-13 (colored by
class)"
```

```
)
legend("topright", legend = levels(df$Class),
       col = c("blue", "red", "green"), pch = 19, title =
"Class", cex = 0.8)

# boxplots for all 13 predictors by class
all_vars <- colnames(df)[-1]

par(mfrow = c(3, 5))
for (v in all_vars) {
  boxplot(
    df[[v]] ~ df$Class,
    col      = c("blue", "red", "green"),
    main     = v,
    xlab     = "Class",
    ylab     = "",
    cex.main = 0.9
  )
}
par(mfrow = c(1, 1))

# TRAIN/TEST SPLIT: 80/20
train <- sample(1:nrow(df), 0.8 * nrow(df))

# scale all predictors using the training set mean and standard
deviation
# applying the same scaling to all models keeps the code
consistent
# NOTE: LDA and RF are scale-invariant so this does not affect
their results
X_scaled <- scale(df[, -1])

X_train <- X_scaled[train, ]
X_test  <- X_scaled[-train, ]
y_train <- df$Class[train]
y_test  <- df$Class[-train]

# combine scaled predictors with class label into dataframes
wine_train <- data.frame(X_train, Class = y_train)
wine_test  <- data.frame(X_test,  Class = y_test)
```

```
cat("Train size:", nrow(wine_train), "  Test size:",
nrow(wine_test), "\n")

# function to help build confusion matrix
eval_model <- function(pred, truth, model_name) {
  cm  <- table(Predicted = pred, Actual = truth)
  acc <- mean(pred == truth)
  cat(sprintf("\n── %s ──\n", model_name))
  cat(sprintf("Test accuracy: %.4f\n", acc))
  cat("Confusion matrix:\n")
  print(cm)
  invisible(list(accuracy = acc, cm = cm))
}



# LDA
lda_fit  <- lda(Class ~ ., data = wine_train)
lda_pred <- predict(lda_fit, newdata = wine_test)$class
lda_res  <- eval_model(lda_pred, y_test, "LDA")

# LD1 vs LD2 scatterplot (colored by class)
lda_proj <- predict(lda_fit, newdata = wine_train)$x
plot(
  lda_proj[, 1], lda_proj[, 2],
  col  = c("blue", "red", "green")[as.integer(y_train)],
  pch  = 19, cex = 0.8,
  xlab = "LD1", ylab = "LD2",
  main = "LDA: Training projections"
)
legend("topright", legend = levels(y_train),
       col = c("blue", "red", "green"), pch = 19)



# KNN
# optimal k to be computed
k_vals  <- c(1, 3, 5, 7, 9)
n_folds <- 10

fold_id <- sample(rep(1:n_folds, length.out = nrow(X_train)))

cv_acc <- sapply(k_vals, function(k) {
```

```
  fold_acc <- sapply(1:n_folds, function(f) {
    trn  <- X_train[fold_id != f, ]
    tst  <- X_train[fold_id == f, ]
    lbl  <- y_train[fold_id != f]
    tru  <- y_train[fold_id == f]
    pred <- knn(train = trn, test = tst, cl = lbl, k = k)
    mean(pred == tru)
  })
  mean(fold_acc)
})

names(cv_acc) <- paste0("k=", k_vals)
cat("\nKNN cross-validation accuracies:\n")
print(round(cv_acc, 4))

best_k <- k_vals[which.max(cv_acc)]

# CV error vs k
plot(
  k_vals, 1 - cv_acc,
  type = "b", pch = 19, col = "blue",
  xlab = "k", ylab = "CV error rate",
  main = "KNN: CV error vs K"
)

# evaluate on test set
knn_pred <- knn(train = X_train, test = X_test, cl = y_train, k
= best_k)
knn_res <- eval_model(knn_pred, y_test, paste0("KNN (K=",
best_k, ")"))


# SVM w/ radial kernel
svm_tune <- tune(
  svm,
  train.x     = X_train,
  train.y     = y_train,
  kernel      = "radial",
  ranges      = list(
    cost  = c(0.1, 1, 10, 100),
    gamma = c(0.01, 0.1, 0.5, 1)
```

```
  ),
  tunecontrol = tune.control(cross = 10)
)

cat("\nSVM tuning results:\n")
print(summary(svm_tune))
cat(sprintf("Best cost: %g   Best gamma: %g\n",
            svm_tune$best.parameters$cost,
            svm_tune$best.parameters$gamma))

svm_pred <- predict(svm_tune$best.model, newdata =
as.data.frame(X_test))

svm_res <- eval_model(svm_pred, y_test, "SVM (radial)")


# random forest
p      <- ncol(X_train)
m_vals <- c(2, 4, 7, 10, 13)

rf_cv_acc <- sapply(m_vals, function(m) {
  fold_acc <- sapply(1:n_folds, function(f) {
    trn  <- wine_train[fold_id != f, ]
    tst  <- wine_train[fold_id == f, ]
    fit  <- randomForest(Class ~ ., data = trn,
                         mtry = m, ntree = 200, importance =
FALSE)
    pred <- predict(fit, newdata = tst)
    mean(pred == tst$Class)
  })
  mean(fold_acc)
})

names(rf_cv_acc) <- paste0("mtry=", m_vals)
cat("\nRandom Forest CV accuracies:\n")
print(round(rf_cv_acc, 4))

best_mtry <- m_vals[which.max(rf_cv_acc)]
cat(sprintf("Best mtry: %d\n", best_mtry))

# final RF model
```

```
rf_fit <- randomForest(
  Class ~ .,
  data       = wine_train,
  mtry       = best_mtry,
  ntree      = 500,
  importance = TRUE
)

rf_pred <- predict(rf_fit, newdata = wine_test)
rf_res  <- eval_model(rf_pred, y_test, "Random Forest")

# variable importance
varImpPlot(rf_fit, main = "Random Forest: Variable Importance",
pch = 19)

# model comparison
model_names <- c("LDA", paste0("KNN (k=", best_k, ")"), "SVM
(radial)", "Random Forest")
test_accs   <- c(lda_res$accuracy, knn_res$accuracy,
svm_res$accuracy, rf_res$accuracy)

models_ordered <- data.frame(Model = model_names, Test.Accuracy
= test_accs)
models_ordered <-
models_ordered[order(models_ordered$Test.Accuracy, decreasing =
TRUE), ]

cat("\nModels Ordered by Accuracy\n")
print(models_ordered, row.names = FALSE, digits = 4)

bp <- barplot(
  models_ordered$Test.Accuracy,
  names.arg = models_ordered$Model,
  col       = c("blue", "red", "green",
"orange")[seq_along(model_names)],
  ylim      = c(0, 1.15),
  main      = "Model Comparison: Test Set Accuracy",
  ylab      = "Test accuracy",
  las       = 2,
  cex.names = 0.85
)
```

```
text(
  x      = bp,
  y      = models_ordered$Test.Accuracy + 0.04,
  labels = sprintf("%.4f", models_ordered$Test.Accuracy),
  cex    = 0.85
)


# VISUAL CONFUSION MATRICES (heatmaps)

if (!require("ggplot2", quietly = TRUE)) {
  install.packages("ggplot2")
  library(ggplot2)
}

# function to plot confusion matrix heatmap
plot_cm <- function(cm, model_name) {
  cm_df <- as.data.frame(cm)
  colnames(cm_df) <- c("Predicted", "Actual", "Freq")

  # set fill to NA for cells with freq < 1 (no background)
  cm_df$Fill <- ifelse(cm_df$Freq < 1, NA,
as.character(cm_df$Actual))

  # class colors: 1=blue, 2=red, 3=green
  class_cols <- c("1" = "blue", "2" = "red", "3" = "green")

  # fill colors based on actual class (diagonal = class colors)
  fill_vals <- setNames(
    c("blue", "red", "green"),
    c("1", "2", "3")
  )

  ggplot(cm_df, aes(x = Predicted, y = Actual, fill = Fill)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Freq), color = "black", size = 6,
fontface = "bold") +
    scale_fill_manual(values = fill_vals, na.value = "white",
guide = "none") +
    scale_x_discrete(drop = FALSE) +
    scale_y_discrete(drop = FALSE) +
```

```
    labs(
      title = paste("Confusion Matrix:", model_name),
      x = "Predicted Class",
      y = "Actual Class"
    ) +
    theme_minimal() +
    theme(
      plot.title   = element_text(hjust = 0.5, size = 14, face =
"bold"),
      axis.text    = element_text(size = 12),
      axis.title   = element_text(size = 12),
      axis.text.x  = element_text(color =
class_cols[levels(cm_df$Predicted)], face = "bold"),
      axis.text.y  = element_text(color =
class_cols[levels(cm_df$Actual)], face = "bold")
    )
}

# plot confusion matrices for all models
par(mfrow = c(2, 2), mar = c(4, 4, 3, 2))

plot_cm(lda_res$cm, "LDA")
plot_cm(knn_res$cm, paste0("KNN (k=", best_k, ")"))
plot_cm(svm_res$cm, "SVM (radial)")
plot_cm(rf_res$cm, "Random Forest")

par(mfrow = c(1, 1))

# function to plot confusion matrix heatmap
plot_cm <- function(cm, model_name) {
  cm_df <- as.data.frame(cm)
  colnames(cm_df) <- c("Predicted", "Actual", "Freq")

  # if freq < 1: white background
  cm_df$Fill <- ifelse(cm_df$Freq < 1, NA,
as.character(cm_df$Actual))

  class_cols <- c("1" = "blue", "2" = "red", "3" = "green")
```

```
  fill_vals <- setNames(
    c("blue", "red", "green"),
    c("1", "2", "3")
  )

  p <- ggplot(cm_df, aes(x = Predicted, y = Actual, fill =
Fill)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Freq), color = "black", size = 6,
fontface = "bold") +
    scale_fill_manual(values = fill_vals, na.value = "white",
guide = "none") +
    scale_x_discrete(drop = FALSE) +
    scale_y_discrete(drop = FALSE) +
    labs(
      title = paste("Confusion Matrix:", model_name),
      x = "Predicted Class",
      y = "Actual Class"
    ) +
    theme_minimal() +
    theme(
      plot.title   = element_text(hjust = 0.5, size = 14, face =
"bold"),
      axis.text    = element_text(size = 12),
      axis.title   = element_text(size = 12),
      axis.text.x  = element_text(color =
class_cols[levels(cm_df$Predicted)], face = "bold"),
      axis.text.y  = element_text(color =
class_cols[levels(cm_df$Actual)], face = "bold")
    )

  return(p)
```

```
}

print(plot_cm(rf_res$cm, "LDA / SVM / Random Forest"))
print(plot_cm(knn_res$cm, paste0("KNN (k=", best_k, ")")))
```