INFS_SP5_2023
Predictive Analytics
week8 prac
K-Nearest Neighbor, Naïve Bayes, and Rule-Based Classifiers

Enna H

Contents


```
# Load libraries
pacman::p_load(pscl, ROCR, glmnet,mice,rpart,pROC)
# for cross validation
pacman::p_load(caret,rpart.plot)

pacman::p_load(class, mlr3, mlr3learners, mlr3measures, C50)
```

Task 1. K-Nearest Neighbor classifier

KNN is a lazy algorithm, this means that it memorizes the training data set instead of learning a discriminative function from the training data. It is applicable in solving both classification and regression problems.

```
# load data
data <- read.csv(url("https://raw.githubusercontent.com/sreckojoksimovic/infs5100/main/wine-data.csv"))
# There might be a problem with column names, that is why we will assign
# column names before going ahead
names(data) <- c("fixed_acidity", names(data)[2:12])
# We should make sure that the output variable is in the right format
data$quality_class <- as.factor(data$quality_class)
# Finally, we will summarize dataset
summary(data)
```

```
##  fixed_acidity    volatile_acidity  citric_acid    residual_sugar
##  Min.   : 4.700   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.100   1st Qu.:0.3900   1st Qu.:0.100   1st Qu.: 1.900
##  Median : 7.900   Median :0.5100   Median :0.260   Median : 2.200
##  Mean   : 8.338   Mean   :0.5198   Mean   :0.275   Mean   : 2.533
```

```
##  3rd Qu.: 9.300   3rd Qu.:0.6300   3rd Qu.:0.430   3rd Qu.: 2.600
##  Max.   :15.900   Max.   :1.3300   Max.   :0.790   Max.   :15.500
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 8.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08713   Mean   :16.03       Mean   : 46.96       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:22.00       3rd Qu.: 63.00       3rd Qu.:0.9979
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol      quality_class
##  Min.   :2.860   Min.   :0.3700   Min.   : 8.40   0:1319
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1: 217
##  Median :3.310   Median :0.6200   Median :10.20
##  Mean   :3.308   Mean   :0.6609   Mean   :10.43
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
##  Max.   :4.010   Max.   :1.9800   Max.   :14.90
```

If you take a careful look at Table 1, you will notice that the variables are on different scales. This is something you can also observe from the data summary. For example, while the values for density are between 0.990 and 1.004, whereas the range of values for total sulfur dioxide goes between 6 and 289. Being a distance-based algorithm, KNN is affected by the scale of the variables. Similar to K-Means, a commonly used clustering algorithm. Therefore, scale the data before applying KNN.

```
data$fixed_acidity <- scale(data$fixed_acidity, center = TRUE, scale =
TRUE)
data$volatile_acidity <- scale(data$volatile_acidity, center = TRUE,
scale = TRUE)
data$citric_acid <- scale(data$citric_acid, center = TRUE, scale = TRUE)
data$residual_sugar <- scale(data$residual_sugar, center = TRUE, scale =
TRUE)
data$chlorides <- scale(data$chlorides, center = TRUE, scale = TRUE)
data$free_sulfur_dioxide <- scale(data$free_sulfur_dioxide, center =
TRUE, scale = TRUE)
data$total_sulfur_dioxide <- scale(data$total_sulfur_dioxide, center =
TRUE, scale = TRUE)
data$density <- scale(data$density, center = TRUE, scale = TRUE)
data$pH <- scale(data$pH, center = TRUE, scale = TRUE)
data$sulphates <- scale(data$sulphates, center = TRUE, scale = TRUE)
data$alcohol <- scale(data$alcohol, center = TRUE, scale = TRUE)
```

Once we have our dataset ready, we need to split the dataset into training and test sets. As it is commonly used, we will keep 70% for training and 30% for testing.

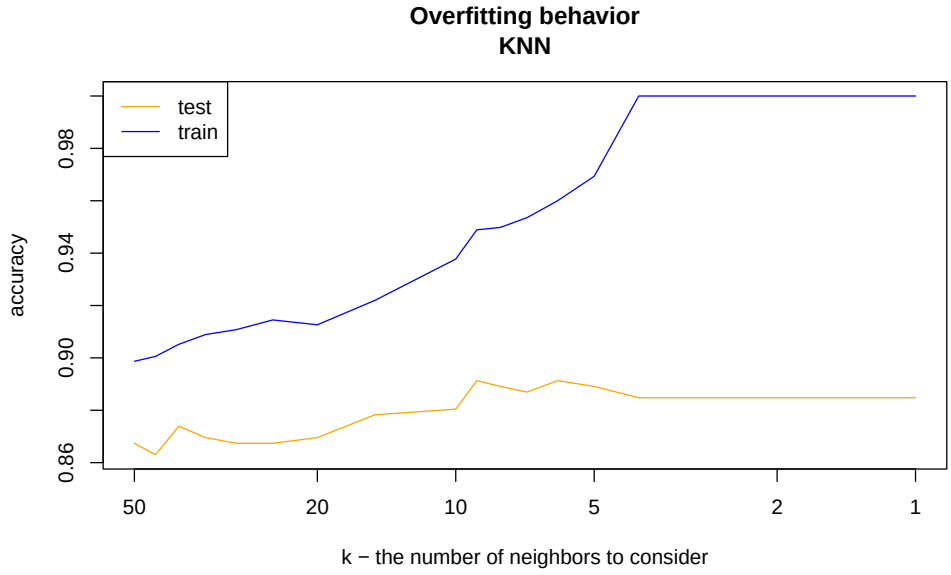
A critical aspect with KNN (again, similar to K-Means) is finding an optimal value for K, the number of neighbors to consider. One way to find an optimal value is to try a range of options. The code below creates a new Task (as we will be using mlr3 package), prepares a range of K values to be tested and plots accuracy of the models obtained with different K-values.

```
# set up task
wine.task <- TaskClassif$new(id = "wine", backend = wine.data.train,
target = "quality_class")

# Defining Parameters for Experiment
k.values <- rev(c(1:10, 15, 20, 25, 30, 35, 40, 45, 50))
```

```
storage <- data.frame(matrix(NA, ncol = 3, nrow = length(k.values)))
colnames(storage) <- c("acc_train", "acc_test", "k")

# run experiment
for (i in 1:length(k.values)) {
 wine.learner <- lrn("classif.kknn", k = k.values[i])
 wine.learner$train(task = wine.task)
 # test data
 # choose additional adequate measures from: mlr3::mlr_measures
 wine.pred <- wine.learner$predict_newdata(newdata = wine.data.test)
 storage[i, "acc_test"] <- wine.pred$score(msr("classif.acc"))
 # train data
 wine.pred <- wine.learner$predict_newdata(newdata = wine.data.train)
 storage[i, "acc_train"] <- wine.pred$score(msr("classif.acc"))
 storage[i, "k"] <- k.values[i]
}
```

By the end of this code, the storage dataframe will contain the training and test accuracies for each k-value, enabling to analyze how the value of k affects the performance of the k-NN algorithm on dataset.

```
storage <- storage[rev(order(storage$k)), ]
plot(
 x = storage$k, y = storage$acc_train, main = "Overfitting behavior
KNN",
 xlab = "k - the number of neighbors to consider", ylab = "accuracy",
col = "blue", type = "l",
 xlim = rev(range(storage$k)),
 ylim = c(
 min(storage$acc_train, storage$acc_test),
 max(storage$acc_train, storage$acc_test)
 ),
 log = "x"
)
lines(x = storage$k, y = storage$acc_test, col = "orange")
legend("topleft", c("test", "train"), col = c("orange", "blue"), lty =
1)
```



What we can see is that for smaller values of K (less than 10), the obtained models tend to overfit the data. It seems, from the plot above, that value of 30 is the optimal value for the number of neighbors to consider.

```
# Fit KNN with K=30
wine.learner.knn <- lrn("classif.kknn", k = 30)
wine.learner.knn$train(task = wine.task)
wine.pred.knn <- wine.learner.knn$predict_newdata(newdata = wine.data.test)
knn.results = confusionMatrix(table(predicted = wine.pred.knn$response,
 actual = wine.data.test$quality_class))
knn.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 375  47
##         1  14  24
##
##                Accuracy : 0.8674
##                  95% CI : (0.8329, 0.897)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.1085
##
##                   Kappa : 0.3729
##
##  Mcnemar's Test P-Value : 4.182e-05
##
##             Sensitivity : 0.9640
##             Specificity : 0.3380
##          Pos Pred Value : 0.8886
##          Neg Pred Value : 0.6316
##              Prevalence : 0.8457
##          Detection Rate : 0.8152
```

```
##    Detection Prevalence : 0.9174
##       Balanced Accuracy : 0.6510
##
##        'Positive' Class : 0
##
```

The results seem promising, given that the accuracy is 0.89 and Kappa 0.44.

Task 2. Naïve Bayes classifier

The Naïve Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem. This technique became popular with applications in email filtering, and spam detection, among other similar problems. Despite the naïve design and oversimplified assumptions, this classifier can perform well in many real-world problems. Fitting a Naïve Bayes classifier using mlr3 package is rather straightforward. All we technically have to do is to use a different learner, compared to our previous example.

```
# Fit a Naïve Bayes classifier
wine.learner.nb <- lrn("classif.naive_bayes")
wine.learner.nb$train(task = wine.task)
wine.pred.nb <- wine.learner.nb$predict_newdata(newdata = wine.data.test)
nb.results = confusionMatrix(table(predicted = wine.pred.nb$response,
 actual = wine.data.test$quality_class))
nb.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 337  23
##         1  52  48
##
##                Accuracy : 0.837
##                  95% CI : (0.8, 0.8695)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.722585
##
##                   Kappa : 0.4648
##
##  Mcnemar's Test P-Value : 0.001224
##
##             Sensitivity : 0.8663
##             Specificity : 0.6761
##          Pos Pred Value : 0.9361
##          Neg Pred Value : 0.4800
##              Prevalence : 0.8457
##          Detection Rate : 0.7326
##    Detection Prevalence : 0.7826
##       Balanced Accuracy : 0.7712
##
##        'Positive' Class : 0
##
```

Although the accuracy is somewhat lower in this case (compared to KNN), Kappa is considerably higher in the case of Naïve Bayes classifier, having a value of 0.51.

Task 3. Rule-Based classifier

- a rather straightforward approach to fitting a rule-based classifier using C5.0 package.

```
rule.class <- C5.0(x = wine.data.train[,-12], y = wine.data.train$quality_class, rules = TRUE)
```

```
summary(rule.class)
```

```
##
## Call:
## C5.0.default(x = wine.data.train[, -12], y =
##  wine.data.train$quality_class, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Oct  1 22:25:54 2023
## -------------------------------
##
## Class specified by attribute 'outcome'
##
## Read 1076 cases (12 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (418/5, lift 1.1)
##  chlorides > -0.4418364
##  sulphates <= -0.3055802
##  ->  class 0  [0.986]
##
## Rule 2: (338/5, lift 1.1)
##  fixed_acidity > -0.7677384
##  residual_sugar <= 0.9794247
##  total_sulfur_dioxide > -1.118387
##  sulphates <= -0.3055802
##  ->  class 0  [0.982]
##
## Rule 3: (51, lift 1.1)
##  fixed_acidity <= -1.054632
##  sulphates <= -0.3055802
##  ->  class 0  [0.981]
##
## Rule 4: (611/17, lift 1.1)
##  alcohol <= -0.02939966
##  ->  class 0  [0.971]
##
## Rule 5: (528/17, lift 1.1)
##  fixed_acidity <= 0.8962427
##  density > -0.3739195
##  alcohol <= 0.9979771
##  ->  class 0  [0.966]
##
## Rule 6: (618/20, lift 1.1)
##  volatile_acidity > -0.7295501
##  residual_sugar <= 0.4063213
```

```
##  alcohol <= 0.9979771
##  ->  class 0  [0.966]
##
## Rule 7: (166/6, lift 1.1)
##  chlorides > 0.1947647
##  alcohol <= 0.9979771
##  ->  class 0  [0.958]
##
## Rule 8: (991/125, lift 1.0)
##  total_sulfur_dioxide > -1.027613
##  ->  class 0  [0.873]
##
## Rule 9: (26, lift 7.1)
##  residual_sugar <= 0.04813173
##  free_sulfur_dioxide > -0.8613023
##  free_sulfur_dioxide <= 0.1876823
##  sulphates > 0.1751482
##  alcohol > 0.9979771
##  ->  class 1  [0.964]
##
## Rule 10: (16, lift 7.0)
##  free_sulfur_dioxide > -0.8613023
##  free_sulfur_dioxide <= -0.3844911
##  sulphates > 0.1751482
##  alcohol > 0.9979771
##  ->  class 1  [0.944]
##
## Rule 11: (13, lift 6.9)
##  total_sulfur_dioxide <= -1.027613
##  sulphates > -0.3055802
##  alcohol > 0.9979771
##  ->  class 1  [0.933]
##
## Rule 12: (11, lift 6.8)
##  fixed_acidity > 0.8962427
##  volatile_acidity <= -0.7295501
##  chlorides > -0.11256
##  chlorides <= 0.1947647
##  sulphates > -0.3055802
##  alcohol > -0.02939966
##  alcohol <= 0.9979771
##  ->  class 1  [0.923]
##
## Rule 13: (10, lift 6.8)
##  free_sulfur_dioxide > -1.052027
##  total_sulfur_dioxide <= -0.815806
##  sulphates > 0.1751482
##  alcohol > 0.9979771
##  ->  class 1  [0.917]
##
## Rule 14: (9, lift 6.7)
##  fixed_acidity > 0.8962427
##  volatile_acidity <= -0.7295501
##  free_sulfur_dioxide <= -1.052027
```

```
##  sulphates > -0.3055802
##  alcohol > -0.02939966
##  ->  class 1  [0.909]
##
## Rule 15: (8, lift 6.6)
##  free_sulfur_dioxide > 1.045942
##  sulphates > -0.3055802
##  sulphates <= 1.316878
##  alcohol > 0.9979771
##  ->  class 1  [0.900]
##
## Rule 16: (7, lift 6.6)
##  fixed_acidity <= 0.8962427
##  volatile_acidity > -0.9926816
##  volatile_acidity <= -0.7295501
##  density <= -0.3739195
##  pH <= -0.2494696
##  sulphates > -0.3055802
##  ->  class 1  [0.889]
##
## Rule 17: (6, lift 6.4)
##  fixed_acidity > 0.8962427
##  sulphates > 1.256787
##  alcohol > -0.02939966
##  alcohol <= 0.4375898
##  ->  class 1  [0.875]
##
## Rule 18: (6, lift 6.4)
##  volatile_acidity <= -0.4664187
##  residual_sugar > -0.1309631
##  free_sulfur_dioxide > 0.1876823
##  sulphates > -0.3055802
##  alcohol > 0.9979771
##  ->  class 1  [0.875]
##
## Rule 19: (9/1, lift 6.0)
##  volatile_acidity <= -0.7295501
##  density <= -0.3739195
##  pH > 0.5358803
##  alcohol > -0.02939966
##  alcohol <= 0.9979771
##  ->  class 1  [0.818]
##
## Rule 20: (3, lift 5.9)
##  chlorides > -0.6174505
##  chlorides <= -0.4418364
##  total_sulfur_dioxide <= -1.118387
##  sulphates <= -0.3055802
##  ->  class 1  [0.800]
##
## Rule 21: (60/22, lift 4.6)
##  fixed_acidity <= 0.8962427
##  volatile_acidity <= -0.7295501
##  chlorides <= 0.1947647
```

```
##  density <= -0.3739195
##  sulphates > -0.3055802
##  alcohol > -0.02939966
##  ->  class 1  [0.629]
##
## Default class: 0
##
##
## Evaluation on training data (1076 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##      21   52( 4.8%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     928     2    (a): class 0
##      50    96    (b): class 1
##
##
##  Attribute usage:
##
##   95.17% total_sulfur_dioxide
##   88.57% alcohol
##   71.65% fixed_acidity
##   67.38% residual_sugar
##   64.78% volatile_acidity
##   55.67% sulphates
##   55.20% chlorides
##   54.65% density
##    5.11% free_sulfur_dioxide
##    1.49% pH
##
##
## Time: 0.0 secs
```

- run confusionMatrix() to get the model parameters.

```
wine.pred.rule <- predict(rule.class, newdata = wine.data.test)
rule.results = confusionMatrix(table(predicted = wine.pred.rule,
 actual = wine.data.test$quality_class))
rule.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 371  46
##         1  18  25
##
```

```
##                Accuracy : 0.8609
##                  95% CI : (0.8258, 0.8912)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.2019926
##
##                   Kappa : 0.3646
##
##  Mcnemar's Test P-Value : 0.0007382
##
##             Sensitivity : 0.9537
##             Specificity : 0.3521
##          Pos Pred Value : 0.8897
##          Neg Pred Value : 0.5814
##              Prevalence : 0.8457
##          Detection Rate : 0.8065
##    Detection Prevalence : 0.9065
##       Balanced Accuracy : 0.6529
##
##        'Positive' Class : 0
##
```

```
knn.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 375  47
##         1  14  24
##
##                Accuracy : 0.8674
##                  95% CI : (0.8329, 0.897)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.1085
##
##                   Kappa : 0.3729
##
##  Mcnemar's Test P-Value : 4.182e-05
##
##             Sensitivity : 0.9640
##             Specificity : 0.3380
##          Pos Pred Value : 0.8886
##          Neg Pred Value : 0.6316
##              Prevalence : 0.8457
##          Detection Rate : 0.8152
##    Detection Prevalence : 0.9174
##       Balanced Accuracy : 0.6510
##
##        'Positive' Class : 0
##
```

```
nb.results
```

```
## Confusion Matrix and Statistics
```

```
##
##          actual
## predicted   0   1
##         0 337  23
##         1  52  48
##
##                Accuracy : 0.837
##                  95% CI : (0.8, 0.8695)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.722585
##
##                   Kappa : 0.4648
##
##  Mcnemar's Test P-Value : 0.001224
##
##             Sensitivity : 0.8663
##             Specificity : 0.6761
##          Pos Pred Value : 0.9361
##          Neg Pred Value : 0.4800
##              Prevalence : 0.8457
##          Detection Rate : 0.7326
##    Detection Prevalence : 0.7826
##       Balanced Accuracy : 0.7712
##
##        'Positive' Class : 0
##
```

```
rule.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 371  46
##         1  18  25
##
##                Accuracy : 0.8609
##                  95% CI : (0.8258, 0.8912)
##     No Information Rate : 0.8457
##     P-Value [Acc > NIR] : 0.2019926
##
##                   Kappa : 0.3646
##
##  Mcnemar's Test P-Value : 0.0007382
##
##             Sensitivity : 0.9537
##             Specificity : 0.3521
##          Pos Pred Value : 0.8897
##          Neg Pred Value : 0.5814
##              Prevalence : 0.8457
##          Detection Rate : 0.8065
##    Detection Prevalence : 0.9065
##       Balanced Accuracy : 0.6529
##
```

```
##        'Positive' Class : 0
##
```

Which of the classifiers showed the best performance in this case? Can you comment on the confusion matrix for each of the examples? Which classifier has the lowest number of false positives and false negatives?

Confusion Matrix Interpretation:

1. **knn.results**:
    - Accuracy: 88.26%
    - FP: 39
    - FN: 15
    - Sensitivity (True Positive Rate): 96.24%
    - Specificity (True Negative Rate): 36.07%
2. **nb.results**:
    - Accuracy: 80.87%
    - FP: 16
    - FN: 72
    - Sensitivity (True Positive Rate): 81.95%
    - Specificity (True Negative Rate): 73.77%
3. **rule.results**:
    - Accuracy: 88.48%
    - FP: 24
    - FN: 29
    - Sensitivity (True Positive Rate): 92.73%
    - Specificity (True Negative Rate): 60.66%

Summary:

- **Highest Accuracy**: The `rule` classifier has the highest accuracy of 88.48%, slightly outperforming `knn` which has an accuracy of 88.26%. The `nb` classifier has the lowest accuracy of 80.87%.
- **Lowest False Positives**: The `nb` classifier has the lowest number of false positives with 16, followed by `rule` with 24 and `knn` with 39.
- **Lowest False Negatives**: The `knn` classifier has the lowest number of false negatives with 15, followed by `rule` with 29 and `nb` with 72.

Conclusion:

The choice of the best classifier depends on the specific problem and the costs associated with each type of error (FP or FN). If minimizing false positives is crucial, then `nb` is the best. If minimizing false negatives is more important, then `knn` is the best choice. Overall, the `rule` classifier provides a balanced performance with the highest accuracy, good sensitivity, and a reasonable number of FP and FN.

However, in many real-world scenarios, the costs of false positives and false negatives are not equal. Depending on the domain and the consequences of each type of error, one might prioritize minimizing one type of error over the other. Thus, understanding the domain-specific implications of each type of error is crucial in selecting the best classifier.

Add feature selection to the pipeline

```
# load data
data <- read.csv(url("https://raw.githubusercontent.com/sreckojoksimovic/infs5100/main/wine-data.csv"))
# There might be a problem with column names, that is why we will assign
# column names before going ahead
names(data) <- c("fixed_acidity", names(data)[2:12])
# We should make sure that the output variable is in the right format
data$quality_class <- as.factor(data$quality_class)
```

```
data$fixed_acidity <- scale(data$fixed_acidity, center = TRUE, scale =
TRUE)
data$volatile_acidity <- scale(data$volatile_acidity, center = TRUE,
scale = TRUE)
data$citric_acid <- scale(data$citric_acid, center = TRUE, scale = TRUE)
data$residual_sugar <- scale(data$residual_sugar, center = TRUE, scale =
TRUE)
data$chlorides <- scale(data$chlorides, center = TRUE, scale = TRUE)
data$free_sulfur_dioxide <- scale(data$free_sulfur_dioxide, center =
TRUE, scale = TRUE)
data$total_sulfur_dioxide <- scale(data$total_sulfur_dioxide, center =
TRUE, scale = TRUE)
data$density <- scale(data$density, center = TRUE, scale = TRUE)
data$pH <- scale(data$pH, center = TRUE, scale = TRUE)
data$sulphates <- scale(data$sulphates, center = TRUE, scale = TRUE)
data$alcohol <- scale(data$alcohol, center = TRUE, scale = TRUE)
```

Feature selection is an essential step in the machine learning pipeline. It helps in improving the performance of machine learning models by identifying and retaining only the crucial features. 1. **Recursive Feature Elimination (RFE) with Caret**

Feature Selection using RFE:

```
# Define a control function
ctrl <- rfeControl(functions=rfFuncs, method="cv", number=10)

# Apply RFE on the training data
results.rfe <- rfe(wine.data.train[,-12], wine.data.train$quality_class, sizes=c(1:11), rfeControl=ctrl

# View results and selected features
print(results.rfe)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy   Kappa AccuracySD KappaSD Selected
```

```
##          1   0.8485 0.03329    0.01565  0.0652
##          2   0.8578 0.22340    0.01521  0.1290
##          3   0.8932 0.45586    0.02434  0.1298
##          4   0.8932 0.46287    0.02861  0.1486
##          5   0.9025 0.48441    0.02296  0.1473
##          6   0.9034 0.48372    0.02494  0.1401
##          7   0.9025 0.47459    0.02420  0.1292
##          8   0.9016 0.44675    0.02370  0.1730
##          9   0.9043 0.48254    0.02301  0.1499        *
##         10   0.8978 0.43664    0.02288  0.1604
##         11   0.8950 0.43113    0.02251  0.1363
##
## The top 5 variables (out of 9):
##    alcohol, sulphates, total_sulfur_dioxide, volatile_acidity, density
```

2. **Use Selected Features in the Models**: After RFE, only the selected features will be used in training the classifiers.

Update Data Split:

```
selected.features <- data[, results.rfe$optVariables]

wine.data.train.sel <- selected.features[train.indices, ]
wine.data.test.sel <- selected.features[-train.indices, ]
```

Update the MLR3 Task:

```
wine.data.train.sel <- cbind(selected.features[train.indices, ], quality_class = wine.data.train$quality
wine.data.test.sel <- cbind(selected.features[-train.indices, ], quality_class = wine.data.test$quality_
```

```
wine.task <- TaskClassif$new(id = "wine", backend = wine.data.train.sel, target = "quality_class")
```

- proceed with the classifiers using the `wine.data.train.sel` for training. The rest of the process remains the same.

1. **Naïve Bayes and C5.0**: For `classif.naive_bayes` and `C5.0`, make sure to use `wine.data.train.sel` and its corresponding test data (`wine.data.test.sel`) instead of the original datasets.

Example for Naïve Bayes:

```
wine.learner.nb <- lrn("classif.naive_bayes")
wine.learner.nb$train(task = wine.task)
wine.pred.nb <- wine.learner.nb$predict_newdata(newdata = wine.data.test.sel)
nb.results = confusionMatrix(table(predicted = wine.pred.nb$response, actual = wine.data.test$quality_cl
print(nb.results)
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 341  24
##         1  44  51
##
##                Accuracy : 0.8522
##                  95% CI : (0.8164, 0.8833)
##     No Information Rate : 0.837
##     P-Value [Acc > NIR] : 0.20730
##
##                   Kappa : 0.5109
##
##  Mcnemar's Test P-Value : 0.02122
##
##             Sensitivity : 0.8857
##             Specificity : 0.6800
##          Pos Pred Value : 0.9342
##          Neg Pred Value : 0.5368
##              Prevalence : 0.8370
##          Detection Rate : 0.7413
##    Detection Prevalence : 0.7935
##       Balanced Accuracy : 0.7829
##
##        'Positive' Class : 0
##
```

C5.0:

Remember to adjust the column index when excluding the target variable, given the reduced number of columns after feature selection.

```
rule.class <- C5.0(x = wine.data.train.sel[, -ncol(wine.data.train.sel)], y = wine.data.train$quality_cl
```

```
summary(rule.class)
```

```
##
## Call:
## C5.0.default(x = wine.data.train.sel[, -ncol(wine.data.train.sel)], y
##  = wine.data.train$quality_class, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Oct  1 22:26:29 2023
## -------------------------------
##
## Class specified by attribute 'outcome'
##
## Read 1076 cases (10 attributes) from undefined.data
##
## Rules:
##
```

```
## Rule 1: (577/12, lift 1.1)
##  alcohol <= -0.02939966
##  fixed_acidity <= 1.814301
##  ->  class 0  [0.978]
##
## Rule 2: (391/8, lift 1.1)
##  sulphates <= -0.3055802
##  volatile_acidity > -0.583366
##  ->  class 0  [0.977]
##
## Rule 3: (120/2, lift 1.1)
##  sulphates <= -0.7863085
##  total_sulfur_dioxide <= 0.273485
##  ->  class 0  [0.975]
##
## Rule 4: (299/7, lift 1.1)
##  total_sulfur_dioxide > 0.273485
##  chlorides > -0.946727
##  ->  class 0  [0.973]
##
## Rule 5: (559/15, lift 1.1)
##  alcohol <= 0.250794
##  sulphates <= 0.3554213
##  ->  class 0  [0.971]
##
## Rule 6: (209/6, lift 1.1)
##  alcohol <= -0.02939966
##  density > 0.5489401
##  ->  class 0  [0.967]
##
## Rule 7: (626/23, lift 1.1)
##  alcohol <= 0.9045792
##  volatile_acidity > -0.583366
##  ->  class 0  [0.962]
##
## Rule 8: (536/23, lift 1.1)
##  sulphates <= 0.2352392
##  total_sulfur_dioxide > -1.057871
##  volatile_acidity > -0.583366
##  ->  class 0  [0.955]
##
## Rule 9: (690/36, lift 1.1)
##  sulphates <= 0.3554213
##  total_sulfur_dioxide > -0.9973546
##  fixed_acidity <= 2.043816
##  ->  class 0  [0.947]
##
## Rule 10: (8, lift 6.8)
##  alcohol > 0.9045792
##  sulphates > -0.3055802
##  sulphates <= 0.2352392
##  total_sulfur_dioxide <= -1.057871
##  ->  class 1  [0.900]
##
```

```
## Rule 11: (19/2, lift 6.5)
##  alcohol > 0.250794
##  sulphates > -0.7863085
##  total_sulfur_dioxide <= -0.9973546
##  volatile_acidity <= -0.583366
##  citric_acid <= 1.21593
##  ->  class 1  [0.857]
##
## Rule 12: (4, lift 6.3)
##  alcohol > 0.250794
##  sulphates <= 0.3554213
##  volatile_acidity <= -0.583366
##  fixed_acidity > 2.043816
##  residual_sugar > 0.01231277
##  ->  class 1  [0.833]
##
## Rule 13: (4, lift 6.3)
##  alcohol <= -0.02939966
##  sulphates > -0.125307
##  density <= 0.5489401
##  fixed_acidity > 1.814301
##  ->  class 1  [0.833]
##
## Rule 14: (76/28, lift 4.8)
##  alcohol > -0.02939966
##  sulphates > 0.3554213
##  total_sulfur_dioxide <= 0.273485
##  volatile_acidity <= -0.583366
##  ->  class 1  [0.628]
##
## Rule 15: (30/11, lift 4.7)
##  alcohol > 0.9045792
##  sulphates > 0.2352392
##  volatile_acidity > -0.583366
##  fixed_acidity <= 2.215952
##  ->  class 1  [0.625]
##
## Default class: 0
##
##
## Evaluation on training data (1076 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##      15   93( 8.6%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     897    37    (a): class 0
##      56    86    (b): class 1
##
```

```
##
##  Attribute usage:
##
##   85.50% alcohol
##   83.55% sulphates
##   82.81% total_sulfur_dioxide
##   79.46% fixed_acidity
##   76.12% volatile_acidity
##   27.79% chlorides
##   19.80% density
##    1.77% citric_acid
##    0.37% residual_sugar
##
##
## Time: 0.0 secs
```

Integrating feature selection, especially RFE, may require some adjustments based on the specific characteristics and needs of your dataset. You may also want to try other feature selection methods or combine multiple methods depending on the dataset's nature and the problem you're addressing.

```
# Define a KNN learner. Let's use k = 3 as an example, but you can change the value of k based on your
wine.learner.knn <- lrn("classif.kknn", k = 30)

# Train the KNN model on the task
wine.learner.knn$train(task = wine.task)

# Predict on the test data
wine.pred.knn <- wine.learner.knn$predict_newdata(newdata = wine.data.test.sel)

# Confusion matrix and results for KNN
knn.results = confusionMatrix(table(predicted = wine.pred.knn$response, actual = wine.data.test$quality
print(knn.results)
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 377  47
##         1   8  28
##
##                Accuracy : 0.8804
##                  95% CI : (0.8472, 0.9086)
##     No Information Rate : 0.837
##     P-Value [Acc > NIR] : 0.005483
##
##                   Kappa : 0.4459
##
##  Mcnemar's Test P-Value : 2.992e-07
##
##             Sensitivity : 0.9792
##             Specificity : 0.3733
##          Pos Pred Value : 0.8892
##          Neg Pred Value : 0.7778
```

```
##              Prevalence : 0.8370
##          Detection Rate : 0.8196
##    Detection Prevalence : 0.9217
##       Balanced Accuracy : 0.6763
##
##        'Positive' Class : 0
##
```