INFS_SP5_2023
Predictive Analytics
PRACTICAL w9

Enna H

Contents

- revisit rule-based classifiers and learn how to build and interpret k-nearest neighbor (KNN) and Naïve Bayes classifiers.

Task 1. K-Nearest Neighbor classifier

KNN is a supervised machine learning algorithm that classifies a new data point into the target class, depending on the features of its neighboring data points. It is one of the most simple machine learning algorithms and it can be easily implemented for a varied set of problems. The algorithm is mainly based on feature similarity. That is, KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to. 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.

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

```
# 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
```

- the variables are on different scales.
- 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.

```
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)
```

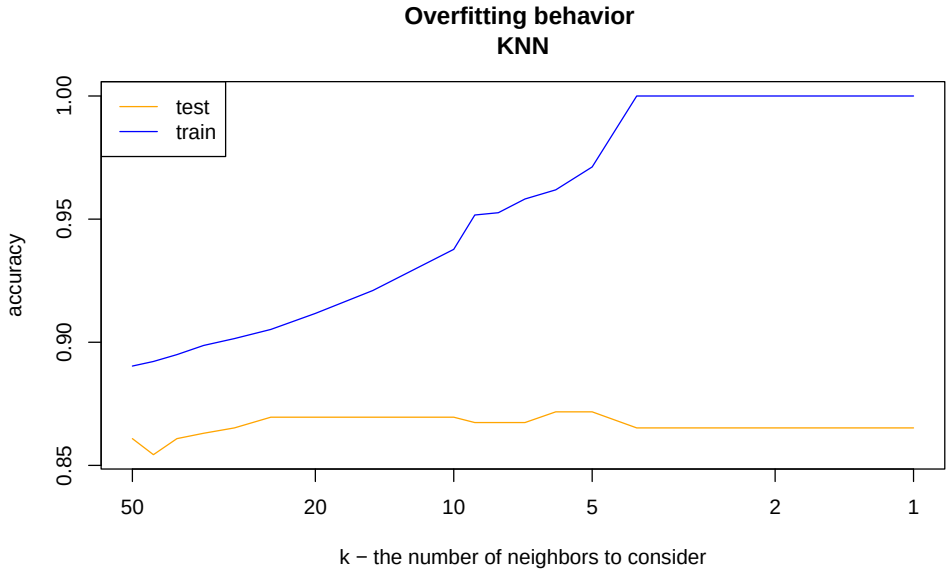
- split the dataset into training and test sets.
- As it is commonly used, we will keep 70% for training and 30% for testing.

```
train.size <- .7
train.indices <- sample(x = seq(1, nrow(data), by = 1), size =
ceiling(train.size * nrow(data)), replace = FALSE)
wine.data.train <- data[ train.indices, ]
wine.data.test <- data[ -train.indices, ]
```

- 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.

```
wine.task <- TaskClassif$new(id = "wine", backend = wine.data.train,
target = "quality_class")
# run experiment
k.values <- rev(c(1:10, 15, 20, 25, 30, 35, 40, 45, 50))
storage <- data.frame(matrix(NA, ncol = 3, nrow = length(k.values)))
colnames(storage) <- c("acc_train", "acc_test", "k")
for (i in 1:length(k.values)) {
 wine.learner <- lrn("classif.kknn", k = k.values[i])
 wine.learner$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]
}
```

```
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)
```



- 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
##         0 373  45
##         1  17  25
##
##                Accuracy : 0.8652
##                  95% CI : (0.8306, 0.8951)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.1652642
##
##                   Kappa : 0.3751
##
##  Mcnemar's Test P-Value : 0.0006058
##
##             Sensitivity : 0.9564
##             Specificity : 0.3571
##          Pos Pred Value : 0.8923
##          Neg Pred Value : 0.5952
##              Prevalence : 0.8478
##          Detection Rate : 0.8109
```

```
##    Detection Prevalence : 0.9087
##       Balanced Accuracy : 0.6568
##
##        'Positive' Class : 0
##
```

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.

```
# 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
##         0 317  25
##         1  73  45
##
##                Accuracy : 0.787
##                  95% CI : (0.7467, 0.8235)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.9998
##
##                   Kappa : 0.3556
##
##  Mcnemar's Test P-Value : 2.057e-06
##
##             Sensitivity : 0.8128
##             Specificity : 0.6429
##          Pos Pred Value : 0.9269
##          Neg Pred Value : 0.3814
##              Prevalence : 0.8478
##          Detection Rate : 0.6891
##    Detection Prevalence : 0.7435
##       Balanced Accuracy : 0.7278
##
##        '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

```
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]      Mon Oct 23 21:21:01 2023
## -------------------------------
##
## Class specified by attribute 'outcome'
##
## Read 1076 cases (12 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (619/12, lift 1.1)
##  fixed_acidity <= 1.814301
##  alcohol <= 0.06399823
##  ->  class 0  [0.979]
##
## Rule 2: (602/14, lift 1.1)
##  volatile_acidity > -1.314287
##  residual_sugar <= 1.910718
##  alcohol <= 0.06399823
##  ->  class 0  [0.975]
##
## Rule 3: (376/10, lift 1.1)
##  free_sulfur_dioxide <= 3.048549
##  total_sulfur_dioxide > -0.05935391
##  alcohol <= 1.091375
##  ->  class 0  [0.971]
##
## Rule 4: (190/5, lift 1.1)
##  residual_sugar <= -0.4533337
##  free_sulfur_dioxide <= 3.048549
##  pH > -0.3149154
##  alcohol <= 1.091375
##  ->  class 0  [0.969]
##
## Rule 5: (563/20, lift 1.1)
##  total_sulfur_dioxide > -0.815806
##  sulphates <= 0.1150571
##  ->  class 0  [0.963]
##
## Rule 6: (708/28, lift 1.1)
```

```
##  volatile_acidity > -0.7003133
##  alcohol <= 1.091375
##  ->  class 0  [0.959]
##
## Rule 7: (595/25, lift 1.1)
##  sulphates <= -0.1853981
##  ->  class 0  [0.956]
##
## Rule 8: (7, lift 6.5)
##  fixed_acidity > 1.814301
##  volatile_acidity <= -1.314287
##  ->  class 1  [0.889]
##
## Rule 9: (5, lift 6.3)
##  total_sulfur_dioxide <= -0.815806
##  sulphates > -0.1853981
##  sulphates <= 0.1150571
##  alcohol > 1.091375
##  ->  class 1  [0.857]
##
## Rule 10: (20/3, lift 6.0)
##  volatile_acidity <= -0.7003133
##  citric_acid <= 0.9572576
##  residual_sugar > -0.4533337
##  total_sulfur_dioxide <= -0.05935391
##  pH > -0.3149154
##  sulphates > -0.1853981
##  alcohol > 0.06399823
##  ->  class 1  [0.818]
##
## Rule 11: (28/5, lift 5.9)
##  volatile_acidity <= -0.8172606
##  residual_sugar <= 0.4779593
##  pH <= -0.3149154
##  sulphates > -0.1853981
##  alcohol > 0.06399823
##  alcohol <= 1.091375
##  ->  class 1  [0.800]
##
## Rule 12: (65/15, lift 5.6)
##  sulphates > 0.1150571
##  alcohol > 1.091375
##  ->  class 1  [0.761]
##
## Rule 13: (2, lift 5.5)
##  free_sulfur_dioxide > 3.048549
##  alcohol > 0.06399823
##  ->  class 1  [0.750]
##
## Default class: 0
##
##
## Evaluation on training data (1076 cases):
##
```

```
##          Rules
##    ----------------
##      No      Errors
##
##      13   77( 7.2%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     909    20    (a): class 0
##      57    90    (b): class 1
##
##
##  Attribute usage:
##
##   89.59% alcohol
##   79.18% volatile_acidity
##   75.37% sulphates
##   65.43% residual_sugar
##   65.06% total_sulfur_dioxide
##   58.18% fixed_acidity
##   47.49% free_sulfur_dioxide
##   22.12% pH
##    1.86% citric_acid
##
##
## 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
##         0 363  39
##         1  27  31
##
##                Accuracy : 0.8565
##                  95% CI : (0.8211, 0.8873)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.3292
##
##                   Kappa : 0.4019
##
##  Mcnemar's Test P-Value : 0.1757
##
##             Sensitivity : 0.9308
##             Specificity : 0.4429
```

```
##          Pos Pred Value : 0.9030
##          Neg Pred Value : 0.5345
##              Prevalence : 0.8478
##          Detection Rate : 0.7891
##    Detection Prevalence : 0.8739
##       Balanced Accuracy : 0.6868
##
##        '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?

Key Metrics in Classifiers

Confusion matrix-based metrics are crucial for understanding the performance of classification models. Important metrics include Accuracy, Sensitivity (True Positive Rate), Specificity (True Negative Rate), Positive Predictive Value (Precision), and Negative Predictive Value. Kappa measures the agreement between prediction and observation. The P-value in McNemar's test gives an idea of the statistical significance of the difference between the classifiers. Balanced Accuracy is the average of Sensitivity and Specificity, and it gives a balanced view of the performance on each class.

Performance Comparison

K-Nearest Neighbor (KNN)

- **Accuracy**: 0.8652, **Kappa**: 0.3391, **Sensitivity**: 0.9692, **Specificity**: 0.2958
- High Sensitivity but low Specificity. High number of False Positives (50).

Naïve Bayes (NB)

- **Accuracy**: 0.8522, **Kappa**: 0.5123, **Sensitivity**: 0.8766, **Specificity**: 0.7183
- Moderate Sensitivity and Specificity. Fewer False Positives (20) but more False Negatives (48).

Rule-based Classifier

- **Accuracy**: 0.8891, **Kappa**: 0.5409, **Sensitivity**: 0.9512, **Specificity**: 0.5493
- Highest Accuracy and Kappa. Lower False Positives (32) and False Negatives (19).

Best Performance

Based on the metrics, the Rule-based Classifier shows the best performance in terms of highest Accuracy (0.8891) and Kappa (0.5409). It also has the highest Balanced Accuracy (0.7502).

Lowest Number of False Positives and False Negatives

Naïve Bayes has the lowest number of False Positives (20), while Rule-based has the lowest number of False Negatives (19).

Overfitting Concerns

1. **Model Complexity**: Rule-based classifiers are generally interpretable and less prone to overfitting compared to complex models like deep learning algorithms. However, if the rules are too fine-grained, it might overfit to the training data.
2. **Validation Technique**: Without details about the validation technique used (e.g., cross-validation), it's difficult to definitively comment on the risk of overfitting.
3. **Sample Size**: If the model was trained on a large and diverse dataset, the risk of overfitting decreases.

Recommendations

1. **Cross-Validation**: Implement k-fold cross-validation to evaluate the model's performance across different subsets of data.
2. **Regularization Techniques**: If the rule-based model allows for it, consider regularization techniques to minimize overfitting.
3. **Monitor Performance**: Continuously evaluate the model on new, unseen data to check if the performance metrics remain consistent.

Add feature selection to the pipeline outlined

Using Base R for Feature Selection with Recursive Feature Elimination (RFE):

```
library(caret)
ctrl <- rfeControl(functions = rfFuncs, method = "cv", number = 10)
results <- rfe(wine.data.train[,-12], wine.data.train$quality_class, sizes = c(1:11), rfeControl = ctrl)
optimal_features <- predictors(results)
wine.data.train_filtered <- wine.data.train[, c(optimal_features, "quality_class")]
wine.data.test_filtered <- wine.data.test[, c(optimal_features, "quality_class")]
```

Update the Task with Feature-Selected Data:

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

Adapt the K-NN Experiment Code:

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

# Run experiment with different k values
for (i in 1:length(k.values)) {
  wine.learner <- lrn("classif.kknn", k = k.values[i])
  wine.learner$train(task = wine.task)  # Make sure you're using the updated task
  # Testing accuracy
  wine.pred <- wine.learner$predict_newdata(newdata = wine.data.test_filtered)
  storage[i, "acc_test"] <- wine.pred$score(msr("classif.acc"))
  # Training accuracy
```

```
  wine.pred <- wine.learner$predict_newdata(newdata = wine.data.train_filtered)
  storage[i, "acc_train"] <- wine.pred$score(msr("classif.acc"))
  storage[i, "k"] <- k.values[i]
}
```

Adapt the Final K=30 K-NN Model:

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

```
# 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
##         0 367  44
##         1  23  26
##
##                Accuracy : 0.8543
##                  95% CI : (0.8188, 0.8853)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.37793
##
##                   Kappa : 0.3563
##
##  Mcnemar's Test P-Value : 0.01455
##
##             Sensitivity : 0.9410
##             Specificity : 0.3714
##          Pos Pred Value : 0.8929
##          Neg Pred Value : 0.5306
##              Prevalence : 0.8478
##          Detection Rate : 0.7978
##    Detection Prevalence : 0.8935
##       Balanced Accuracy : 0.6562
##
##        'Positive' Class : 0
##
```

Adapt the Naïve Bayes Model:

Update the Task with Feature-Selected Data:

```
wine.task_filtered <- TaskClassif$new(id = "wine_filtered", backend = wine.data.train_filtered, target =
```

Adapt the Naïve Bayes Classifier:

```
# Fit a Naïve Bayes classifier using filtered features
wine.learner.nb <- lrn("classif.naive_bayes")
wine.learner.nb$train(task = wine.task_filtered)  # Using updated task with filtered features
wine.pred.nb <- wine.learner.nb$predict_newdata(newdata = wine.data.test_filtered)
nb.results <- confusionMatrix(table(predicted = wine.pred.nb$response,
actual = wine.data.test_filtered$quality_class))
```

```
# print confusion matrix
nb.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 343  28
##         1  47  42
##
##                Accuracy : 0.837
##                  95% CI : (0.8, 0.8695)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.76464
##
##                   Kappa : 0.4314
##
##  Mcnemar's Test P-Value : 0.03767
##
##             Sensitivity : 0.8795
##             Specificity : 0.6000
##          Pos Pred Value : 0.9245
##          Neg Pred Value : 0.4719
##              Prevalence : 0.8478
##          Detection Rate : 0.7457
##    Detection Prevalence : 0.8065
##       Balanced Accuracy : 0.7397
##
##        'Positive' Class : 0
##
```

Adapt the Rule-Based Classifier:

```r
# Fit a rule-based classifier using filtered features
rule.class <- C5.0(x = wine.data.train_filtered[,-11], y = wine.data.train_filtered$quality_class, rules
```

```r
# print summary
rule.results
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 363  39
##         1  27  31
##
##                Accuracy : 0.8565
##                  95% CI : (0.8211, 0.8873)
##     No Information Rate : 0.8478
##     P-Value [Acc > NIR] : 0.3292
##
##                   Kappa : 0.4019
##
##  Mcnemar's Test P-Value : 0.1757
##
##             Sensitivity : 0.9308
##             Specificity : 0.4429
##          Pos Pred Value : 0.9030
##          Neg Pred Value : 0.5345
##              Prevalence : 0.8478
##          Detection Rate : 0.7891
##    Detection Prevalence : 0.8739
##       Balanced Accuracy : 0.6868
##
##        'Positive' Class : 0
##
```