CSMDM16 Assignment

Deyan Dyankov (24885531)

21/10/2016

Problem #1 - Hierarchical Clustering

You are required to extract and provide some statistics about the dataset "teeth.csv" They should include the number of records, the number of attributes, the range and mean value of each attribute, and the histogram of for each attribute. You are required to apply hierarchical clustering to all the records in the dataset and to visualise the resulting dendrogram. The dendrogram should report the class labels of the data records at the leaf nodes. The generation of the dendrogram should be performed twice: with KNIME and with R. The corresponding KNIME workflow and the R code should be reported and discussed. In your coursework report, you should include:

- the statistics and discussion on them
- and image of the KNIME workflow and a presentation of relevant code configurations
- the dendrogram generated with KNIME
- the R code
- the dendrogram generated with R code

Solution in R

The *teeth.csv* dataset is comprised of data on teeth configuration among different animals. This presents an opportunity to divide animals in groups based on the configuration of their teeth.

We will be using some external libraries to aid the analysis.

```
library(dplyr) # general data cleaning and manipulation
library(ggplot2) # nicer plots
library(rpart) # needed for rpart()
library(readr) # needed for read_csv()
library(tidyr) # needed for spread() and gather()
library(knitr) # knitr functions
```

Let's read the data from *teeth.csv*.

```
# read the data file
data_filename <- "data/teeth.csv"
df <- read_csv(data_filename) # headers applied
```

This is how the data frame looks like:

```
head(df)
```

```
## # A tibble: 6 x 9
##            Animal TopInc BotInc TopCan BotCan TopPre BotPre TopMol BotMol
##             <chr>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
## 1       BROWN BAT      2      3      1      1      3      3      3      3
```

```
## 2            MOLE      3      2      1      0      3      3      3      3
## 3 SILVER HAIR BAT      2      3      1      1      2      3      3      3
## 4       PIGMY BAT      2      3      1      1      2      2      3      3
## 5       HOUSE BAT      2      3      1      1      1      2      3      3
## 6         RED BAT      1      3      1      1      2      2      3      3
```

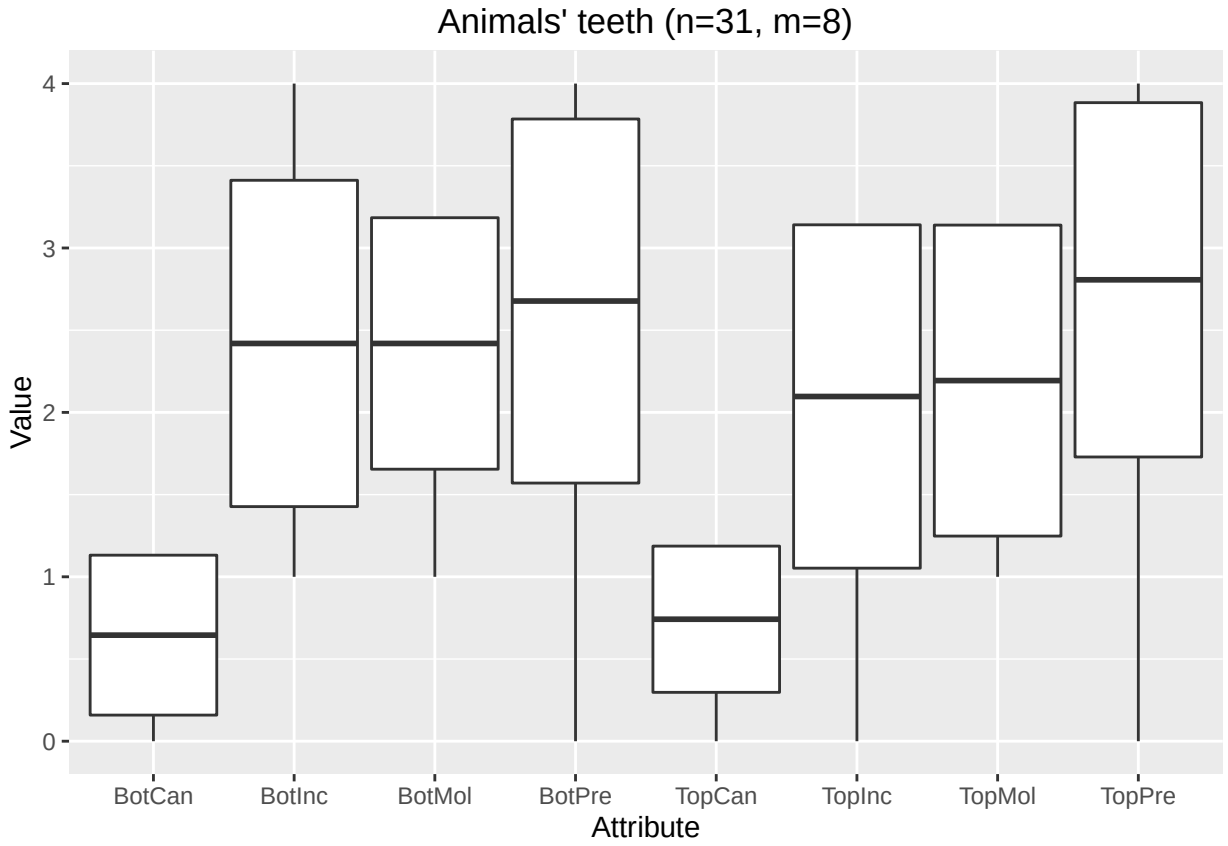
There are 31 rows and 8 columns in the dataset. Let's retrieve required statistics for each variable - min, mean, max values and standard deviation.

```
df.s <- select(df, -Animal) %>% # all columns except Animal
        gather() %>%
        group_by(key) %>%
        summarise(
                "mean" = mean(value),
                "min" = min(value),
                "max" = max(value),
                "lower" = mean(value) - sd(value),
                "upper" = mean(value) + sd(value),
                "sd" = sd(value)
        )
kable(df.s)
```

key	mean	min	max	lower	upper	sd
BotCan	0.6451613	0	1	0.1587878	1.131535	0.4863735
BotInc	2.4193548	1	4	1.4269103	3.411799	0.9924446
BotMol	2.4193548	1	3	1.6545371	3.184173	0.7648178
BotPre	2.6774194	0	4	1.5702579	3.784581	1.1071614
TopCan	0.7419355	0	1	0.2971328	1.186738	0.4448027
TopInc	2.0967742	0	3	1.0525891	3.140959	1.0441851
TopMol	2.1935484	1	3	1.2477029	3.139394	0.9458455
TopPre	2.8064516	0	4	1.7288198	3.884083	1.0776318

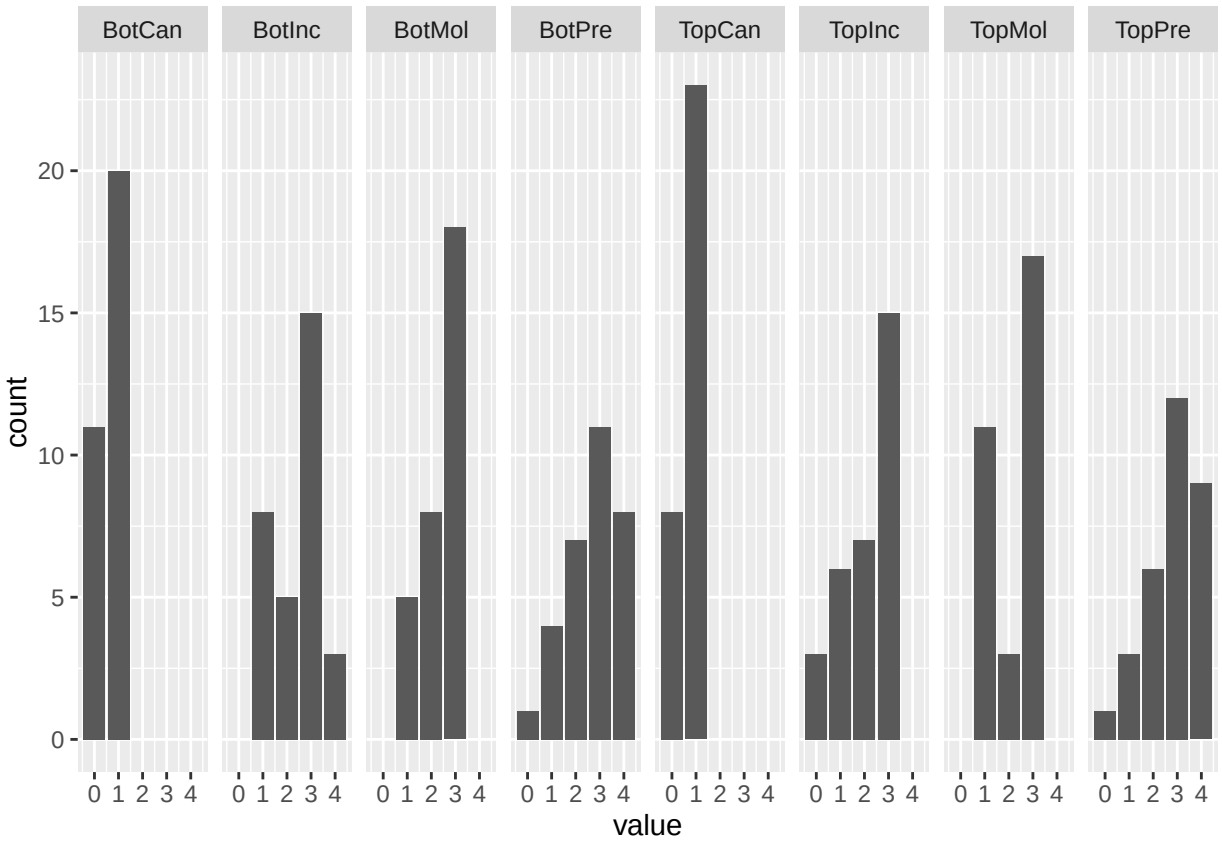
Now let's plot the information from the table above.

```
ggplot(df.s, aes(key)) +
        geom_boxplot(
                aes(lower = lower, ymin = min, middle = mean, ymax = max, upper = upper),
                stat = 'identity') +
        ggtitle(paste0("Animals' teeth (n=", nrow(df), ", m=", ncol(df)-1, ")")) +
        xlab("Attribute") +
        ylab("Value")
```



And here is a histogram of each and every attribute of the dataset.

```
hists <- gather(select(df, -Animal))
hists$key <- as.factor(hists$key)
ggplot(data = hists, aes(x = value)) +
        geom_bar() +
        facet_grid(. ~ key)
```



Proceeding with hierarchical clustering.

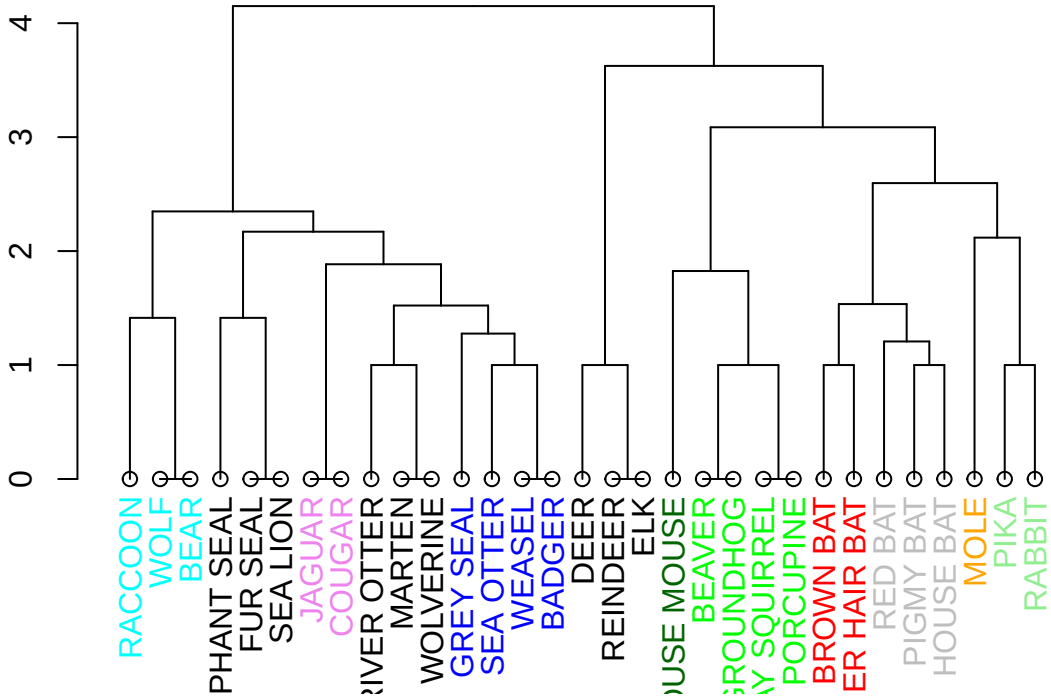
```
# assign row names to be the value of Animal column,
# this is later used in the dendrogram as labels
rownames(df) <- df$Animal

# hierarchical clustering
# the average method was chosen for best results
hc <- hclust(dist(select(df, -Animal)), method = "average")

# cut the tree and group animals
cluster_membership <- cutree(hc, 12)

# colour the dendrogram
label_colours <- c(
  'red', 'orange', 'grey', 'lightgreen', 'green',
  'darkgreen', 'cyan', 'black', 'blue', 'violet')
colour_label <- function(n) {
  if (is.leaf(n)) {
    a <- attributes(n)
    lc <- label_colours[cluster_membership[which(names(cluster_membership) == a$label)]]
    attr(n, "nodePar") <- c(a$nodePar, lab.col = lc)
  }
  return(n)
}
# apply the colouring function
cluster_dendrogram <- dendrapply(as.dendrogram(hc), colour_label)
```

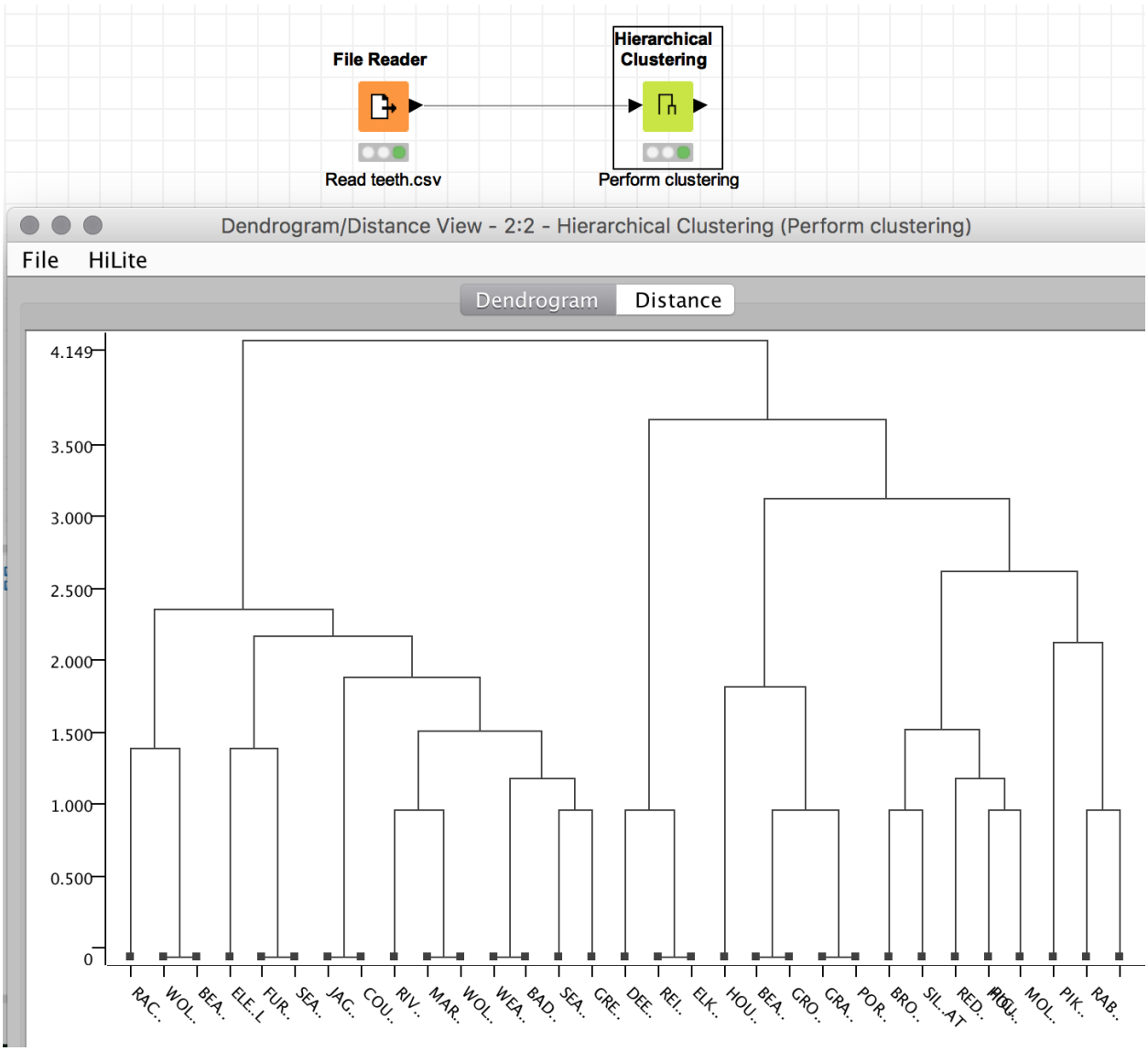
```r
# show the dendrogram with coloured groups of animals
plot(cluster_dendrogram)
```



Solution in KNIME

The solution in KNIME only requires hierarchical clustering and no exploratory analysis so the workflow is very simple. We first load the CSV file using a *FileReader* node where we provide the filename and set the "read row IDs" option. This makes the *FileReader* node set row names according to the first column of the dataset which in our case is *Animal*. Once the datafile has been read and columns have been set, we perform hierarchical clustering using the *HierarchicalClustering* node. We use the *Euclidian* distance function and set linkage type to $AVERAGE$ to get similar results to the clustering performed in R previously.

The image below shows the KNIME flow and the output of the hierarchical clustering.



This flow is exported as *csmdm_assignment_problem1.knar.knwf*.

Problem #2 - Classification

You are required to build and test a classification model for two datasets:

- https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data
- a dataset of your choice (excluding iris, wine, teeth)

A description of the wine dataset (including column headers) is provided in the corresponding file with extension ".names".

You can use any classification algorithm. You are required to carry out a comparison of the following four model evaluation methods for each of the two datasets using the same classification algorithm.

1. *resub*: resubstitution error method
2. *hold-out-10%*: hold-out method with 10% - 90% partition split

3. *xVal-10f*: 10-fold cross-validation method
4. *LOOCV*: leave-one-out cross-validation method

For each method you need to compute the average and standard deviation of the accuracy over 20 trials. The results, the KNIME workflows (with relevant node configurations) and the R code must be included and commented in the coursework report.

Solution in R - WINE

The wine dataset is comprised of different wines with their properties and *class* spelled out. This provides an opportunity to build a decisioon tree model that can allow us to predict the *class* of a wine given its properties.

```
# load the data files
wdf <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data")
# create a vector to use for column headers
wdf_names <- c('class', 'alcohol', 'malic.acid', 'ash',
               'alcalinity.of.ash', 'magnesium', 'total.phenols', 'flavanoids',
               'nonflavanoid.phenols', 'proanthocyanins', 'color.intensity',
               'hue', 'od280od315.of.dliuted.wines', 'proline')
# set column headers
names(wdf) <- wdf_names
# transform class from integer to a factor variable
wdf$class <- as.factor(wdf$class)
# show a summary of the data
str(wdf)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    177 obs. of  14 variables:
##  $ class                       : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ alcohol                     : num  13.2 13.2 14.4 13.2 14.2 ...
##  $ malic.acid                  : num  1.78 2.36 1.95 2.59 1.76 1.87 2.15 1.64 1.35 2.16 ...
##  $ ash                         : num  2.14 2.67 2.5 2.87 2.45 2.45 2.61 2.17 2.27 2.3 ...
##  $ alcalinity.of.ash           : num  11.2 18.6 16.8 21 15.2 14.6 17.6 14 16 18 ...
##  $ magnesium                   : int  100 101 113 118 112 96 121 97 98 105 ...
##  $ total.phenols               : num  2.65 2.8 3.85 2.8 3.27 2.5 2.6 2.8 2.98 2.95 ...
##  $ flavanoids                  : num  2.76 3.24 3.49 2.69 3.39 2.52 2.51 2.98 3.15 3.32 ...
##  $ nonflavanoid.phenols        : num  0.26 0.3 0.24 0.39 0.34 0.3 0.31 0.29 0.22 0.22 ...
##  $ proanthocyanins             : num  1.28 2.81 2.18 1.82 1.97 1.98 1.25 1.98 1.85 2.38 ...
##  $ color.intensity             : num  4.38 5.68 7.8 4.32 6.75 5.25 5.05 5.2 7.22 5.75 ...
##  $ hue                         : num  1.05 1.03 0.86 1.04 1.05 1.02 1.06 1.08 1.01 1.25 ...
##  $ od280od315.of.dliuted.wines: num  3.4 3.17 3.45 2.93 2.85 3.58 3.58 2.85 3.55 3.17 ...
##  $ proline                     : int  1050 1185 1480 735 1450 1290 1295 1045 1045 1510 ...
```

Now that we've loaded the data, we need to divide it into training and test sets using different sampling methods. Below are definitions of the functions that we use for sampling. All four sampling algorithms return a list of the form list("test" = test_data_frame, "train" = train_data_frame).

```
# implementation of the holdout method
myholdout <- function(x, pct, shuffle=TRUE) {
        if (shuffle)
                x <- x[sample(nrow(x), nrow(x)), ]
```

```
        test_index <- 1:floor(nrow(x) * pct)
        test <- x[test_index, ]
        train <- x[-test_index, ]
        result <- list("test" = test, "train" = train)
        return(result)
}
# resubstitution method uses holdout but returns
# a test set that is part of the training set
myresub <- function(x, pct) {
        holdout <- myholdout(x, pct, shuffle=FALSE)
        train <- holdout$train
        train <- train[sample(nrow(train), nrow(train)), ] # shuffle training set
        test <- train[1:nrow(holdout$test), ] # test set is part of training set
        result <- list("test" = test, "train" = train)
        return(result)
}

# implementation of 10-fold cross validation
my10fold <- function(x, k, shuffle=TRUE) {
        if (shuffle)
                x <- x[sample(nrow(x), nrow(x)), ]
        test_rows <- floor(nrow(x) * 0.10)
        end_idx <- k * test_rows
        start_idx <- end_idx - test_rows + 1
        test_idx <- start_idx:end_idx
        test <- x[test_idx, ]
        train <- x[! 1:nrow(x) %in% test_idx, ]
        result <- list("test" = test, "train" = train)
        return(result)
}

# implementation of leave-one-out using 10-fold cross validation
myloocv <- function(x, k, shuffle=TRUE) {
        result <- my10fold(x, k, shuffle)
        result$test <- result$test[1, ]
        return(result)
}
```

Since we've separated sampling from training methods, we can very easily plug in and test two classification algorithms instead of one. The first model builds a decision tree that predicts the *class* variable based on the rest of the variables. The second model uses the same formula but sets *control = rpart.control(minsplit=5)* which results in a better performing model.

```
# first model predicts class based on all other variables
mypredict_wine_model1 <- function(train, test) {
        dt <- rpart(class ~ ., data=train)
        predicted <- predict(dt, select(test, -class), type = "class")
        hit <- predicted == test$class
        return(sum(hit) / nrow(test))
}
# second model is a pruned version of the first model
mypredict_wine_model2 <- function(train, test) {
        dt <- rpart(class ~ ., data=train, control = rpart.control(minsplit = 5))
```

```
        predicted <- predict(dt, select(test, -class), type = "class")
        hit <- predicted == test$class
        return(sum(hit) / nrow(test))
}
```

The function below receives a dataframe and splits it in training and test data set using the four different sampling algorithms. It then applies the model it's being passed on every training and test set returned by the different sampling functions. The *class* variable is predicted by the model and crosschecked with the true value. The function finally returns a data frame with mean model accuracy for every model.

```
get_accuracy_stats <- function(df, f_model, model_name) {
        accuracy <- data.frame()
        resubstitution.data <- myresub(df, 0.10)
        holdout.data <- myholdout(df, 0.10)
        accuracy[1, "resubstitution"] <- f_model(resubstitution.data$train, resubstitution.data$test)
        accuracy[1, "holdout"] <- f_model(holdout.data$train, holdout.data$test)

        kfold_accuracy <- c()
        loocv_accuracy <- c()
        for (k in 1:10) {
                kfold.data <- my10fold(df, k)
                loocv.data <- myloocv(df, k)
                kfold_accuracy <- c(kfold_accuracy, f_model(kfold.data$train, kfold.data$test))
                loocv_accuracy <- c(loocv_accuracy, f_model(loocv.data$train, loocv.data$test))
        }
        accuracy[1, "kfold"] <- mean(kfold_accuracy)
        accuracy[1, "loocv"] <- mean(loocv_accuracy)
        accuracy[1, "model_name"] <- model_name
        return(accuracy)
}
```

We proceed to run the $get_accuracy_stats$ twenty times with our two models, $wine_model1$ and $wine_model2$. This will give us model accuracies for the two models using the four different sampling methods.

```
num_trials <- 20
accuracy <- data.frame()
for (i in 1:num_trials)  {
        accuracy <- rbind(
                get_accuracy_stats(wdf, mypredict_wine_model1, "wine_model1"),
                get_accuracy_stats(wdf, mypredict_wine_model2, "wine_model2"),
                accuracy
        )
}
```

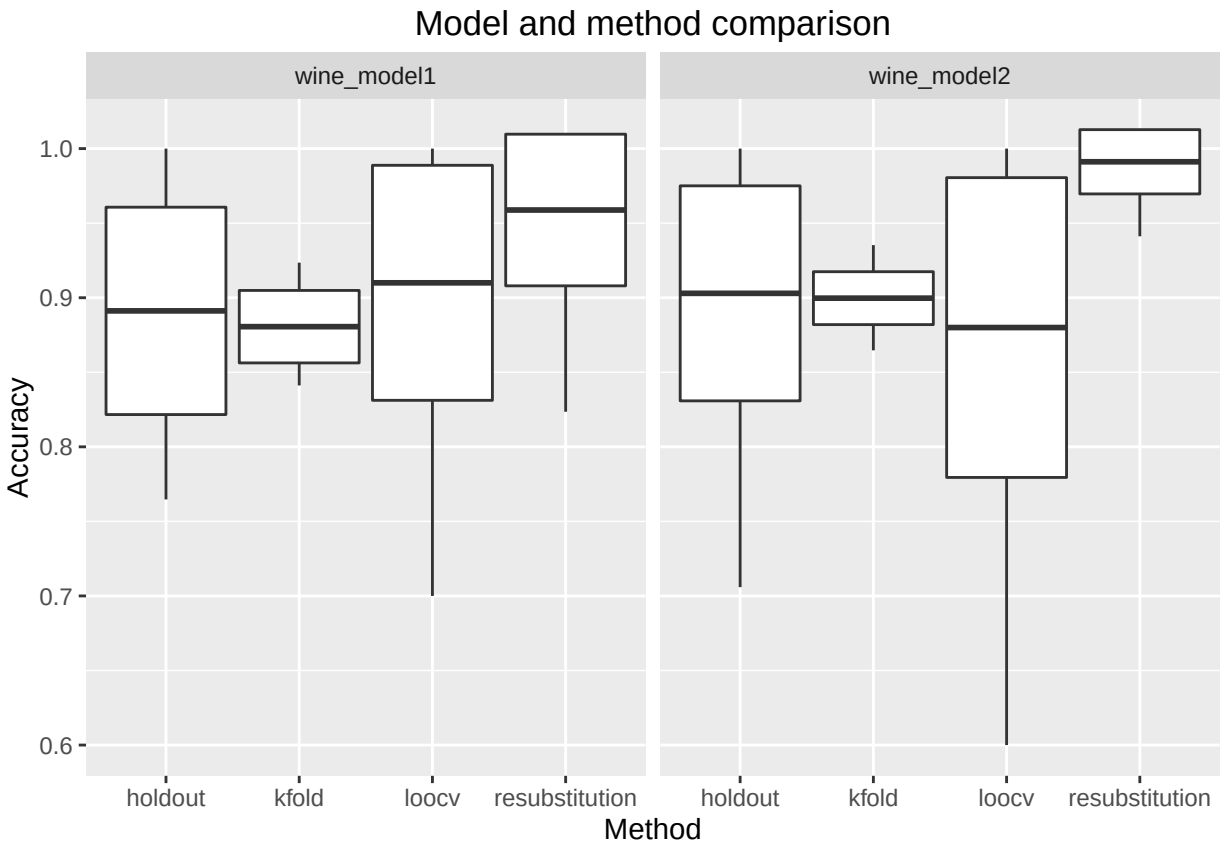
Group the results in an easy to plot structure.

```
accuracy.s <- accuracy %>% gather(key=model_name)
names(accuracy.s)[2] <- "method_name"
accuracy.s <- accuracy.s %>%
        group_by(model_name, method_name) %>%
        summarise(
                "mean" = mean(value),
```

```
                "min" = min(value),
                "max" = max(value),
                "lower" = mean(value) - sd(value),
                "upper" = mean(value) + sd(value),
                "sd" = sd(value)
        )
```

Plot accuracies for different sampling methods and models.

```
ggplot(accuracy.s, aes(method_name)) +
        geom_boxplot(
                aes(lower = lower, ymin = min, middle = mean, ymax = max, upper = upper),
                stat = 'identity') +
        ggtitle("Model and method comparison") +
        xlab("Method") +
        ylab("Accuracy") +
        facet_grid(. ~ model_name)
```



The chart above shows average accuracies over twenty trials for each of the four sampling methods and each of the two models. There are several things worth mentioning.

1. resubstitution method prefoerms very well, as expected, since the test data is a subset of the training data.
2. the leave-one-out method gives us high average accuracy but this can vary dramatically between specific training and test sets.
3. the holdout and 10-fold cross validation methods give us similar accuracy which is more stable than the one provided by the leave-one-out method.

4. the most robust method with both models is 10-fold cross validation.

Looking at the differences between the accuracies of both models we can see that pruning the tree has improved model performance. Setting *minsplit=5* results in about *2%* accuracy boost of the 10-fold cross validation method from *model*1 to *model*2 using this particular dataset.

This concludes our wine case study. We will now move on to the Titanic dataset.

Solution in R - TITANIC

The titanic dataset allows us to predict whether a Titanic passenger would survive or not depending on their features like age, sex, passenger class, etc. Here we read the data and print the first few records.

```
tdf <- read_csv("data/titanic.csv")
tdf$PassengerId <- NULL
tdf$Survived <- as.factor(tdf$Survived)
head(tdf)
```

```
## # A tibble: 6 x 11
##   Survived Pclass                                                Name
##     <fctr>  <int>                                               <chr>
## 1        0      3                             Braund, Mr. Owen Harris
## 2        1      1 Cumings, Mrs. John Bradley (Florence Briggs Thayer)
## 3        1      3                              Heikkinen, Miss. Laina
## 4        1      1        Futrelle, Mrs. Jacques Heath (Lily May Peel)
## 5        0      3                            Allen, Mr. William Henry
## 6        0      3                                    Moran, Mr. James
## # ... with 8 more variables: Sex <chr>, Age <dbl>, SibSp <int>,
## #   Parch <int>, Ticket <chr>, Fare <dbl>, Cabin <chr>, Embarked <chr>
```

We will predict *Survived* using two different models defined below.

```
# predict survival based solely on sex
mypredict_titanic_model1 <- function(train, test) {
        dt <- rpart(Survived ~ Sex, data=train)
        predicted <- predict(dt, select(test, -Survived), type = "class")
        hit <- predicted == test$Survived
        return(sum(hit) / nrow(test))
}
# predict survival based on passenger class, sex, age and other parameters
# prune the tree at minsplit=5
mypredict_titanic_model2 <- function(train, test) {
        dt <- rpart(
                Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
                data=train, control = rpart.control(minsplit = 5)
                )
        predicted <- predict(dt, select(test, -Survived), type = "class")
        hit <- predicted == test$Survived
        return(sum(hit) / nrow(test))
}
```

For sampling we will use the functions we defined while exploring the wine dataset. We will again test twenty trials of four sampling methods and two different models.

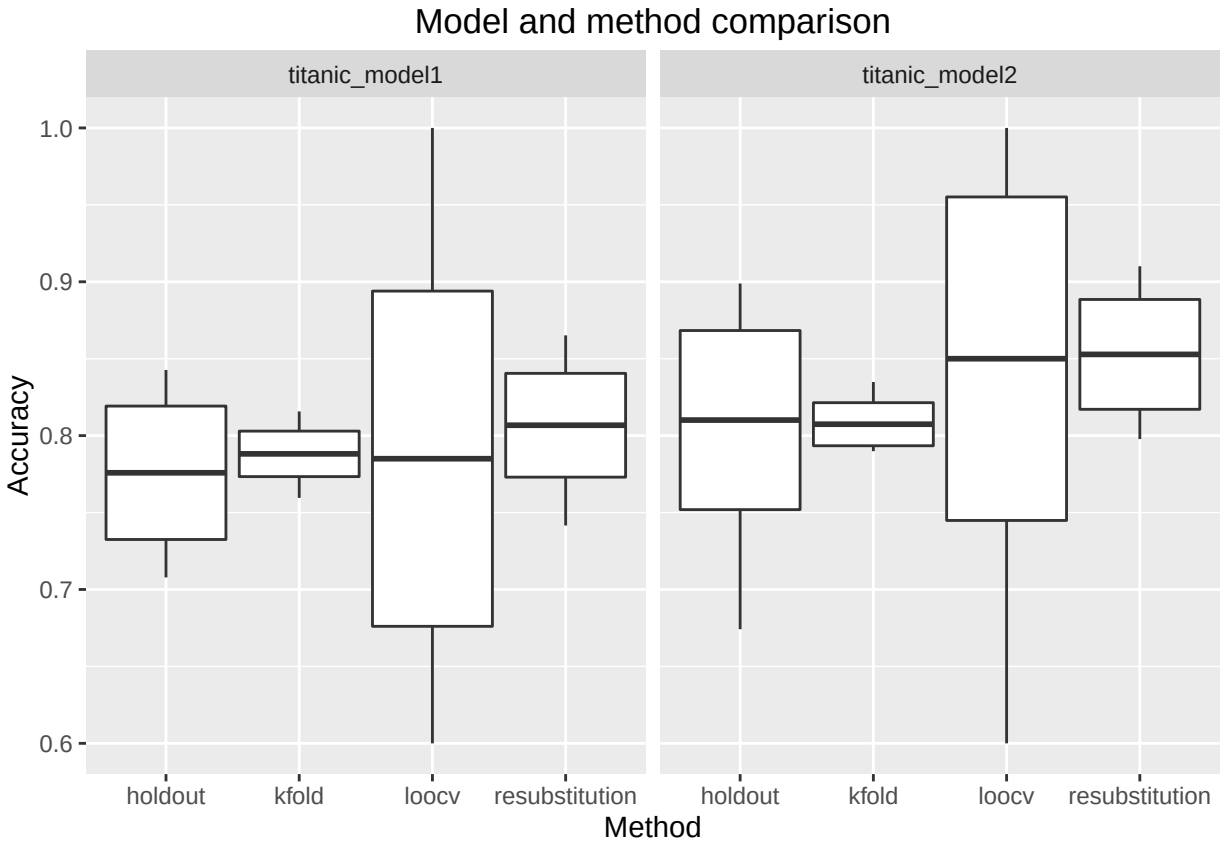
```
num_trials <- 20
accuracy <- data.frame()
for (i in 1:num_trials)  {
        accuracy <- rbind(
                get_accuracy_stats(tdf, mypredict_titanic_model1, "titanic_model1"),
                get_accuracy_stats(tdf, mypredict_titanic_model2, "titanic_model2"),
                accuracy
        )
}
```

Gather the results in an easy to plot format...

```
accuracy.s <- accuracy %>% gather(key=model_name)
names(accuracy.s)[2] <- "method_name"
accuracy.s <- accuracy.s %>%
        group_by(model_name, method_name) %>%
        summarise(
                "mean" = mean(value),
                "min" = min(value),
                "max" = max(value),
                "lower" = mean(value) - sd(value),
                "upper" = mean(value) + sd(value),
                "sd" = sd(value)
        )
```

Plot accuracy statistics.

```
ggplot(accuracy.s, aes(method_name)) +
        geom_boxplot(
                aes(lower = lower, ymin = min, middle = mean, ymax = max, upper = upper),
                stat = 'identity') +
        ggtitle("Model and method comparison") +
        xlab("Method") +
        ylab("Accuracy") +
        facet_grid(. ~ model_name)
```



The results we get are similar to the ones from the wine case study. 10-fold cross validation is still the most robust method. Model-wise, our first model only uses *Sex* to predict survival whereas the second model, which is more accurate, uses more features and prunes the tree at $minsplit = 5$.

This concludes our TITANIC case study.

Solution in KNIME - WINE

We are now going to solve problem #2 in KNIME using the WINE dataset.

To make things easier, we've downloaded *wine.data* as *wine.csv* and we've set the following header row: class, alcohol, malic.acid, ash, alcalinity.of.ash, magnesium, total.phenols, flavanoids, nonflavanoid.phenols, proanthocyanins, color.intensity, hue, od280od315.of.dliuted.wines, proline

We then go through the following KNIME workflow:

The flow loads the wine dataset, changes the *class* column from Integer to String and proceeds to run 20 iterations of four different sampling methods. The result from the sampling is passed to decision tree learners and predictors using the same parameters. Accuracy statistics for each sampling method are gathered and finally combined and plotted using a boxplot.

The resulting dataframe contains mean accuracy of each method for every one of the 20 iterations.

The boxplot of the mean accuracies is shown below:

.

Let's proceed with applying the same technique to the Titanic dataset.

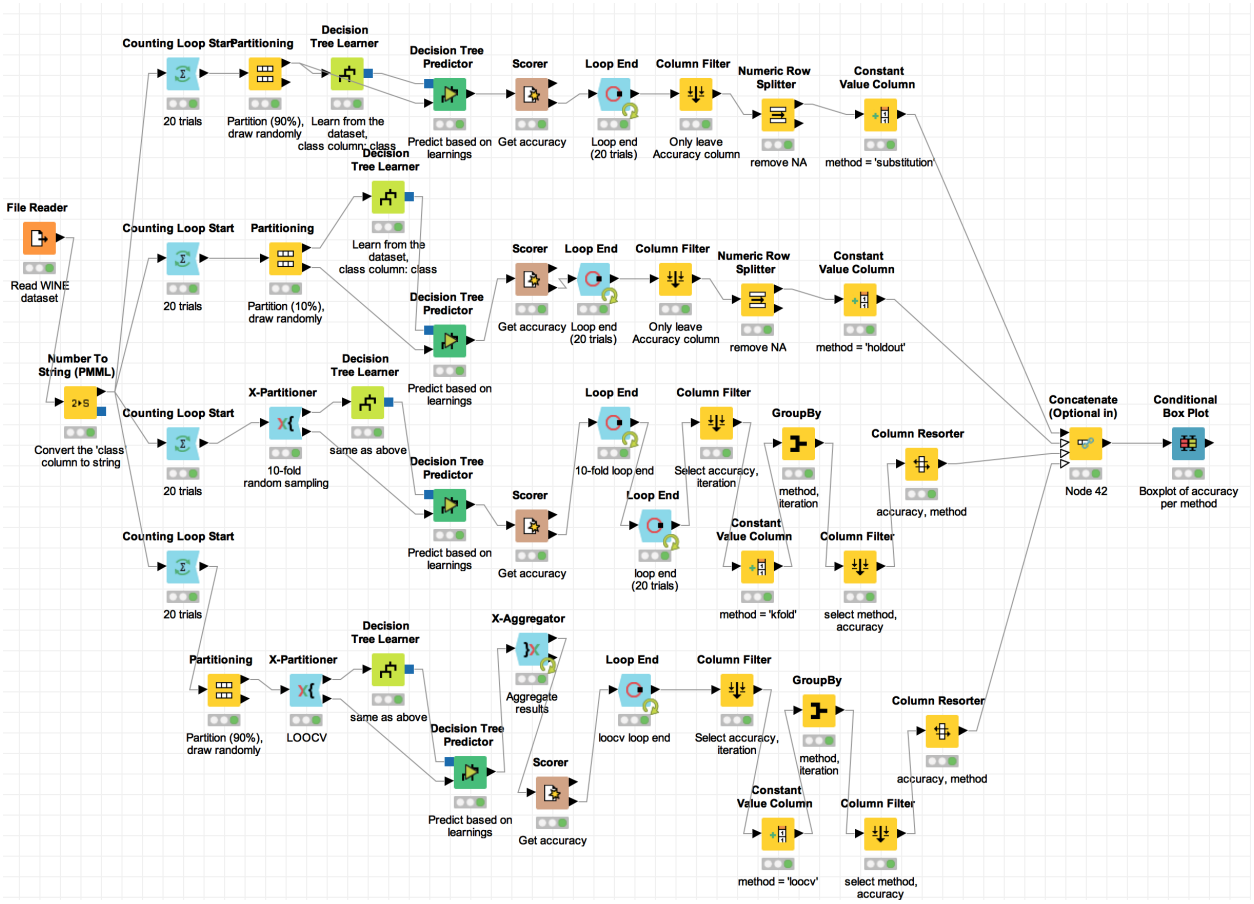


Figure 1: WINE Workflow

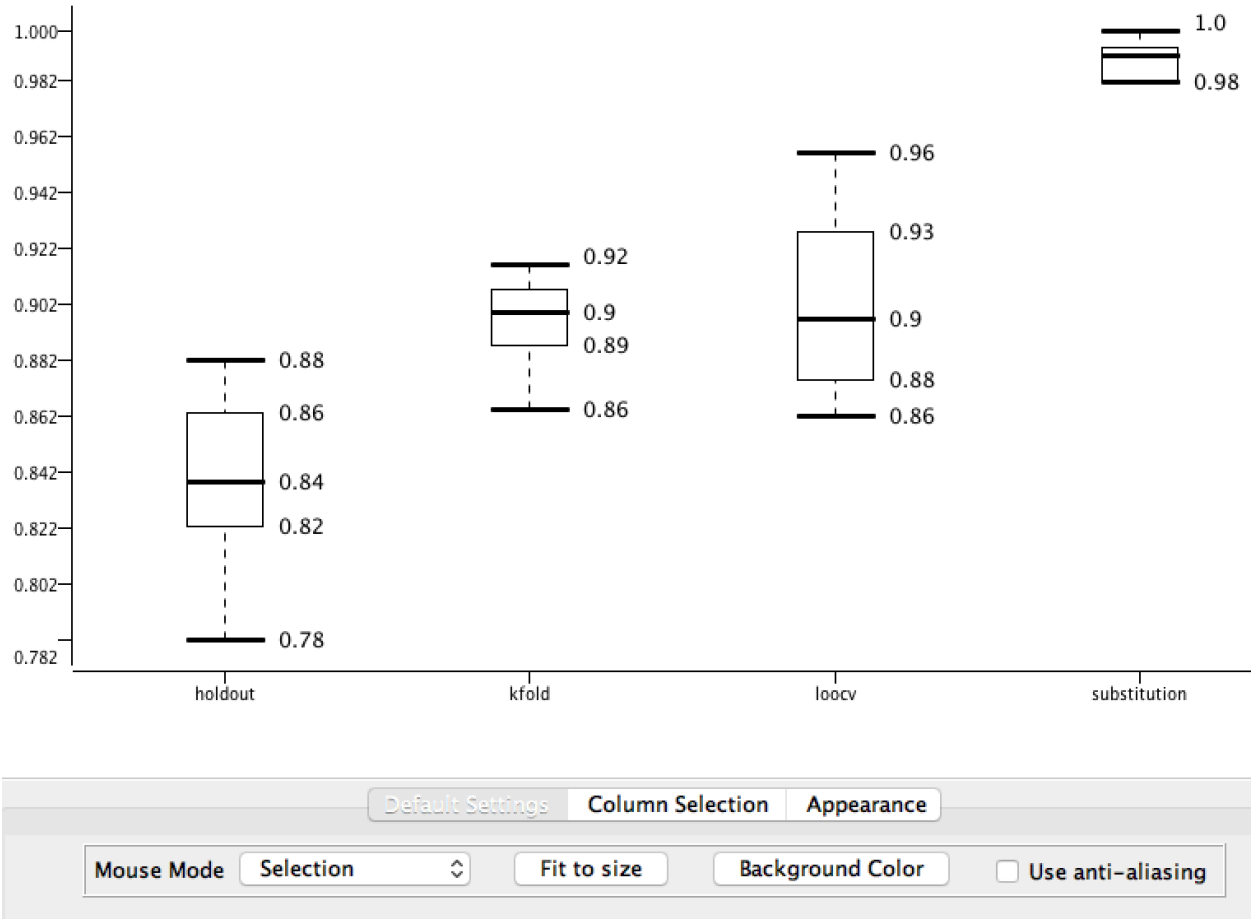


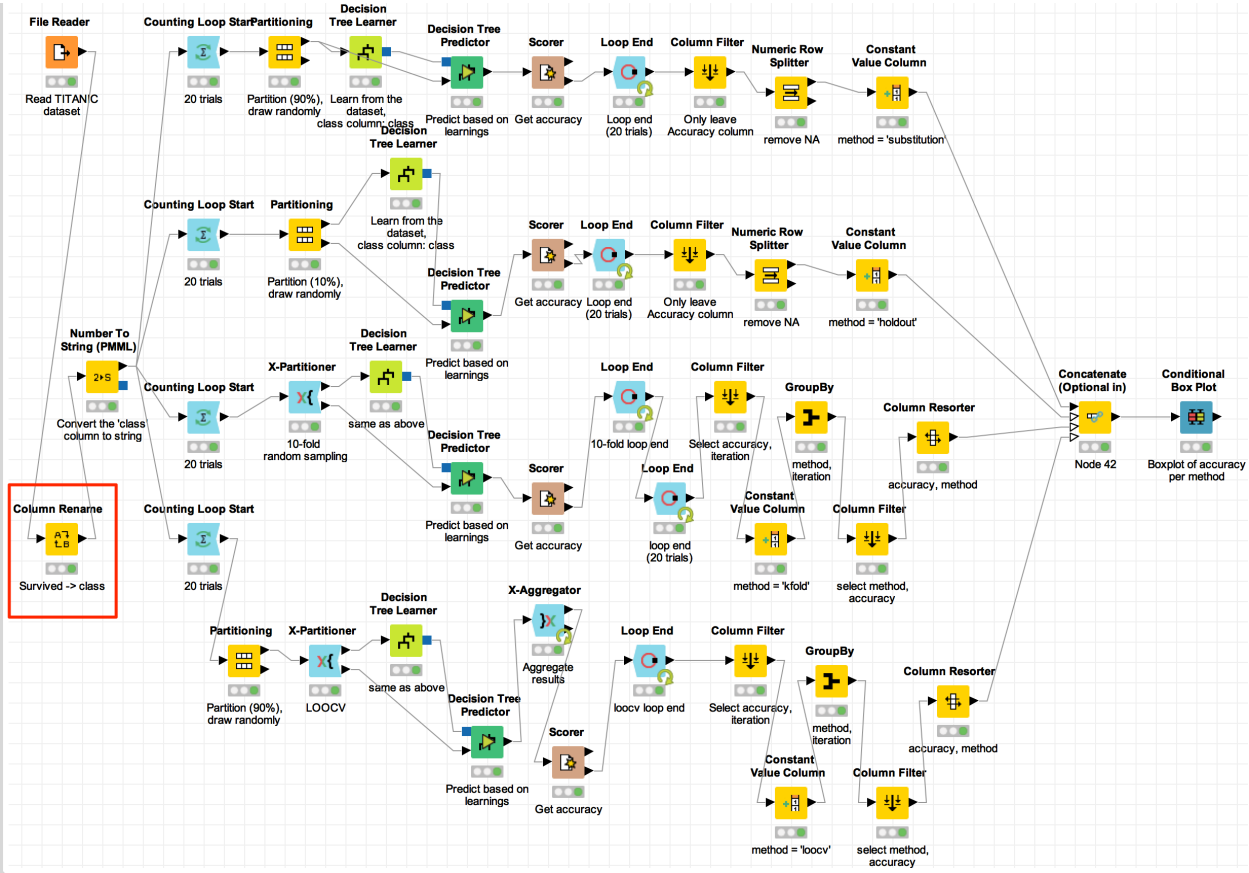
Figure 2: WINE Accuracy Boxplot

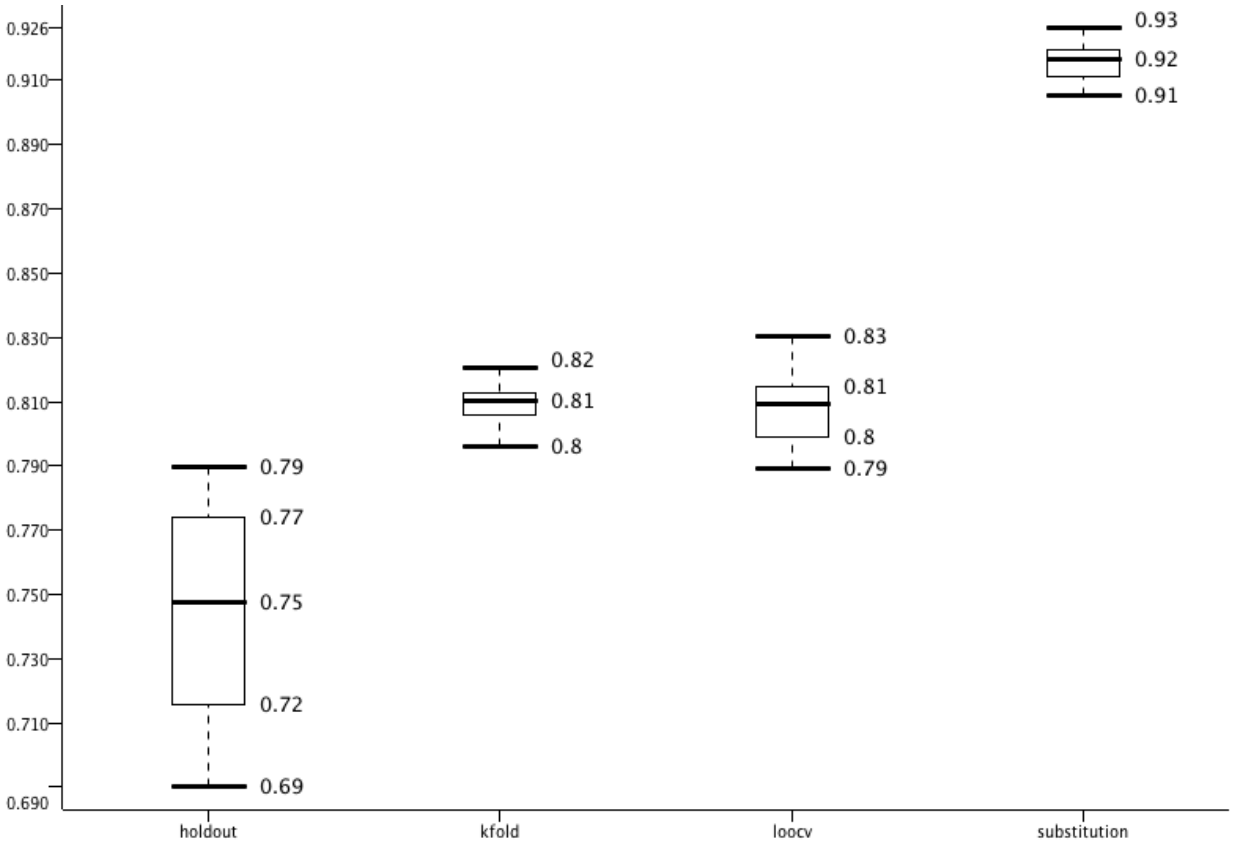
Solution in KNIME - TITANIC

We are now going to solve problem #2 in KNIME using the TITANIC dataset.

The nature of the TITANIC dataset is discussed in the R solution of Problem #2 - its label column is called *Survived*. In order to reuse as much as possible from our KNIME workflow for the WINE dataset, we will transform the TITANIC dataset to a WINE-like dataset. This involves renaming the *Survived* column to *class*. Once we have done this, we can reuse all of the logic in the loops and the boxplot.

The flow then works exactly as it does on the WINE dataset but it uses the TITANIC dataset instead.





Discussion

Both problem one and problem two allow us to showcase features of two environments that can interact very well with each other, R and KNIME. Solving the problems in two different environments but using the same methods and concepts serves as a reminder that one can almost always solve a problem using more than one tool, however the tools that we use define our frame of reference.

Problem one shows us that we can use a number of clustering algorithms and distance metrics, we can "cut the tree" at different levels, etc. but ultimately for best results one needs to try multiple methods and parameters and choose the best combination according to his or her own views.

Problem two helps us build a framework for general purpose classifications tasks where we have one class variable that we ultimately try to predict using the rest of the attributes in the dataset. There are different sampling methods we can use to split a datasaet into training and test sets and what is important is that the way we sample the data affects our model accuracy. Testing accuracies for different sampling methods has presented us the opportunity to also measure performance of different models in addition to sampling methods. This kind of framework gives a practitioner a solid approach to working on classification problems and being able to easily compare accuracies of different models and sampling methods for the same datasets.