Zeszyty Naukowe WWSI, No 23, Vol. 14 2020, s. 27-49
DOI: 10.26348/znwwsi.23.27

Data mining tasks and methods – implementations in R

Ewa Figielska[*]

Warsaw School of Computer Science

Abstract

The aim of the paper is to present how some of the data mining tasks can be solved using the R programming language. The full R scripts are provided for preparing data sets, solving the tasks and analyzing the results.

Keywords **–** data mining, R programming language, classification, prediction, clustering, association

[*] E-mail: efigielska@poczta.wwsi.edu.pl

Manuscript received October 7, 2020.

1. Introduction

Data mining is the process of discovering useful patterns and trends in large data sets [1, 2]. In this paper, we present several data mining tasks, such as prediction, classification, clustering and association as well as data cleaning, and show how they can be dealt with using the R programming language. We present several computational examples in which real-life and benchmark data sets are used. The full R scripts are provided for preparing data sets, solving the tasks and analyzing the obtained results.

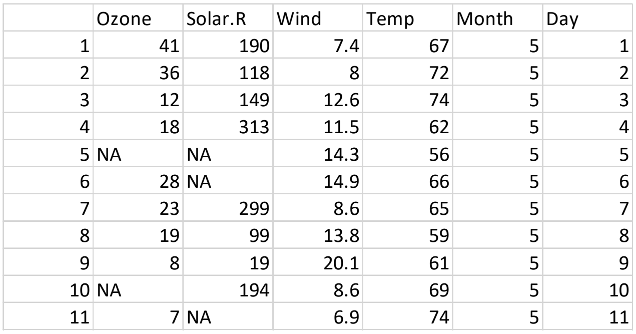
2. Data preprocessing

Databases often contain raw data in a form which is not suitable for data mining algorithms and models. For example, they may contain missing values, outliers, expired values, values not consistent with common sense or other unusual values which can mislead the results of subsequent data analysis [1]. Therefore, before any knowledge can be discovered from a database, data usually should be cleaned and transformed. In this section, we show how missing values and outliers can be tackled with.

2.1. Missing values

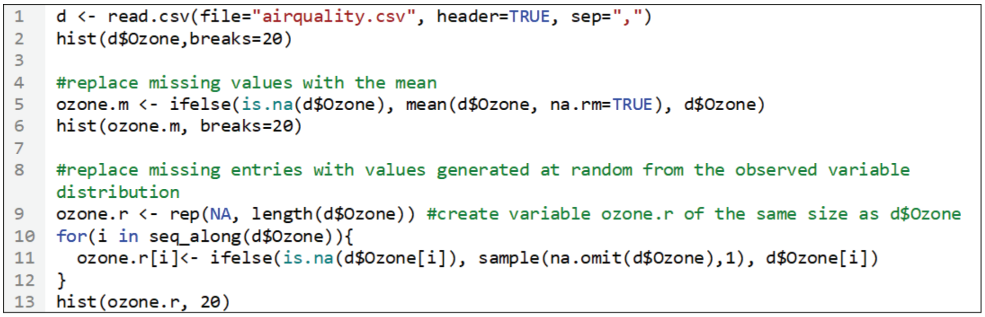
The simplest way to deal with a missing value is to ignore it, e.g. to remove a record containing a field with no data. However, this may lead to biased sets of data. The other way is to replace missing values with substituted values, which will be shown in the following example.

In the experiment, data set “airquality.csv” [3] is used (see Figure 1). We focus on variable $Ozone$ for which some values are not available (they are denoted by NA). The R script and the resulting histograms are presented in Figures 2 and 3, respectively. In Figure 3a the histogram for original variable $Ozone$ (with the NA values omitted) is shown. Figure 3b presents the histogram for variable $ozone.m$ obtained by replacing the NA values by the mean (in line 5 of the R script). The histogram has a clear peak in the frequency for the interval containing the mean value. Figure 3c shows variable $ozone.r$ created by replacing the NA values with the values generated at random from the observed variable distribution (in lines 9-12 of the R script). In this case, the histogram looks better than in Figure 3b, its shape is very similar to the shape of the histogram in Figure 3a.



	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
10	NA	194	8.6	69	5	10
11	7	NA	6.9	74	5	11

Figure 1. Excerpt from data set “airquality.csv”



```
1   d <- read.csv(file="airquality.csv", header=TRUE, sep=",")
2   hist(d$Ozone,breaks=20)
3   
4   #replace missing values with the mean
5   ozone.m <- ifelse(is.na(d$Ozone), mean(d$Ozone, na.rm=TRUE), d$Ozone)
6   hist(ozone.m, breaks=20)
7   
8   #replace missing entries with values generated at random from the observed variable
    distribution
9   ozone.r <- rep(NA, length(d$Ozone)) #create variable ozone.r of the same size as d$Ozone
10  for(i in seq_along(d$Ozone)){
11    ozone.r[i]<- ifelse(is.na(d$Ozone[i]), sample(na.omit(d$Ozone),1), d$Ozone[i])
12  }
13  hist(ozone.r, 20)
```

Figure 2. R script for handling missing values

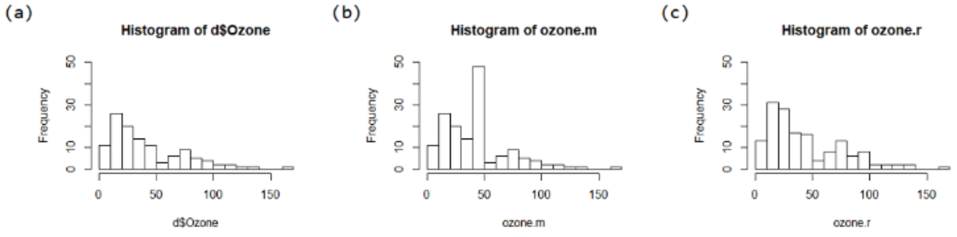
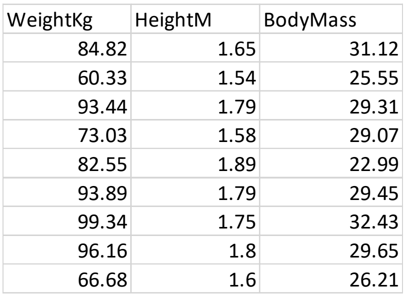


Figure 3. Histograms for variable $Ozone$: with NA omitted (a), after replacing the NA values with the mean (b), after replacing the NA values with the values generated at random from the observed variable distribution (c)

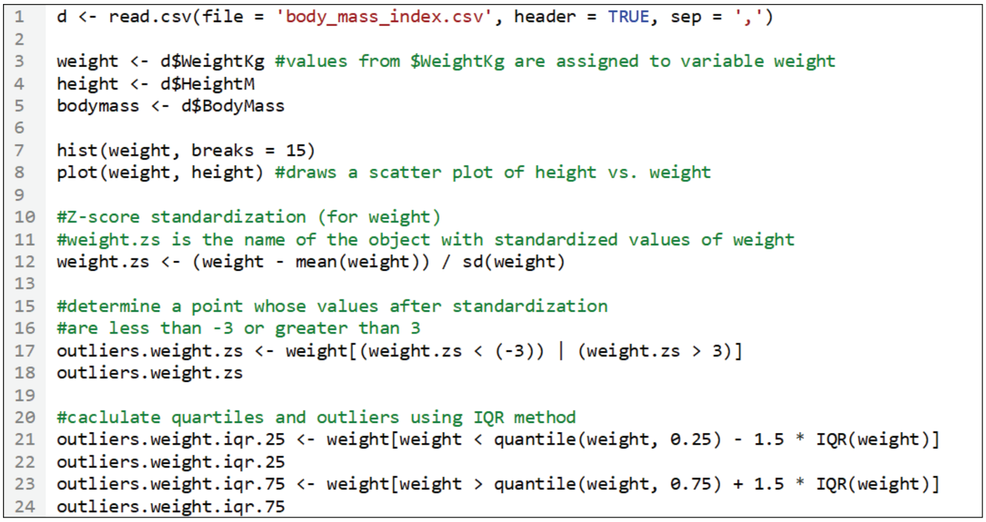
2.2 Outliers

Outliers are values which go against the trend of the remaining data. They may indicate errors in data entry or represent unusual observations which carry important information, e.g. for fraud detection or medical analysis. Because of the presence of outliers, statistical methods may deliver unstable results [1, 4, 5].



WeightKg	HeightM	BodyMass
84.82	1.65	31.12
60.33	1.54	25.55
93.44	1.79	29.31
73.03	1.58	29.07
82.55	1.89	22.99
93.89	1.79	29.45
99.34	1.75	32.43
96.16	1.8	29.65
66.68	1.6	26.21

Figure 4. Excerpt from data set “body_mass_index.csv”



```r
d <- read.csv(file = 'body_mass_index.csv', header = TRUE, sep = ',')

weight <- d$WeightKg #values from $WeightKg are assigned to variable weight
height <- d$HeightM
bodymass <- d$BodyMass

hist(weight, breaks = 15)
plot(weight, height) #draws a scatter plot of height vs. weight

#Z-score standardization (for weight)
#weight.zs is the name of the object with standardized values of weight
weight.zs <- (weight - mean(weight)) / sd(weight)

#determine a point whose values after standardization
#are less than -3 or greater than 3
outliers.weight.zs <- weight[(weight.zs < (-3)) | (weight.zs > 3)]
outliers.weight.zs

#caclulate quartiles and outliers using IQR method
outliers.weight.iqr.25 <- weight[weight < quantile(weight, 0.25) - 1.5 * IQR(weight)]
outliers.weight.iqr.25
outliers.weight.iqr.75 <- weight[weight > quantile(weight, 0.75) + 1.5 * IQR(weight)]
outliers.weight.iqr.75
```

Figure 5. R script for detecting outliers

In this section, four methods for detecting outliers are considered: a histogram, a two-dimensional scatter plot (graphical methods), a Z-score method and an interquartile range (numerical methods). In the experiment, data set “body_mass_index.csv” [6] is used (see Figure 4), which stores 50 records with weights, heights and body mass indexes for 50 people. The R script is presented in Figure 5. The histograms and scatter plots are shown in Figure 6. In the histogram for weight (Figure 6a), there appears to be one lonely value in the right tail of the distribution, $weight = 155.66$ (circled in red). This point can be clearly identified as an outlier. The scatter plots of $height$ against $weight$, and of $bodymass$ against $weight$ (Figures 6d and 6e) show the same outlier. Figures 6a, 6c, 6d, 6e suggest that the point with $weight = 111.13$ and $bodymass = 39.22$ should be also taken into account as a possible outlier (circled in orange).

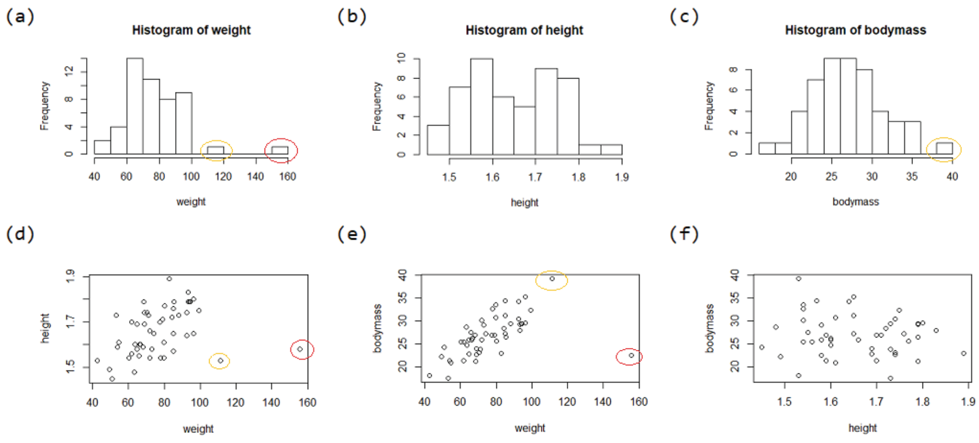


Figure 6. Detecting outliers: histograms and scatter plots

Z-score method for identifying outliers states that a data value is an outlier if it has a Z-score which is either less than -3 or greater than 3. Z-score for an observed x_i value of variable x is calculated according to the following formula:

$$Z_i = (x_i - \bar{x})/\sigma,$$

where $\bar{x}$ is the mean and σ is the standard deviation of all observations of x.

Z-score for variable *weight* is calculated in line 12 of the script (Figure 5) and stored in variable *weight.zs*. In line 17, the outliers are detected as those of the *weight* values for which the *weight.zs* values are less than -3 or greater than 3.

The interquartile range (IQR) based method for detecting outliers, states that a data value is an outlier if:

- it is located $1.5 * IQR$ or more below $Q1$ or
- it is located $1.5 * IQR$ or more above $Q3$,

where $Q1$ and $Q3$ denote the first and the third quartile, respectively, and $IQR = Q3 - Q1$.

The IQR method is implemented in lines 21 and 23 of the script (Figure 5) and its results (for weight) are held in variables *outliers.weight.iqr.25* and *outliers.weight.iqr.75* (data points lying, respectively, $1.5 * IQR$ or more below $Q1$ and $1.5 * IQR$ or more above $Q3$).

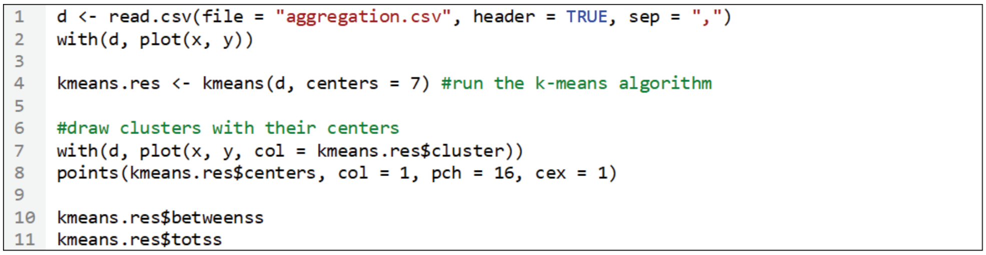
As the result of the computations, the Z-score method indicated the weight value of 155.66 as an outlier, and the IQR method, beside the weight with value 155.66, indicated also as an outlier the body mass index value of 39.22. This confirms the results suggested by the graphical methods. Weight of 111.13 was not detected as an outlier by any numerical method.

3. Clustering

Clustering is a technique for grouping data points (records, observations). A clustering algorithm tries to group data so that data points with similar features are put together in one cluster and different clusters contain highly dissimilar points [1, 7, 8]. Clustering can be used for example to segment financial behavior into benign and suspicious categories, to identify fake news or as a dimension reduction tool.

One of the most popular clustering methods is the k-means algorithm. It works with the vector of cluster centers (centroids). In each iteration, the distance between each data point and each cluster centroid is calculated. Then, every data point is assigned to the closest cluster and the centroids are updated. The process stops when there is no change in the assignment of data points to clusters in two subsequent iterations.

In the experiment, data set “Aggregation.csv” [9] is used. It contains coordinates of 788 points belonging to 7 groups. The R script, presented in Figure 7, simply invokes the $kmeans$ function to which the number of clusters, 7, is passed (line 4). The data set with clearly visible 7 groups of points and the clustering results (with centroids marked as black circles) are presented in Figures 8a and 8b, respectively. We can see, that the k-means algorithm identified some of the groups in a bit different way than we can see them in Figure 8a.



```
1   d <- read.csv(file = "aggregation.csv", header = TRUE, sep = ",")
2   with(d, plot(x, y))
3
4   kmeans.res <- kmeans(d, centers = 7) #run the k-means algorithm
5
6   #draw clusters with their centers
7   with(d, plot(x, y, col = kmeans.res$cluster))
8   points(kmeans.res$centers, col = 1, pch = 16, cex = 1)
9
10  kmeans.res$betweenss
11  kmeans.res$totss
```

Figure 7. R script for clustering with k-means algorithm

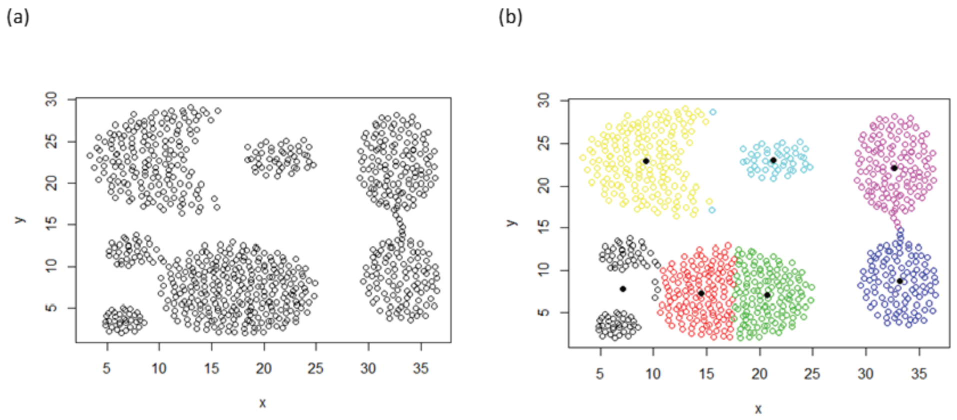


Figure 8. Data set with 7 groups of points (a), and the result of clustering by k-means algorithm (b)

As a quality index for clustering, we can take the ratio of the sum of squares between clusters (SSB) to the total sum of squares (SST), defined as follows:

- $SSB = \sum_{k=1}^{K} n_k d(c_k, \bar{x})^2$, where K is the number of clusters, n_k is the number of records in cluster k, $d(c_k, \bar{x})$ is the distance between the center of cluster k, c_k, and the mean, $\bar{x}$,
- $SST = \sum_{i=1}^{n} d(x_i, \bar{x})^2$, where n is the number of records, $d(x_i, \bar{x})$ is the distance between data point x_i and the mean.

For the obtained clustering result, the value of SSB/SST is equal to 91.5% which means a quite good quality partition (SSB and SST values can be shown by executing lines 10 and 11 of the script).

4. Classification

Classification is the most common data mining task. In classification, there is a target categorical variable, which is partitioned into predetermined categories (classes), and a set of input (predictor) variables [1]. For example, in the breast cancer diagnosis, patients may be assigned to either a benign (non-cancerous) category or malignant (cancerous) category based on the features computed from a digitized image of a fine needle aspirate of a breast mass, which describe characteristics of the cell nuclei present in the image [10].

A classification algorithm is given a great number of records, each record containing a set of input variables for which the information on a target variable is provided. The algorithm learns which values of the target variable are associated with which values of the input variables. In this way, a model (a classifier) is built which can be used, after a proper evaluation, to predict the unknown value of the target for new values of the input variable set.

4.1 Cross validation

A common technique for model evaluation is k-fold cross validation [2, 11]. It proceeds as follows.

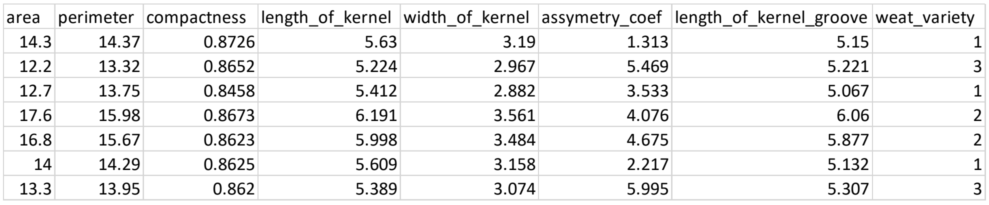
1. The original data are partitioned into k independent subsets.

2. The model is built using the training set containing data from $k-1$ subsets and tested using the test set containing data from the kth subset. This is done iteratively until k different models created.
3. The results, e.g. error rates, from the k models are combined using averaging or voting.

The k-fold cross validation method ensures that every observation from the original dataset has a chance of appearing in a test set.

The following example shows how to use the k-fold cross validation technique for evaluating the performance of one of the classification algorithms, namely the k-nearest neighbor (knn) algorithm. The knn algorithm is an example of instance-based learning, in which the training data set is stored, so that the class for a new unclassified record may be found simply by comparing it to the k closest records in the training set.

In the experiment, data set "seeds.csv" is used [12]. It contains 210 records with 7 geometric parameters of wheat kernels for three different varieties of wheat: Kama, Rosa and Canadian (see Figure 9).

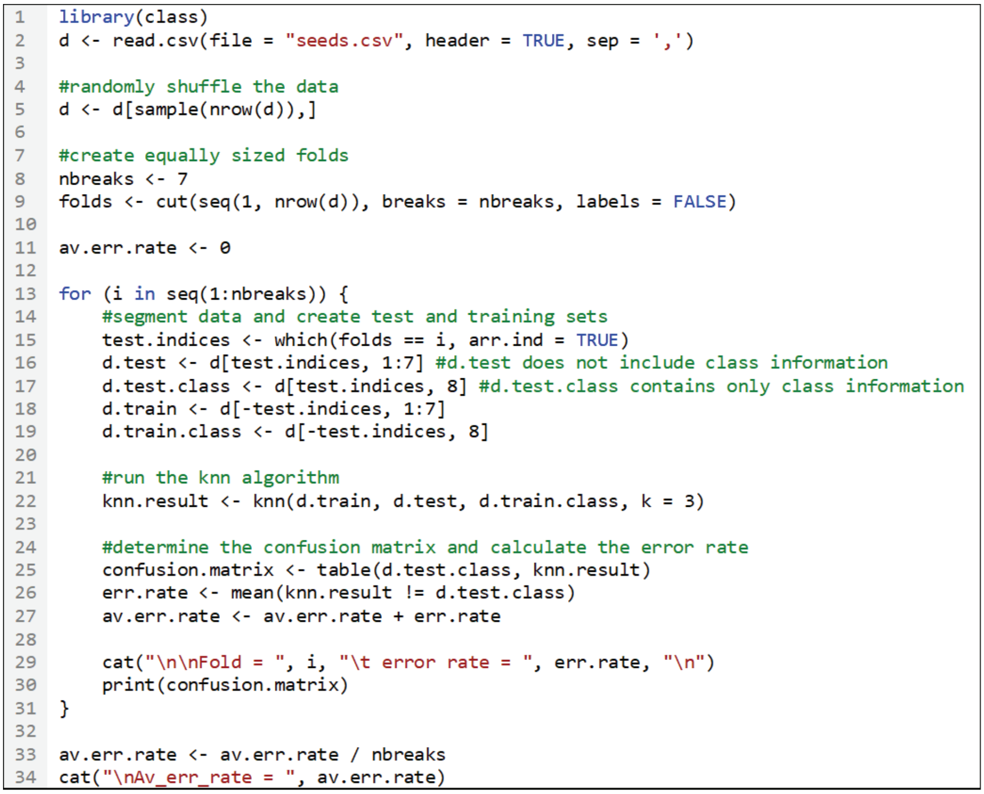


area	perimeter	compactness	length_of_kernel	width_of_kernel	assymetry_coef	length_of_kernel_groove	weat_variety
14.3	14.37	0.8726	5.63	3.19	1.313	5.15	1
12.2	13.32	0.8652	5.224	2.967	5.469	5.221	3
12.7	13.75	0.8458	5.412	2.882	3.533	5.067	1
17.6	15.98	0.8673	6.191	3.561	4.076	6.06	2
16.8	15.67	0.8623	5.998	3.484	4.675	5.877	2
14	14.29	0.8625	5.609	3.158	2.217	5.132	1
13.3	13.95	0.862	5.389	3.074	5.995	5.307	3

Figure 9. Excerpt from data set "seeds.csv"

In Figure 10, the R script for the k-fold cross validation is shown. Before partitioning the records of the data set into folds, they are placed in a random order (line 5). The number of folds, stored in variable *nbreaks*, is set to 7, which means that each training set will contain 180 records and tests will be carried out using 30 records. The folds are created in line 9. The successive folds are used one by one in the successive iterations of the loop (lines 13-31) to determine the indices of records to be included in the test and training sets. For example, in the first iteration, numbers from 1 to 30 are assigned to variable *test.indices*, which is used for creating the test set with the first 30 records from the data set (lines 15-17). The remaining records of the data set are assigned to the training set (lines 18-19). Variables *d.test.class* and *d.train.class* contain only values of the class (*weat_variety*) attribute, while *d.train* and *d.test* have all the attributes without a class attribute.

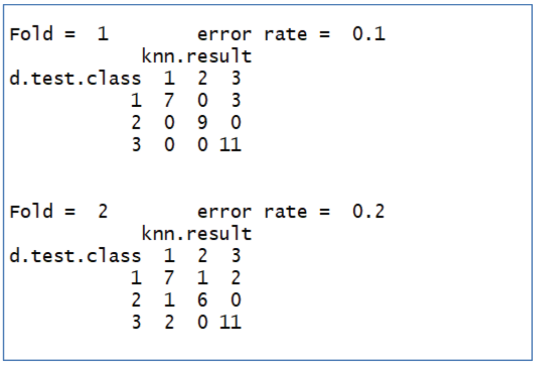
The knn algoritm, executed in line 22, classifies each record from the test set (*d.test*) by comparing it with its 3 nearest neighbors from the training set. Mark, that two variables *d.train* and *d.train.class* are passed to the knn procedure as a training set. The algorithm results are evaluated based on the known classes using the confusion matrix and calculating the error rate (lines 25-26).



```
1   library(class)
2   d <- read.csv(file = "seeds.csv", header = TRUE, sep = ',')
3
4   #randomly shuffle the data
5   d <- d[sample(nrow(d)),]
6
7   #create equally sized folds
8   nbreaks <- 7
9   folds <- cut(seq(1, nrow(d)), breaks = nbreaks, labels = FALSE)
10
11  av.err.rate <- 0
12
13  for (i in seq(1:nbreaks)) {
14      #segment data and create test and training sets
15      test.indices <- which(folds == i, arr.ind = TRUE)
16      d.test <- d[test.indices, 1:7] #d.test does not include class information
17      d.test.class <- d[test.indices, 8] #d.test.class contains only class information
18      d.train <- d[-test.indices, 1:7]
19      d.train.class <- d[-test.indices, 8]
20
21      #run the knn algorithm
22      knn.result <- knn(d.train, d.test, d.train.class, k = 3)
23
24      #determine the confusion matrix and calculate the error rate
25      confusion.matrix <- table(d.test.class, knn.result)
26      err.rate <- mean(knn.result != d.test.class)
27      av.err.rate <- av.err.rate + err.rate
28
29      cat("\n\nFold = ", i, "\t error rate = ", err.rate, "\n")
30      print(confusion.matrix)
31  }
32
33  av.err.rate <- av.err.rate / nbreaks
34  cat("\nAv_err_rate = ", av.err.rate)
```

Figure 10. R script for the *k*-fold cross validation

In Figure 11, the confusion matrix is presented for the first two folds. It shows that, for fold 1, three records belonging to class 1 were misclassified and assigned to class 3. Other records are classified correctly. The error rate for this fold is equal to 3/30 = 0.1. The error rates from 7 models are averaged and displayed in line 34 of the script.



```
Fold =  1        error rate =  0.1
            knn.result
d.test.class  1  2  3
           1  7  0  3
           2  0  9  0
           3  0  0 11


Fold =  2        error rate =  0.2
            knn.result
d.test.class  1  2  3
           1  7  1  2
           2  1  6  0
           3  2  0 11
```

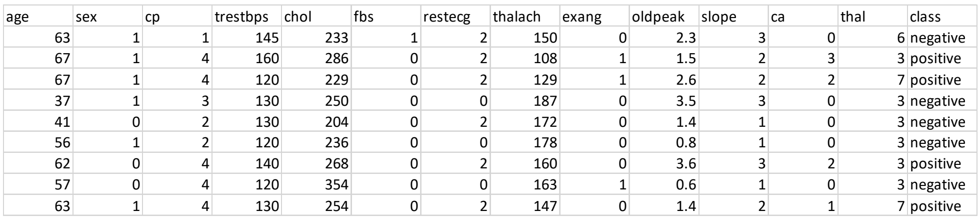
Figure 11. Results of the k-fold cross validation: error rates and confusion matrices for the first 2 folds

4.2 Overfitting

After creating a classification model (or other data mining model), it may happen that the model very closely represents training data and due to this is not able to work well for new data. This situation is called overfitting [1, 2].

In this section, the example is provided in which as a classification model a decision tree is created and the influence of the model complexity, i.e. the decision tree size, on the accuracy of the classification is examined.

In the experiment, data set "cleveland.csv" is used [13] (see Figure 12), in which the target (variable $class$) refers to the presence of heart disease in the patient. The original set was subjected to some slight modifications, namely records with missing values were deleted, and the original target num was replaced by $class$ with value $positive$ if $num > 0$ (disease is present), and $negative$ if $num = 0$ (no disease).



age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	class
63	1	1	145	233	1	2	150	0	2.3	3	0	6	negative
67	1	4	160	286	0	2	108	1	1.5	2	3	3	positive
67	1	4	120	229	0	2	129	1	2.6	2	2	7	positive
37	1	3	130	250	0	0	187	0	3.5	3	0	3	negative
41	0	2	130	204	0	2	172	0	1.4	1	0	3	negative
56	1	2	120	236	0	0	178	0	0.8	1	0	3	negative
62	0	4	140	268	0	2	160	0	3.6	3	2	3	positive
57	0	4	120	354	0	0	163	1	0.6	1	0	3	negative
63	1	4	130	254	0	2	147	0	1.4	2	1	7	positive

Figure 12. Excerpt from data set "cleveland.csv" (after modifications of the target variable).

The example of a decision tree is shown in Figure 13. It consists of 5 decision nodes. The split in the root (node 1) partitions the data set into two subsets: with $thal$ (thalassemia) no greater and greater than 3. Other splits (made in nodes 2, 7, 4 and 8) take into account the values of ca (number of major vessels colored by fluoroscopy), cp (chest pain type) and $exang$ (exercise induced angina). The leaf nodes, i.e. nodes which terminate the tree, are not homogenous but most of the records in each leaf belong to the same class, thus leaf nodes 3, 5 and 9 represent class label $negative$, and leaf nodes 6, 10 and 11 represent class label $positive$.

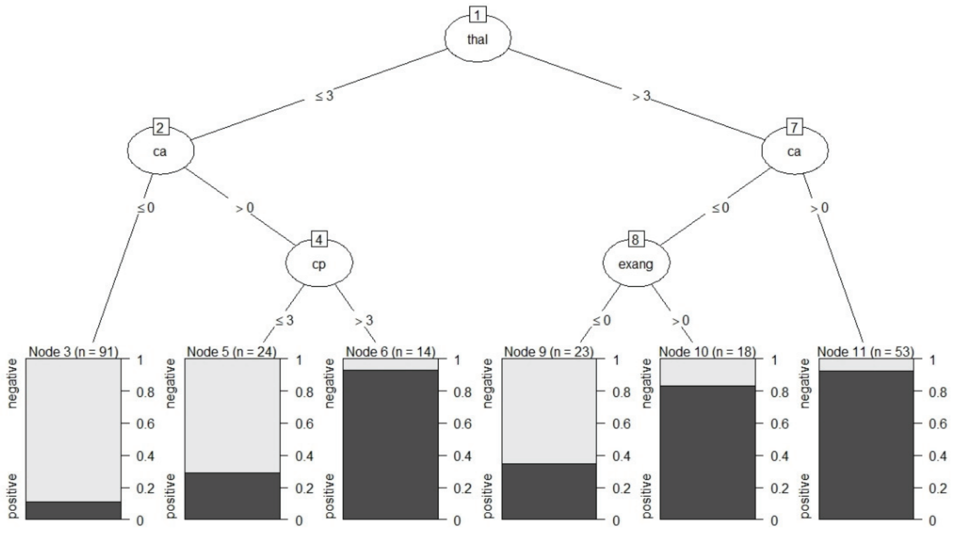


Figure 13. Decision tree

The decision tree shown in Figure 13 was induced by the C4.5 algorithm trained on the 0.75% of records from "cleveland.csv".

The C4.5 algorithm uses the concept of information gain, or entropy (degree of non-uniformity) reduction, to select the best split. This is done in the following way. Suppose that we have a candidate split S, which partitions the data set T into k subsets, $T_1, T_2, \ldots, T_k$. The information gain $g_T(S)$ achieved by split S can be calculated as follows:

$g_T(S) = H(T) - H_S(T)$,

where:

- $H(T)$ denotes the entropy for set T, $H(T) = -\sum_{j=1}^{c} p_j \log_2 p_j$, where c is the number of classes (possible values of a target variable), p_j is the ratio of the number of records belonging to class j to the total number of records in T.
- $H_S\,(T)$ is the weighted sum of the entropies for subsets created by split S, $H_S(T) = \sum_{i=1}^{k} P_i H_S(T_i)$, where P_i is the ratio of the number of records in subset T_i to the total number of records in T.

The value of $g_T(S)$ indicates an increase in information produced by split S for data set T. The C4.5 algorithm, when creating a decision tree, favors splits leading to subsets with small entropies by choosing, for each node, a split with the greatest value of the information gain. The smallest value of entropy is equal to 0 and is achieved when all the records of a subset belong to the same class. A decision tree for which all the leaf nodes have entropy of value 0 represents the training data very well, however, for new data, its performance may be quite poor. Therefore, to avoid the overfitting to the training data and achieve better classification abilities, a decision tree is usually pruned.

In the experiment showing the phenomenon of overfitting, procedure $C5.0$ (implementing the extended version of the C4.5 algorithm) is used with various values of parameter $minCases$ which controls the size of the created tree. According to R documentation [14] $minCases$ is an integer for the smallest number of samples that must be put in at least two of the splits. In practice, the smaller the value of $minCases$ is, the greater is the size of a tree.

Figure 14 presents the results obtained using 4-fold cross validation on the "cleveland.csv" data set: the average error rate on the test sets, denoted by $test$, and the average error rate on the training sets, denoted by $train$. The $minCases$ parameter was changing from 1 to 30. The R script used to obtain these results is shown in Figure 15.

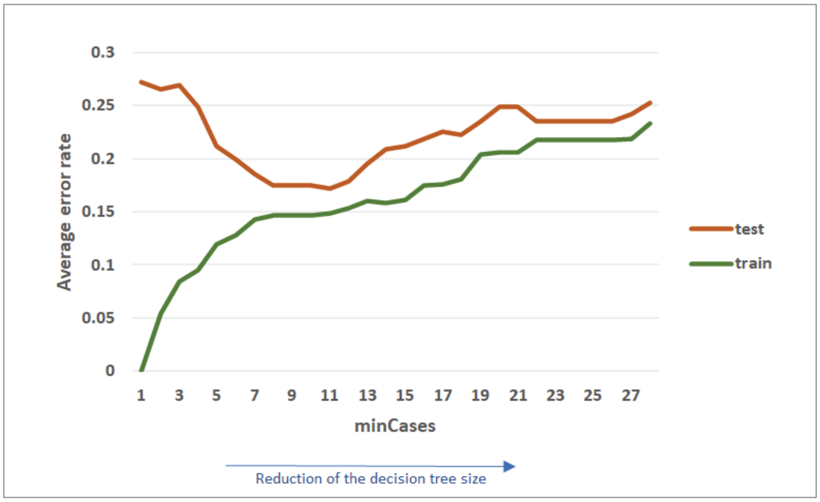
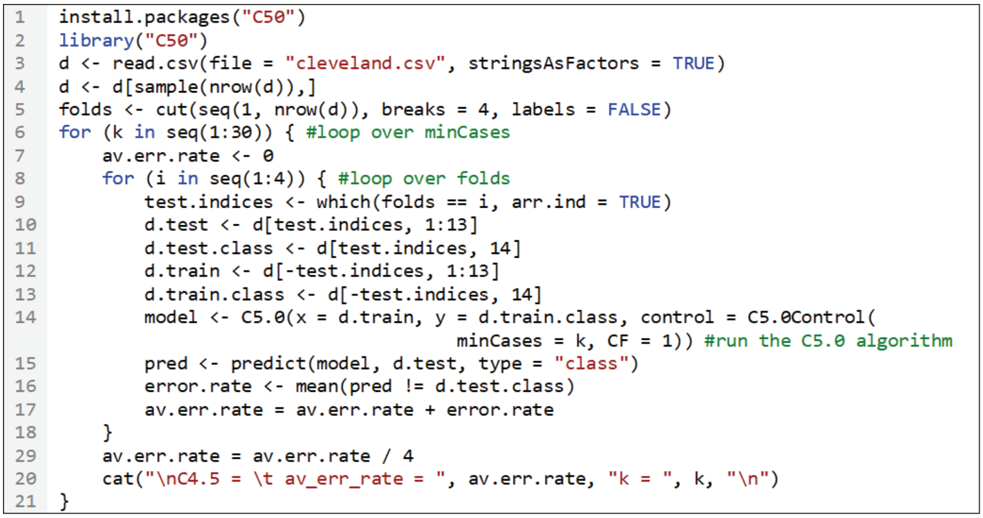


Figure 14. Average error rate for test and training data depending on the model size



```r
install.packages("C50")
library("C50")
d <- read.csv(file = "cleveland.csv", stringsAsFactors = TRUE)
d <- d[sample(nrow(d)),]
folds <- cut(seq(1, nrow(d)), breaks = 4, labels = FALSE)
for (k in seq(1:30)) { #loop over minCases
    av.err.rate <- 0
    for (i in seq(1:4)) { #loop over folds
        test.indices <- which(folds == i, arr.ind = TRUE)
        d.test <- d[test.indices, 1:13]
        d.test.class <- d[test.indices, 14]
        d.train <- d[-test.indices, 1:13]
        d.train.class <- d[-test.indices, 14]
        model <- C5.0(x = d.train, y = d.train.class, control = C5.0Control(
                                      minCases = k, CF = 1)) #run the C5.0 algorithm
        pred <- predict(model, d.test, type = "class")
        error.rate <- mean(pred != d.test.class)
        av.err.rate = av.err.rate + error.rate
    }
    av.err.rate = av.err.rate / 4
    cat("\nC4.5 = \t av_err_rate = ", av.err.rate, "k = ", k, "\n")
}
```

Figure 15. R script for the overfitting examination

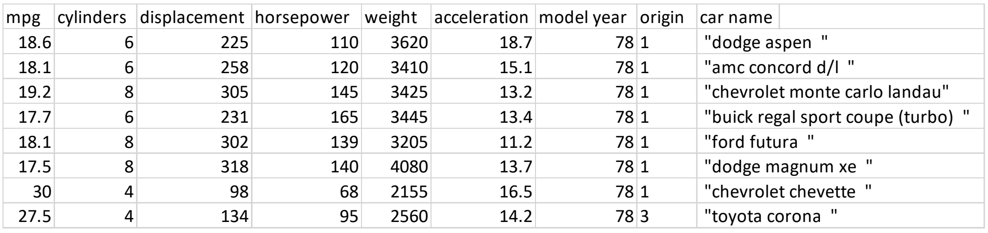
A clear overfitting of the created decision trees can be observed for small values of $minCases$ (great size trees). The average error rate on the test sets decreases with the decision tree size achieving the smallest value of about 0.17 if $minCases = 11$. Further reduction of the tree complexity deteriorates its results. In the case of the training sets, as was expected, the smallest error rate is achieved by the greatest tree (with $minCases = 1$).

5. Regression

Regression modeling is used to estimate the value of a continuous target (response) variable. In general, the regression model can approximate the relationship between a response variable and one or more predictor variables [2]. In what follows, we focus on so called simple linear regression where a single predictor variable is used, and thus, the model has the form of a straight line. The regression line (estimated regression line) can be written in the following form:

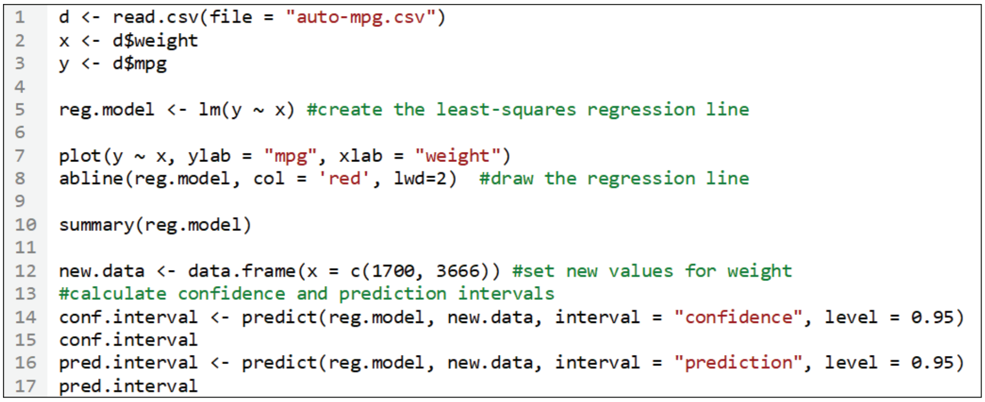
$$\hat{y} = b_1 x + b_0,$$

where $\hat{y}$ is the estimated value of the response variable, x is the predictor variable, b_0, b_1 are the regression coefficients, i.e. the y-intercept and the slope, respectively.



mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
18.6	6	225	110	3620	18.7	78	1	"dodge aspen "
18.1	6	258	120	3410	15.1	78	1	"amc concord d/l "
19.2	8	305	145	3425	13.2	78	1	"chevrolet monte carlo landau"
17.7	6	231	165	3445	13.4	78	1	"buick regal sport coupe (turbo) "
18.1	8	302	139	3205	11.2	78	1	"ford futura "
17.5	8	318	140	4080	13.7	78	1	"dodge magnum xe "
30	4	98	68	2155	16.5	78	1	"chevrolet chevette "
27.5	4	134	95	2560	14.2	78	3	"toyota corona "

Figure 16. Excerpt from data set “auto-mpg.csv”



```r
d <- read.csv(file = "auto-mpg.csv")
x <- d$weight
y <- d$mpg

reg.model <- lm(y ~ x) #create the least-squares regression line

plot(y ~ x, ylab = "mpg", xlab = "weight")
abline(reg.model, col = 'red', lwd=2)  #draw the regression line

summary(reg.model)

new.data <- data.frame(x = c(1700, 3666)) #set new values for weight
#calculate confidence and prediction intervals
conf.interval <- predict(reg.model, new.data, interval = "confidence", level = 0.95)
conf.interval
pred.interval <- predict(reg.model, new.data, interval = "prediction", level = 0.95)
pred.interval
```

Figure 17. R script for simple linear regression.

The regression line is created so as to minimize the sum of squared residuals over all the data points, where residual is the vertical distance from the data point to the regression line, $y - \hat{y}$.

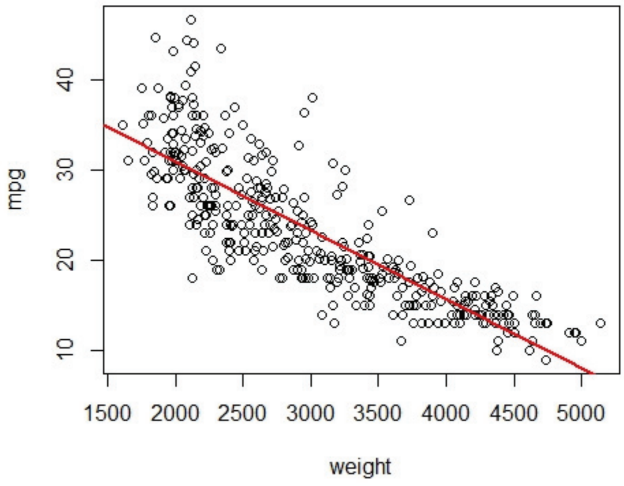
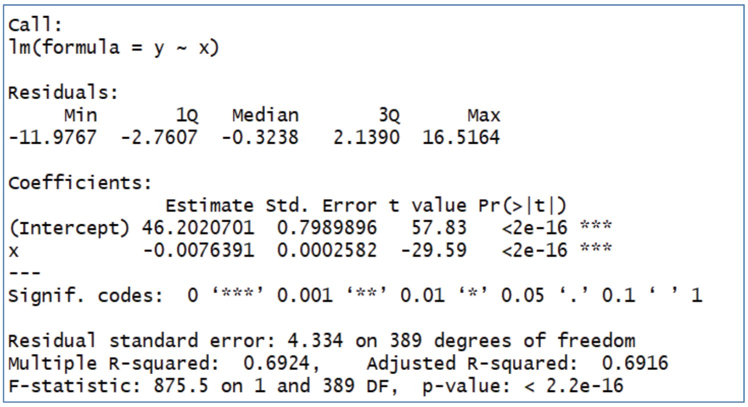


Figure 18. Scatter plot of mpg versus $weight$ and the regression line

In the experiment, data set “auto-mpg.csv” [15] is used (Figure 16) which contains 391 records with car characteristics (7 records with missing values were deleted). The regression line $\hat{y} = -0.0076391x + 46.2020701$, approximating the relationship between mpg (denoted by y) and $weight$ (denoted by x), is created by means of the lm function in line 5 of the R script (Figure 17) and shown as a red line in Figure 18. By calling function $summary$ (line 10) detailed information about the model and its quality can be obtained (see Figure 19) including:

- the values of the regression coefficients: $b_0 = 46.2020701$ and $b_1 = -0.0076391$;
- the minimal and maximal values of the residual: -11.9767 and 16.5164, respectively;
- the p-value which appears to be very low ($< 2e - 16$) for t-value equal to -29.59; this indicates that the null hypothesis H_0, which states that there is no relationship between y and x, can be rejected, t-value $= b_1/s_{b_1}$, $s_{b_1} = \sqrt{\sum_i (y_i - \hat{y}_i)^2/(n-2)}/\sqrt{\sum_i (x_i - \bar{x})^2}$, $s_{b_1}$is a standard error of b_1, it measures the variability in the slope of the regression line observed among the samples (large values of s_{b_1} tend to reduce the size of t-value), $\bar{x}$ is the mean of x.
- residual standard error (RSE) equal to 4.334, $RSE = \sqrt{\sum_i (y_i - \hat{y}_i)^2/(n-2)}$
- coefficient of determination $R^2 = 0.69$, R^2 measures how closely the linear regression fits the data, values of R^2 close to 1 indicate a perfect fit. $R^2 = SSR/SST$, where $SSR = \sum_i (\hat{y}_i - \bar{y})^2$ is the regression sum of squares representing the improvement in estimation resulting from using the regression model instead of the mean of y, $SST = \sum_i (y_i - \bar{y})^2$ is the total sum of squares representing the variability in y.

Using the regression model, the values of mpg are estimated for two new values of $weight$ and the confidence as well as the prediction intervals are determined with confidence level of 0.95 (lines 14 and 16 of the script). A confidence interval gives the range which is likely to include the mean of all values of y for a given x, while a prediction interval determines the range which is likely to include a randomly chosen value of y for a given x. The prediction interval is wider than the analogous confidence interval, as can be seen in Figure 20.



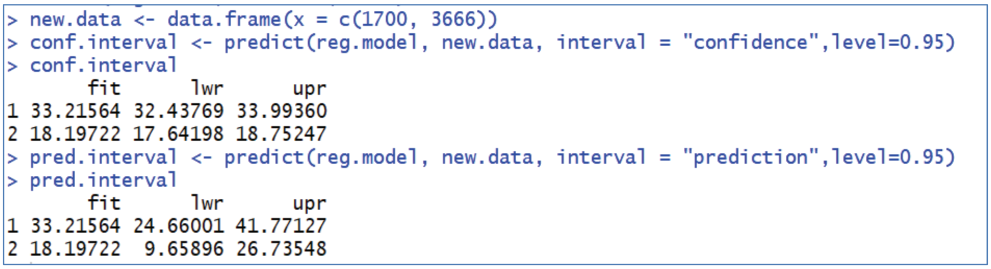
```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-11.9767  -2.7607  -0.3238   2.1390  16.5164

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 46.2020701  0.7989896   57.83   <2e-16 ***
x           -0.0076391  0.0002582  -29.59   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.334 on 389 degrees of freedom
Multiple R-squared:  0.6924,    Adjusted R-squared:  0.6916
F-statistic: 875.5 on 1 and 389 DF,  p-value: < 2.2e-16
```

Figure 19. Summary of the regression model



```
> new.data <- data.frame(x = c(1700, 3666))
> conf.interval <- predict(reg.model, new.data, interval = "confidence",level=0.95)
> conf.interval
       fit      lwr      upr
1 33.21564 32.43769 33.99360
2 18.19722 17.64198 18.75247
> pred.interval <- predict(reg.model, new.data, interval = "prediction",level=0.95)
> pred.interval
       fit      lwr      upr
1 33.21564 24.66001 41.77127
2 18.19722  9.65896 26.73548
```

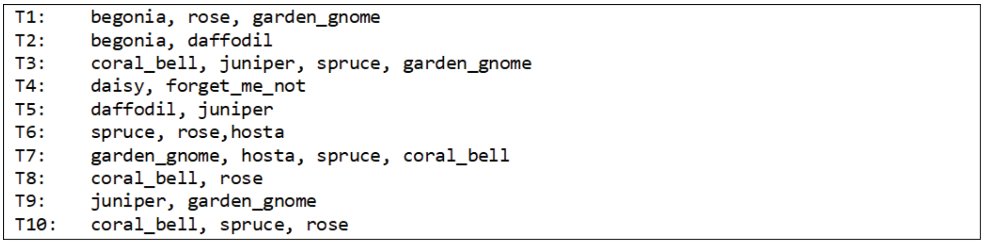
Figure 20. Confidence and prediction intervals for the new data

6. Association rules

Association rules is a technique of affinity analysis which seeks to uncover the associations among attributes [1, 2]. For example, association rules can be applied to finding which items are frequently purchased together, which in turn can help in establishing a profitable sales strategy. Another example is to predicting the clinical response to anti-cancer drugs for individual patients (in combination with other methods) [16].

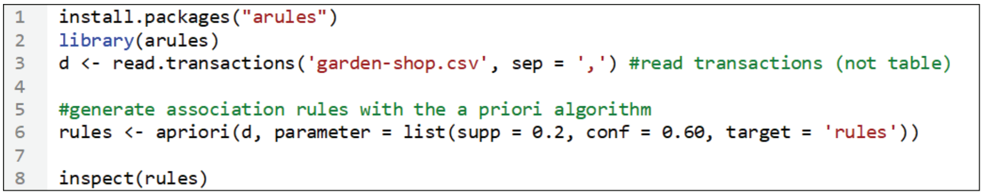
Association rules take the form: $if\ antecedent\ then\ consequent$. They are created by searching a data set for frequent patterns and applying support and confidence criteria to identify the most important relationships.

Suppose that we are given a set containing 10 customer transactions from a garden shop as shown in Figure 21.



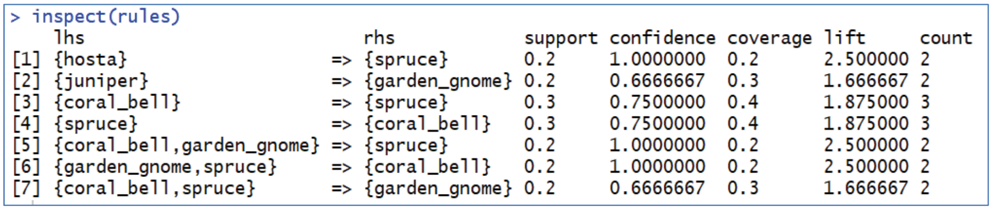
```
T1:    begonia, rose, garden_gnome
T2:    begonia, daffodil
T3:    coral_bell, juniper, spruce, garden_gnome
T4:    daisy, forget_me_not
T5:    daffodil, juniper
T6:    spruce, rose,hosta
T7:    garden_gnome, hosta, spruce, coral_bell
T8:    coral_bell, rose
T9:    juniper, garden_gnome
T10:   coral_bell, spruce, rose
```

Figure 21. Transactions in a garden shop



```
1  install.packages("arules")
2  library(arules)
3  d <- read.transactions('garden-shop.csv', sep = ',') #read transactions (not table)
4
5  #generate association rules with the a priori algorithm
6  rules <- apriori(d, parameter = list(supp = 0.2, conf = 0.60, target = 'rules'))
7
8  inspect(rules)
```

Figure 22. R script generating association rules for transactions in the garden shop



```
> inspect(rules)
    lhs                          rhs            support confidence coverage lift     count
[1] {hosta}                   => {spruce}       0.2     1.0000000  0.2      2.500000 2
[2] {juniper}                 => {garden_gnome} 0.2     0.6666667  0.3      1.666667 2
[3] {coral_bell}              => {spruce}       0.3     0.7500000  0.4      1.875000 3
[4] {spruce}                  => {coral_bell}   0.3     0.7500000  0.4      1.875000 3
[5] {coral_bell,garden_gnome} => {spruce}       0.2     1.0000000  0.2      2.500000 2
[6] {garden_gnome,spruce}     => {coral_bell}   0.2     1.0000000  0.2      2.500000 2
[7] {coral_bell,spruce}       => {garden_gnome} 0.2     0.6666667  0.3      1.666667 2
```

Figure 23. Association rules for transactions in the garden shop

The R script is presented in Figure 22. In line 6, function *apriori*, generating association rules is called with the minimal required values for support and confidence set at 0.2 and 0.6, respectively. Figure 23 shows the resulting association rules along with their evaluation given by support, confidence, coverage and lift, which are explained below.

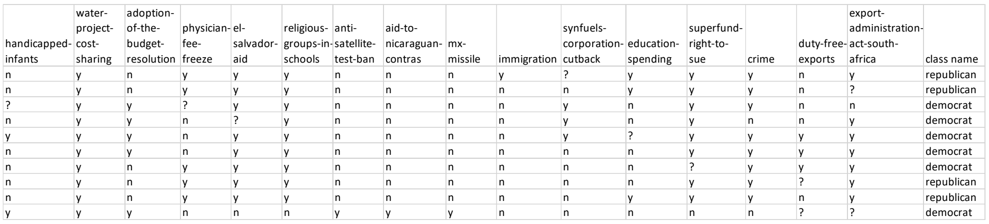
Let us focus on association rule 7 with antecedent $A = \{coral\_bell, spruce\}$ and consequent $C = \{garden\_gnome\}$. Observe that three transactions (T3, T7 and T10) contain itemset A (i.e. items $coral\_bell$ and $spruce$), four transactions (T1, T3, T7 and T9) contain itemset C (i.e. $garden\_gnome$) and two transactions (T3 and T7) contain itemset $A \cup C$ (i.e. $coral\_bell$, $spruce$ and $garden\_gnome$). The rule quality is evaluated as follows [17].

- $support(A \Rightarrow C) = \frac{number\ of\ transactions\ containing\ both\ A\ and\ C}{total\ number\ of\ transactions}$. Support shows how popular is an itemset containing items from A and C among all the transactions. For rule 7, support is equal to $2/10$.

- $confidence(A \Rightarrow C) = \frac{number\ of\ transactions\ containing\ both\ A\ and\ C}{number\ of\ transactions\ containing\ A}$. Confidence determines how frequently items in C appear in transactions that contain A. For rule 7, confidence is equal to $2/3$.
- $coverage(A \Rightarrow C) = support(A)$, thus, for rule 7, coverage is equal to $3/10$.
- $lift(A \Rightarrow C) = \frac{confidence(A \Rightarrow C)}{support(C)}$. Lift is the ratio between the rule confidence and the support of the itemset in the rule consequent. For rule 7, lift has value of $(2/3)/(4/10) = 1.67$.

Function $apriori$ used in the example implements the a priori algorithm. In general, algorithms creating association rules must apply some mechanisms for the reduction of search space as the number of possible association rules increases exponentially with the number of attributes.

The a priori algorithm works with so called frequent itemsets (an itemset is frequent if it appears in at least a certain number of transactions) and takes advantage of the a priori property which states the if an itemset, denoted by Z, is not frequent then for any item i, $Z \cup \{i\}$ will not be frequent. More precisely, in successive iterations, the algorithm creates candidate k-itemsets (sets with k items) on the basis of frequent $(k-1)$-itemsets, prunes candidates by applying the a priory property, and takes the remaining frequent candidates as frequent k-itemsets. Then, from the created frequent itemsets, it generates association rules satisfying the minimum support and confidence conditions.

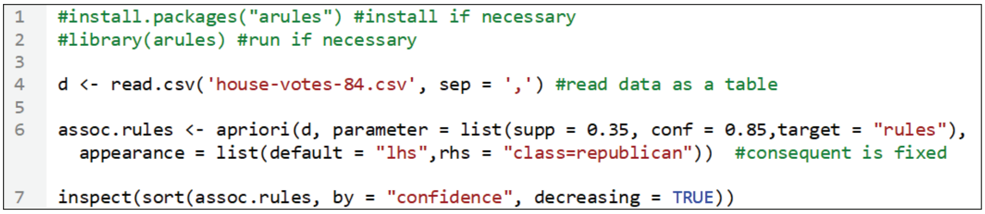


handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras	mx-missile	immigration	synfuels-corporation-cutback	education-spending	superfund-right-to-sue	crime	duty-free-exports	export-administration-act-south-africa	class name
n	y	n	y	y	y	n	n	n	y	?	y	y	y	n	y	republican
n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	?	republican
?	y	y	?	y	y	n	n	n	n	y	n	y	y	n	n	democrat
n	y	y	n	?	y	n	n	n	n	y	n	y	n	n	y	democrat
y	y	y	n	y	y	n	n	n	n	y	?	y	y	y	y	democrat
n	y	y	n	y	y	n	n	n	n	n	n	y	y	y	y	democrat
n	y	n	y	y	y	n	n	n	n	n	n	?	y	y	y	democrat
n	y	n	y	y	y	n	n	n	n	n	n	y	y	?	y	republican
n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	y	republican
y	y	y	n	n	n	y	y	y	n	n	n	n	n	?	?	democrat

Figure 24. Excerpt from data set "house-votes-84.csv"

In the next example, association rules are created in which a consequent is fixed. Data set "house-votes-84.csv" is used which includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes [18], see Figure 24. Unknown votes are replaced by randomly chosen values of y and n.

The R script (Figure 25) creates association rules with the value of consequent set to $republican$. The rules with support and confidence not less than 0.35 and 0.85, respectively, are shown in Figure 26.



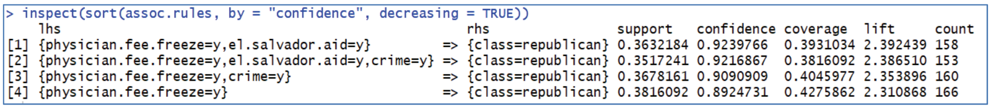
```
#install.packages("arules") #install if necessary
#library(arules) #run if necessary

d <- read.csv('house-votes-84.csv', sep = ',') #read data as a table

assoc.rules <- apriori(d, parameter = list(supp = 0.35, conf = 0.85,target = "rules"),
  appearance = list(default = "lhs",rhs = "class=republican"))  #consequent is fixed

inspect(sort(assoc.rules, by = "confidence", decreasing = TRUE))
```

Figure 25. R script generating association rules for voting results



```
> inspect(sort(assoc.rules, by = "confidence", decreasing = TRUE))
    lhs                                                rhs                support   confidence coverage  lift     count
[1] {physician.fee.freeze=y,el.salvador.aid=y}       => {class=republican} 0.3632184 0.9239766  0.3931034 2.392439 158
[2] {physician.fee.freeze=y,el.salvador.aid=y,crime=y} => {class=republican} 0.3517241 0.9216867  0.3816092 2.386510 153
[3] {physician.fee.freeze=y,crime=y}                 => {class=republican} 0.3678161 0.9090909  0.4045977 2.353896 160
[4] {physician.fee.freeze=y}                         => {class=republican} 0.3816092 0.8924731  0.4275862 2.310868 166
```

Figure 26. Association rules with $republican$ as a consequent for voting results.

7. Summary

The aim of the paper was to show how the most common data mining tasks, such as clustering, classification, prediction, association as well as data cleaning, can be dealt with using the R programming language. The data mining tasks were introduced along with some solution methods. A number of illustrative examples were provided, each presenting the R script and explaining the results obtained for benchmark or real-life data sets.

Literature

[1] D.T. Larose, Ch.D. Larose, *Discovering Knowledge in Data. An Introduction to Data Mining*, Hoboken: Wiley, 2014.

[2] D.T. Larose, Ch.D. Larose, *Data Mining and Predictive Analytics*, Hoboken: Wiley, 2015.

[3] *Data sets distributed with R: airquality data set*. [On-line]. https://forge.scilab.org/index.php/p/rdataset/source/tree/master/csv/datasets/airquality.csv. [18.11.2020].

[4] I. Ben-Gal, *Outlier Detection*. [in:] *Data Mining and Knowledge Discovery Handbook*. (Eds.) O. Maimon and L. Rokach., Boston, MA: Springer, 117-130, 2005. https://doi.org/10.1007/0-387-25465-X_7.

[5] *An Introduction to Outliers – What are Outliers – Types of Outliers*. [On-line]. https://www.anblicks.com/resources/insights-blogs/an-introduction-to-outliers/. [18.11.2020].

[6] R. Rakotomalala, *Tanagra: Body Mass Index Data Set*. [On-line]. eric.univ-lyon2.fr/~ricco/tanagra/fichiers/body_mass_index.xls. [18.11.2020].

[7] G. Seif, *The 5 Clustering Algorithms Data Scientists Need to Know*. [On-line]. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68. [18.11.2020].

[8] P. Fränti, S. Sieranoja, *K-means properties on six clustering benchmark datasets*, "Applied Intelligence" 48 (12), 4743-4759, 2018. https://doi.org/10.1007/s10489-018-1238-7.

[9] P. Fränti, S. Sieranoja, *Clustering basic benchmark: Aggregation*, [On-line]. http://cs.joensuu.fi/sipu/datasets/. [18.11.2020].

[10] H. You, G. Rumbe, *Comparative Study of Classification Techniques on Breast Cancer FNA Biopsy Data*, "International Journal of Interactive Multimedia and Artificial Intelligence" 1(3): 5-12, 2010.

[11] C. Elkan, *Predictive analytics and data mining*, [On-line]. https://www.researchgate.net/publication/228780185_Predictive_analytics_and_data_mining. [20.11.2020].

[12] *UCI Machine Learning Repository: Seeds Data Set*, [On-line]. https://archive.ics.uci.edu/ml/datasets/seeds. [20.11.2020].

[13] *UCI Machine Learning Repository: Heart Disease Data Set*. [On-line]. https://archive.ics.uci.edu/ml/datasets/heart+disease. [20.11.2020].

[14] *RDocumentation: C5.0 Control*. [On-line]. https://www.rdocumentation.org/packages/C50/versions/0.1.3.1/topics/C5.0Control. [20.11.2020],

[15] *UCI Machine Learning Repository: Auto MPG Data Set*, [On-line]. https://archive.ics.uci.edu/ml/datasets/auto+mpg. [20.11.2020].

[16] K. Vougas, M. Krochmal, T. Jackson, A. Polyzos, A. Aggelopoulos, I.S. Pateras, M. Liontos, A. Varvarigou, E.O. Johnson, V. Georgoulias, A. Vlahou, P. Townsend, D. Thanos, J. Bartek, V. G. Gorgoulis, *Deep Learning and Association Rule Mining for Predicting Drug Response in Cancer. A Personalised Medicine Approach*, bioRxiv. [On-line]. https://www.biorxiv.org/content/10.1101/070490v3.full. [24.11.2020].

[17] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer, 2009.

[18] *UCI Machine Learning Repository: Congressional Voting Records Data Set*. [On-line]. https://archive.ics.uci.edu/ml/datasets/congressional+voting+records. [20.11.2020].