# OneR - Establishing a New Baseline for Machine Learning Classification Models

***An R package by Holger K. von Jouanne-Diedrich***

***2016-10-24***

**Note:** You can find a step-by-step introduction on YouTube: Quick Start Guide for the OneR package

## Introduction

The following story is one of the most often told in the Data Science community: Some time ago the military built a system which aim it was to distinguish military vehicles from civilian ones. They chose a neural network approach and trained the system with pictures of tanks, humvees and missile launchers on the one hand and normal cars, pickups and trucks on the other. After having reached a satisfactory accuracy they brought the system into the field (quite literally). It failed completely, performing no better than a coin toss. What had happened? No one knew, so they re-engineered the black box (no small feat in itself) and found that most of the military pics where taken at dusk or dawn and most civilian pics under brighter weather conditions. The neural net had learned the difference between light and dark!

Although this might be an urban legend the fact that it is so often told wants to tell us something:

1. Many of our Machine Learning models are so complex that we cannot understand them ourselves.
2. Because of 1. we cannot differentiate between the simpler aspects of a problem which can be tackled by simple models and the more sophisticated ones which need specialized treatment.

The above is not only true for neural networks (and especially deep neural networks) but for most of the methods used today, especially Support Vector Machines and Random Forests and in general all kinds of ensemble based methods.

In one word: We need a good baseline which builds “the best simple model” that strikes a balance between the best accuracy possible with a model that is still simple enough to understand: I have developed the OneR package for finding this sweet spot and thereby establishing a new baseline for classification models in Machine Learning (ML).

This package is filling a longstanding gap because only a JAVA based implementation was available so far (RWeka package as an interface for the OneR JAVA class). Additionally several enhancements have been made (see below).

## Design principles for the OneR package

The following design principles were followed for programming the package:

- Easy: The learning curve for new users should be minimal. Results should be obtained with ease and only minimal preprocessing and modeling steps should be necessary.
- Versatile: All types of data, i.e. categorical and numeric, should be computable - as input variable as well as as target.
- Fast: The running times of model trainings should be short.
- Accurate: The accuracy of trained models should be good overall.

- Robust: Models should not be prone to overfitting; the reached accuracy on training data should be comparable to the accuracy of predictions from new, unseen cases.
- Comprehensible: It should be easy to understand which rules the model has learned. Not only should the rules be easily comprehensible but they should serve as heuristics that are usable even without a computer.
- Reproducible: Because the used algorithms are strictly deterministic one will always get the same models on the same data. Many ML algorithms have stochastic components so that the data scientist will get a different model every time.
- Intuitive: Model diagnostics should be presented in form of simple tables and plots.
- Native R: The whole package is written in native R code. Thereby the source code can be easily checked and the whole package is very lean. Additionally the package has no dependencies at all other than base R itself.

The package is based on the – as the name might reveal – one rule classification algorithm [Holte93]. Although the underlying method is simple enough (basically 1-level decision trees, you can find out more here: OneR) several enhancements have been made:

- Discretization of numeric data: The OneR algorithm can only handle categorical data, so numeric data has to be discretized. The original OneR algorithm separates the respective values in ever smaller and smaller buckets until the best possible accuracy is being reached. It can be argued that this is the definition of overfitting and contradicts the original spirit of OneR because tons of rules (one for every bucket) will result. One can of course introduce a new parameter “maximum bucket size” but finding the right value for this one doesn’t come naturally either. Therefore I take a radically different approach: There are several methods for handling numeric data in the package (in the bin and the optbin function), the most promising one is the (default) “logreg” method in the optbin function which gives only as many bins as there are target categories and which optimizes the cut points according to pairwise logistic regressions.
- Missing values: In the original algorithm missing values were always handled as a separate level in the respective attribute. While missing values can sometimes reveal interesting patterns in other cases they are, well, just values that are missing. In the OneR package missing values can be handled as separate levels (level “NA”) or they can be omitted (the default).
- Tie breaking: Sometimes the OneR algorithm will find several attributes that provide rules which all give the same best accuracy. The original algorithm just took the first attribute. While this is implemented in the OneR function as the default too a different method for tie breaking can be chosen: The contingency tables of all “best” rules are tested against each other with a Pearson’s Chi squared test and the one with the smallest p-value is being chosen. The rationale behind this is that thereby the attribute with the best signal-to-noise ratio is being found.

## Getting started with a simple example

You can also watch this video which goes through the following example step-by-step:

Quick Start Guide for the OneR package (Video)

After installing from CRAN load package

```
library(OneR)
```

Use the famous Iris dataset and determine optimal bins for numeric data

```
data <- optbin(iris)
```

Build model with best predictor

```
model <- OneR(data, verbose = TRUE)
```

```
##
##     Attribute    Accuracy
## 1 * Petal.Width  96%
## 2   Petal.Length 95.33%
## 3   Sepal.Length 74.67%
## 4   Sepal.Width  55.33%
## ---
## Chosen attribute due to accuracy
## and ties method (if applicable): '*'
```

Show learned rules and model diagnostics

```
summary(model)
```

```
##
## Call:
## OneR(data = data, verbose = TRUE)
##
## Rules:
## If Petal.Width = (0.0976,0.791] then Species = setosa
## If Petal.Width = (0.791,1.63]   then Species = versicolor
## If Petal.Width = (1.63,2.5]     then Species = virginica
##
## Accuracy:
## 144 of 150 instances classified correctly (96%)
##
## Contingency table:
##             Petal.Width
## Species      (0.0976,0.791] (0.791,1.63] (1.63,2.5] Sum
##   setosa               * 50            0          0  50
##   versicolor              0         * 48          2  50
##   virginica               0            4       * 46  50
##   Sum                    50           52         48 150
## ---
## Maximum in each column: '*'
##
## Pearson's Chi-squared test:
## X-squared = 266.35, df = 4, p-value < 2.2e-16
```

Plot model diagnostics

```
plot(model)
```


Use model to predict data

```
prediction <- predict(model, data)
```

Evaluate prediction statistics

```
eval_model(prediction, data)
```

```
##
## Confusion matrix (absolute):
##            Actual
## Prediction   setosa versicolor virginica Sum
##   setosa         50          0         0  50
##   versicolor      0         48         4  52
##   virginica       0          2        46  48
##   Sum            50         50        50 150
##
## Confusion matrix (relative):
##            Actual
## Prediction   setosa versicolor virginica  Sum
##   setosa       0.33       0.00      0.00 0.33
##   versicolor   0.00       0.32      0.03 0.35
##   virginica    0.00       0.01      0.31 0.32
##   Sum          0.33       0.33      0.33 1.00
```

```
##
## Accuracy:
## 0.96 (144/150)
##
## Error rate:
## 0.04 (6/150)
##
## Error rate reduction (vs. base rate):
## 0.94 (p-value < 2.2e-16)
```

Please note that the very good accuracy of 96% is reached effortlessly.

“Petal.Width” is identified as the attribute with the highest predictive value. The cut points of the intervals are found automatically (via the included optbin function). The results are three very simple, yet accurate, rules to predict the respective species.

The nearly perfect separation of the areas in the diagnostic plot give a good indication of the model’s ability to separate the different species.

## A more sophisticated real-world example

The next example tries to find a model for the identification of breast cancer. The data were obtained from the UCI machine learning repository (see also the package documentation). According to this source the best out-of-sample performance was 95.9%, so let’s see what we can achieve with the OneR package…

```
data(breastcancer)
data <- breastcancer
```

Divide training (80%) and test set (20%)

```
set.seed(12) # for reproducibility
random <- sample(1:nrow(data), 0.8 * nrow(data))
data_train <- optbin(data[random, ], method = "infogain")
```

```
## Warning in optbin(data[random, ], method = "infogain"): 12 instance(s)
## removed due to missing values
```

```
data_test <- data[-random, ]
```

Train OneR model on training set

```
model_train <- OneR(data_train, verbose = TRUE)
```

```
##
##     Attribute                  Accuracy
## 1 * Uniformity of Cell Size    92.32%
## 2   Uniformity of Cell Shape   91.59%
## 3   Bare Nuclei                90.68%
```

```
## 4   Bland Chromatin             90.31%
## 5   Normal Nucleoli             90.13%
## 6   Single Epithelial Cell Size 89.4%
## 7   Marginal Adhesion           85.92%
## 8   Clump Thickness             84.28%
## 9   Mitoses                     78.24%
## ---
## Chosen attribute due to accuracy
## and ties method (if applicable): '*'
```

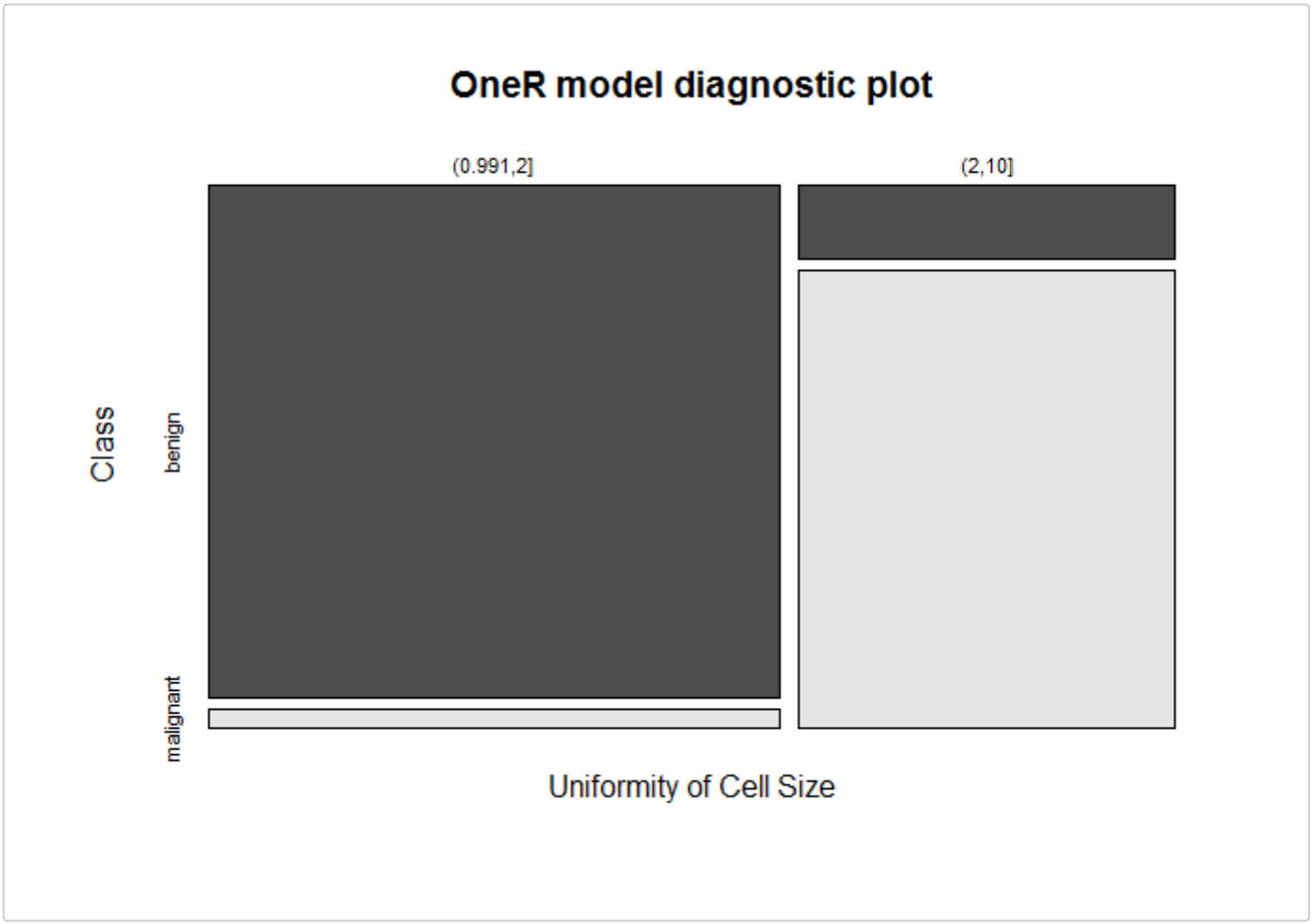
### Show model and diagnostics

```
summary(model_train)
```

```
##
## Call:
## OneR(data = data_train, verbose = TRUE)
##
## Rules:
## If Uniformity of Cell Size = (0.991,2] then Class = benign
## If Uniformity of Cell Size = (2,10]    then Class = malignant
##
## Accuracy:
## 505 of 547 instances classified correctly (92.32%)
##
## Contingency table:
##            Uniformity of Cell Size
## Class       (0.991,2] (2,10] Sum
##   benign        * 318     30 348
##   malignant        12  * 187 199
##   Sum             330    217 547
## ---
## Maximum in each column: '*'
##
## Pearson's Chi-squared test:
## X-squared = 381.78, df = 1, p-value < 2.2e-16
```

### Plot model diagnostics

```
plot(model_train)
```




Use trained model to predict test set

```
prediction <- predict(model_train, data_test)
```

Evaluate model performance on test set

```
eval_model(prediction, data_test)
```

```
##
## Confusion matrix (absolute):
##            Actual
## Prediction  benign malignant Sum
##   benign        92         0  92
##   malignant      8        40  48
##   Sum          100        40 140
##
## Confusion matrix (relative):
##            Actual
## Prediction  benign malignant  Sum
##   benign      0.66      0.00 0.66
##   malignant   0.06      0.29 0.34
##   Sum         0.71      0.29 1.00
##
## Accuracy:
```

```
## 0.9429 (132/140)
##
## Error rate:
## 0.0571 (8/140)
##
## Error rate reduction (vs. base rate):
## 0.8 (p-value = 7.993e-12)
```

The best reported out-of-sample accuracy on this dataset was at 95.9% and it was reached with considerable effort. The reached accuracy for the test set here lies at 94.3%! This is achieved with just one simple rule that when “Uniformity of Cell Size” is bigger than 2 the examined tissue is malignant. The cut points of the intervals are again found automatically (via the included optbin function). The very good separation of the areas in the diagnostic plot give a good indication of the model’s ability to differentiate between benign and malignant tissue. Additionally when you look at the distribution of misclassifications not a single malignant instance is missed, which is obviously very desirable in a clinical context.

# Included functions

## OneR

OneR is the main function of the package. It builds a model according to the One Rule machine learning algorithm for categorical data. All numerical data is automatically converted into five categorical bins of equal length. When verbose is TRUE it gives the predictive accuracy of the attributes in decreasing order.

## bin

bin discretizes all numerical data in a dataframe into categorical bins of equal length or equal content or based on automatically determined clusters.

Examples

```
data <- iris
str(data)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
str(bin(data))
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: Factor w/ 5 levels "(4.3,5.02]","(5.02,5.74]",..: 2 1 1 1 1 2 1 1 1 1 ...
##  $ Sepal.Width : Factor w/ 5 levels "(2,2.48]","(2.48,2.96]",..: 4 3 3 3 4 4 3 3 2 3 ...
##  $ Petal.Length: Factor w/ 5 levels "(0.994,2.18]",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ Petal.Width : Factor w/ 5 levels "(0.0976,0.58]",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
str(bin(data, nbins = 3))
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: Factor w/ 3 levels "(4.3,5.5]","(5.5,6.7]",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sepal.Width : Factor w/ 3 levels "(2,2.8]","(2.8,3.6]",..: 2 2 2 2 2 3 2 2 2 2 ...
##  $ Petal.Length: Factor w/ 3 levels "(0.994,2.97]",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Petal.Width : Factor w/ 3 levels "(0.0976,0.9]",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
str(bin(data, nbins = 3, labels = c("small", "medium", "large")))
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: Factor w/ 3 levels "small","medium",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sepal.Width : Factor w/ 3 levels "small","medium",..: 2 2 2 2 2 3 2 2 2 2 ...
##  $ Petal.Length: Factor w/ 3 levels "small","medium",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Petal.Width : Factor w/ 3 levels "small","medium",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Difference between methods “length” and “content”

```
set.seed(1); table(bin(rnorm(900), nbins = 3))
```

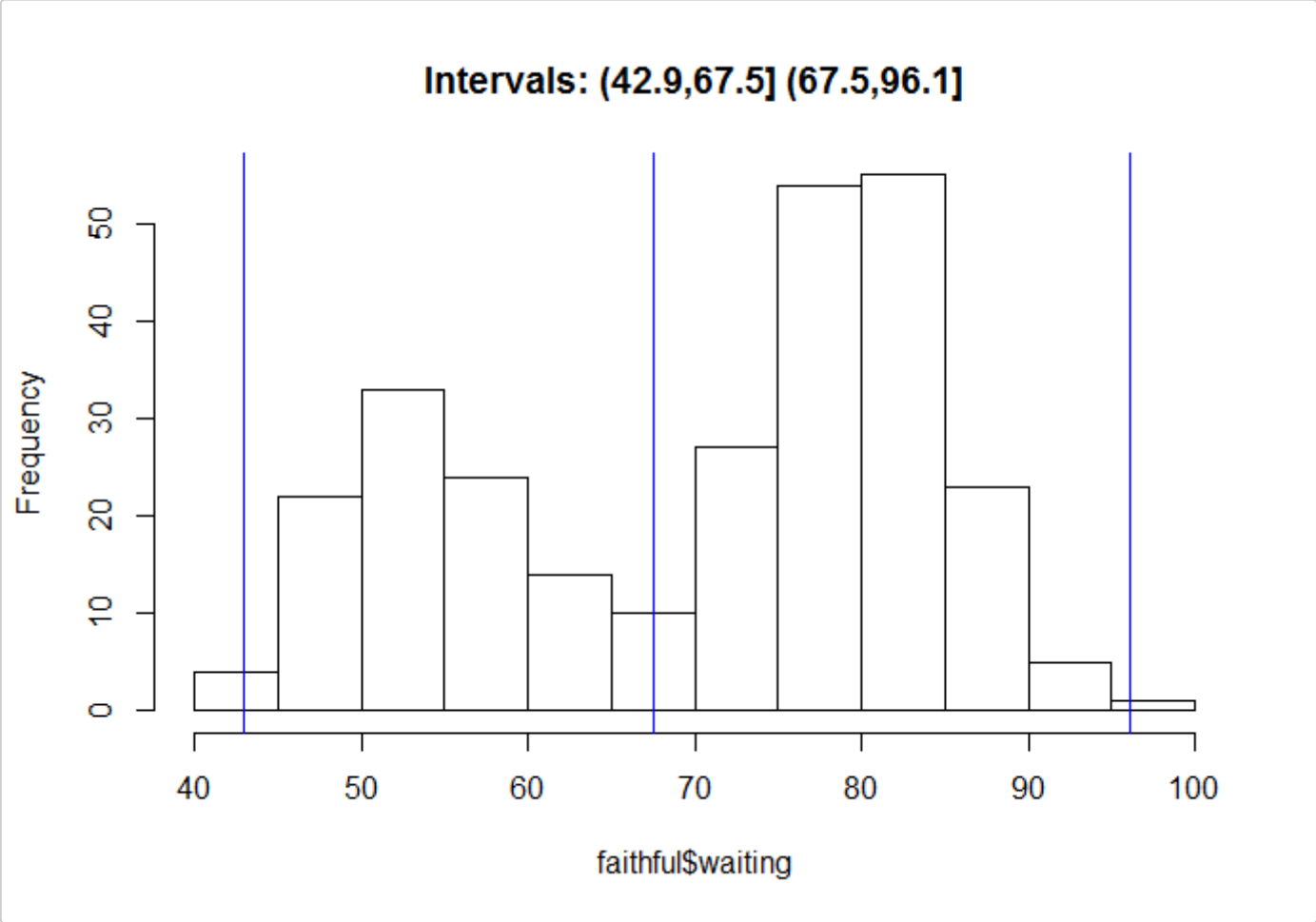
```
##
## (-3.01,-0.735]  (-0.735,1.54]    (1.54,3.82]
##            212            623             65
```

```
set.seed(1); table(bin(rnorm(900), nbins = 3, method = "content"))
```

```
##
## (-3.01,-0.423] (-0.423,0.444]   (0.444,3.82]
##            300            300            300
```

Method “clusters”

```
intervals <- paste(levels(bin(faithful$waiting, nbins = 2, method = "cluster")), collapse = " ")
hist(faithful$waiting, main = paste("Intervals:", intervals))
abline(v = c(42.9, 67.5, 96.1), col = "blue")
```




Handling of missing values

```
bin(c(1:10, NA), nbins = 2, na.omit = FALSE) # adds new level "NA"
```

```
##  [1] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5]
##  [6] (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]
## [11] NA
## Levels: (0.991,5.5] (5.5,10] NA
```

```
bin(c(1:10, NA), nbins = 2)
```

```
## Warning in bin(c(1:10, NA), nbins = 2): 1 instance(s) removed due to
## missing values
```

```
##  [1] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5]
##  [6] (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]
## Levels: (0.991,5.5] (5.5,10]
```

## optbin

optbin discretizes all numerical data in a dataframe into categorical bins where the cut points are optimally aligned with the target categories, thereby a factor is returned. When building a OneR model this could result in fewer rules with enhanced accuracy. The cutpoints are calculated by pairwise logistic regressions (method “logreg”) or as the means of the expected values of the respective classes (“naive”). The function is likely to give unsatisfactory results when the distributions of the respective classes are not (linearly) separable. Method “naive” should only be used when distributions are (approximately) normal, although in this case “logreg” should give comparable results, so it is the preferable (and therefore default) method.

Method “infogain” is an entropy based method which calculates cut points based on information gain. The idea is that uncertainty is minimized by making the resulting bins as pure as possible. This method is the standard method of many decision tree algorithms.

## maxlevels

maxlavels removes all columns of a dataframe where a factor (or character string) has more than a maximum number of levels. Often categories that have very many levels are not useful in modelling OneR rules because they result in too many rules and tend to overfit. Examples are IDs or names.

```
df <- data.frame(numeric = c(1:26), alphabet = letters)
str(df)
```

```
## 'data.frame':    26 obs. of  2 variables:
##  $ numeric : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ alphabet: Factor w/ 26 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
```

```
str(maxlevels(df))
```

```
## 'data.frame':    26 obs. of  1 variable:
##  $ numeric: int  1 2 3 4 5 6 7 8 9 10 ...
```

## predict

predict is a S3 method for predicting cases or probabilites based on OneR model objects. The second argument “newdata”" can have the same format as used for building the model but must at least have the feature variable that is used in the OneR rules. The default output is a factor with the predicted classes.

```
model <- OneR(iris)
predict(model, data.frame(Petal.Width = seq(0, 3, 0.5)))
```

```
## (-Inf,0.0976] (0.0976,0.58]   (0.58,1.06]   (1.06,1.54]   (1.54,2.02]
##        UNSEEN        setosa    versicolor    versicolor     virginica
##    (2.02,2.5]    (2.5, Inf]
##     virginica        UNSEEN
## Levels: UNSEEN setosa versicolor virginica
```

If “type = prob” a matrix is returned whose columns are the probability of the first, second, etc. class.

```
predict(model, data.frame(Petal.Width = seq(0, 3, 0.5)), type = "prob")
```

```
##               setosa versicolor  virginica
## (-Inf,0.0976]     NA         NA         NA
## (0.0976,0.58]  1.000  0.0000000 0.00000000
## (0.58,1.06]    0.125  0.8750000 0.00000000
## (1.06,1.54]    0.000  0.9268293 0.07317073
## (1.54,2.02]    0.000  0.1724138 0.82758621
## (2.02,2.5]     0.000  0.0000000 1.00000000
## (2.5, Inf]        NA         NA         NA
```

### eval_model

eval_model is a simple function for evaluating a OneR classification model. It prints confusion matrices with prediction vs. actual in absolute and relative numbers. Additionally it gives the accuracy, error rate as well as the error rate reduction versus the base rate accuracy together with a p-value. The second argument “actual” is a dataframe which contains the actual data in the last column. A single vector is allowed too.

For the details please consult the available help entries.

## Help overview

From within R:

```
help(package = OneR)
```

…or as a pdf here: OneR.pdf

Issues can be posted here: https://github.com/vonjd/OneR/issues

The latest version of the package (and full sourcecode) can be found here: https://github.com/vonjd/OneR

## Sources

[Holte93] R. Holte: Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, 1993. Available online here: http://www.mlpack.org/papers/ds.pdf.

## Contact

I would love to hear about your experiences with the OneR package. Please drop me a note - you can reach me at my university account: Holger K. von Jouanne-Diedrich

## License