recorder 0.8.1 is now available on CRAN. recorder is a lightweight toolkit to  
validate new observations before computing their corresponding predictions with  
a predictive model.

With recorder the validation process consists of two steps:

1. record relevant statistics and meta data of the variables in the original  
   training data for the predictive model
2. use these data to run a set of basic validation tests on the new set of  
   observations.

Now we will take a deeper look into, what recorder has to offer.

**[PLAY]**

**Motivation**

There can be many data specific reasons, why you might not be confident in the  
predictions of a predictive model on new data.

Some of them are obvious, e.g.:

* One or more variables in training data are not found in new data
* The class of a given variable differs in training data and new data

Others are more subtle, for instance when observations in  
new data are not within the “span” of the training data. One example of this could  
be, when a variable is “N/A” (missing) for a new observation to be predicted,  
but no missing values appeared for the same variable in the training data.  
This implies, that the new observation is not within the “span” of the training  
data. Another way of putting this: the model has never encountered an  
observation like this before, therefore there is good reason to doubt the  
quality of the prediction.

**recorder workflow**

We will need some data in order to demonstrate the recorder workflow. As so  
many times before the famous iris data set will be used as an example. The  
data set is divided into training data, that can be used for model development,  
and new data for predictions after modelling, which we can validate with  
recordr.

set.seed(1)

trn\_idx <- sample(seq\_len(nrow(iris)), 100)

data\_training <- iris[trn\_idx, ]

data\_new <- iris[-trn\_idx, ]

**Record statistics and meta data of variables in training data**

What we want to achieve is to validate the new observations (before computing  
their predictions with a predictive model) based on relevant  
statistics and meta data of the variables in the training data. Therefore  
relevant statistics and meta data of the variables must first be learned  
(recorded) from the trainingdata of the model. This is done with the record()  
function.

library(recorder)

tape <- record(data\_training)

#>

#> [RECORD]

#>

#> ... recording meta data and statistics of 100 rows with 5 columns...

#>

#> [STOP]

This provides us with an object belonging to the data.tape class.  
The data.tape contains the statistics and meta data recorded from the training  
data.

str(tape)

#> List of 2

#> $ class\_variables:List of 5

#> ..$ Sepal.Length: chr "numeric"

#> ..$ Sepal.Width : chr "numeric"

#> ..$ Petal.Length: chr "numeric"

#> ..$ Petal.Width : chr "numeric"

#> ..$ Species : chr "factor"

#> $ parameters :List of 5

#> ..$ Sepal.Length:List of 3

#> .. ..$ min : num 4.3

#> .. ..$ max : num 7.9

#> .. ..$ any\_NA: logi FALSE

#> ..$ Sepal.Width :List of 3

#> .. ..$ min : num 2

#> .. ..$ max : num 4.2

#> .. ..$ any\_NA: logi FALSE

#> ..$ Petal.Length:List of 3

#> .. ..$ min : num 1

#> .. ..$ max : num 6.9

#> .. ..$ any\_NA: logi FALSE

#> ..$ Petal.Width :List of 3

#> .. ..$ min : num 0.1

#> .. ..$ max : num 2.5

#> .. ..$ any\_NA: logi FALSE

#> ..$ Species :List of 2

#> .. ..$ levels: chr [1:3] "setosa" "versicolor" "virginica"

#> .. ..$ any\_NA: logi FALSE

#> - attr(\*, "class")= chr [1:2] "list" "data.tape"

As you see, which meta data and statistics are recorded for the individual  
variables depends on the class of the given variable, e.g. for a numeric  
variable min and max values are computed, whilst levels is recorded for  
factor variables.

**Validate new data**

First, to spice things up, we will give the new observations a twist by inserting  
some extreme values and some missing values. On top of that we will create a new  
column, that was not observed in training data.

# create sample of row indices.

samples <- lapply(1:3, function(x) {

set.seed(x)

sample(nrow(data\_new), 5, replace = FALSE)})

# create numeric values without range, -Inf and Inf.

data\_new$Sepal.Width[samples[[1]]] <- -Inf

data\_new$Petal.Width[samples[[2]]] <- Inf

# insert NA's in numeric vector.

data\_new$Petal.Length[samples[[3]]] <- NA\_real\_

# insert new column.

data\_new$junk <- "junk"

Now, we will validate the new observations by running a number of basic  
validation tests on each of the new observations. The tests are based on the  
data.tape with the recorded statistics and meta data of variabels in the  
training data.

You can get an overview over the validation tests with get\_tests\_meta\_data().

get\_tests\_meta\_data()

#> test\_name evaluate\_level evaluate\_class

#> 1: missing\_variable col all

#> 2: mismatch\_class col all

#> 3: mismatch\_levels col factor

#> 4: new\_variable col all

#> 5: outside\_range row numeric, integer

#> 6: new\_level row factor

#> 7: new\_NA row all

#> 8: new\_text row character

#> description

#> 1: variable observed in training data but missing in new data

#> 2: 'class' in new data does not match 'class' in training data

#> 3: 'levels' in new data and training data are not identical

#> 4: variable observed in new data but not in training data

#> 5: value in new data outside recorded range in training data

#> 6: new 'level' in new data compared to training data

#> 7: NA observed in new data but not in training data

#> 8: new text in new data compared to training data

To run the tests simply invoke the play() function with the recorded data.tape  
on the new data.

playback <- play(tape, data\_new)

#>

#> [PLAY]

#>

#> ... playing data.tape on new data with 50 rows with 6 columns ...

#>

#> [STOP]

What we actually have here is an object belonging to the new data.playback  
class.

class(playback)

#> [1] "data.playback" "list"

Great, now let us have a detailed look at the test results with the print()  
method.

playback

#>

#> [PLAY]

#>

#> # of rows in new data: 50

#> # of rows passing all tests: 0

#> # of rows failing one or more tests: 50

#>

#> Test results (failures):

#> > 'missing\_variable': no failures

#> > 'mismatch\_class': no failures

#> > 'mismatch\_levels': no failures

#> > 'new\_variable': junk

#> > 'outside\_range': Sepal.Width[row(s): #1, #4, #7, #23, #34, #39],

#> Petal.Width[row(s): #6, #15, #21, #32, #48]

#> > 'new\_level': no failures

#> > 'new\_NA': Petal.Length[row(s): #5, #12, #36, #39, #40]

#> > 'new\_text': no failures

#>

#> Test descriptions:

#> 'missing\_variable': variable observed in training data but missing in new data

#> 'mismatch\_class': 'class' in new data does not match 'class' in training data

#> 'mismatch\_levels': 'levels' in new data and training data are not identical

#> 'new\_variable': variable observed in new data but not in training data

#> 'outside\_range': value in new data outside recorded range in training data

#> 'new\_level': new 'level' in new data compared to training data

#> 'new\_NA': NA observed in new data but not in training data

#> 'new\_text': new text in new data compared to training data

#>

#> [STOP]

As you can see, we are in a lot of trouble here. All rows failed, because  
a new variable (junk), that did not appear in the training data, was  
suddenly observed in new data. By assumption this invalidates all rows.

Besides from that, some rows failed, because values Inf and -Inf were outside  
the recorded range in the training data for variables Sepal.Width and  
Petal.Width. Also, a handful of NA values were encountered in new data  
for Petal.Length. This is a new phenomenon compared to the training data,  
where no NA values were observed.

**Extract test results**

recorder allows you extract the results of the validation tests in a number  
of ways.

**Get failed tests as data.frame**

You might want to extract the results as a data.frame with the results of the  
(failed) tests as columns. To do this, invoke get\_failed\_tests() on  
playback:

knitr::kable(head(get\_failed\_tests(playback), 15))

| **outside\_range.Sepal.Width** | **outside\_range.Petal.Width** | **new\_NA.Petal.Length** | **new\_variable.junk** |
| --- | --- | --- | --- |
| TRUE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| TRUE | FALSE | FALSE | TRUE |
| FALSE | FALSE | TRUE | TRUE |
| FALSE | TRUE | FALSE | TRUE |
| TRUE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | FALSE | FALSE | TRUE |
| FALSE | TRUE | FALSE | TRUE |

**Get failed tests as character**

It can also be useful to get the results of the (failed) tests as a string with  
one entry per row in new data, where names of the failed tests for the given  
row are concatenated.

head(get\_failed\_tests\_string(playback))

#> [1] "outside\_range.Sepal.Width;new\_variable.junk;"

#> [2] "new\_variable.junk;"

#> [3] "new\_variable.junk;"

#> [4] "outside\_range.Sepal.Width;new\_variable.junk;"

#> [5] "new\_NA.Petal.Length;new\_variable.junk;"

#> [6] "outside\_range.Petal.Width;new\_variable.junk;"

**Get clean rows**

As a third option you can extract a logical vector, that indicates which rows,  
that passed the validation tests.

get\_clean\_rows(playback)

#> [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

#> [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

#> [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

#> [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

#> [45] FALSE FALSE FALSE FALSE FALSE FALSE

TRUE means, that a given row is clean and has passed all tests, FALSE  
on the other hand implies that a given row failed one or more tests.

In this case, all rows are invalid due to the strange column  
junk, that appears in the new data (you might think, this is a strict rule,  
but it is consistent nonetheless).

**Ignore specific test results**

It might be, that the user – for various reasons – wants to ignore one or more  
of the failed tests. You can handle this easily with recorder, whenever you  
invoke one of the functions get\_clean\_rows(), get\_failed\_tests() or  
get\_failed\_tests\_string().

**Ignore test results from specific tests**

Let us assume, that we do not care about, if there is a new column in  
the new data, that was not observed in the training data. The results of a  
specific test can be ignored with the ignore\_tests argument.

Let us try it out and ignore the results of the new\_variable validation test.

get\_clean\_rows(playback, ignore\_tests = "new\_variable")

#> [1] FALSE TRUE TRUE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE

#> [12] FALSE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE FALSE TRUE

#> [23] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE

#> [34] FALSE TRUE FALSE TRUE TRUE FALSE FALSE TRUE TRUE TRUE TRUE

#> [45] TRUE TRUE TRUE FALSE TRUE TRUE

According to this – less restrictive – selection, 35  
of the new observations are now valid.

**Ignore test results from tests of specific columns**

Maybe you – for some reason – do not care about the tests results for a specific  
column. You can ignore results from tests of a specific variable with the  
ignore\_cols argument. Let us go ahead and suppress the test results from  
tests of the Petal.Length variable.

get\_clean\_rows(playback,

ignore\_tests = "new\_variable",

ignore\_cols = "Petal.Length")

#> [1] FALSE TRUE TRUE FALSE TRUE FALSE FALSE TRUE TRUE TRUE TRUE

#> [12] TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE FALSE TRUE

#> [23] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE

#> [34] FALSE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE

#> [45] TRUE TRUE TRUE FALSE TRUE TRUE

Now, with this modification a total of 39  
of the new observations are now valid.

**Ignore test results from specific tests of specific columns**

It is also possible to ignore the test results of specific tests of specific  
columns with the ignore\_combinations argument. Let us try to ignore the  
outside\_range test, but only for the Sepal.Width variable.

knitr::kable(head(get\_failed\_tests(playback,

ignore\_tests = "new\_variable",

ignore\_cols = "Petal.Length",

ignore\_combinations = list(outside\_range = "Sepal.Width")),

15))

| **outside\_range.Petal.Width** |
| --- |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| TRUE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| FALSE |
| TRUE |

As you see – with this additional removal – the only test failures that remain  
are the ones from the outside\_range test of the Petal.Width variable.

That is it, I hope, that you will enjoy the recorder package