Data Mining with Roassal: Modelisation and Examples

Pierre Chanson

Version of 2015-08-05

Contents

Chapter 1

Introduction

This book has the goal of proving the efficiency and simplicity of Roassal concerning modelisations for data mining, associated with SciSmalltalk for the statistics calculs. It exposes most commons techniques of modelisation for data mining, each of them performed using Roassal and explained. Highly inspired from "R and Data Mining: Examples and Case Studies" (http://cran.r-project.org/doc/contrib/Zhao_R_and_data_mining.pdf), Y. Zhao, as reference. As to simplify the comparison of the two documents, we will use here the exact same datasets.

1.1 Data Mining

Book Todo:
petite introduction au data mining ?

1.2 Roassal and SciSmalltalk

In some of the subjects, the datas are going to be processed using SciSmalltalk, a statistical package from "Object-Oriented Implementation of Numerical Methods An Introduction with Smalltalk" Didier H. Besset. See https://github.com/SergeStinckwich/SciSmalltalk. Then we will modelize these datas graphically using Roassal, http://smalltalkhub.com/#!/~ObjectProfile/Roassal2. This imply another project, the R-SimpleStatistic depending on both Roassal2 and SciSmalltalk. Here it is http://smalltalkhub.com/#!/~PierreChanson/R-SimpleStat. This project reunite Roassal2 and SciSmalltalk to visualize statistics. The aim is

not only its modular characteristic, but also the simplified selection and use of SciSmalltalk classes needed for classic Data Mining analyzes.

1.3 A Principle of builders

The idea of a Rossal builder is to make the creation of a modelisation faster, easier in term of code understanding but still flexible as much as possible. Based on that principle and given a list of builders, Roassal cover all the typical modelisations.

1.4 The DataSets

We will use in our demonstrations and example the Iris Dataset. It contains 50 samples from each three classes of iris flowers Frank and Asuncion, 2010. One class is linearly separable from the other two, while the latter are not linearly separable from each other. There are five attributes in the dataset:

– sepal length in cm – sepal width in cm – petal length in cm – petal width in cm – class: Iris Setosa, Iris Versicolour, and Iris Virginica

1.5 Let's start !

From now on, it will be assumed that the reader has a Pharo image opened with a recent version of Roassal and R-SimpleStat installed in order to try the examples.

To get R-SimpleStat, Roassal2 and SciSmalltalk execute in your fresh Pharo image:

```
Gofer new
smalltalkhubUser: 'PierreChanson' project: 'R−SimpleStat';
package: 'ConfigurationOfSimpleStat';
load. ConfigurationOfSimpleStat load
```

For NeoCSV, later use to import csv files:

```
Gofer new
smalltalkhubUser: 'SvenVanCaekenberghe' project: 'Neo';
package: 'ConfigurationOfNeoCSV';
load. ConfigurationOfNeoCSV load
```

In any case we will try to make this examples as clear as possible, and will always be executable in a Pharo workspace. Moreover, notice that we will

show simple examples, the most direct way to the objective, but as Roassal is part of Pharo, a fully oriented-object programmation language and environnement, there is always a lot of other ways as complex and complete as desired to modelise our datas.

Chapter 2

Datasets, Import and Export

From inside Pharo, Roassal can take in entry different kind of datas, depending on the builder used. Simple data structures like collections or complexe dataset structures. For example, a RTBoxPlotDataSet will take a collection while the RTGrapher will take the RTBoxPlotDataSet itself. We will discuss the complex structure datasets that Roassal offers in the coming chapters.

The challenge here will most likely be to bring the datas inside Pharo before modulate them as will.

2.1 Import from CSV

We will use the package NeoCSV to work with our CSV format datas. You can load NeoCSV in your image using the catalog browser or directly:

```
Gofer new
smalltalkhubUser: 'SvenVanCaekenberghe' project: 'Neo';
package: 'ConfigurationOfNeoCSV';
load. ConfigurationOfNeoCSV load
```

Let's assume that we dispose of this "iris.csv" file in the workdirectory of our Pharo image, that looks like this:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
```

```
4.9,3.1,1.5,0.1,Iris−setosa
5.4,3.7,1.5,0.2,Iris−setosa
4.8,3.4,1.6,0.2,Iris−setosa
4.8,3.0,1.4,0.1,Iris−setosa
4.3,3.0,1.1,0.1,Iris−setosa
5.8,4.0,1.2,0.2,Iris−setosa
5.7,4.4,1.5,0.4,Iris−setosa
5.4,3.9,1.3,0.4,Iris−setosa
5.1,3.5,1.4,0.3,Iris−setosa
5.7,3.8,1.7,0.3,Iris−setosa
5.1,3.8,1.5,0.3,Iris−setosa
etc.
```

Note:
more info in the Introduction

The first step is to import the content of the file in the workspace and create a read stream:

```
('iris.csv' asFileReference contents readStream)
```

Then we read this stream with NeoCSVReader, knowing or separators are comas, the delimiter by default:

```
(NeoCSVReader on: ('iris.csv' asFileReference contents readStream) ) upToEnd.
```

At this point or datas are imported in Pharo and perfectly usable !

Note that by inspecting it we get an Array (size 150), of arrays (size 5), which represent well our iris datas.

Book Todo:
could be interesting to have a method summary that create a table resume of our datas

We can also import the csv file from an URL. In that case:

```
(NeoCSVReader on: ('myUrl' asUrl retrieveContents readStream) ) upToEnd.
```

2.2 Export to CSV

We can easily export datas to csv format using NeoCSVWritter. For example:

```
String streamContents: [ :stream |
    (NeoCSVWriter on: stream)
    nextPutAll: #( (x y z) (10 20 30) (40 50 60) (70 80 90) ) ].
```

2.3 Import via SQLite3

It exist other pharo packages, external from Roassal, to import datas into Pharo. Here is an example with the NBSQLite3 package to import .db files and extarct a perfectly usable content for Roassal.

Book Todo:
a complete example ? Reference other Sources (more complex installation, specially Linux)?

Book Todo:
Other formats like SAS, ODBC, XLS ?

Chapter 3

Common graphs and builders

In the introduction we talked about the Roassal builders to cover the most common modelisation in Data Mining. In this Chapter we will present and explain the functionning of these builders and create in examples histograms, bar plots, scatter plots, radar and pie charts etc... We will also discover and manage our datas.

In Roassal, simplicity comes with modularisation. In that case It is always easy to create a composition of graphs. Most of graphs like an histogram will need axises in adition to the visualization of datas itself. That is why in Roassal we will create this graph using a composition of two builders. Indeed the structure of a common graph will be composed of a RTGrapher, to manage the axises and then the dataSet is added to it. The RTGrapher is finally built.

The graph structure using RTGrapher:

```
graph := RTGrapher new.
dataSet := <aRTAbstractDataSet>.
graph add: dataSet.
graph build.
```

3.1 Histogram

Coming back to our iris.csv content saw in the previous chapter we will now manage this datas to create an histogram on the frequency of the epal length.

First we will process a bit our data to obtain usable datas for our histogram. As we just imported our datas from csv we will change the for first

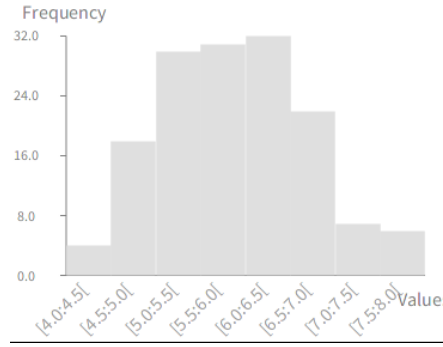

Figure 3.1: Histogram 1.

fields from String to Integer, and puts our datas in a variable:

```
iris := (NeoCSVReader on: ('iris.csv' asFileReference contents readStream) )
    addIntegerField; addIntegerField; addIntegerField; addIntegerField; addField;
     upToEnd.
```

We now need to take the first row of theses datas and ask RSimpleStat to make the histogram:

```
(iris collect: [ :i | i first ]) simpleHistogram
```

and obtain our first histogram Figure1.

Note:
As reminder, this simple histogram is made using the class RSHistogram (RSimpleStat package). This class use the SOptimizedDistribution class (SimpleStat package) depending on the DhbHistogram class from SciSmalltalk. RSimpleStat reunite Roassal2 and SciSmalltalk to visualize statistics.

Having a look to the method #simpleHistogram:, we can easily find our base builders structure and customize our own visualization easily:

```
b := RTGrapher new.

ds := RTHistogramSet new.
ds distribution: SOptimizedHistogram new.
ds points: (iris collect: [ :i | i first ]).
ds barShape color: Color orange.
```

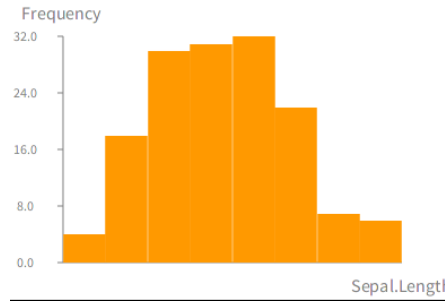

Figure 3.2: Histogram 2.

```
b add: ds.
b axisX title: 'Sepal.Length'.
b axisY title: 'Frequency'.
b build
```

We create an instance of RTGrapher and an instance of RTHistogramSet. We define the points of our data set, we define which column to take in account, we also define that we want the SOptimizedHistogram distribution, using SciSmalltalk. We can change the color, add the data set to the grapher, change the axis title and build.

3.2 Density plot

In a same way than histogram, you can generate a density plot based on a normal distribution using the SDensity class of SimpleStat or from a SequenceableCollection with #density, a method from the RSimpleStat package.

```
(iris collect: [ :i | i first ]) density
```

3.3 Pie chart

The frequency of each iris names can be obtain in an annoted array this way:

```
ds := RTDistribution new on: (iris collect: [ :i | i last ]);
      annotatedFrequencyOfDifferentObjects
```

The algorithm is quite easy, we can do another one on our own:

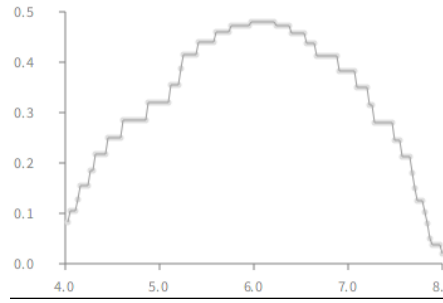

Figure 3.3: Density plot.

```
ds := iris collect: [ :i | i last ].

ds := (ds asSet collect: [ :name |
   {(ds count: [ :i | i = name]). name}
    ]) asArray.
```

Then we can get a pie chart with:

```
b := RTPieBuilder new.
b interaction popup.
b objects: ds.
b slice: #first.
b normalizer distinctColor.
b labelled: #second.
b build.
```

Book Todo:
RTPieBuilder does not work with a Set for objects, problems with labels size

3.4 Bar plot

To do a bar plot we use now the RTGrapher and add it a RTStackedDataSet:

```
b := RTGrapher new.
dt := RTStackedDataSet new.
dt points: ds.
dt y: #first.
dt barShape color: Color darkGray; width: 50.
dt barChartWithBarTitle: #second.
```

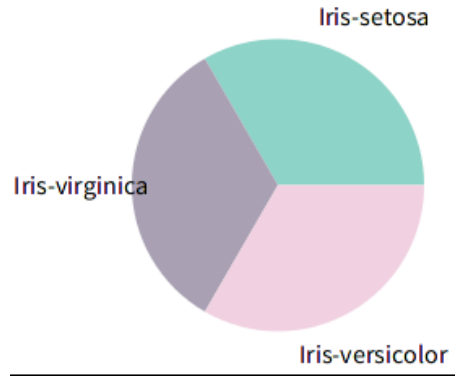

Figure 3.4: Pie chart.

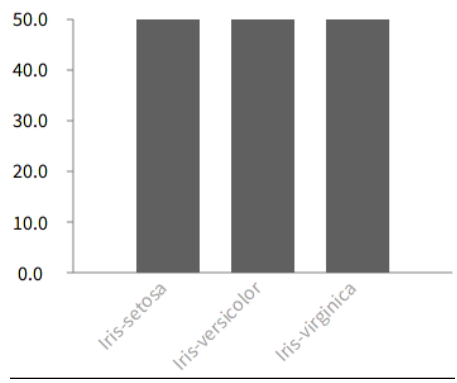

Figure 3.5: Bar plot.

```
b add: dt.
b axisX noTick; noLabel.
b build.
```

Note that in this case the histogram can also be used:

```
(iris collect: [ :i | i last ]) histogram
```

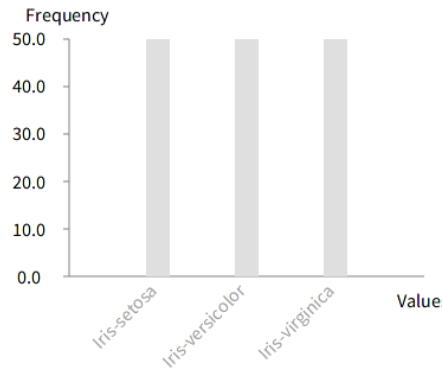

Figure 3.6: Histogram 3.

Chapter 4

More graphs and multiple variables

This chapter is about investigations on multiple variables in the same of our data set. We will then present this investigations using the proper graphs.

Note:
more about covariance can be found on a more detailed document: NumericalMethods, see Introduction, the classes implemented in this Document are going to be needed available at https://github.com/SergeStinckwich/SciSmalltalk

---

4.1 Covariance and correlation

First we will investigate covariance between columns of our datas. Let's take out the last column:

```
iris := (NeoCSVReader on: ('iris.csv' asFileReference contents readStream) )
    addIntegerField; addIntegerField; addIntegerField; addIntegerField; addField;
      upToEnd.

ds := iris collect: [ :i | i allButLast ].
```

Then we can get or covariance matrix using the DhbCovarianceAccumulator class:

```
accumulator := DhbCovarianceAccumulator new: 4.
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0.6811222	-0.0390067	1.2651911	0.5134578
-0.0390067	0.1867507	-0.319568	-0.1171947
1.2651911	-0.319568	3.0924249	1.2877449
0.5134578	-0.1171947	1.2877449	0.5785316

Figure 4.1: Covariance matrix.

```
ds do: [ :x | accumulator accumulate: x].
covarianceMatrix := accumulator covarianceMatrix.
```

Book Todo:
DhbCovarianceAccumulator missing method to have a direct covariance between two columns ? Add method to easily round all cells ?

Book Todo:
nice tables in Roassal ?

And print the result in a nice table:

```
view := RTView new.
substrings := #('Sepal.Length' 'Sepal.Width' 'Petal.Length' 'Petal.Width')
      asOrderedCollection.
table do: [:r |
   r do: [ :cell | substrings add: cell asString ]].
view addAll: (RTLabel elementsOn: substrings).
RTCellLayout new lineItemsCount:4; gapSize: 10; applyOn: view elements.
^ view
```

Book Todo:
not exactly same result than in R ?

Book Todo:
Easy way for correlation from the covariance matrix ?

About correlation, We will need to divide our tables values by respective standard deviations of their respective columns.

We can get a standard deviation:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0.9933333	-0.1086401	0.8659425	0.8125006
-0.1086401	0.9933333	-0.4177127	-0.3541671
0.8659425	-0.4177127	0.9933333	0.9563387
0.8125006	-0.3541671	0.9563387	0.9933333

Figure 4.2: Correlation matrix.

```
sd := DhbFastStatisticalMoments new.
(ds collect: [ :m | m first ]) do: [ :x | sd accumulate: x ].
sd standardDeviation.
```

So now we can have our correlation table by:

```
table := covarianceMatrix rows collectWithIndex: [:c :i | c collectWithIndex: [ :cell :j |
sd1 := DhbFastStatisticalMoments new.
sd2 := DhbFastStatisticalMoments new.
(ds collect: [ :m | m at: i ]) do: [ :x | sd1 accumulate: x ].
(ds collect: [ :m | m at: j ]) do: [ :x | sd2 accumulate: x  ].
(cell / (sd1 standardDeviation * sd2 standardDeviation)) round: 7]].
```

Book Todo:
problems of rounded when calculating covariance.

Book Todo:
having a stat summary table creation ?

4.2 Box plot

We will create in this part a graph of box plots over the sepal length, one for each of the different species of iris. We first need to divide our data to collect a collection per different species:

```
species := (iris collect: [ :i | i last ]) asSet.
speciesCollections := species collect: [ :s | iris select: [:i | i last = s]].
```

Then we collect their sepal length:

```
sepalLengthCollections := speciesCollections collect: [ :col | col collect: [ :i | i first ] ].
```

We now use the Roassal class RTBoxPlotDataSet combined with RTGrapher:

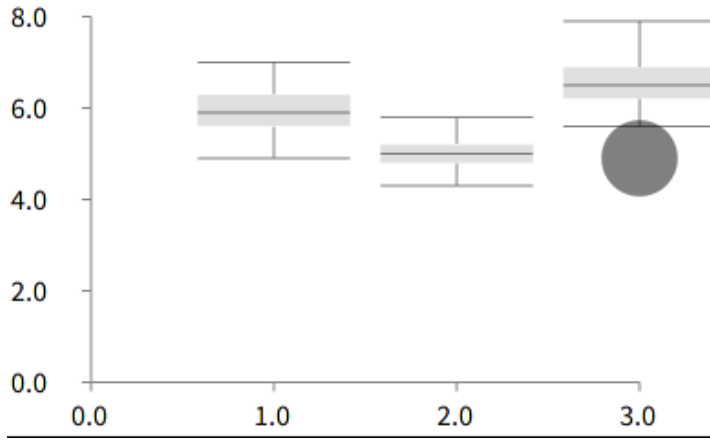

Figure 4.3: Boxplot.

```
b := RTGrapher new.
sepalLengthCollections do: [ :i | | ds |
    ds := RTBoxPlotDataSet new.
    ds points: i.
    b add: ds.].
b build.
```

Book Todo:
some fixes on RTBoxPlotDataSet, outliers aren't too big when not much box plots ? Probably it is defined on width of the box, box too big ?

4.3 scatter plot

Here we create a scatter plot with the Sepal Width and the Sepal Length. As before we collect Sepal Width and lenght for each of different species:

```
sepalWidthLengthCollections := speciesCollections collect: [ :col | col collect: [ :i | {i
    first. i second} ] ].
```

Now we can define a color identity of each species to difference them on the graph, and create our scatter plot using RTGrapher:

```
colorId := RTMultiLinearColorForIdentity new colors: {Color red. Color yellow. Color
    green }; objects: (sepalWidthLengthCollections asArray).
```

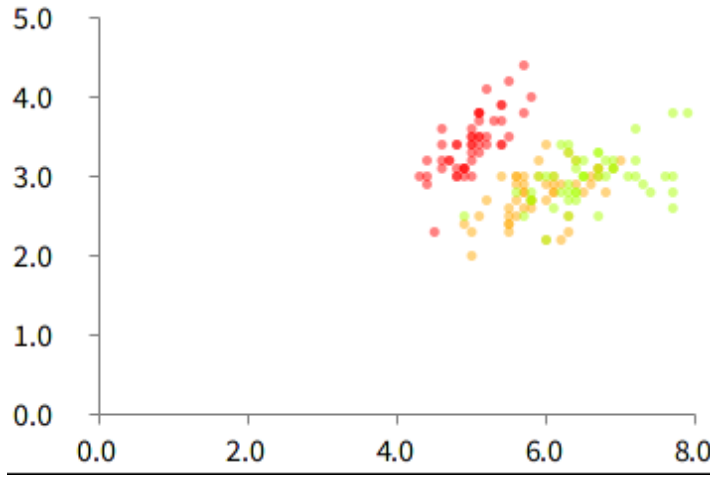

Figure 4.4: Scatterplot.

```
b := RTGrapher new.
sepalWidthLengthCollections do: [ :i |
    | ds |
    ds := RTDataSet new
    points: i;
    x: [ :m | m first ];
    y: [ :m | m second ].
    ds dotShape circle color: ((colorId rtValue: i) alpha: 0.5); size: 5.
    b add: ds.
    ].
b build.
```

Note:
To distinguish the probable overlaps between points, we can define a weaker alpha value for colors

---

Book Todo:
R use jitter to add noise

Book Todo:
a matrix of scatter plots ?

Book Todo:
heatmap ?

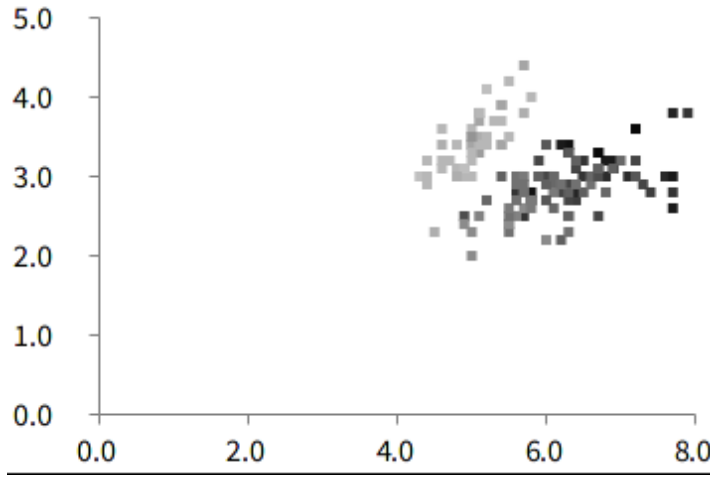

Figure 4.5: Level Plot.

4.4 Level Plot

From here we can easily produce a level plot, giving to our scatter plot a gradient color to represent the Petal Width of our points. We also have to collect the Petal Width:

```
sepalWidthLengthCollections := speciesCollections collect: [ :col | col collect: [ :i | {i
        first. i second. i fourth} ] ].

colorId := RTMultiLinearColorForIdentity new colors: {(Color veryLightGray). (Color
        black) }; objects: ((sepalWidthLengthCollections flatCollect: [:col | col collect: [:i
        | i third] ]) asArray sort).
```

And the grapher, only the shape and color change, using the Petal Width:

```
b := RTGrapher new.
sepalWidthLengthCollections do: [ :i |
    | ds |
    ds := RTDataSet new
    points: i;
    x: [ :m | m first ];
    y: [ :m | m second ].
    ds dotShape rectangle color: [ :m | (colorId rtValue: m third)] ; size: 5.
    b add: ds.
    ].
b build.
```

Book Todo:
ad a gradient scale next to the scatter maybe in a level plot builder ?

Book Todo:
The contour plot ?

Chapter 5

Exporting modelisations

This part shows how to export your modelisation in different formats. Almost any kind of visualizations made with Roassal can be exported in Pdf, Svg or Html using one of the Exporter: RTHTML5Exporter, RTSVGExporer etc.

All the previous examples can be exported: