Data Mining with Roassal: Modelisation and Examples

Pierre Chanson

Version of 2015-07-02

Contents

Chapter 1

Introduction

This book has the goal of proving the efficiency and simplicity of Roassal concerning modelisations for data mining. 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.

Moreover, for more detailed explanations on some of incoming subjects see the Document "Object-Oriented Implementation of Numerical Methods An Introduction with Smalltalk" Didier H. Besset. The classes implemented in this Document are going to be needed. The SciSmalltalk project includes these, available at https://github.com/SergeStinckwich/SciSmalltalk.

1.1 Data Mining

Book Todo:
petite introduction au data mining ?

1.2 Roassal

Book Todo:
intro Roassal, how to install it

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.

Histogtam: RTGrapher combined with a RTHistogramSet

> **Book Todo:**
> list of graphs and builders associated ?

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 installed in order to try the examples.

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. 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.

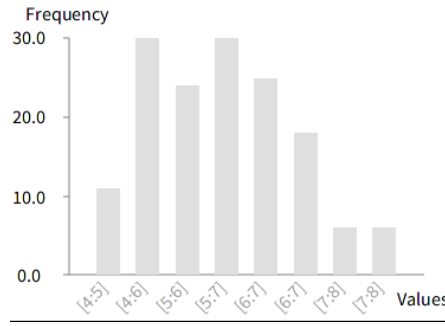

Figure 3.1: Histogram 1.

Book Todo:
RTHistogramSet can't manage a gapSize

Book Todo:
RTHistogramSet should use SciSmalltalk histograms

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 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 Roassal to make the histogram:

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

and obtain our first histogram Figure1.

Having a look to the method histogram we can customize it easily, following the structure:

```
b := RTGrapher new.
b extent: 400@200.

ds := RTHistogramSet new.
ds points: (iris collect: [ :i | i first ]).
ds barShape width: 40; color: Color darkGray.
ds barChartWithBarTitle: [ :vls |  vls second average round:1  ] rotation: 0 color:
      Color black.
```

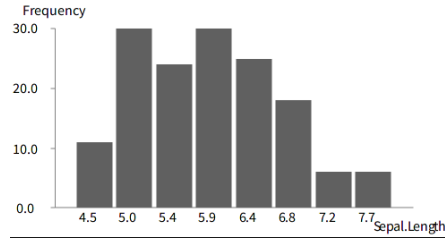

Figure 3.2: Histogram 2.

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

b build.
```

Book Todo:
RTDistribution does not work with String, merge with SciSmalltalk ? The histogram construction can be improved by knowing the width of the dataset

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, add the data set to the grapher and build it.

3.2 Density plot

3.3 Pie chart

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

Book Todo:
Distribution should be from SciSmalltalk

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

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

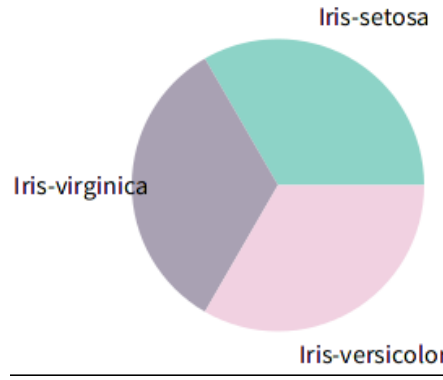

Figure 3.3: Pie chart.

```
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.
```

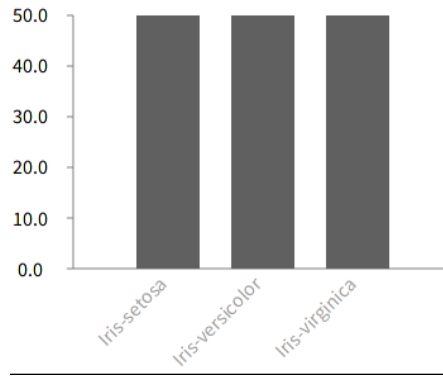

Figure 3.4: Bar plot.

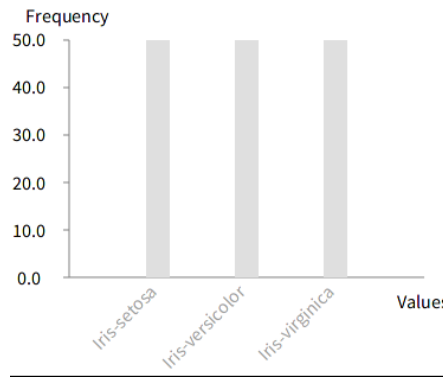

Figure 3.5: Histogram 3.

```
dt y: #first.
dt barShape color: Color darkGray; width: 50.
dt barChartWithBarTitle: #second.
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
```

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 ? Why DhbStatisticalMoments can't take 4 columns ??

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]].
```

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: