CHAPTER 15

Getting Started with RapidMiner

For someone who has never attempted any analysis using RapidMiner, this chapter would be the best place to start. In this chapter the attention will be turned away from the data science concepts and processes toward an actual tool set needed for data science. The goal for this chapter is to get rid of any trepidation that one may have about using the tool if this entire field of analytics is totally new. If perhaps someone has done some data science with RapidMiner but gotten frustrated or stuck somewhere during the process of self-learning using this powerful set of tools, then this chapter should hopefully help.

RapidMiner is an open source data science platform developed and maintained by RapidMiner Inc. The software was previously known as YALE (Yet Another Learning Environment) and was developed at the University of Dortmund in Germany (Mierswa, 2006).

RapidMiner Studio is a graphical user interface or GUI-based software where data science *workflows* can be built and deployed. Some of the advanced features are offered at a premium. In this chapter, some of the common functionalities and terminologies of the RapidMiner Studio platform will be reviewed. Even though one specific data science tool is being emphasized, the approach, process, and terms are all similar to other commercial and open source data science tools.

Firstly a brief introduction to the RapidMiner Studio GUI will be given to set the stage. The first step in any data analytics exercise is of course to bring the data to the tool, and this is what will be covered next. Once the data is imported, one may want to actually visualize the data and if necessary select subsets or transform the data. Basic visualization is covered, followed by selecting data by subsets. An overview of the fundamental data scaling and transformation tools will be provided and data sampling and missing value handling tools will be explained. Then some advanced capabilities of RapidMiner will be presented such as process design and optimization.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00015-0

15.1 USER INTERFACE AND TERMINOLOGY

It is assumed at this point that the software is already downloaded and installed on a computer.[1] Once RapidMiner is launched, the screen in Fig. 15.1 will be seen.

We start by assuming that a new process needs to be created. With that in mind, a brand new process is started with by clicking on the "Blank" option seen at the top of Fig. 15.1. Once this is done, the view changes to that shown in Fig. 15.2. Only two of the main sections: Design and Results panels will be introduced, as the others (Turbo Prep and Auto Model) are not available in the free edition.

Views: The RapidMiner GUI offers two main *views*. The Design view is where one can create and design all the data science processes and can be thought of as the canvas where all the data science programs and logic will be created. This can also be thought of as a workbench. The Results view is where all the recently executed analysis results are available. Switching back and forth between the Design and Results views several times during a session is very

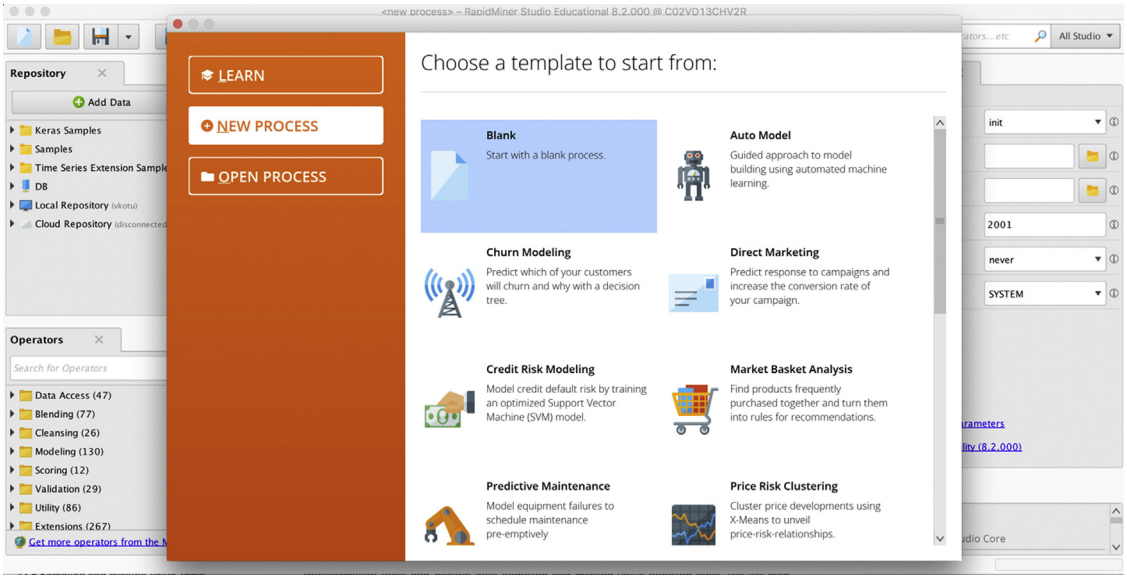


FIGURE 15.1
Launch view of RapidMiner.

[1] Download the appropriate version from http://rapidminer.com.

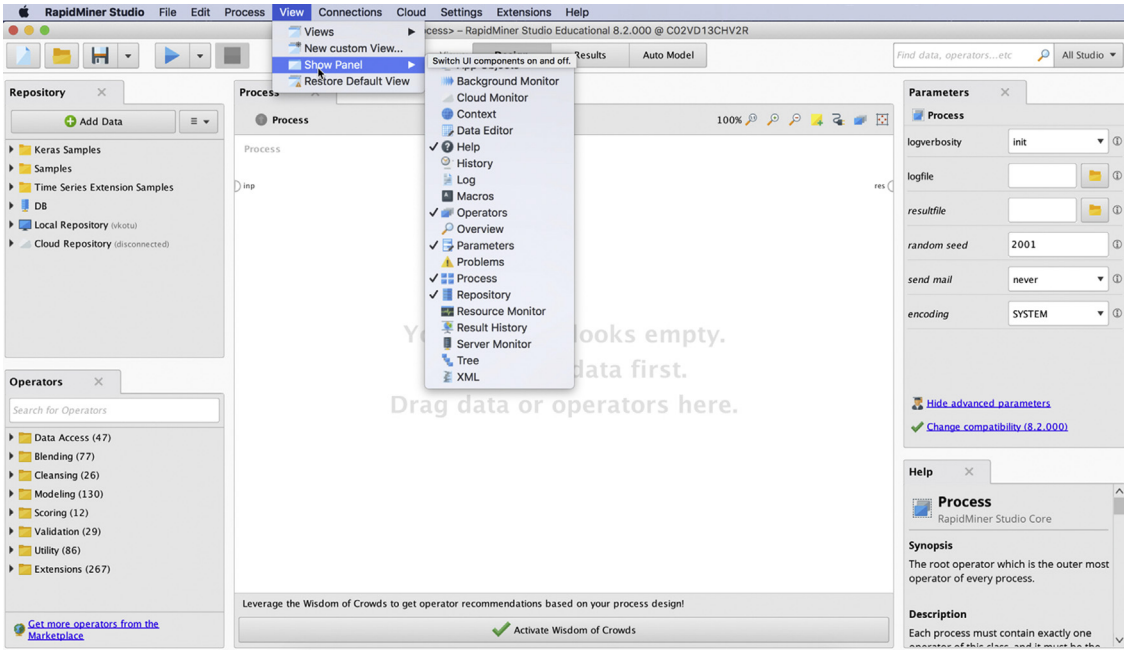


FIGURE 15.2
Activating different views inside RapidMiner.

much an expected behavior for all users. When a new process is created one starts with a blank canvas or uses a wizard-style functionality that allows starting from predefined processes for applications such as direct marketing, predictive maintenance, customer churn modeling, and sentiment analysis.

Panel: When a given view is entered, there will be several display elements available. For example, in the Design view, there is a panel for all the available operators, stored processes, help for the operators, and so on. These panels can be rearranged, resized, removed, or added to a given view. The controls for doing any of these are shown right on the top of each panel tab.

First-time users sometimes accidentally delete some of the panels. The easiest way to bring back a panel is to use the main menu item View → Show View and then select the view that was lost or reset using View → Restore Default View, see Fig. 15.2.

Terminology

There are a handful of terms that one must be comfortable with in order to develop proficiency in using RapidMiner.

Repository: A *repository* is a folder-like structure inside RapidMiner where users can organize their data, processes, and models. The repository is, thus, a central place for all the data and analysis processes. When RapidMiner is launched for the first time, an option to set up the New Local Repository will be given (Fig. 15.3). If for some reason this was not done correctly, it can always be fixed by clicking on the New Repository icon (the one with a "+Add Data" button) in the Repositories panel. When that icon is clicked on, a dialog box like the one shown in Fig. 15.3 will be given where one can specify the name of a repository under *Alias* and its location under *Root Directory*. By default, a standard location automatically selected by the software is checked, which can be unchecked to specify a different location.

Within this repository, folders and subfolders can be organized to store data, processes, results, and models. The advantage of storing datasets to be analyzed in the repository is that metadata describing those datasets are stored alongside. This metadata is propagated through the process as it is built. Metadata is basically data about the data and contains information such as the number of rows and columns, types of data within each column, missing values if any, and statistical information (mean, standard deviation, and so on).

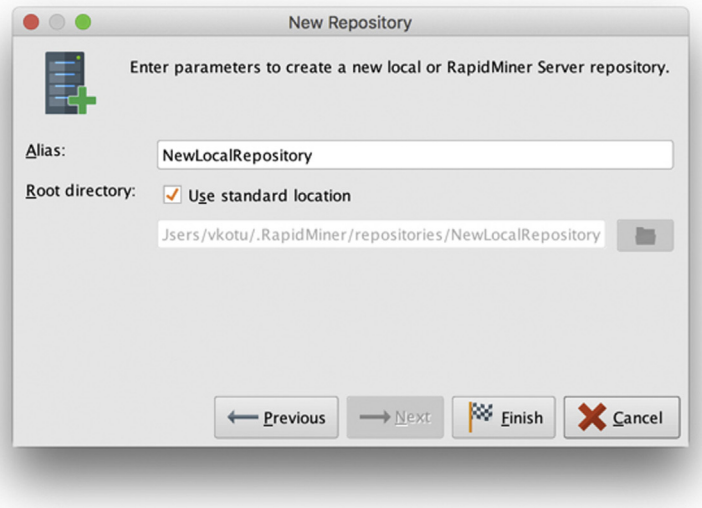


FIGURE 15.3
Setting up a repository on one's local machine.

Attribute

Example

Row No.	Play	Outlook	Temperature	Humidity	Wind
1	no	sunny	85	85	false
2	no	sunny	80	90	true
3	yes	overcast	83	78	false
4	yes	rain	70	96	false
5	yes	rain	68	80	false
6	no	rain	65	70	true
7	yes	overcast	64	65	true
8	no	sunny	72	95	false
9	yes	sunny	69	70	false
10	yes	rain	75	80	false
11	yes	sunny	75	70	true
12	yes	overcast	72	90	true
13	yes	overcast	81	75	false
14	no	rain	71	80	true

FIGURE 15.4
RapidMiner terminology: attributes and examples.

Attributes and examples: A *data set* or data table is a collection of columns and rows of data. Each column represents a type of measurement. For example, in the classic Golf data set (Fig. 15.4) that is used to explain many of the algorithms within this book, there are columns of data containing Temperature levels and Humidity levels. These are numeric data types. There is also a column that identifies if a day was windy or not or if a day was sunny, overcast, or rainy. These columns are categorical or nominal data types. In all cases, these columns represent attributes of a given day that would influence whether golf is played or not. In RapidMiner terminology, columns of data such as these are called *attributes*. Other commonly used names for attributes are variables, factors, or features. One set of values for such attributes that form a row is called an *example* in RapidMiner terminology. Other commonly used names for examples are records, samples, or instances. An entire dataset (rows of examples) is called an *example set* in RapidMiner.

Operator: An *operator* is an atomic piece of functionality (which in fact is a chunk of encapsulated code) performing a certain task. This data science task

can be: importing a data set into the RapidMiner repository, cleaning it by getting rid of spurious examples, reducing the number of attributes by using feature selection techniques, building predictive models, or scoring new datasets using models built earlier. Each task is handled by a chunk of code, which is packaged into an operator (see Fig. 15.5).

Thus, there is an operator for importing an Excel spreadsheet, an operator for replacing missing values, an operator for calculating information gain–based feature weighting, an operator for building a decision tree, and an operator for applying a model to new unseen data. Most of the time an operator requires some sort of input and delivers some sort of output (although there are some operators that do not require an input). Adding an operator to a process adds a piece of functionality to the workflow. Essentially, this amounts to inserting a chunk of code into a data science program and, thus, operators are nothing but convenient visual mechanisms that will allow RapidMiner to be a GUI-driven application rather than an application which uses programming language like R or Python.

Process: A single operator by itself cannot perform data science. All data science problem solving requires a series of calculations and logical operations. There is typically a certain flow to these problems: import data, clean and prepare data, train a model to learn the data, validate the model and rank its performance, then finally apply the model to score new and unseen data. All of these steps can be accomplished by connecting a number of different operators, each uniquely customized for a specific task as seen earlier. When such a series of operators are connected together to accomplish the desired data science, a *process* has been built which can be applied in other contexts. A process that is created visually in RapidMiner is stored by RapidMiner as a platform-independent XML code that can be exchanged between RapidMiner users (Fig. 15.6). This allows different users in different locations and on different platforms to run *your* RapidMiner process on *their* data with minimal

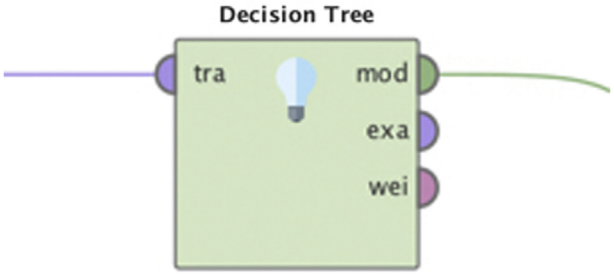


FIGURE 15.5
An operator for building a decision tree.

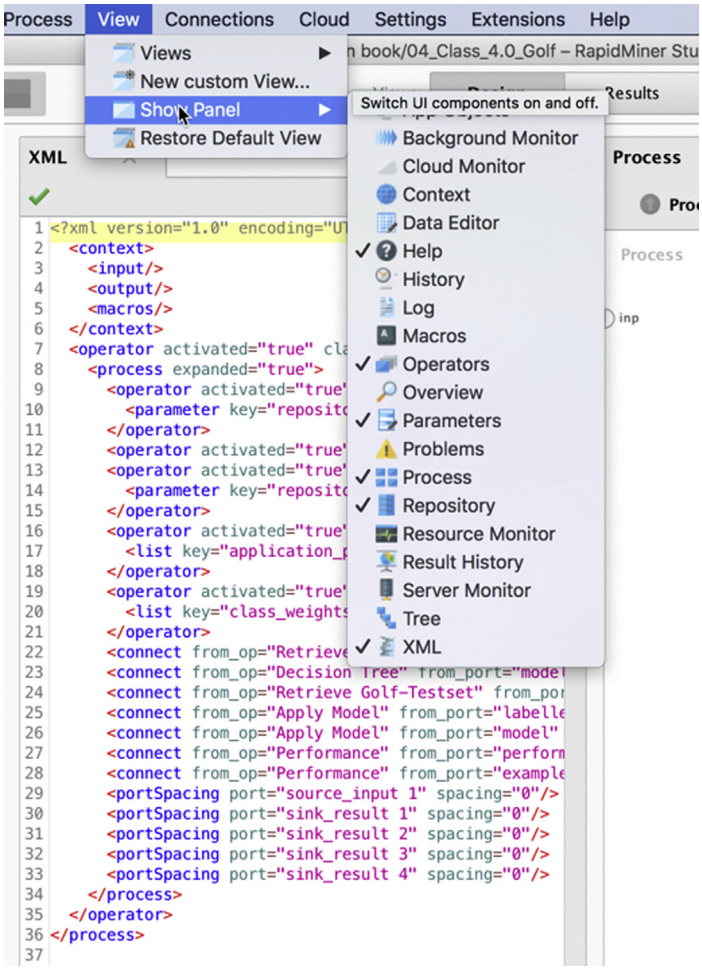


FIGURE 15.6
Every process is automatically translated to an XML document.

reconfiguration. All that needs to be done is to send the XML code of the process to a colleague across the aisle (or across the globe). They can simply copy and paste the XML code in the XML tab in the Design view and switch back to the Process tab (or view) to see the process in its visual representation and then run it to execute the defined functionality.

15.2 DATA IMPORTING AND EXPORTING TOOLS

RapidMiner offers dozens of different operators or ways to connect to data. The data can be stored in a flat file such as a comma-separated values (CSV)

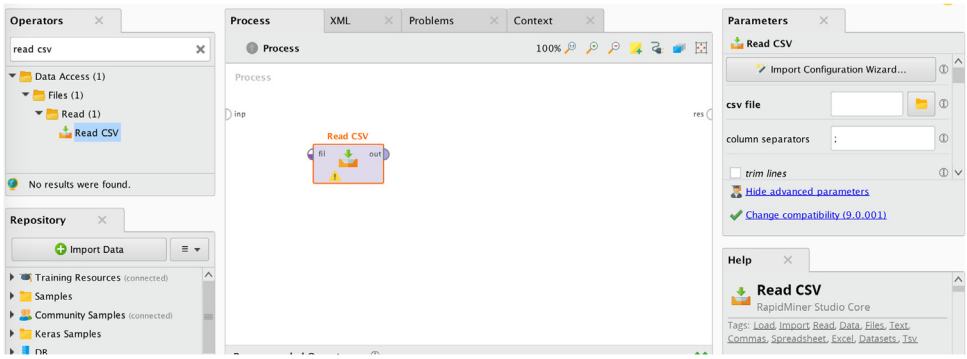


FIGURE 15.7
Steps to read in a CSV file. *CSV*, Comma-Separated Values.

file or spreadsheet, in a database such as a Microsoft SQLServer table, or it can be stored in other proprietary formats such as SAS or Stata or SPSS, etc. If the data is in a database, then at least a basic understanding of databases, database connections and queries is essential in order to use the operator properly. One could choose to simply connect to their data (which is stored in a specific location on disk) or to import the data set into the local RapidMiner repository itself so that it becomes available for any process within the repository and every time RapidMiner is opened, this data set is available for retrieval. Either way, RapidMiner offers easy-to-follow wizards that will guide through the steps. As shown in Fig. 15.7, to simply connect to data in a CSV file on disk using a *Read CSV* operator, drag and drop the operator to the main process window. Then the *Read CSV* operator would need to be configured by clicking on the Import Configuration Wizard, which will provide a sequence of steps to follow to read the data in[2]. The search box at the top of the operator window is also useful—if one knows even part of the operator name then it's easy to find out if RapidMiner provides such an operator. For example, to see if there is an operator to handle CSV files, type "CSV" in the search field and both Read and Write will show up. Clear the search by hitting the red X. Using search is a quick way to navigate to the operators if part of their name is known. Similarly try "principal" and the operator for principal component analysis can be seen, if there is uncertainty about the correct and complete operator name or where to look initially. Also, this search shows the hierarchy of where the operators exist, which helps one learn where they are.

[2] Data example shown here is available from IMF, (2012, October). World economic outlook database. International Monetary Fund. Retrieved from <http://www.imf.org/external/pubs/ft/weo/2012/02/weodata/index.aspx> Accessed 15.03.13.

On the other hand, if the data is to be imported into a local RapidMiner repository, click on the down arrow "Import Data" button in the Repositories tab (as shown in Fig. 15.7) and select Import CSV File. The same five-step data import wizard will immediately be presented. In either case, the data import wizard consists of the following steps:

1. Select the file on the disk that should be read or imported.
2. Specify how the file should be parsed and how the columns are delimited. If the data have a comma "," as the column separator in the configuration parameters, be sure to select it.
3. Annotate the attributes by indicating if the first row of the dataset contains attribute names (which is usually the case). If the dataset has first row names, then RapidMiner will automatically indicate this as Name. If the first few rows of the dataset has text or information, that would have to be indicated for each of the example rows. The available annotation choices are "Name," "Comment," and "Unit." See the example in Fig. 15.8.
4. In this step the data type of any of the imported attributes can be changed and whether each column or attribute is a regular attribute or a special

Annota...	att1	att2	att3	att4	att5	att6
Name ▼	Current ...	General...	Gross d...	Gross n...	Total in...	Country
–	3.877	21.977	0.037	30.398	26.521	Afghani...
Name	-11.372	25.835	0.032	14.509	25.886	Albania
Comment	7.489	36.458	0.337	48.947	41.428	Algeria
Unit	9.024	43.479	0.147	21.692	12.668	Angola
–	-13.109	22.43	0.002	16.194	29.303	Antigua ...
–	0.658	37.199	0.863	22.595	24.451	Argentina
–	-14.653	20.97	0.023	16.66	31.313	Armenia
–	-2.87	31.846	1.175	23.925	26.794	Australia
–	3.009	48.105	0.447	24.611	21.602	Austria
–	28.423	45.652	0.122	46.955	18.532	Azerbai...
–	3.578	27.174	0.04	34.544	30.965	Bahrain
–	1.646	11.514	0.349	29.356	24.808	Banglad...

FIGURE 15.8
Properly annotating the data.

type of attribute can be identified. By default, RapidMiner autodetects the data types in each column. However, sometimes one may need to override this and indicate if a particular column is of a different data type. The special attributes are columns that are used for identification (e.g., patient ID, employee ID, or transaction ID) only or attributes that are to be predicted. These are called "label" attributes in RapidMiner terminology.

5. In this last step, if connecting to the data on disk using Read CSV, simply hit Finish (Fig. 15.9). If the data are being imported into a RapidMiner repository (using Import CSV File), then the location in the repository for this will need to be specified.

Preview uses only first 100 rows.

Current acc	General gov	Gross dome	Gross natio	Total invest	Country
real	real	real	real	real	polyn...
attribute	attribute	attribute	attribute	attribute	label
3.877	21.977	0.037	30.398	26.521	Afghani...
-11.372	25.835	0.032	14.509	25.886	Albania
7.489	36.458	0.337	48.947	41.428	Algeria
9.024	43.479	0.147	21.692	12.668	Angola
-13.109	22.430	0.002	16.194	29.303	Antigua ...
0.658	37.199	0.863	22.595	24.451	Argentina
-14.653	20.970	0.023	16.660	31.313	Armenia
-2.870	31.846	1.175	23.925	26.794	Australia
3.009	48.105	0.447	24.611	21.602	Austria
28.423	45.652	0.122	46.955	18.532	Azerbai...
3.578	27.174	0.040	34.544	30.965	Bahrain

FIGURE 15.9
Finishing the data import.

When this process is finished, there should be either a properly connected data source on disk (for Read CSV) or a properly imported example set in their repository that can be used for any data science process. Exporting data from RapidMiner is possible in a similar way using the *Write CSV* operator.

15.3 DATA VISUALIZATION TOOLS

Once a dataset is read into RapidMiner, the next step is to explore the dataset visually using a variety of tools. Before visualization is discussed, however, it is a good idea to check the metadata of the imported data to verify if all the correct information is there. When the simple process described in Section 15.2 is run (be sure to connect the output of the read operator to the "result" connector of the process), an output is posted to the Results view of RapidMiner. The data table can be used to verify that indeed the data has been correctly imported under the *Data* tab on the left (see Fig. 15.10).

By clicking on the Statistics tab (see Fig. 15.11), one can examine the type, missing values, and basic statistics for all the imported dataset attributes. The data type of each attribute (integer, real, or binomial), and some basic statistics can also be identified. This high-level overview is a good way to ensure that a dataset has been loaded correctly and exploring the data in more detail using the visualization tools described later is possible.

Row No.	Country	Current account balance	General government revenue	Gross domestic pr...	Gross nati...	Total inves...
1	Afghanistan	3.877	21.977	0.037	30.398	26.521
2	Albania	−11.372	25.835	0.032	14.509	25.886
3	Algeria	7.489	36.458	0.337	48.947	41.428
4	Angola	9.024	43.479	0.147	21.692	12.668
5	Antigua and...	−13.109	22.430	0.002	16.194	29.303
6	Argentina	0.658	37.199	0.863	22.595	24.451
7	Armenia	−14.653	20.970	0.023	16.660	31.313
8	Australia	−2.870	31.846	1.175	23.925	26.794
9	Austria	3.009	48.105	0.447	24.611	21.602
10	Azerbaijan	28.423	45.652	0.122	46.955	18.532
11	Bahrain	3.578	27.174	0.040	34.544	30.965
12	Bangladesh	1.646	11.514	0.349	29.356	24.808

FIGURE 15.10
Results view that is shown when the data import process is successful.

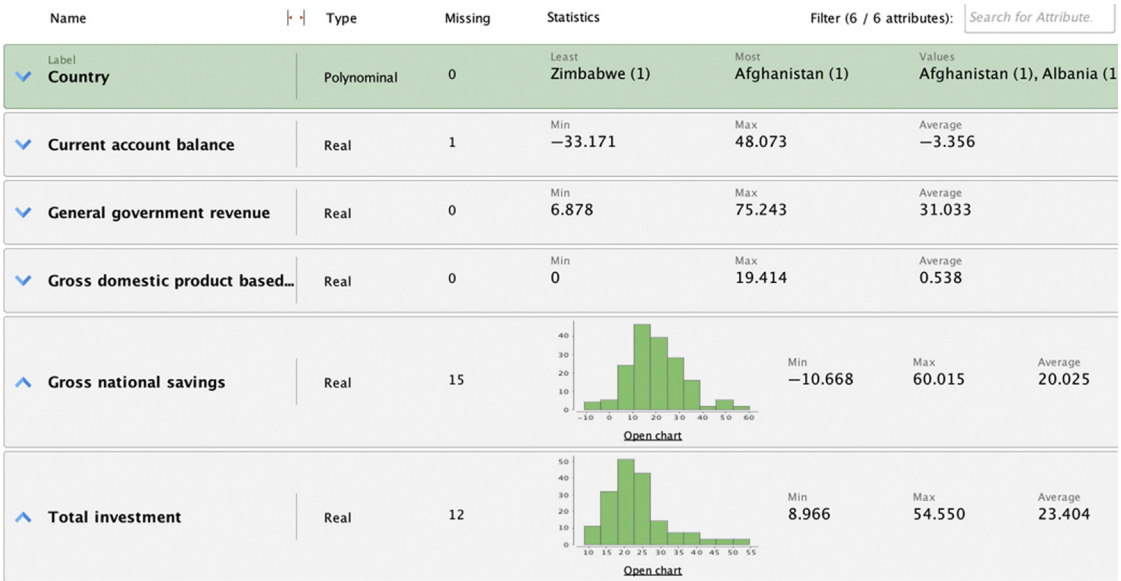


FIGURE 15.11
Metadata is visible under the Statistics tab.

There are a variety of visualization tools available for univariate (one attribute), bivariate (two attributes), and multivariate analysis. Select the Charts tab in the Results view to access any of the visualization tools or plotter. General details about visualization are available in Chapter 3, Data Exploration.

Univariate Plots

1. *Histogram:* A density estimation for numeric plots and a counter for categorical ones.
2. *Quartile (Box and Whisker):* Shows the mean value, median, standard deviation, some percentiles, and any outliers for each attribute.
3. *Series (or Line):* Usually best used for time series data.

Bivariate Plots

All 2D and 3D charts show dependencies between tuples (pairs or triads) of variables.[3]

[3] A 2D plot can also depict three dimensions, for example using color. Bubble plots can even depict four dimensions! This categorization is done somewhat loosely.

1. *Scatter:* The simplest of all 2D charts, which shows how one variable changes with respect to another. RapidMiner allows the use of color; the points can be colored to add a third dimension to the visualization.
2. *Scatter Multiple:* Allows one axis to be fixed to one variable while cycling through the other attributes.
3. *Scatter Matrix:* Allows all possible pairings between attributes to be examined. Color as usual adds a third dimension. Be careful with this plotter because as the number of attributes increases, rendering all the charts can slow down processing.
4. *Density:* Similar to a 2D scatter chart, except the background may be filled in with a color gradient corresponding to one of the attributes.
5. *SOM:* Stands for a self-organizing map. It reduces the number of dimensions to two by applying transformations. Points that are similar along many attributes will be placed close together. It is basically a clustering visualization method. There are more details in Chapter 8, Model Evaluation, on clustering. Note that SOM (and many of the parameterized reports) do not run automatically, so switch to that report, there will be a blank screen until the inputs are set and then in the case of SOM the calculate button is pushed.

Multivariate Plots

1. *Parallel:* Uses one vertical axis for each attribute, thus, there are as many vertical axes as there are attributes. Each row is displayed as a line in the chart. Local normalization is useful to understand the variance in each variable. However, a deviation plot works better for this.
2. *Deviation:* Same as parallel, but displays mean values and standard deviations.
3. *Scatter 3D:* Quite similar to the 2D scatter chart but allows a three-dimensional visualization of three attributes (four, the color of the points is included)
4. *Surface:* A surface plot is a 3D version of an area plot where the background is filled in.

These are not the only available plotters. Some additional ones are not described here such as pie, bar, ring, block charts, etc. Generating any of the plots using the GUI is pretty much self-explanatory. The only words of caution are that when a large dataset is encountered, generating some of the graphics intensive multivariate plots can be quite time consuming depending on the available RAM and processor speed.

15.4 DATA TRANSFORMATION TOOLS

Many times the raw data is in a form that is not ideal for applying standard machine learning algorithms. For example, suppose there are categorical attributes such as gender, and the requirement is to predict purchase amounts based on (among several other attributes) the gender. In this case, the categorical (or nominal) attributes need to be converted into numeric ones by a process called *dichotomization*. In this example, two new variables are introduced called Gender = Male and Gender = Female, which can take (numeric) values of 0 or 1.

In other cases, numeric data may be given but the algorithm can only handle categorical or nominal attributes. A good example, is where the label variable being numeric (such as the market price of a home in the Boston Housing example set discussed in Chapter 5 on regression) and one wants to use logistic regression to predict if the price will be higher or lower than a certain threshold. Here the need is to convert a numeric attribute into a binomial one.

In either of these cases, the underlying data types may need to be transformed into some other types. This activity is a common data preparation step. The four most common data type conversion operators are:

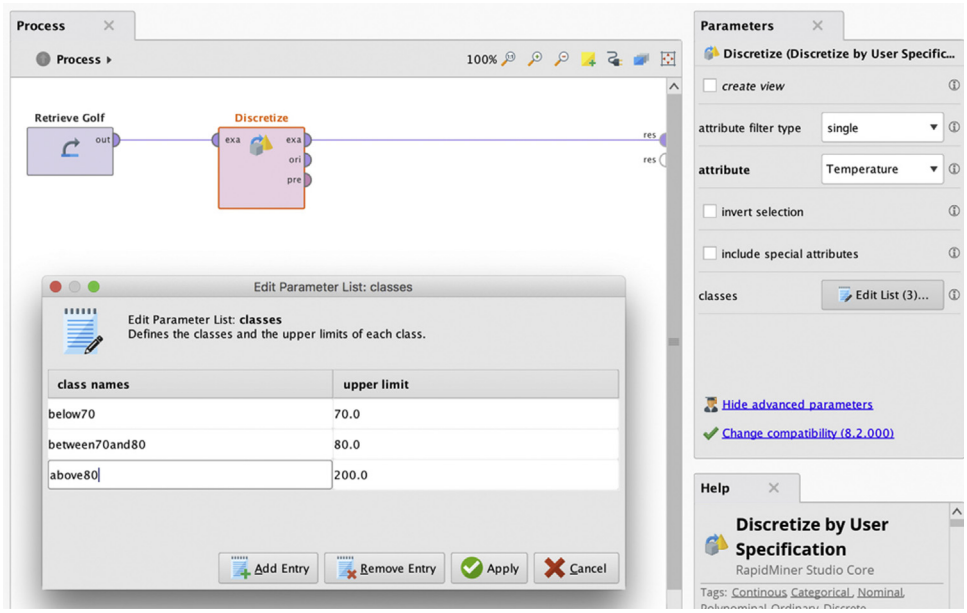
1. ***Numerical to Binominal:*** The *Numerical to Binominal* operator changes the type of numeric attributes to a binary type. Binominal attributes can have only two possible values: true or false. If the value of an attribute is between a specified minimal and maximal value, it becomes false; otherwise it is true. In the case of the market price example, the threshold market price is $30,000. Then all prices from $0 to $30,000 will be mapped to false and any price above $30,000 is mapped to true.
2. ***Nominal to Binominal:*** Here if a nominal attribute with the name "Outlook" and possible nominal values "sunny," "overcast," and "rain" is transformed, the result is a set of three binominal attributes, "Outlook = sunny," "Outlook = overcast," and "Outlook = rain" whose possible values can be true or false. Examples (or rows) of the original dataset where the Outlook attribute had values equal to sunny, will, in the transformed example set, have the value of the attribute Outlook = sunny set to true, while the value of the Outlook = overcast and Outlook = rain attributes will be false.
3. ***Nominal to Numerical:*** This works exactly like the *Nominal to Binominal* operator if one uses the Dummy coding option, except that instead of true/false values, one would see 0/1 (binary values). If unique integers option was used, each of the nominal values will get assigned a unique integer from 0 upwards. For example, if Outlook was sunny, then "sunny" gets replaced by the value 1, "rain" may get replaced by 2, and "overcast" may get replaced by 0.

4. ***Numerical to Polynominal:*** Finally, this operator simply changes the type (and internal representation) of selected attributes, that is, every new numeric value is considered to be another possible value for the polynominal attribute. In the golf example, the Temperature attribute has 12 unique values ranging from 64 to 85. Each value is considered a unique nominal value. As numeric attributes can have a huge number of different values even in a small range, converting such a numeric attribute to polynominal form will generate a huge number of possible values for the new attribute. A more sophisticated transformation method uses the discretization operator, which is discussed next.
5. ***Discretization:*** When converting numeric attributes to polynominal, it is best to specify how to set up the discretization to avoid the previously mentioned problem of generating a huge number of possible values—each numeric value should not appear as a unique nominal one but would rather be binned into some intervals. The Temperature in the golf example can be discretized by several methods: one can discretize using equal-sized bins with the *Discretize by Binning* operator. If two bins (default) are selected there will be two equal ranges: [below 74.5] and [above 74.5], where 74.5 is the average value of 64 and 85. Based on the actual Temperature value, the example will be assigned into one of the two bins. The number of rows falling into each bin (*Discretize by Size* operator) can instead be specified rather than equal bin ranges. One could also discretize by bins of equal number of occurrences by choosing to *Discretize by Frequency*, for example. Probably the most useful option is to *Discretize by User Specification*. Here explicit ranges can be provided for breaking down a continuous numeric attribute into several distinct categories or nominal values using the table shown in Fig. 15.12A. The output of the operator performing that discretization is shown in Fig. 15.12B.

Sometimes the structure of an example set may need to be transformed or rotated about one of the attributes, a process commonly known as pivoting or creating pivot tables. Here is a simple example of why this operation would be needed. The table consists of three attributes: a customer ID, a product ID, and a numeric measure called Consumer Price Index (CPI) (see Fig. 15.13A). It can be seen that this simple example has 10 unique customers and 2 unique product IDs. What should preferably be done is to rearrange the data set so that one gets two columns corresponding to the two product IDs and aggregate[4] or group the CPI data by customer IDs. This is

[4] CAUTION: The Pivot operator does not aggregate! If the source dataset contains combinations of product ID and customer ID occurring multiple times, one would have to aggregate before applying the Pivot operator in order to produce a dataset containing only unique combinations first.

(A)



(B)

Row No.	Play	temp	Temperature	Outlook	Humidity	Wind
1	no	85	above80	sunny	85	false
2	no	80	between70and80	sunny	90	true
3	yes	83	above80	overcast	78	false
4	yes	70	below70	rain	96	false
5	yes	68	below70	rain	80	false
6	no	65	below70	rain	70	true
7	yes	64	below70	overcast	65	true
8	no	72	between70and80	sunny	95	false
9	yes	69	below70	sunny	70	false
10	yes	75	between70and80	rain	80	false

FIGURE 15.12

(A) Discretize operator. (B) The output of the operation.

because the data needs to be analyzed on the customer level, which means that each row has to represent one customer and all customer features have to be encoded as attribute values. Note that there are two missing values (Fig. 15.13A, in the 'customer id' column indicated by the thin rectangles. These are missing entries for customer ids 6 and 8. However the result of the pivot operation will have 10 x 2 = 20 entries because there are 10 customers (c1:c10) and 2 products (v1:v2).

(A)

Row No.	customer id	product id	CPI
1	c1	v1	0.970
2	c2	v1	0.860
3	c3	v1	?
4	c4	v1	0.530
5	c5	v1	0.330
6	c7	v1	0.190
7	c9	v1	0.650
8	c10	v1	0.440
9	c1	v2	0.790
10	c2	v2	0.600
11	c3	v2	0.730

(B)

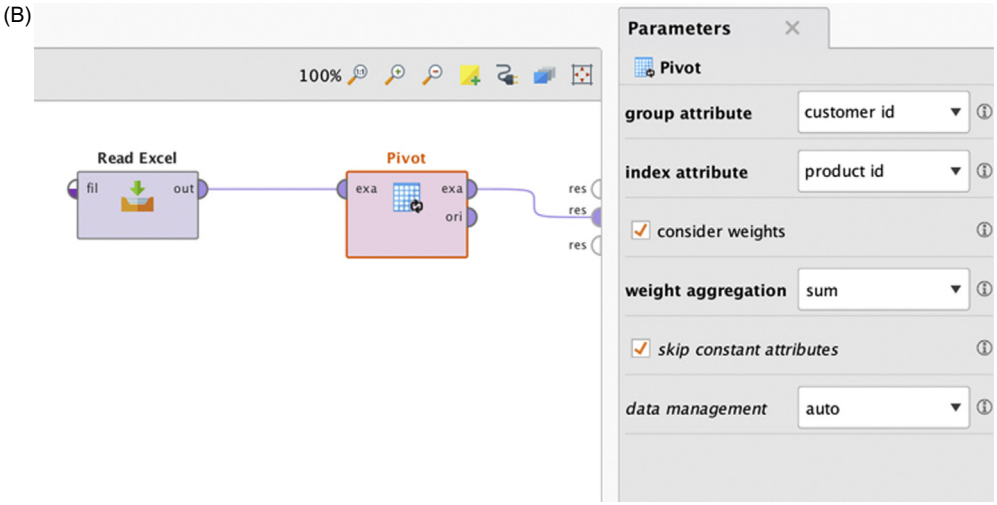


FIGURE 15.13

(A) A simple dataset to explain the pivot operation using RapidMiner. (B) Configuring the Pivot operator. (C) Results of the pivot operation.

This is accomplished simply with the *Pivot* operator. Select "customer id" as the group attribute and "product id" as the index attribute as shown in Fig. 15.13B. If familiar with Microsoft Excel's pivot tables, the group attribute parameter is similar to "row label" and the index attribute is akin to "column

(C)

Row No.	customer id	CPI_v1	CPI_v2
1	c1	0.970	0.790
2	c10	0.440	0.420
3	c2	0.860	0.600
4	c3	?	0.730
5	c4	0.530	0.660
6	c5	0.330	0.780
7	c6	?	?
8	c7	0.190	?
9	c8	?	0.040
10	c9	0.650	0.910

FIGURE 15.13
(Continued).

label." The result of the pivot operation is shown in Fig. 15.13C. Observe that the column labels are prefixed by the name of the column label attribute e.g. CPI_v1. Missing entries from the original table now become missing values indicated by "?".

A converse of the *Pivot* operator is the *De-pivot* operator, which reverses the process described and may sometimes also be required during the data preparation steps. In general a *De-pivot* operator converts a pivot table into a relational structure.

In addition to these operators, one may also need to use the *Append* operator to add examples to an existing dataset. Appending an example set with new rows (examples) works just as the name sounds— the new rows end up getting attached to the end of the example set. One has to make sure that the examples match the attributes exactly with the main dataset. Also useful is the classic *Join* operator, which combines two example sets with the same observations units but different attributes. The *Join* operator offers the traditional inner, outer, and left and right join options. An explanation for joins is available in any of the books that deal with SQL programming as well as the RapidMiner help, which also provides example processes. They will not be repeated here.

Some of other common operators used in the various chapters of the book (and are explained there in context) are: Rename attributes, Select attributes, Filter examples, Add attributes, Attribute weighting, etc.

15.5 SAMPLING AND MISSING VALUE TOOLS

Data sampling might seem out of place in today's big data–charged environments. Why bother to sample when one can collect and analyze all the data they can? Sampling is a perhaps a vestige of the statistical era when data was costly to acquire and computational effort was costlier still. However, there are many situations today where "targeted" sampling is of use. A typical scenario is when building models on data where some class representations are extremely low. Consider the case of fraud prediction. Depending on the industry, fraudulent examples range from less than 1% of all the data collected to about 2%–3%. When classification models are built using such data, the models tend to be biased and would not be able to detect fraud in the majority of the cases with new unseen data, literally because they have not been trained on enough fraudulent samples!

Such situations call for balancing datasets where the training data needs to be sampled and the proportion of the minority class needs to be increased so that the models can be trained better. The plot in Fig. 15.14 shows an example of imbalanced data: the negative class indicated by a circle is disproportionately higher than the positive class indicated by a cross.

This can be explored using a simple example. The dataset shown in the process in Fig. 15.15 is available in RapidMiner's Samples repository and is called "Weighting." This is a balanced dataset consisting of about 500 examples with the label variable consisting of roughly 50% positive and 50% negative classes. Thus, it is a balanced dataset. When one trains a decision tree to classify this data, they would get an overall accuracy of 84%. The main thing to note here is that the decision tree recall on both the classes is roughly the same: ~86% as seen in Fig. 15.15.

A sub-process called *Unbalance* is now introduced, which will resample the original data to introduce a skew: the resulting dataset has more positive class examples than negative class examples. Specifically, the dataset is now with 92% belonging to the positive class (as a result the model inflates the predicted class recall to 99.2%) and 8% belonging to negative class (as a result the model underestimates the predicted class recall to 21.7%). The process and the results are shown in Fig. 15.16. So how do we address this data imbalance?

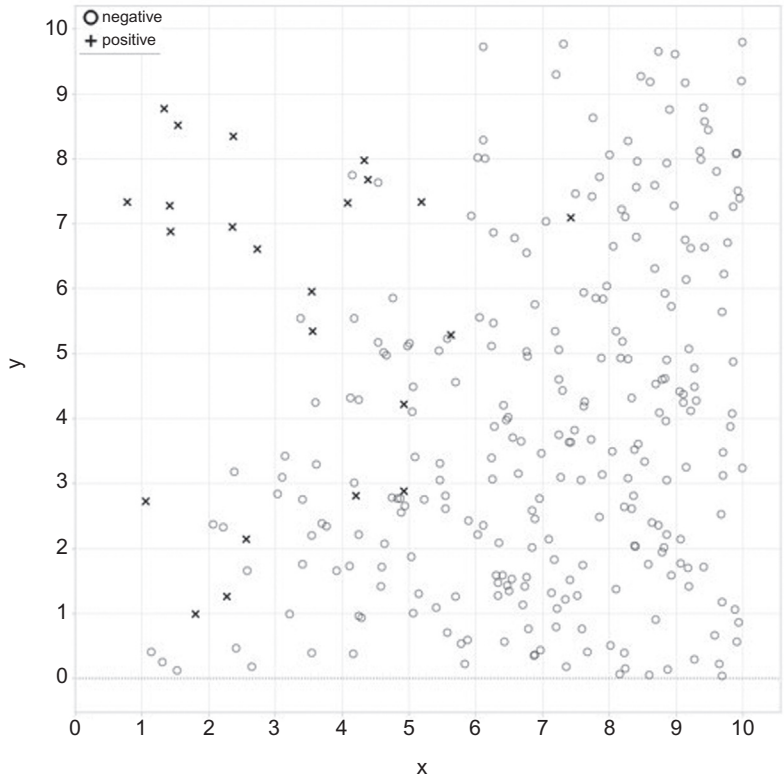
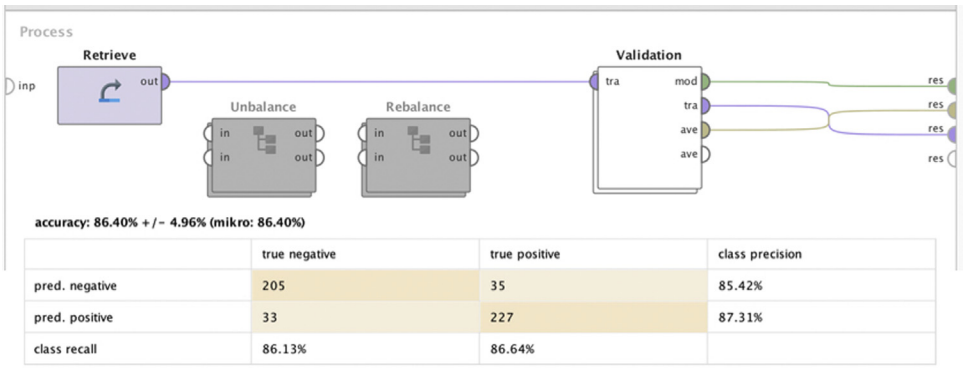


FIGURE 15.14
Snapshot of an imbalanced dataset.



	true negative	true positive	class precision
pred. negative	205	35	85.42%
pred. positive	33	227	87.31%
class recall	86.13%	86.64%	

FIGURE 15.15
Performance of decision trees on well-balanced data.

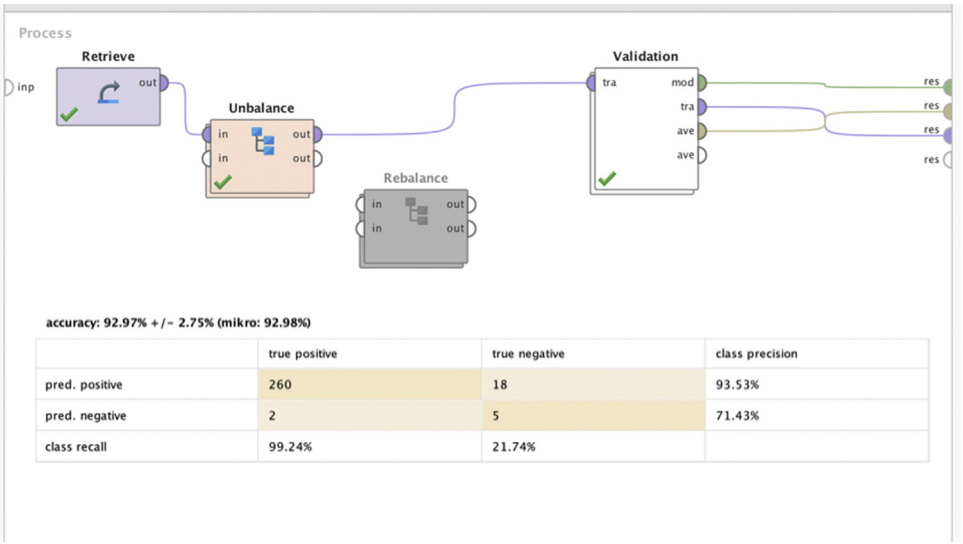
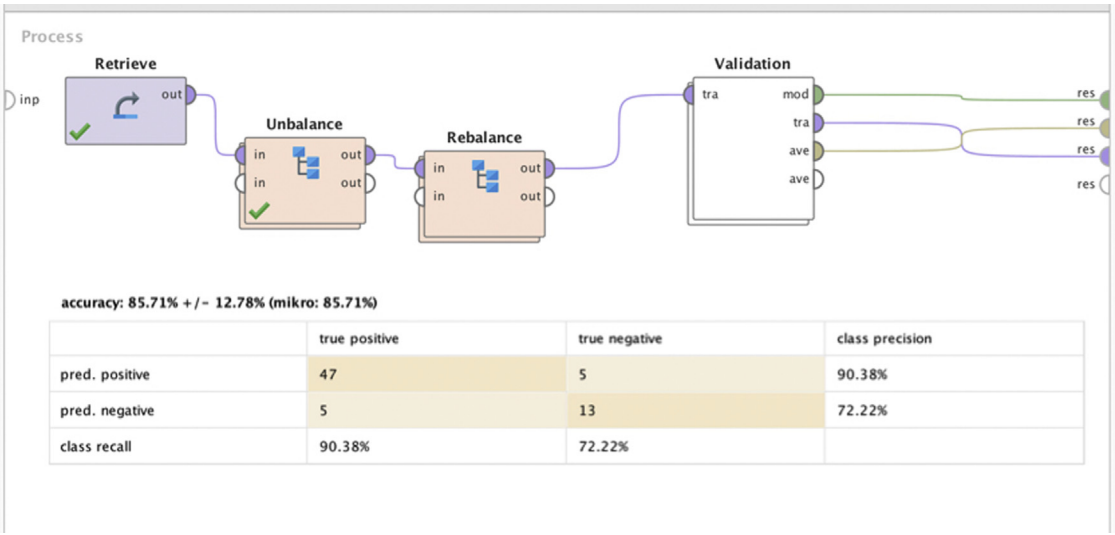


FIGURE 15.16
Unbalanced data and the resulting accuracy.

There are several ways to fix this situation. The most commonly used method is to resample the data to restore the balance. This involves undersampling the more frequent class—in this case, the positive class—and oversampling the less frequent negative class. The *rebalance* sub-process achieves this in the final RapidMiner process. As seen in Fig. 15.17, the overall accuracy is now back to the level of the original balanced data. The decision tree also looks a little bit similar to the original (not visible in the figures shown, but the reader can verify with the completed processes loaded into RapidMiner), whereas, for the unbalanced dataset it was reduced to a stub. An additional check to ensure that accuracy is not compromised by unbalanced data is to replace the accuracy by what is called *balanced accuracy*. It is defined as the arithmetic mean of the class recall accuracies, which represent the accuracy obtained on positive and negative examples, respectively. If the decision tree performs equally well on either class, this term reduces to the standard accuracy (i.e., the number of correct predictions divided by the total number of predictions).

There are several built-in RapidMiner processes to perform sampling: sample, sample (bootstrapping), sample (stratified), sample (model-based), and sample (Kennard-stone). Specific details about these techniques are well described in the software help. Only the bootstrapping method will be remarked on here because it is a common sampling technique. Bootstrapping works by sampling repeatedly within a base dataset with replacement. So, when this operator is used to generate new samples, one



accuracy: 85.71% +/- 12.78% (mikro: 85.71%)

	true positive	true negative	class precision
pred. positive	47	5	90.38%
pred. negative	5	13	72.22%
class recall	90.38%	72.22%	

FIGURE 15.17
Rebalanced data and resulting improvement in class recall.

may see repeated or nonunique examples. There is the option of specifying an absolute sample size or a relative sample size and RapidMiner will randomly pick examples from the base data set with replacement to build a new bootstrapped example set.

This section will be closed with a brief description of missing value handling options available in RapidMiner. The basic operator is called *Replace Missing Values*. This operator provides several alternative ways to replace missing values: minimum, maximum, average, zero, none, and a user-specified value. There is no median value option. Basically, all missing values in a given column (attribute) are replaced by whatever option is chosen. A better way to treat missing values is to use the *Impute Missing Values* operator. This operator changes the attribute with missing values to a label or target variable, and trains models to determine the relationship between this label variable and other attributes so that it may then be predicted.

15.6 OPTIMIZATION TOOLS[5]

Recall that in Chapter 4: Classification, on decision trees, there was an opportunity to specify parameters to build a decision tree for the credit risk

[5] Readers may skip this section if completely new to RapidMiner and return to it after developing some familiarity with the tool and data science in general.

example (Section 4.1.2, step 3) but default values were simply used. Similar situations arose when building a support vector machine model (Section 4.6.3) or logistic regression model (Section 5.2.3), where the default model parameter values were simply chosen. When a model evaluation was run, the performance of the model is usually an indicator as to whether the right parameter combinations were chosen for the model.[6] But what if one is not happy with the model accuracy (or its *r*-squared value)? Can it be improved? How?

RapidMiner provides several unique operators that will allow one to discover and choose the best combination of parameters for pretty much all of the available operators that need parameter specifications. The fundamental principle on which this works is the concept of a nested operator. A nested operator was first encountered in Section 4.1.2, step 2—the *Split Validation* operator. Another nested operator was also described in Section 14.5 in the discussion on wrapper-style feature selection methods. The basic idea is to iteratively change the parameters for a learner until some stated performance criteria are met. The *Optimize* operator performs two tasks: it determines what values to set for the selected parameters for each iteration, and when to stop the iterations. RapidMiner provides three basic methods to set parameter values: grid search, greedy search, and an evolutionary search (also known as genetic) method. Each method will not be described in detail, but a high-level comparison between them will be done and when each approach would be applicable will be mentioned.

To demonstrate the working of an optimization process, consider a simple model: a polynomial function (Fig. 15.18). Specifically, for a function $y = f(x) = x^6 + x^3 - 7x^2 - 3x + 1$ one wishes to find the minimum value of y within a given domain of x. This is of course the simplest form of optimization—to select an interval of values for x where y is minimum. As seen in the functional plot, for x in $[-1.5, 2]$, there are two minima: a local minimum of $y = -4.33$, $x = -1.3$ and a global minimum of $y = -7.96$, $x = 1.18$. It will be demonstrated how to use RapidMiner to search for these minima using the *Optimize* operators. As mentioned before, the optimization happens in a nested operator, what is placed inside the optimizer will be discussed first before discussing the optimizer itself.

The nested process itself, also called the inner process, is quite simple as seen in Fig. 15.19A: *Generate Data* randomly generates values for x between an upper bound and a lower bound (see Fig. 15.19B).

[6] Normally one can't judge from just one performance estimate whether the right parameters were chosen. Multiple performance values and their dependency on the parameter values would need to be examined to infer that the right/optimal parameter values were chosen.

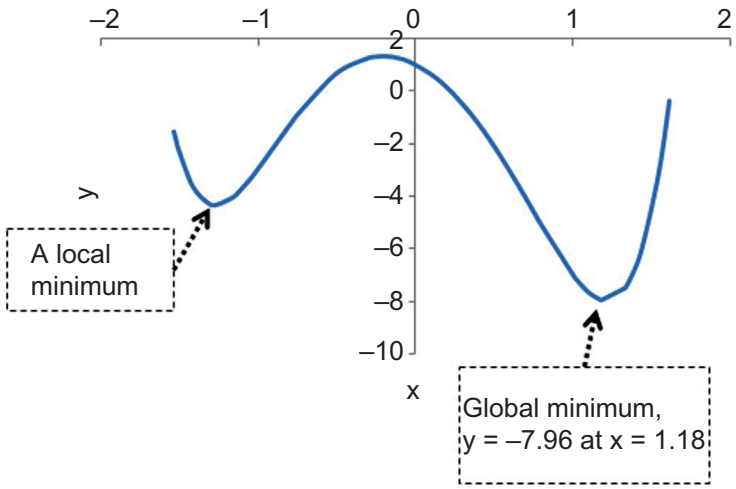


FIGURE 15.18
A simple polynomial function to demonstrate optimization.

Generate Attributes will calculate y for each value of x in this interval. *Performance (Extract Performance)* will store the minimum value of y within each interval. This operator has to be configured as shown on the right of Fig. 15.19A in order to ensure that the correct performance is optimized. In this case, y is selected as the attribute that has to be minimized. The *Rename*, *Select Attributes*, and *Log* operators are plugged in to keep the process focused on only two variables and to track the progress of optimization.

This nested process can be inserted into any of the available *Optimize Parameters* operators. It will be described how one can do this with the *Optimize Parameters (Grid)* operator first. In this exercise, the interval [*lower bound, upper bound*] is basically being optimized so that the objective of minimizing the function $y = f(x)$ can be achieved. As seen in the function plot, the entire domain of x has to be traversed in small enough interval sizes so that the exact point at which y hits a global minimum can be found.

The grid search optimizer simply moves this interval window across the entire domain and stops the iterations after all the intervals are explored (Fig. 15.20). Clearly it is an exhaustive but inefficient search method. To set this process up, simply insert the inner process inside the outer *Optimize Parameters (Grid)* operator and select the *attributes upper bound* and *attributes lower bound* parameters from the *Generate Data* operator. To do this, click on the *Edit Parameter Settings* option for the optimizer, select *Generate Data* under the *Operators* tab of the dialog box, and further select *attributes_upper_bound* and *attributes_lower_bound* under the *Parameters* tab (Fig. 15.21).

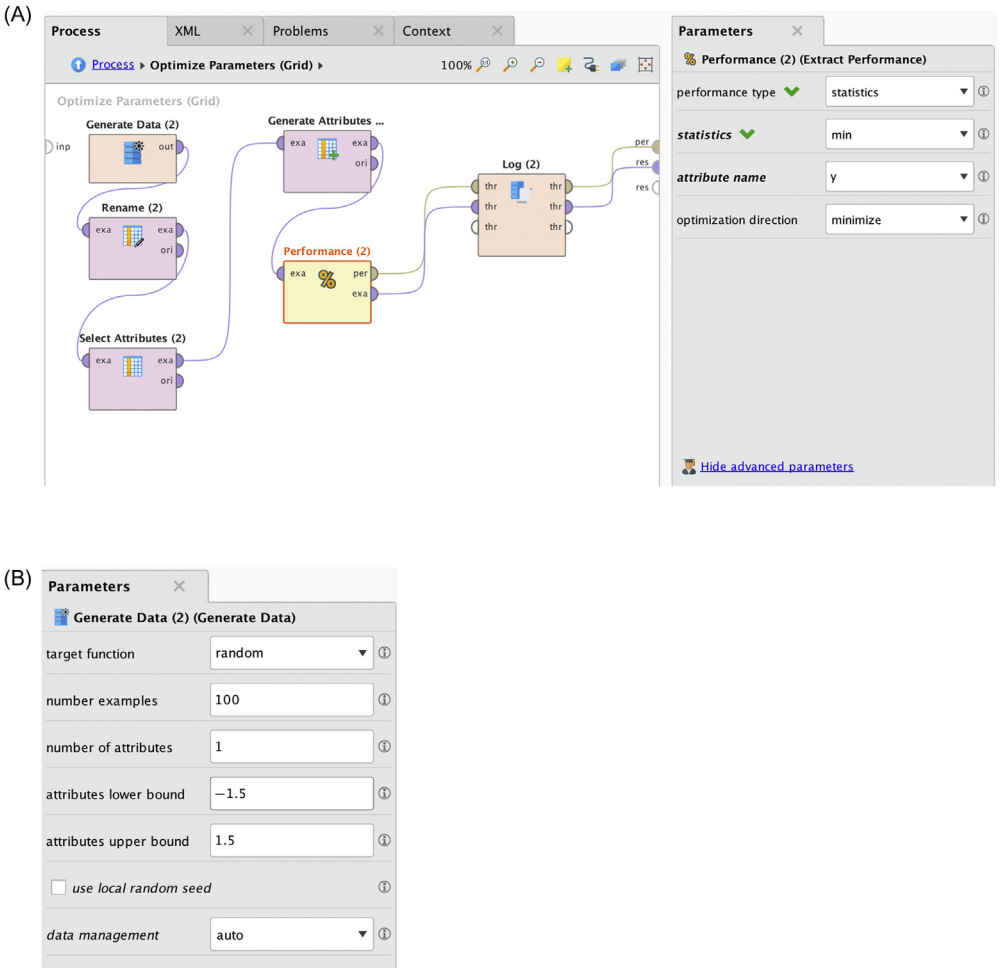


FIGURE 15.19

(A) The inner process that is nested inside an optimization loop. (B) Configuration of the generated data.

One would need to provide ranges for the grid search for each of these parameters. In this case, the lower bound is set to go from -1.5 to -1 and the upper bound to go from 0 to 1.5 in steps of 10. So the first interval (or window) will be $x = [-1.5, 0]$, the second one will be $[-1.45, 0]$ and so on until the last window, which will be $[-1, 1.5]$ for a total of 121 iterations. The *Optimize Performance (Grid)* search will evaluate y for each of these windows and store the minimum y in each iteration. The iterations will only stop after all 121 intervals are evaluated, but the final output will indicate the window that resulted in the smallest minimum y. The plot in Fig. 15.22 shows the progress of the iterations. Each point in the chart

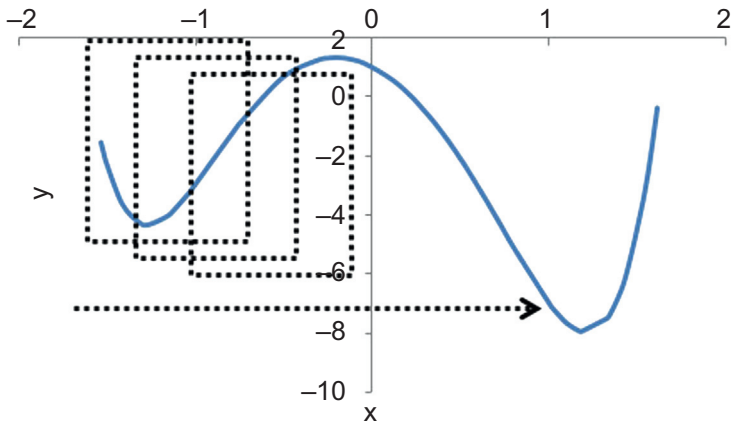


FIGURE 15.20
Searching for an optimum within a fixed window that slides across.

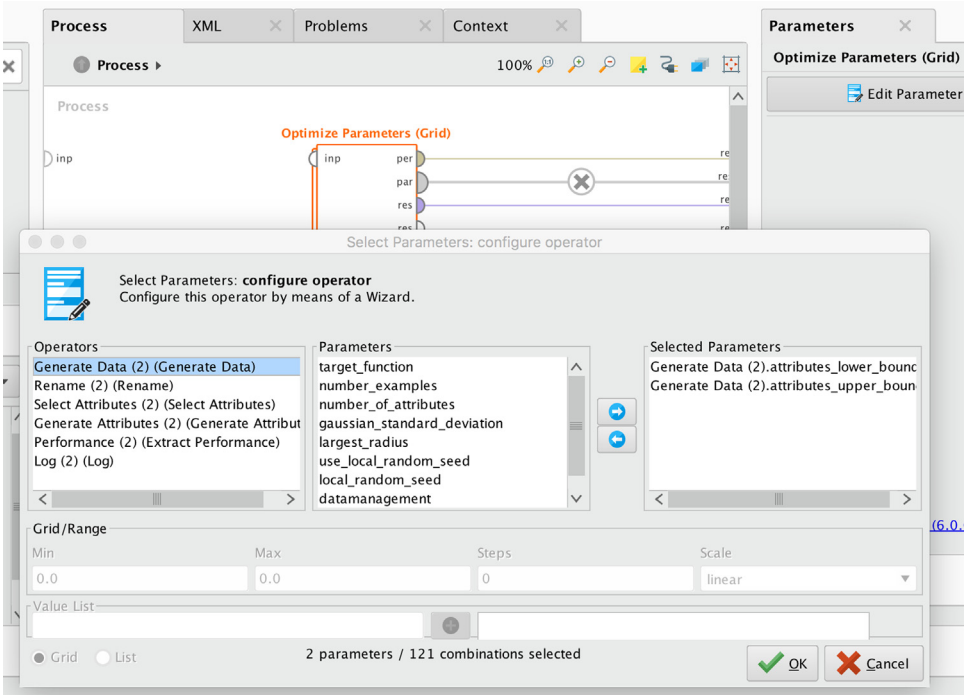


FIGURE 15.21
Configuring the grid search optimizer.

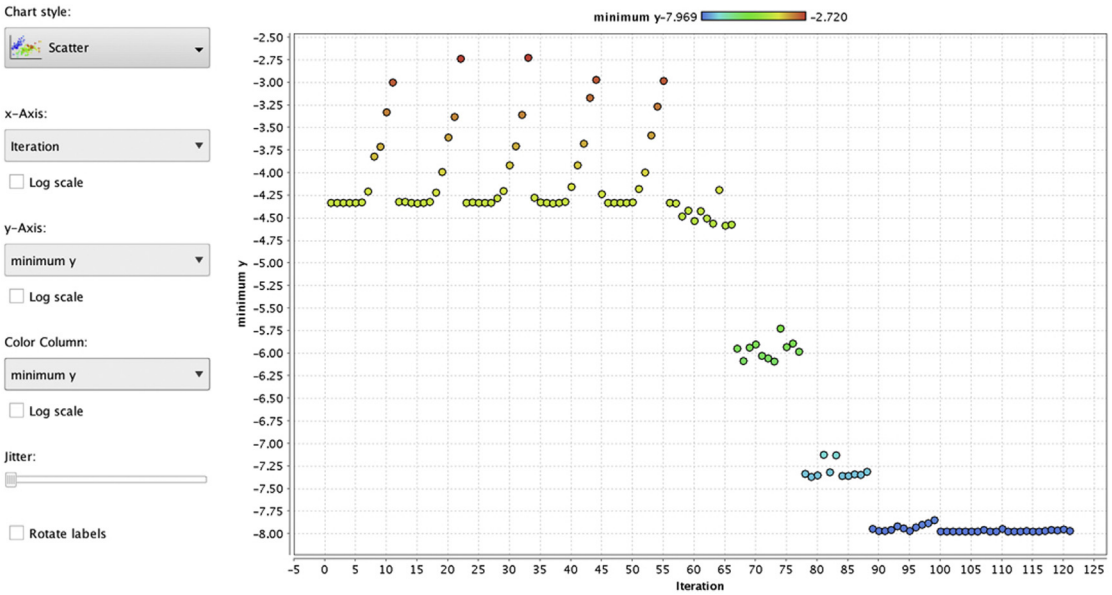


FIGURE 15.22
Progression of the grid search optimization.

corresponds to the lowest value of y evaluated by the expression within a given interval. The local minimum of $y = -4.33$ @ $x = -1.3$ is found at the very first iteration. This corresponds to the window $[-1.5, 0]$. If the grid had not spanned the entire domain $[-1.5, 1.5]$, the optimizer would have reported the local minimum as the best performance. This is one of the main disadvantages of a grid search method.

The other disadvantage is the number of redundant iterations. Looking at the plot above, one can see that the global minimum was reached by about the 90th iteration. In fact, for iteration 90, $y_{\text{minimum}} = -7.962$, whereas, the final reported lowest y_{minimum} was -7.969 (iteration 113), which is only about 0.09% better. Depending on the tolerances, the computations could have been terminated earlier. But a grid search does not allow early terminations and there end up running nearly 30 extra iterations. Clearly as the number of optimization parameters increase, this would become a significant cost.

The *Optimize Parameters (Quadratic)* operator is next applied to the inner process. Quadratic search is based on a *greedy* search methodology. A greedy methodology is an optimization algorithm that makes a locally optimal decision at each step (Ahuja, 2000; Bahmani, 2013). While the decision may be locally optimal at the current step, it may not necessarily be the best for all future steps. k-Nearest neighbor is one good example of a greedy algorithm.

In theory, greedy algorithms will only yield local optima, but in special cases, they can also find globally optimal solutions. Greedy algorithms are best suited to find approximate solutions to difficult problems. This is because they are less computationally intense and tend to operate over a large dataset quickly. Greedy algorithms are by nature typically biased toward coverage of a large number of cases or a quick payback in the objective function.

In this case, the performance of the quadratic optimizer is marginally worse than a grid search requiring about 100 shots to hit the global minimum (compared to 90 for a grid), as seen in Fig. 15.23. It also seems to suffer from some of the same problems encountered in the grid search.

Finally the last available option: Optimize Parameters (Evolutionary) will be employed. Evolutionary (or genetic) algorithms are often more appropriate than a grid search or a greedy search and lead to better results, This, is because they cover a wider variety of the search space through mutation and can iterate onto good minima through cross-over of successful models based on the success criteria. As seen in the progress of iterations in Fig. 15.24, the global optimum was hit without getting stuck initially at a

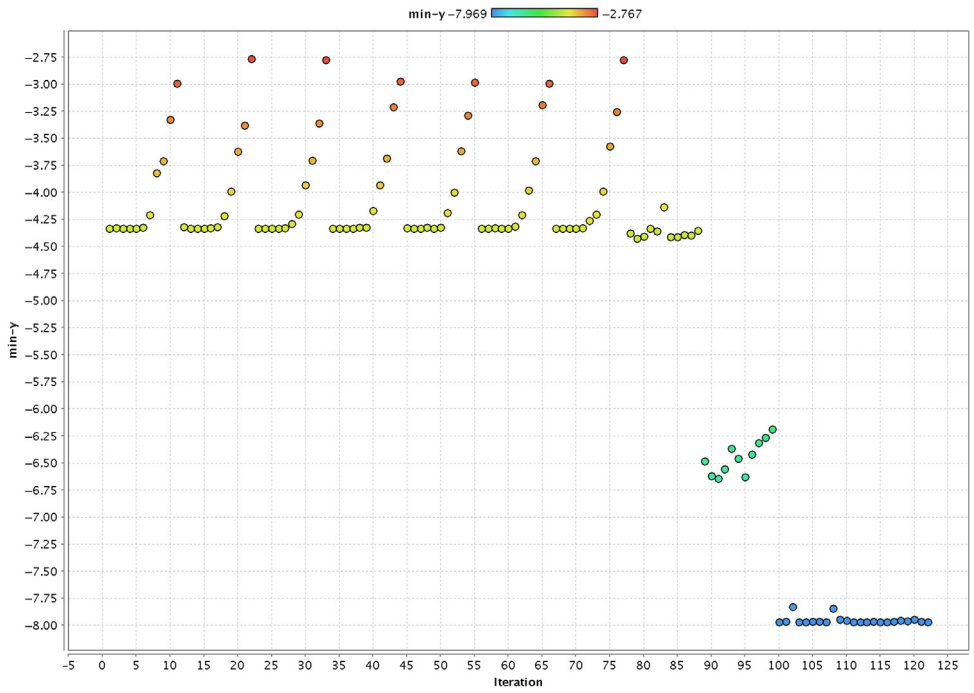


FIGURE 15.23
Progression of the quadratic greedy search optimization.

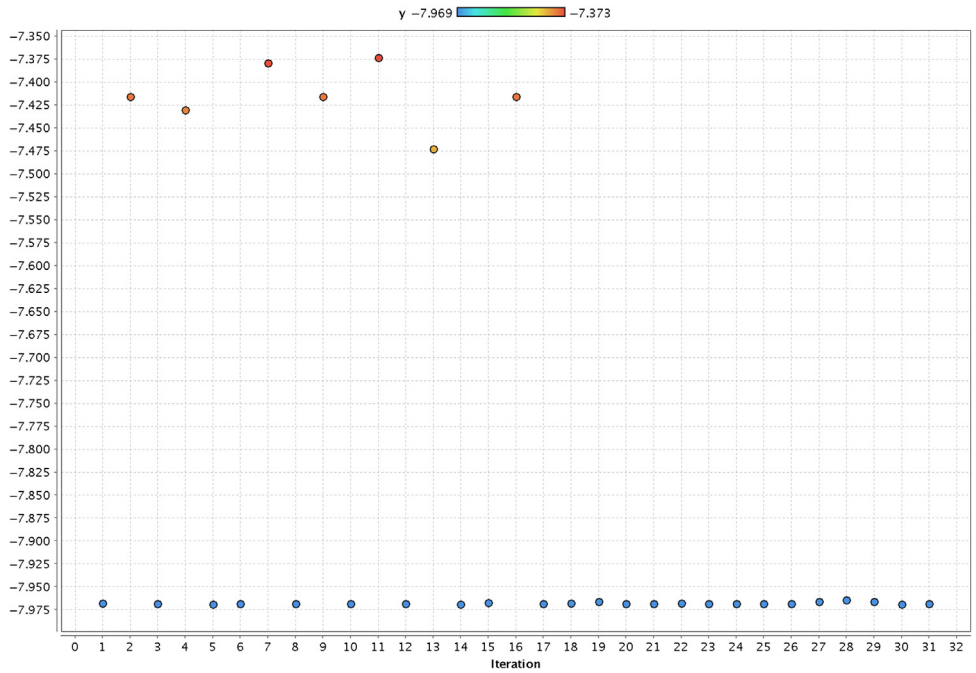


FIGURE 15.24
Progression of the genetic search optimization.

local minimum—right from the first few iterations it can be seen that the neighborhood of the lowest point has been approached. The evolutionary method is particularly useful if one does not initially know the domain of the functions, unlike in this case where they are known. It is evident that it takes far fewer steps to get to the global minimum with a high degree of confidence—about 18 iterations as opposed to 90 or 100. Key concepts to understanding this algorithm are *mutation* and *cross-over*, both of which are possible to control using the RapidMiner GUI. More technical details of how the algorithm works are beyond the scope of this book and some excellent resources are listed at the end of this chapter (Weise, 2009).

To summarize, there are three optimization algorithms available in RapidMiner all of which are nested operators. The best application of optimization is for the selection of modeling parameters, for example, split size, leaf size, or splitting criteria in a decision tree model. One builds their machine learning process as usual and inserts this process or nests it inside of the optimizer. By using the Edit Parameter Settings . . . control button, one can select the parameters of any of the inner process operators (for example a *Decision Tree*, *W-Logistic*, or *SVM*) and define ranges to sweep. Grid search

is an exhaustive search process for finding the right settings, but is expensive and cannot guarantee a global optimum. Evolutionary algorithms are extremely flexible and fast and are usually the best choice for optimizing machine learning models in RapidMiner.

15.7 INTEGRATION WITH R

R is a popular programmable open source statistical package that can execute scripts. The RapidMiner process can invoke R, send data, process it in R and receive back the data for further processing, modeling, and visualization in RapidMiner. RapidMiner offers an operator *Execute R* to invoke R and run the script that is contained in the operator. The script can be edited under the *Edit Text* parameter. Fig. 15.25 shows a process with the R operator and a sample script.

The script contains a function rm_main that can accept the datasets connected to the input port of the Execute R operator. Similarly, the return part

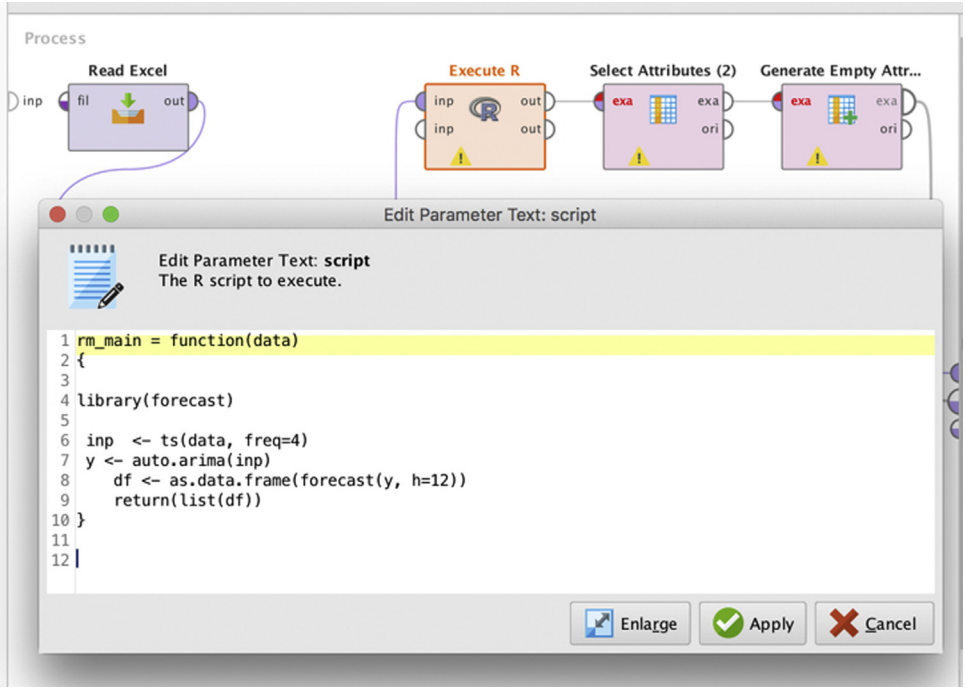


FIGURE 15.25
Integration with R.

of the function sends the dataframe (dataset in R) back to the output port of the operator.

15.8 CONCLUSION

As with other chapters in this book, the RapidMiner process explained and developed in this discussion can be accessed from the companion site of the book at www.IntroDataScience.com. The RapidMiner process (*.rmp files) can be downloaded to the computer and can be imported to RapidMiner from File > Import Process. The data files can be imported from File > Import Data.

This chapter provided a high-level view of the main tools that one would need to become familiar with in building data science models using RapidMiner. Firstly the basic graphical user interface for the program was introduced. Then options by which data can be brought into and exported out of RapidMiner were discussed. An overview of the data visualization methods that are available within the tool were provided, because quite naturally, the next step of any data science process after ingesting the data is to understand in a descriptive sense the nature of the data. Tools were then introduced that allow one to transform and reshape the data by changing the type of incoming data and restructuring them in different tabular forms to make subsequent analysis easier. Tools that would allow us to resample available data and account for any missing values were also introduced. Once familiar with these essential data preparation options, it is possible to apply any of the appropriate algorithms described in the earlier chapters for analysis. Optimization operators were also introduced that allow one to fine-tune their machine learning algorithms so that an optimized and good quality model can be developed to extract any insights.

With this overview, one can go back to any of the earlier chapters to learn about a specific technique and understand how to use RapidMiner to build models using that machine learning algorithm.

References

Ahuja, R. O. (2000). A greedy genetic algorithm for quadratic assignment problem. *Computers and Operations Research, 27*(10), 917–934.

Bahmani, S.R. (2013). Greedy sparsity-constrained optimization. *Statistical Machine Learning, 14*, 807–841.

Mierswa, I.W. (2006). YALE: Rapid prototyping for complex data mining tasks. *Association for computing machinery – Knowledge discovery in databases* (pp. 935–940).

Weise, T. (2009). *Global optimization algorithms – Theory and application*. <http://www.it-weise.de/>.