Chapter No. 9
"Resource Constraints"
In this package, you will find:

A Biography of the author of the book
A preview chapter from the book, Chapter NO.9 "Resource Constraints"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Andrew Chisholm completed his degree in Physics from Oxford University nearly thirty years ago. This coincided with the growth in software engineering and it led him to a career in the IT industry. For the last decade he has been very involved in mobile telecommunications, where he is currently a product manager for a market-leading test and monitoring solution used by many mobile operators worldwide.

Throughout his career, he has always maintained an active interest in all aspects of data. In particular, he has always enjoyed finding ways to extract value from data and presenting this in compelling ways to help others meet their objectives. Recently, he completed a Master's in Data Mining and Business Intelligence with first class honors. He is a certified RapidMiner expert and has been using this product to solve real problems for several years. He maintains a blog where he shares some miscellaneous helpful advice on how to get the best out of RapidMiner.

For More Information:
He approaches problems from a practical perspective and has a great deal of relevant hands-on experience with real data. This book draws this experience together in the context of exploring data—the first and most important step in a data mining process.

He has published conference papers relating to unsupervised clustering and cluster validity measures and contributed a chapter called *Visualizing cluster validity measures* to an upcoming book entitled *RapidMiner: Use Cases and Business Analytics Applications*, Chapman & Hall/CRC

I would like to thank my family, and in particular my wife Jennie for putting up with me while I wrote this book.

For More Information:
Exploring Data with RapidMiner

This book is a practical guide to exploring data using RapidMiner Studio. Something like 80 percent of a data mining or predictive analytics project is spent importing, cleaning, visualizing, restructuring, and summarizing data in order to understand it. This book focuses on this vital aspect and gives practical advice using RapidMiner Studio to help with the process.

A number of techniques are illustrated and it is the nature of exploratory data analysis that they can be re-used and modified in different places. By drawing these techniques together into a context, the reader will get a better sense of how RapidMiner Studio can be used in general and gain more confidence to use it.

What This Book Covers

Chapter 1, Setting the Scene, describes the main challenges when mining real data. These challenges arise because data is big and, in the real world, it is unstructured, difficult to visualize, and time consuming to bring order to.

Chapter 2, Loading Data, describes the different ways of loading data into RapidMiner Studio and the advanced techniques sometimes needed to transform raw unstructured data into a common format.

Chapter 3, Visualizing Data, describes the visualization techniques available in RapidMiner Studio to help make sense of data.

Chapter 4, Parsing and Converting Attributes, explains that data is rarely in precisely the right format and, therefore, needs to be parsed to extract specific information or converted into a different representation.

Chapter 5, Outliers, explains that real data contains values that do not seem to fit the rest of the data. There are many reasons for this and it is important to have a strategy for identifying and dealing with them, otherwise model accuracy risks can be severely compromised.

Chapter 6, Missing Values, explains that real data inevitably contains missing values. Simple deletion of rows containing missing values can quickly lead to a significant reduction in the performance of a data mining algorithm. Much better techniques exist.

Chapter 7, Transforming Data, covers techniques to restructure the data into new representations that can assist its exploration and understanding.

Chapter 8, Reducing Data Size, explains that reducing the number of rows will generally speed up processing but will reduce accuracy. Balancing this is important for large datasets. Reducing the number of columns of data can often improve model accuracy and for large datasets it is doubly valuable as it can speed up processing in general.

Chapter 9, Resource Constraints, explains that processing large amounts of data requires a lot of physical processing power and memory, to say nothing of the amount of time. This chapter gives some techniques to help measure process performance. Sometimes, it is not possible to process the data using available resources and in this situation, some techniques can be adopted to persuade the process to complete.

Chapter 10, Debugging, explains that when something goes wrong, it can be frustrating and time consuming to detect and resolve the problem. This chapter gives some generic methods for making this process a bit easier.

Chapter 11, Taking Stock, explains that having reached this point, the reader will have a greater visibility of the possibilities to process, clean, and explore data as part of the data mining process. This will be a stepping stone to more complex data mining techniques.

For More Information:  
Resource Constraints

Processing large amounts of data requires a lot of physical processing power and memory, to say nothing of the amount of time needed for the processing. Sometimes, it is not possible to process the data using the available resources, and in this situation, some techniques can be adopted to induce the process to complete.

This becomes particularly important when building models where the processing time can depend exponentially on the number of attributes, the number of examples, and the model itself. It is therefore important to know how long something will take by doing a measurement on a smaller sample of the data. From there an estimate can be made of what performance will be for all the data, and then steps can be taken to improve performance if needed. This chapter is therefore structured as follows:

- Measuring and estimating performance
- Splitting data into batches
- Parallel processing

Measuring and estimating performance

Often, when building a model or eliminating correlated attributes, I find that more than 10 minutes have elapsed, and out of frustration I stop the process. In reality, I may have no idea whether another 1 minute is needed or whether it will take a month. In fact, this is an important general point because large data and complex processing inevitably takes time. So, it is not an unreasonable question to ask how much processing time and what resources are needed in a production context. For example, a classification process might occasionally require that the classification model be recreated. It will be important to know how long this will take, so appropriate plans can be made.

Given that we have a data mining product in front of us, we can use it to predict how long something will take if we take some measurements.

For More Information:  
Resource Constraints

Measuring performance

It is very straightforward to measure how long an operator takes to execute. One simple approach is to use the Log operator to record time using the built-in values recorded by all operators. An example is shown in the following screenshot:

The dropdown has the following possible values that are relevant for time measurements:

- cpu-execution-time
- cpu-time
- execution-time
- looptime
- time

Of these, looptime and time are measured in milliseconds and give a measure of how much time elapsed since the last time the operator was called. The execution-time measure is the time required to execute the operator itself, also measured in milliseconds, and is the one that is most useful.

For More Information:
By logging this data and converting it into an example set, calculations can be performed to see how an operator is performing. This is straightforward to implement and is potentially very accurate. For measurements where sequences of operators are involved and an approximate view of performance is acceptable, an alternative is possible. This alternative is to create a macro containing a timestamp immediately before the part of the process of interest and another immediately after. By subsequently using the Generate Macro operator, calculations can be done to determine a delta time between the start and stop timestamps.

In addition to logging timings, it is important to record some starting environmental information—such as the number of attributes and number of examples being processed—that will potentially drive the elapsed time. By repeating the process for different starting conditions, a full set of training data can be obtained; this can be used to make a prediction about what performance would be with different numbers of attributes and examples.

An example process that can form the basis of such an investigation is shown in the following screenshot (the process is called measurePerfBookVersion.xml and is available with the files that accompany this book; note that the downloadable version has a few bonus features):
To help understand the Loop Parameters operator, its parameters are shown in the following screenshot:

The top-left pane shows all the inner operators that are within the loop operator (these will be shown in the next screenshot). The top-right pane shows that one of the operators has been selected, and the bottom right pane shows the list of Parameters that will be applied in combination with the other parameters to produce a single run of the inner operators. In the example, the SetExamples.value operator has 11 possible values starting at 10, 20, and 30, as does the SetAttributes.value operator. The combination of 11 and 11 leads to 121 combinations — by simple multiplication — and this will be the number of times the loop will execute the inner operators.
Let's move on to the inner operators; these are shown in the following screenshot:

![Screenshot of inner operators in RapidMiner](image)

The first two of these—named **SetExamples** and **SetAttributes**—are **Set Macro** operators, and these **Set Macro** operators are used by the operator named **Execute Process**. The **macros** are called **NumberOfExamples** and **NumberOfAttributes** and the values for these are set by the loop operator. The **Performance** operator is simply present to provide a performance vector for the loop operator so that it functions properly. The performance vector is ignored and can be any convenient operator. I typically use the **Attribute Count** performance operator.

The macros used by the **Execute Process** operator and the parameters for this are shown in the following figure:

![Edit Parameter List: macros](image)
Resource Constraints

The **Execute Process** operator runs another process that has previously been created using RapidMiner and has been saved in the repository (the process is `dataGeneratorAndModel.xml`). This ability to execute processes is a very powerful and modular technique that facilitates the building up of libraries of processes to be used in different situations. Such a process can take named macros as parameters.

The names of the macros that will be used inside the **Execute Process** operator are shown on the right-hand side of the preceding figure and their values are shown on the left. The values are taken from the current value of the macros defined within the loop operator. By virtue of being inside the loop operator, the process is executed multiple times with different parameters (in this case 121 times).

The process that is executed takes care of performing the modeling or other time consuming processes, and this example returns an example set with one row that contains an attribute for the number of examples, another for the number of attributes, and one more for the measured elapsed time.

The operators at the heart of the whole process are shown in the following screenshot:

In this example, the operator performing the time consuming task is called **Validation**. The other three are **Generate Macro** operators and are extremely simple. The **StartTheClock** operator creates a macro called `startTheClock` as shown in the following image:
The function calculates the difference in milliseconds between the present time and the time at the beginning of the UNIX epoch.

The operator called `StopTheClock` creates a macro — creatively called `stopTheClock` — using the same function expression, and finally, the calculation of the elapsed time is done in the operator `CalculateElapsedTime` by generating a macro called `elapsed`, which simply subtracts the two macros. This is shown in the following image:

All macros can be recorded in the log by first using the `Provide Macro as Log Value` and then simply logging the macro's value from this operator. The log file can then be converted to an example set using the `Log to Data` operator.
Resource Constraints

The end result of this is an example set that looks something similar to the following table:

<table>
<thead>
<tr>
<th>Row No.</th>
<th>Elapsed Time</th>
<th>Number of Attributes</th>
<th>Number of Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>105</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>34</td>
<td>205</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>38</td>
<td>285</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>35</td>
<td>390</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>39</td>
<td>550</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>36</td>
<td>565</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>669</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>41</td>
<td>785</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>42</td>
<td>855</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>43</td>
<td>1090</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>44</td>
<td>1092</td>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>

Visual inspection is often revealing, and this data can be plotted as a block plot as shown in the following figure:

---

For More Information:  
This data shows the time performance of a neural network as it models data of different sizes. The color of a block is a measure of the elapsed time for the execution of a neural network as a function of NumberOfAttributes and NumberOfExamples. A darker shade means more elapsed time, and the range is 219 ms at the bottom-left and 7,920 ms at the top-right.

From here, it is quite simple to estimate the performance as the number of attributes or examples increases. Of course, it would also be possible to fit some sort of function to the data to make a prediction. It is beyond the scope of this book to go into these details, but from the data we just saw, a model I made predicted 94 seconds to process 100 attributes and 100 examples, 3,910 seconds to process 200 attributes and 200 examples, and 1.5 million years to process 1,000 attributes and 1,000 examples. The model turned out to be inaccurate because when I re-ran it with training data that explicitly had 100 examples and 100 attributes, the actual time was near 55 seconds and the prediction for 1,000 by 1,000 was near 10,000 years. Nonetheless, this is still a long time and there is enough accuracy to illustrate the main point.

Adding memory

Adding more memory often speeds things up and allows some processes to complete. It is always worth checking to ensure that you are using the maximum amount of memory that is available.

The first thing to say is that 32-bit operating systems can generally address a maximum of 4 GB, so it is always worth getting a 64-bit version where this limit is much higher. A suitable processor on which the operating system can be run is also required. The general rule is to go for the 64-bit architecture; however, you can consult an expert to get clarity. Secondly, RapidMiner Studio can be run in a number of ways. I generally launch the GUI from a batch file because this gives more control, particularly in a Windows environment, where a separate console is launched. The log information is written to this console. This can be very useful if things become unresponsive. The file to launch the GUI is called RapidMiner-Studio.bat on Windows machines and is located in the scripts folder where RapidMiner is installed. In Linux environments, it is called RapidMiner-Studio.sh.

By defining the JAVA environment variable MAX_JAVA_MEMORY, it is possible to change the amount of memory RapidMiner uses. For example, to get 4 GB of memory, this variable would be set to 4,096 via the Control Panel in Windows or an appropriate configuration change in UNIX. If you find that processes are running slowly and the system monitor in the GUI shows memory getting low in RapidMiner, it is always worth setting the upper limit to the highest value that you can. Buying more memory is also a good option.

However, there will always be a time when you won't have enough memory and the question will arise as to what to do then.
Parallel processing

If faced with a process that is simply taking too long, clearly more memory can help—as already discussed. If that fails, a more powerful processor is obviously something to consider. If there is not enough money to do that or if it simply does not work, a parallel approach can be considered. Some operators can be run in parallel, and RapidMiner Studio allows this to be done where the processor has two cores. To take advantage of this, it is necessary to download the parallel processing extension available from the Rapid-I Marketplace. Once this is done, a new configuration option checkbox appears on some operators; it allows them to be executed in parallel. Affected operators include the main process operator, looping operators, the subprocess operator, the branch and select operators, and the process evaluation operators. Typically, an operator that contains a loop or an implied subprocess can be made to run in a parallel fashion. Operators such as those in the Modeling and Data Transformation groups do not have this option.

For example, the X-Validation operator allows the partitions containing training and testing to be run in parallel. This is possible because the cross validation operation is inherently parallel as the individual partitions are self-contained and do not depend on each other.

Examples of processes that could not be carried out in parallel would include ones where calculations that require all the data are being performed. For example, normalizing an attribute within an example set requires all the data to be processed to determine various statistics to then apply to the individual examples.

In the context of exploring data, some activities could be carried out in parallel. For example, if multiple files are to be read in and processed so that the processing of one file depends only on the contents of that file, it would be possible to take advantage of parallel execution. The simplest possible process would be two Read CSV operators reading two files. If these are placed in the main process and the parallelize main process option is set to true, RapidMiner will execute the file reading across the available CPUs.

A word of caution about parallel processing. Even if the process can be done in a parallel fashion, there is still a risk that one instance will interfere with another. Perhaps macros are shared between instances, or it could be that the data is shared. Either way, this can cause processes to fail, so be careful.
Restructuring processes

It is also always worth seeing whether the process can be restructured to be more efficient. A review of a process may reveal that expensive operations are being performed repeatedly and unnecessarily. In this situation there are some operators that can help.

The Store and Retrieve operators allow objects to be stored in the repository. Objects such as example sets, models, and weights can be stored and retrieved in this way. Typically, a process will perform an expensive operation once and store it in the repository. Subsequent operators can retrieve the object whenever it is needed. These operators can also be used to implement a checkpoint regime within a big process. This is relatively complex to set up, but it may be worth having the process determine where it reached in a long processing step so that it starts up where it left off.

The Recall and Remember operators are similar except they do not persist data to the repository. These can be used in a way similar to the Store and Retrieve operators except that the type of the object must be specified, and when using the Recall operator, the option to remove it from the store can be specified. These operators will consume memory but are likely to be relatively quicker than those that interact with the repository.

Summary

It is part of human nature to always push the limits of what is possible, and so it is inevitable that you will encounter performance problems. This chapter has given some insight into ways to measure performance and some basic approaches to improve it.

The next chapter gives some tips to help debug processes when things aren't going your way.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.