SimaFore

Decision Tree Digest – An eBook

Understand, build and use decision trees for common business problems with RapidMiner

Bala Deshpande, Ph.D., MBA

Table of Contents

Chapter 1: What are Classification and Regression Trees?

Those beginning to explore predictive analytics tools are confused by the dozens of techniques that are available, apparently to address the same type of problem. Classification (also known as Decision tree) and Regression trees are probably one of the most common and easily understood tools. Some vendors refer to these techniques collectively as CART.

There are several types of "trees" that beginners must get a grip on. Of these, two types are probably the most significant.

The first type is a **Classification** tree. This is also referred to as a **Decision** tree by default. The other basic decision tree in common use is a **Regression** tree, which also works in a very similar fashion. This chapter summarizes the main differences between them: when to use each, how they differ and some cautions.

1. When to use classification versus regression trees

This might seem like a trivial issue - after you know the difference! **Classification trees**, as the name implies are used to separate a dataset into classes belonging to the response variable. Usually the response variable has two classes: ***Yes or No (1 or 0).*** If the response variable has ***more*** than 2 categories, then a variant of the algorithm, called C4.5, is used. For binary splits however, the standard CART procedure is used. Thus classification trees are used when the response or target variable is categorical in nature.

Regression trees are needed when the response variable is numeric or continuous, for example, when you need to predict the price of a consumer good based on several input factors. Thus regression trees are applicable for *prediction* type of problems as opposed to *classification*.

Keep in mind that in either case, the predictors or independent variables may be categorical or numeric. It is the **target variable** that determines the type of decision tree needed.

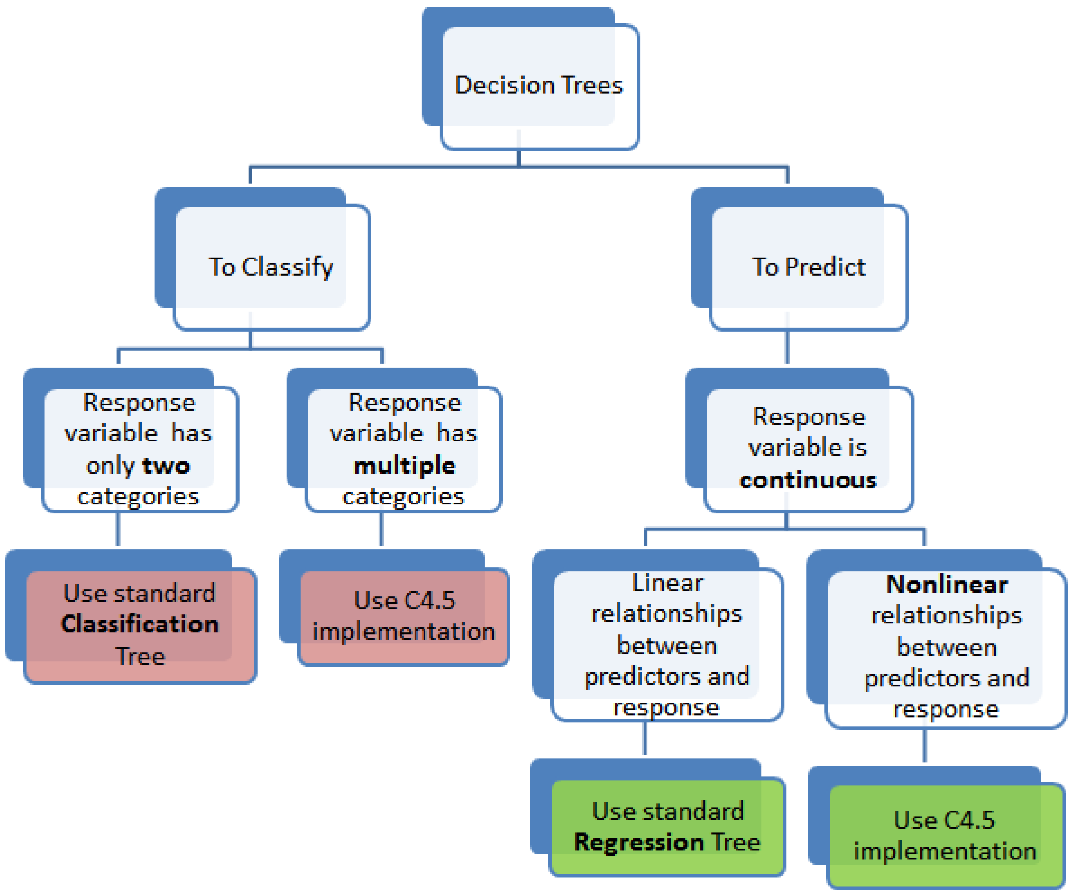
2: How they work

In a standard classification tree, the idea is to split the dataset based on homogeneity of data. Let us say for example we have two variables: age and weight that predict if a person is going to sign up for a gym membership or not. In our training data if it showed that 90% of the people who are older than 40 signed up, we split the data here and age becomes a top node in the tree. We can almost say that this split has made the data "90% pure". Rigorous measures of impurity, based on computing proportion of the data that belong to a class, such as entropy or Gini index are used to quantify the homogeneity in Classification trees.

In a regression tree the idea is this: since the target variable does not have classes, we fit a regression model to the target variable using each of the independent variables. Then for each independent variable, the data is split at several split points. At each split point, the "error" between the predicted value and the actual values is squared to get a "Sum of Squared Errors (SSE)". The split point errors across the variables are compared and the variable/point yielding the lowest SSE is chosen as the root node/split point. This process is recursively continued.

A C4.5 classification tree (for more than 2 categories of target variable) uses information gain to decide on which variable to split. In a corresponding regression tree, standard deviation is used to make that decision in place of information gain. Regression trees, by virtue of using regression models lose the one strength of standard decision trees: ability to handle highly non-linear parameters. In such cases, it may be better to use the C4.5 type implementation.

This tree below summarizes at a high level the types of decision trees used in practice.


Decision Trees
To Classify
To Predict
Response variable has only **two** categories
Response variable has **multiple** categories
Response variable is **continuous**
Use standard **Classification** Tree
Use C4.5 implementation
Linear relationships between predictors and response
Nonlinear relationships between predictors and response
Use standard **Regression** Tree
Use C4.5 implementation

Chapter 2: Advantages of using Decision Trees for predictive analytics

There are several distinct advantages of using decision trees in many classification and prediction applications. While some vendors use names such as **Classification and Regression trees** (CART or C&RT), they still refer to the same analytics technique at the core. Hence we will simply call them **Decision trees**. Here are some of the advantages of using decision trees for predictive analytics.

Advantage 1: Decision trees implicitly perform variable screening or feature selection

We explained in a separate article why feature selection is important in analytics. We also introduced a few common techniques for performing feature selection or variable screening. When we fit a decision tree to a training dataset, the top few nodes on which the tree is split are essentially the most important variables within the dataset and feature selection is completed automatically!

Advantage 2: Decision trees require relatively little effort from users for data preparation

To overcome scale differences between parameters - for example if we have a dataset which measures revenue in millions and loan age in years, this will require some form of normalization or scaling before we can fit a regression model and interpret the coefficients. Such variable transformations are not required with decision trees because the tree structure will remain the same with or without the transformation.

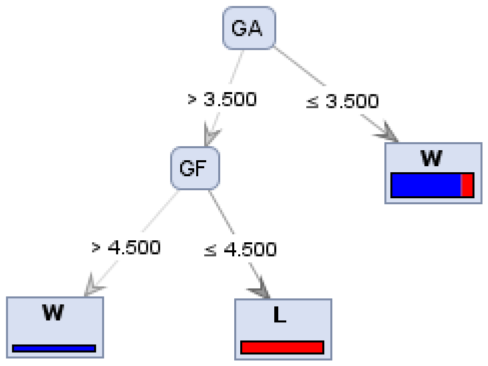
Another feature which saves data prep time: missing values will not prevent splitting the data for building trees. Decision trees are also not sensitive to outliers since the splitting happens based on proportion of samples within the split ranges and not on absolute values.

Advantage 3: Nonlinear relationships between parameters do not affect tree performance

As we described here, highly nonlinear relationships between variables will result in failing checks for simple regression models and thus make such models invalid. However, decision trees do not require any assumptions of linearity in the data. Thus, we can use them in scenarios where we *know* the parameters are nonlinearly related.

Advantage 4: The best feature of using trees for analytics - easy to interpret and explain to executives!

Decision trees are very intuitive and easy to explain as this simple tree shows the factors



affecting winning (W) or losing (L)!

However, all these advantages need to be tempered with one key disadvantage of decision trees: without proper pruning or limiting tree growth, they tend to overfit the training data, making them somewhat poor predictors.

In the next chapter we will briefly examine the algorithms which enable building decision trees.

Chapter 3: The technical kernel of decision tree algorithms

The striped patterns on zebras exist for several reasons: Camouflage, body temperature control and probably the most relevant for us here, for *identification*. Humans also make decisions rapidly by identifying and matching patterns. Patterns quickly reveal similarities and differences. While patterns can tell a great story, the key is to convert a pattern into a number or a set of numbers that distinguish one pattern from another.

Such techniques are available in information theory. Formulations such as entropy and mutual information encapsulate these ideas. (This **video** explains in a minute how entropy can work for measuring uncertainty. You can continue reading below or watch the video).

Imagine a box that can contain one of three colored balls inside - red, yellow and blue. Without opening the box, if you were to guess what colored ball is inside, you are basically dealing with uncertainty. Now what is the highest number of "yes"/"no" questions that can be asked to reduce this uncertainty?

Is it red? No.

Is it yellow? No.

Then it must be blue. That is *two* questions. If there was a fourth color, green, then the highest number of (yes/no) questions is *three*. If you extend this reasoning, it can be mathematically shown that the maximum number of binary questions needed to reduce uncertainty is essentially **log (T)** where the log is taken to base 2 and T is the number of possible outcomes.

(ex: If you have only 1 outcome, then log (1) = 0 which means there is no uncertainty)! If there are T events with equal probability of occurrence then T = 1/P.

Claude Shannon used this idea to define entropy as *log (1/P)* or **-log P** where P is the probability of an event occurring. If the probability for all events is not identical, we need a weighted expression and thus entropy, H

H = -Summation (pi log pi)

Entropy is a very useful tool for any predictive analytics or risk management professional. A very common application of information entropy is in building decision trees. The main ideas behind using entropy for building a decision tree are these:

1. Using shannon entropy, sort the dataset into homogenous and non-homogenous variables. Homogenous variables have low entropy and non-homogenous variables have high entropy

2. Weight the influence of each independent variable on the target or dependent variable using the concept of joint entropy.

3. Compute the information gain, which is essentially the reduction in the entropy of the target variable due to its relationship with each independent variable. This is simply the difference between the target entropy found in 1 minus the joint entropy calculated in 2.

4. The independent variable with the highest information gain will become the "root" or the first node on which the data set is divided.

5. Repeat this process for each variable for which the shannon entropy is non-zero. If the entropy of a variable is zero, then that variable becomes a "leaf" node.

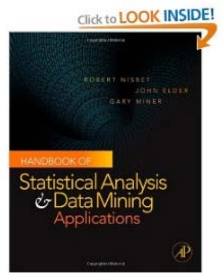
More detailed description of this algorithm using a simple example is found here. In the following three chapters, we will describe how to use RapidMiner, the open source data mining software to build decision trees to perform credit scoring.

Chapter 4: Building Decision Trees for Credit Scoring using RapidMiner in 4 steps

Credit scoring is one fairly common business analytics application. Some types of problems where credit scoring could be applied are:

1. **Prospect filtering**: Identify which prospects to extend credit to and how much credit would be an acceptable risk
2. **Default Risk detection**: Decide if a particular customer is likely to default
3. **Bad debt collection**: Sort out those debtors who will yield a good cost (of collection) to benefit (of receiving payment) performance.

We use a data set from the book shown here and describe how to use the open source software RapidMiner to build a decision tree for addressing **prospect filtering problem**. If you have this book, you can compare the solution shown here to the one given in the book, which uses STATISTICA, a commercial analytics tool.



Setting up a decision tree analysis in Rapidminer is presented very nicely in a video by Thomas Ott. However our article lays down the steps for applying decision trees for credit scoring applications and some common problems encountered by non-expert users applying these immensely valuable open source tools for analysis.

There are four main steps in setting it up:

1. Read in the data (from a spreadsheet)

2. Split data into training and testing samples

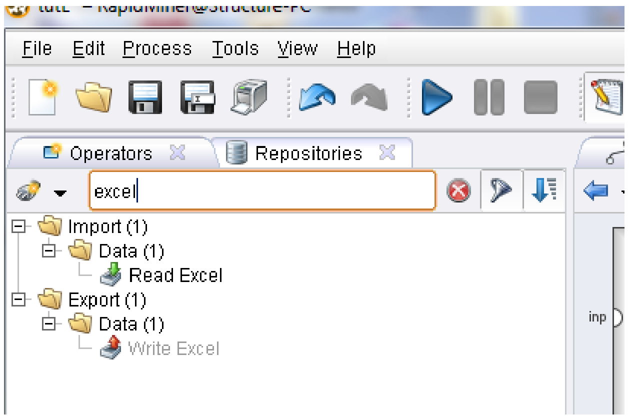
3. Train the decision tree

4. Apply the model and evaluate the performance

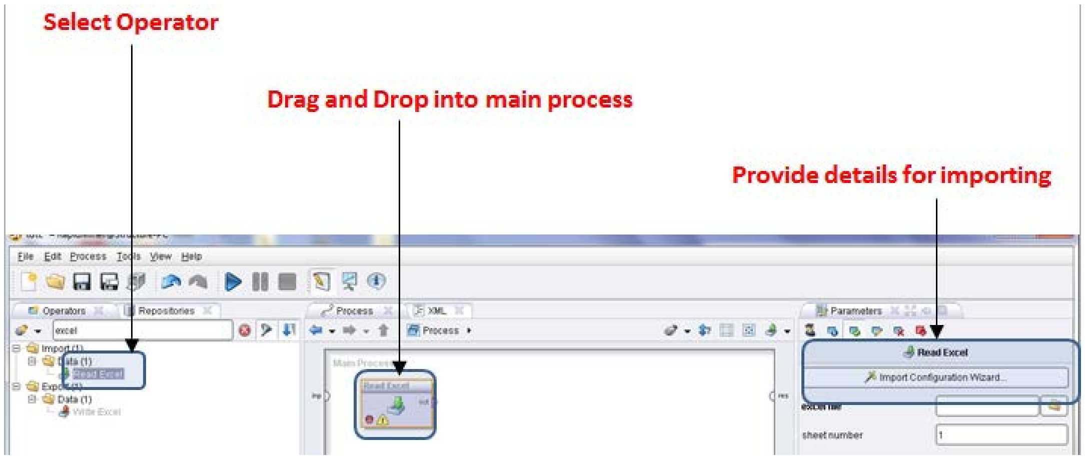
This first part of the series focuses on step 1, which may seem rather elementary, but can consume a lot of time if not done properly. The next few parts will describe other steps in detail.

Step 1: Read in the data

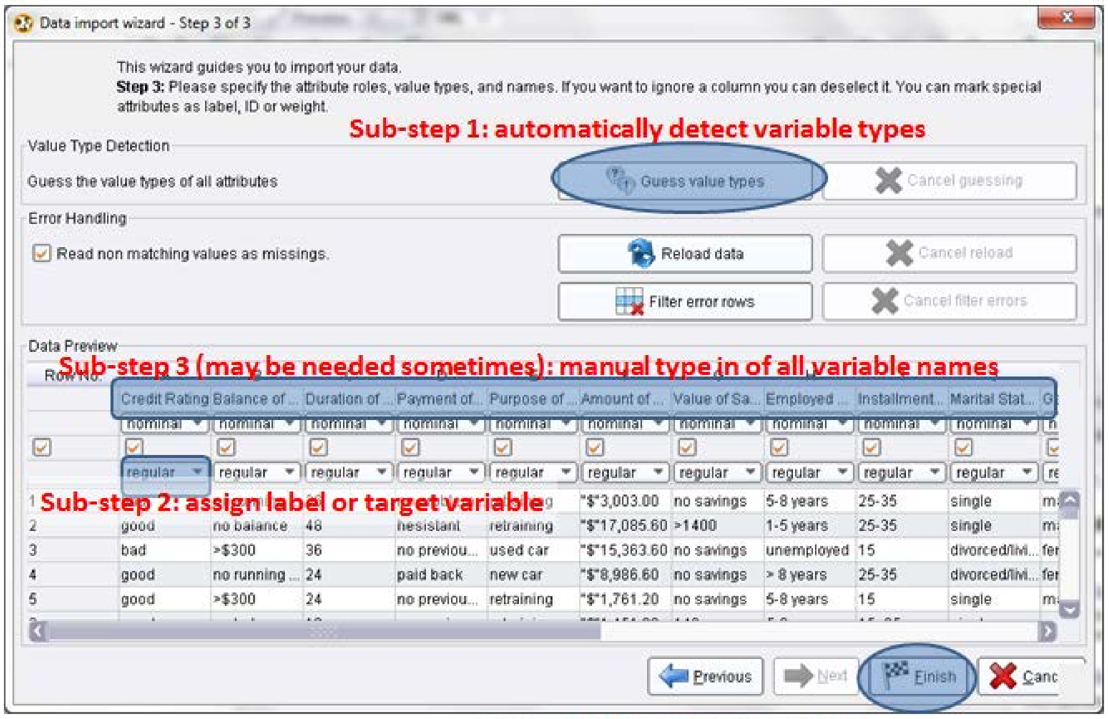
RapidMiner's easy interface allows quick importing of spreadsheets. The best part about the interface is the panel on the left, called the "Operators". By simply typing in text in the box provided automatically pulls up all available RapidMiner operators that match the text - pretty handy! In this case, we need an operator to read an XL spreadsheet, and so we simply type "excel" in the box. As you can see, the two XL operators are immediately shown below: one for reading and one for exporting data.



Either double click on the "Read Excel" operator or drag and drop it into the "Main Process" panel - the effect is the same. Once the Read Excel operators appears in the main process window, we need to configure the data import process. What this means is telling RapidMiner which columns to import, what is contained in the columns and if any of the columns need special treatment.


Select Operator
Drag and Drop into main process
Provide details for importing
Operators
Repositories
Process
XML
Parameters
Read Excel
Import Configuration Wizard...
sheet number
1

This is probably the most "cumbersome" part about this step. RapidMiner has a feature to automatically detect (or Guess Value types). But it is a good exercise for the analyst to make sure that the right columns are picked (or excluded). ***Also, sometimes, the tool is unable to treat the first row of the spreadsheet as names in which case the user has to manually enter the names for all the columns in the attribute field as shown here.***



This wizard guides you to import your data.
Step 3: Please specify the attribute roles, value types, and names. If you want to ignore a column you can deselect it. You can mark special attributes as label, ID or weight.

Sub-step 1: automatically detect variable types

Value Type Detection

Guess the value types of all attributes

Error Handling

☑ Read non matching values as missings.

Data Preview

Sub-step 3 (may be needed sometimes): manual type in of all variable names

Row No.	Credit Rating	Balance of ...	Duration of ...	Payment of...	Purpose of ...	Amount of ...	Value of Sa...	Employed ...	Installment...	Marital Stat...	G
	nominal	nominal	nominal	nominal	nominal	nominal	nominal	nominal	nominal	nominal	n
☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
	regular	regular	regular	regular	regular	regular	regular	regular	regular	regular	re
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	"$"3,003.00	no savings	5-8 years	25-35	single	m
2	good	no balance	48	hesistant	retraining	"$"17,085.60	>1400	1-5 years	25-35	single	m
3	bad	>$300	36	no previou...	used car	"$"15,363.60	no savings	unemployed	15	divorced/livi...	fer
4	good	no running ...	24	paid back	new car	"$"8,986.60	no savings	> 8 years	25-35	divorced/livi...	fer
5	good	>$300	24	no previou...	retraining	"$"1,761.20	no savings	5-8 years	15	single	m

Sub-step 2: assign label or target variable

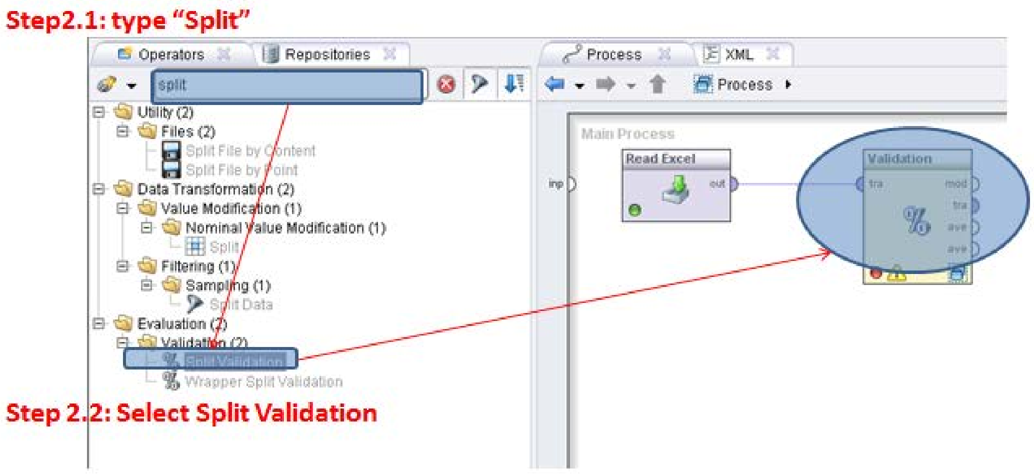
Sub-step 4: finish loading data

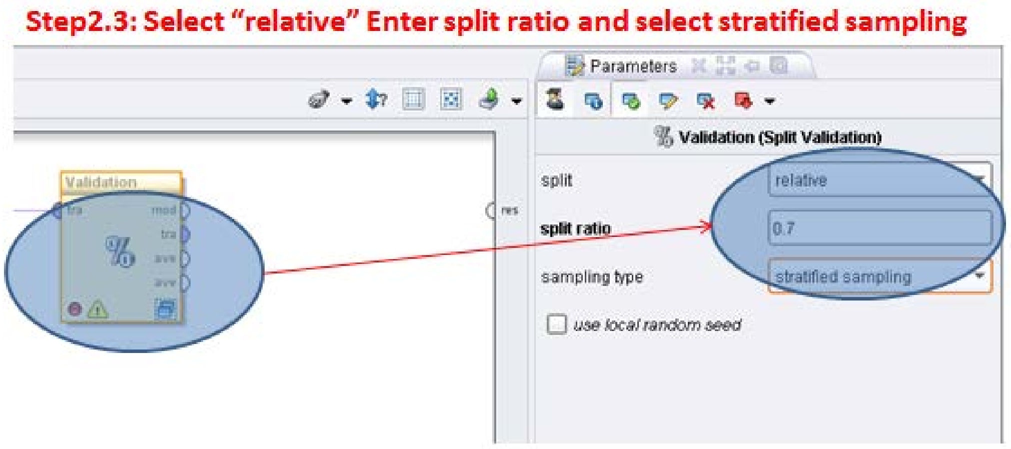
Once the data is imported, we must assign the target variable for analysis, also known as a "Label". In this case, it is the Credit Rating (column A) - see sub-step 2 shown in figure above. Finally it is a good idea to "run" RapidMiner and generate results to ensure that all columns are read correctly as demonstrated by Tom Ott's video above.

Step 2: Split validation setup

As with all modeling efforts, data must be separated into two sets: one for "training" or developing an acceptable model, and the other for "validating" or ensuring that the model works equally well on a different data set. The standard practice is to split the available data into a training set and a validation set. Typically the training set contains 70-80% of the original data. The remainder is set aside for testing or validation.

The graphics below show how to do this in RapidMiner. Among the many useful features of RapidMiner, the "Split Validation" tool allows setting up of splitting, adding training model *and* the validation check in one operator! **While this is very handy for someone experienced in data mining, for beginners, this may be a bit confusing**.

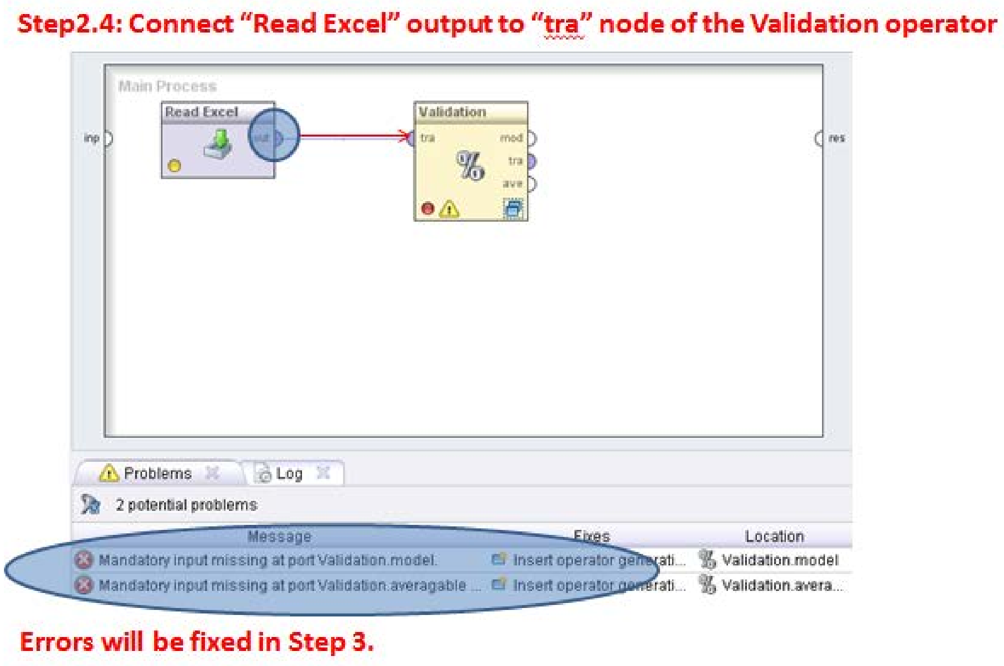



Step2.3: Select "relative" Enter split ratio and select stratified sampling
Parameters
Validation
Validation (Split Validation)

split: relative
split ratio: 0.7
sampling type: stratified sampling
use local random seed

Choose **stratified sampling** with a split ratio of 0.7 (70% training). Stratified sampling will ensure that all variables have more or less similar distributions of class values. Otherwise, there is a possibility that one or more of the variables will have unequal proportions (for example: of the 1000 samples in this data set, ***Balance of current account*** variable has about 27% of the entries with $0 as the class value. It is ***possible*** that a randomly split data set may have 90% [27% = 90% of 0.3] of entries for Balance of current account which are $0!)

The final sub step here is to connect the XL operator output to the Split Validation operator input. This will cause some errors to show up in the log window. This is fine because the process is not yet completed.

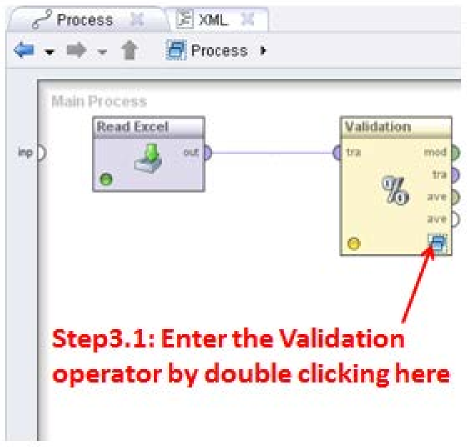

Step2.4: Connect "Read Excel" output to "tra" node of the Validation operator
Main Process
Read Excel
out
Validation
tra
mod
tra
ave
inp
res
Problems
Log
2 potential problems
Errors will be fixed in Step 3.

Message	Fixes	Location
Mandatory input missing at port Validation.model.	Insert operator generati...	Validation.model
Mandatory input missing at port Validation.averagable ...	Insert operator generati...	Validation.avera...

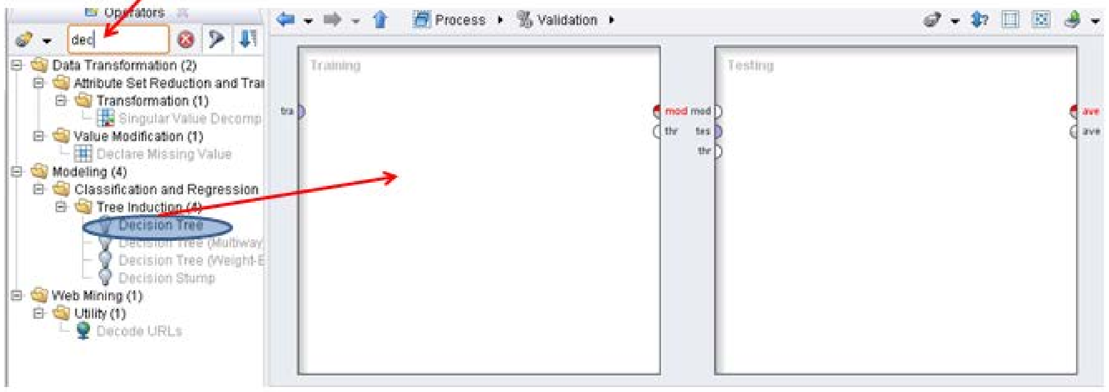
In the next step, we will set up the Decision tree parameters, discuss in detail the implication of each setting and run the model. The final step will be an evaluation of this analysis and comparison to the book solution.

Step 3: Training the decision tree

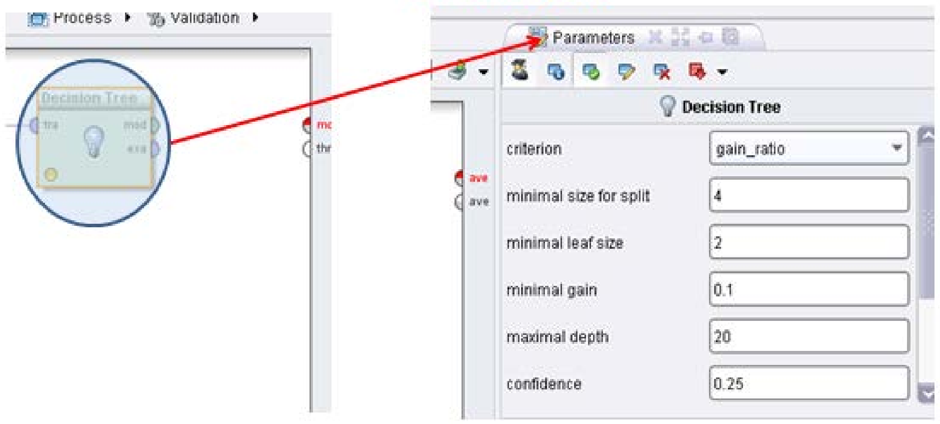
We will now show how to implement step 3. We will also discuss the various parameters to pay attention to while using the Decision Tree operator and what they mean. Upon following steps 3.1 to 3.3 below, we can modify tree parameters.



Step3.2: Type "dec" … in the operator box and select Decision Tree. Drag and drop into the training window



Step3.3: Selecting the Decision Tree operator in the Training Window will bring up the Parameters box



Decision Tree

criterion	gain_ratio
minimal size for split	4
minimal leaf size	2
minimal gain	0.1
maximal depth	20
confidence	0.25

The main parameters to pay attention to are the "Criterion" pull down menu and the minimal gain box. The criterion is essentially a splitting decision factor that answers when a node should be split.

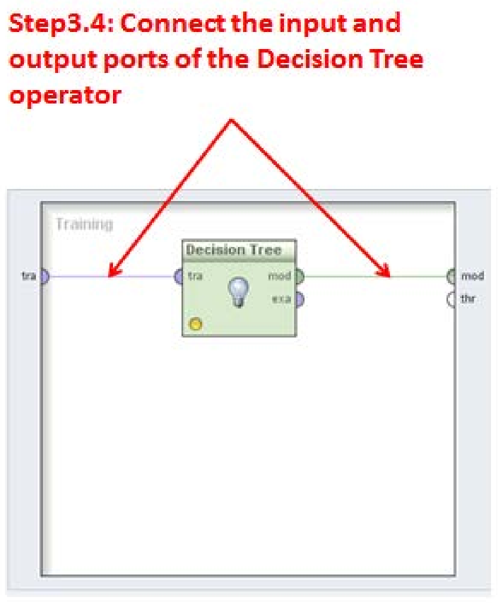
As discussed in chapter 3, decision trees are built up in a simple five step process by increasing information contained in the reduced data set following each split. If that sounds a bit topsy-turvy, think about it like this. Data by its nature contains uncertainties. We may be able to systematically reduce uncertainties and thus increase information by activities like sorting or classifying. When we have sorted or classified to achieve the greatest reduction in uncertainty, we have basically achieved greatest increase in information. Chapter 3 also explained why entropy is a good measure of uncertainty and how keeping track of it allows us to quantify information. So this brings up back to the 3 options which are available within RapidMiner for splitting decision trees.

1. ***Information Gain:*** Simply put this is computed as the information before the split minus information after the split. It works fine for most cases, unless you have a few variables which have a large number of values (or classes). Then these variables tend to end up as root nodes. This is not a problem, except in extreme cases. For example, each customer ID is unique and thus the variable has too many classes (each ID is a class). A tree that is split along these lines has no predictive value.
2. ***Gain Ratio (default):*** is usually a good option. Gain ratio overcomes the problem with Information gain by taking into account the number of branches that would result before making the split.
3. ***Gini Index:*** is also used sometimes, but does not have too many advantages over gain ratio.

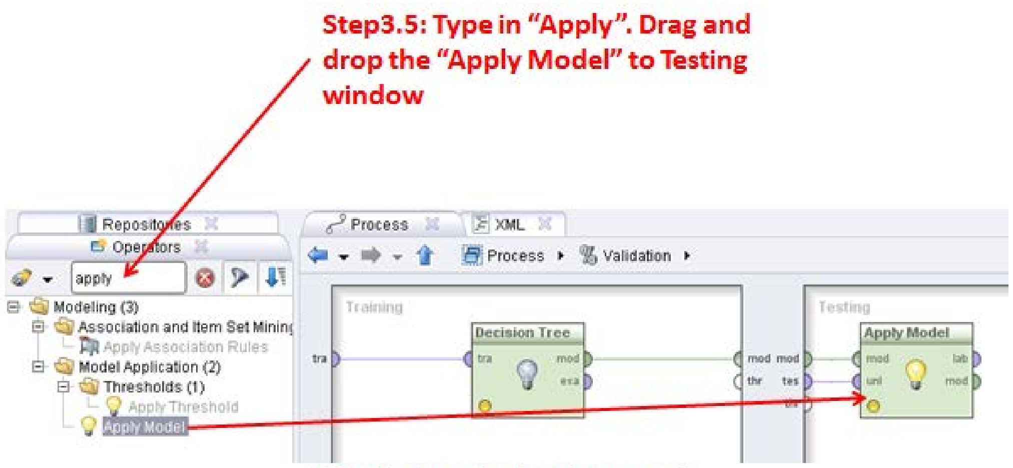
The other important parameter is the "***minimal gain***" value. Theoretically this can take any range from 0 upwards. In practice, a minimal gain of 0.2-0.3 is considered usable. Default is 0.1.

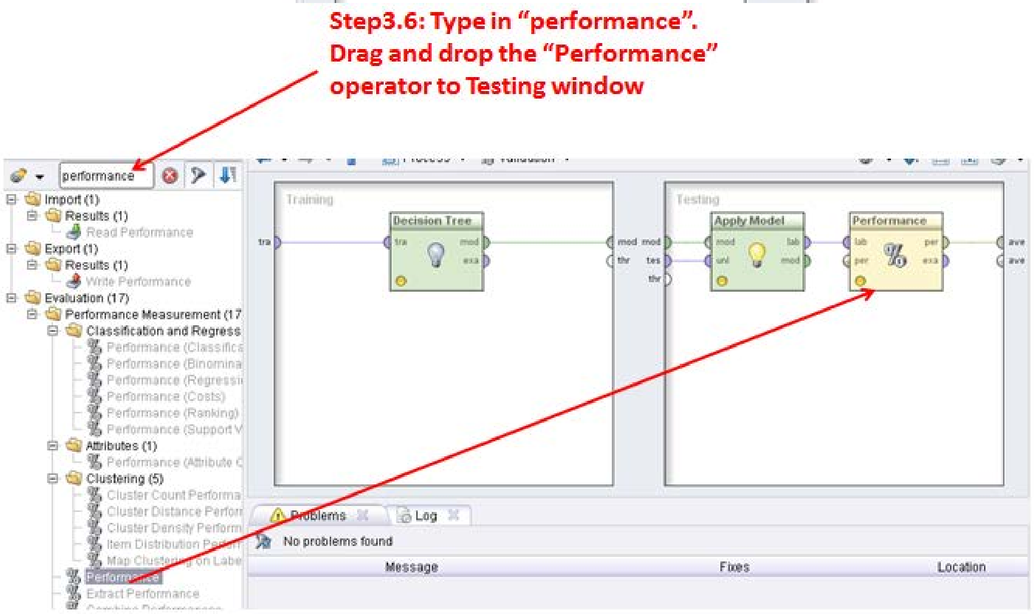
The other parameters (***minimal size for a split, minimal leaf size, maximal depth***) are determined by the size of the data set. In this case, we proceed with default values.

The last step in training the decision tree is to connect the input ports ("tra"ining) and output ports ("mod"el) as shown.


Step3.4: Connect the input and output ports of the Decision Tree operator
Training
Decision Tree
tra
tra
mod
exa
mod
thr

The model is ready for training. Next add two more operators: Apply Model and Performance and we are ready to run the analysis.





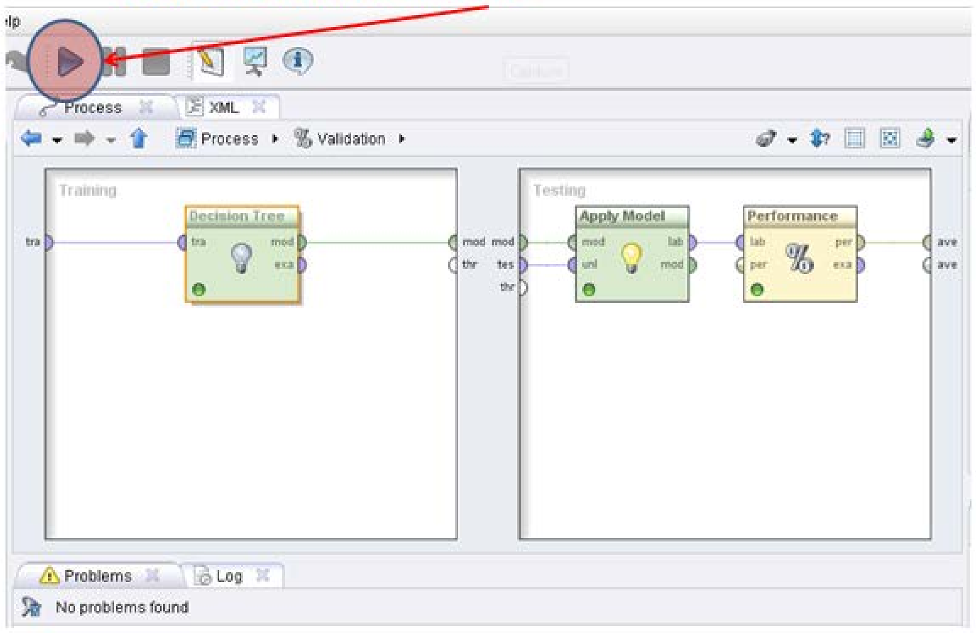
Remember to connect the ports correctly as this is another common newbie confusion:

- "mod"el of the Training window to "mod" on Apply Model
- "tes"ting of the Training window to "unl"abeled on Apply Model
- "lab"eled of Apply Model to "lab"eled on Performance
- "per"formance on Performance operator to "ave"rageable on output port

The final step before running the model is to go back to the main perspective by clicking on the blue up arrow (see step 3.5) and connect the output ports "mod"el and "ave" of Validation operator to the main outputs.

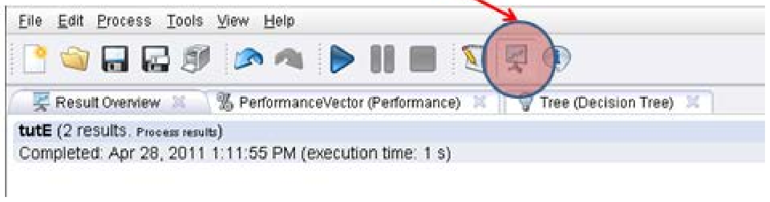
Step 4. Evaluate the performance

Step 4.1: Once you have connected all the ports correctly, you will see the "No problems found" message in the log window. Then click PLAY button to run model!



No problems found

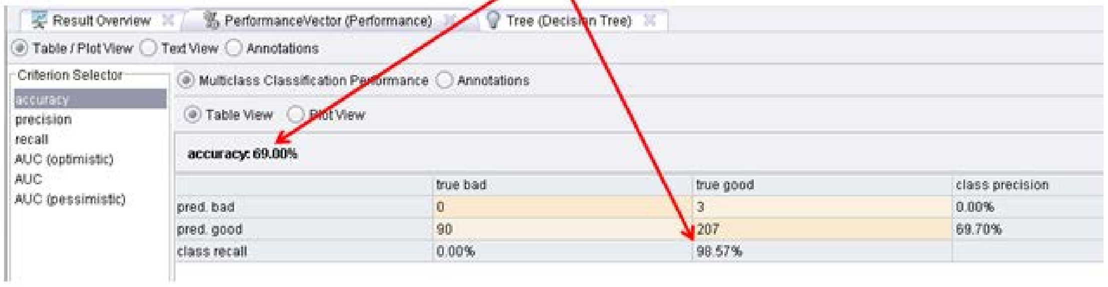
Step 4.2: After the model runs, click on the results icon to display 3 tabs



tutE (2 results. Process results)
Completed: Apr 28, 2011 1:11:55 PM (execution time: 1 s)

Click on the "Performance" tab to see this table below

Step 4.3: Verify the performance on the validation data: Accuracy is 69% and Sensitivity is 99%

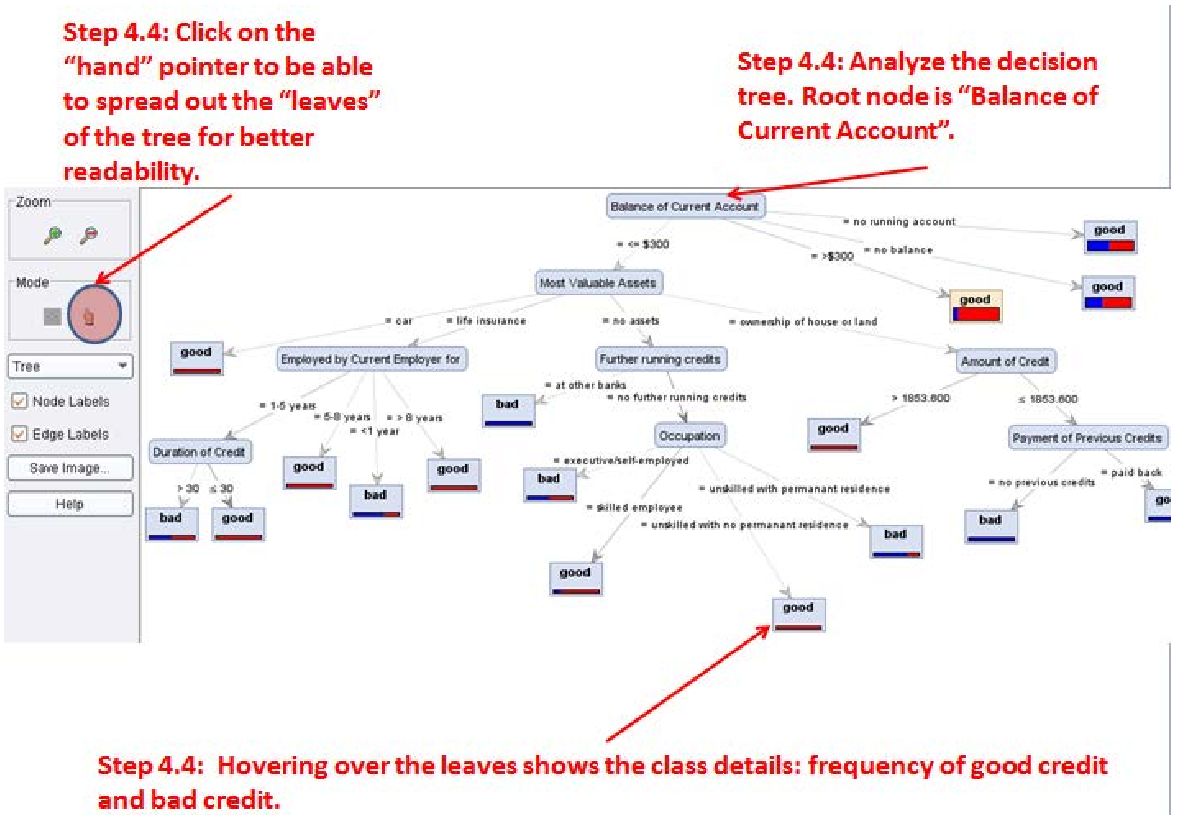


accuracy: 69.00%

	true bad	true good	class precision
pred. bad	0	3	0.00%
pred. good	90	207	69.70%
class recall	0.00%	98.57%	

RapidMiner Performance operator provides several options to check the model validity: accuracy, precision, recall, ROC and AUC charts. A discussion of common methods to assess classification model quality is available here.

Finally click on the "Tree" tab to see the Decision Tree itself. Note the instructions in the graphic below.

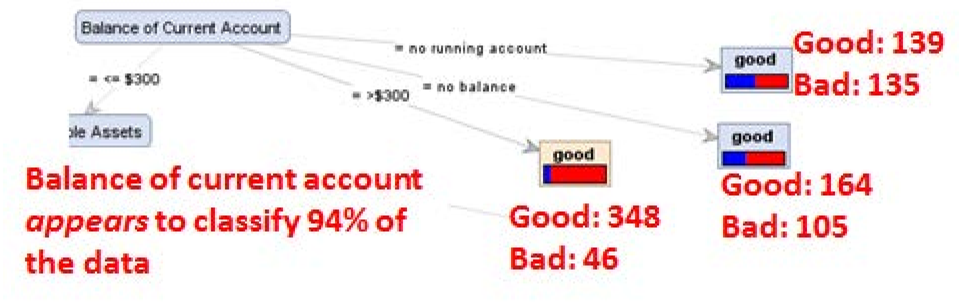

Step 4.4: Click on the "hand" pointer to be able to spread out the "leaves" of the tree for better readability.

Step 4.4: Analyze the decision tree. Root node is "Balance of Current Account".

Step 4.4: Hovering over the leaves shows the class details: frequency of good credit and bad credit.

Several important points must be highlighted:

1. The root node - **Balance of Current Account** - manages to classify nearly 94% of the data set. This can be verified by hovering the mouse over each of the 3 terminal leaves for this node. The total occurrences (of *good* and *bad*) for this node alone are 937. Specifically, if someone has a **Balance of Current Account** >= $300, then the chances of them having a "*good*" score is 88% (=348/394, see graphic below).



2. However, the tree is unable to clearly pick out good or bad scores, if there is "*no running account*" (only a 51% chance). A similar conclusion results if someone has "*no balance*".

3. If the **Balance of Current Account** is less than $300, then the other parameters come into effect and play an increasingly important role in deciding if someone is likely to have "*good*" or "*bad*" credit.

4. However, the fact that there are numerous terminal leaves with frequencies of occurrence as low as 2 (for example, "**Duration of credit**"), it implies that the tree suffers from "overfitting". One way we could have avoided this situation is by changing the **Decision Tree** criterion "*Minimal leaf size*" to something like 10 (instead of default, 2). But doing so, we would also lose the classification influence of all the other parameters, except the root node [try it!]

How does this solution compare to the book solution? There they follow a slightly different route: first they eliminate some of the variables which are deemed unimportant by a feature selection method. With the resulting 10 (out of 17) predictors, they still obtain an accuracy of about 63% using a **CHAID** model. With a **boosting tree** model, they achieve accuracy of 66%.

In addition to assessing the model's performance by static measures such as accuracy, we can also use Gain/Lift charts, Receiver Operator Characteristic (ROC) charts, and Area Under ROC curve (AUC) charts. An explanation of how these charts are constructed and interpreted is available here.

RapidMiner provides AUC chart comparisons: typically when a classifier is unable to distinguish between two classes, the AUC will be closer to 0.5. In this exercise, the AUC ranged from a pessimistic estimate of 0.559 to an optimistic estimate of 0.81, with an average of 0.684.

If you liked this ebook tutorial on analytics, sign up for visTASC, "a visual thesaurus of analytics, statistics and complex systems" for more like these. Sign up is FREE and allows you to search for techniques for other common business problems.

Sign up Now!