CHAPTER THIRTEEN:
EVALUATION AND DEPLOYMENT

HOW FAR WE'VE COME

The purpose of this book, which was explained in Chapter 1, is to introduce non-experts and non-computer scientists to some of the methods and tools of data mining. Certainly there have been a number of processes, tools, operators, data manipulation techniques, etc., demonstrated in this book, but perhaps the most important lesson to take away from this broad treatment of data mining is that the field has become huge, complex, and dynamic. You have learned about the CRISP-DM process, and had it shown to you numerous times as you have seen data mining models that classified, predicted and did both. You have seen a number of data processing tools and techniques, and as you have done this, you have hopefully noticed thy myriad other operators in RapidMiner that we did not use or discuss. Although you may be feeling like you're getting good at data mining (and we hope you do), please recognize that there is a world of data mining that this book has *not* touched on—so there is still much for you to learn.

This chapter and the next will discuss some cautions that should be taken before putting any real-world data mining results into practice. This chapter will demonstrate a method for using RapidMiner to conduct some validation for data mining models; while Chapter 14 will discuss the choices you will make as a data miner, and some ways to guide those choices in good directions. Remember from Chapter 1 that CRISP-DM is cyclical—you should always be learning from the work you are doing, and feeding what you've learned from your work back into your next data mining activity.

For example, suppose you used a Replace Missing Values operator in a data mining model to set all missing values in a data set to the average for each attribute. Suppose further that you used results of that data mining model in making decisions for your company, and that those decisions turned out to be less than ideal. What if you traced those decisions back to your data mining activities and found that by using the average, you made some general assumptions that weren't really very

realistic. Perhaps you don't need to throw out the data mining model entirely, but for the next run of that model you should be sure to change it to either remove observations with missing values, or use a more appropriate replacement value based upon what you have learned. Even if you used your data mining results and had excellent outcomes, remember that your business is constantly moving, and through the day-to-day operations of your organization, you are gathering more data. Be sure to add this data to training data sets, compare actual outcomes to predictions, and tune your data mining models in accordance with your experience and the expertise you are developing. Consider Sarah, our hypothetical sales manager from Chapters 4 and 8. Certainly now that we've helped her predict heating oil usage by home through a linear regression model, Sarah can track these homes' *actual* heating oil orders to see how well their actual use matches our predictions. Once these customers have established several months or years of actual heating oil consumption, their data can be fed into Sarah's model's training data set, helping it to be even more accurate in its predictions.

One of the benefits of connecting RapidMiner to a database or data warehouse, rather than importing data via a file (CSV, etc.) is that data can be added to the data sets in real time and fed straight into the RapidMiner models. If you were to acquire some new training data, as Sarah could in the scenario just proposed in the previous paragraph, it could be immediately incorporated into the RapidMiner model if the data were in a connected database. With a CSV file, the new training data would have to be added into the file, and then re-imported into the RapidMiner repository.

As we tune and hone our models, they perform better for us. In addition to using our growing expertise and adding more training data, there are some built-in ways that we can check a model's performance in RapidMiner.

LEARNING OBJECTIVES

After completing the reading and exercises in this chapter, you should be able to:

- Explain what cross-validation is, and discuss its role in the Evaluation and Deployment phases of CRISP-DM.
- Define false positives and explain why their existence is not all bad in data mining.
- Perform a cross-validation on a training data set in RapidMiner.

- Interpret and discuss the results of cross-validation matrix.

CROSS-VALIDATION

Cross-validation is the process of checking for the likelihood of false positives in predictive models in RapidMiner. Most data mining software products will have operators for cross-validation and for other forms of false positive detection. A **false positive** is when a value is predicted incorrectly. We will give one example here, using the decision tree we built for our hypothetical client Richard, back in Chapter 10. Complete the following steps:

1) Open RapidMiner and start a new, blank data mining process.
2) Go to the Repositories tab and locate the Chapter 10 training data set. This was the one that had attributes regarding peoples' buying habits on Richard's employer's web site, along with their category of eReader adoption. Drag this data set into your main process window. You can rename it if you would like. In Figure 13-1, we have renamed it eReader Train.

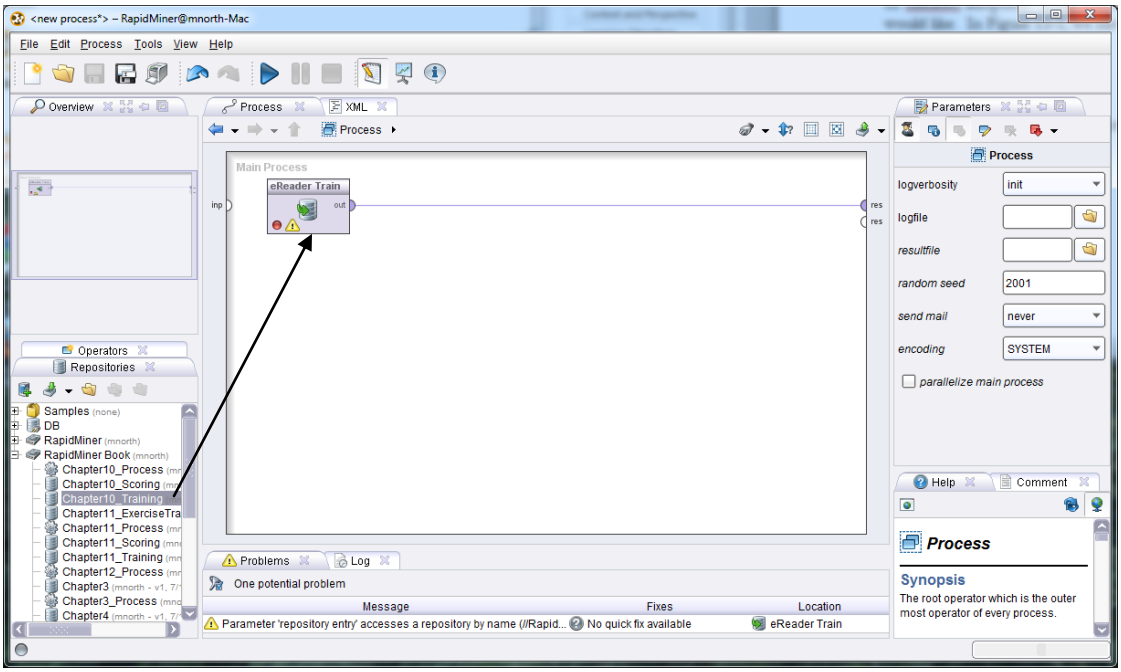


Figure 13-1. Adding the Chapter 10 training data to a new model in order to cross-validate its predictive capabilities.

3) Add a Set Role operator to the stream. We'll learn a new trick here with this operator. Set the User_ID attribute to be 'id'. We know we still need to set eReader_Adoption to be

'label' (the thing we want to predict). Back in Chapter 10, we did this by adding another Set Role operator, but this time, click on the 'set additional roles: Edit List' button in the Parameters area. This is indicated by the black arrow in Figure 13-2.

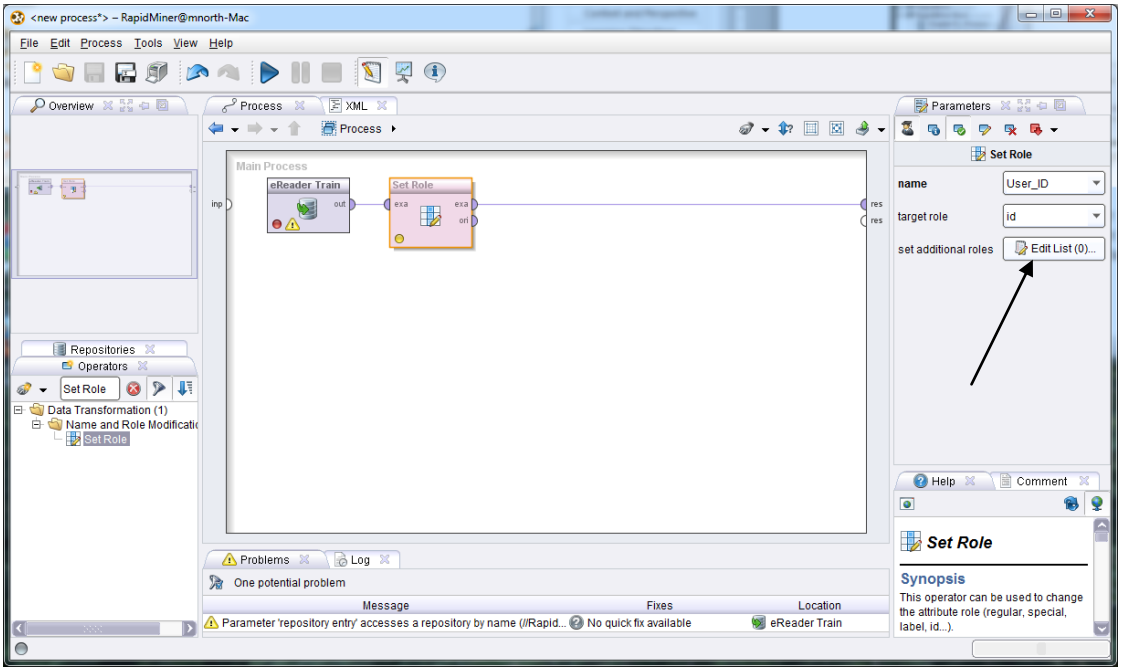


Figure 13-2. Setting multiple roles with a single Set Role operator.

4) In the resulting pop-up window, set the name field to be eReader_Adoption and the target role field to be label. (Note that we could use the Add Entry button to use this single Set Role operator to handle role assignments for many attributes all at once.)

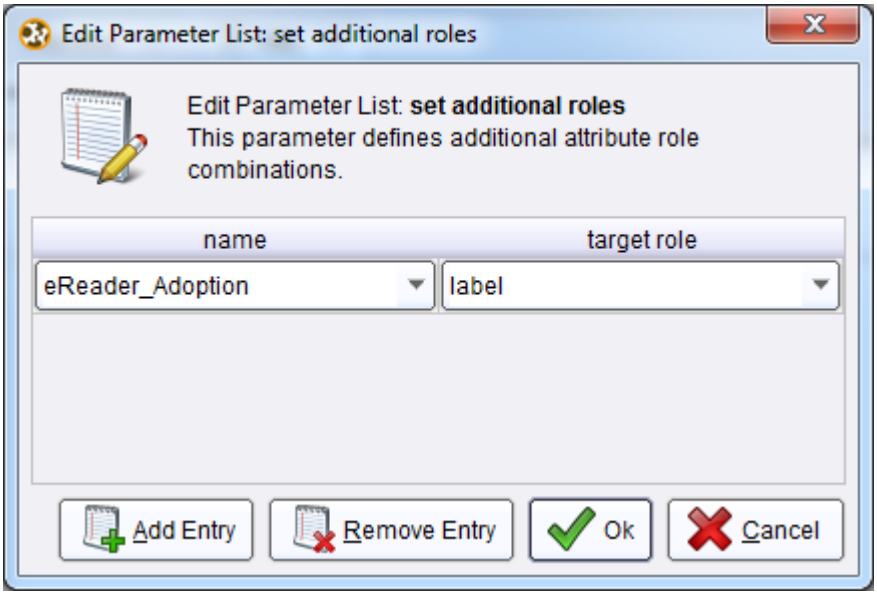


Figure 13-3. Setting additional roles by editing the parameters of a single Set Role operator.

5) When we used this data set previously, we added our Decision Tree operator at this point. This time, we will use the search field in the Operators tab to find x-Validation operators. There are four of them, but we will use the basic cross-validation operator in this example:

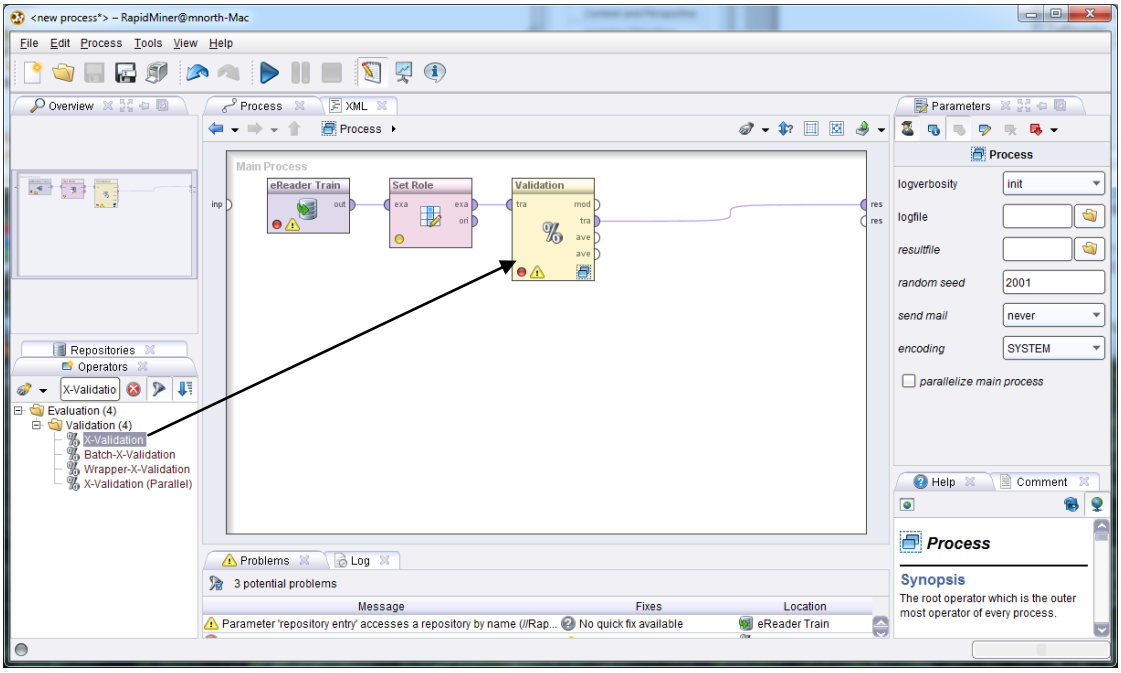


Figure 13-4. Adding a cross-validation operator to our stream.

6) The cross-validation operator requires a two-part sub-process. In the first part of the sub-process, we will add our Decision Tree operator to build a model, and in the second part we will apply our model and check its performance. Double click the Validation operator to enter the sub-process window.

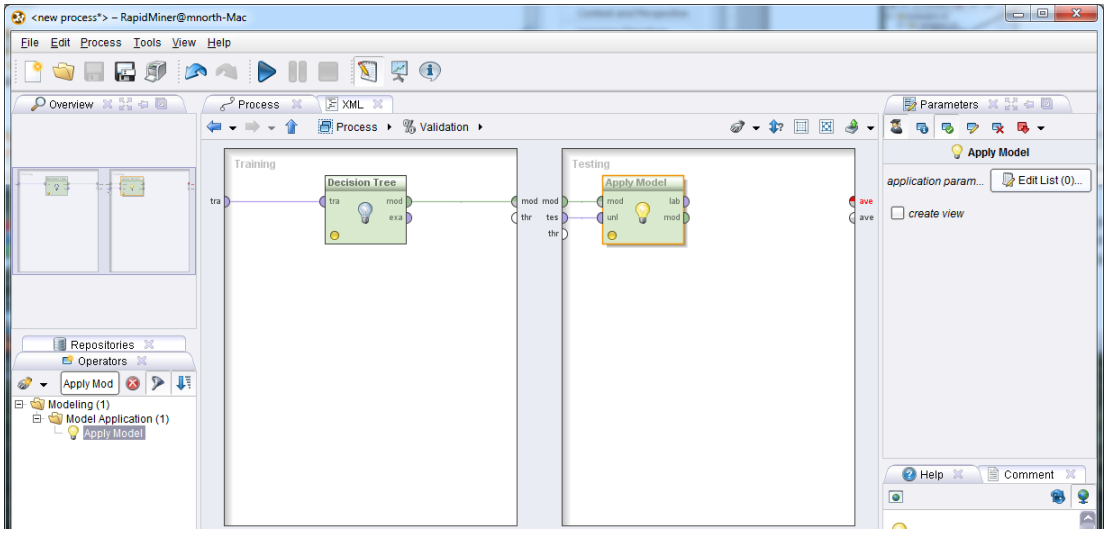


Figure 13-5. Modeling and applying the model in the cross-validation sub-process.

7) In Figure 13-5, add the Decision Tree operator in the Training side of the cross-validation sub-process, and the Apply Model operator on the Testing side. Leave the Decision Tree's operator as gain_ratio for now. The splines depicted here are automatically drawn when you drag these operators into these two areas. If for any reason you do not have these splines configured in this way, connect the ports as shown so that your sub-process matches Figure 13-5. We must now complete the Testing portion of the sub-process. In the Operators search field, search for an operator called 'Performance'. There are a number of these. We will use the first onw: Performance (Classification). The reason for this is that a decision tree predicts a classification in an attribute—in our example, the adopter class (innovator, early adopter, etc.).

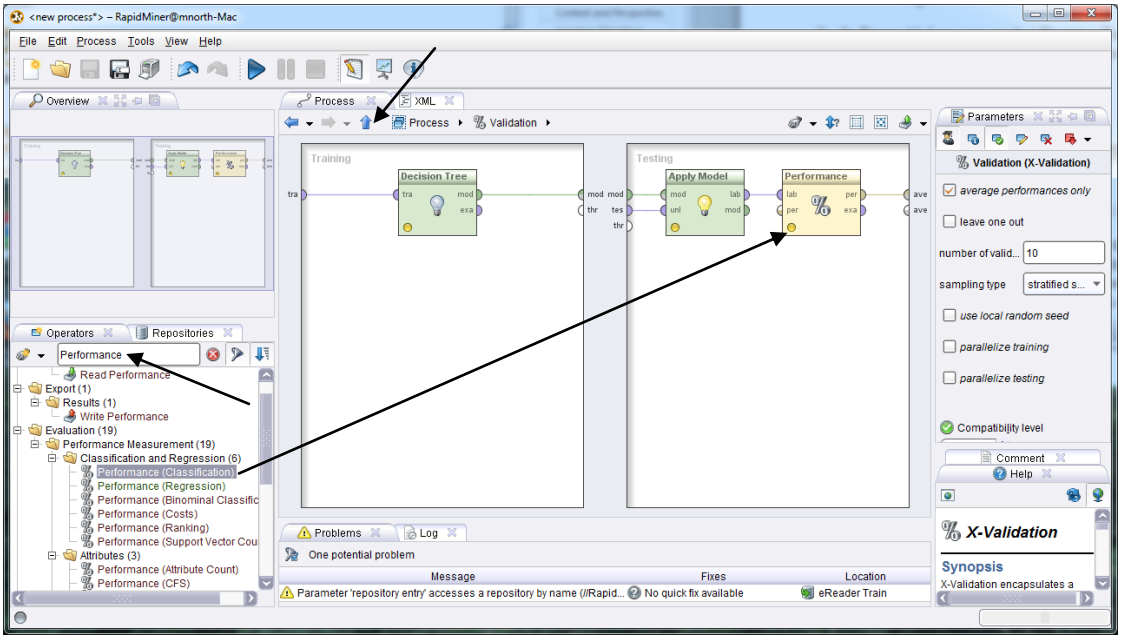


Figure 13-6. The configuration of the cross-validation sub-process.

8) Once your sub-process is configured, click the blue up arrow to return to the main process. Connect the *mod, tra* and *ave* ports to *res* ports as shown in Figure 13-7. The *mod* port will generate the visual depiction of our decision tree, the *tra* port will create the training data set's attribute table, and the *avg* port will calculate a True Positive table showing the training data set's ability to predict accurately.

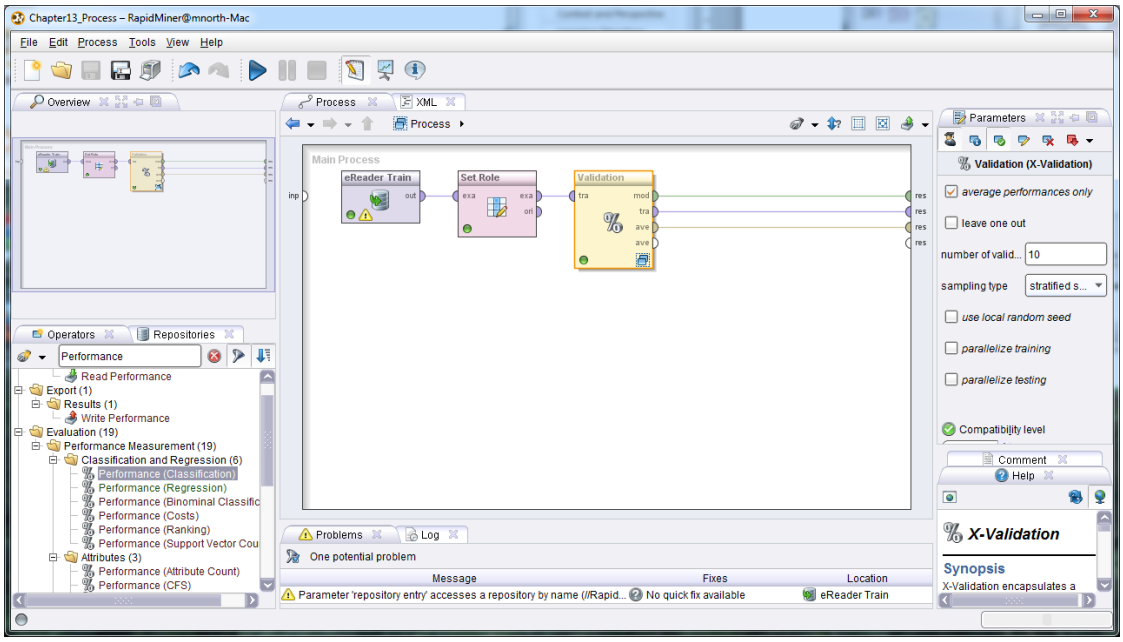


Figure 13-7. Splines to create the three desired outputs from our cross-validated Decision Tree data mining model.

9) Run the model. The ExampleSet (*tra* port) and Tree (*mod* port) tabs will be familiar to you. The PerformanceVector (*avg* port) is new, and in the context of Evaluation and Deployment, this tab is the most interesting to us. We see that using this training data set and Decision Tree algorithm (gain_ratio), RapidMiner calculates a 54% accuracy rate for this model. This overall accuracy rate reflects the class precision rates for each possible value in our eReader_Adoption attribute. For pred. Late Majority as an example, the class precision (or true positive rate) is 69.8%, leaving us with a 30.2% false positive rate for this value. If all of the possible eReader_Adoption values had true positive class precisions of 69.8%, then our model's overall accuracy would be 69.8% as well, but they don't—some are lower, and so when they are weighted and averaged, our model's overall accuracy is only 54%.

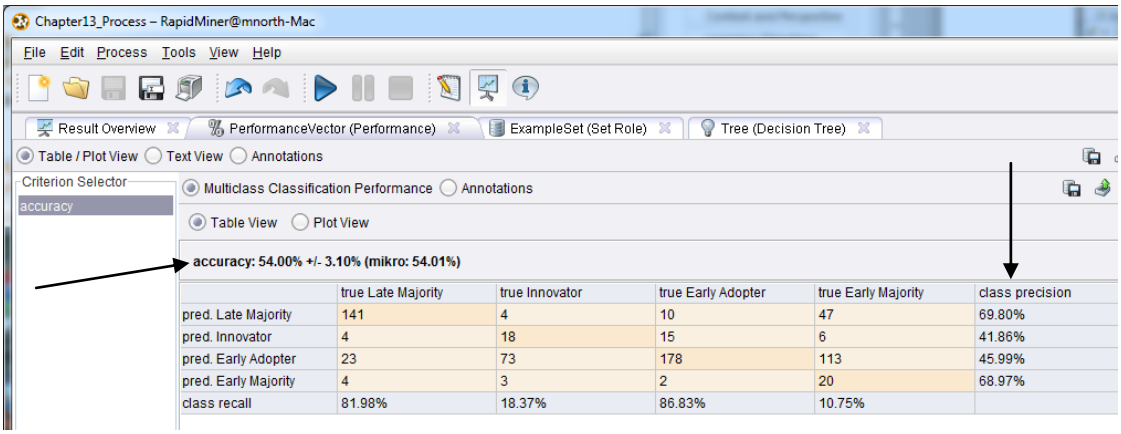


Figure 13-8. Evaluating the predictive quality of our decision tree model.

10) An overall accuracy of 54% might seem alarming, and even individual class precisions in the 40-60% range might seem discouraging, but remember, life is unpredictable, or at least inconsistent, so 100% true positives are probably a pipe dream. The probability of false positives shouldn't even be that surprising to us, because back in Chapter 10, we evaluated our Confidence Percentage attributes, and we knew back then that most of our observations had partial confidences in the predicted value. In Figure 10-10, person 77373 had some chance of landing in any one of three of the four possible adopter categories—of course there is a chance of a false positive! But that doesn't render our model useless, and perhaps we can improve it. Return to design perspective and double click on the Validation operator to re-open the sub-process. Click on the Decision Tree operator to change its criterion parameter to use gini_index as its underlying algorithm.

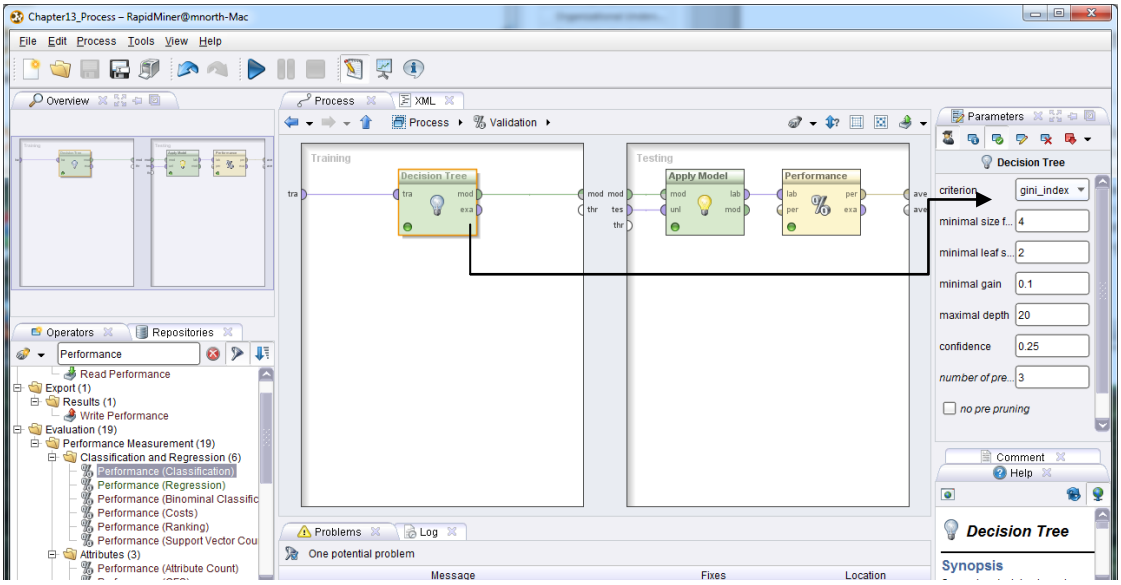
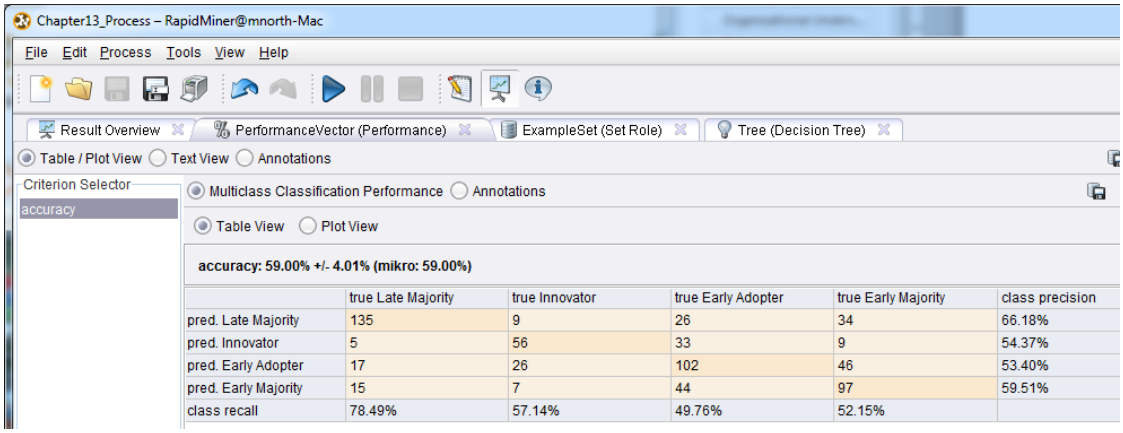


Figure 13-9. Changing the Decision Tree operator to use Gini.

11) Re-run the model. You do not need to switch back to the main process to re-run the model. You may if you wish, but you can stay in sub-process view and run it too. When you return from results perspective to design perspective, you will see whichever design window you were last in. When you re-run the model, you will see a new performance matrix, showing the model's predictive power using Gini as the underlying algorithm.



	true Late Majority	true Innovator	true Early Adopter	true Early Majority	class precision
pred. Late Majority	135	9	26	34	66.18%
pred. Innovator	5	56	33	9	54.37%
pred. Early Adopter	17	26	102	46	53.40%
pred. Early Majority	15	7	44	97	59.51%
class recall	78.49%	57.14%	49.76%	52.15%	

Figure 13-10. New cross-validation performance results based on the gini_index decision tree model.

We see in Figure 13-10 that our model's ability to predict is significantly improved if we use Gini for our decision tree model. This should also not come as a great surprise. We knew from Chapter 10 that the granularity in our tree's detail under Gini was much greater. Greater detail in the predictive tree *should* result in a more reliably predictive model. Feeding more and better training data into the training data set would likely raise this model's reliability even more.

CHAPTER SUMMARY: THE VALUE OF EXPERIENCE

So now we have seen one way to statistically evaluate a model's reliability. You have seen that there are a number of cross-validation and performance operators that you can use to check a training data set's ability to perform. But the bottom line is that there is no substitute for experience and expertise. Use subject matter experts to review your data mining results. Ask them to give you feedback on your model's output. Run pilot tests and use focus groups to try out your model's predictions before rolling them out organization-wide. Do not be offended if someone questions or challenges the reliability of your model's results—be humble enough to take their

questions as an opportunity to validate and strengthen your model. Remember that 'pride goeth before the fall'! Data mining is a process. If you present your data mining results and recommendations as infallible, you are not participating in the cyclical nature of CRISP-DM, and you'll likely end up looking foolish sooner or later. CRISP-DM is such a good process precisely because of its ability to help us investigate data, learn from our investigation, and then do it again from a more informed position. Evaluation and Deployment are the two steps in the process where we establish that more informed position.

REVIEW QUESTIONS

1) What is cross-validation and why should you do it?

2) What is a false positive and why might one be generated?

3) Why would false positives not negate all value for a data mining model?

4) How does a model's overall performance percentage relate to the target attribute's (label's) individual performance percentages?

5) How can changing a data mining methodology's underlying algorithm affect a model's cross-validation performance percentages?

EXERCISE

For this chapter's exercise, you will create a cross-validation model for your Chapter 10 exercise training data set. Complete the following steps.

1) Open RapidMiner to a new, blank process and add the training data set you created for your Chapter 10 exercise (the Titanic survival data set).
2) Set roles as necessary.
3) Apply a cross-validation operator to the data set.
4) Configure your sub-process using gain_ratio for the Decision Tree operator's algorithm. Apply the model and run it through a Performance (Classification) operator.

5) Report your training data set's ability to predict.
6) Change your Decision Tree operator's algorthim to gini_index and re-run your model.
7) Report your results in the context of any changes that occurred in your training data set's ability to predict.

Challenge Step!

8) Change your Decision Tree operator's algorithm to one of the other options, such as information_schema, and report your results again, comparative to gain_ratio and gini_index.

Extra Challenge Step!

9) Repeat steps 1-7 for the linear regression training data set (Chapter 8). You will need to use a slightly different Performance operator. Report your results. If you would like, repeat step 8 for your Chapter 8 exercise training data set and report your results.