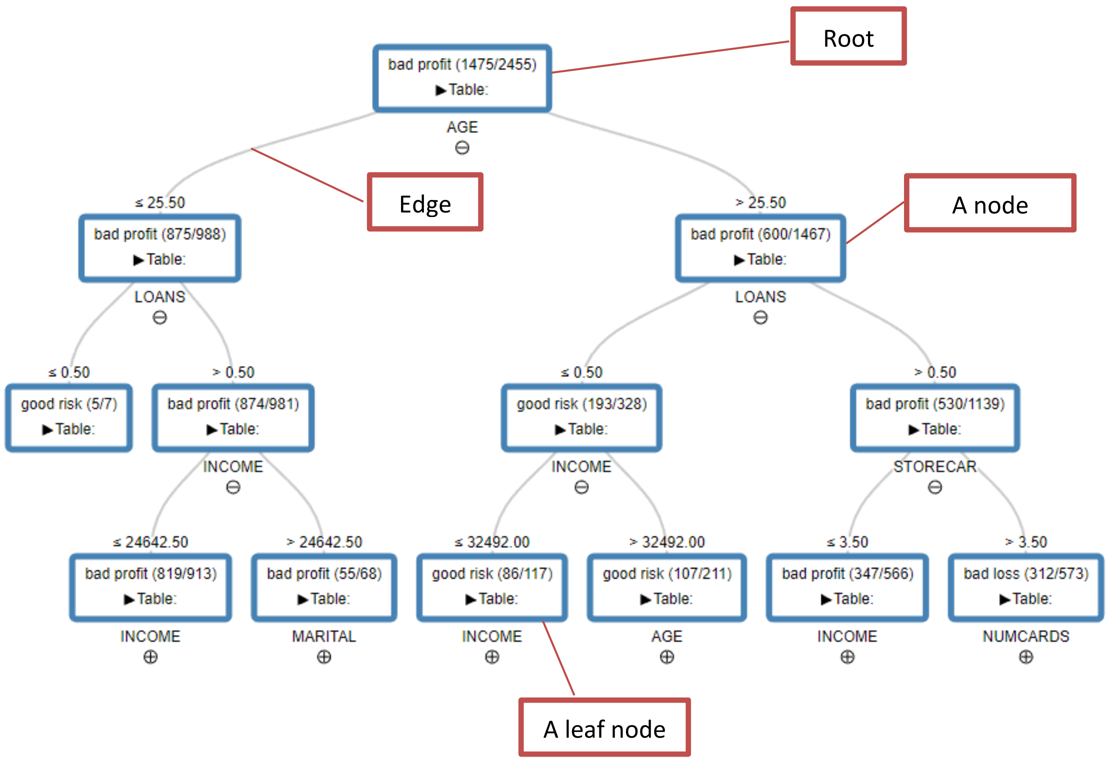
Implementing a Decision Tree using KNIME

In this section, we will use the KNIME Decision Tree node to see how to classify customers with credit cards into multiple classes. This will help us predict if a particular customer is of high or low risk in defaulting on his or her credit card loan.

Description

Referring to the figure below, a decision tree is a tree-like graph or model usually shown inverted. The root of the tree is shown at the top while the nodes or leaves are at the bottom. As in most of the cases for modelling, we require an input set of data to construct a model. In this case, we are constructing a tree model. Once the tree model is constructed, we can subsequently use it to predict the class or category when we are given an input record with the specified input attributes.



In a decision tree, each node of the tree corresponds to one of the input attributes. The number of edges of a nominal interior node is equal to the number of possible values of the corresponding input attribute. Outgoing edges of numerical attributes are labelled with disjoint ranges (e.g. > 0.5).

Each leaf node represents a value of the label attribute and the value of the label attribute is dependent on the values of the input attributes. In other words, the values of the input attributes will determine the path we travel from the root to the leaf of the tree.

Data

To create the tree model, we will be using data contained in the file *datasets/risktrain.csv*. The data comes from a risk assessment study in which customers with credit cards were assigned to one of three categories (class) in the risk classification attribute:

- Good risk (Potentially profitable with low probability of default)
- Bad risk (There are some payments missed or other problems, but were profitable for the issuing company)
- Bad risk (The loan is too risky and have a high probability of default)

In addition to the risk classification attribute, a number of demographics data, including age, income, number of children, number of credit cards, number of store credit cards, having a mortgage, and marital status, were included with a total of about 2,500 examples.

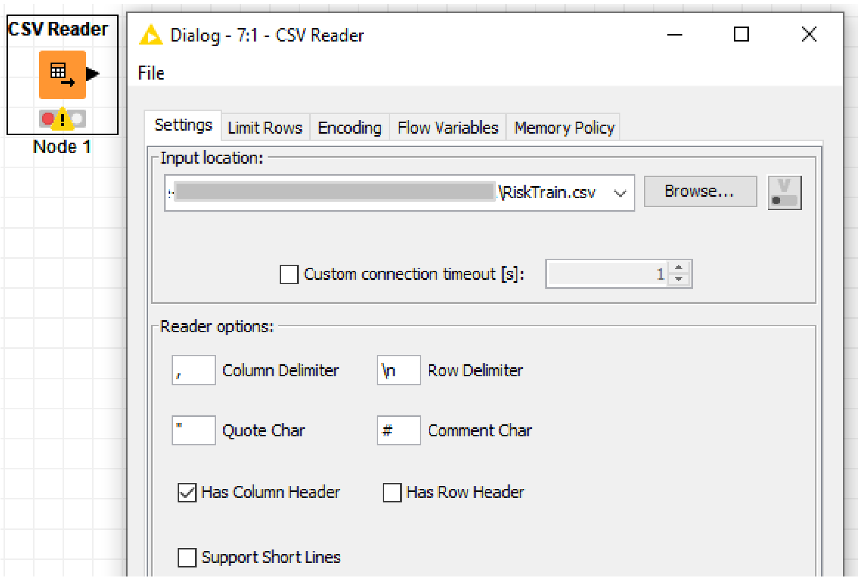
Method

We will be using KNIME (https://www.knime.com/), which stands for Konstanz Information Miner. KNIME is a free and open-source data analytics, reporting and integration platform. KNIME integrates various components for machine learning and data mining through its modular data pipelining "Building Blocks of Analytics" concept . A graphical user interface and use of JDBC allows assembly of nodes blending different data sources, including preprocessing (ETL: Extraction, Transformation, Loading), for modeling, data analysis and visualization without, or with only minimal, programming.

Building the model

Follow the steps listed below to see how to construct a decision tree based on the data from the file *risktrain.csv*.

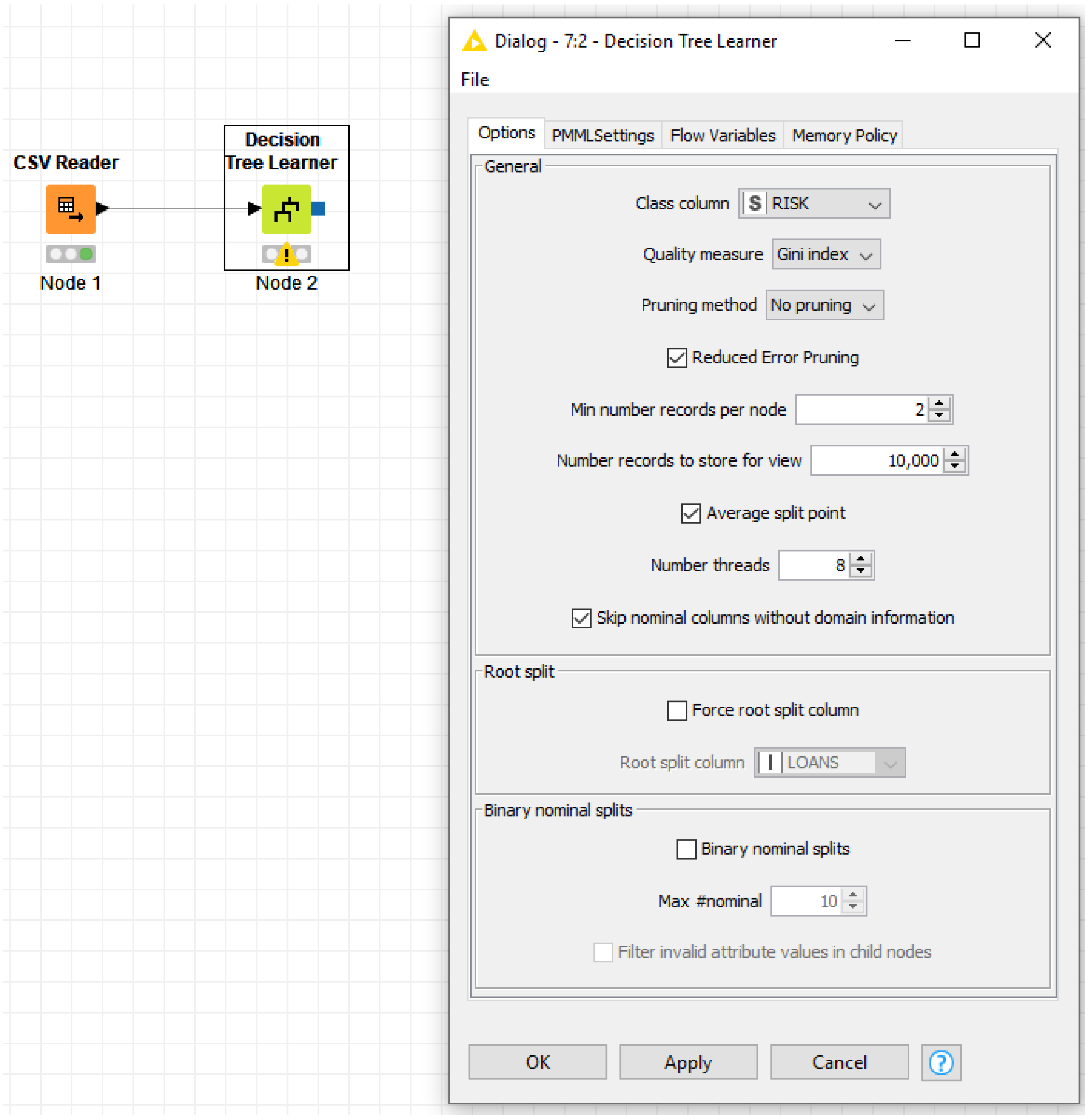
1. Use a *CSV Reader* node to read in the risktrain.csv file. Remember to uncheck the "Has Row Header" checkbox.



Input location:
\RiskTrain.csv
☐ Custom connection timeout [s]: 1

Reader options:
, Column Delimiter
\n Row Delimiter
" Quote Char
Comment Char
☑ Has Column Header
☐ Has Row Header
☐ Support Short Lines

2. Insert a *Decision Tree Learner* node and connect the nodes as shown in the figure below. Configure the decision tree learner as shown also. Once done, execute the learner:



General

Class column: S RISK

Quality measure: Gini index

Pruning method: No pruning

☑ Reduced Error Pruning

Min number records per node: 2

Number records to store for view: 10,000

☑ Average split point

Number threads: 8

☑ Skip nominal columns without domain information

Root split

☐ Force root split column

Root split column: I LOANS

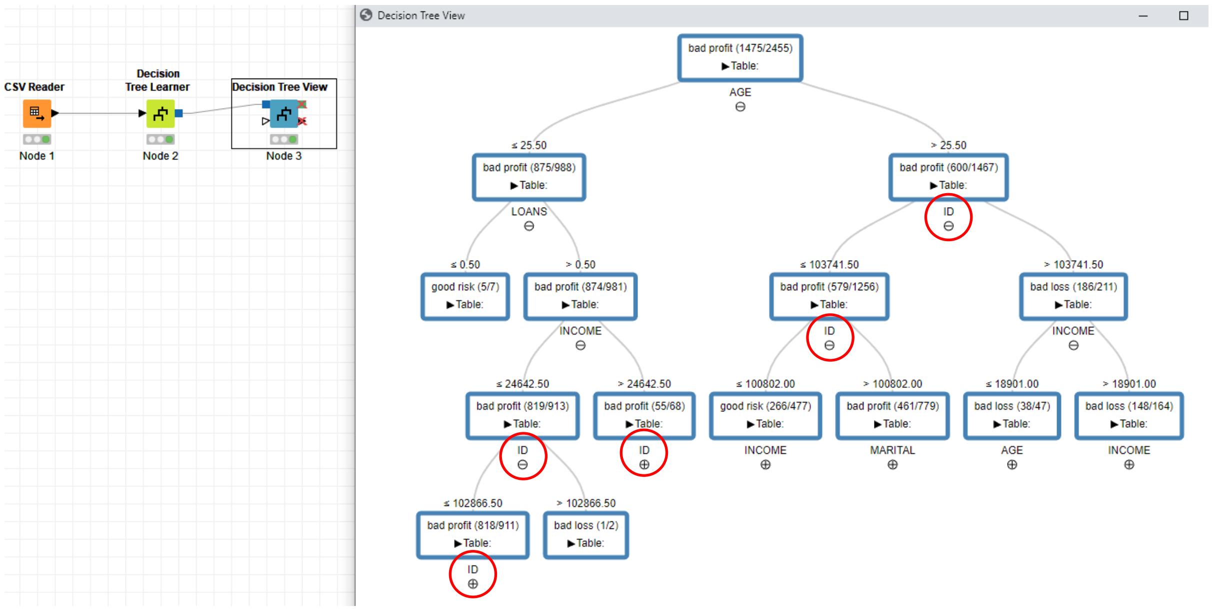
Binary nominal splits

☐ Binary nominal splits

Max #nominal: 10

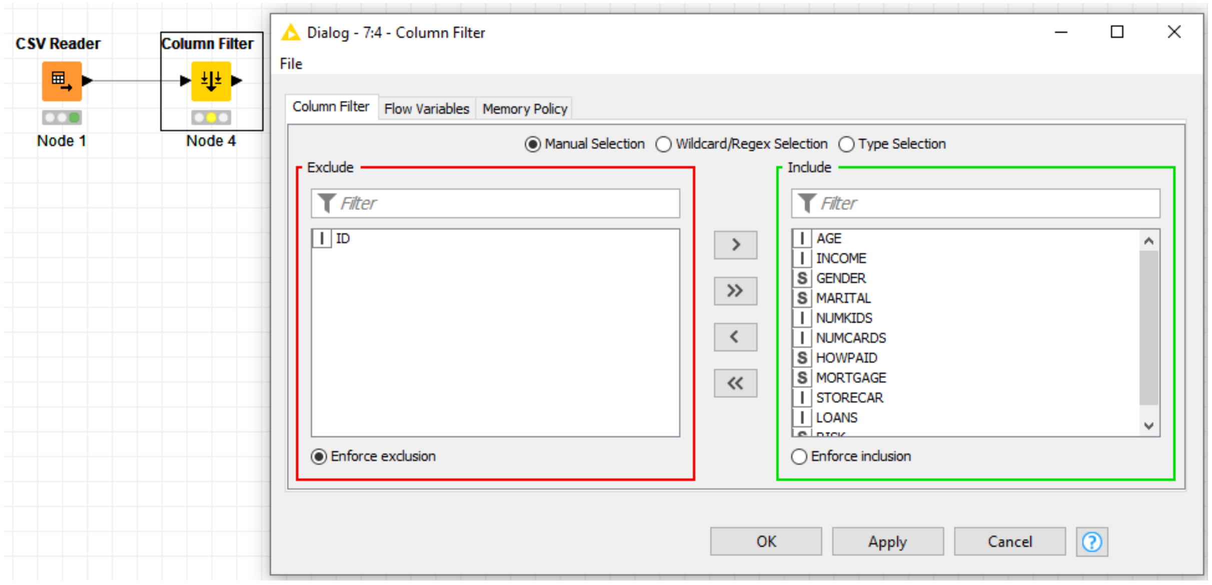
☐ Filter invalid attribute values in child nodes

3. To view the model, we bring in a *Decision Tree View* node.



Notice that the result appears incorrect, the decision along the tree path are based on the IDs of the rows of data. This is obviously useless to us as IDs for each example is unique and possibly assigned in a random manner and should not have any correlation with the risk assessment!

4. We can use a *Column Filter* node remove the ID attribute. You should connect the nodes such that it is between the *CSV Reader* and the *Decision Tree Learner* node.



[✓ Manual Selection] Wildcard/Regex Selection Type Selection

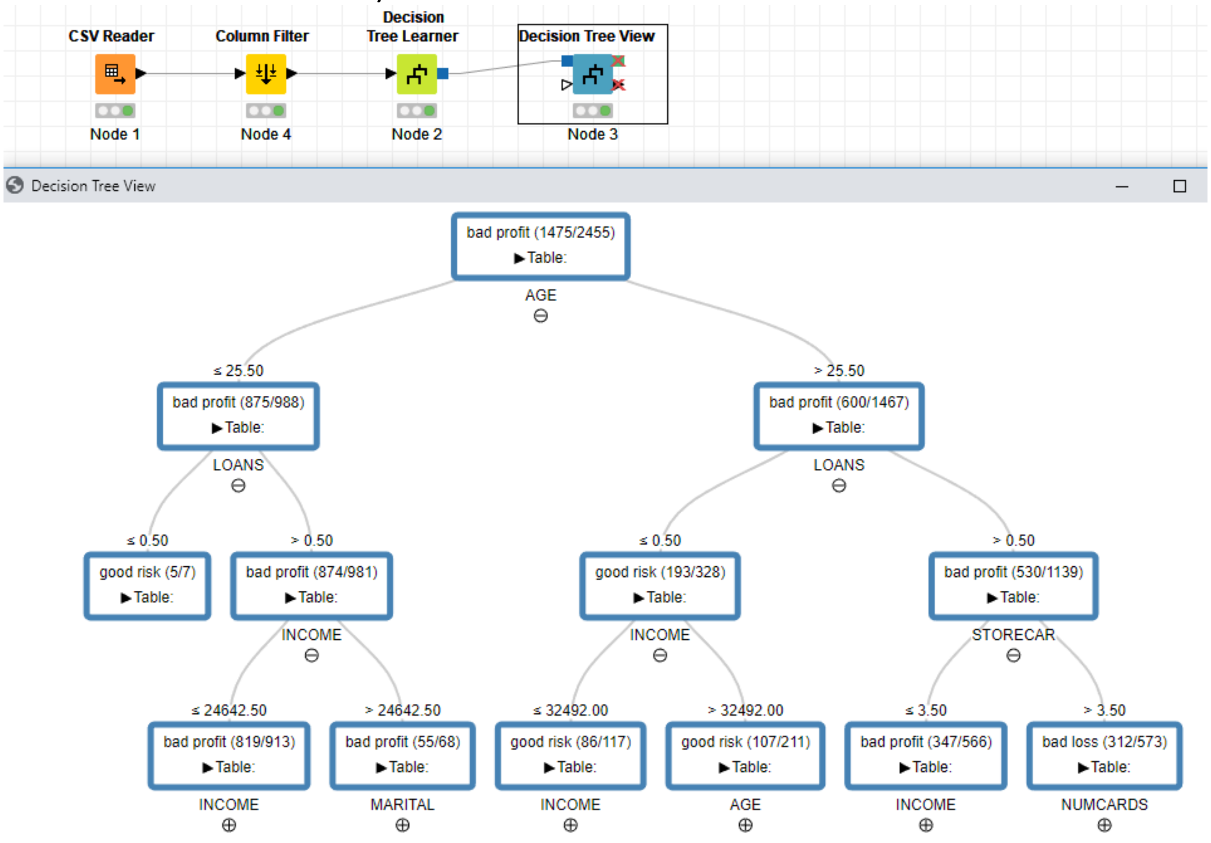
Exclude: ID

[✓ Enforce exclusion]

Include: AGE, INCOME, GENDER, MARITAL, NUMKIDS, NUMCARDS, HOWPAID, MORTGAGE, STORECAR, LOANS, RISK

Enforce inclusion

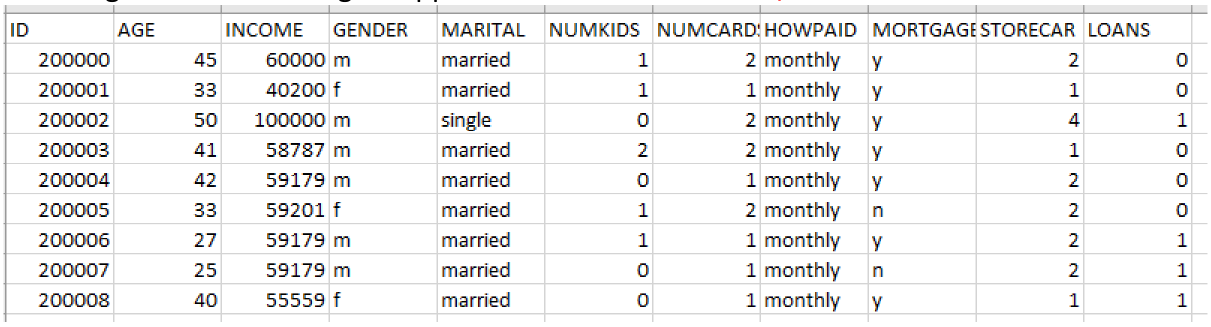
5. Rerun the workflow and you should see the results as follows:



The figure provides a good overview of the tree model generated from the input data. To predict a customer's risk assessment category, we travel down the tree and make a decision at each of the node based on the various attribute values of the customer. We stop when we reach the leaf node which will tell us the predicted category of the risk (good risk, bad loss or bad profit).

<u>Applying model to new data</u>

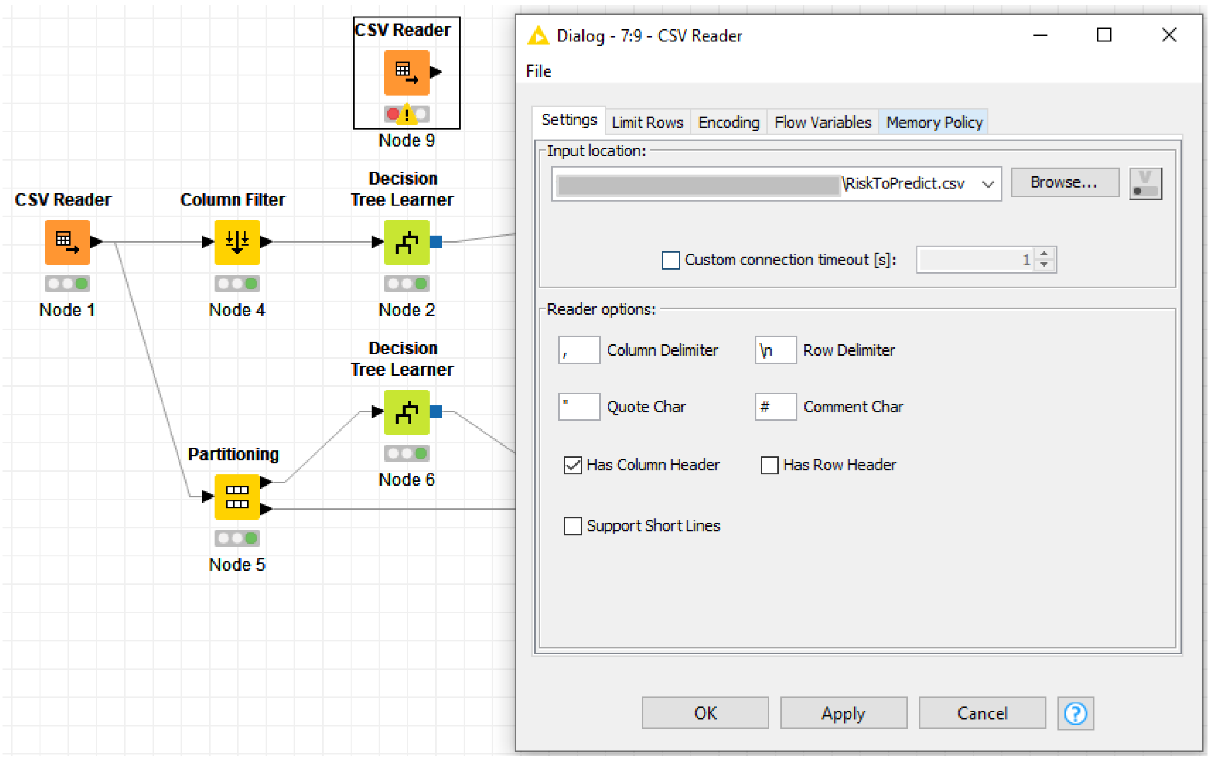
We are given the following 10 applicants in the file *datasets/RiskToPredict.csv*:



ID	AGE	INCOME	GENDER	MARITAL	NUMKIDS	NUMCARD	HOWPAID	MORTGAGE	STORECAR	LOANS
200000	45	60000	m	married	1	2	monthly	y	2	0
200001	33	40200	f	married	1	1	monthly	y	1	0
200002	50	100000	m	single	0	2	monthly	y	4	1
200003	41	58787	m	married	2	2	monthly	y	1	0
200004	42	59179	m	married	0	1	monthly	y	2	0
200005	33	59201	f	married	1	2	monthly	n	2	0
200006	27	59179	m	married	1	1	monthly	y	2	1
200007	25	59179	m	married	0	1	monthly	n	2	1
200008	40	55559	f	married	0	1	monthly	y	1	1

Let's explore how to decide if the bank is asked to approve or reject further loans from the customer. We will load the data into KNIME and use the *Decision Tree Learner* that we have just generated to help us decide which of the customers we should approve or reject the loans.

a. Read in data:



Dialog - 7:9 - CSV Reader

File

Settings | Limit Rows | Encoding | Flow Variables | Memory Policy

Input location:
[redacted]\RiskToPredict.csv   Browse...

Custom connection timeout [s]: 1

Reader options:

, Column Delimiter   \n Row Delimiter

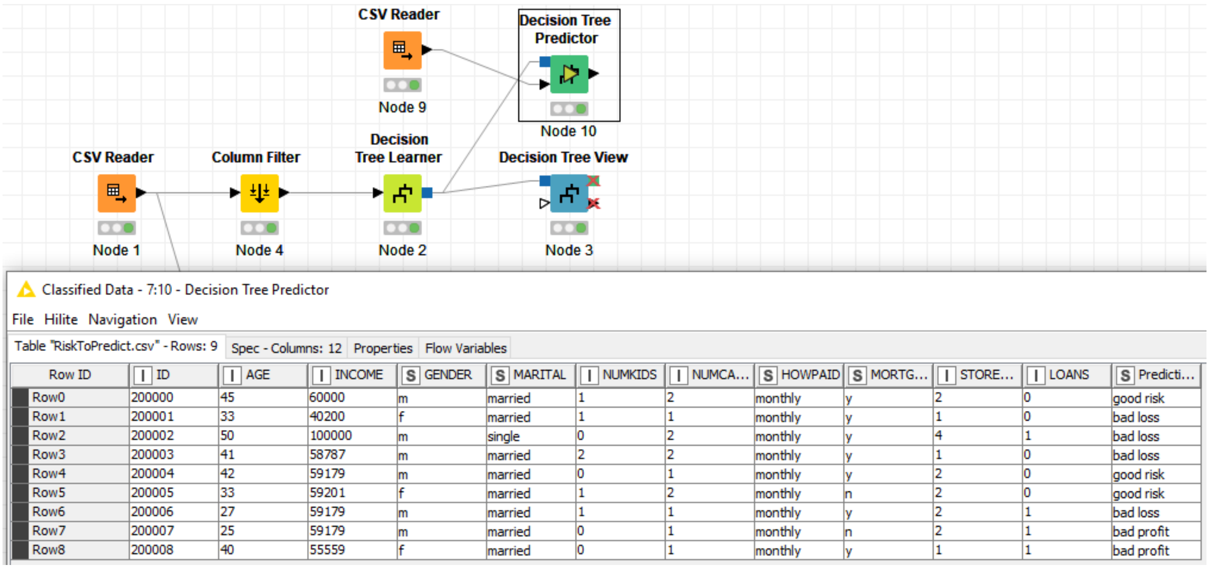
" Quote Char   # Comment Char

☑ Has Column Header   ☐ Has Row Header

☐ Support Short Lines

OK | Apply | Cancel

b. Use the model trained to predict the classification of the 10 rows of data:



Classified Data - 7:10 - Decision Tree Predictor

File Hilite Navigation View

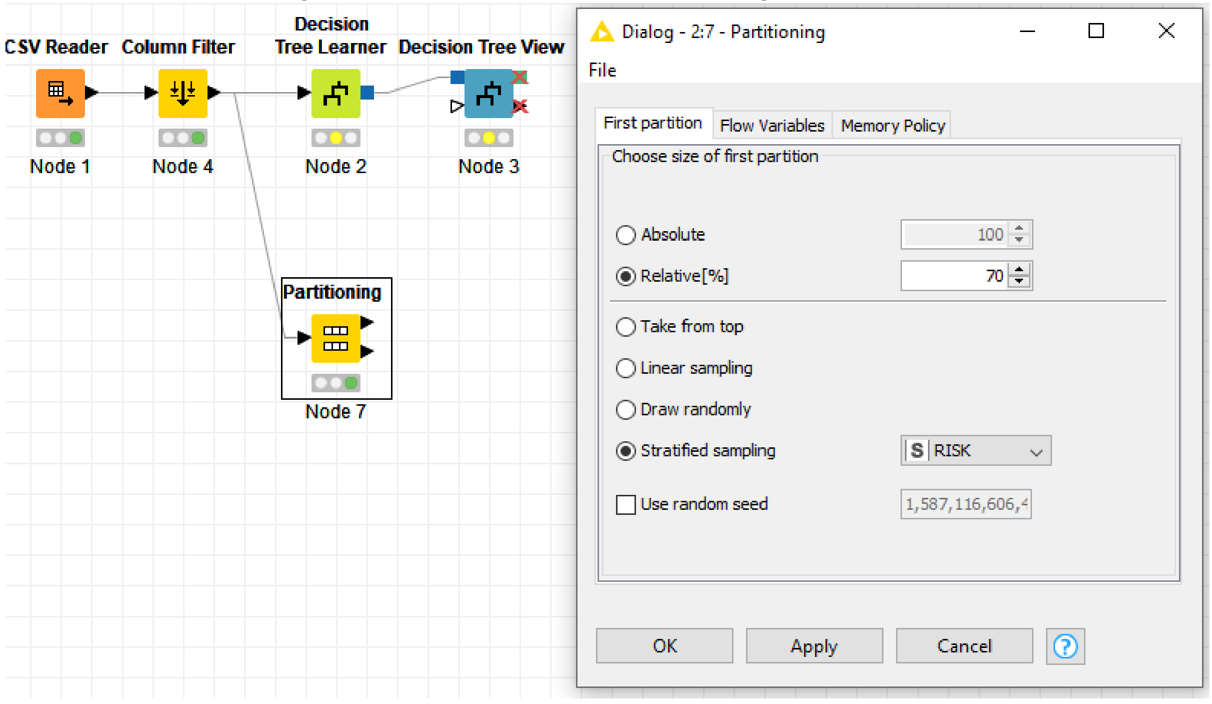
Table "RiskToPredict.csv" - Rows: 9 | Spec - Columns: 12 | Properties | Flow Variables

Row ID	I ID	I AGE	I INCOME	S GENDER	S MARITAL	I NUMKIDS	I NUMCA...	S HOWPAID	S MORTG...	I STORE...	I LOANS	S Predicti...
Row0	200000	45	60000	m	married	1	2	monthly	y	2	0	good risk
Row1	200001	33	40200	f	married	1	1	monthly	y	1	0	bad loss
Row2	200002	50	100000	m	single	0	2	monthly	y	4	1	bad loss
Row3	200003	41	58787	m	married	2	2	monthly	y	1	0	bad loss
Row4	200004	42	59179	m	married	0	1	monthly	y	2	0	good risk
Row5	200005	33	59201	f	married	1	2	monthly	n	2	0	good risk
Row6	200006	27	59179	m	married	1	1	monthly	y	2	1	bad loss
Row7	200007	25	59179	m	married	0	1	monthly	n	2	1	bad profit
Row8	200008	40	55559	f	married	0	1	monthly	y	1	1	bad profit

Split Validation

The current workflow does not tell us how accurate the classification performed by the Decision Tree Learner is. We can use Split Validation to test the accuracy of the model.

6. Introduce a *Partitioning* node into the workflow and configure it as such:



Dialog - 2:7 - Partitioning

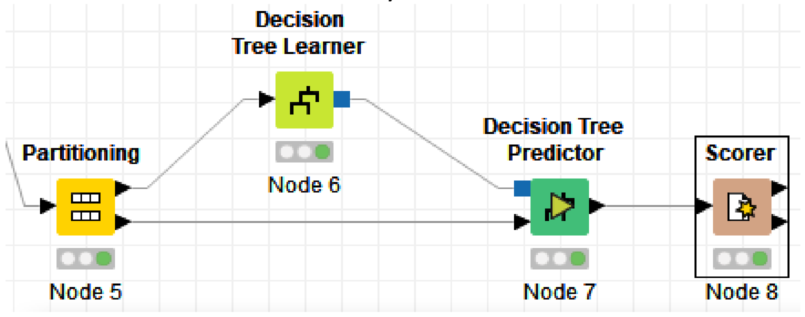
File

First partition | Flow Variables | Memory Policy

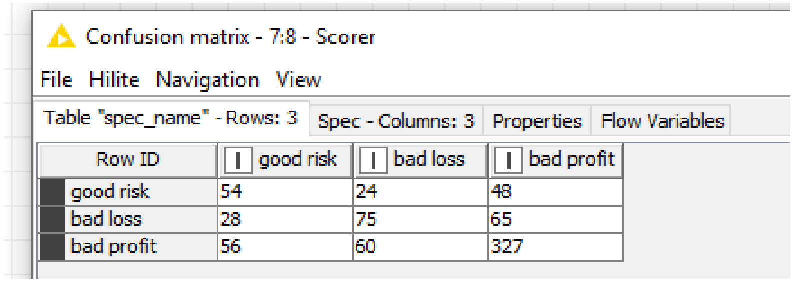
Choose size of first partition

- Absolute: 100
- [✓ Relative[%]]: 70
- Take from top
- Linear sampling
- Draw randomly
- [✓ Stratified sampling]: S RISK
- Use random seed: 1,587,116,606,4

7. Connect the top output (the training set) of the *Partitioning* node to another similarly configured *Decision Tree Learner*, and use the bottom output (the test set) of the Partitioning node for prediction using a *Decision Tree Predictor*. Finally, use a *Scorer* node to measure the accuracy.



8. View the confusion matrix and accuracy statistics at the *Scorer* node:

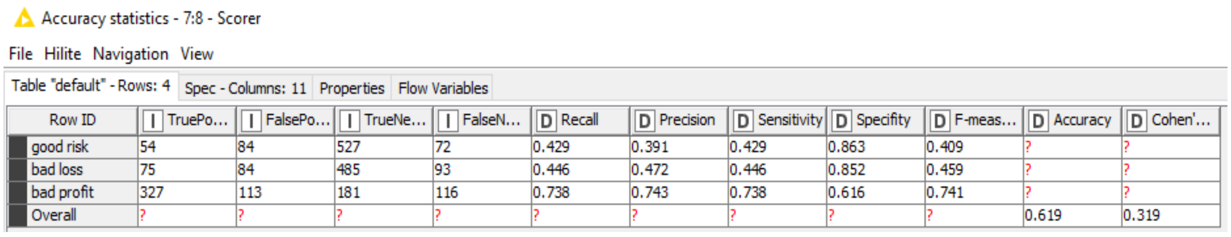


Confusion matrix - 7:8 - Scorer

File Hilite Navigation View

Table "spec_name" - Rows: 3 | Spec - Columns: 3 | Properties | Flow Variables

Row ID	good risk	bad loss	bad profit
good risk	54	24	48
bad loss	28	75	65
bad profit	56	60	327



Accuracy statistics - 7:8 - Scorer

File Hilite Navigation View

Table "default" - Rows: 4 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifty	F-meas...	Accuracy	Cohen'...
good risk	54	84	527	72	0.429	0.391	0.429	0.863	0.409	?	?
bad loss	75	84	485	93	0.446	0.472	0.446	0.852	0.459	?	?
bad profit	327	113	181	116	0.738	0.743	0.738	0.616	0.741	?	?
Overall	?	?	?	?	?	?	?	?	?	0.619	0.319

We can see that the model is more accurate in predicting "bad profit" as there are simply more data for that category. To increase the performance of the decision tree model, we can either get more data for the other categories, or do a balanced sampling before performing the predictive modelling. Alternatively, we can tune the parameters of the *Decision Tree Learner*.

Parameter Optimization

9. Configure the Decision Tree Learner after the Partitioning node to
   - Use MDL for "Pruning method" instead of switching pruning off.

     Note: Pruning reduces tree size and avoids overfitting which increases the generalization performance, and thus, the prediction quality (for predictions, use the "Decision Tree Predictor" node). Available is the "Minimal Description Length" (MDL) pruning or it can also be switched off.

   - Set "Min number records per node" as 10 instead of 2.

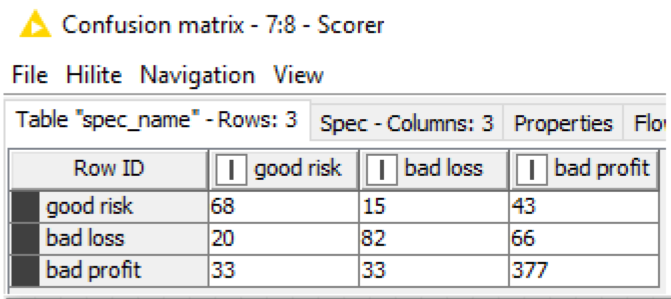
     Note: To select the minimum number of records at least required in each node. If the number of records is smaller or equal to this number the tree is not grown any further. This corresponds to a stopping criteria (pre pruning).

   - Check "Binary nominal splits".

     Note 1: If checked, nominal attributes are split in a binary fashion. Binary splits are more difficult to calculate but result also in more accurate trees. The nominal values are divided in two subsets (one for each child). If unchecked, for each nominal value one child is created.

     Note 2: Regarding the parameter "**Max #nominal**": The subsets for the binary nominal splits are difficult to calculate. To find the best subsets for n nominal values 2^n calculations need to be performed. In case of many different nominal values this can be prohibitively expensive. Thus, the maximum number of nominal values can be defined for which all possible subsets are calculated. Above this threshold, a heuristic is applied that first calculates the best nominal value for the second partition, then the second best value, and so on; until no improvement can be achieved.

10. Re-execute the workflow and have a look at the performance of the model at the *Scorer* node:

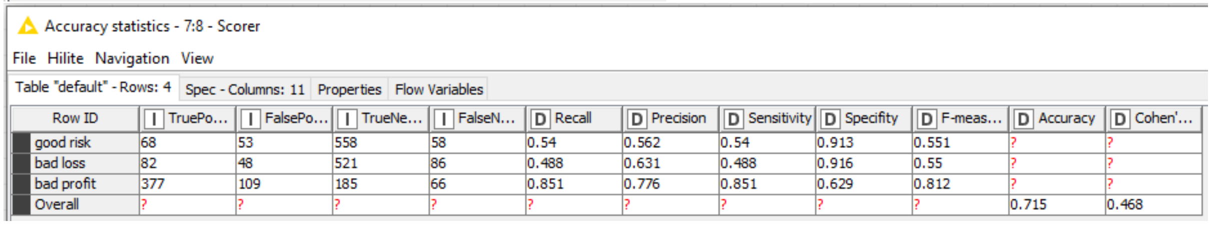


Confusion matrix - 7:8 - Scorer

File Hilite Navigation View

Table "spec_name" - Rows: 3 | Spec - Columns: 3 | Properties | Flo

Row ID	good risk	bad loss	bad profit
good risk	68	15	43
bad loss	20	82	66
bad profit	33	33	377



Accuracy statistics - 7:8 - Scorer

File Hilite Navigation View

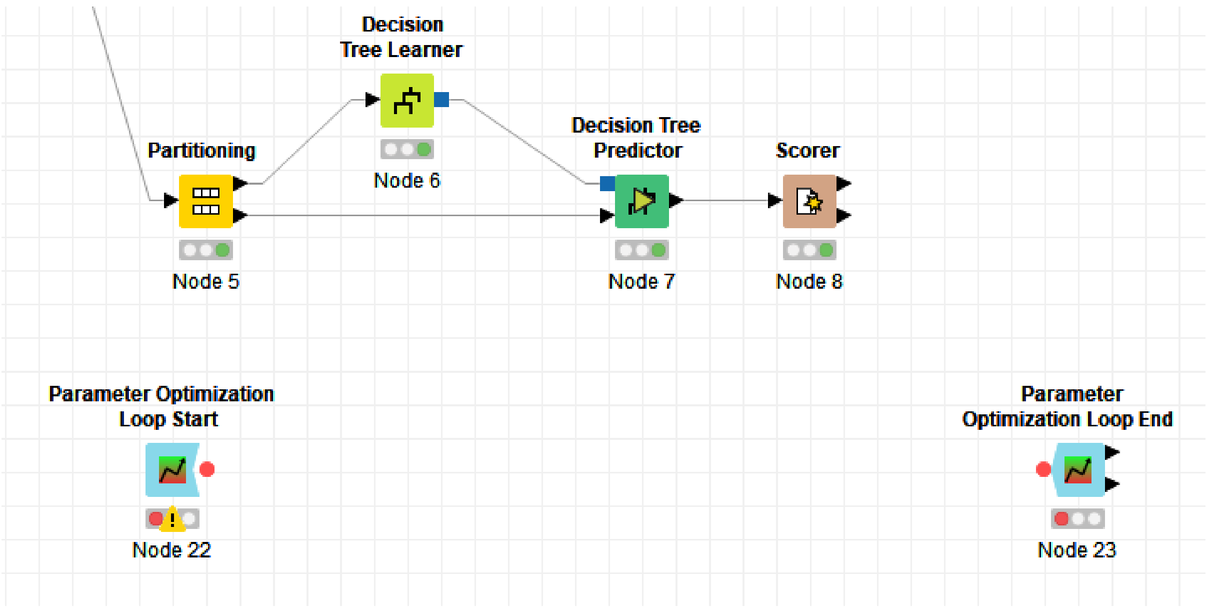
Table "default" - Rows: 4 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
good risk	68	53	558	58	0.54	0.562	0.54	0.913	0.551	?	?
bad loss	82	48	521	86	0.488	0.631	0.488	0.916	0.55	?	?
bad profit	377	109	185	66	0.851	0.776	0.851	0.629	0.812	?	?
Overall	?	?	?	?	?	?	?	?	?	0.715	0.468

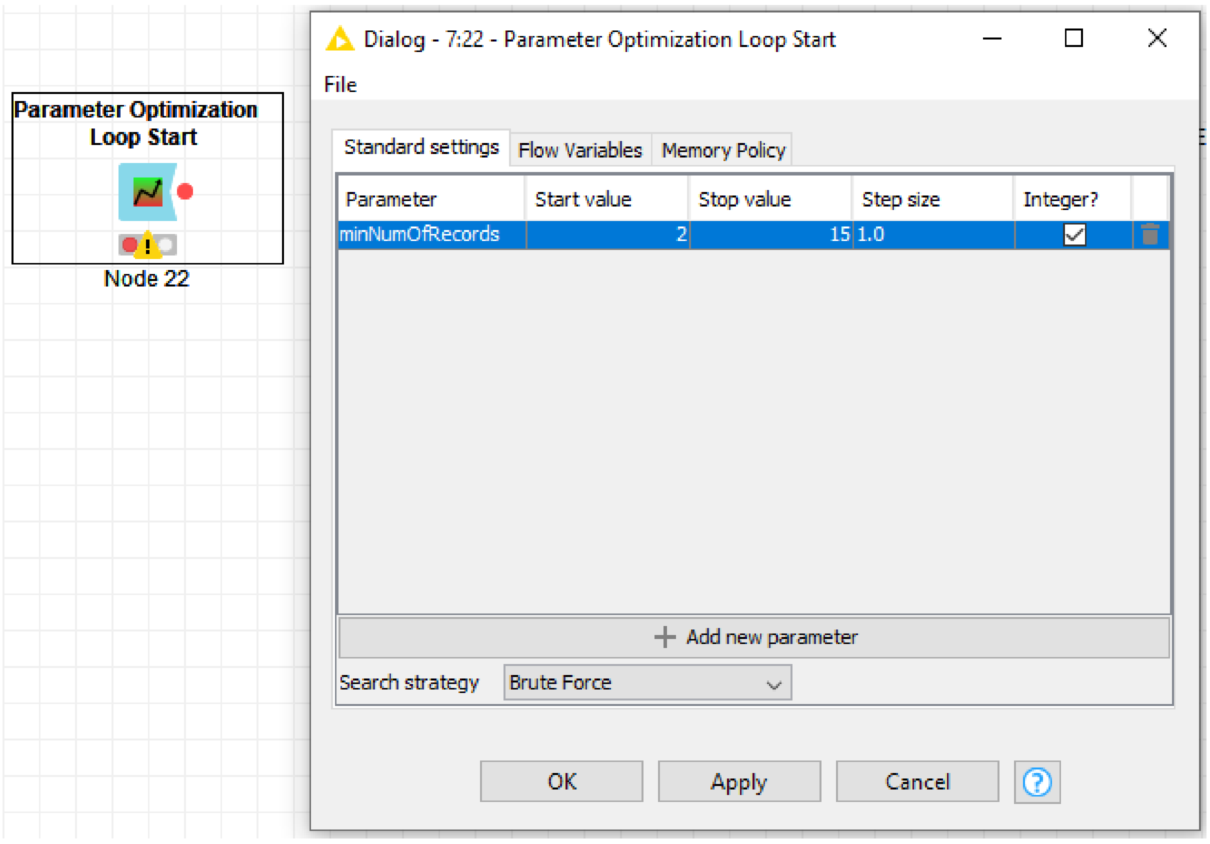
We can see that by tuning certain parameters, the overall accuracy has increased from 61.9% to 71.5%, and that the recall for all the three categories of classification has increased.

The parameter "Min number records per node" was chosen as 10 arbitrarily. Could other values improve the model performance? Instead of changing numerical parameters manually, KNIME provides a pair of *Parameter Optimization Loop* nodes that we can use to automate the changing of parameters and re-executing the models.

11. For a start, put the pair of nodes in the workflow as shown:



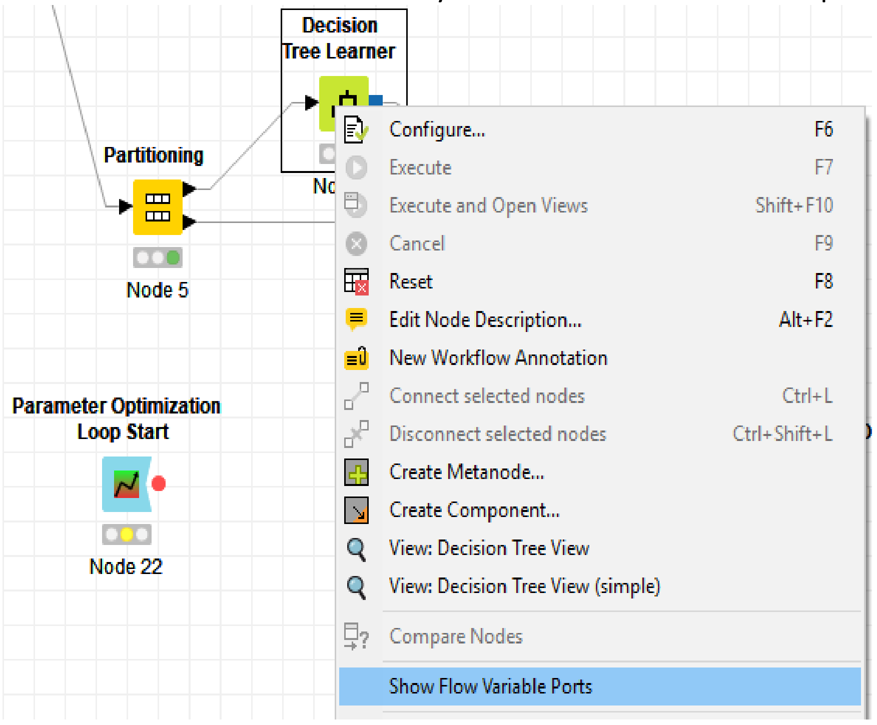
12. Configure the Parameter Optimization Loop Start node by indicating what parameter (i.e. *minNumOfRecords*) we wish to tune:



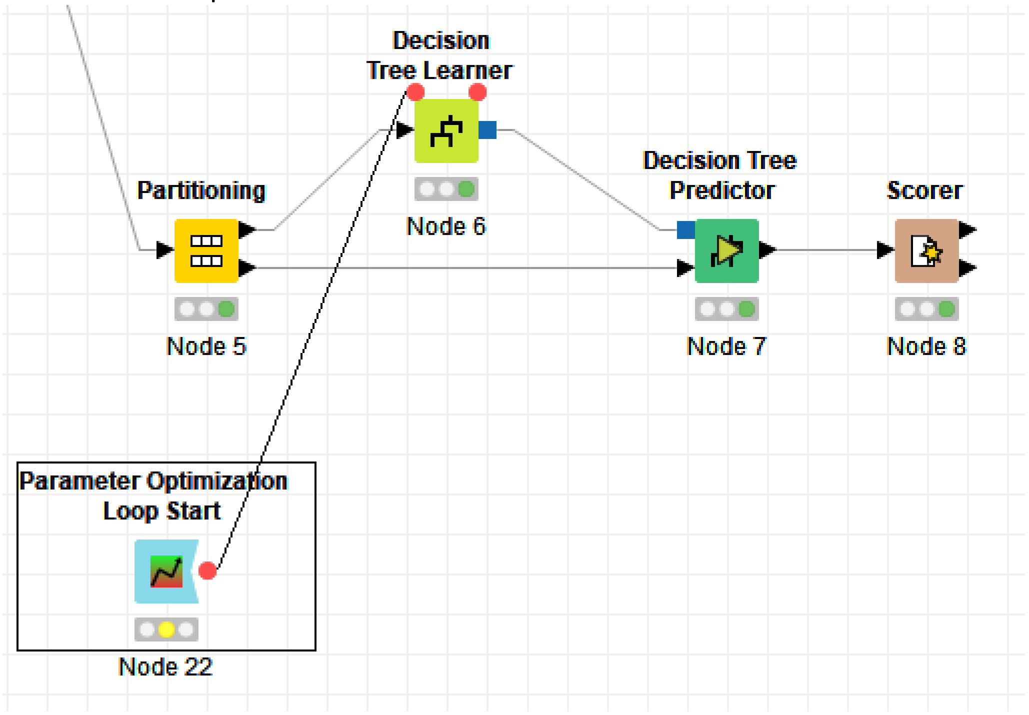
Parameter	Start value	Stop value	Step size	Integer?
minNumOfRecords	2	15	1.0	✓

Search strategy: Brute Force

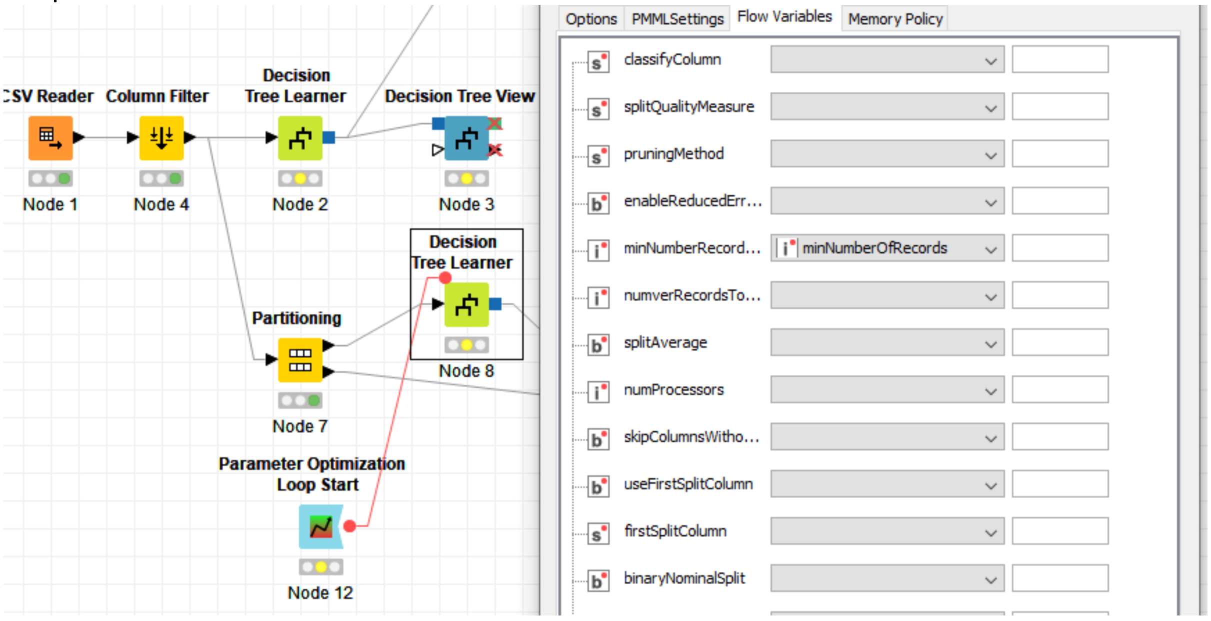
13. You may notice that the loop start node has a red output port. These are flow variable ports, and they have to be connected to other flow variable ports. Right-click on the *Decision Tree Learner* so that you can see the flow variable ports.



14. Connect the loop start node to the Decision Tree Learner node as such:

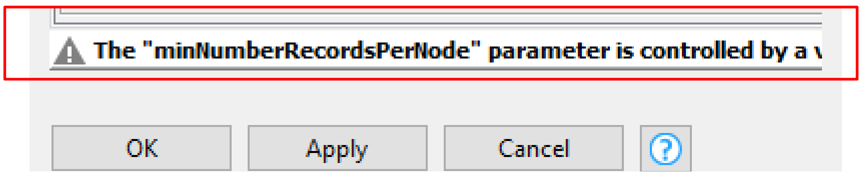


15. Next, we need to configure the Decision Tree Learner so that it understands which parameter is to be tuned:



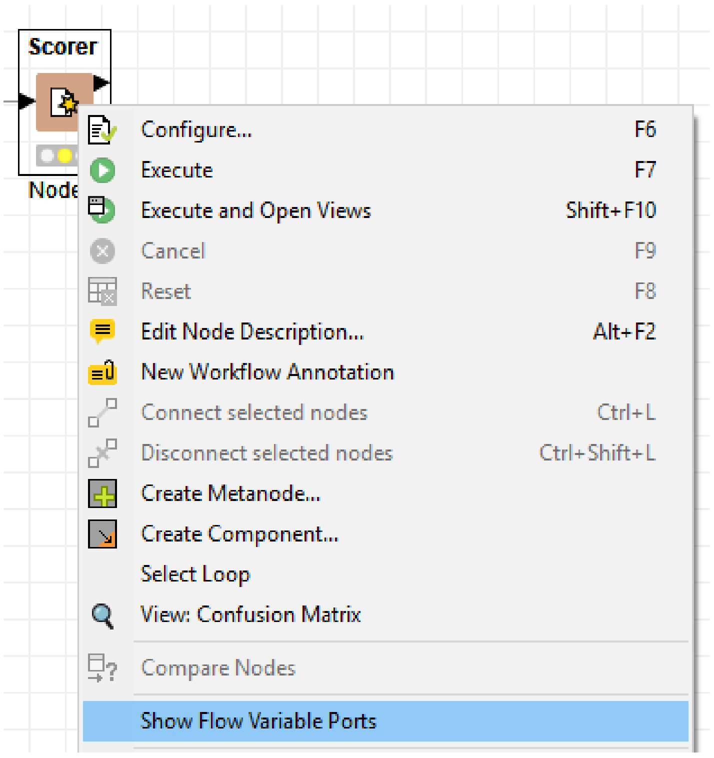
classifyColumn	
splitQualityMeasure	
pruningMethod	
enableReducedErr...	
minNumberRecord...	minNumberOfRecords
numverRecordsTo...	
splitAverage	
numProcessors	
skipColumnsWitho...	
useFirstSplitColumn	
firstSplitColumn	
binaryNominalSplit	

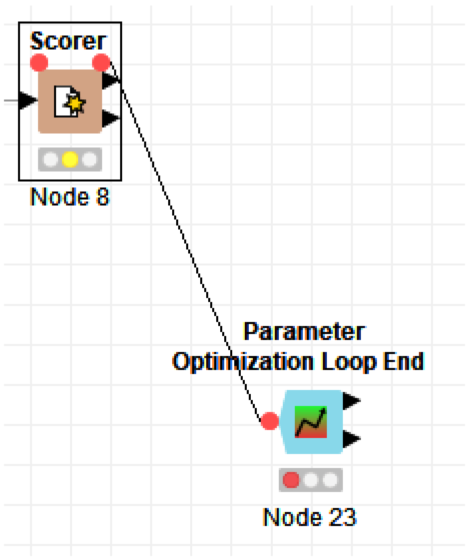
16. Once the Flow Variable is set, when you navigate to the Options tab in the configuration panel of the same *Decision Tree Learner*, you will see this message at the bottom of the dialog box:



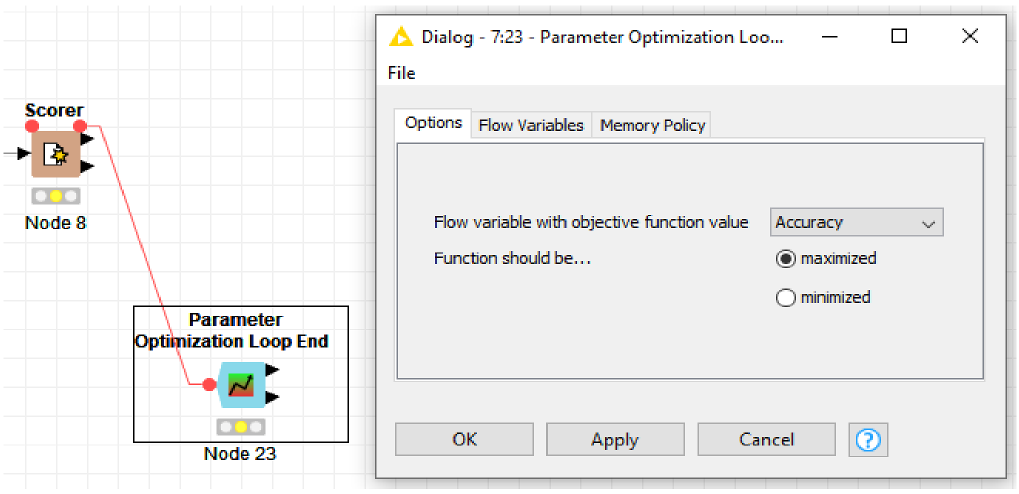
The "minNumberRecordsPerNode" parameter is controlled by a v

17. The *Scorer* node needs to be connected to the loop end node, so you have to connect the flow variables ports together also.





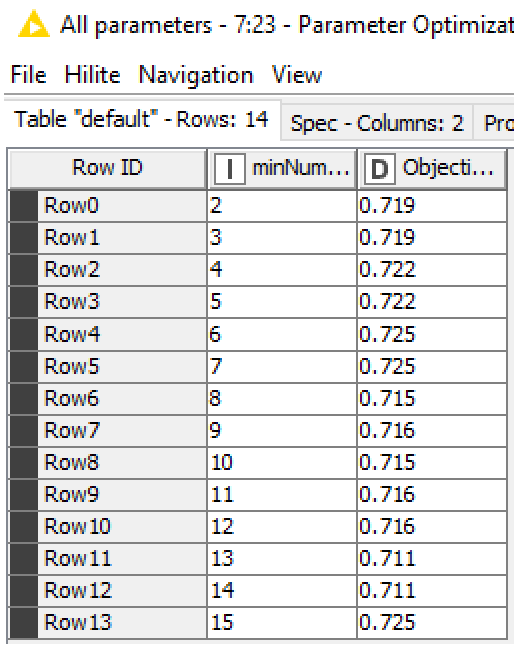
18. Finally, configure the *Parameter Optimization Loop End* node as such:


Scorer
Node 8
Parameter Optimization Loop End
Node 23
Dialog - 7:23 - Parameter Optimization Loo...
File
Options
Flow Variables
Memory Policy
OK
Apply
Cancel

Flow variable with objective function value: Accuracy

Function should be... [✓ maximized] minimized

19. Execute the whole workflow again and view the results at the loop end node:


All parameters - 7:23 - Parameter Optimizat
File Hilite Navigation View
Table "default" - Rows: 14
Spec - Columns: 2
Pro

Row ID	I minNum...	D Objecti...
Row0	2	0.719
Row1	3	0.719
Row2	4	0.722
Row3	5	0.722
Row4	6	0.725
Row5	7	0.725
Row6	8	0.715
Row7	9	0.716
Row8	10	0.715
Row9	11	0.716
Row10	12	0.716
Row11	13	0.711
Row12	14	0.711
Row13	15	0.725

We note that minNumOfRecords of 6, 7 or 15 correspond to the highest accuracy. So in using the model for prediction, we may want to use these numbers of minimum records instead.

Conclusion

We have seen how we could use KNIME to build and apply a decision tree learner. We also saw how optimization nodes in KNIME can be used to fine-tune the models built.