Implementing Support Vector Machines (SVM) using KNIME

Introduction

Support vector machine (SVM) is a robust non-probabilistic **binary** linear classifier. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. SVM is particularly well suited to analyzing data with a large number of predictor fields.

Developing an acceptable SVM model usually requires trying various model settings rather than accepting the default node settings.

Background Information on SVM Models

SVM models were developed to handle difficult classification/prediction problems where simpler linear models were unable to accurately separate the categories of an outcome field. A typical complicated problem, in two dimensions, is shown in Figure 1. Assume that the X and Y axis represent two predictors, while the circles and squares represent the two categories of a target field we wish to predict.

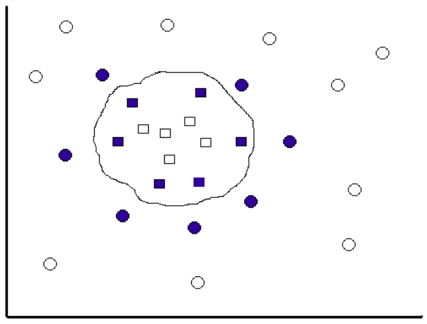


FIGURE 1: PREDICTING A BINARY OUTCOME FIELD

There is no simple straight line that can separate the categories, but the curve drawn around the squares shows that there is a complex curve that will completely separate the two categories.

The central task of SVM is to transform the data from this space into another space where the curve that separates the data points will be much simpler. Typically this means transforming the data so that a hyperplane (in higher dimensional space) can be used to separate the points.

The mathematical function used for the transformation is known as a *kernel function*. After transformation, the data points might be represented as shown in Figure 2.

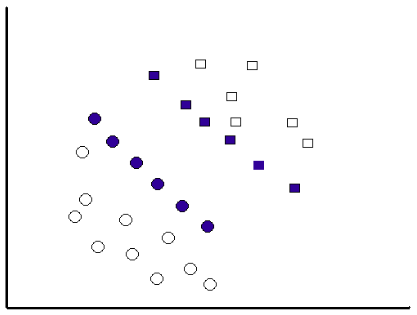


FIGURE 2: KERNEL TRANSFORMATION OF ORIGINAL DATA

The squares and circles can now be separated by a straight line in this two-dimensional space. The solid circles and squares are the cases (called *vectors* in the SVM literature) that are on the boundary between the two classes. They are the same points in Figure 1 and Figure 2. The solid circles and squares are all the data that is needed to separate the two categories, and these key points are called *support vectors* because they support the solution and boundary definition. Because SVM models were developed in the machine learning tradition, this technique was called *support vector machine*, hence the model name.

Even though it appears that we have a solution, there is more than one straight line (hyperplane) that could be used to separate the two categories, as illustrated in Figure 3.

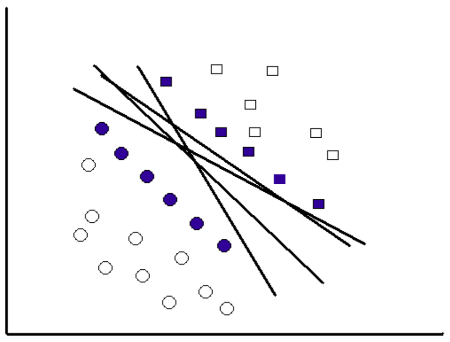


FIGURE 3: MULTIPLE POSSIBLE SEPARATING LINES

SVM models try to find the best hyperplane that maximizes the margin (separation) between the categories while balancing the tradeoff of potentially overfitting the data. The narrower the margin between the support vectors, the more accurate the model will be on the current data. Thus, a separating line as shown in Figure 4 maximizes the margin between the support vectors.

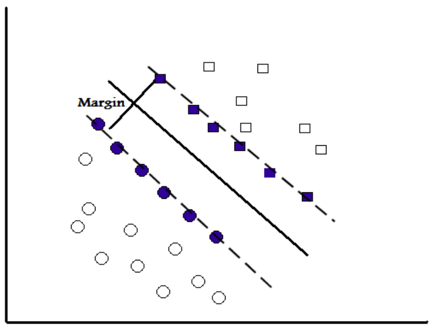


FIGURE 4: CREATING MAXIMUM SEPARATION BETWEEN SUPPORT VECTORS

Aim of demonstration

At the end of this demonstration, you should be able to

- understand the foundations of the Support Vector Machines (SVM) model
- construct an SVM model to predict customer churn
- apply kernel functions and tune model parameters to improve the model
- use cross validation to evaluate the performance of a model
- illustrate how to handle nominal data in SVM

Data

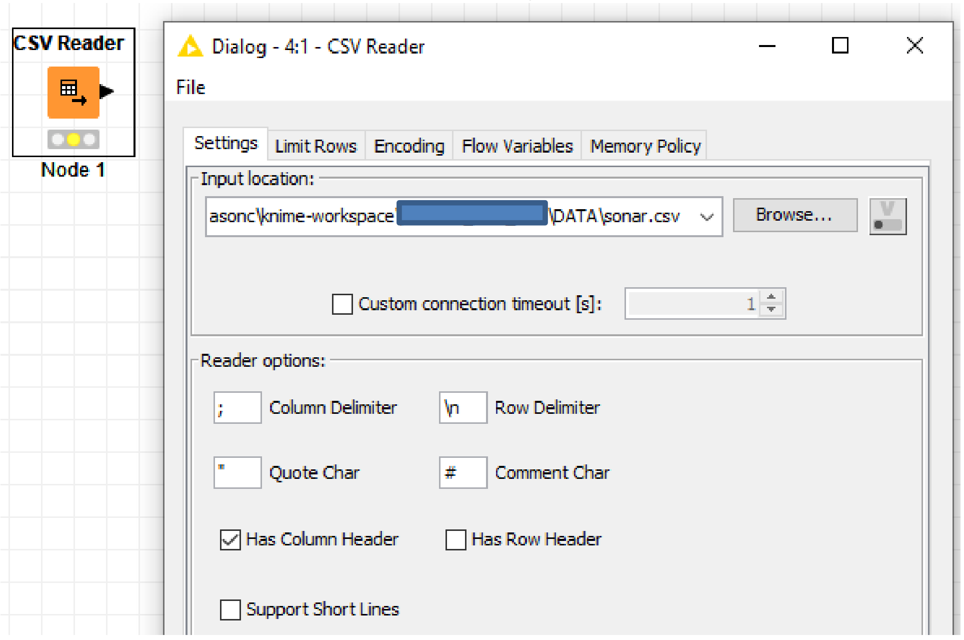
In the first part of this demonstration, we will be using the data set *sonar.csv*. The data set consists of 60 attributes that is used to predict one label attribute "class". We will see how we can use the SVM model to predict the attribute "class" as either *Rock* or *Mine*.

We will also use *Titanic.csv* as input to the SVM model to predict who would survive the historical accident. The file contains fields measuring both customer demographics and boarding details of the Titanic to use as predictors. SVM models can use many input fields, some of the fields needs to be changed in order to serve as input to our SVM operator.

Building a SVM model to predict sonar results

1. Open a new KNIME workflow.

2. Select a CSV Reader node and configure it to read from sonar.csv. Note that you should uncheck the "Has Row Header" option as our data as no column that gives each row a header. Execute the node to pull in the data.



Input location: asonc\knime-workspace[redacted]\DATA\sonar.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

3. Check for successful import of the "Sonar" data set using the CSV Reader node's output. You should see that there are 61 attributes. 60 of the attributes are named *attribute_n* (where n ranges from 1 to 60). The last attribute (shown first column here using *Column Resorter* node), "class", will be our label that we want to predict using the SVM operator.

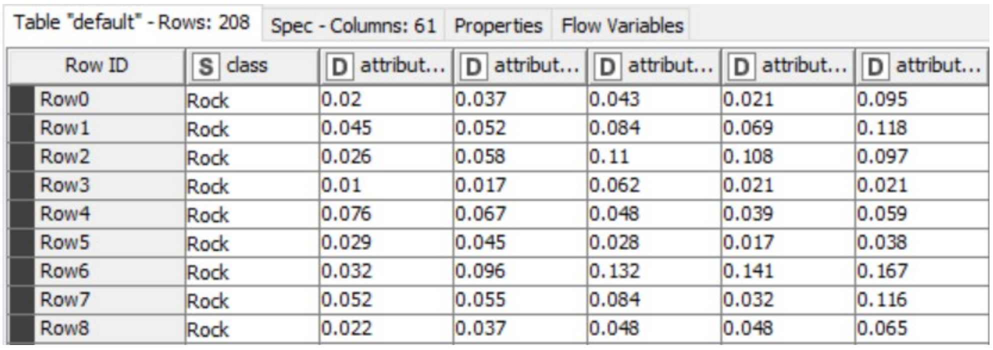
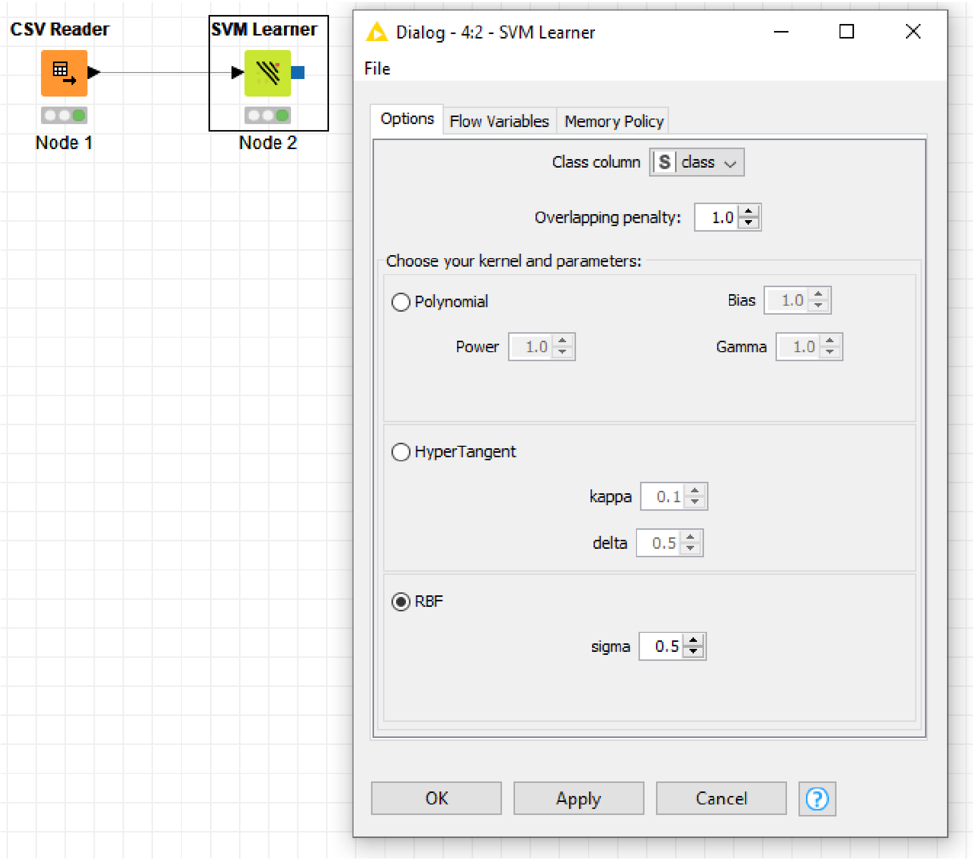


Table "default" - Rows: 208 | Spec - Columns: 61 | Properties | Flow Variables

Row ID	S class	D attribut...	D attribut...	D attribut...	D attribut...	D attribut...
Row0	Rock	0.02	0.037	0.043	0.021	0.095
Row1	Rock	0.045	0.052	0.084	0.069	0.118
Row2	Rock	0.026	0.058	0.11	0.108	0.097
Row3	Rock	0.01	0.017	0.062	0.021	0.021
Row4	Rock	0.076	0.067	0.048	0.039	0.059
Row5	Rock	0.029	0.045	0.028	0.017	0.038
Row6	Rock	0.032	0.096	0.132	0.141	0.167
Row7	Rock	0.052	0.055	0.084	0.032	0.116
Row8	Rock	0.022	0.037	0.048	0.048	0.065

4. Create a SVM Learner node on the workflow and connect the output of the CSV Reader node to this new node. You can check the default configuration. Make sure that you select the "class" attribute as the class column for the SVM to predict.



Class column: S class

Overlapping penalty: 1.0

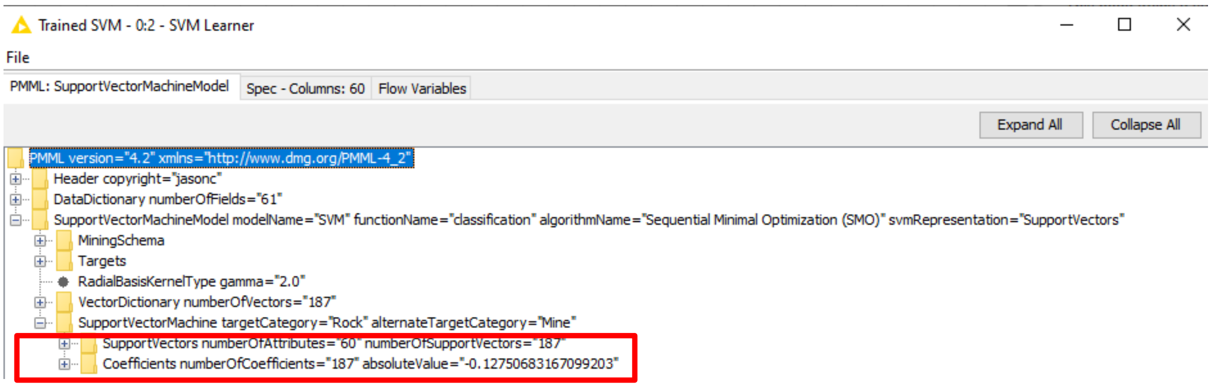
Choose your kernel and parameters:

( ) Polynomial — Bias 1.0, Power 1.0, Gamma 1.0

( ) HyperTangent — kappa 0.1, delta 0.5

(●) RBF — sigma 0.5

5. After executing the *SVM Learner*, you can see the model at the node itself. Right-click on the node, select "Trained SVM", and look at the PMML tab. The support vectors and their coefficients can be found if you expand the respective lists. For a full explanation of the PMML document, you can go to this web page:
http://dmg.org/pmml/v4-2-1/SupportVectorMachine.html

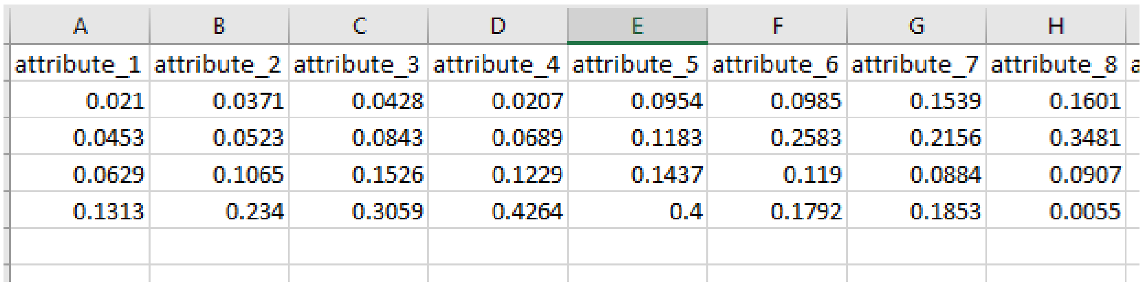

Trained SVM - 0:2 - SVM Learner
File
PMML: SupportVectorMachineModel
Spec - Columns: 60
Flow Variables
Expand All
Collapse All

```
PMML version="4.2" xmlns="http://www.dmg.org/PMML-4_2"
  Header copyright="jasonc"
  DataDictionary numberOfFields="61"
  SupportVectorMachineModel modelName="SVM" functionName="classification" algorithmName="Sequential Minimal Optimization (SMO)" svmRepresentation="SupportVectors"
    MiningSchema
    Targets
    RadialBasisKernelType gamma="2.0"
    VectorDictionary numberOfVectors="187"
    SupportVectorMachine targetCategory="Rock" alternateTargetCategory="Mine"
      SupportVectors numberOfAttributes="60" numberOfSupportVectors="187"
      Coefficients numberOfCoefficients="187" absoluteValue="-0.12750683167099203"
```

Applying SVM Model on new data

Once we have trained and generated the model, we can use it to make predictions about the "class" attribute given the 60 (*attribute_n*) input attributes. Let us see how we can do that:

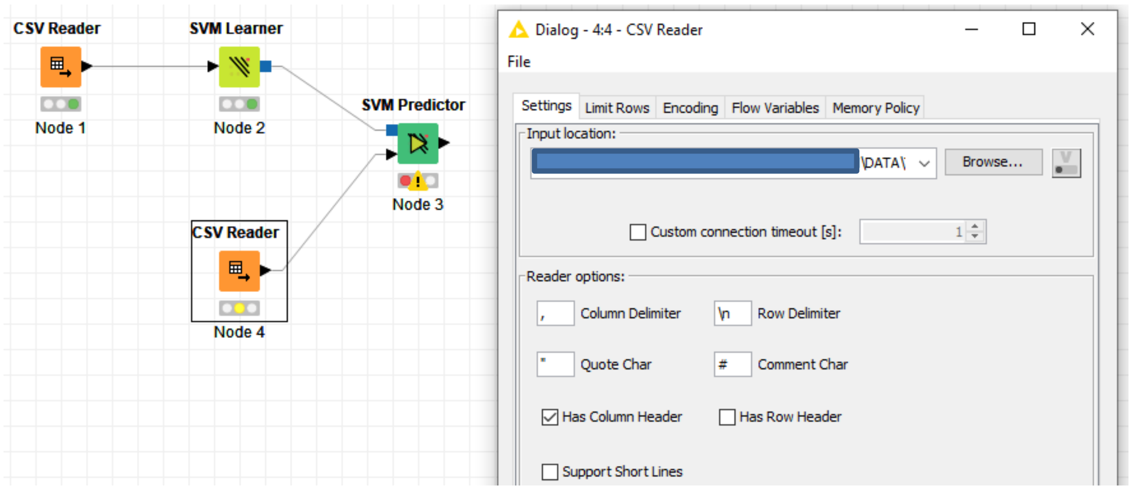
A file containing 60 input attributes named TestSVM.csv is prepared and as shown below:



A	B	C	D	E	F	G	H
attribute_1	attribute_2	attribute_3	attribute_4	attribute_5	attribute_6	attribute_7	attribute_8
0.021	0.0371	0.0428	0.0207	0.0954	0.0985	0.1539	0.1601
0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481
0.0629	0.1065	0.1526	0.1229	0.1437	0.119	0.0884	0.0907
0.1313	0.234	0.3059	0.4264	0.4	0.1792	0.1853	0.0055

6. Import the file *TestSVM.csv* file using the "CSV Reader" node. Remember to uncheck the "Has Row Header" checkbox. Verify the output and note that the *TestSVM.csv* file contains records with only the 60 input attributes and there is no "class" attribute. The "class" attribute is what our model is supposed to predict.

7. Add a "SVM Predictor" node and connect up the workflow as shown below:



Input location: \DATA\

☐ Custom connection timeout [s]: 1

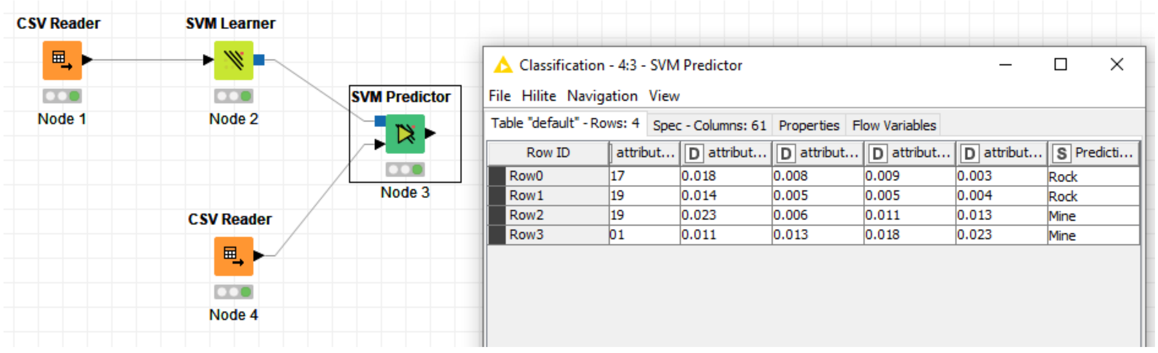
Reader options:

,	Column Delimiter	\n	Row Delimiter
"	Quote Char	#	Comment Char

☑ Has Column Header ☐ Has Row Header

☐ Support Short Lines

8. Execute the SVM Predictor node and take a look at the results of the prediction.



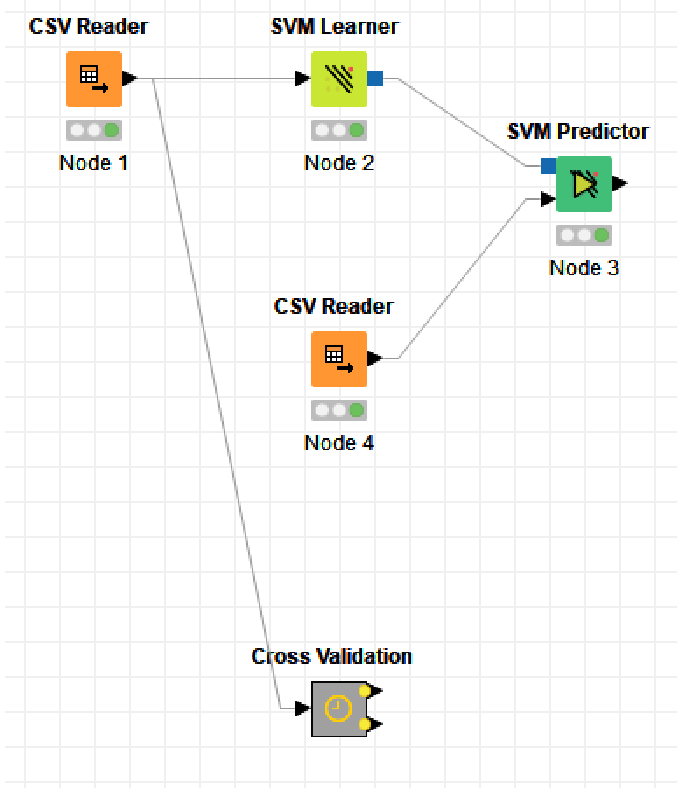
Row ID	attribut...	D attribut...	D attribut...	D attribut...	D attribut...	S Predicti...
Row0	17	0.018	0.008	0.009	0.003	Rock
Row1	19	0.014	0.005	0.005	0.004	Rock
Row2	19	0.023	0.006	0.011	0.013	Mine
Row3	01	0.011	0.013	0.018	0.023	Mine

As can be seen from the output, the support vector machine has predicted the "class" attribute. The first 2 rows are classified as "Rock" and the next 2 rows are classified as "Mine" (which is pretty accurate).

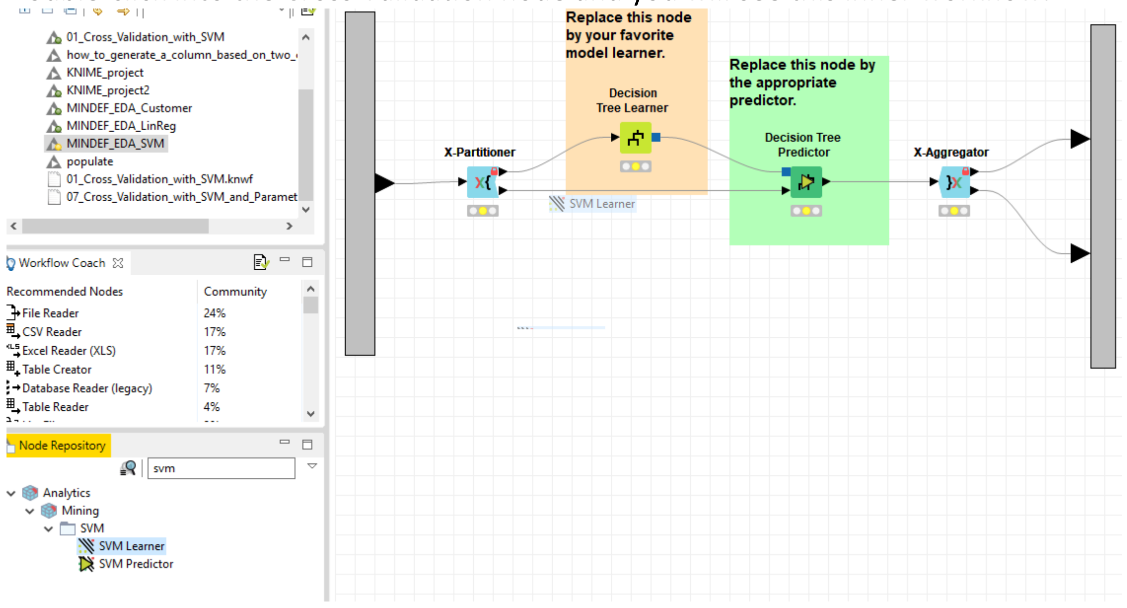
Cross-Validation

The previous section shows how to train a model using training data to generate a model. The training data contains both the input attributes (*attribute_n*) as well as the actual outcome (*class*). This will create a model that can be used to predict an outcome (*class*) given the input (*attribute_n*). Of course, the more training data we have, the more accurate the prediction will be. However, we need to ask how good our model actually is. We can test (validate) our model using cross validation. Let us see how we can do that using the *Cross-Validation* node our generated model.

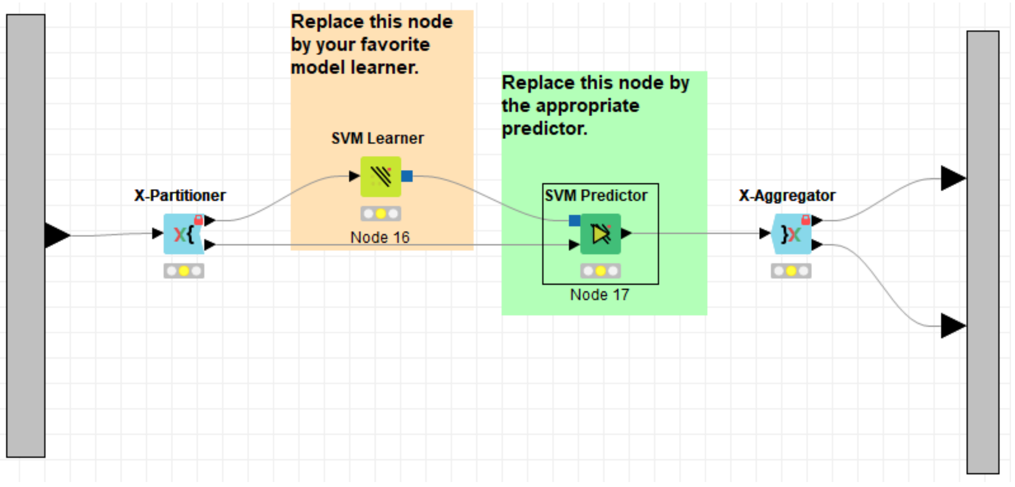
9. Connect the sonar.csv data read in by the first CSV Reader to a Cross Validation Node.



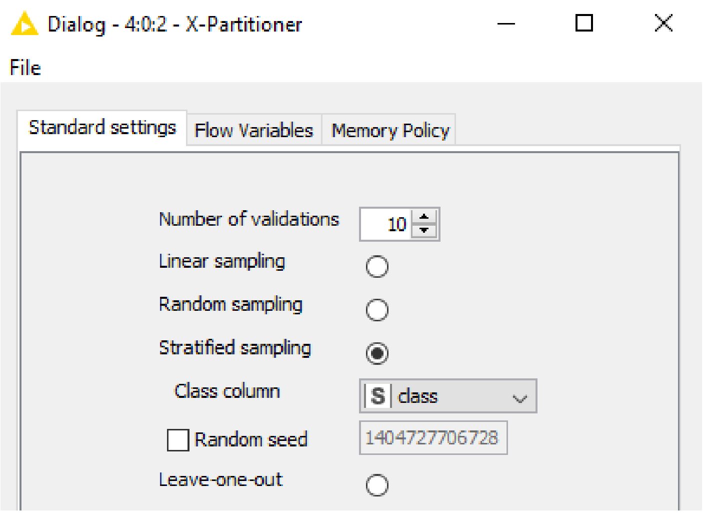
10. Double-click into the Cross Validation node and you will see this inner workflow:



11. Replace the model learner node with a SVM Learner node, and the predictor node with a SVM Predictor node.


Replace this node by your favorite model learner.
SVM Learner
Node 16
Replace this node by the appropriate predictor.
SVM Predictor
Node 17
X-Partitioner
X-Aggregator

12. Configure the X-Partitioner node to the following settings:

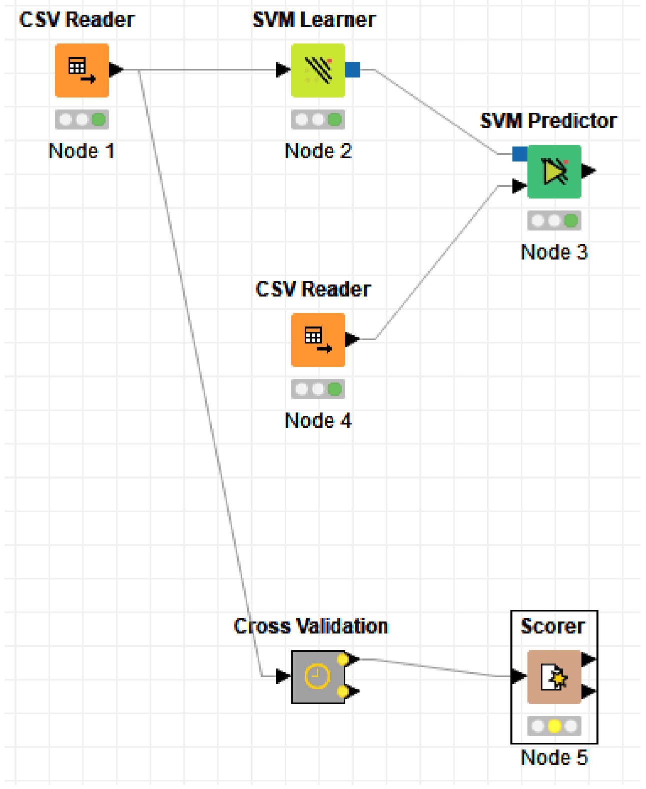

Dialog - 4:0:2 - X-Partitioner

File

Standard settings | Flow Variables | Memory Policy

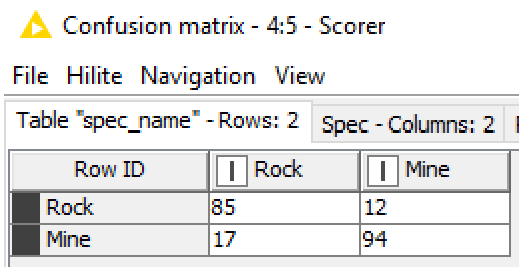
Number of validations	10
Linear sampling	○
Random sampling	○
Stratified sampling	●
Class column	S class
☐ Random seed	1404727706728
Leave-one-out	○

13. Either close this inner workflow tab or select back the tab that contains the main workflow. Connect a Scorer node to check the performance of the model.



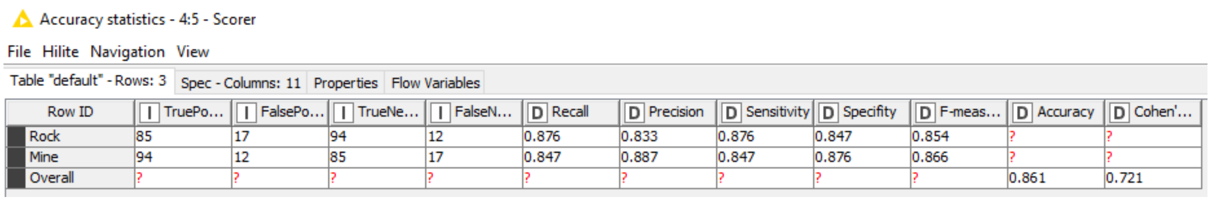
14. The results of the Scorer node are as shown:

Confusion Matrix


Confusion matrix - 4:5 - Scorer
File Hilite Navigation View
Table "spec_name" - Rows: 2
Spec - Columns: 2

Row ID	I Rock	I Mine
Rock	85	12
Mine	17	94

Accuracy Statistics


Accuracy statistics - 4:5 - Scorer
File Hilite Navigation View
Table "default" - Rows: 3
Spec - Columns: 11
Properties
Flow Variables

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specifty	D F-meas...	D Accuracy	D Cohen'...
Rock	85	17	94	12	0.876	0.833	0.876	0.847	0.854	?	?
Mine	94	12	85	17	0.847	0.887	0.847	0.876	0.866	?	?
Overall	?	?	?	?	?	?	?	?	?	0.861	0.721

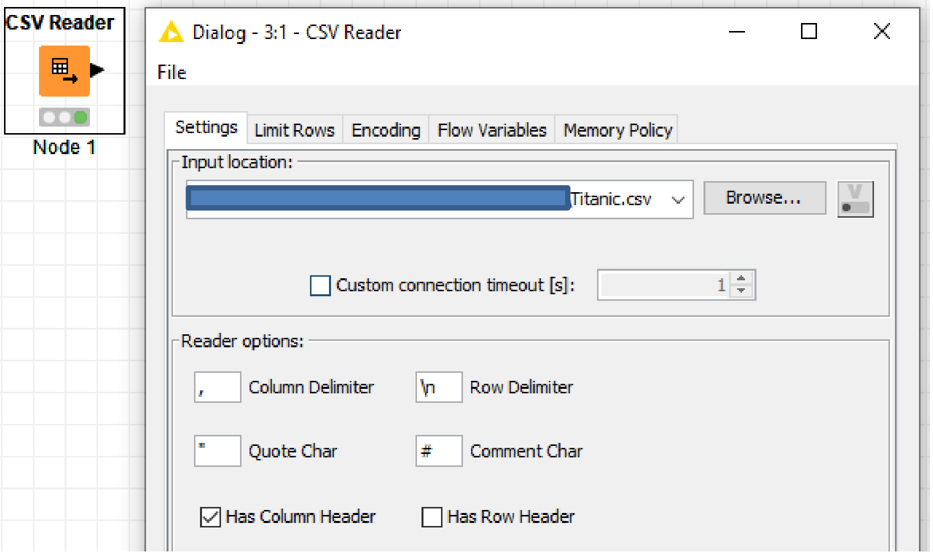
We note that the SVM model has an overall accuracy of 86.1%, and the other accuracy statistics do not show any adverse result. The model can be considered as being quite a good model. If the results are not satisfactory, we will need more training data in order to improve the various criterions. Alternatively, we can tune the SVM operator to see if our algorithm can provide a better result.

Dealing with Non-Numerical Attributes

The Sonar example set we used previously contains 60 input attributes of numerical data type and 1 label attribute of categorical type. What happens if our data contains input categorical data type? SVM node will not accept examples with categorical data type. Also, the SVM node has problem with data with missing values.

We will now see how to handle these using a database of Titanic passengers. The database contains a field (*Survived*) that measures whether or not a passenger survived the famous shipwreck. It also has quite a number of nominal data type, as well as issues like missing values that we need to manage. Let us now see how to handle these in order to train a model and assess its performance.

15. Read in the data using the CSV Reader node. Remember to uncheck the "Has Row Header"option.



CSV Reader — Node 1

Dialog - 3:1 - CSV Reader

File

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

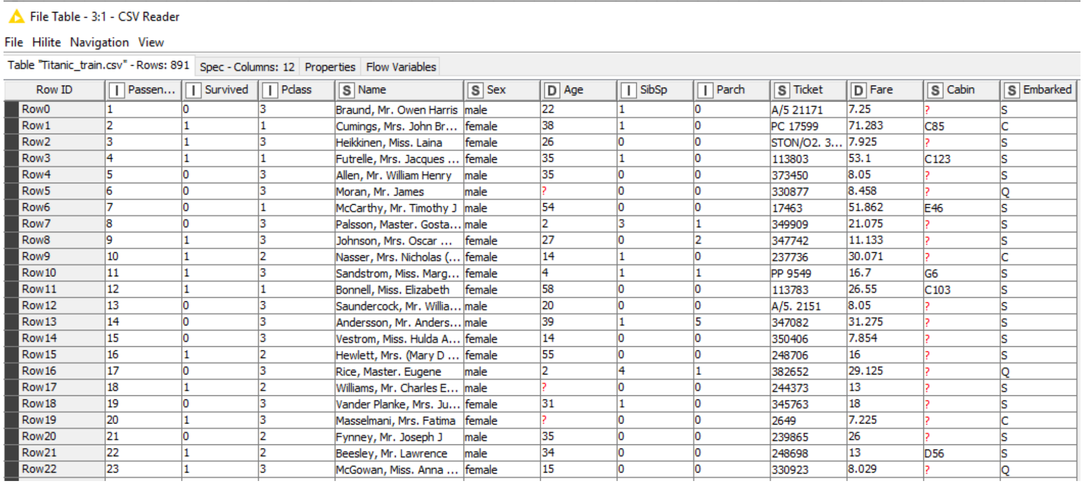
Input location: Titanic.csv — Browse...

☐ Custom connection timeout [s]: 1

Reader options:

- Column Delimiter: ,
- Row Delimiter: \n
- Quote Char: "
- Comment Char: #
- ☑ Has Column Header
- ☐ Has Row Header

16. Check the imported data:



File Table - 3:1 - CSV Reader

File Hilite Navigation View

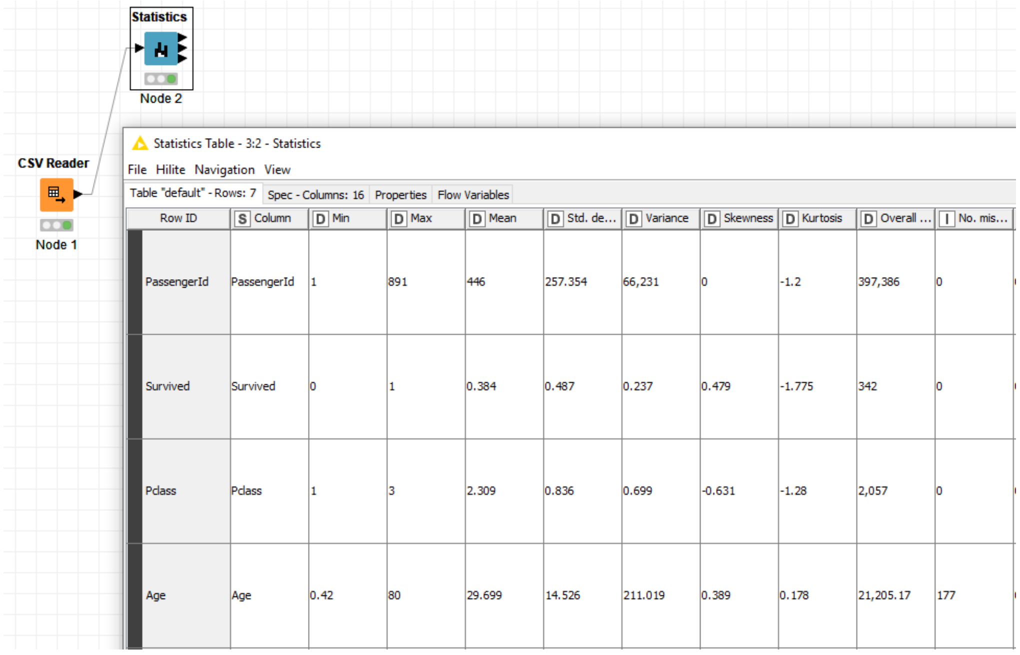
Table "Titanic_train.csv" - Rows: 891 | Spec - Columns: 12 | Properties | Flow Variables

Row ID	Passen...	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Row0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	?	S
Row1	2	1	1	Cumings, Mrs. John Br...	female	38	1	0	PC 17599	71.283	C85	C
Row2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3...	7.925	?	S
Row3	4	1	1	Futrelle, Mrs. Jacques ...	female	35	1	0	113803	53.1	C123	S
Row4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	?	S
Row5	6	0	3	Moran, Mr. James	male	?	0	0	330877	8.458	?	Q
Row6	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.862	E46	S
Row7	8	0	3	Palsson, Master. Gosta...	male	2	3	1	349909	21.075	?	S
Row8	9	1	3	Johnson, Mrs. Oscar ...	female	27	0	2	347742	11.133	?	S
Row9	10	1	2	Nasser, Mrs. Nicholas (...	female	14	1	0	237736	30.071	?	C
Row10	11	1	3	Sandstrom, Miss. Marg...	female	4	1	1	PP 9549	16.7	G6	S
Row11	12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
Row12	13	0	3	Saundercock, Mr. Willia...	male	20	0	0	A/5. 2151	8.05	?	S
Row13	14	0	3	Andersson, Mr. Anders...	male	39	1	5	347082	31.275	?	S
Row14	15	0	3	Vestrom, Miss. Hulda A...	female	14	0	0	350406	7.854	?	S
Row15	16	1	2	Hewlett, Mrs. (Mary D ...	female	55	0	0	248706	16	?	S
Row16	17	0	3	Rice, Master. Eugene	male	2	4	1	382652	29.125	?	Q
Row17	18	1	2	Williams, Mr. Charles E...	male	?	0	0	244373	13	?	S
Row18	19	0	3	Vander Planke, Mrs. Ju...	female	31	1	0	345763	18	?	S
Row19	20	1	3	Masselmani, Mrs. Fatima	female	?	0	0	2649	7.225	?	C
Row20	21	0	2	Fynney, Mr. Joseph J	male	35	0	0	239865	26	?	S
Row21	22	1	2	Beesley, Mr. Lawrence	male	34	0	0	248698	13	D56	S
Row22	23	1	3	McGowan, Miss. Anna ...	female	15	0	0	330923	8.029	?	Q

We note that the data has missing values in a number of columns, as well as input categorical data. There are also columns that may not be useful for machine learning. Let's check for the statistics of the data using a node instead of relying on visual inspection.

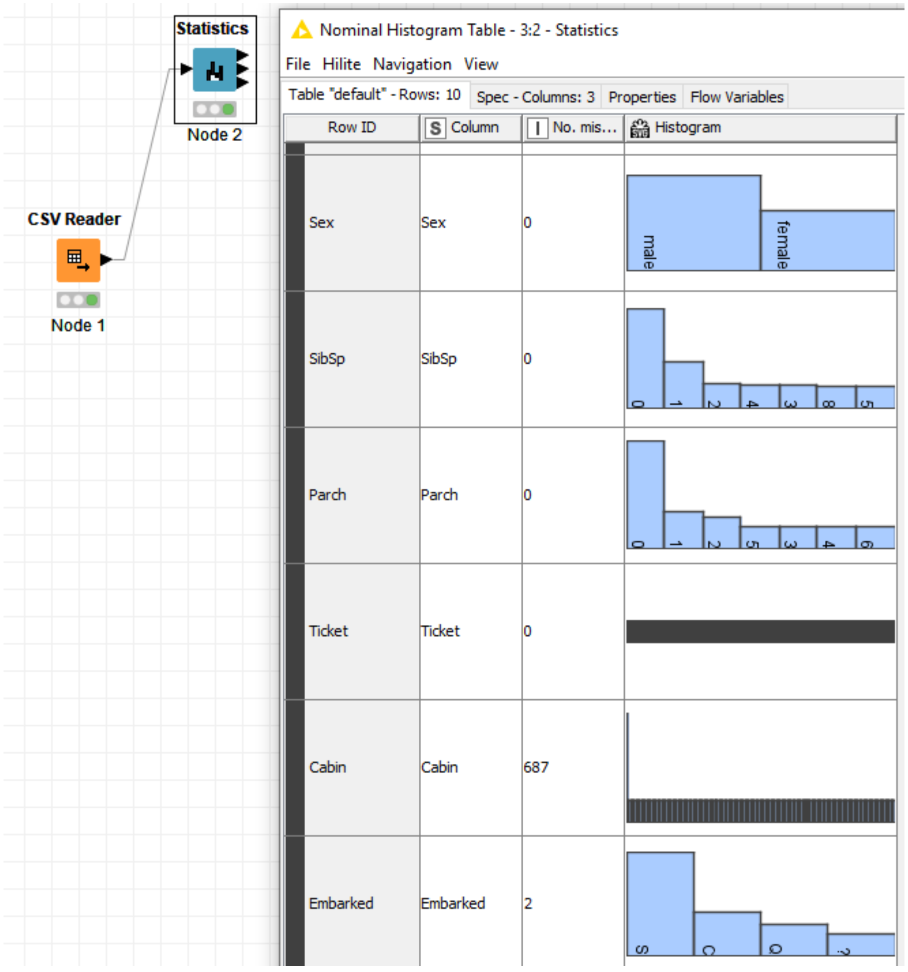
17. Connect the output of the CSV Reader to a Statistics node. Execute the Statistics node and check the output.

Numerical data statistics


Statistics
Node 2
CSV Reader
Node 1
Statistics Table - 3:2 - Statistics
File Hilite Navigation View
Table "default" - Rows: 7
Spec - Columns: 16
Properties
Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...
PassengerId	PassengerId	1	891	446	257.354	66,231	0	-1.2	397,386	0
Survived	Survived	0	1	0.384	0.487	0.237	0.479	-1.775	342	0
Pclass	Pclass	1	3	2.309	0.836	0.699	-0.631	-1.28	2,057	0
Age	Age	0.42	80	29.699	14.526	211.019	0.389	0.178	21,205.17	177

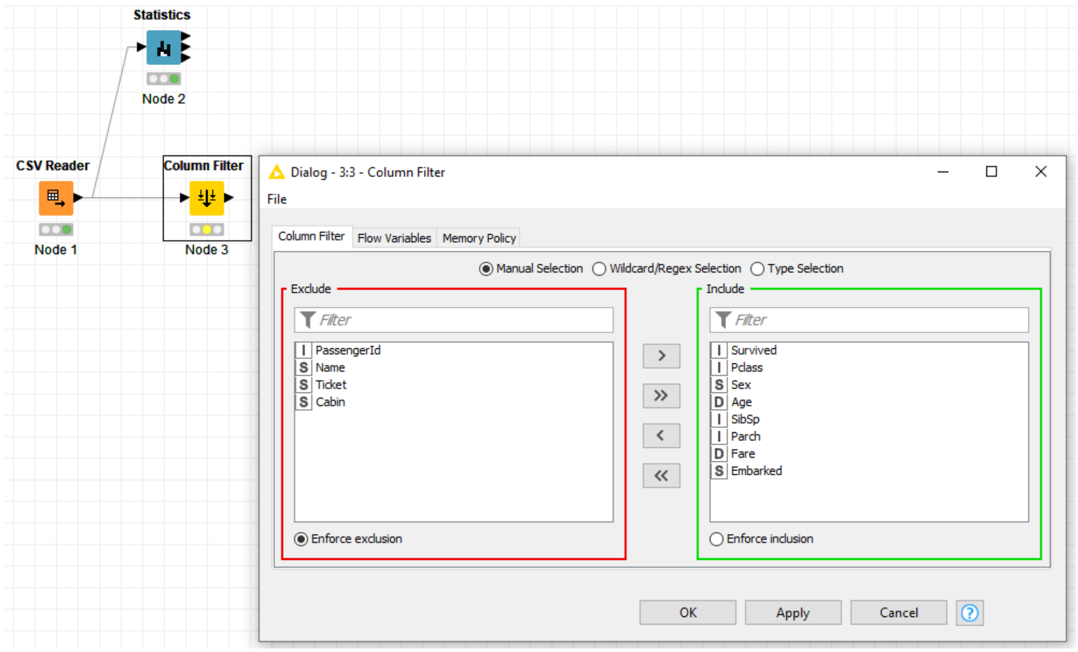
Categorical data statistics


Statistics
Node 2
CSV Reader
Node 1
Nominal Histogram Table - 3:2 - Statistics
File Hilite Navigation View
Table "default" - Rows: 10
Spec - Columns: 3
Properties
Flow Variables

Row ID	S Column	I No. mis...	Histogram
Sex	Sex	0	male, female
SibSp	SibSp	0	0, 1, 2, 4, 3, 8, 5
Parch	Parch	0	0, 1, 2, 5, 3, 4, 6
Ticket	Ticket	0	
Cabin	Cabin	687	
Embarked	Embarked	2	S, C, Q, ?

Let's first filter out the columns that are not useful for machine learning. We are going to remove the following columns: "PassengerId", "Name", "Ticket", and "Cabin". The first two candidates for removal do not give useful information for prediction, while "Ticket" holds the serial number of the tickets held by the passengers. "Cabin" has 687 missing values for 891 rows of data, so we might as well remove it.

18. Connect a Column Filter node to the output of the CSV Reader and configure it as such:



[✓ Manual Selection] Wildcard/Regex Selection Type Selection

Exclude: PassengerId (I), Name (S), Ticket (S), Cabin (S)

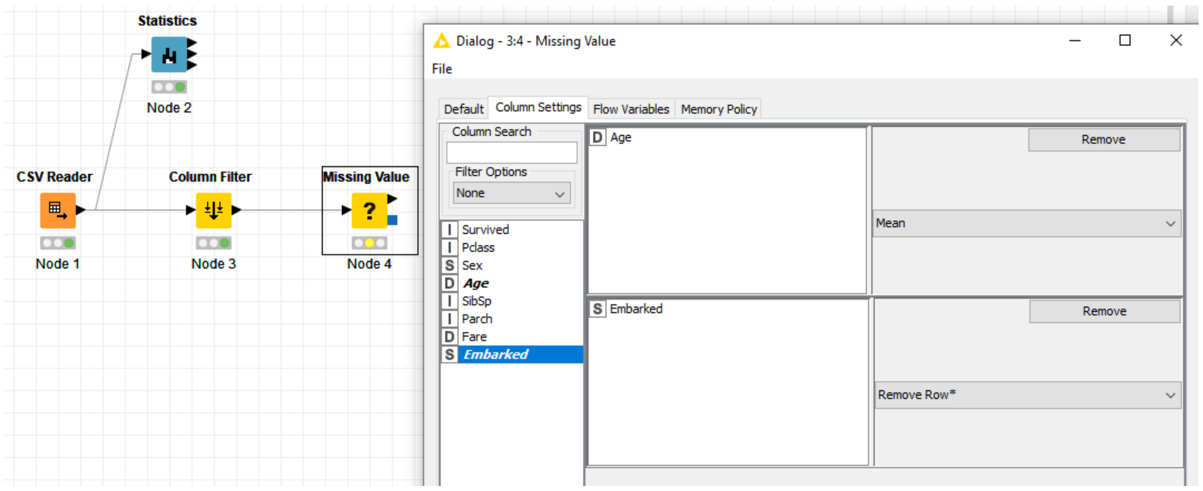
[✓ Enforce exclusion]

Include: Survived (I), Pclass (I), Sex (S), Age (D), SibSp (I), Parch (I), Fare (D), Embarked (S)

Enforce inclusion

The output of this column filter will have these 4 columns removed. Next, let's deal with missing values. The "Age" column has 177 missing values, so instead of removing them, we will use the mean of the ages present to impute the missing values. "Embarked" only has 2 missing values, so we will simply remove them.

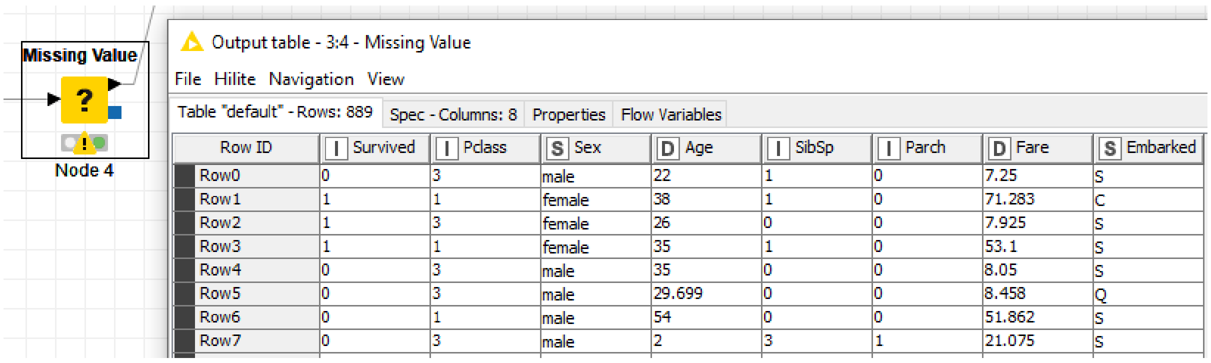
19. Connect a Missing Value node to the output of the Column Filter and configure it as such:



D Age: Mean

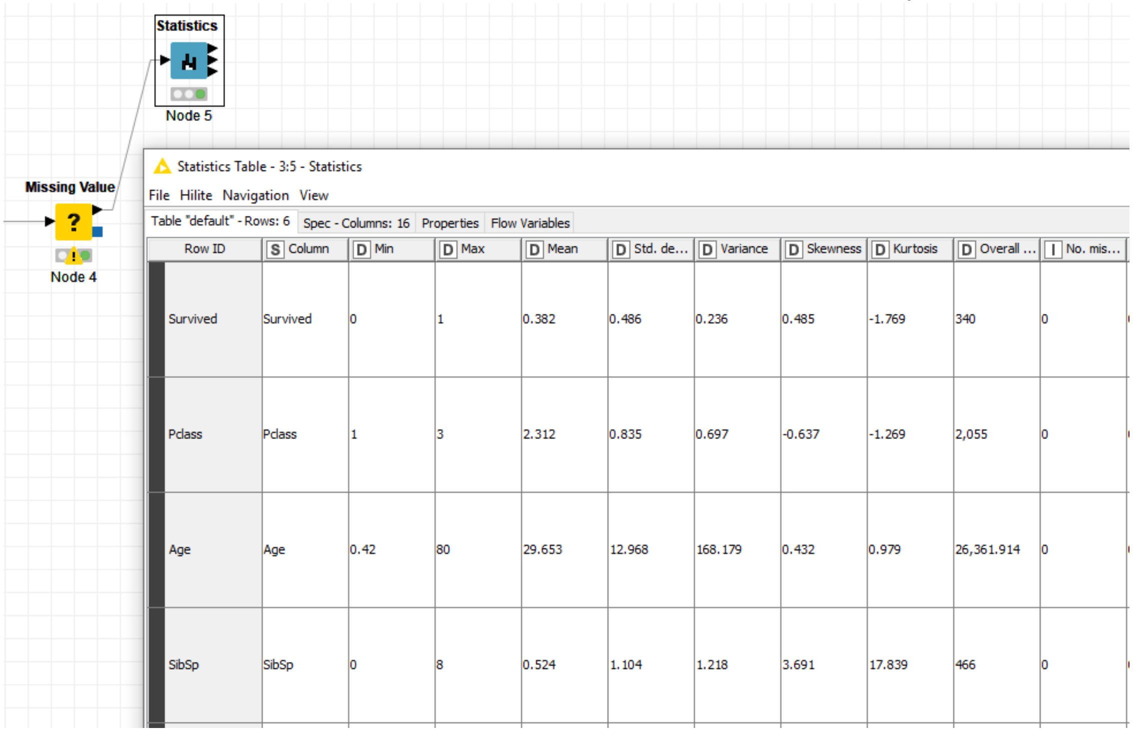
S Embarked: Remove Row*

20. You can check the output of the Missing Values node itself visually.

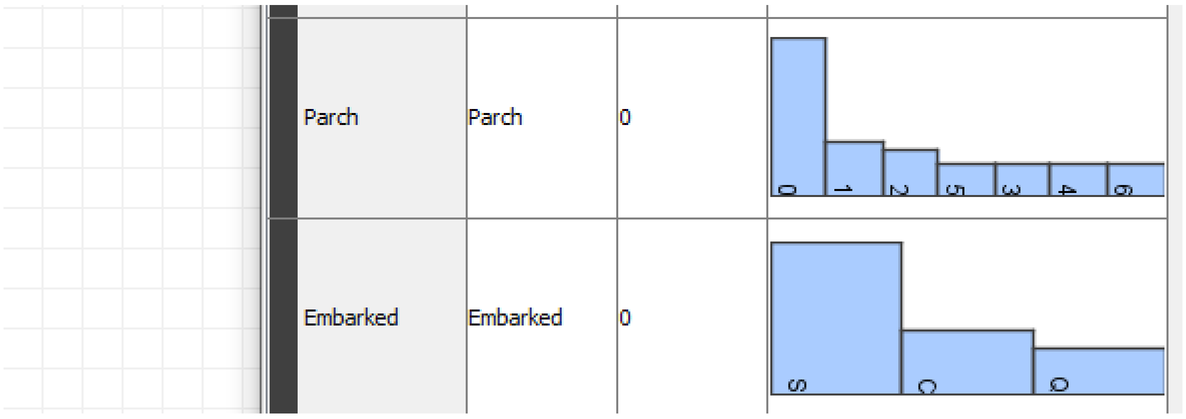

Missing Value
Node 4
Output table - 3:4 - Missing Value
File Hilite Navigation View
Table "default" - Rows: 889
Spec - Columns: 8
Properties
Flow Variables

Row ID	I Survived	I Pclass	S Sex	D Age	I SibSp	I Parch	D Fare	S Embarked
Row0	0	3	male	22	1	0	7.25	S
Row1	1	1	female	38	1	0	71.283	C
Row2	1	3	female	26	0	0	7.925	S
Row3	1	1	female	35	1	0	53.1	S
Row4	0	3	male	35	0	0	8.05	S
Row5	0	3	male	29.699	0	0	8.458	Q
Row6	0	1	male	54	0	0	51.862	S
Row7	0	3	male	2	3	1	21.075	S

21. If you wish, you can connect another Statistics node to the output of the Missing Values node to check that the desired actions were executed correctly.


Statistics
Node 5
Missing Value
Node 4
Statistics Table - 3:5 - Statistics
File Hilite Navigation View
Table "default" - Rows: 6
Spec - Columns: 16
Properties
Flow Variables

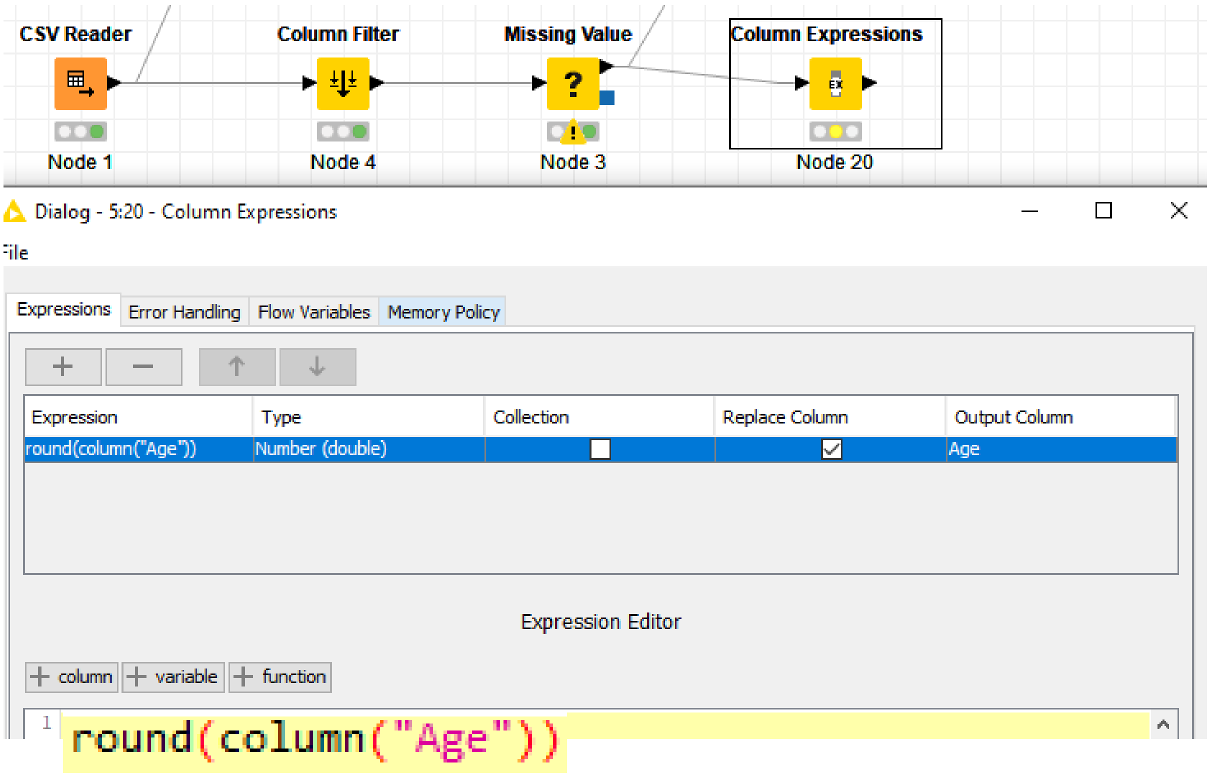
Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...
Survived	Survived	0	1	0.382	0.486	0.236	0.485	-1.769	340	0
Pclass	Pclass	1	3	2.312	0.835	0.697	-0.637	-1.269	2,055	0
Age	Age	0.42	80	29.653	12.968	168.179	0.432	0.979	26,361.914	0
SibSp	SibSp	0	8	0.524	1.104	1.218	3.691	17.839	466	0



Parch	Parch	0	
Embarked	Embarked	0	

Due to the imputation of the missing age values to the mean of the ages present, the output now shows decimal floating points for the imputed values. Let's round the values to the nearest integer for conformity's sake.

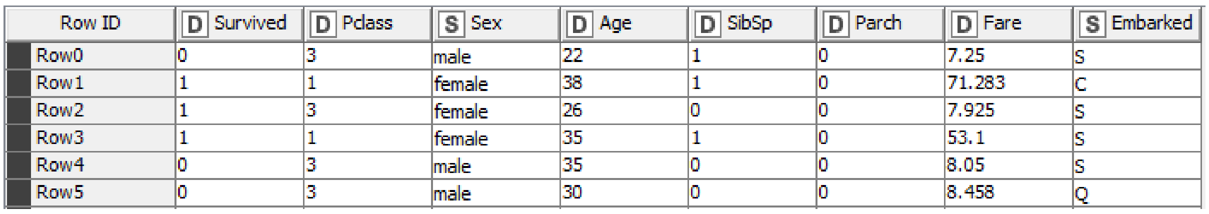
22. Connect a *Column Expressions* node to the output of the *Missing Value* node and configure it as shown:



Expression	Type	Collection	Replace Column	Output Column
round(column("Age"))	Number (double)	☐	✓	Age

```
round(column("Age"))
```

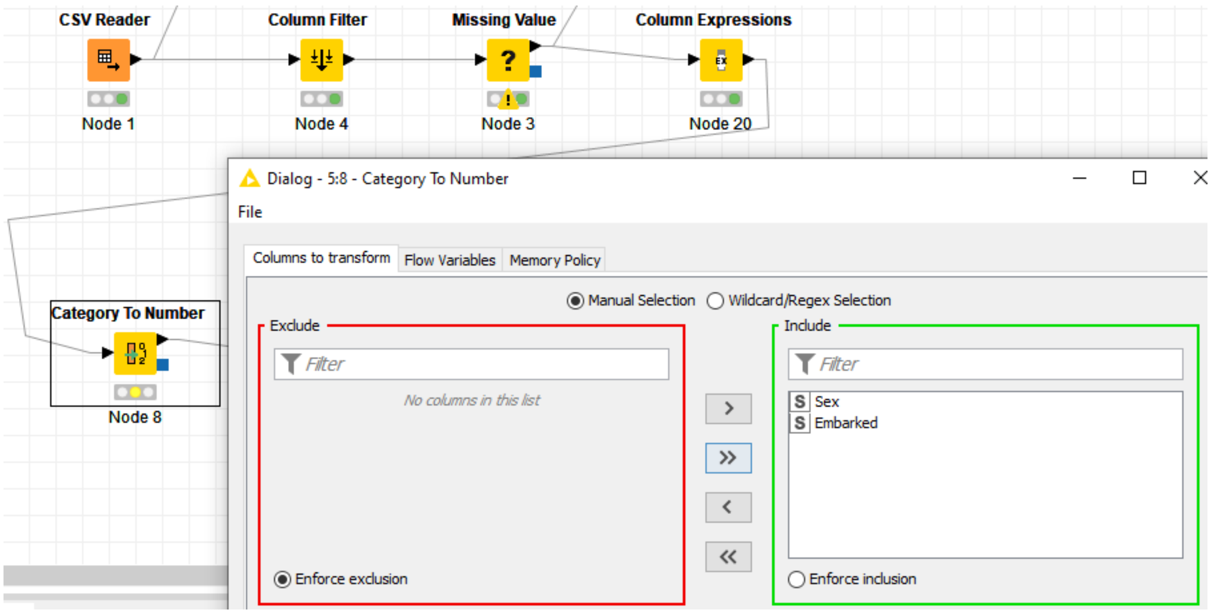
23. Execute the *Column Expressions* node and check that there are no floating point values in the age column.



Row ID	D Survived	D Pclass	S Sex	D Age	D SibSp	D Parch	D Fare	S Embarked
Row0	0	3	male	22	1	0	7.25	S
Row1	1	1	female	38	1	0	71.283	C
Row2	1	3	female	26	0	0	7.925	S
Row3	1	1	female	35	1	0	53.1	S
Row4	0	3	male	35	0	0	8.05	S
Row5	0	3	male	30	0	0	8.458	Q

As was mentioned, the SVM Learner cannot deal with categorical inputs. Let's change the two categorical inputs, "Sex" and "Embarked", to numerical values.

24. Connect a Category to Number node to the *Column Expressions* node, set the following configuration and execute the node.



Manual Selection [✓] Wildcard/Regex Selection

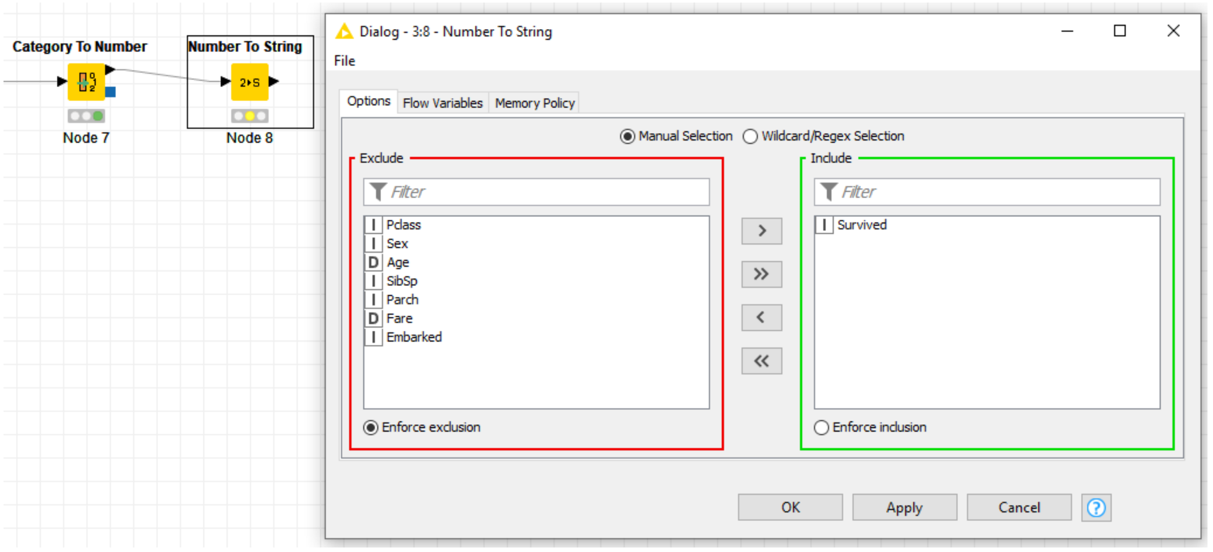
Exclude: No columns in this list

Include: S Sex; S Embarked

Enforce exclusion [✓] Enforce inclusion

On the other hand, SVM Learner expects the label column to be categorical. So we need to change the data type of the "Survived" column.

25. Connect a Number to String node to the output of the Category to Number node. Select the "Survived" column for transformation and execute the node.



Manual Selection [✓] Wildcard/Regex Selection

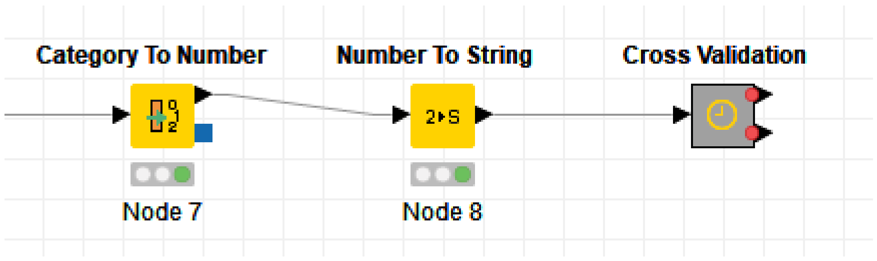
Exclude: I Pclass; I Sex; D Age; I SibSp; I Parch; D Fare; I Embarked

Include: I Survived

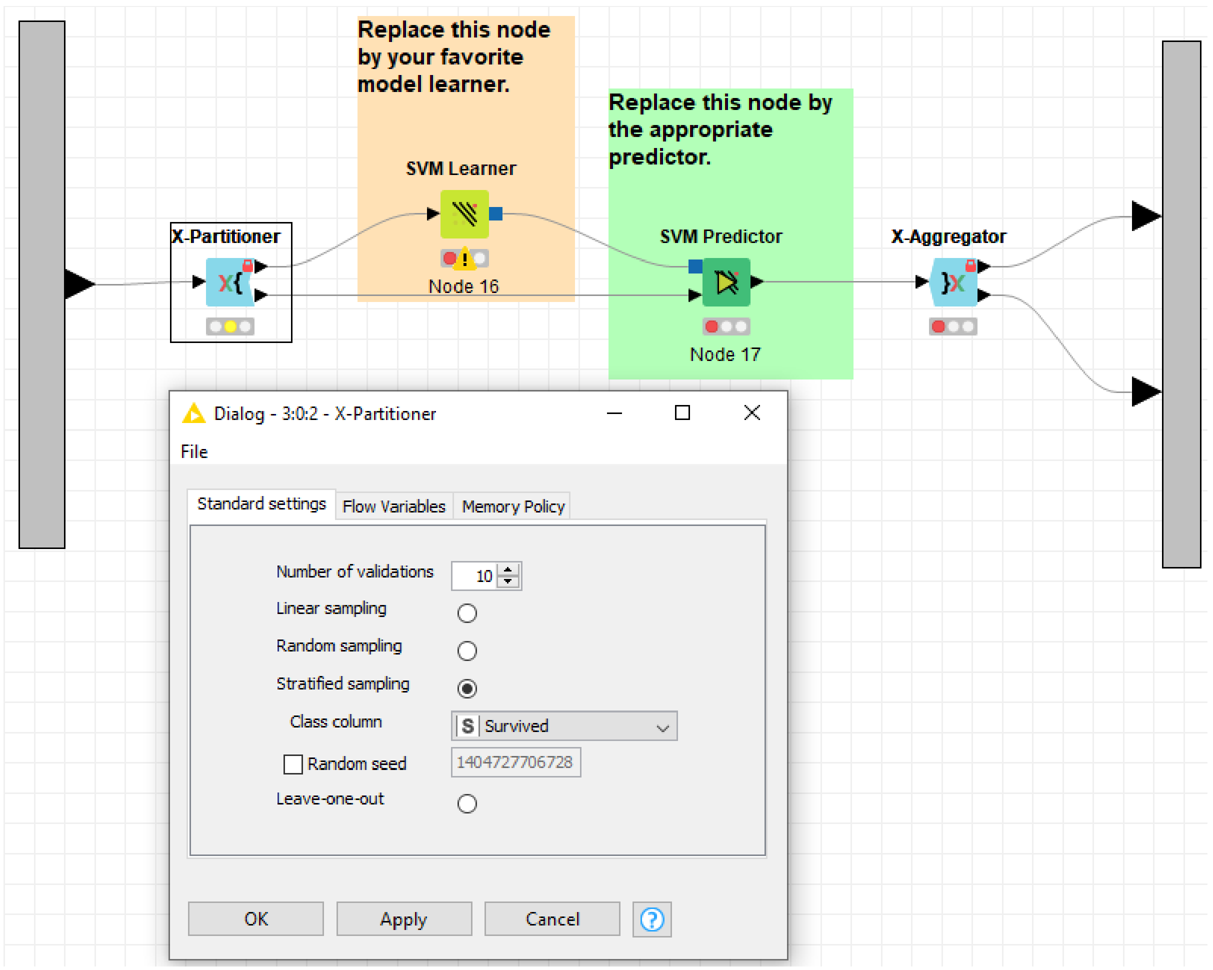
Enforce exclusion [✓] Enforce inclusion

We are now ready to perform the SVM learning and prediction.

26. Connect a Cross Validation node to the latest output.

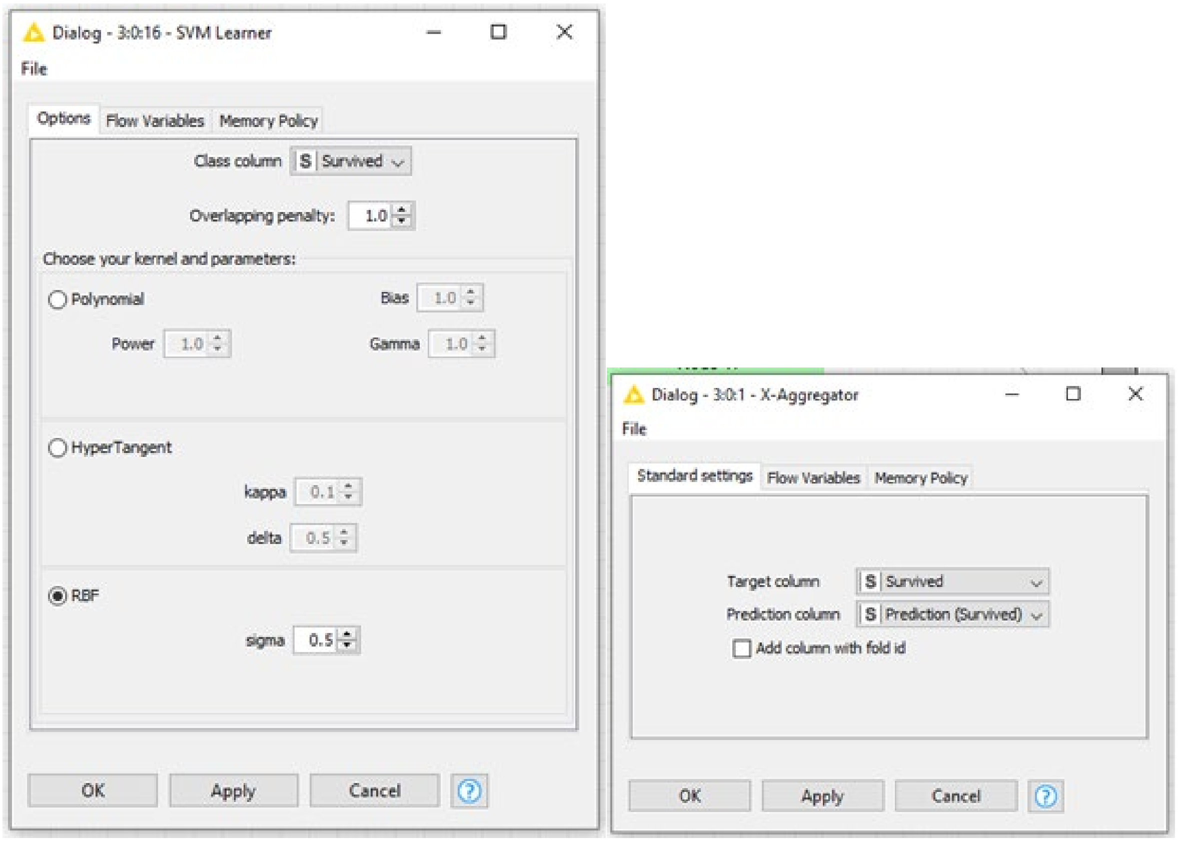


27. Double-click to enter the inner workflow of the Cross Validation node and change the learner and predictor to SVM. Configure the X-Partitioner accordingly.


Replace this node by your favorite model learner.
Replace this node by the appropriate predictor.
X-Partitioner
SVM Learner
Node 16
SVM Predictor
Node 17
X-Aggregator
Dialog - 3:0:2 - X-Partitioner
File
Standard settings
Flow Variables
Memory Policy
OK
Apply
Cancel

Number of validations: 10
Linear sampling: ○
Random sampling: ○
Stratified sampling: ◉
Class column: S Survived
☐ Random seed: 1404727706728
Leave-one-out: ○

28. Check that the Learner, Predictor and X-Aggregator nodes are correctly configured e.g.



Dialog - 3:0:16 - SVM Learner

Class column: S Survived

Overlapping penalty: 1.0

Choose your kernel and parameters:

○ Polynomial — Bias 1.0, Power 1.0, Gamma 1.0

○ HyperTangent — kappa 0.1, delta 0.5

● RBF — sigma 0.5

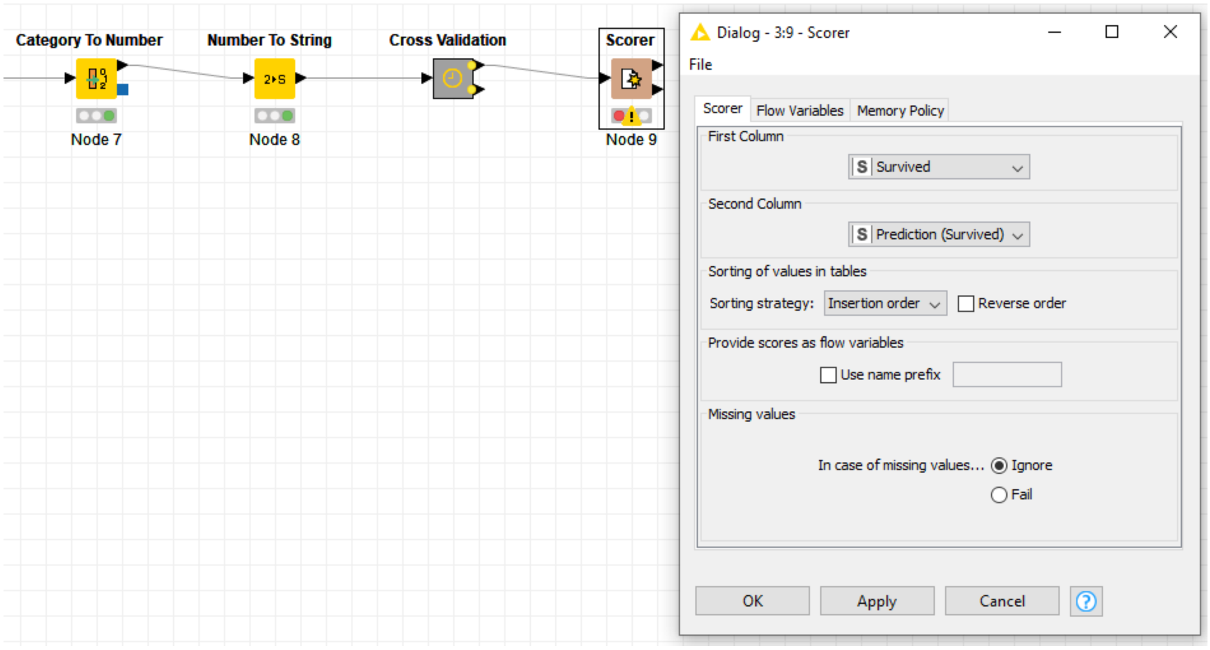
Dialog - 3:0:1 - X-Aggregator

Target column: S Survived

Prediction column: S Prediction (Survived)

☐ Add column with fold id

29. Go back to the main workflow and connect a Scorer node to measure the performance of the model.



Dialog - 3:9 - Scorer

First Column: S Survived

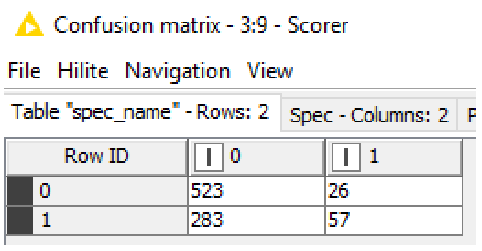
Second Column: S Prediction (Survived)

Sorting of values in tables — Sorting strategy: Insertion order ☐ Reverse order

Provide scores as flow variables — ☐ Use name prefix

Missing values — In case of missing values... ● Ignore ○ Fail

30. Check the output of the Scorer node:

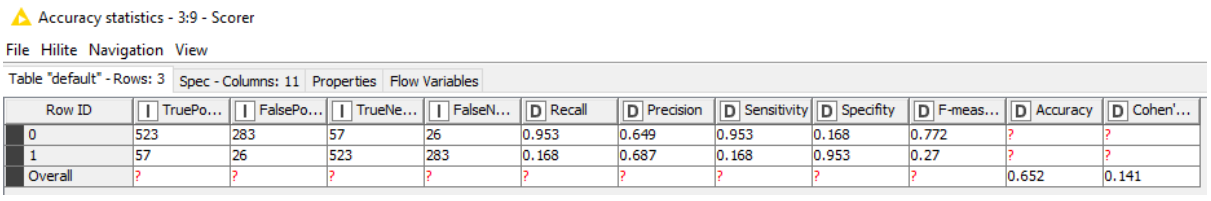


Confusion matrix - 3:9 - Scorer

File Hilite Navigation View

Table "spec_name" - Rows: 2 | Spec - Columns: 2

Row ID	0	1
0	523	26
1	283	57



Accuracy statistics - 3:9 - Scorer

File Hilite Navigation View

Table "default" - Rows: 3 | Spec - Columns: 11 | Properties | Flow Variables

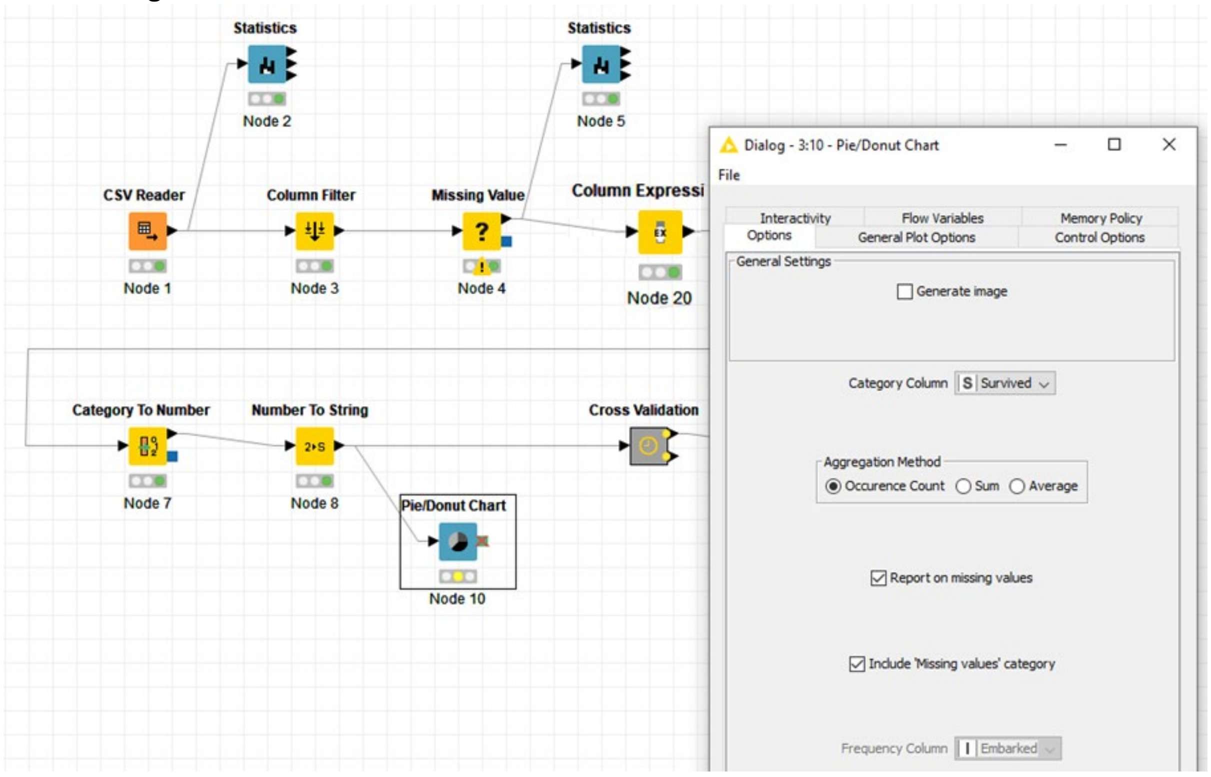
Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
0	523	283	57	26	0.953	0.649	0.953	0.168	0.772	?	?
1	57	26	523	283	0.168	0.687	0.168	0.953	0.27	?	?
Overall	?	?	?	?	?	?	?	?	?	0.652	0.141

We can see that the Recall and Sensitivity of the positive values for "Survived" has quite low values. The overall accuracy is also on the low side of 65.2%. Let's see how we can improve the accuracy of the model.

Equal Size Sampling

One common issue with machine learning is imbalanced data. Models do not perform well or give good predictions when the label column is imbalanced. Let's check the distribution of the label class of our data first.

31. Connect a Pie/Donut Chart node to the output of the Number to String node. Configure it as such:



Dialog - 3:10 - Pie/Donut Chart

File

Interactivity | Flow Variables | Memory Policy | Options | General Plot Options | Control Options

General Settings

☐ Generate image

Category Column: S Survived

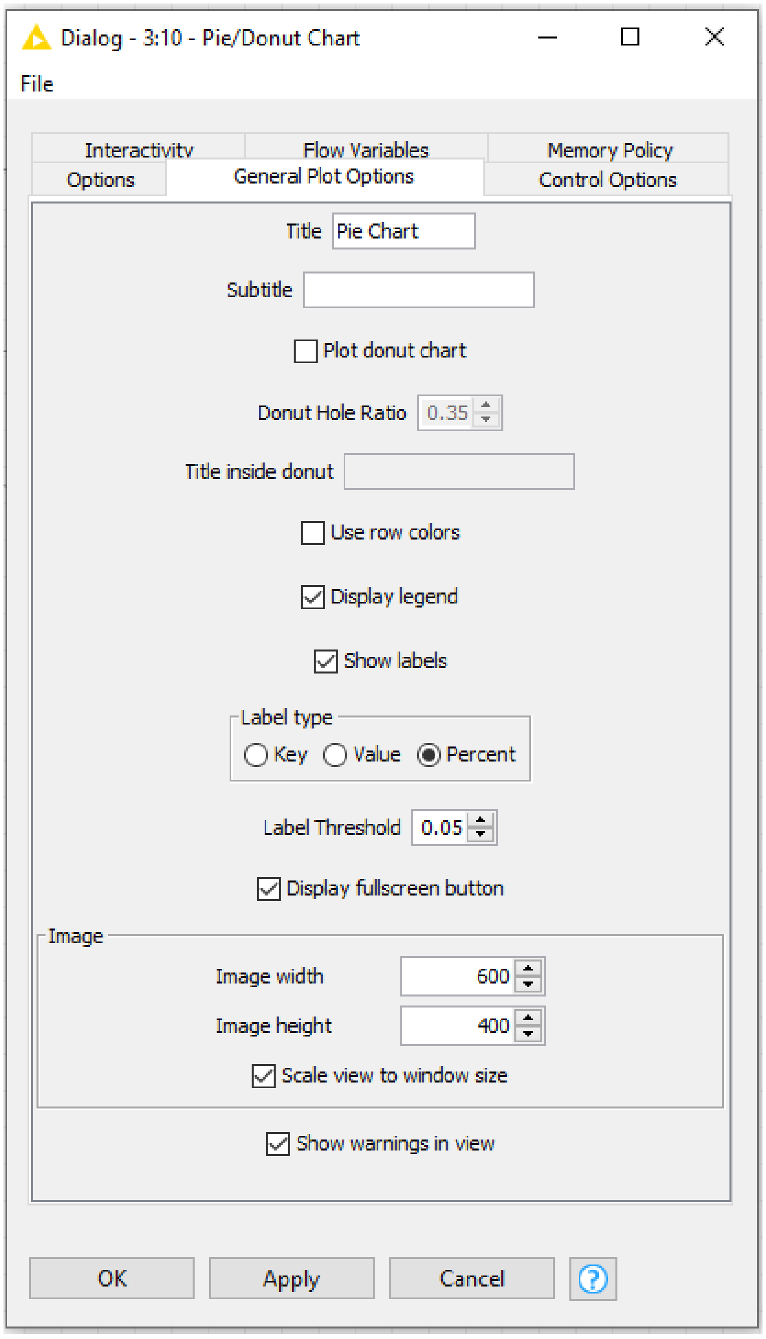
Aggregation Method: [✓ Occurence Count] Sum Average

☑ Report on missing values

☑ Include 'Missing values' category

Frequency Column: I Embarked

32. Under the General Plot Options tab, configure the following settings:


Dialog - 3:10 - Pie/Donut Chart

File

Interactivity | Flow Variables | Memory Policy | Options | General Plot Options | Control Options

Title: Pie Chart

Subtitle:

[ ] Plot donut chart

Donut Hole Ratio: 0.35

Title inside donut:

[ ] Use row colors

[✓] Display legend

[✓] Show labels

Label type: Key Value [✓ Percent]

Label Threshold: 0.05

[✓] Display fullscreen button

Image

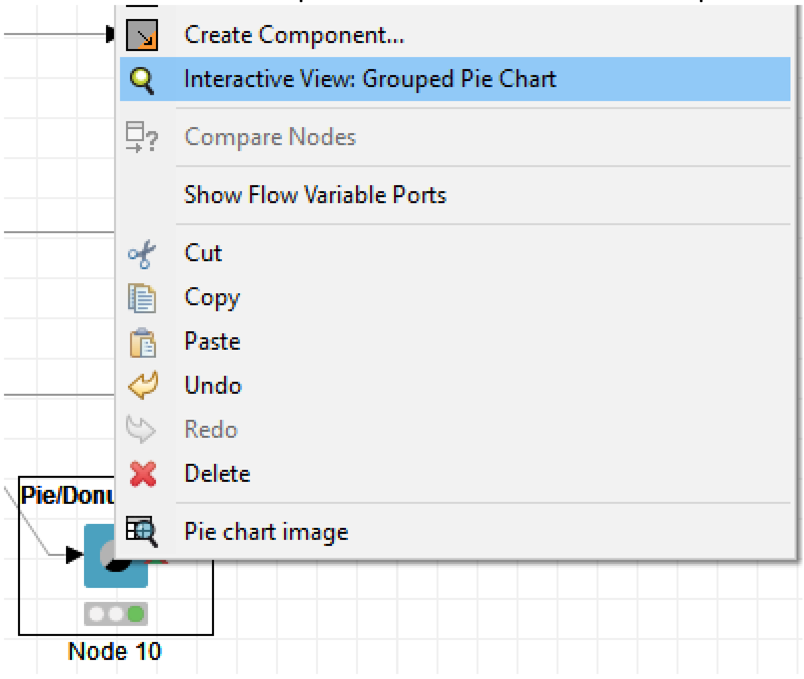
Image width: 600

Image height: 400

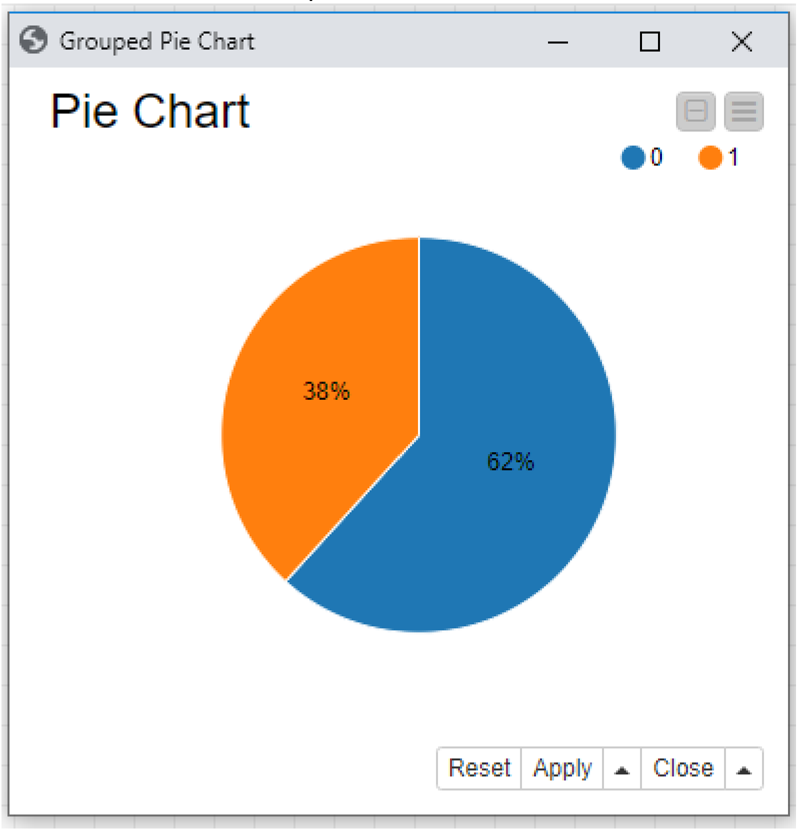
[✓] Scale view to window size

[✓] Show warnings in view

33. Execute the node and open the Interactive View: Grouped Pie Chart:

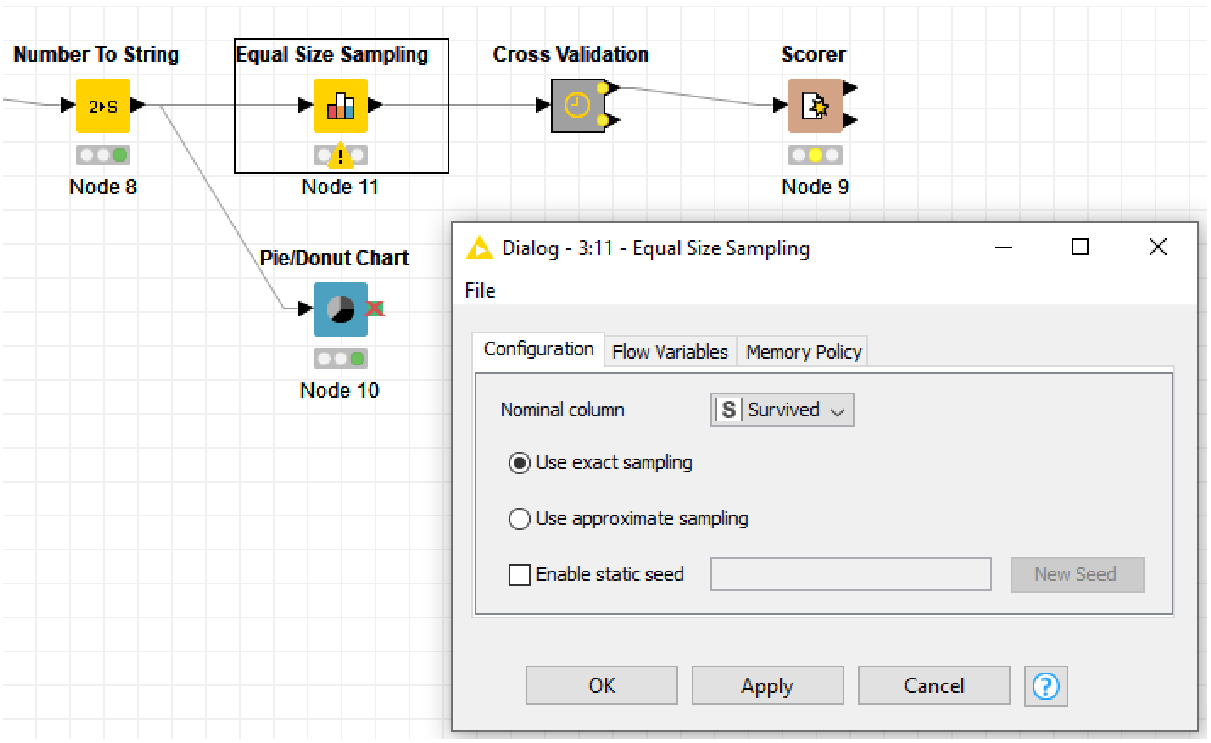


This is the visualization produced:



We see that indeed, the label column has imbalanced data i.e. the survivors and non-survivors do not partition the data into 50-50 proportion. Thankfully, KNIME has a node to help us do equal size sampling.

34. Remove the connection between the Number to String and Cross Validation nodes, and put an Equal Size Sampling node between them, before connecting the nodes again. Configure the Equal Size Sampling node as such:

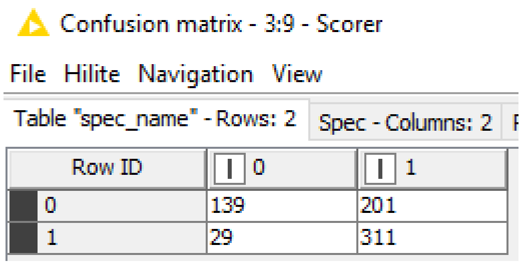


Nominal column: S Survived

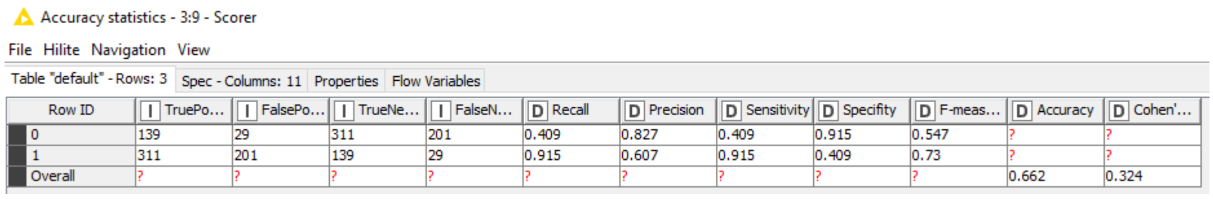
(● Use exact sampling) ○ Use approximate sampling

☐ Enable static seed

35. Run the workflow again and check the output of the Scorer node to see if there is an improvement.


Confusion matrix - 3:9 - Scorer
File Hilite Navigation View
Table "spec_name" - Rows: 2
Spec - Columns: 2

Row ID	0	1
0	139	201
1	29	311

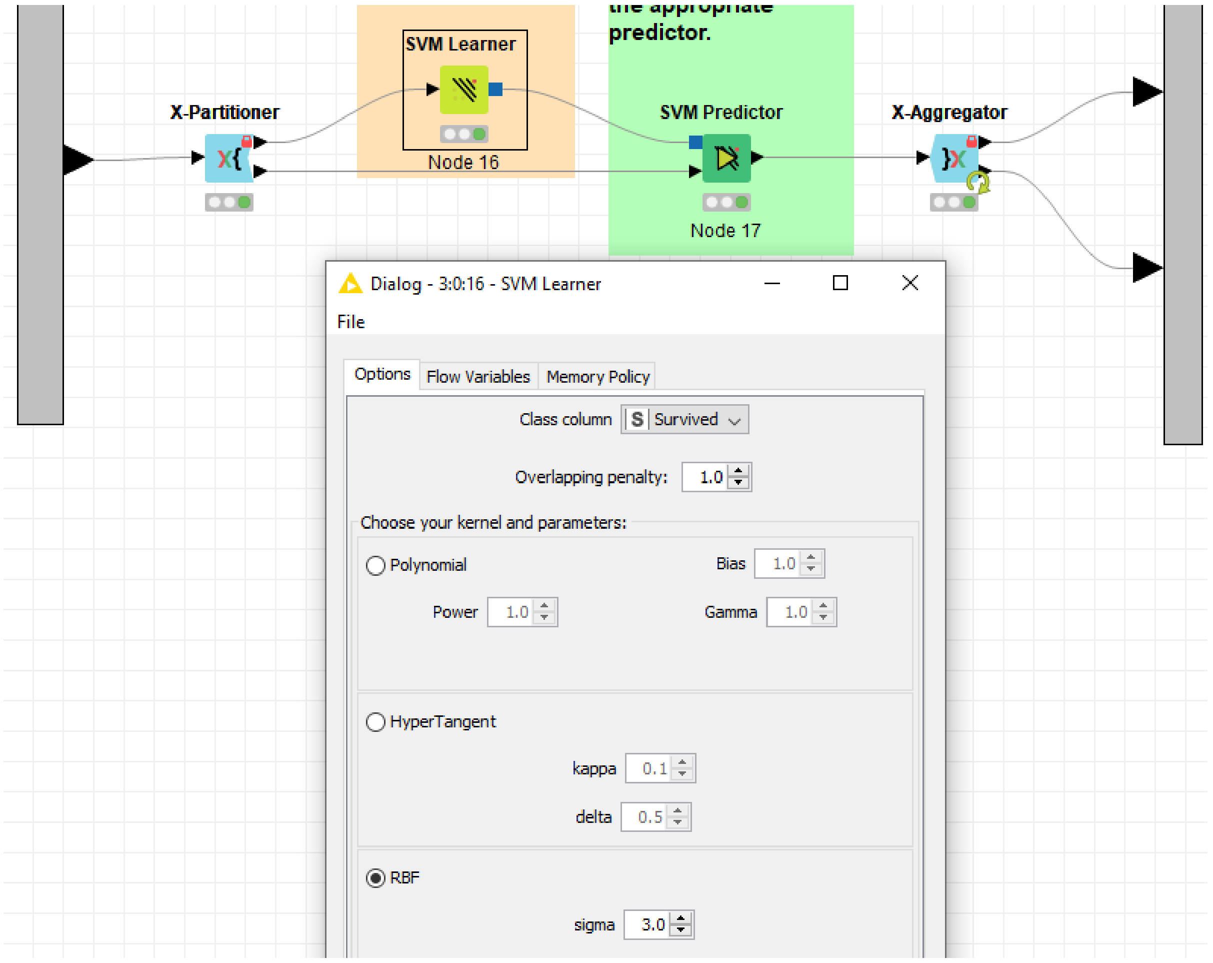

Accuracy statistics - 3:9 - Scorer
File Hilite Navigation View
Table "default" - Rows: 3
Spec - Columns: 11
Properties
Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
0	139	29	311	201	0.409	0.827	0.409	0.915	0.547	?	?
1	311	201	139	29	0.915	0.607	0.915	0.409	0.73	?	?
Overall	?	?	?	?	?	?	?	?	?	0.662	0.324

We note that there is indeed an improvement in the Recall and Sensitivity of the positive values for "Survived". The overall accuracy has also improved slightly.

To improve the accuracy of the model, we can do some parameter tweaking.

36. Enter the inner workflow of the Cross Validation node and change the SVM Learner's RBF kernel settings from the default sigma of 0.5 to a sigma of 3:



Class column: S Survived

Overlapping penalty: 1.0

Choose your kernel and parameters:

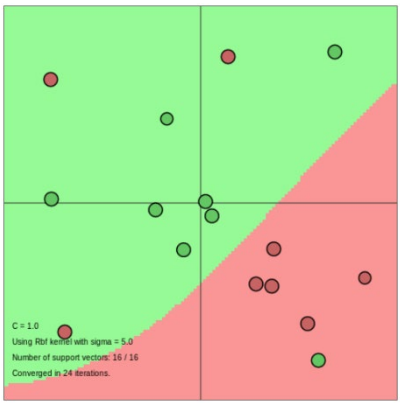
( ) Polynomial — Bias 1.0, Power 1.0, Gamma 1.0

( ) HyperTangent — kappa 0.1, delta 0.5

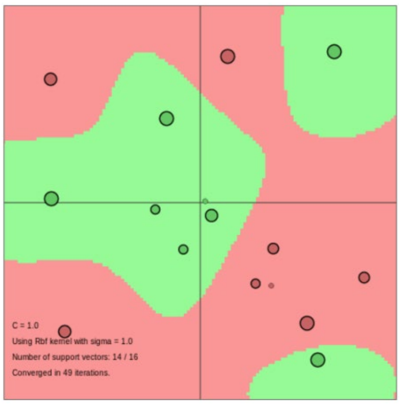
[✓ RBF] — sigma 3.0

About the RBF kernel of the SVM

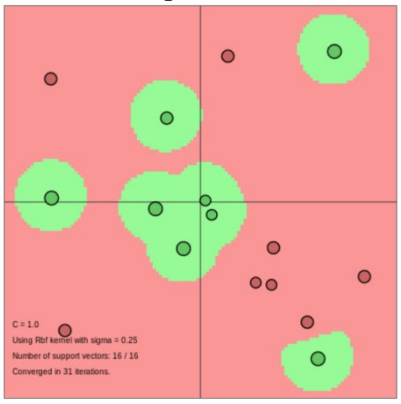
We can show how sigma affects the decision boundary qualitatively. Let's look at the following figures. The green circles are the positive labels while the red circles are the negative ones. The green and red sections are the prediction regions by the SVM according to the different sigma values.



Sigma = 5



Sigma = 1

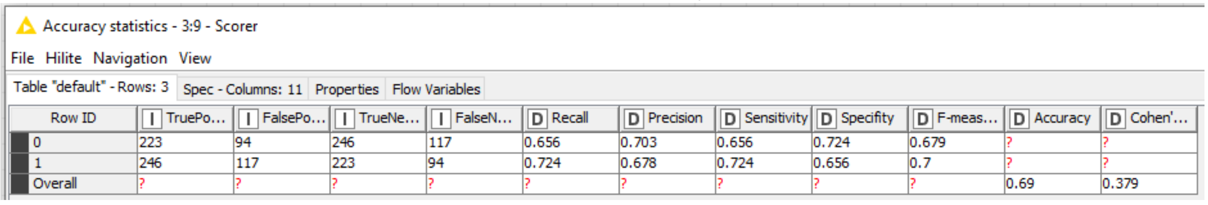


Sigma = 0.25

We can see from the SVM decision boundaries shown by sigma=5/2/0.25, that a larger sigma leads to a decision that tends to be flexible and smooth. It tends to make wrong classification while predicting, but avoids the hazard of overfitting. For a smaller sigma, the decision boundary tends to be strict and sharp, in contrast to the former situation, so it tends to overfit.

Source: http://haohanw.blogspot.com/2014/03/ml-how-sigma-matters-in-svm-rbf-kernel.html

37. Re-run the workflow and check the Scorer's output.


Accuracy statistics - 3:9 - Scorer

File Hilite Navigation View

Table "default" - Rows: 3 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specifity	D F-meas...	D Accuracy	D Cohen'...
0	223	94	246	117	0.656	0.703	0.656	0.724	0.679	?	?
1	246	117	223	94	0.724	0.678	0.724	0.656	0.7	?	?
Overall	?	?	?	?	?	?	?	?	?	0.69	0.379

The overall accuracy improved from 66.2% to 69%, and the other accuracy statistics are now more balanced.

Conclusion

We have seen how to build and apply the SVM model on numerical data. In case of nominal data, we have also seen how convert them into numeric data before using SVM. Equal-size sampling usually result in better and more accurate outcomes.