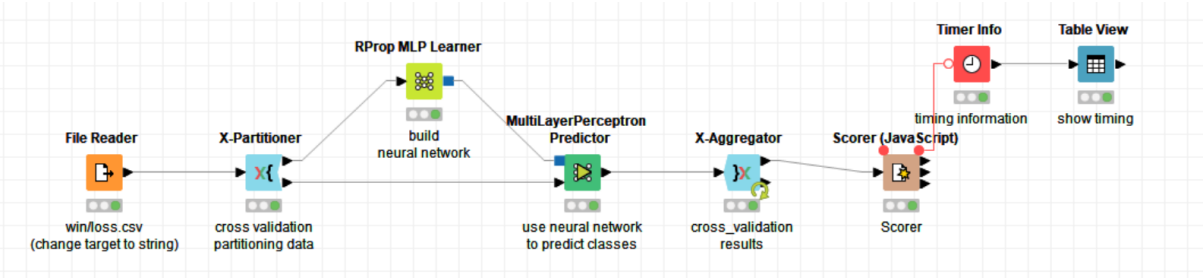
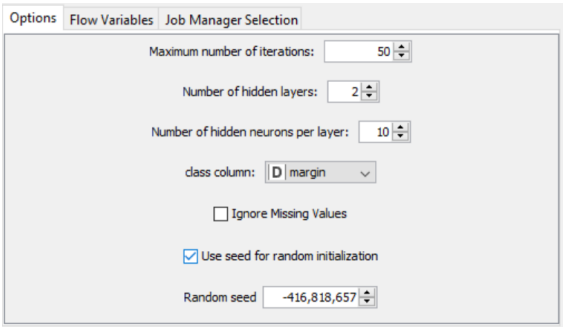
Question 1:

First, I build this workflow:



This workflow builds a neural network by its default setting



and predict by cross-validation with 5 folds. In partitioning, set the random seed to let each time's folding be the same. Let me show its performance by its error rate, score, and timing.

Error rate in each folder:

Row ID	D Error in %	I Size of Test Set	I Error Count
fold 0	18.575	393	73
fold 1	20.611	393	81
fold 2	23.469	392	92
fold 3	21.939	392	86
fold 4	22.959	392	90

Scorer View

Confusion Matrix

Rows Number : 393	0 (Predicted)	1 (Predicted)	
0 (Actual)	155	42	78.68%
1 (Actual)	43	153	78.06%
	78.28%	78.46%	

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
78.37%	21.63%	0.567	308	85

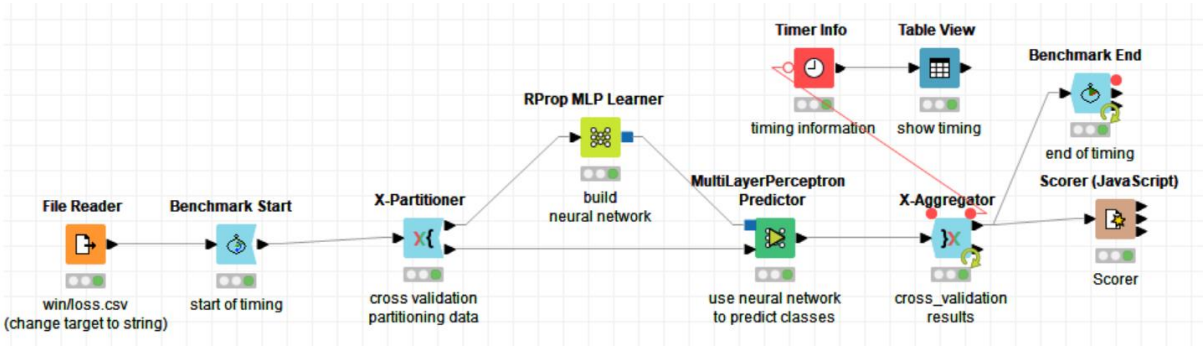
The overall Accuracy is 78.37%. It's similar for the accuracy to predict 1 and 0.

For timing, we can use the time info node, to show general timing information for each node,

RowID	Name	Execution Time	Execution Time since last Reset	Execution Time since Start	Nr of Executions since last Reset	Nr of Executions since Start	NodeID
Node 18	MultiLayerPerceptron Predictor	328	328	328	1	1	3:18
Node 19	RProp MLP Learner	812	812	815	1	3	3:19
Node 22	Parameter Optimization Loop Start	?	0	0	0	0	3:22
Node 23	Parameter Optimization Loop End	?	0	0	0	0	3:23
Node 48	File Reader	47	47	47	1	1	3:48
Node 101	Partitioning	12	12	28	1	2	3:101
Node 102	Scorer (JavaScript)	21	21	21	1	1	3:102
Node	X-Partitioner	8	54	1023	5	60	3:103

Reset Apply Clo

But I think what we need now is total execution time, since it is easier to compare. So, I changed the workflow:

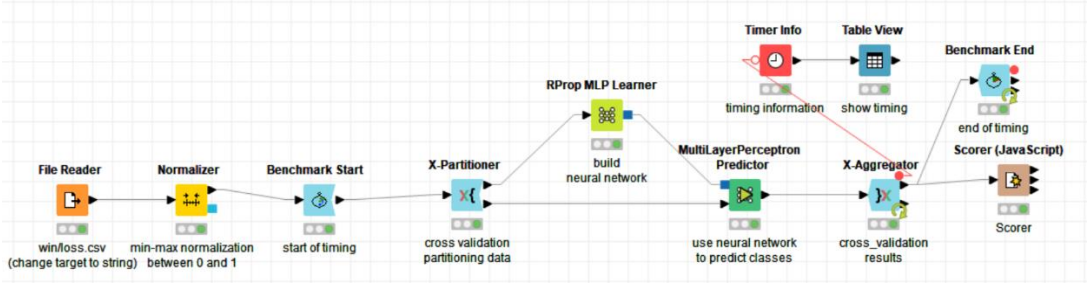


By the new nodes, benchmark start and benchmark end, it's easy to calculate the total execution time for the cross-validation and neural network.

Row ID	Iteration	Start Time	End Time	Executi...
Row_1	1	2020-10-05T00:21:18.038-04:00[America/New_Yo...	2020-10-05T00:21:20.051-04:00[America/New_Yo...	2.013

It's 2.013 seconds.

Then, add the normalization, between 0 and 1.



The result is:

Row ID	Error in %	Size of ...	Error C...
fold 0	11.196	393	44
fold 1	11.959	393	47
fold 2	9.439	392	37
fold 3	13.01	392	51
fold 4	9.949	392	39

(error rate in each folders)

Scorer View

Confusion Matrix

Rows Number : 1962	0 (Predicted)	1 (Predicted)	
0 (Actual)	882	99	89.91%
1 (Actual)	108	873	88.99%
	89.09%	89.81%	

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
89.45%	10.55%	0.789	1755	207

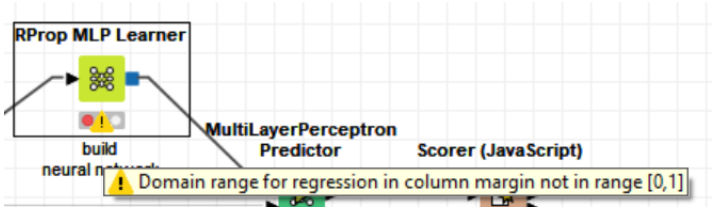
Row ID	Iteration	Start Time	End Time	Executi...
Row_1	1	2020-10-05T00:30:50-04:00[America/New_Yo...	2020-10-05T00:30:52.566-04:00[America/New_Yo...	2.566

Accuracy is 89.45%, it's still similar between the prediction accuracy between 1 and 0. The execution time is 2.566 seconds.

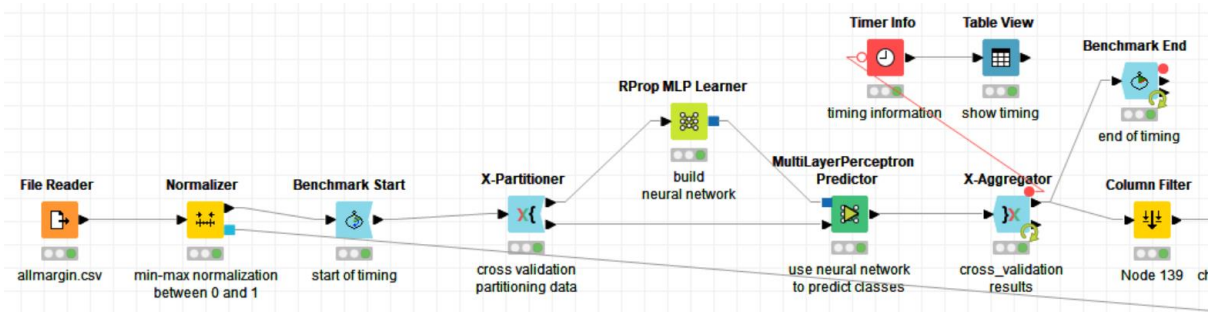
So, by the solution above, with normalization, it has a higher accuracy, but its computational cost is higher since it costs more time.

Question 2

The regression does not work if the data is not normalized for margin.csv.



KNIME says the range for regression should in [0,1]. So, I need to normalize the data.



This is similar as question 1. I read the data, normalize the data, start to calculate the time, since I want to calculate how much time the neutral network works. Then build the neural network, by default setting, and cross-validation with 5 folds.

Here is the result:

(for the result of cross validation)

Row ID	Total squared error	Mean squared error	Size of Test Set
fold 0	2.511	0.006	393
fold 1	2.119	0.005	393
fold 2	2.609	0.007	392
fold 3	2.024	0.005	392
fold 4	2.778	0.007	392

It calculates each fold's total squared error and mean square error.

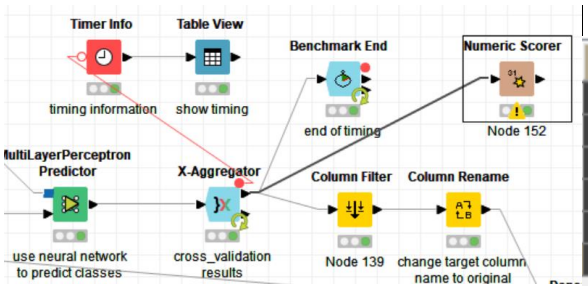
Row ID	Iteration	Start Time	End Time	Executi...
Row_1	1	2020-10-08T12:24:43.052-04:00[America/New_Yo...	2020-10-08T12:24:45.573-04:00[America/New_Yo...	2.521

For the timing, it is similar as classification with normalization.

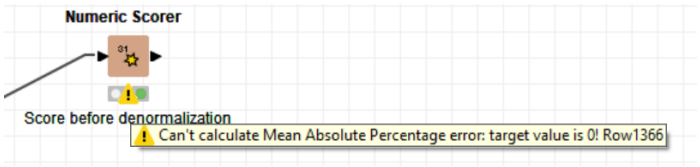
(prediction result)

margin	Predicti...
0.616	0.627
0.545	0.573
0.518	0.463
0.696	0.553
0.688	0.598
0.518	0.378
0.509	0.593

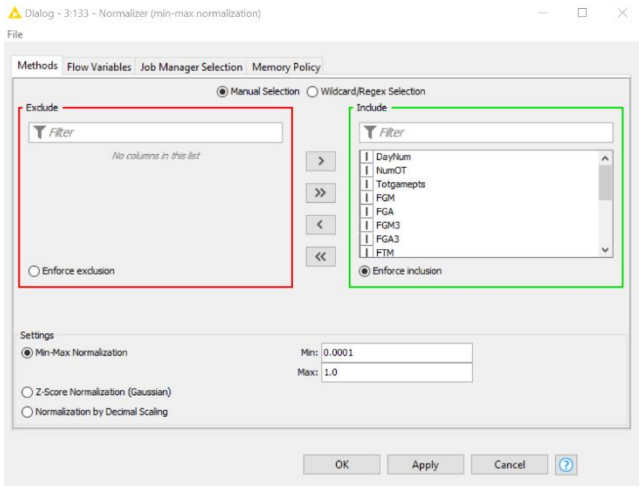
Since we did normalization for the prediction, we can do the denormalization for Prediction column to get the numerical result.



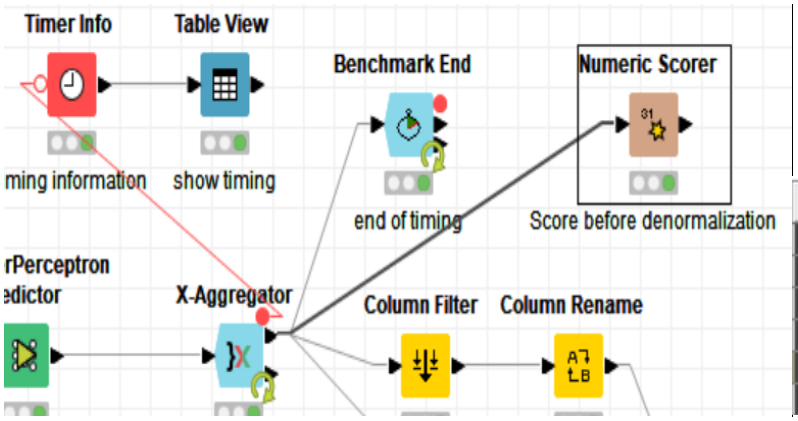
Row ID	Prediction (margin)
R^2	0.624
mean absolute error	0.06
mean squared error	0.006
root mean squared error	0.078
mean signed difference	-0.001
mean absolute percentage error	NaN



I add a numerical scorer after our prediction. There is an error with this node because it cannot calculate its mean absolute percentage error. Mean absolute error tells us the average size of error relates to the actual value. So I want to calculate it. To solving this problem, I change the scaler from 0.0001 to 1.



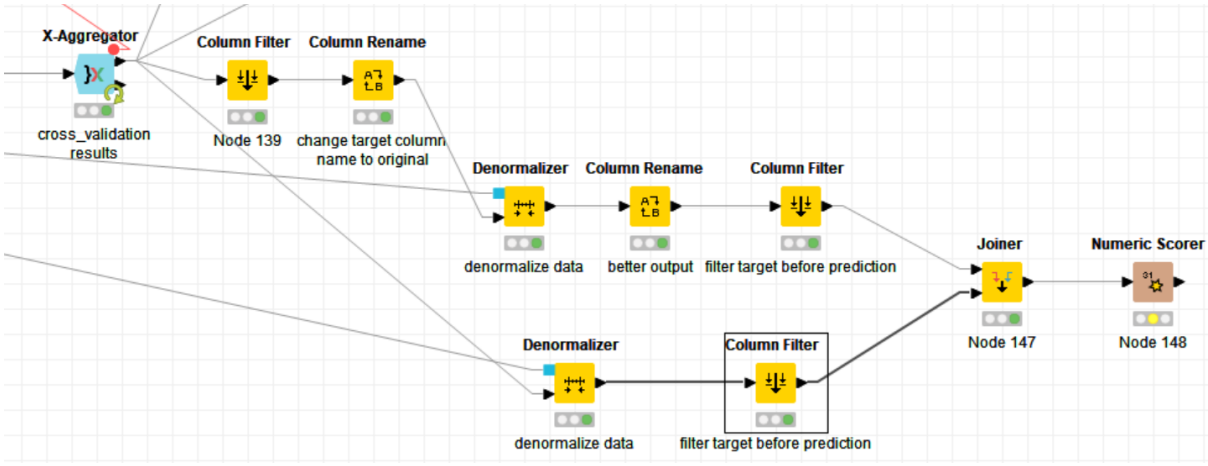
Then it works.



Row ID	Predicti...
R^2	0.642
mean absolut...	0.059
mean square...	0.006
root mean sq...	0.076
mean signed ...	0.001
mean absolut...	1.683

It has root mean square error 0.076 before denormalization.

Also, I'm trying do the denormalization, and see the score again.



On the top of the joiner, these nodes denormalized the target column, and select the denormalized predicted target column. (this is complex since I need to satisfy the Denormalizer node's conditions, to make sure the column names are the same before denormalization)

On the bottom, I select the original margin column, to make a comparison. Here is the output:

Row ID	Predicti...	margin
Row2	14.203	13
Row8	8.169	5
Row17	-4.155	2
Row22	5.929	22
Row29	10.941	21
Row30	-13.703	2
Row33	10.385	1
Row34	13.626	11
Row37	10.124	9
Row38	8.028	8
Row45	0.311	22
Row51	1.834	6
Row53	-10.633	2
Row54	-4.795	11
Row69	18.754	27
Row70	0.337	15
Row90	13.389	19
Row95	13.453	12
Row100	17.074	19
Row109	-0.387	4
Row110	-3.814	1
Row116	1.639	4

Row ID	Predicti...
Row2	14.203
Row8	8.169
Row17	-4.155
Row22	5.929
Row29	10.941
Row30	-13.703
Row33	10.385
Row34	13.626
Row37	10.124
Row38	8.028
Row45	0.311
Row51	1.834

Row ID	margin
Row2	13
Row8	5
Row17	2
Row22	22
Row29	21
Row30	2
Row33	1
Row34	11
Row37	9
Row38	8
Row45	22
Row51	6
Row53	2
Row54	11
Row69	27
Row70	15
Row90	19
Row95	12

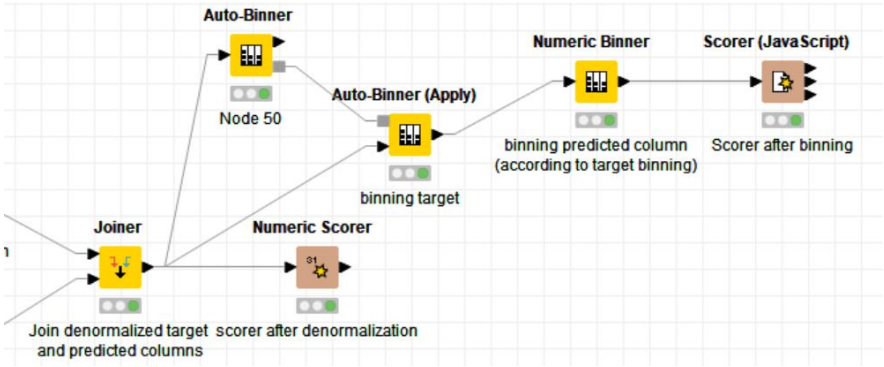
So, by the normalization and denormalization, I finished to predict the output with numerical values. Then I run the scorer.

Row ID	Prediction(margin)
R^2	0.624
mean absolute error	6.718
mean squared error	76.983
root mean squared error	8.774
mean signed difference	-0.058
mean absolute percentage error	1.116

Now it's our result for numerical scorer.

So the regression has root mean squared error with 8.774, the classifier has accuracy 89.45% with normalizer. To do the question 'Does regression outperform classification', I need to use grouping to compare the result with classifier, since we cannot compare mean squared error with classification accuracy. So that I'm trying to grouping target and predicted result by width or frequency into 2 groups.

(grouping by width)



Row ID	S Predicti...	S margin
Row2	(0,56]	(0,56]
Row8	(0,56]	(0,56]
Row17	[-56,0]	(0,56]
Row22	(0,56]	(0,56]
Row29	(0,56]	(0,56]
Row30	[-56,0]	(0,56]
Row33	(0,56]	(0,56]
Row34	(0,56]	(0,56]
Row37	(0,56]	(0,56]
Row38	(0,56]	(0,56]
Row45	(0,56]	(0,56]
Row51	(0,56]	(0,56]
Row53	[-56,0]	(0,56]
Row54	[-56,0]	(0,56]
Row69	(0,56]	(0,56]
Row70	(0,56]	(0,56]
Row90	(0,56]	(0,56]
Row95	(0,56]	(0,56]
Row100	(0,56]	(0,56]
Row109	[-56,0]	(0,56]
Row110	[-56,0]	(0,56]
Row116	(0,56]	(0,56]
Row120	(0,56]	(0,56]
Row123	[-56,0]	(0,56]
Row127	(0,56]	(0,56]

Scorer View

Confusion Matrix

Rows Number : 1962	(0,56] (Predicted)	[-56,0] (Predicted)	
(0,56] (Actual)	837	144	85.32%
[-56,0] (Actual)	129	852	86.85%
	86.65%	85.54%	

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
86.09%	13.91%	0.722	1689	273

The overall accuracy is 86.09%.

(grouping by frequency, similar as above)

Scorer View

Confusion Matrix

Rows Number : 1962	(-1,56] (Predicted)	[-56,-1] (Predicted)	
(-1,56] (Actual)	858	123	87.46%
[-56,-1] (Actual)	162	819	83.49%
	84.12%	86.94%	

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
85.47%	14.53%	0.709	1677	285

The overall accuracy is 85.47%, lower than 86.09%.

Also, I need to change the normalizer to 0.0001-1 in classifier with normalization. Here is the result:

Scorer View

Confusion Matrix

Rows Number : 1962	0 (Predicted)	1 (Predicted)	
0 (Actual)	869	112	88.58%
1 (Actual)	106	875	89.19%
	89.13%	88.65%	

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
88.89%	11.11%	0.778	1744	218

It has overall accuracy 88.89%. So, by comparing with the overall accuracy, Classifier with normalizer has a higher performance than regression.

In question 1, the accuracy for classifier without normalizer is 78.37% accuracy. So, classifier without normalizer has a lower performance than regression.

(performance: classifier without normalizer < regression (grouping by frequency) < regression (grouping by width) < classifier with normalizer)

Summary for question 2:

By compare with accuracy, regression outperforms classifier when classifier does not have normalizer, but classifier outperforms regression when classifier has normalizer. The timing is similar as classification with normalization, but it is longer than classification without normalization.

Appendix: Overall workflow in question 2, with explanations on the nodes.