



Machine Predictive Maintenance Classification

Names:	ID number:	Task distribution:
Ruyuf alharbi	44510064	<p>Leader/Supervisor. Creating&Editing the report.</p> <p>Writing the introduction, the goal, and the conclusion.</p> <p>Applied three algorithms:</p> <ol style="list-style-type: none"> 1- Randomforset 2- Naïve bayes 3- K- nearest
Alaa Abdulrahman ajlan	445013943	<p>Cleaned the data.</p> <p>Applied three algorithms:</p> <ol style="list-style-type: none"> 1- Logisitc regression. 2- OneR. 3- Decision tree.
Dana Hassan Bafaqih	445013918	<p>Cleaned the data.</p> <p>Applied three algorithms:</p> <ol style="list-style-type: none"> 1- DecisionStump. 2- AdaBoost. 3- DecisionTable
Hadeel Mohammad Turki	445006064	<p>Created the accuracy graph.</p> <p>Applied three algorithms:</p> <ol style="list-style-type: none"> 1- REP tree. 2- Bayes Net. 3- SimpleLogistic.
Refan Salem Almfariji	445000906	<p>Writing about the ZeroR.</p> <p>Applied three algorithms:</p> <ol style="list-style-type: none"> 1- SMO. 2- Lmt.

Table of content

The introduction:	4
The goal:	4
1) The original dataset information:	5
2) Data cleaning preparation:	7
Machine learning algorithms and methods:	13
1) ZeroR – Zero Rule Classifier	13
2) SMO - Sequential Minimal Optimization	14
3) JRip – Java Repeated Incremental Pruning	16
4) LMT – Logistic Model Tree	17
5) DecissionStump:	19
6) AdaBoostM1:	23
7) DecisionTable:	28
8) Logistic Regression (LR)	35
9) (J48) Tree :	40
10) OneR:	46
11) Simple logistic	50
12) Bayes Network	53
13) REP tree	56
14) Random Forest:	60
15) Naive Bayes	62
14) IBk (K-Nearest Neighbors)	65
The graph	67
<i>A comparison of all the algorithms with the best accuracy</i>	67
The conclusion:	67
References:	68

The introduction:

Predicting when a machine will fail is important to avoid costly repairs and downtime. With data from sensors like temperature, vibration, and pressure, we can use machine learning to spot signs of failure before they actually happen. In This project each one tested three different algorithms to see which one works best at predicting machine failures, helping to keep machines running smoothly and reduce unexpected problems.

The goal:

The goal of this project is to find the best algorithm to predict if a machine will fail or not. We cleaned the data and wanted to build a model that can clearly tell which machines might fail. This helps plan maintenance early and avoid surprise breakdowns.

1)The original dataset information:

Machine Predictive Maintenance Classification Dataset

Real predictive maintenance datasets are hard to get and share, so this dataset is synthetic but designed to closely reflect real industrial scenarios.

- The dataset has **10,000 rows** (data points) and **14 features** (columns).

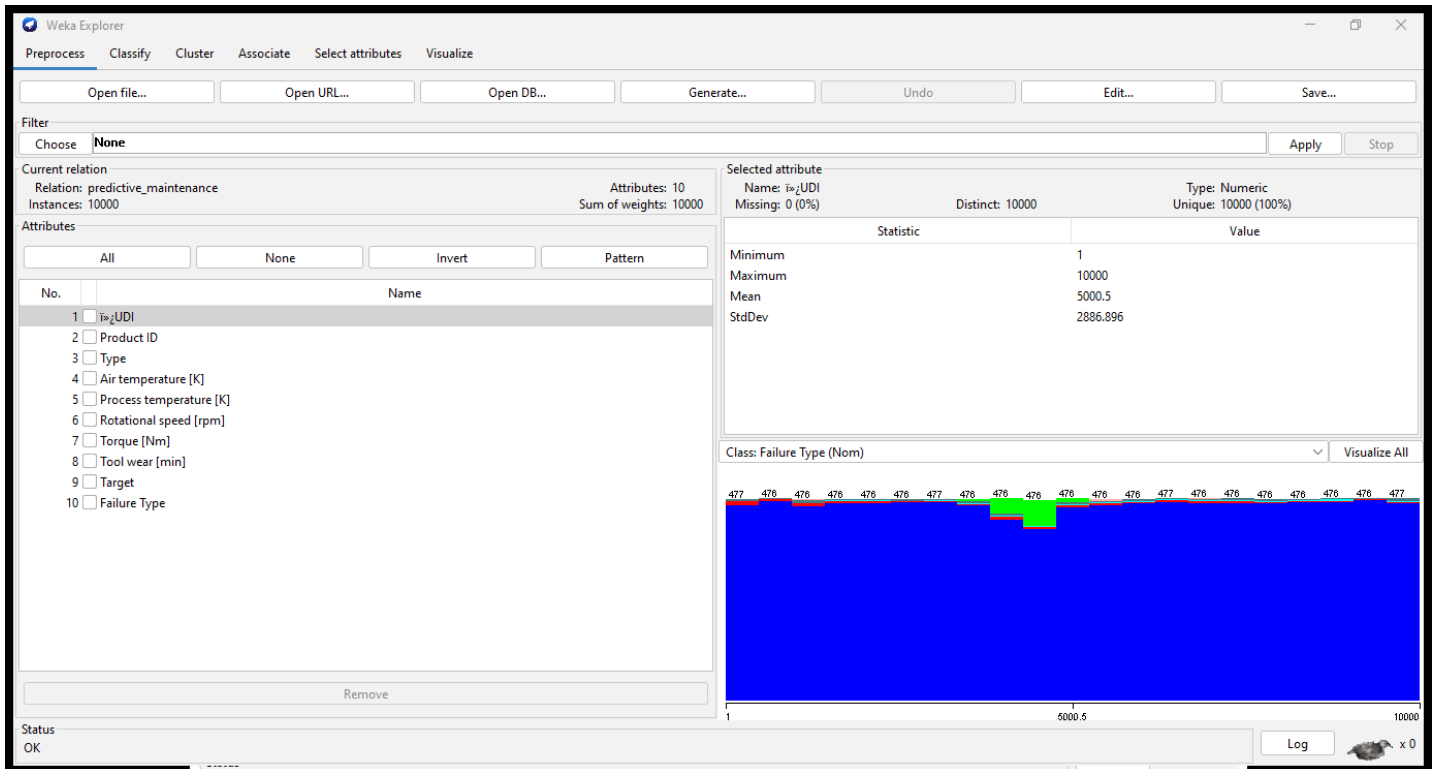
Features include:

- **UID:** Unique identifier from 1 to 10,000.
- **productID:** Product quality variant labeled as L (low, 50%), M (medium, 30%), or H (high, 20%), followed by a serial number.
- **Air temperature [K]:** Generated by a random walk and normalized with a standard deviation of 2 K around 300 K.
- **Process temperature [K]:** Based on air temperature plus 10 K, with additional noise normalized at 1 K.
- **Rotational speed [rpm]:** Calculated from power of 2860 W with added normal noise.
- **Torque [Nm]:** Normally distributed around 40 Nm ($\sigma = 10$ Nm), no negative values.
- **Tool wear [min]:** Quality variants add 5, 3, or 2 minutes of tool wear for H, M, and L respectively.
- **Machine failure label:** Indicates if the machine failed at this data point due to any failure mode.

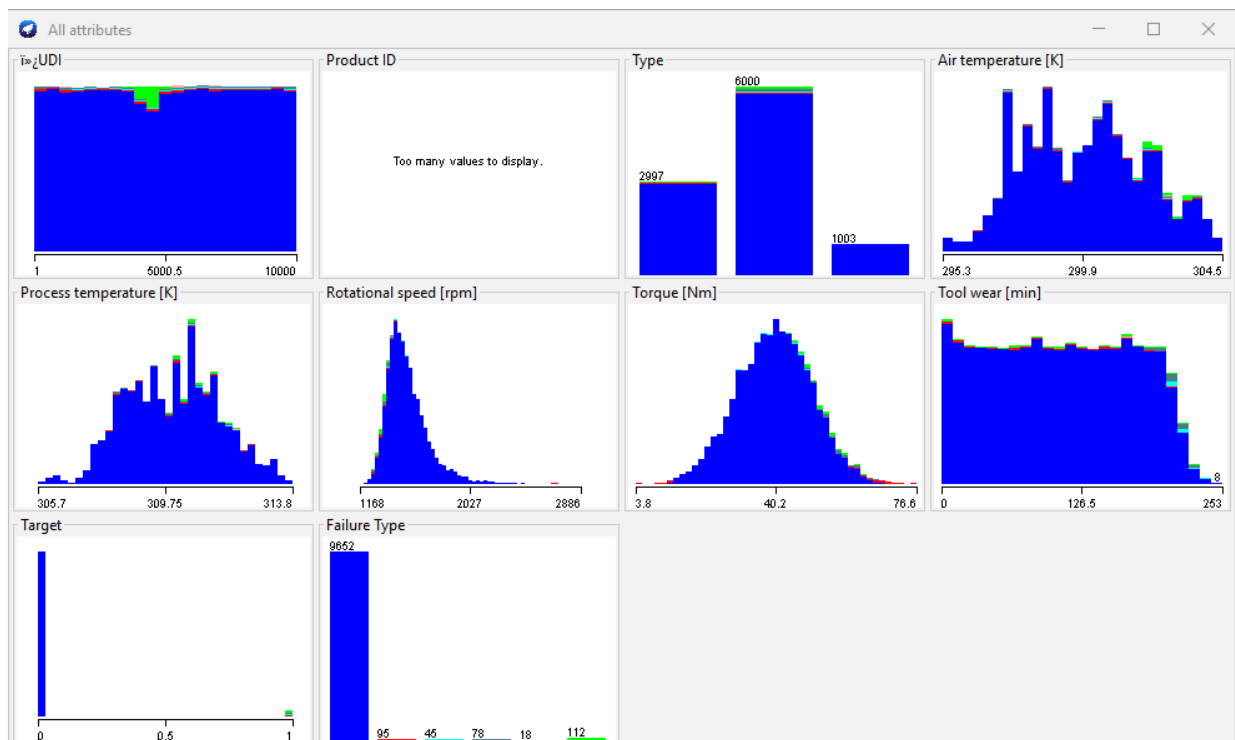
Important: The dataset contains **two target variables**:

- **Target:** Whether the machine failed or not.
- **Failure Type:** The type of failure.

- The dataset



- Visualization



2)Data cleaning preparation:

We applied some filter in order to clean and prepare the dataset for model training and evaluation:

- First, we removed the columns (UID, Product ID) that were not useful for the prediction process. This helped improve the model's performance and reduce complexity.
- We also removed the (failure type)column because we want to predict whether the machine will fail or not, and that column does not help with the prediction.

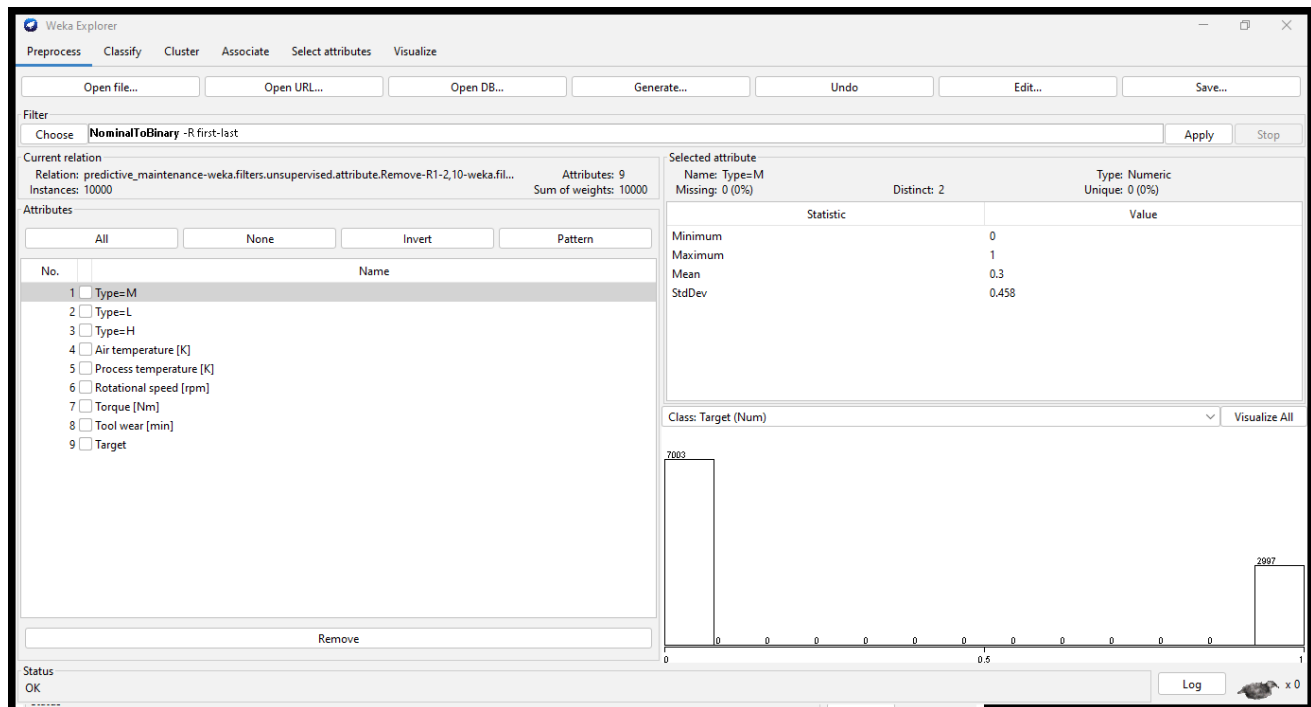
The top screenshot shows the Orange3 software interface with the 'NominalToBinary' filter applied. The 'Current relation' is 'predictive_maintenance' with 10000 instances. The 'Selected attribute' is 'Failure Type' (Nominal, 6 distinct values). The 'Attributes' list includes 'Failure Type' (checked). The 'Class: Failure Type (Nom)' bar chart shows a single bar of height 9652.

No.	Label	Count	Weight
1	No Failure	9652	9652
2	Power Failure	95	95
3	Tool Wear Failure	45	45
4	Overstrain Failure	78	78
5	Random Failures	18	18
6	Heat Dissipation Failure	112	112

The bottom screenshot shows the Orange3 software interface with the 'Remove' filter applied. The 'Current relation' is 'predictive_maintenance-weka.filters.unsupervised.attribute.Remove-R 1-2,10' with 10000 instances. The 'Selected attribute' is 'Type' (Nominal, 3 distinct values). The 'Attributes' list includes 'Type' (checked). The 'Class: Target (Num)' bar chart shows three bars of heights 2997, 6000, and 1003.

No.	Label	Count	Weight
1	M	2997	2997
2	L	6000	6000
3	H	1003	1003

- We converted the "Type" column to binary to make it easier to handle. Additionally, some algorithms cannot work with categorical values directly and require numerical input for processing.

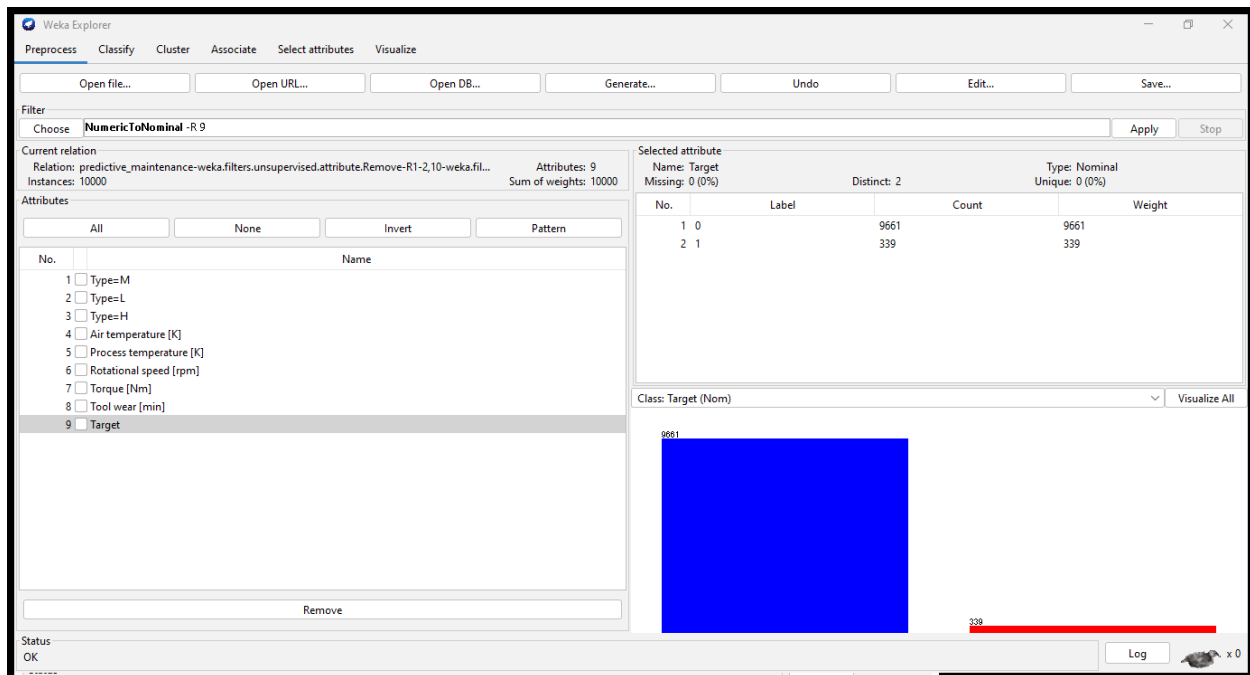


- The target column had numerical values, and since we want to use it for classification, we converted it to nominal.
- We chose only the target column

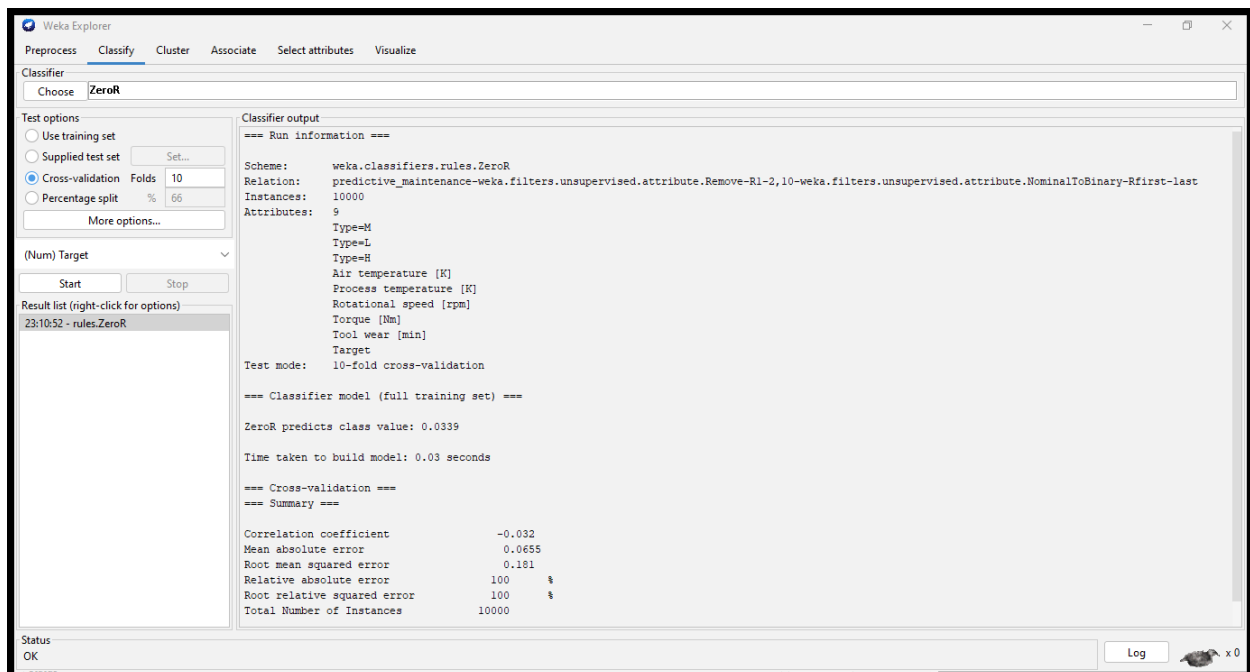
The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.filters.unsupervised.attribute.NumericToNominal' filter. The 'About' section describes the filter as 'A filter for turning numeric attributes into nominal ones.' Below this, there are four configuration options:

- attributeIndices:** 9
- debug:** False
- doNotCheckCapabilities:** False
- invertSelection:** False

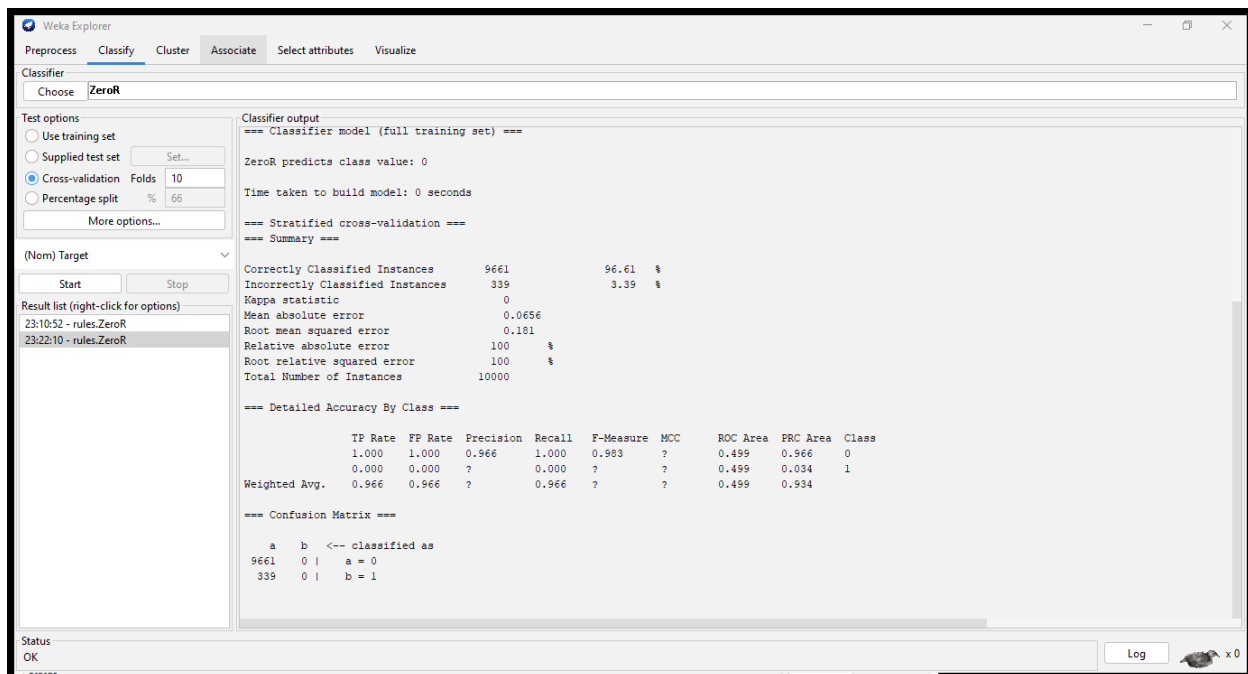
At the bottom, there are buttons for 'Open...', 'Save...', 'OK', and 'Cancel'.



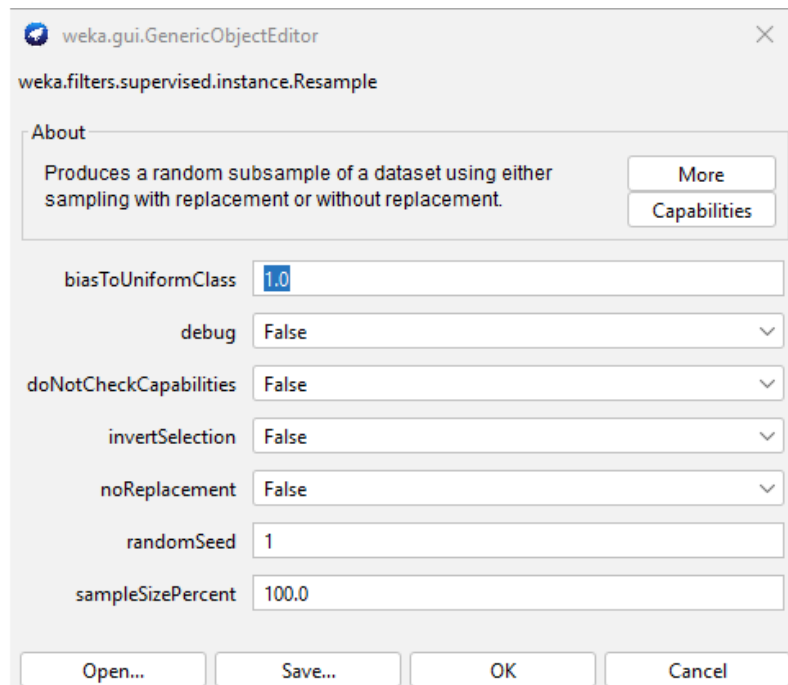
- The result before converting the column was treated as a regression problem, so the model predicted a number instead of a class.

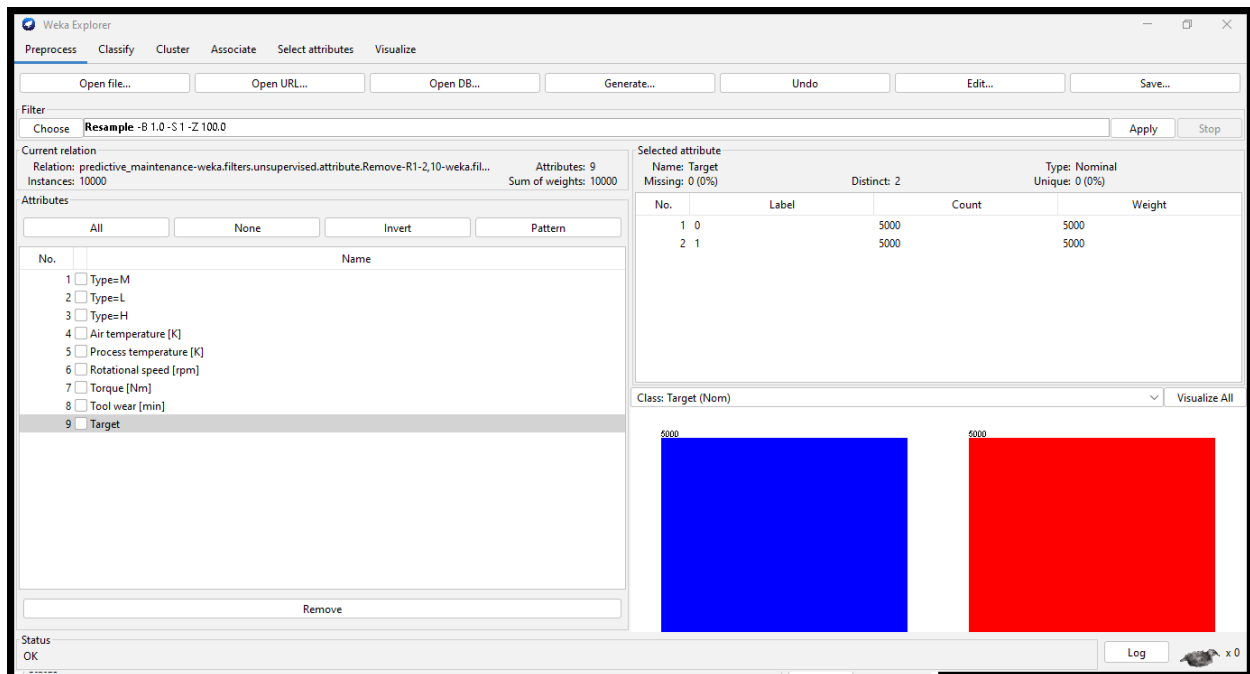


- The result after the conversion

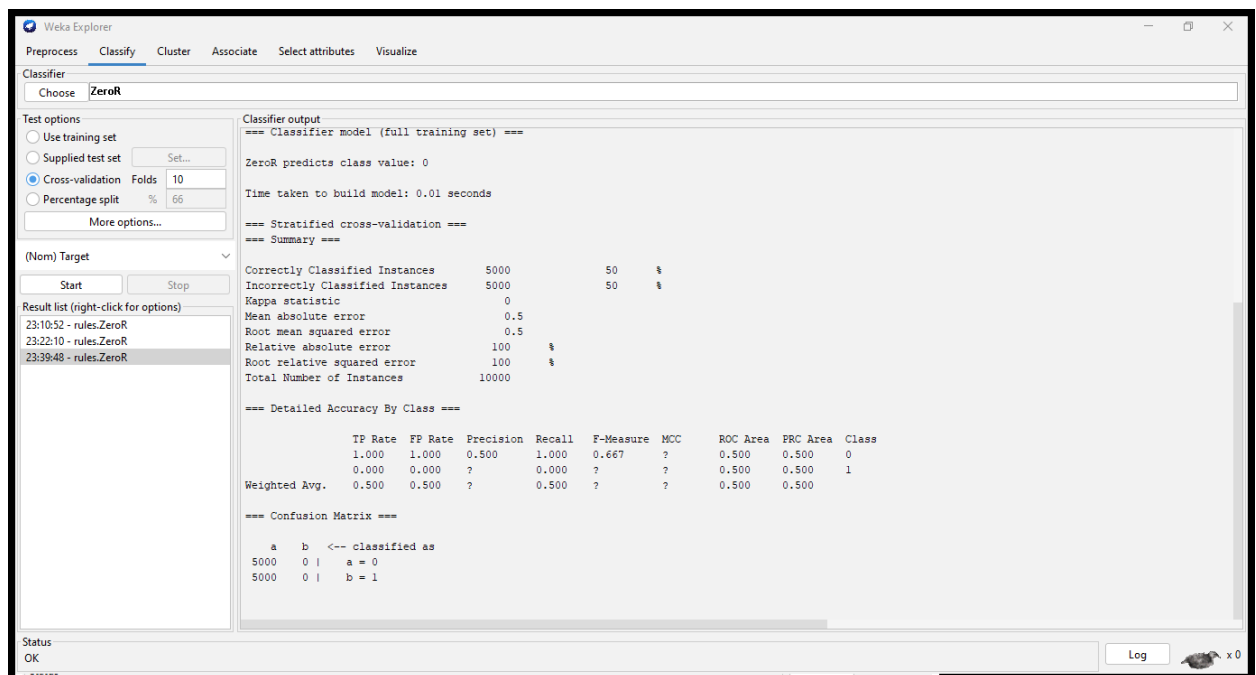


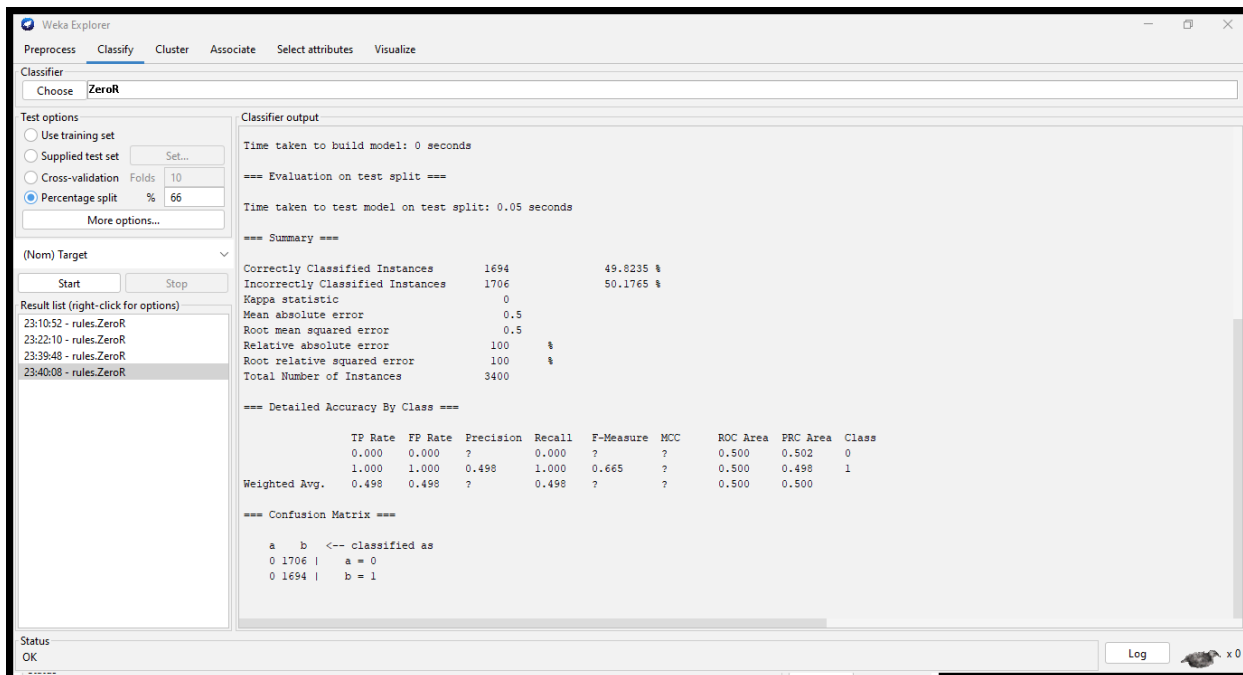
- As we can see, our data is imbalanced, and this causes a problem in classification. To solve this, we used the **Resample filter**, which helps make the data more balanced.
 - **Bias to Uniform Class = 1**, which means it will try to make the class distribution equal.
 - **Sample Size = 100%**, so the number of instances stays the same.
 - **No Replacement = false**, which means Weka is allowed to repeat some data.





- The result after the cleaning:





We did not perform normalization or discretization on the dataset manually, because during our research we found that these preprocessing steps can be applied directly through the algorithm's parameter settings in Weka if needed. Therefore, we relied on the algorithm's internal options instead of applying them externally.

Machine learning algorithms and methods:

1) ZeroR – Zero Rule Classifier

ZeroR is the simplest classification algorithm. It ignores all input features and always predicts the majority class in the training data. It is mainly used as a baseline model to evaluate the performance of more advanced classifiers.

Experimental Result:

In our experiment, ZeroR predicted only class 0 for all instances. As a result, it achieved exactly 50% accuracy, since the dataset was balanced (50% class 0, 50% class 1). The confusion matrix confirmed that all 5000 instances of class 0 were classified correctly, while all 5000 instances of class 1 were misclassified

The screenshot displays the Weka GUI with the ZeroR classifier selected. The 'weka.classifiers.rules.ZeroR' dialog box is open, showing settings for batchSize (100), debug (False), doNotCheckCapabilities (False), and numDecimalPlaces (2). Below the dialog, the main Weka window shows the 'Classifier' tab with ZeroR selected. The 'Test options' section shows 'Cross-validation' with 'Folds' set to 10. The 'Result list' on the left shows a list of classifiers, with '12:26:24 - rules.ZeroR' selected. The 'Classifier output' pane on the right displays the following results:

```
==== Classifier model (full training set) ====
ZeroR predicts class value: 0
Time taken to build model: 0 seconds
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      5000      50 %
Incorrectly Classified Instances    5000      50 %
Kappa statistic                    0
Mean absolute error                 0.5
Root mean squared error             0.5
Relative absolute error             100 %
Root relative squared error         100 %
Total Number of Instances          10000

==== Detailed Accuracy By Class ====
      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC   ROC Area  PRC Area  Class
      1.000    1.000    0.500     1.000    0.667    ?     0.500    0.500    0
      0.000    0.000    ?         0.000    ?         ?     0.500    0.500    1
Weighted Avg.   0.500    0.500    ?         0.500    ?         ?     0.500    0.500

==== Confusion Matrix ====
      a      b  <-- classified as
5000   0 |   a = 0
5000   0 |   b = 1
```

2) SMO - Sequential Minimal Optimization

SMO trains an SVM model by breaking down the large optimization problem into a series of smaller problems involving only two variables at a time. Each small problem is solved analytically, without the need for complex iterative methods. This process is repeated until the entire model is trained efficiently

- The default parameter:

The screenshot displays the Weka GUI's 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.functions.SMO' classifier. The 'About' tab is active, showing the description: 'Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.' Below this, various parameters are configured: 'batchSize' is 100, 'buildCalibrationModels' is False, 'c' is 2.0, 'calibrator' is set to 'Logistic -R 1.0E-8 -M -1 -num-decimal-places 4', 'checksTurnedOff' is False, 'debug' is False, 'doNotCheckCapabilities' is False, 'epsilon' is 1.0E-4, 'filter Type' is 'Standardize training data', 'kernel' is 'PolyKernel -E 2.0 -C 250007', 'numDecimalPlaces' is 2, 'numFolds' is -1, 'randomSeed' is 1, and 'toleranceParameter' is 0.001.

Below the parameter settings, the 'Test options' section shows 'Cross-validation' selected with 'Folds' set to 10. The 'Classifier output' tab is active, displaying the following information:

```
+ 10.5468 * (normalized) Rotational speed [rpm]
+ 11.4742 * (normalized) Torque [Nm]
+ 2.3604 * (normalized) Tool wear [min]
- 10.6892
```

Number of kernel evaluations: 15067320 (49.314% cached)

Time taken to build model: 3.2 seconds

=== Stratified cross-validation ===
=== Summary ===

Metric	Value	%
Correctly Classified Instances	8167	81.67
Incorrectly Classified Instances	1833	18.33
Kappa statistic	0.6224	
Mean absolute error	0.1893	
Root mean squared error	0.4201	
Relative absolute error	36.66	%
Root relative squared error	85.6271	%
Total Number of Instances	10000	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	FRC Area	Class
0	0.021	0.107	0.014	0.021	0.017	0.633	0.017	0.756	0
1	0.813	0.175	0.815	0.813	0.816	0.633	0.817	0.759	1
Weighted Avg.	0.817	0.183	0.817	0.817	0.817	0.622	0.817	0.759	

=== Confusion Matrix ===

a \ b	0	1	<-- classified as
4103 897	1	0	a = 0
536 4064	0	1	b = 1

Best Accuracy Result:

SMO achieved its highest accuracy (**93.95%**) after changing the **kernel** to **PUK**, applying the **Normalize filter** to the training data, and **increasing the number of folds**

weka.gui.GenericObjectEditor

weka.classifiers.functions.SMO

About

Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier. [More](#) [Capabilities](#)

batchSize 100

buildCalibrationModels False

c 2.0

calibrator Choose **Logistic -R 1.0E-8 -M -1 -num-decimal-places 4**

checksTurnedOff False

debug False

doNotCheckCapabilities False

epsilon 1.0E-4

filterType **Normalize training data**

kernel Choose **Puk -O 1.0 -S 1.0 -C 250007**

numDecimalPlaces 2

numFolds 5

randomSeed 1

toleranceParameter 0.001

Open... Save... OK Cancel

Classifier

Choose **SMO -C 2.0 -L 0.001 -P 1.0E-4 -N 0 -V 5 -W 1 -K "weka.classifiers.functions.supportVector.Puk -O 1.0 -S 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"**

Test options

☐ Use training set

☐ Supplied test set

☒ Cross-validation Folds 10

☐ Percentage split % 66

More options...

Classifier output

+ 0.0966 * <0 1 0 0.648132 0.7 0.105355 0.649725 0.035573 > * X]

+ 1.8375

Number of support vectors: 3526

Number of kernel evaluations: 82913086 (16.208% cached)

(Nom) Target

Start Stop

Result list (right-click for options)

22:45:52 - functions.SMO

22:57:06 - functions.SMO

23:06:39 - functions.SMO

23:11:41 - functions.SMO

23:12:12 - functions.SMO

Time taken to build model: 10.74 seconds

=== Stratified cross-validation ===

=== Summary ===

	Correctly Classified Instances	9395	93.95 %
Incorrectly Classified Instances	605	6.05 %	
Kappa statistic	0.879		
Mean absolute error	0.0405		
Root mean squared error	0.246		
Relative absolute error	12.1 %		
Root relative squared error	49.1935 %		
Total Number of Instances	10000		

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.914	0.035	0.963	0.914	0.938	0.880	0.940	0.923	0
	0.965	0.086	0.918	0.965	0.941	0.880	0.940	0.904	1
Weighted Avg.	0.940	0.061	0.941	0.940	0.939	0.880	0.940	0.913	

=== Confusion Matrix ===

	a	b	<-- classified as
4570 430	a = 0		
175 4825	b = 1		

Test options

☐ Use training set

☐ Supplied test set

☒ Cross-validation Folds 10

☐ Percentage split % 66

More options...

(Nom) Target

Start Stop

Result list (right-click for options)

2:45:52 - functions.SMO

2:57:06 - functions.SMO

3:06:39 - functions.SMO

3:11:41 - functions.SMO

3:16:12 - functions.SMO

3:24:27 - meta.Stacking

3:30:50 - rules.JRip

3:37:10 - rules.JRip

3:41:01 - rules.JRip

3:42:40 - rules.JRip

0:24:42 - trees.LMT

0:28:12 - trees.LMT

4:43:30 - rules.JRip

2:26:24 - rules.ZeroR

1:52:15 - functions.SMO

Classifier output

Number of support vectors: 3526

Number of kernel evaluations: 82913086 (16.208% cached)

Time taken to build model: 12.37 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.74 seconds

=== Summary ===

	Correctly Classified Instances	3189	93.7941 %
Incorrectly Classified Instances	211	6.2059 %	
Kappa statistic	0.8759		
Mean absolute error	0.0621		
Root mean squared error	0.2491		
Relative absolute error	12.4117 %		
Root relative squared error	49.8228 %		
Total Number of Instances	3400		

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.899	0.022	0.576	0.899	0.936	0.879	0.938	0.928	0
	0.978	0.101	0.505	0.978	0.940	0.879	0.938	0.896	1
Weighted Avg.	0.938	0.062	0.541	0.938	0.938	0.879	0.938	0.912	

=== Confusion Matrix ===

	a	b	<-- classified as
1533 173	a = 0		
38 1656	b = 1		

3) JRip – Java Repeated Incremental Pruning

JRip is a rule-based classification algorithm that builds simple if-then rules to classify data. It improves accuracy by pruning unnecessary parts of the rules and is useful when clear and interpretable models are needed

- The default parameter:

The default JRip parameters already achieved high accuracy (98.64%). However, I made a slight adjustment to one of the parameters to further improve the performance

The screenshot shows the Weka GUI with the JRip classifier selected. The parameters are set to: batchSize: 100, checkErrorRate: True, debug: False, doNotCheckCapabilities: False, folds: 3, minNo: 2.0, numDecimalPlaces: 2, optimizations: 2, seed: 1, usePruning: True. The classifier output shows a stratified cross-validation summary with an accuracy of 98.64%.

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.973	0.000	1.000	0.973	0.986	0.973	0.991	0.994	0
1.000	0.027	0.974	1.000	0.987	0.973	0.991	0.984	1
Weighted Avg. 0.986 0.014 0.987 0.986 0.986 0.973 0.991 0.989								

Best Accuracy Result:

JRip achieved its highest accuracy (98.74%) after increasing the optimizations parameter , decreasing number of fold ,and decreasing numNO

The screenshot shows the Weka GUI with the JRip classifier selected. The parameters are set to: batchSize: 100, checkErrorRate: True, debug: False, doNotCheckCapabilities: False, folds: 5, minNo: 1.0, numDecimalPlaces: 2, optimizations: 3, seed: 1, usePruning: True. The classifier output shows a stratified cross-validation summary with an accuracy of 98.74%.

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.975	0.000	1.000	0.975	0.987	0.975	0.991	0.994	0
1.000	0.025	0.975	1.000	0.988	0.975	0.991	0.983	1
Weighted Avg. 0.987 0.013 0.988 0.987 0.987 0.975 0.991 0.989								

Test options

☐ Use training set
 ☐ Supplied test set

Set...

☐ Cross-validation

Folds 10

☒ Percentage split

% 66

More options...

(Nom) Target

▼

Start

Stop

Result list (right-click for options)

22:45:52 - functions.SMO

22:57:06 - functions.SMO

23:06:39 - functions.SMO

23:11:41 - functions.SMO

23:16:12 - functions.SMO

23:24:27 - meta.Sparkling

23:30:50 - rules.RIP

23:37:10 - rules.RIP

23:41:01 - rules.RIP

23:42:40 - rules.RIP

00:24:42 - trees.LMT

00:28:12 - trees.LMT

04:43:30 - rules.RIP

12:36:24 - rules.ZeroR

11:52:15 - functions.SMO

11:53:47 - rules.RIP

Classifier output

(Tool wear [min] <= 20) and (Tool wear [min] >= 20) and (Torque [Nm] >= 47.7) => Target=1 (29.0/2.0)

=> Target=0 (4933.0/0.0)

Number of Rules : 30

Time taken to build model: 3.61 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances335398.6176 %

Incorrectly Classified Instances471.3824 %

Kappa statistic0.9724

Mean absolute error0.0177

Root mean squared error0.1165

Relative absolute error3.5388 %

Root relative squared error23.2965 %

Total Number of Instances3400

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	FRC Area	Class
	0.974	0.002	0.998	0.974	0.986	0.973	0.987	0.987	0
	0.998	0.026	0.975	0.998	0.986	0.973	0.987	0.975	1
Weighted Avg.	0.986	0.014	0.986	0.986	0.986	0.973	0.987	0.981	

=== Confusion Matrix ===

a b <-- classified as

1662 44 | a = 0

3 1691 | b = 1

4) LMT – Logistic Model Tree

LMT is a hybrid classification algorithm that combines decision trees and logistic regression. It builds a tree where each leaf node contains a logistic regression model, allowing the algorithm to handle both non-linear and linear patterns in data effectively.

weka.gui.GenericObjectEditor
reka.classifiers.trees.LMT
About
Classifier for building 'logistic model trees', which are classification trees with logistic regression functions at the leaves.
More
Capabilities
batchSize 100
convertNominal False
debug False
doNotCheckCapabilities False
doNotMakeSplitPointActualValue False
errorOnProbabilities False
fastRegression True
minNumInstances 15
numBoostingIterations -1
numDecimalPlaces 2
splitOnResiduals False
useAIC False
weightTrimBeta 0.0
Open... Save... OK Cancel

Classifier
Choose LMT 4-1-44 15-4W 0.0
Test options
☐ Use training set
☐ Supplied test set
☒ Cross-validation Folds 10
☐ Percentage split % 66
More options...
(Nom) Target
Start Stop
Result list (right-click for options)
12:45:52 - functions.SMO
12:57:06 - functions.SMO
13:06:39 - functions.SMO
13:11:41 - functions.SMO
13:16:12 - functions.SMO
13:24:27 - meta.Stacking
13:30:50 - rules.JRip
13:37:10 - rules.JRip
13:41:01 - rules.JRip
13:42:40 - rules.JRip
13:45:36 - LMT
Classifier output
[Type=I] * 3.45 +
[Type=R] * 0.28 +
[Air temperature [K]] * 0 +
[Process temperature [K]] * -0.07 +
[Rotational speed [rpm]] * 0.03 +
[Torque [Nm]] * 0.52 +
[Tool wear [min]] * 0.12
Time taken to build model: 3.5 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 9906 99.06 %
Incorrectly Classified Instances 94 0.94 %
Kappa statistic 0.9812
Mean absolute error 0.0106
Root mean squared error 0.0956
Relative absolute error 2.1153 %
Root relative squared error 19.1608 %
Total Number of Instances 10000
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	FRC Area	Class
0	0.981	0.000	1.000	0.981	0.991	0.981	0.994	0.996	0
1	1.000	0.019	0.982	1.000	0.991	0.981	0.994	0.988	1
Weighted Avg.	0.991	0.009	0.991	0.991	0.991	0.981	0.994	0.992	

=== Confusion Matrix ===

a	b	-- classified as
4906	94	a = 0
0	3000	b = 1

Best Accuracy:

This was the best accuracy achieved in all experiments, reaching 99.1% after adjusting the minNumInstances parameter

The screenshot displays the Weka GUI with the LMT classifier selected. The configuration window on the left shows various parameters, with 'minNumInstances' set to 10. The classifier output window on the right shows the resulting model equation and performance metrics.

Classifier Configuration (Left Panel):

- Classifier: weka.classifiers.trees.LMT
- batchSize: 100
- convertNominal: False
- debug: False
- doNotCheckCapabilities: False
- doNotMakeSplitPointActualValue: False
- errorOnProbabilities: False
- fastRegression: True
- minNumInstances: 10
- numBoostingIterations: -1
- numDecimalPlaces: 2
- splitOnResiduals: False
- useAIC: False
- weightTrimBeta: 0.0

Classifier Output (Right Panel):

Test options:
☐ Use training set
☐ Supplied test set
☒ Cross-validation Folds: 10
☐ Percentage split %: 66

Classifier output:
[Type=1] * 3.45 +
[Type=0] * 0.28 +
[Air temperature [K]] * 0 +
[Process temperature [K]] * -0.07 +
[Rotational speed [rpm]] * 0.03 +
[Torque [Nm]] * 0.52 +
[Tool wear [min]] * 0.12

Time taken to build model: 3.16 seconds

==== Stratified cross-validation ====

==== Summary ====

Metric	Value	%
Correctly Classified Instances	9910	99.1
Incorrectly Classified Instances	90	0.9
Kappa statistic	0.982	
Mean absolute error	0.0097	
Root mean squared error	0.0944	
Relative absolute error	1.9396	%
Root relative squared error	18.8818	%
Total Number of Instances	10000	

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.982	0.000	1.000	0.982	0.991	0.982	0.994	0.996	0
	1.000	0.018	0.982	1.000	0.991	0.982	0.994	0.989	1
Weighted Avg.	0.991	0.009	0.991	0.991	0.991	0.982	0.994	0.992	

==== Confusion Matrix ====

a \ b	<-- classified as
4910	90 a = 0
0 5000	b = 1

The screenshot displays the Weka GUI with the LMT classifier selected. The classifier output window shows the resulting model equation and performance metrics.

Classifier Output:

[Process temperature [K]] * -0.07 +
[Rotational speed [rpm]] * 0.03 +
[Torque [Nm]] * 0.52 +
[Tool wear [min]] * 0.12

Time taken to build model: 3.1 seconds

==== Evaluation on test split ====

Time taken to test model on test split: 0 seconds

==== Summary ====

Metric	Value	%
Correctly Classified Instances	3371	99.1471
Incorrectly Classified Instances	29	0.8529
Kappa statistic	0.9829	
Mean absolute error	0.0098	
Root mean squared error	0.0916	
Relative absolute error	1.9651	%
Root relative squared error	18.3277	%
Total Number of Instances	3400	

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.983	0.000	1.000	0.983	0.991	0.983	0.994	0.996	0
	1.000	0.017	0.983	1.000	0.992	0.983	0.994	0.990	1
Weighted Avg.	0.991	0.008	0.992	0.991	0.991	0.983	0.994	0.993	

==== Confusion Matrix ====

a \ b	<-- classified as
1677	29 a = 0
0 1694	b = 1

5) DecisionStump:

The Decision Stump focuses on only **one feature** at a time and finds a threshold that best separates the data.

In our data, it makes the decision based on the '**Rotational speed**' feature.

```
Decision Stump

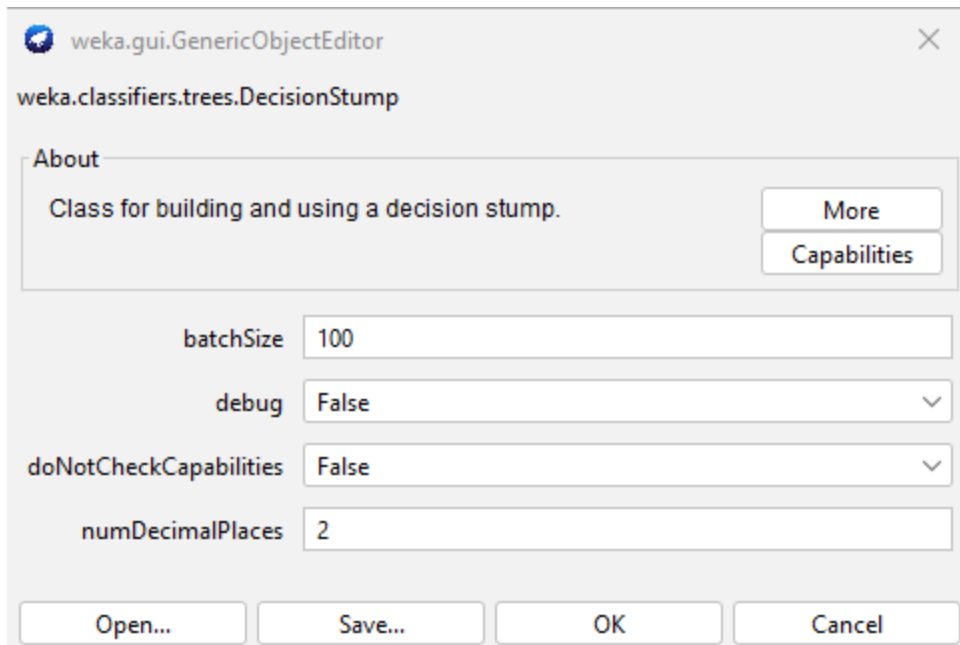
Classifications

Rotational speed [rpm] <= 1381.5 : 1
Rotational speed [rpm] > 1381.5 : 0
Rotational speed [rpm] is missing : 0

Class distributions

Rotational speed [rpm] <= 1381.5
0      1
0.15446106240330068      0.8455389375966993
Rotational speed [rpm] > 1381.5
0      1
0.7188827180659915      0.2811172819340085
Rotational speed [rpm] is missing
0      1
0.5      0.5
```

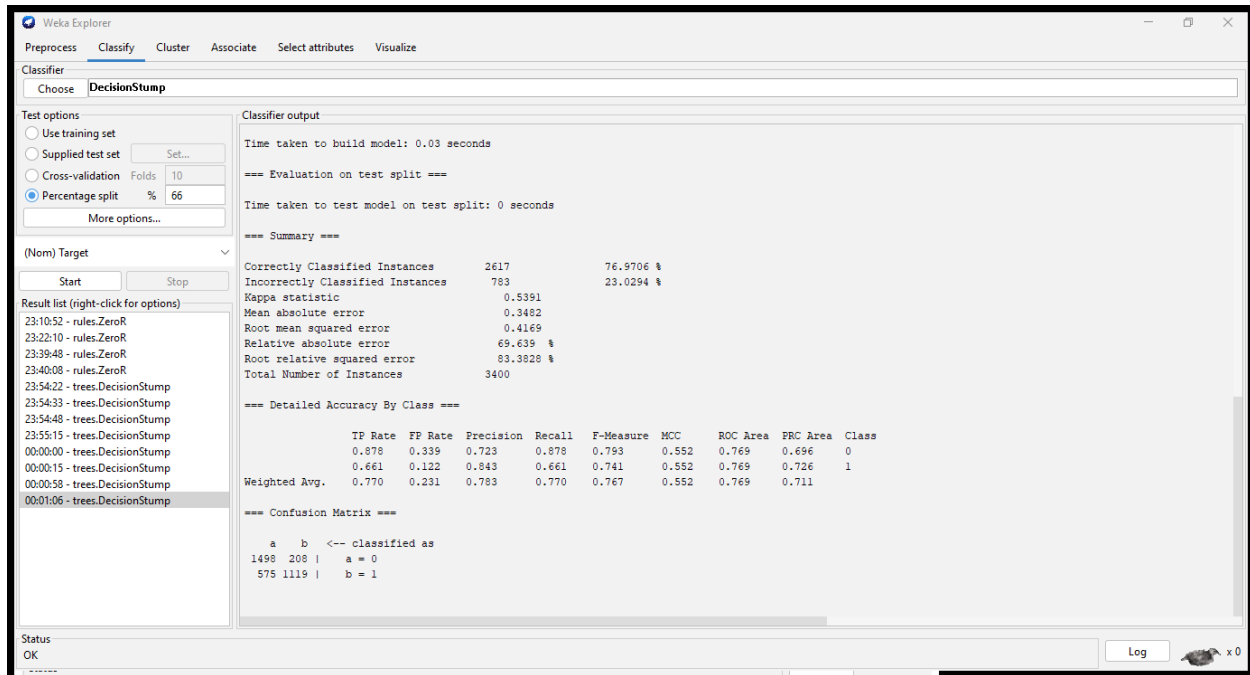
- The default parameters:



The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.trees.DecisionStump' classifier. It includes an 'About' section with a description and buttons for 'More' and 'Capabilities'. Below this are four parameter fields: 'batchSize' (100), 'debug' (False), 'doNotCheckCapabilities' (False), and 'numDecimalPlaces' (2). At the bottom are buttons for 'Open...', 'Save...', 'OK', and 'Cancel'.

Parameter	Value
batchSize	100
debug	False
doNotCheckCapabilities	False
numDecimalPlaces	2

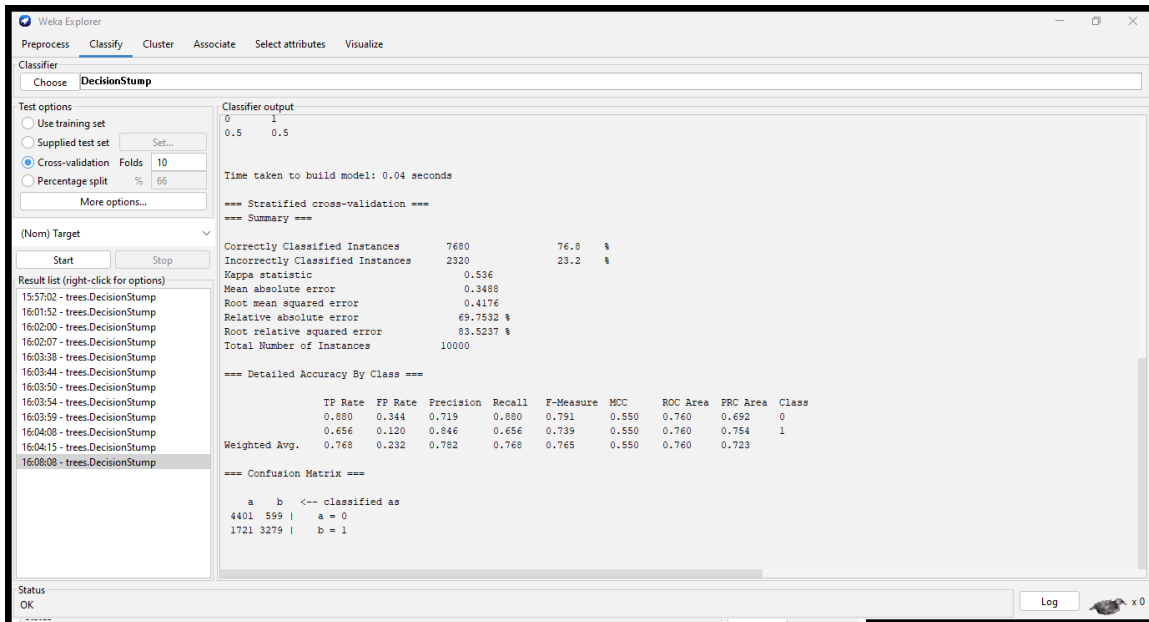
- The result:



The accuracy: 76.97%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1498	208
Actual 1	575	1119



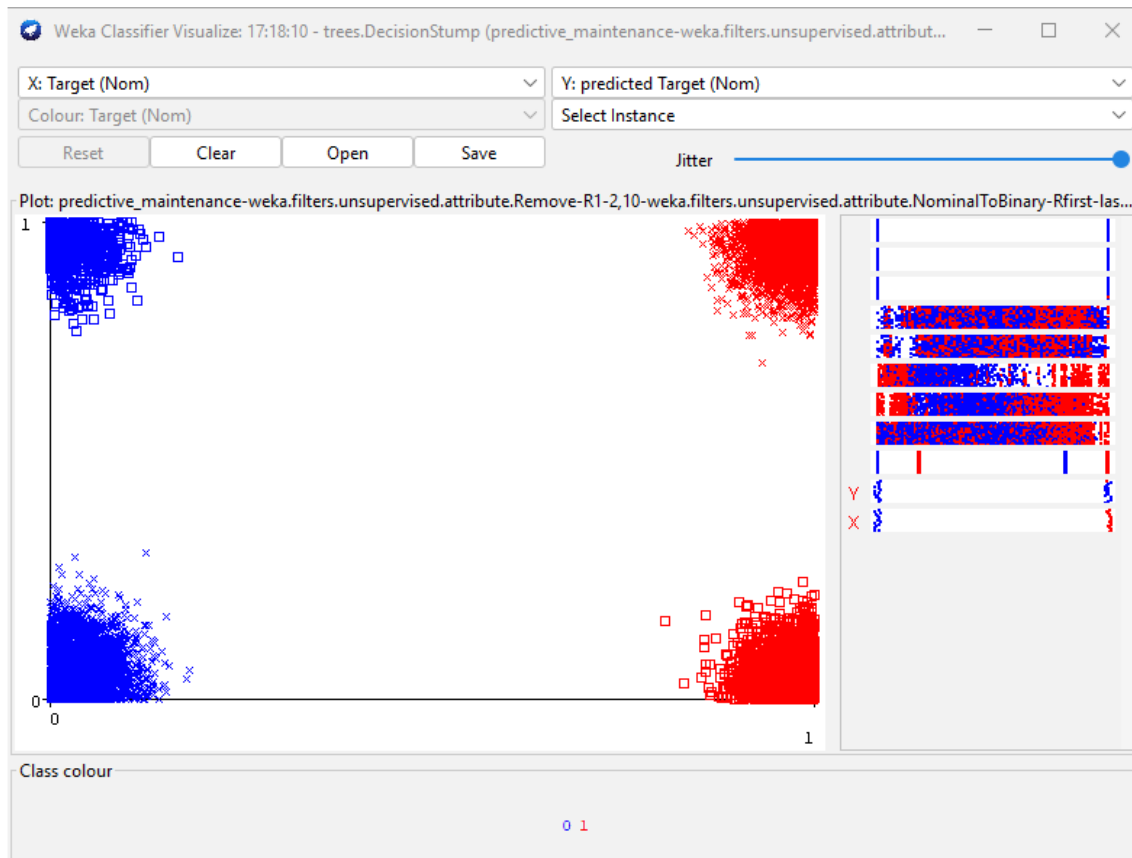
The accuracy: 76.8%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4401	599
Actual 1	1721	3279

- The algorithm with different parameters:
- I used the algorithm with different parameters, but the results did **not** change, and the accuracy remained the same. However, the overall accuracy is **not** satisfactory.

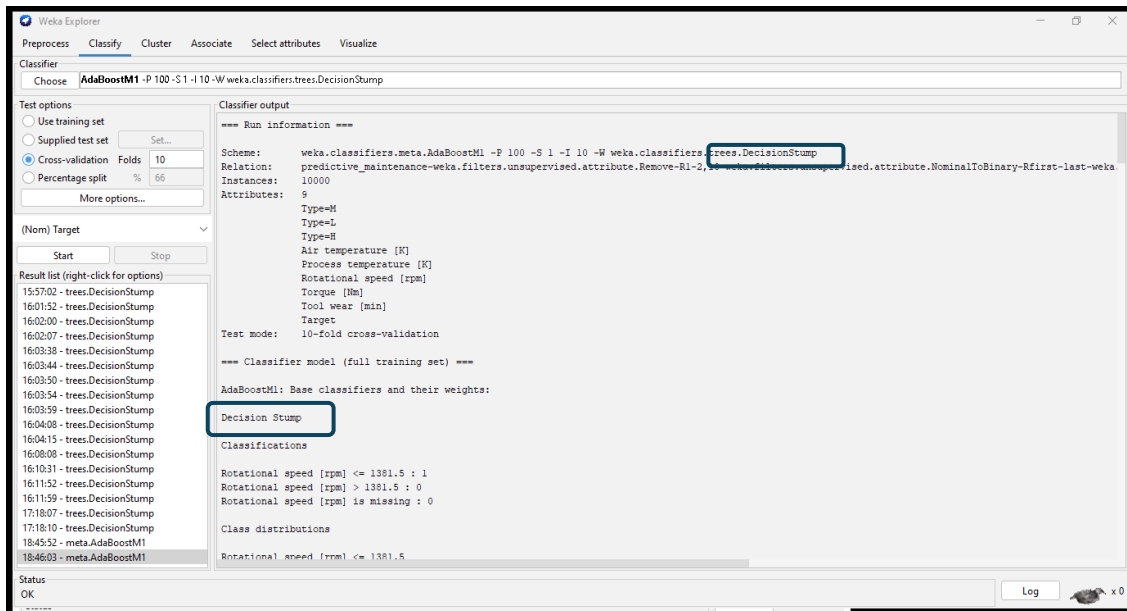
- Visualize classifier Error:



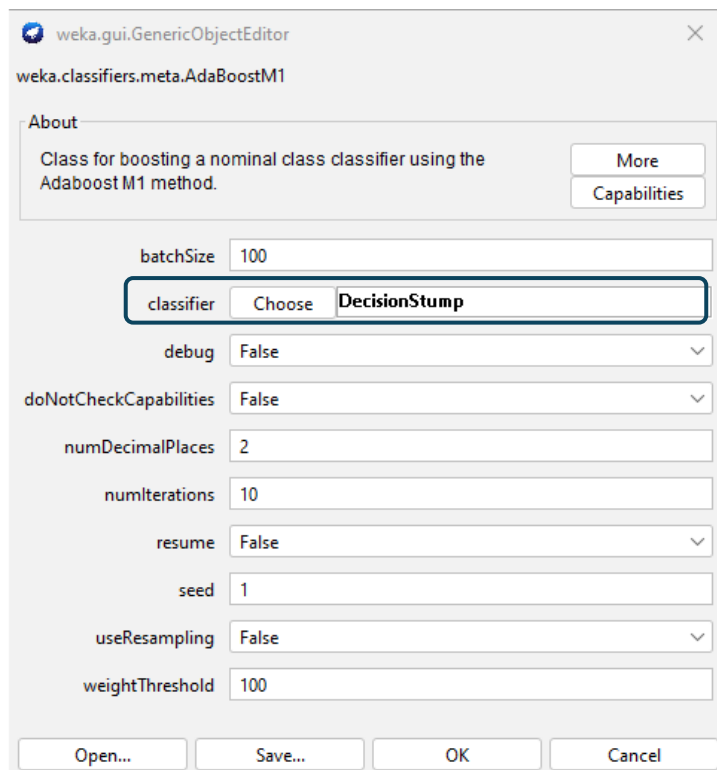
The x-axis represent the **true** values, while the y-axis represent the **predicted** values

6) AdaBoostM1:

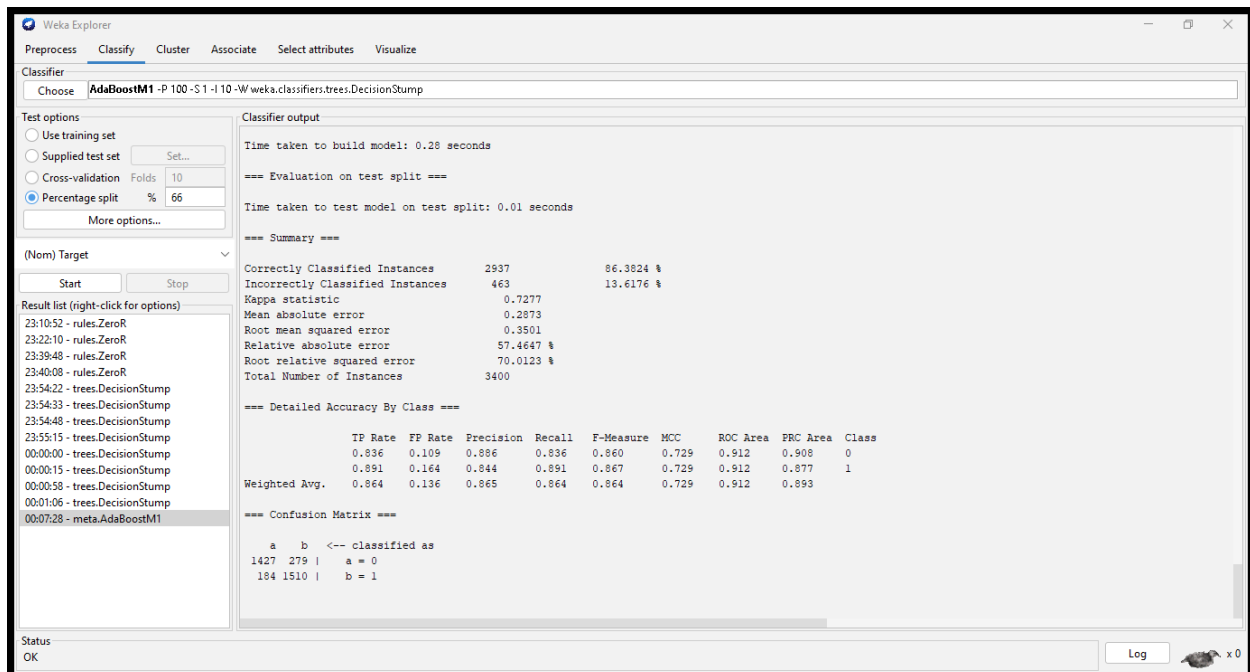
- The AdaBoostM1 combines multiple weak classifiers to create a strong classifier by adjusting the weights of training samples
- By default it use the **DecisionStump**



- The default parameters



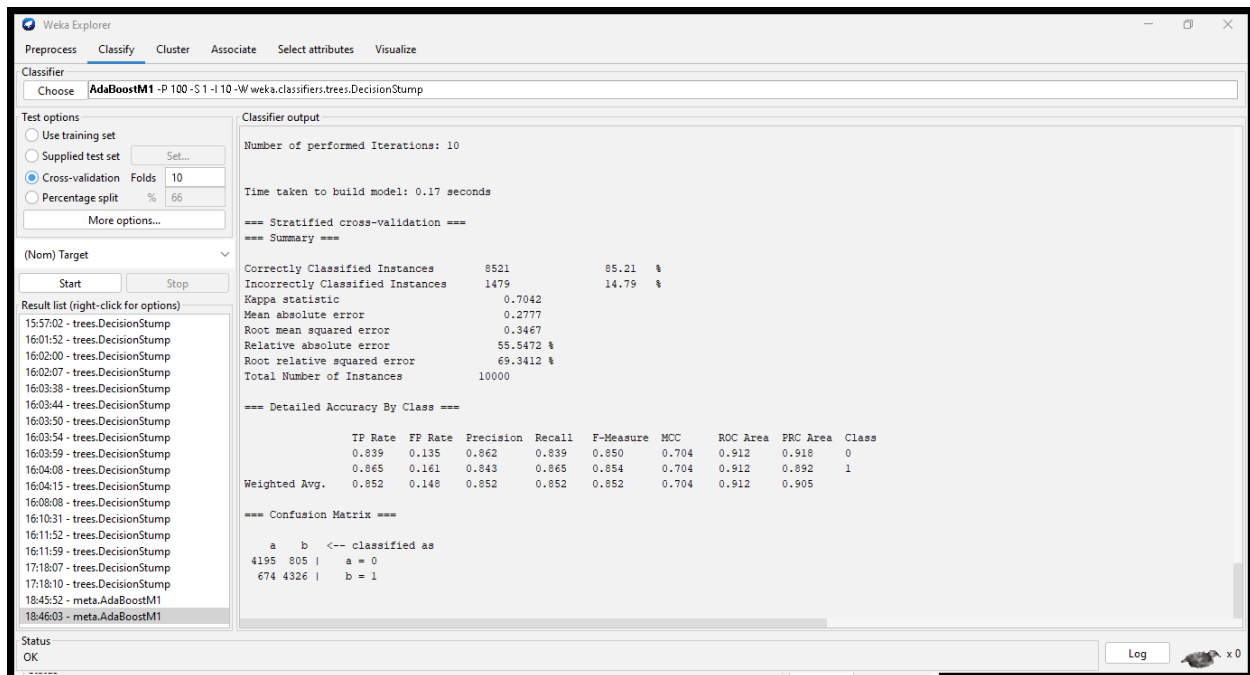
- The result:



The accuracy: 86.38%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1427	279
Actual 1	184	1510

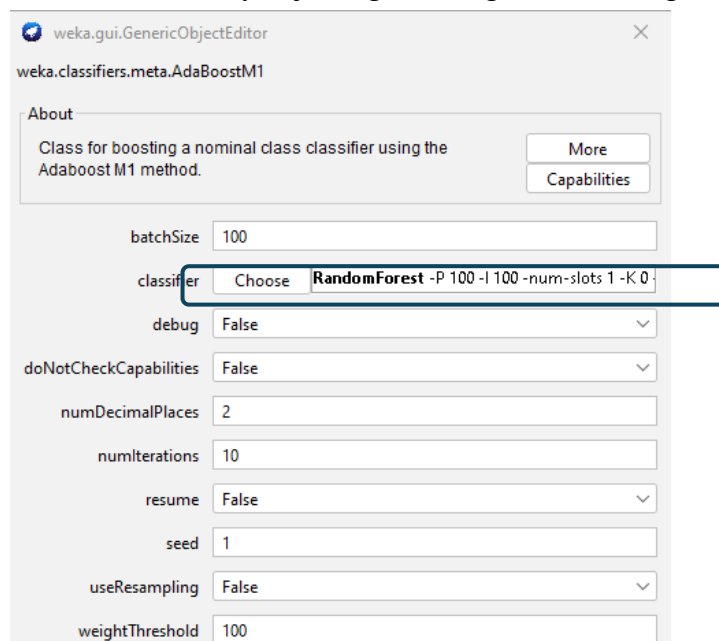


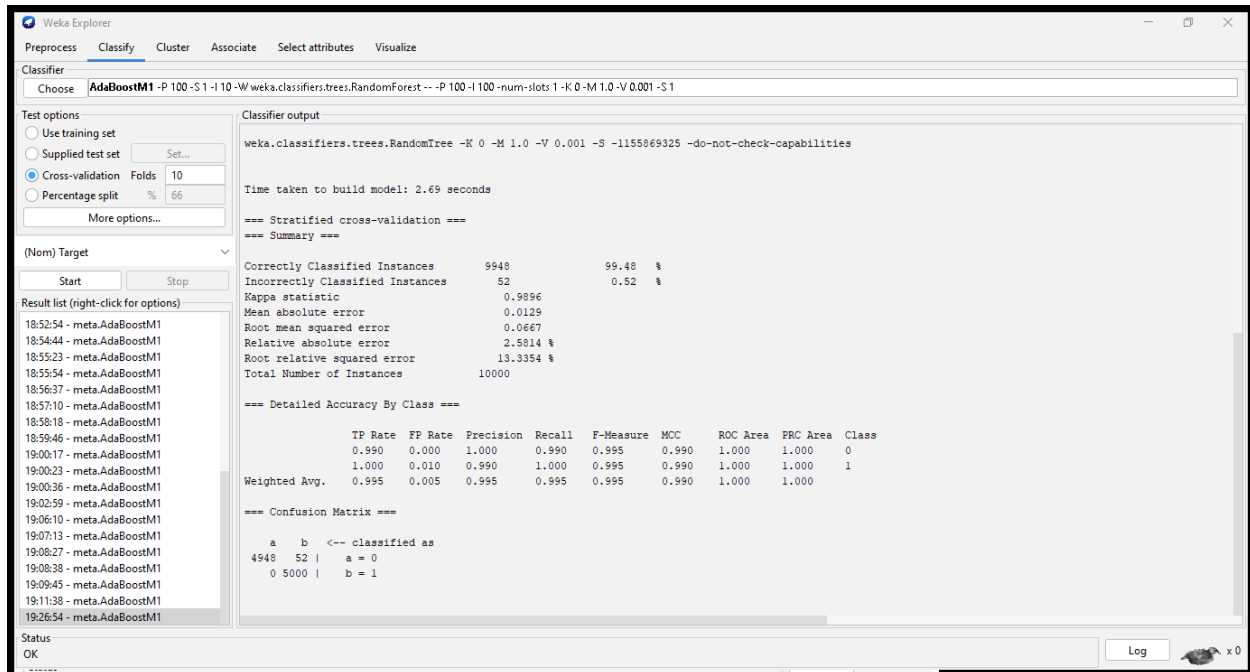
The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4195	805
Actual 1	674	4326

The algorithm with different parameters:

- When I changed the classifier to **RandomForest**, it gave higher accuracy than the default setting, but it took more time to run.
-
- **classifier** refers to the base classifier that AdaBoostM1 uses for boosting. It is the model that AdaBoostM1 enhances by adjusting the weights of training samples.

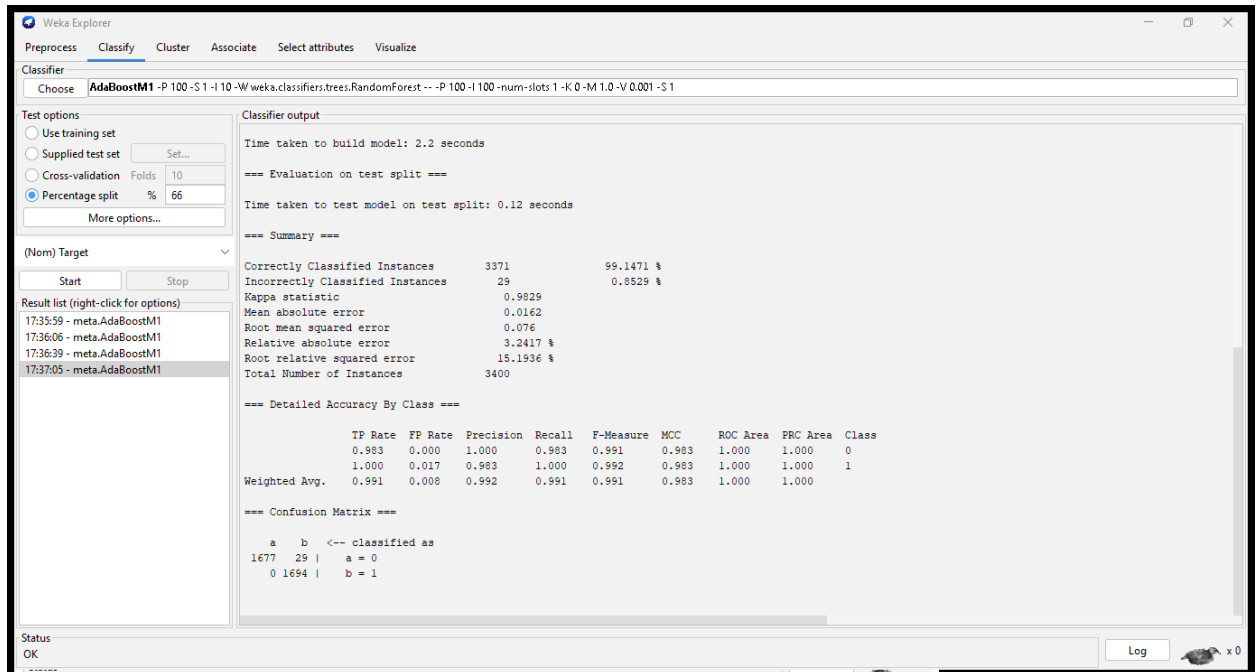




The accuracy: 99.48%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4948	52
Actual 1	0	5000

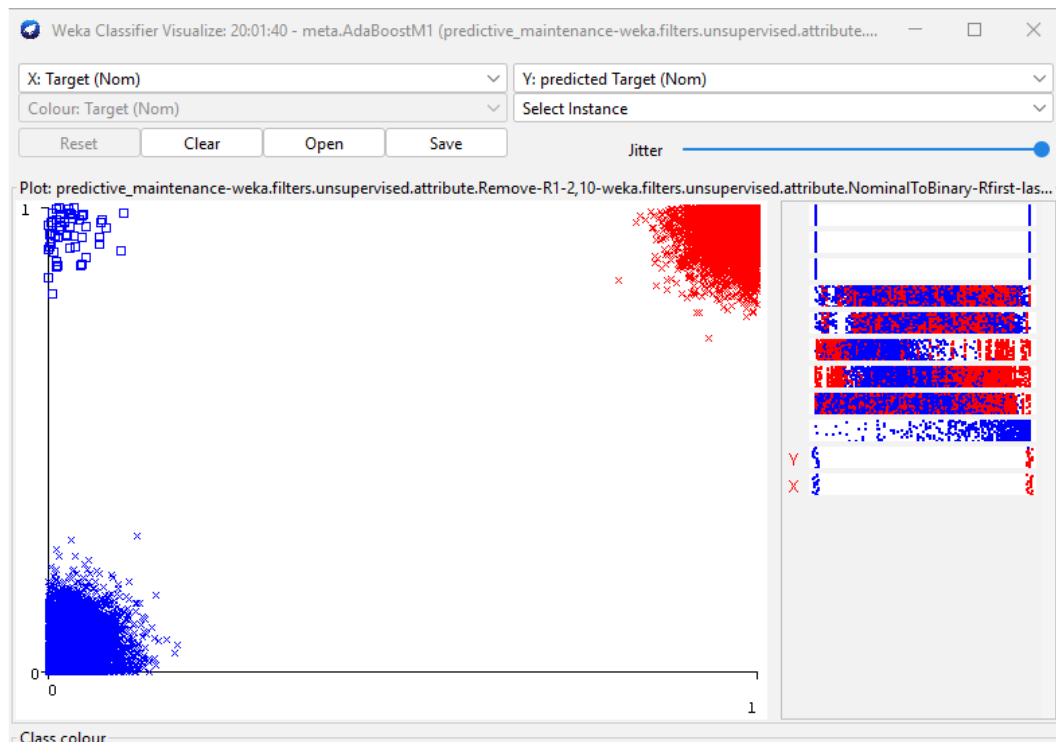


The accuracy: 99.14%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1677	29
Actual 1	0	1694

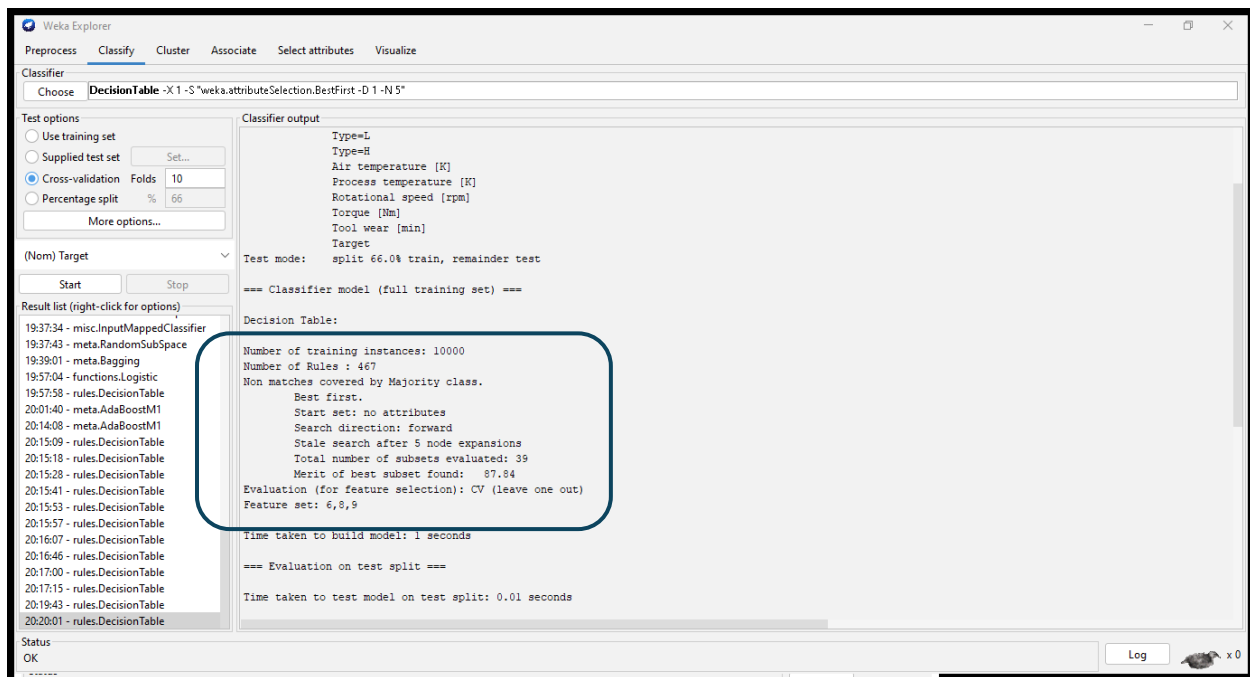
Visualize classifier Error:



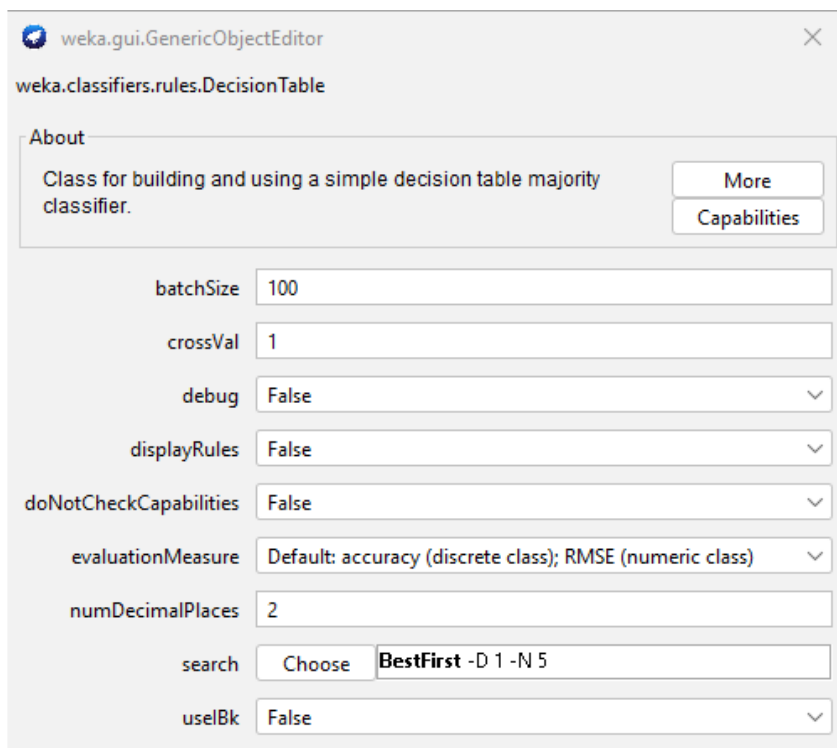
7) DecisionTable:

Class for building and using a simple decision table majority classifier

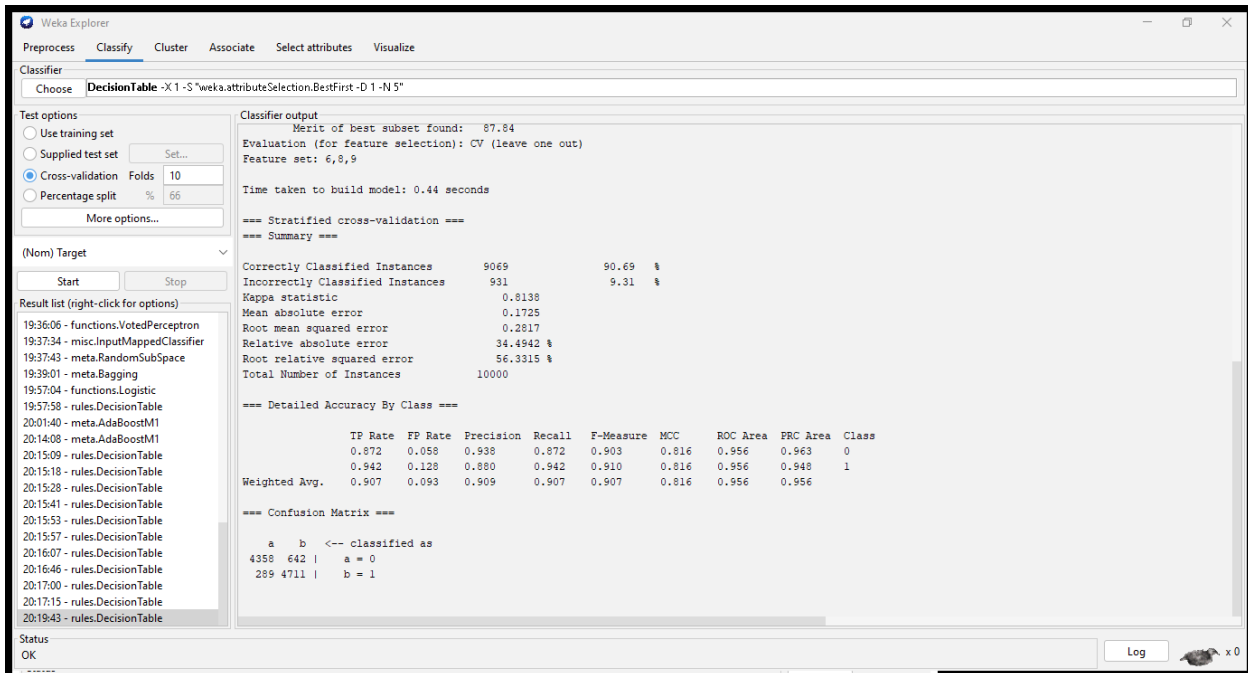
- Total **467 rules** were generated from the training data.
- If no rule matches, it assigns the **majority class**.
- Selected only **3 important features** (features 6, 8, and 9).



- The default parameters:



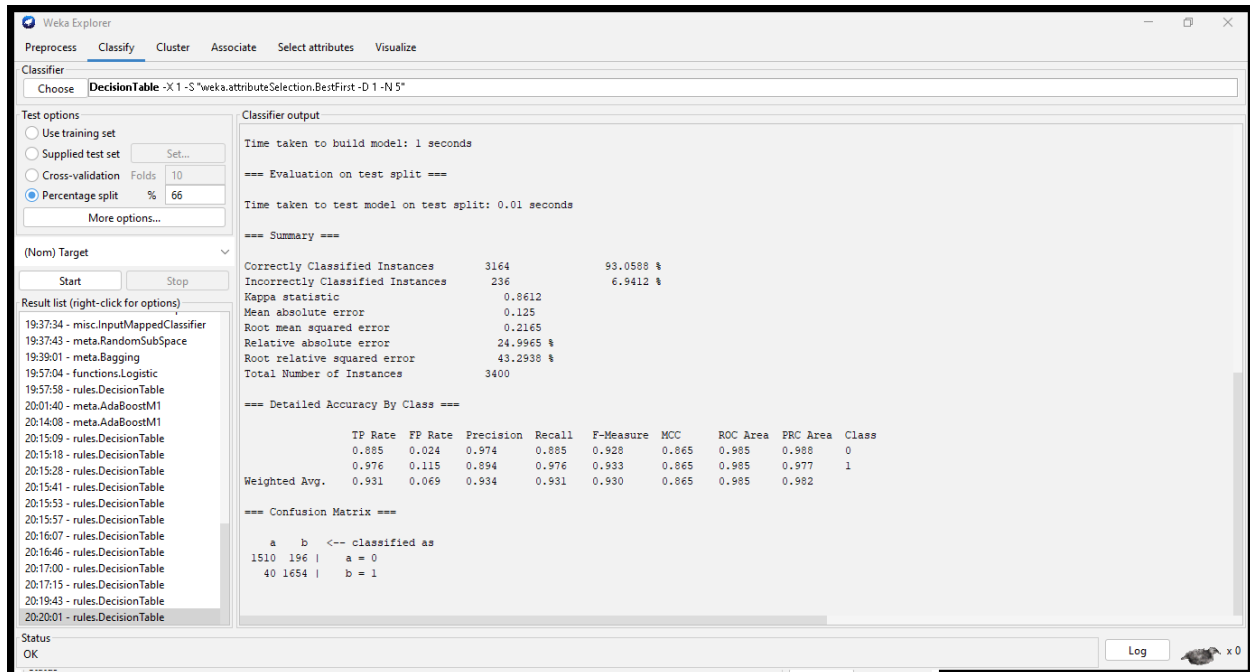
The result:



The accuracy: 90.69%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4358	642
Actual 1	289	4711



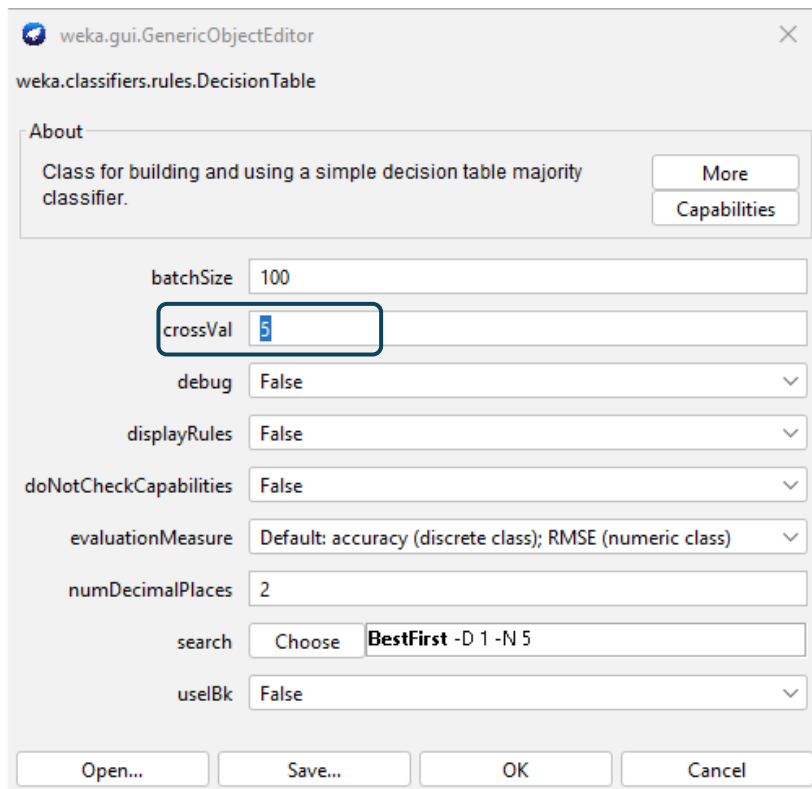
The accuracy: 93.05%

The confusion matrix:

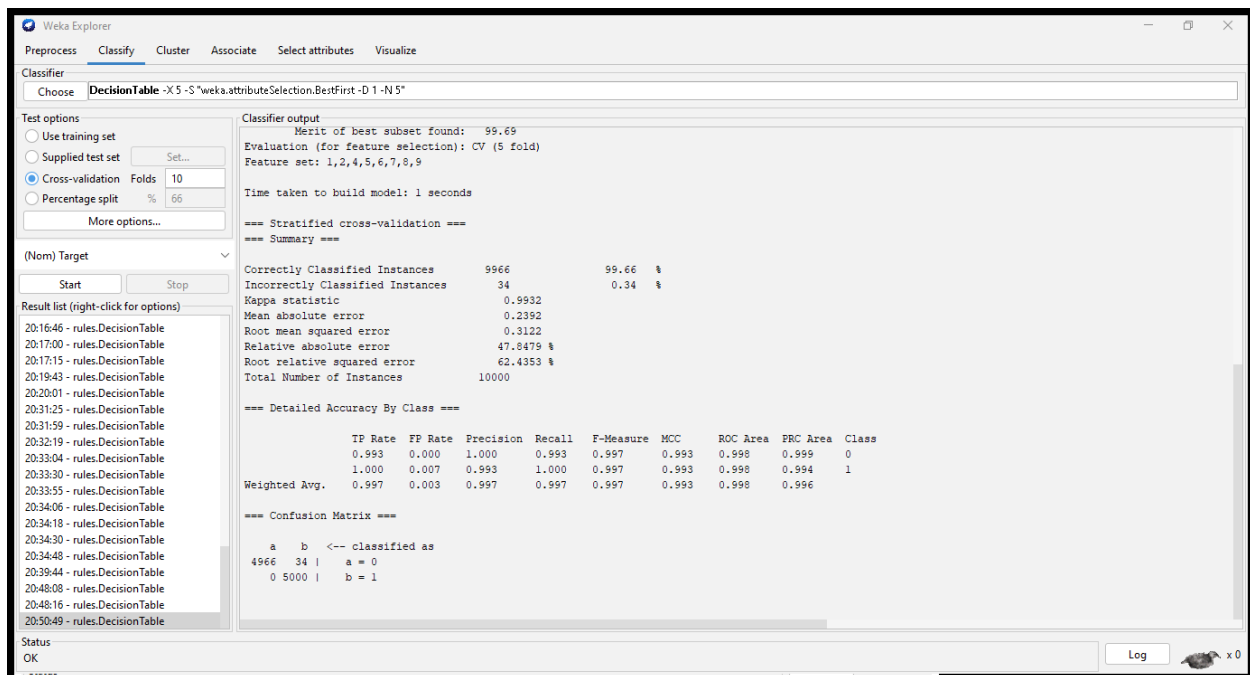
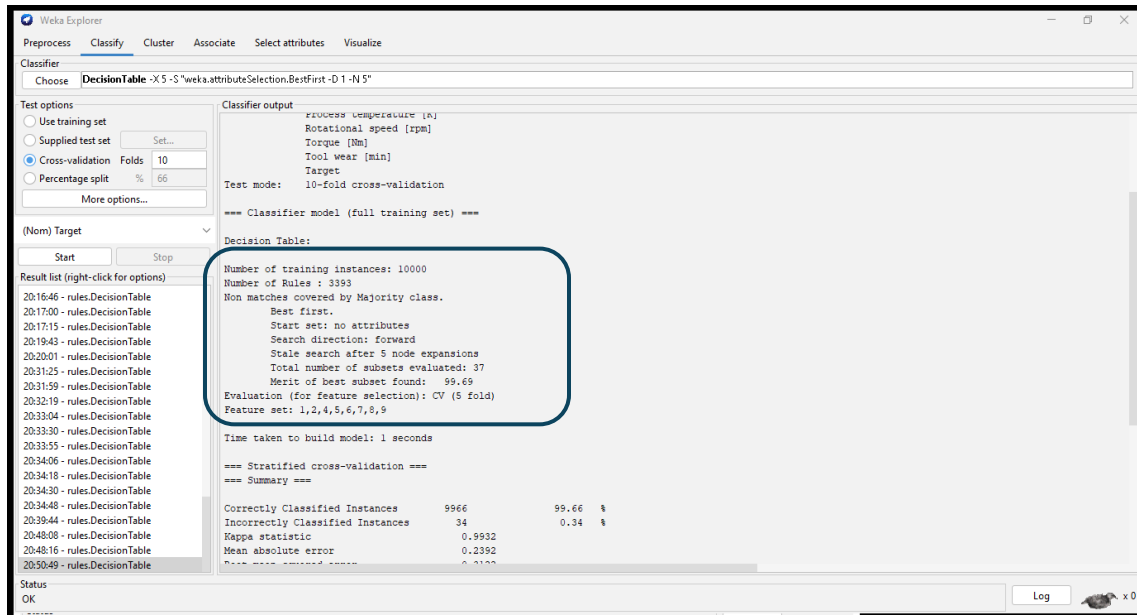
	Predicted 0	Predicted 1
Actual 0	1510	196
Actual 1	40	1654

- Percentage split better than cross validation

- **The algorithm with different parameters:**
- When I changed the **crossVal** parameter and increased its value (**from 1 to 5**), the algorithm showed **better** performance compared to the default setting.
- **crossVal** determines how many times cross-validation is used during feature selection. Increasing it allows for a more thorough evaluation of features, which can improve the model's accuracy.



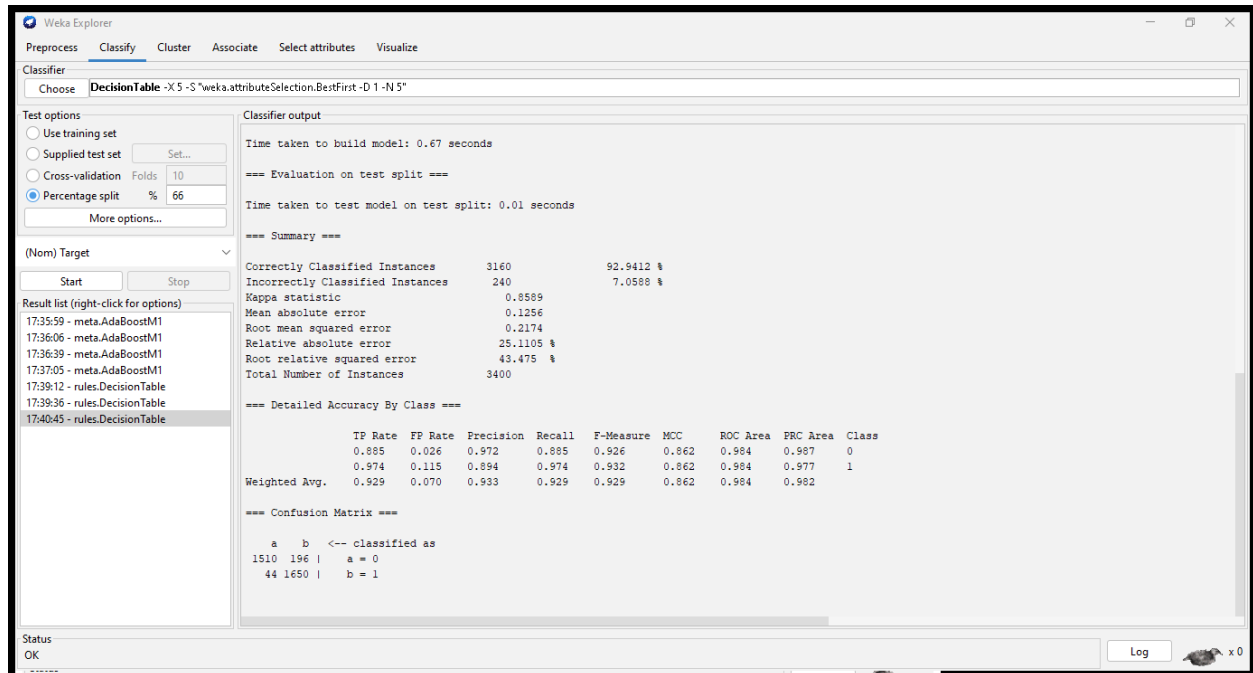
- Total **3393** rules were generated from the training data.
- If no rule matches, it assigns the **majority class**.
- Selected **8 important features** (features 1,2,4,5,6,8, and 9).



The accuracy: 99.66%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4966	34
Actual 1	0	5000



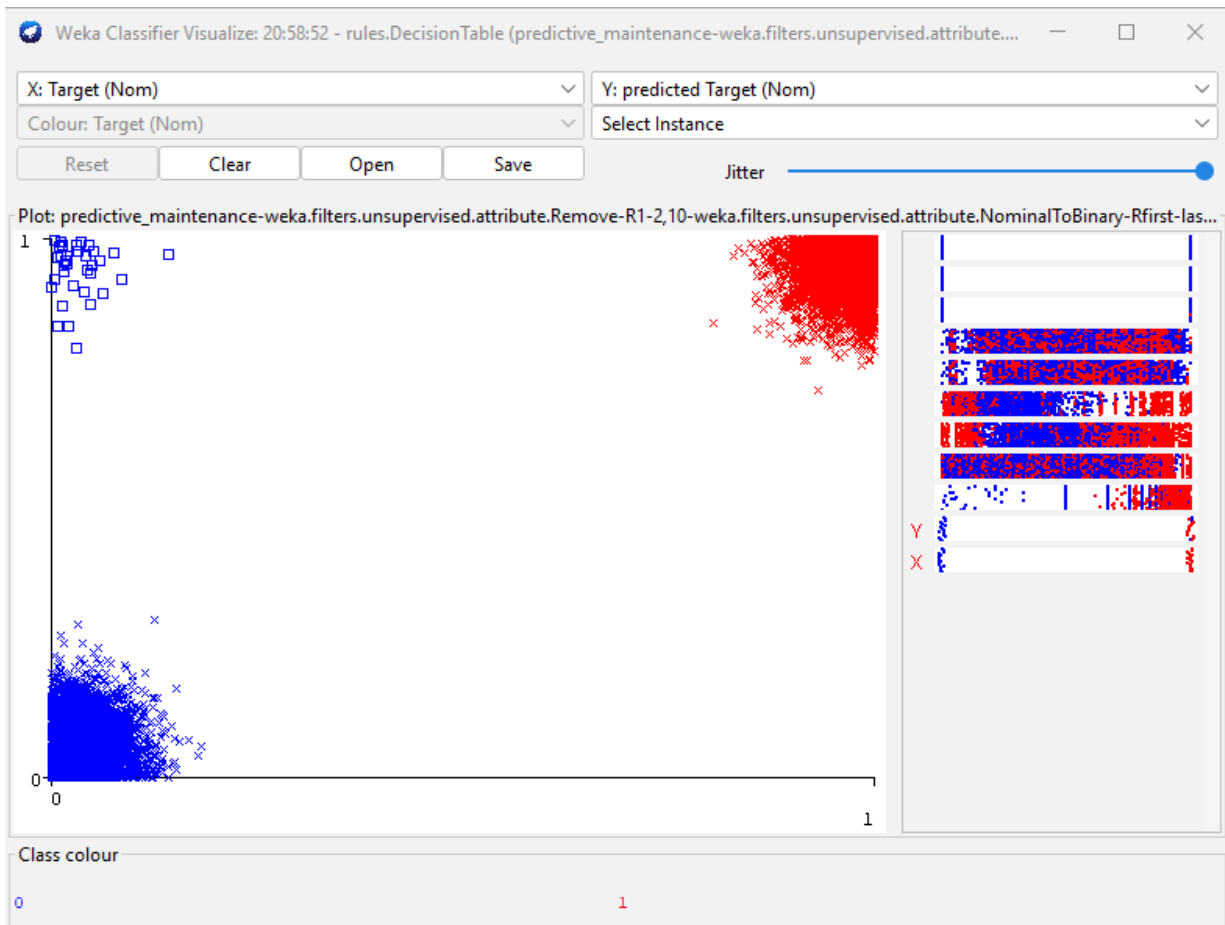
The accuracy: 92.94%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1510	196
Actual 1	44	1650

- At first, with the default parameters, the Percentage Split method gave a higher accuracy than Cross Validation. But after I changed the algorithm's parameters, Cross Validation gave a better result.

- **Visualize classifier Error:**



- **Final result:**

- The **DecisionStump** algorithm achieved an accuracy of **76.97%**.
- The **AdaBoostM1** algorithm achieved an accuracy of **99.48%**.
- The **DecisionTable** algorithm achieved an accuracy of **99.66%**.

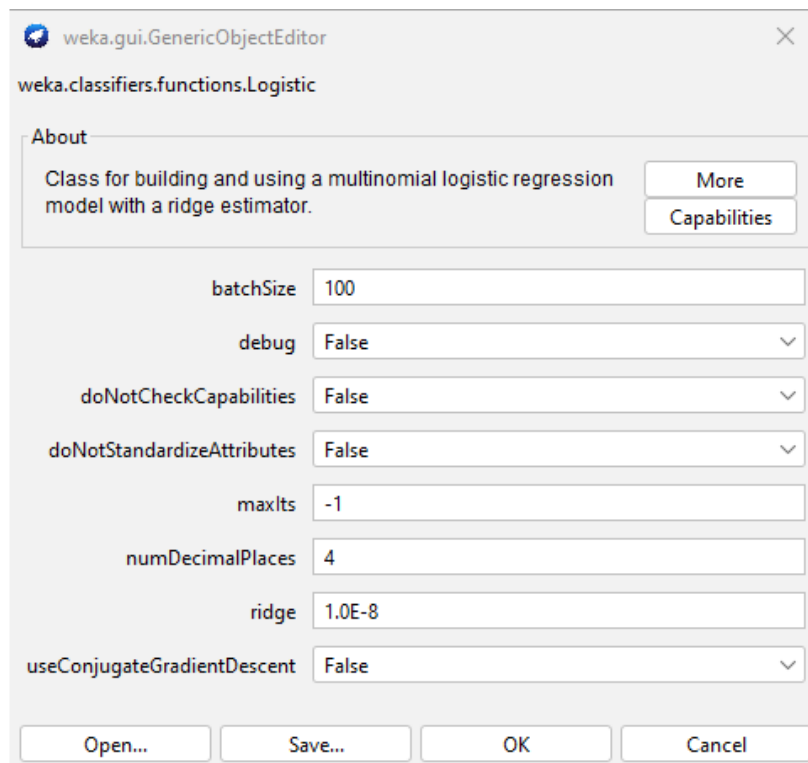
This indicates that the **DecisionTable** algorithm performed the **best** among the three.

8) Logistic Regression (LR)

Logistic Regression is a classification method used to predict whether the outcome is 0 or 1 (such as success or failure). It aims to learn the relationship between the input features in order to estimate the probability that a given instance belongs to a particular class or not.

In our data, we applied Logistic Regression to predict whether a machine would fail or succeed based on features like air temperature, rotational speed, and torque.

- The default parameters:

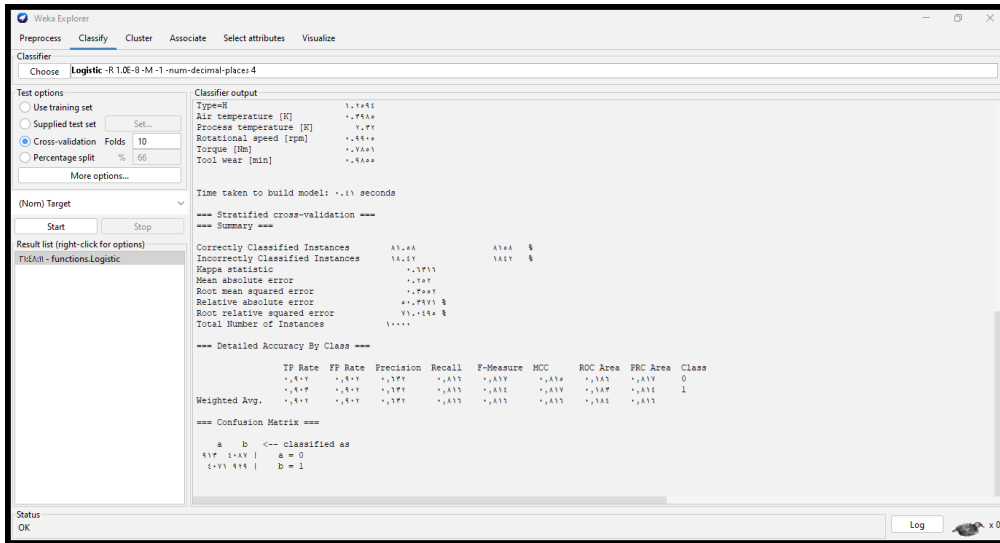


The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.functions.Logistic' classifier. The 'About' tab is active, displaying the description: 'Class for building and using a multinomial logistic regression model with a ridge estimator.' Below this, there are two buttons: 'More' and 'Capabilities'. The main area lists several parameters with their default values:

Parameter	Value
batchSize	100
debug	False
doNotCheckCapabilities	False
doNotStandardizeAttributes	False
maxIts	-1
numDecimalPlaces	4
ridge	1.0E-8
useConjugateGradientDescent	False

At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

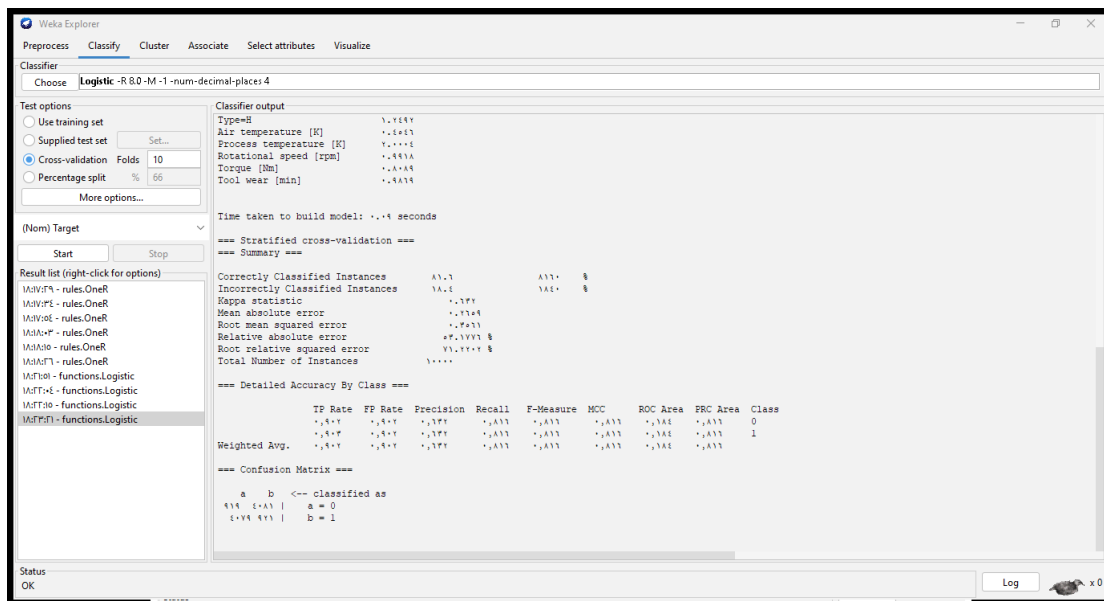
- The result



The accuracy: 81.58%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4087	913
Actual 1	929	4071

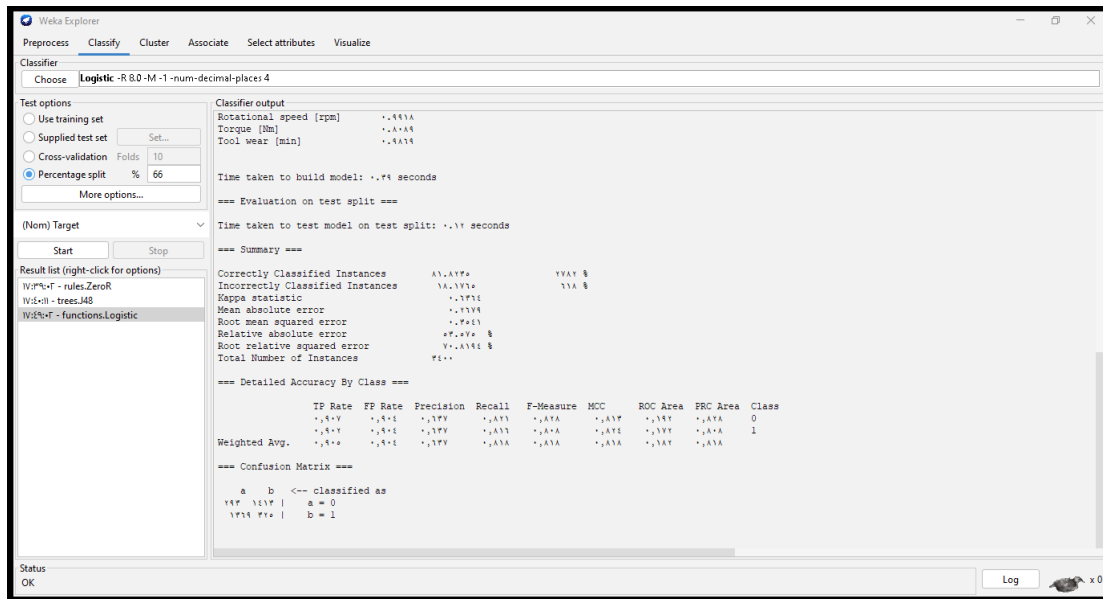


The accuracy : 81.6%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4081	919
Actual 1	921	4079

- Percentage split



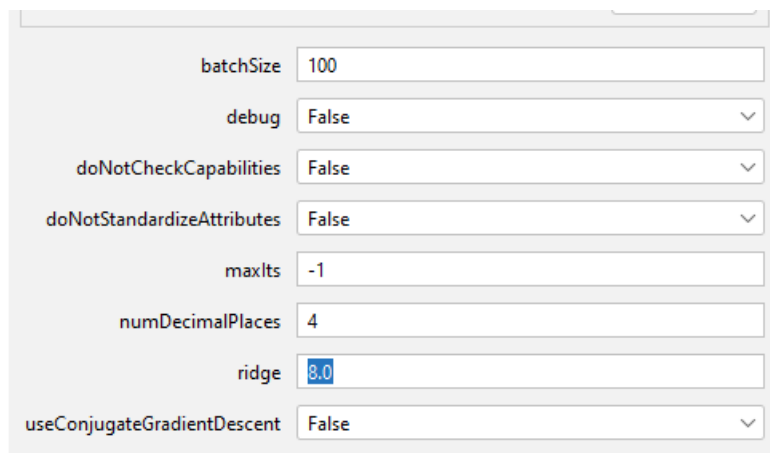
The accuracy : 81.8%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1413	293
Actual 1	325	1369

The algorithm with different parameters:

- When I changed the ridge parameter (from 1.0E-8 to 8.0) and increased its value, the algorithm gave better performance compared to the default setting."
- ridge** Set the Ridge value in the log-likelihood.



- Visualize classifier Error:



9) (J48) Tree :

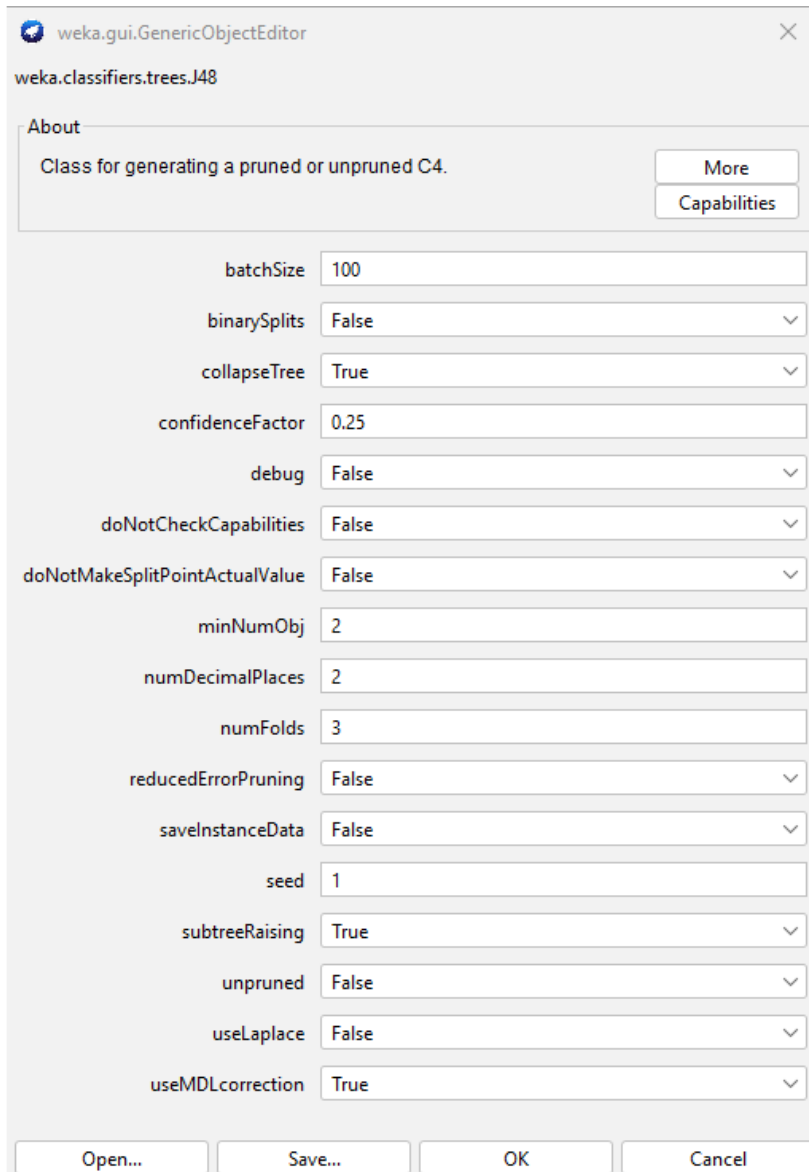
J48 classifier is a Decision tree algorithm that builds a decision tree based on the attribute values of the dataset.

The J48 classifier is popular because it produces interpretable models and can handle both numeric and nominal attributes.

It supports pruning techniques to reduce overfitting and provides options for parameter tuning to optimize the model's accuracy.

In our data , we used J48 to predict whether the machine would fail or work.

- The default parametets:

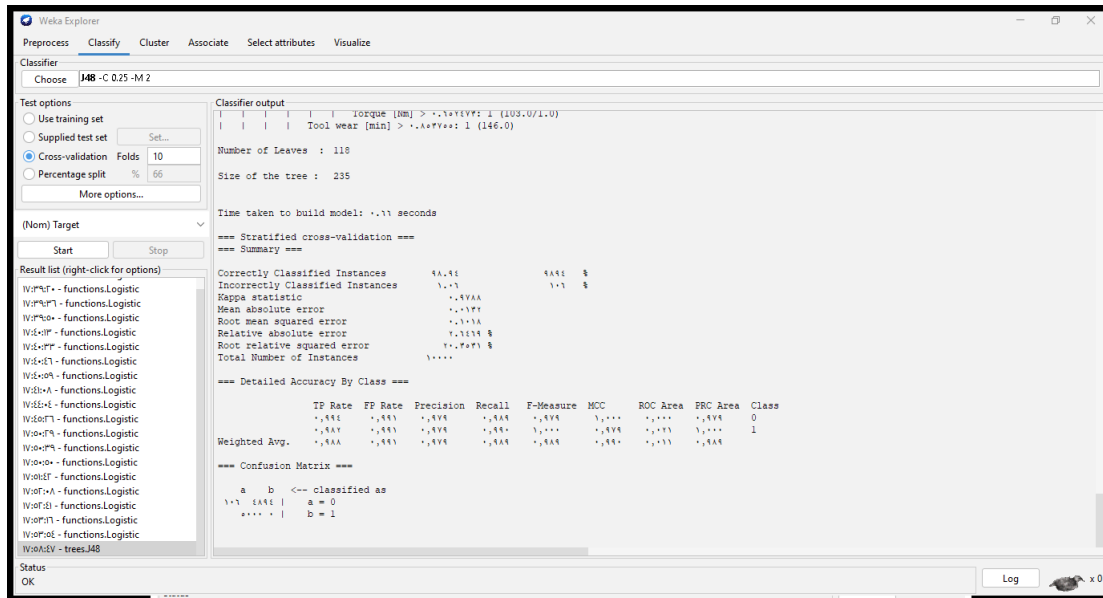


The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.trees.J48' classifier. The window has a title bar with a close button. Below the title bar, there is an 'About' section with the text 'Class for generating a pruned or unpruned C4.' and two buttons: 'More' and 'Capabilities'. The main area contains a list of parameters with their current values:

Parameter	Value
batchSize	100
binarySplits	False
collapseTree	True
confidenceFactor	0.25
debug	False
doNotCheckCapabilities	False
doNotMakeSplitPointActualValue	False
minNumObj	2
numDecimalPlaces	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False
useMDLcorrection	True

At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

- The result



The accuracy: 98.94%

The confusion matrix:

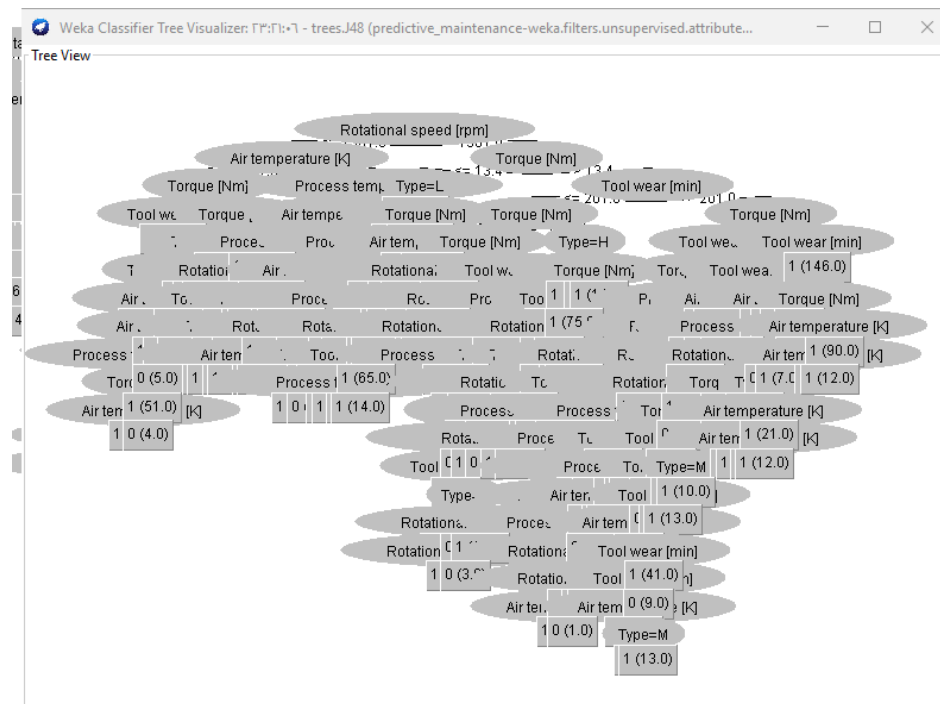
	Predicted 0	Predicted 1
Actual 0	4894	106
Actual 1	0	5000

The accuracy : 98.88%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1668	38
Actual 1	0	1694

- Tree J48



The algorithm with different parameters:

When I changed the binarySplits parameter (from false to true), the confidenceFactor (from 0.25 to 0.3), and the minNumObj (from 2 to 1), the algorithm improved performance."

- **binarySplits**: Whether to use binary splits on nominal attributes when building the trees.
- **confidenceFactor**: The confidence factor used for pruning (smaller values incur more pruning).
- **minNumObj**: The minimum number of instances per leaf.

The parameter

weka.gui.GenericObjectEditor

weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

More

Capabilities

batchSize 100

binarySplits True

collapseTree True

confidenceFactor 0.3

debug False

doNotCheckCapabilities False

doNotMakeSplitPointActualValue False

minNumObj 1

numDecimalPlaces 2

numFolds 3

reducedErrorPruning False

saveInstanceData False

seed 1

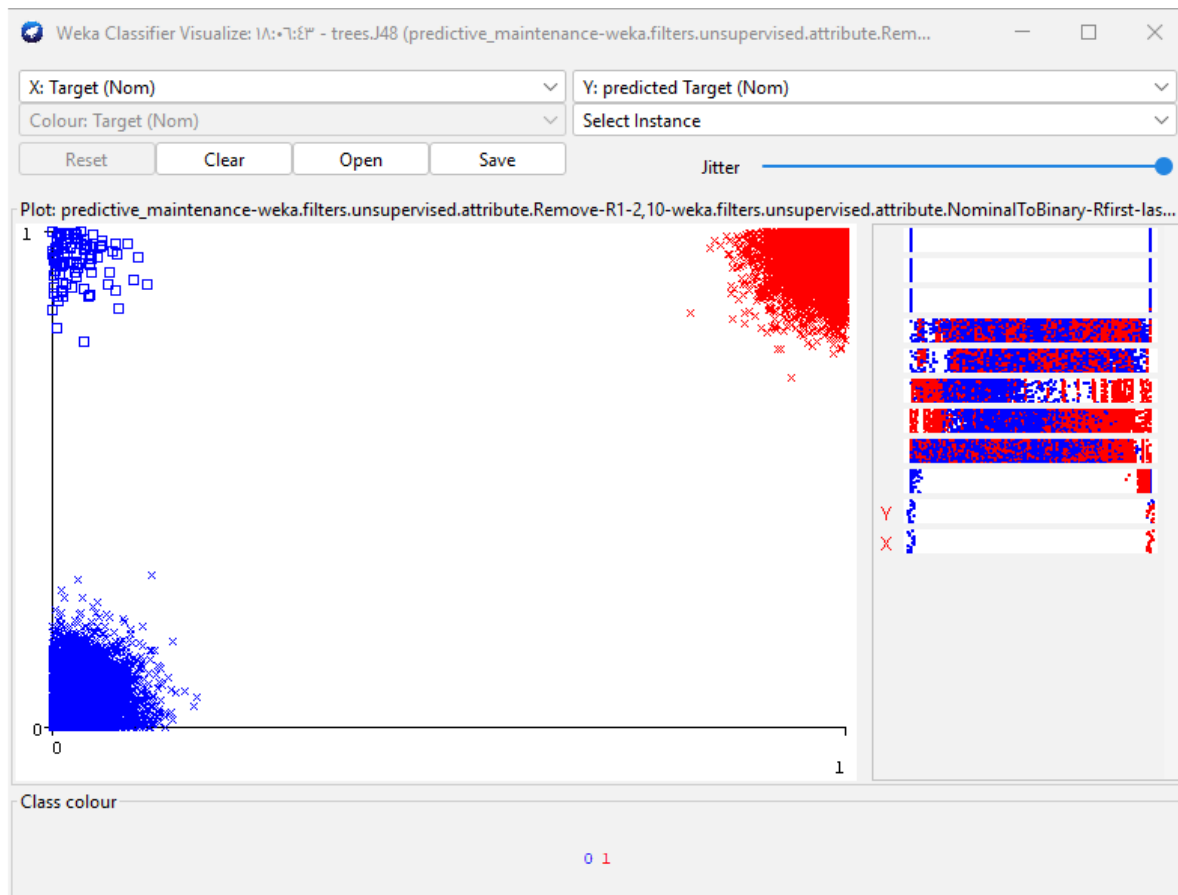
subtreeRaising True

unpruned False

useLaplace False

useMDLcorrection True

Open... Save... OK Cancel

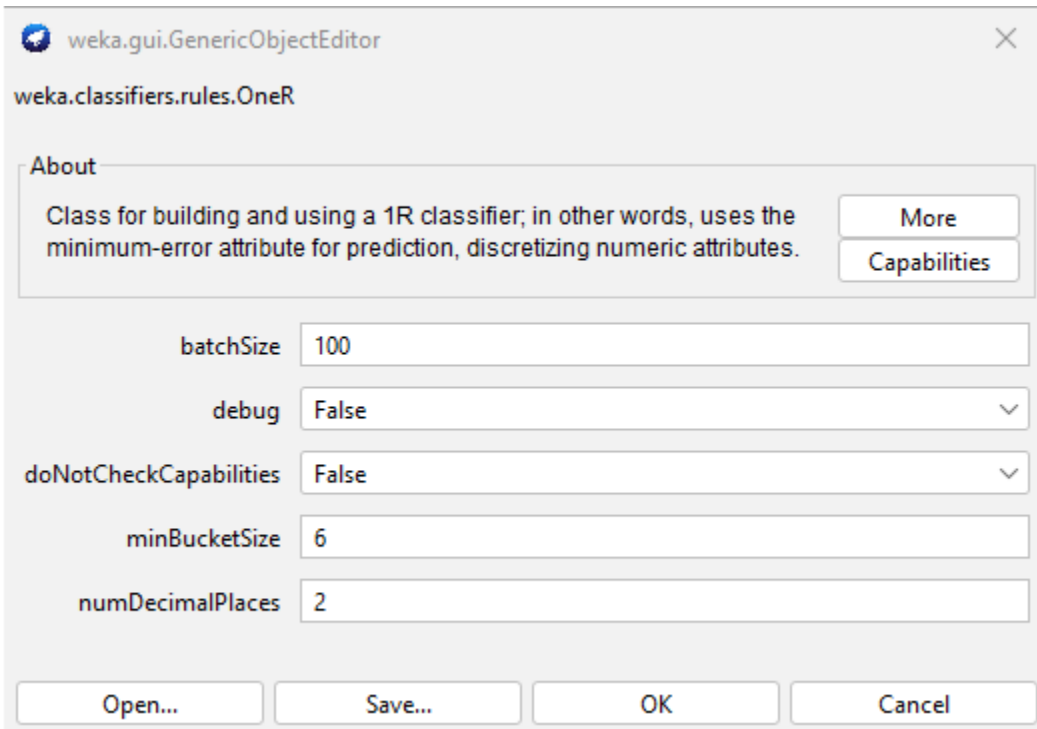


10) OneR:

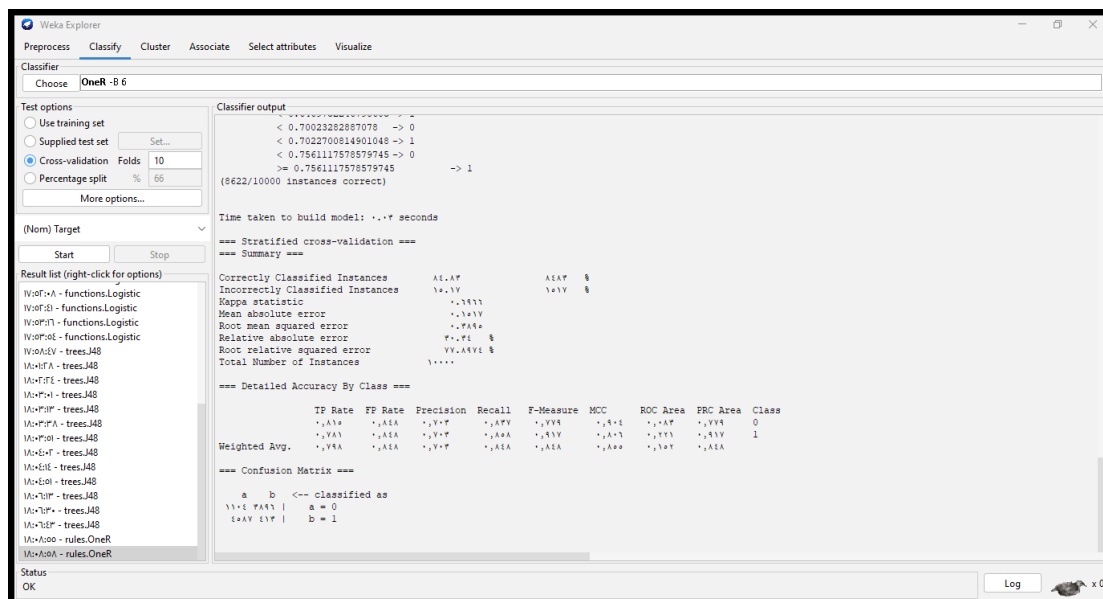
OneR is a simple classification algorithm that learns rules from a single attribute. It is characterized by its simplicity and ease of interpretation. The algorithm examines all attributes and selects the one that results in the lowest classification error.

In our data, rotational speed was chosen as the most important attribute

- The default parameters:



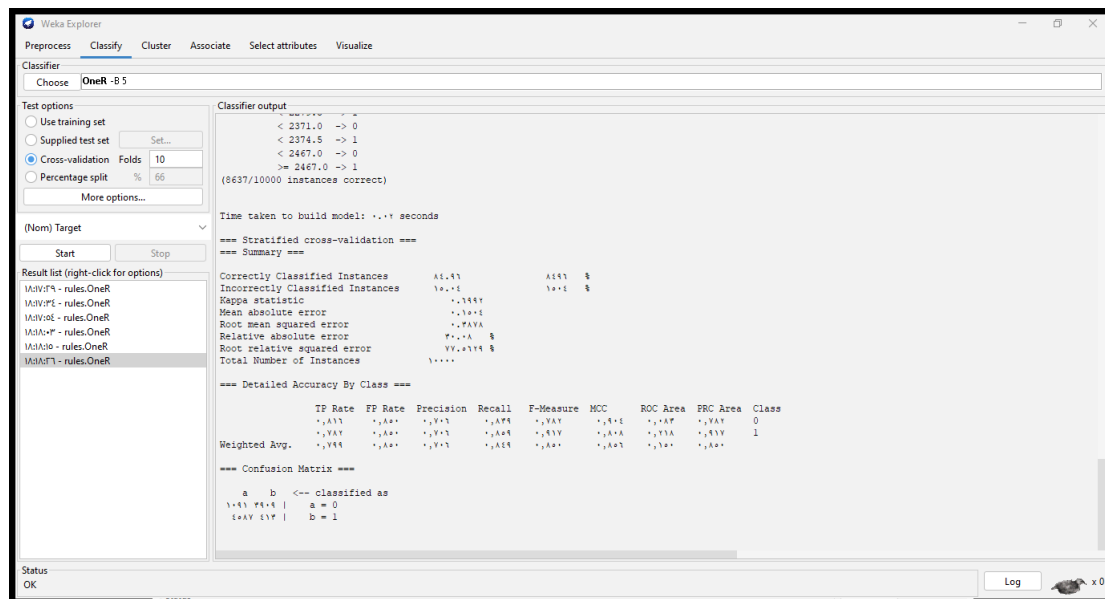
- The result



The accuracy : 84.83%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	3891	1104
Actual 1	413	4587

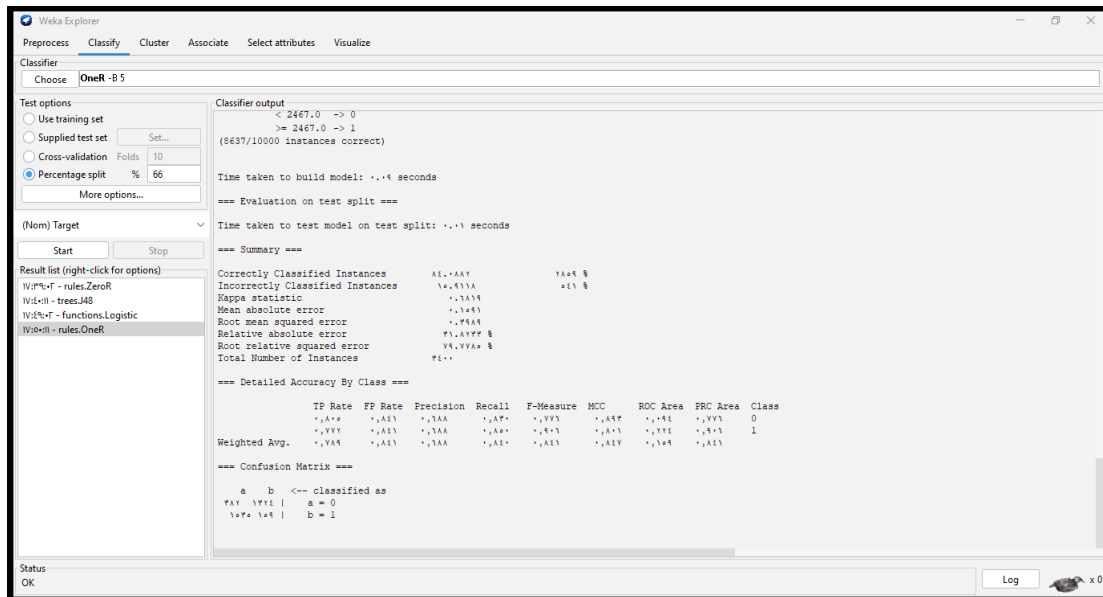


The accuracy : 85.96%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	3909	1091
Actual 1	413	4587

- Percentage split



The accuracy : 84.08%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	1324	382
Actual 1	159	1535

The algorithm with different parameters:

When I changed the minBucketSize parameter (from 6 to 5), the algorithm showed improved performance."

- **minBucketSize:** The minimum bucket size used for discretizing numeric attributes.

batchSize

debug

False

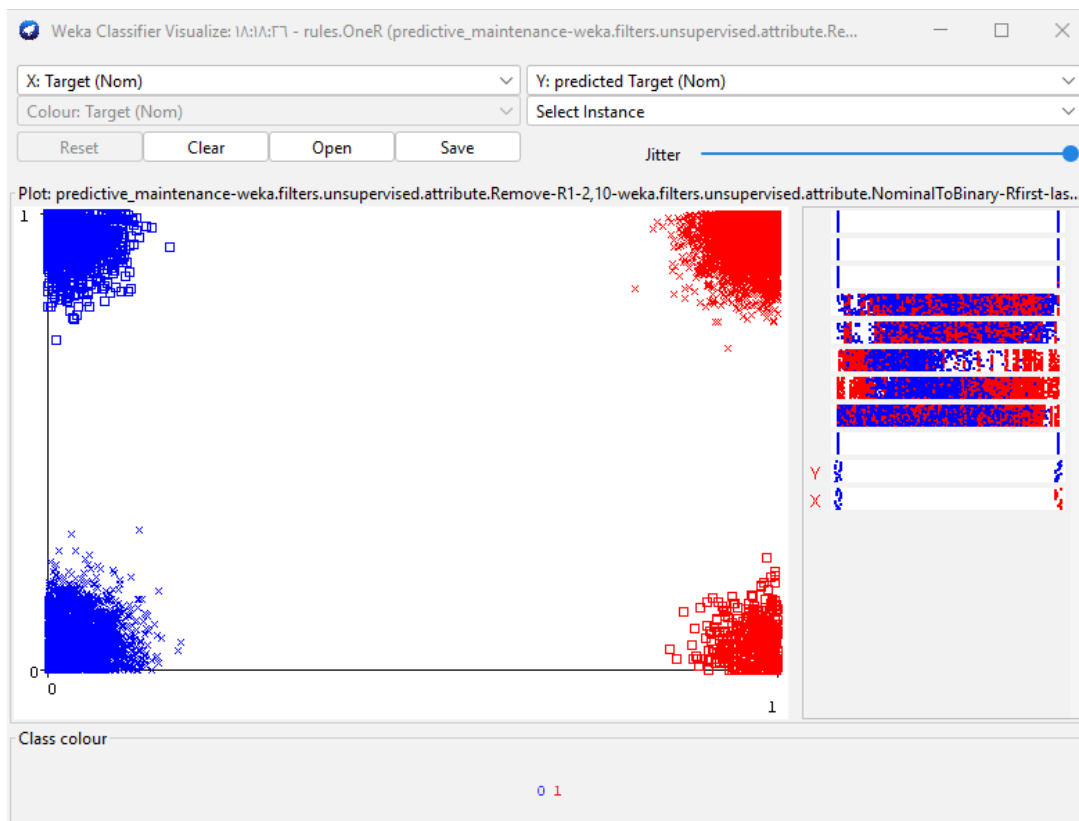
doNotCheckCapabilities

False

minBucketSize

numDecimalPlaces

- Visualize classifier Error:



- **Final result:**

- The Logistic Regression (LR) algorithm achieved an accuracy of 81.6%.
- The Decision Tree (J48) algorithm achieved an accuracy of 99.5%.
- The OneR algorithm achieved an accuracy of 85.96%.

This indicates that the Decision Tree (J48) algorithm performed the best among the three.

11) Simple logistic

The model Focus on all features. It's a linear model that assigns weights to each feature to predict probabilities

Focus on our data: Likely emphasizes Process temperature, Torque, and Tool wear

Most important features:

- Air temperature
- Torque
- Type L

- The result:

```
SimpleLogistic:

Class 0 :
39.45 +
[Type=L] * -0.16 +
[Air temperature [K]] * -0.24 +
[Process temperature [K]] * 0.14 +
[Rotational speed [rpm]] * -0 +
[Torque [Nm]] * -0.09 +
[Tool wear [min]] * -0

Class 1 :
-39.45 +
[Type=L] * 0.16 +
[Air temperature [K]] * 0.24 +
[Process temperature [K]] * -0.14 +
[Rotational speed [rpm]] * 0 +
[Torque [Nm]] * 0.09 +
```

Classifier output

Time taken to build model: 1.03 seconds

=== Stratified cross-validation ===

=== Summary ===

Metric	Value	%
Correctly Classified Instances	8219	82.19
Incorrectly Classified Instances	1781	17.81
Kappa statistic	0.6438	
Mean absolute error	0.2909	
Root mean squared error	0.3635	
Relative absolute error	58.1834	%
Root relative squared error	72.6978	%
Total Number of Instances	10000	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
Weighted Avg.	0.822	0.178	0.822	0.822	0.822	0.644	0.897	0.897	0
	0.824	0.180	0.821	0.824	0.822	0.644	0.897	0.899	1

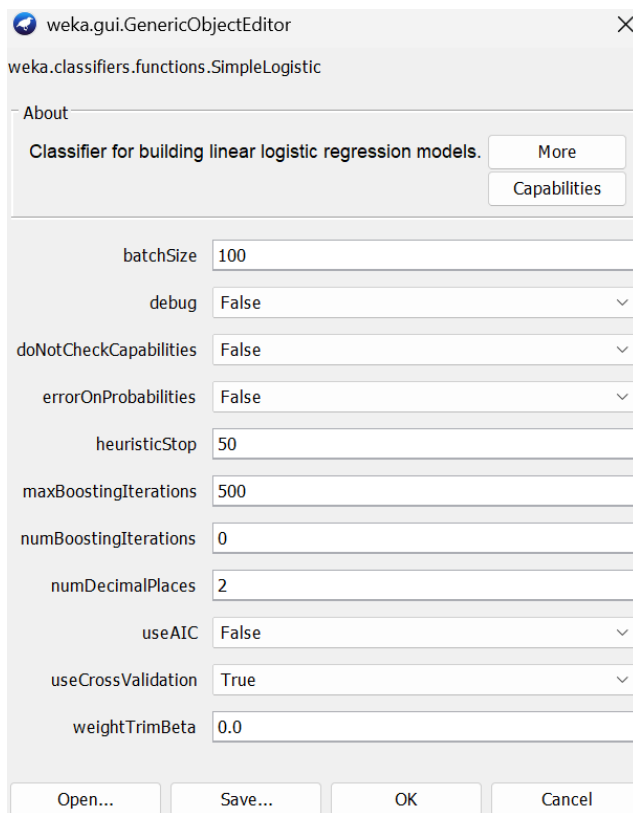
=== Confusion Matrix ===

a	b	<-- classified as
4099	901	a = 0
880	4120	b = 1

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4099	901
Actual 1	880	4120

- Default parameter



The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.functions.SimpleLogistic' classifier. The window has a title bar with a close button. Below the title bar, there's a tab labeled 'About'. The 'About' tab contains the text 'Classifier for building linear logistic regression models.' and two buttons: 'More' and 'Capabilities'. Below the 'About' tab, there are several parameters listed with their current values in text boxes or dropdown menus:

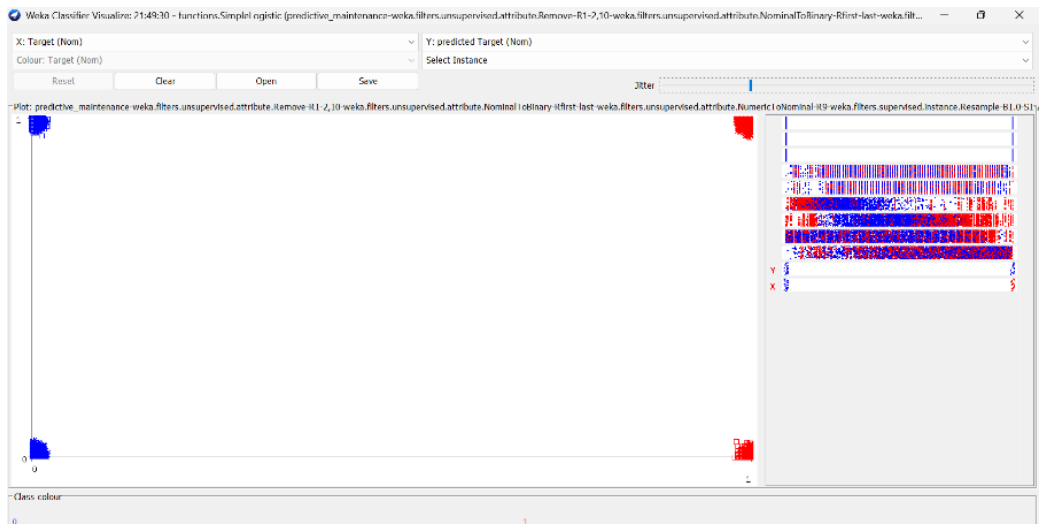
- batchSize: 100
- debug: False
- doNotCheckCapabilities: False
- errorOnProbabilities: False
- heuristicStop: 50
- maxBoostingIterations: 500
- numBoostingIterations: 0
- numDecimalPlaces: 2
- useAIC: False
- useCrossValidation: True
- weightTrimBeta: 0.0

At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

- Different parameters

When trying to change the `heuristicStop` so the model could try to learn and optimize more than 50 iterations the accuracy did not change, and the same thing happened to `maxBoostingIterations` when increasing the number of iterations, the accuracy has stayed to 82.19%.

- **Visualize errors**



12) Bayes Network

The model can use all features but only models dependencies between them. Some features may be conditionally independent and ignored.

On our data every feature is directly linked to Target — so all are used.

```
Bayes Network Classifier
not using ADTree
#attributes=9 #classindex=8
Network structure (nodes followed by parents)
Type=M(2): Target
Type=L(2): Target
Type=H(2): Target
Air temperature [K] (5): Target
Process temperature [K] (9): Target
Rotational speed [rpm] (29): Target
Torque [Nm] (35): Target
Tool wear [min] (10): Target
Target(2):
LogScore Bayes: -102153.74379441359
LogScore BDeu: -102841.32867280956
LogScore MDL: -102850.64282791069
LogScore ENTROPY: -102053.94838573477
LogScore AIC: -102226.94838573477
```

```
Time taken to build model: 0.22 seconds
```

- The result:

```

Correctly Classified Instances      8941           89.41 %
Incorrectly Classified Instances    1059           10.59 %
Kappa statistic                    0.7882
Mean absolute error                 0.129
Root mean squared error             0.2796
Relative absolute error             25.81 %
Root relative squared error         55.914 %
Total Number of Instances          10000

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC       ROC Area  PRC Area  Class
               0.860   0.072   0.923     0.860   0.890     0.790   0.963   0.966     0
               0.928   0.140   0.869     0.928   0.898     0.790   0.963   0.960     1
Weighted Avg.   0.894   0.106   0.896     0.894   0.894     0.790   0.963   0.963

=== Confusion Matrix ===

   a    b  <-- classified as
4300  700 |    a = 0
 359 4641 |    b = 1

```

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4300	700
Actual 1	359	4641

-Default parameters

The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.bayes.BayesNet' classifier. The 'About' tab is active, displaying a description of the Bayes Network learning process. Below the description, various parameters are listed with their default values or selected options:

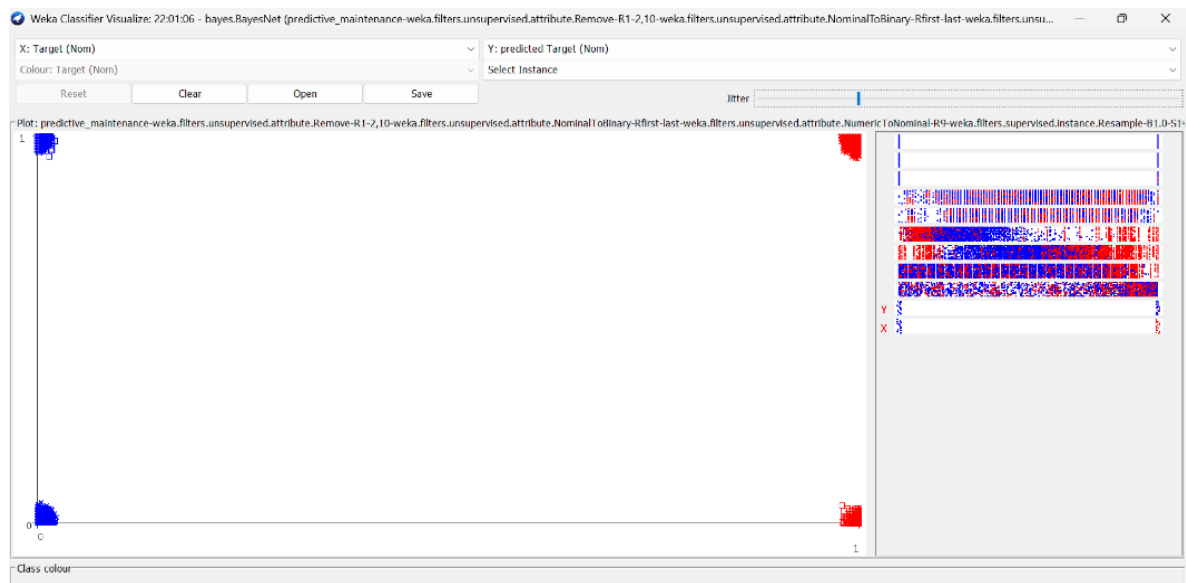
- BIFFile:** (empty text field)
- batchSize:** 100
- debug:** False
- doNotCheckCapabilities:** False
- estimator:** Choose **SimpleEstimator -A 0.5**
- numDecimalPlaces:** 2
- searchAlgorithm:** Choose **K2 -P 1 -S BAYES**
- useADTree:** False

At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

-Different parameters

When decreasing the Estimator number to 0.2 and to 0.1 the accuracy has **increase to 89.46%**.

-Visualize errors



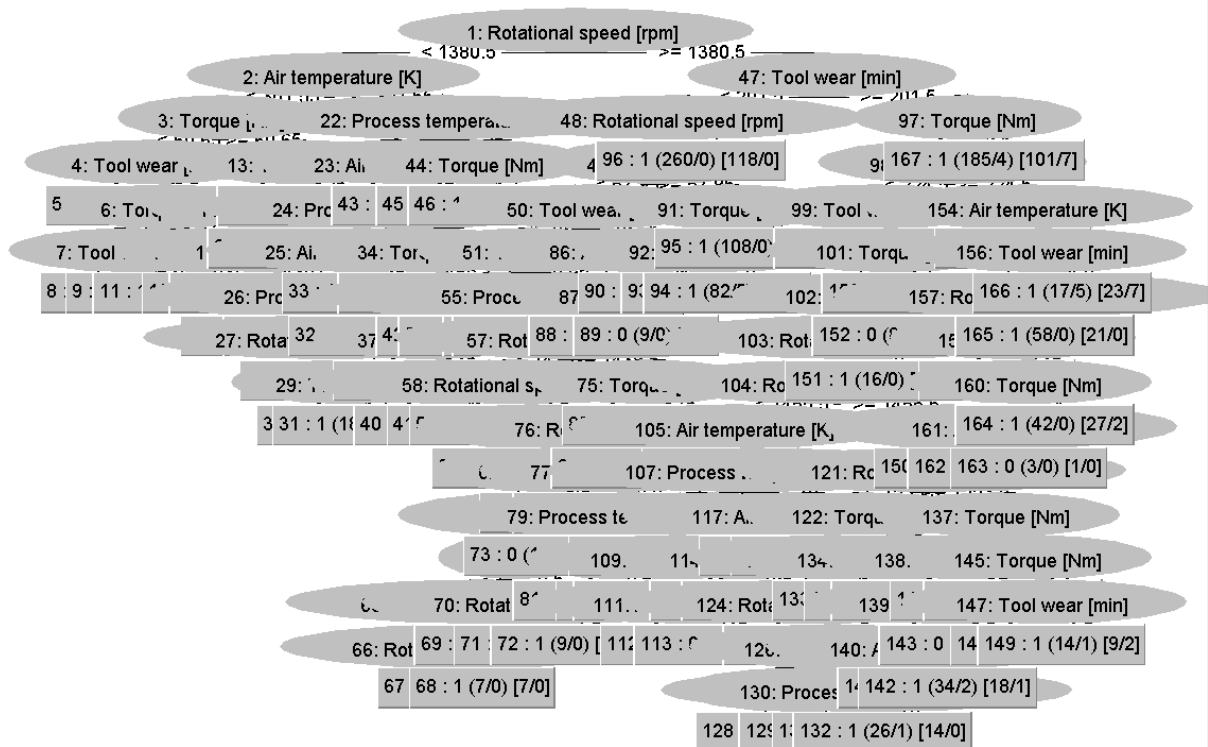
13) REP tree

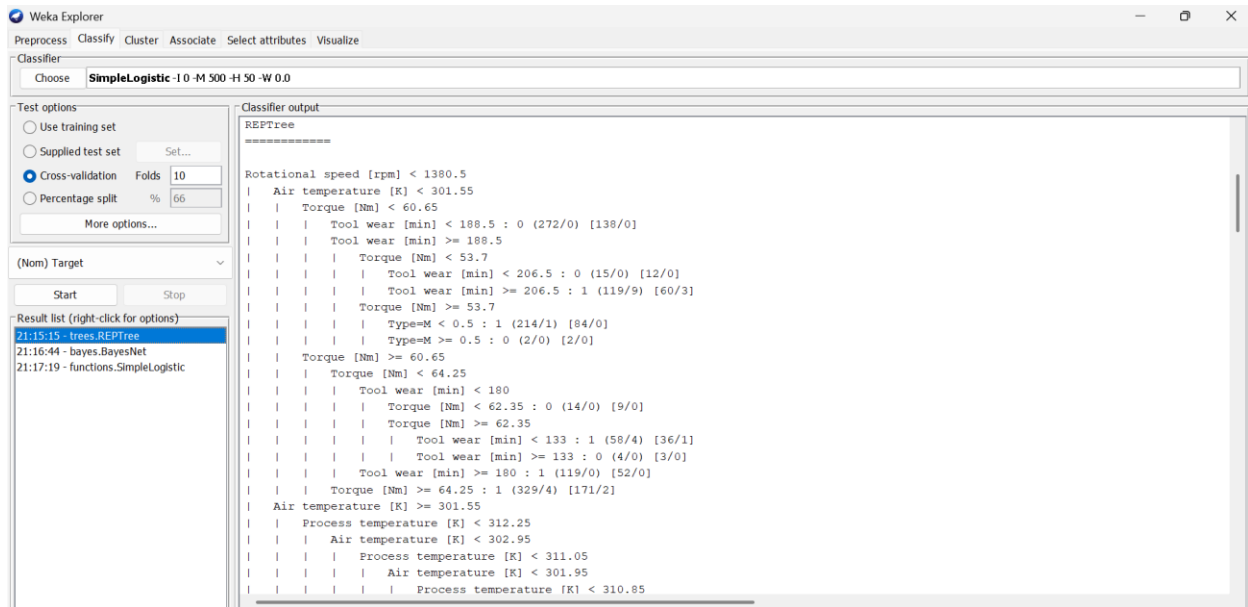
This model uses a subset of features, based on decision splits that most reduce error.

Focus on our data: Features used heavily:

- Rotational speed and Tool wear are top decision nodes.
- Torque and Process temperature are used deeper in the tree.
- Air temperature and Type are used occasionally, less dominant.
- Focus: Primarily on Rotational speed, Tool wear, Torque, and Process temperature. These features likely provide the highest information gain.

Tree View





- The result:

```

Correctly Classified Instances      9832           98.32 %
Incorrectly Classified Instances    168           1.68 %
Kappa statistic                    0.9664
Mean absolute error                 0.0247
Root mean squared error             0.1246
Relative absolute error             4.9345 %
Root relative squared error         24.9176 %
Total Number of Instances          10000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0.968    0.001    0.999    0.968    0.983     0.967    0.992    0.994     0
      0.999    0.032    0.969    0.999    0.983     0.967    0.992    0.985     1
Weighted Avg.   0.983    0.017    0.984    0.983    0.983     0.967    0.992    0.989

=== Confusion Matrix ===

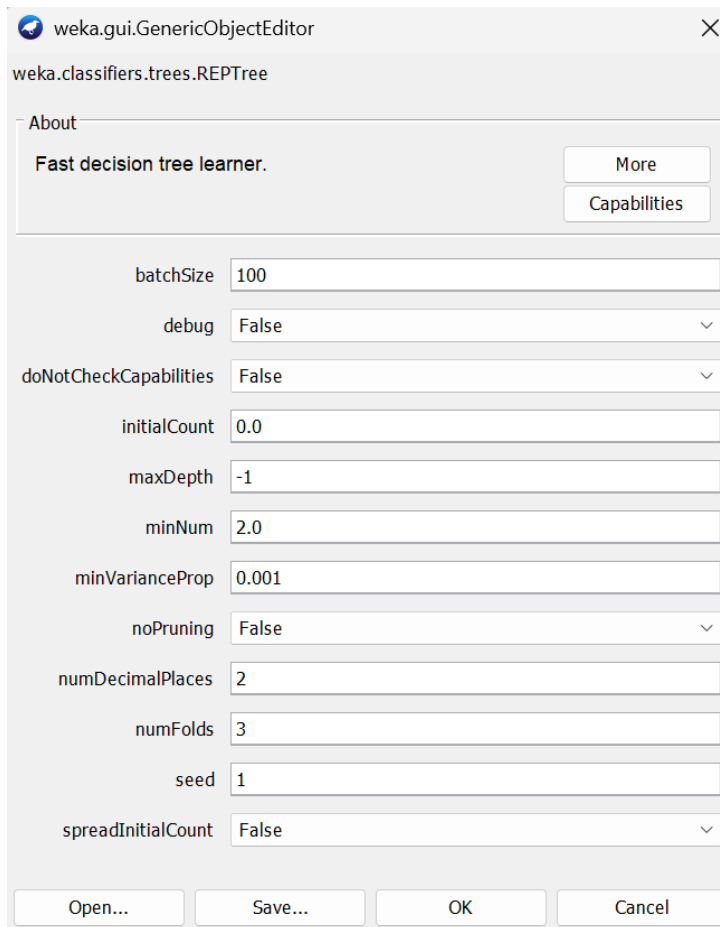
  a    b  <-- classified as
4839 161 |  a = 0
 7 4993 |  b = 1

```

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4839	161
Actual 1	7	4993

-Default parameter



The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.trees.REPTree' classifier. The 'About' tab is active, displaying the text 'Fast decision tree learner.' with 'More' and 'Capabilities' buttons. Below this, various parameters are listed with their default values: batchSize (100), debug (False), doNotCheckCapabilities (False), initialCount (0.0), maxDepth (-1), minNum (2.0), minVarianceProp (0.001), noPruning (False), numDecimalPlaces (2), numFolds (3), seed (1), and spreadInitialCount (False). At the bottom are 'Open...', 'Save...', 'OK', and 'Cancel' buttons.

Parameter	Value
batchSize	100
debug	False
doNotCheckCapabilities	False
initialCount	0.0
maxDepth	-1
minNum	2.0
minVarianceProp	0.001
noPruning	False
numDecimalPlaces	2
numFolds	3
seed	1
spreadInitialCount	False

-Different parameter

When changing the minNum to 1, the data becomes more detailed, and the accuracy has increased to 98.51

-Visualize errors



Using cross validation with the best Algorithm accuracy REP Tree 98.51% comparing to Bayes Network and simple logistics algorithms

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier Choose **REPtree -M 1 -V 0.001 -N 3 -S 1 -L -1 -I 0.0**

Test options

☐ Use training set

☐ Supplied test set Set...

☐ Cross-validation Folds

☒ Percentage split %

More options...

(Nom) Target

Start Stop

Result list (right-click for options)

- 14:28:41 - trees.REPtree
- 14:29:26 - trees.REPtree
- 14:29:34 - trees.REPtree

Classifier output

==== Validation on test split ====

Time taken to test model on test split: 0 seconds

==== Summary ====

Correctly Classified Instances	3336	98.1176 %
Incorrectly Classified Instances	64	1.8824 %
Kappa statistic	0.9624	
Mean absolute error	0.0276	
Root mean squared error	0.1328	
Relative absolute error	5.5234 %	
Root relative squared error	26.5505 %	
Total Number of Instances	3400	

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.964	0.002	0.998	0.964	0.981	0.963	0.988	0.992	0
	0.998	0.036	0.965	0.998	0.981	0.963	0.988	0.977	1
Weighted Avg.	0.981	0.019	0.982	0.981	0.981	0.963	0.988	0.984	

==== Confusion Matrix ====

a	b	<-- classified as	
1645	61	a = 0	
3	1691	b = 1	

Status OK Log x 0

14) Random Forest:

Random Forest predicts if the machine will fail by building many decision trees from different parts of the data. It combines their results to make a final call. This cuts down mistakes and gives a solid yes/no on failure, helping avoid surprise breakdowns.

```
Classifier output
Time taken to build model: 1.27 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      9854           98.54 %
Incorrectly Classified Instances    146           1.46 %
Kappa statistic                    0.9708
Mean absolute error                 0.0408
Root mean squared error             0.112
Relative absolute error             8.1607 %
Root relative squared error        22.4096 %
Total Number of Instances         10000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
0.974   0.004   0.996   0.974   0.985   0.971   0.999   0.999   0
0.996   0.026   0.975   0.996   0.986   0.971   0.999   0.999   1
Weighted Avg.   0.985   0.015   0.986   0.985   0.985   0.971   0.999   0.999

=== Confusion Matrix ===
      a    b  <-- classified as
4872  128 |    a = 0
  18 4982 |    b = 1
```

The Accuracy: The model correctly classified **98.54%** of instances, meaning it has a high prediction accuracy.

The confusion matrix:

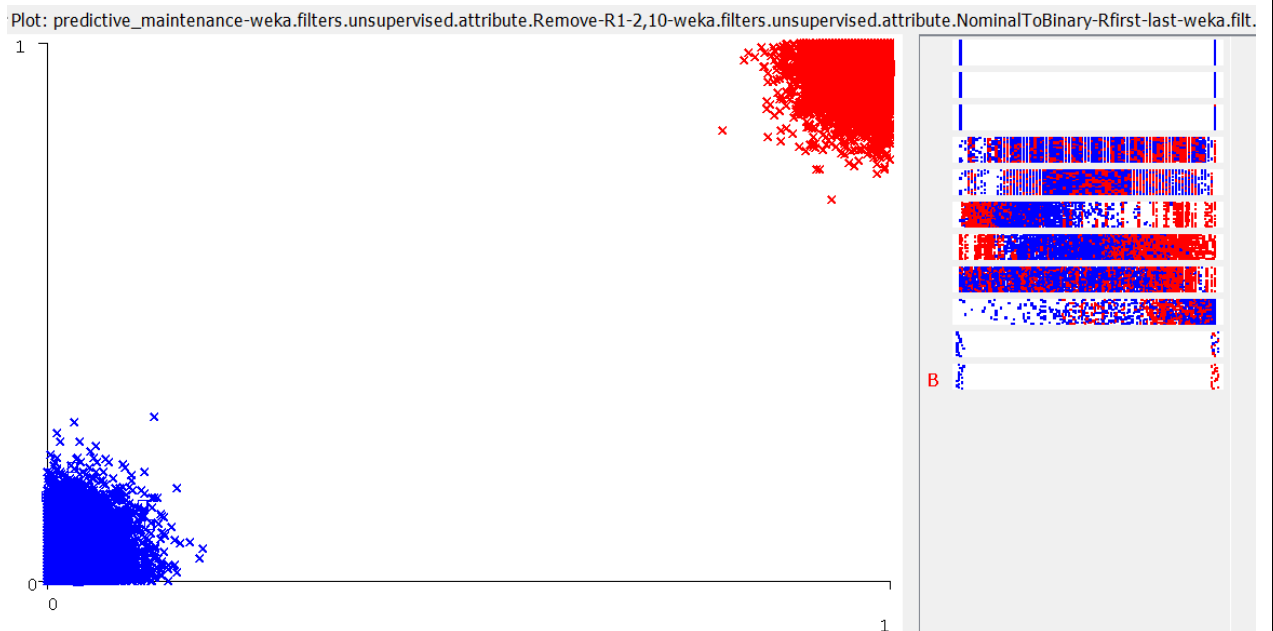
	Predicted 0	Predicted 1
Actual 0	4872	120
Actual 1	18	4902

-The parameter

bagSizePercent	100
batchSize	50
breakTiesRandomly	False
calcOutOfBag	False
computeAttributeImportance	False
debug	False
doNotCheckCapabilities	False
maxDepth	10
numDecimalPlaces	2
numExecutionSlots	1
numFeatures	0
numIterations	100
outputOutOfBagComplexityStatistics	False
printClassifiers	False
seed	1
storeOutOfBagPredictions	False

- I set the Random Forest to build 100 trees with a max depth of 10, using all data for training each tree. We kept the process simple by skipping extra calculations like feature importance and out-of-bag errors, used one CPU core, and fixed the seed for consistent results. This balances accuracy and speed.

-The visualization



15) Naive Bayes

looks at each data feature separately and calculates the chance the machine will fail based on those features. It's fast and works well if features don't depend on each other.

```
Classifier output

Time taken to build model: 0.13 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      8938           89.38 %
Incorrectly Classified Instances    1062           10.62 %
Kappa statistic                     0.7876
Mean absolute error                 0.1306
Root mean squared error             0.2804
Relative absolute error              26.1158 %
Root relative squared error         56.0796 %
Total Number of Instances          10000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0.860    0.072    0.923    0.860    0.890      0.789    0.963    0.965     0
      0.928    0.140    0.869    0.928    0.897      0.789    0.963    0.960     1
Weighted Avg.   0.894    0.106    0.896    0.894    0.894      0.789    0.963    0.963

=== Confusion Matrix ===

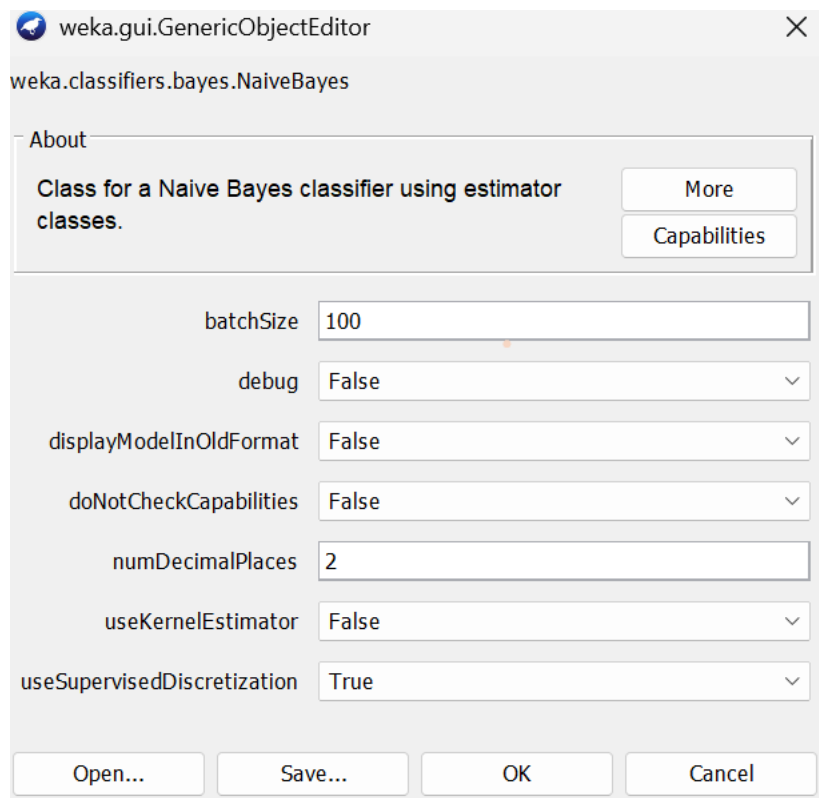
  a    b  <-- classified as
4299  701 |    a = 0
 361 4639 |    b = 1
```

The accuracy: is 89.38%, which is decent but still lower than Random Forest, making it less reliable and too simple for our data.

The confusion matrix:

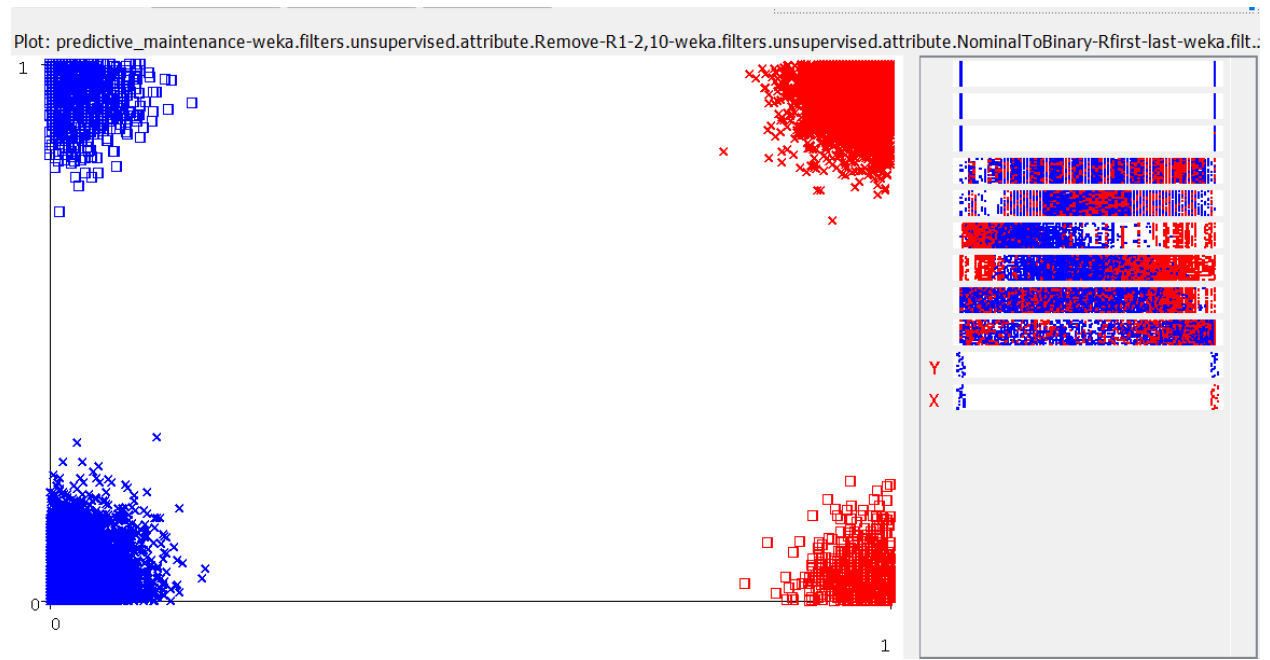
	Predicted 0	Predicted 1
Actual 0	4299	701
Actual 1	361	4636

- The parameter



I kept all parameters at their default settings except for *useSupervisedDiscretization*, which I enabled because it significantly improved accuracy from 82% to 89%. This happens by converting continuous data into class-based discrete categories, helping Naive Bayes perform better.

- The visualization



14) IBk (K-Nearest Neighbors)

is a simple algorithm that decides if a machine will fail by looking at the closest similar machines in the data. It checks the 'k' nearest neighbors and picks the majority outcome to predict failure or not. This way, it helps tell apart machines that might fail from those that won't by comparing them to machines with similar behavior.

```
Classifier output
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      9692           96.92 %
Incorrectly Classified Instances    308           3.08 %
Kappa statistic                    0.9384
Mean absolute error                 0.0312
Root mean squared error             0.1549
Relative absolute error             6.2427 %
Root relative squared error        30.9815 %
Total Number of Instances         10000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0.938   0.000   1.000    0.938   0.968     0.940   1.000    1.000    0
      1.000   0.062   0.942    1.000   0.970     0.940   1.000    1.000    1
Weighted Avg.   0.969   0.031   0.971    0.969   0.969     0.940   1.000    1.000

=== Confusion Matrix ===
      a    b  <-- classified as
4692  308 |    a = 0
      0 5000 |    b = 1
```

The accuracy is 96.92%, which is good but still lower than Random Forest. KNN is considered less reliable and might not fully capture the data's complexity.

The confusion matrix:

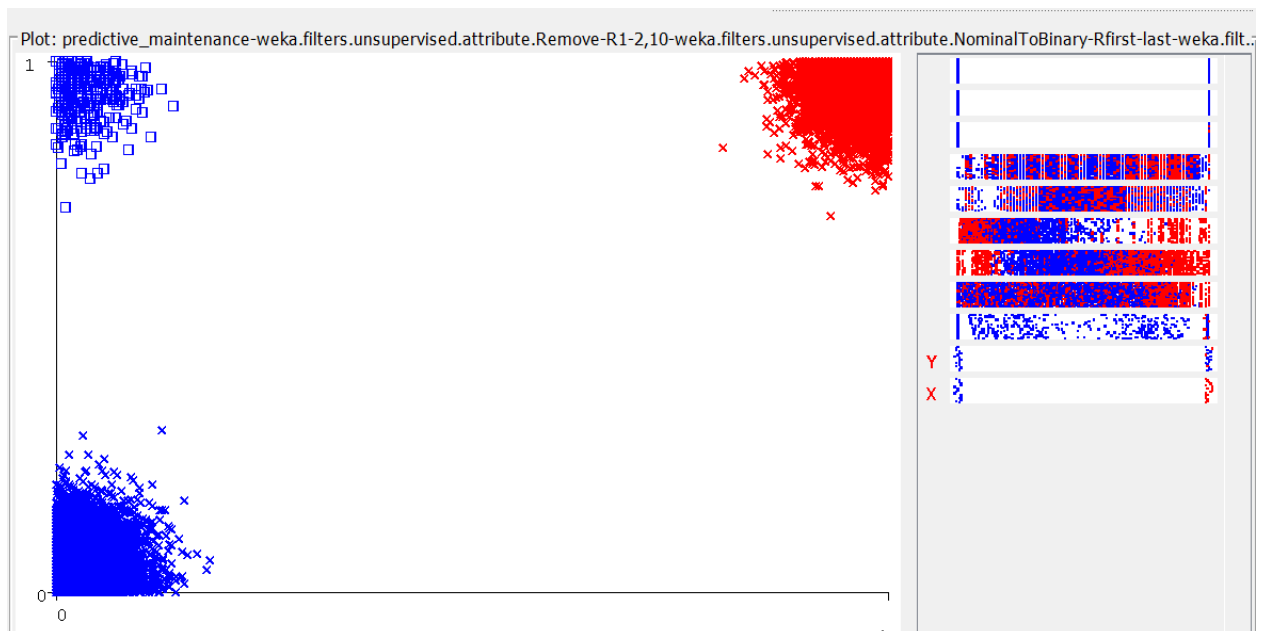
	Predicted 0	Predicted 1
Actual 0	4692	308
Actual 1	0	5000

- The parameter

KNN	6
batchSize	100
crossValidate	False
debug	False
distanceWeighting	Weight by 1/distance
doNotCheckCapabilities	False
meanSquared	False
nearestNeighbourSearchAlgorithm	Choose LinearNNSearch -A "weka.:
numDecimalPlaces	2
windowSize	0

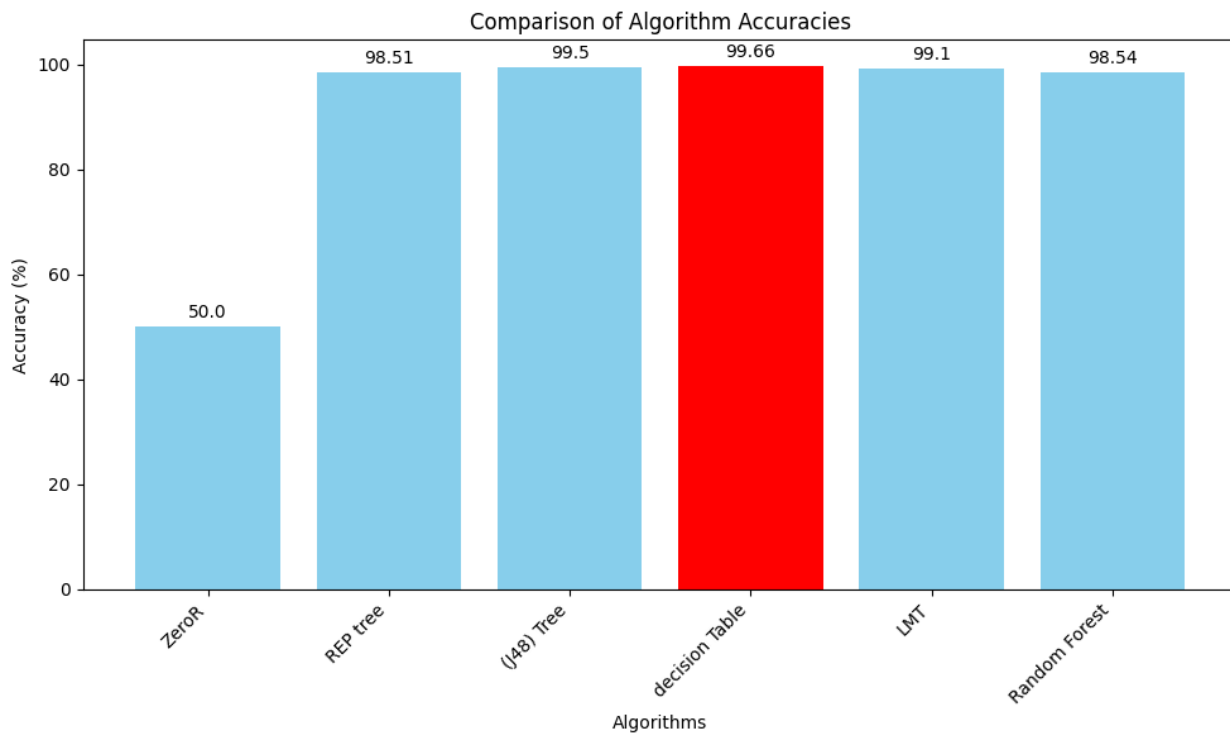
We set KNN to use 6 nearest neighbors and process data in batches of 100 without cross-validation. The model gives more weight to closer neighbors using a simple linear search method. Results are shown with two decimal places, and all neighbors are considered without any window limit. This setup keeps things simple while focusing on the most relevant data points for accurate predictions.

- The visualization



The graph

A comparison of all the algorithms with the best accuracy:



The conclusion:

We tested different algorithms to predict if a machine will fail or not. The best one was the **Decision Table**, with the highest accuracy of **99.66%**. The **J48 Decision Tree** and **AdaBoostM1** also did really well, with accuracy above **99%**.

Other models like **Random Forest** and **REP Tree** worked well too, around **98.5%** accuracy. But simpler models like **Naive Bayes** and **OneR** didn't do as good, which means they might not handle the data complexity well.

The baseline model, **ZeroR**, only got **50%** accuracy, so we need better models to make good predictions.

In short, tree-based models and Decision Table worked best for this data and can help predict machine failures more accurately.

References:

- *About AdaBoostM1 :* <https://www.sciencedirect.com/topics/computer-science/adaboost-algorithm>
- *About Logistic Regression (LR):*
https://youtu.be/J5CM5pdCFgA?si=nhhlnL17WPPz_r8h
- *About Decision Tree (J48):*
<https://youtu.be/YbDTv8Y4cvI?si=shHWhgWx301EXB0r>
- <https://www.geeksforgeeks.org/building-a-machine-learning-model-using-j48-classifier/>
- *About OneR :*
<https://christophm.github.io/interpretable-ml-book/rules.html>
- *About DecisionStump:*
<https://medium.com/geekculture/decision-stump-b8e93c1f54d7>
- *Machine Predictive Maintenance Classification dataset:*
<https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification?>
- *OpenAI for understanding:*
<https://chatgpt.com/>

