Kingdom of Saudi Arabia
Ministry of Education
Umm Al-Qura University
031

المملكة العربية السعودية
وزارة التعليم
جامعة أم القرى
٠٣١

Machine Predictive Maintenance Classification

Names:	**ID number:**	**Task distribution:**
Ruyuf alharbi	44510064	Leader/Supervisor. Creating&Editing the report. Writing the introduction, the goal, and the conclusion. Applied three algorithms: 1- Randomforset 2- Naïve bayes 3- K- nearest
Alaa Abdulrahman ajlan	445013943	Cleaned the data. Applied three algorithms: 1- Logisitc regression. 2- OneR. 3- Decision tree.
Dana Hassan Bafaqih	445013918	Cleaned the data. Applied three algorithms: 1- DecisionStump. 2- AdaBoost. 3- DecisionTable
Hadeel Mohammad Turki	445006064	Created the accuracy graph. Applied three algorithms: 1- REP tree. 2- Bayes Net. 3- SimpleLogistic.
Refan Salem Almfarriji	445000906	Writing about the ZeroR. Applied three algorithms: 1- SMO. 2- Lmt.

		3- JiRp.

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

A comparison of all the algorithms with the best accuracy 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).

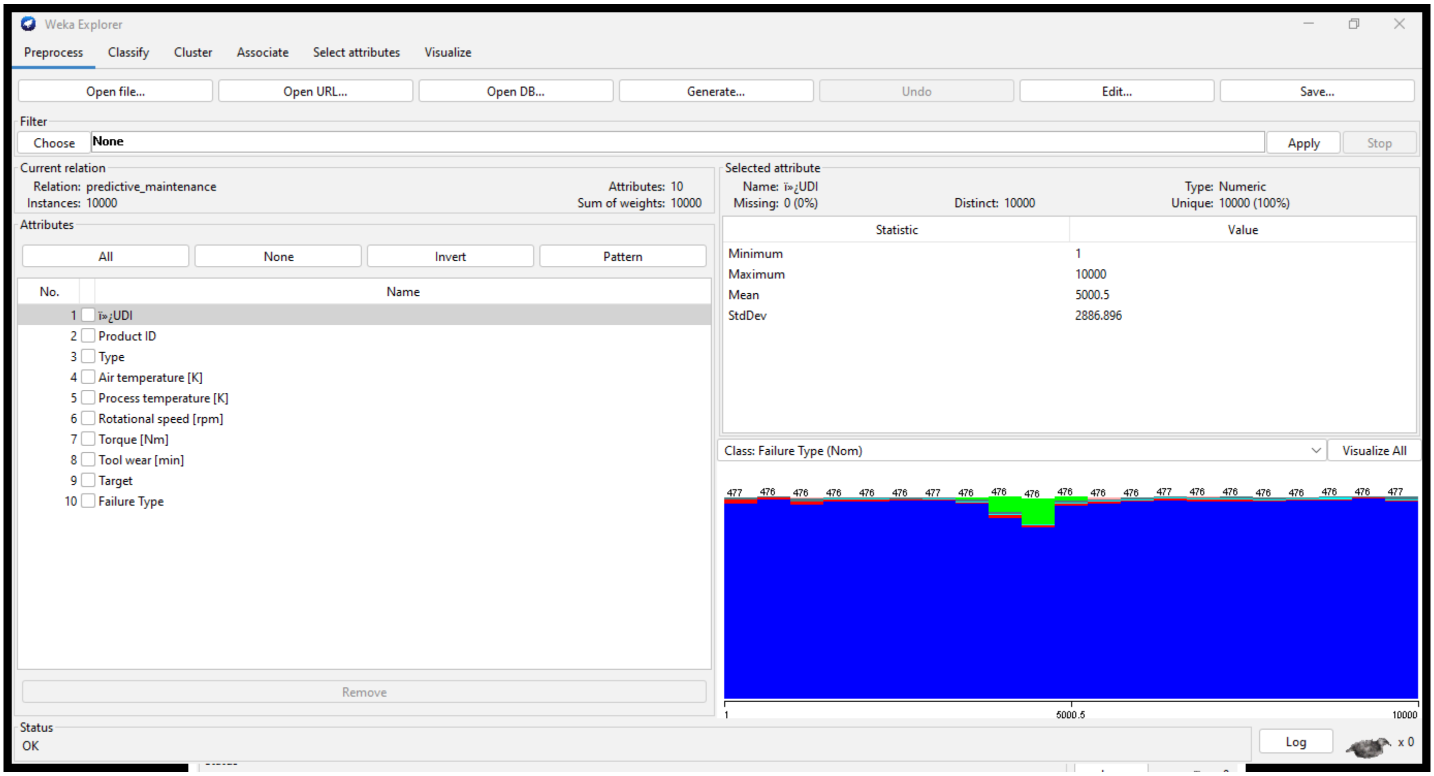
Features include:

- **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 (σ = 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.

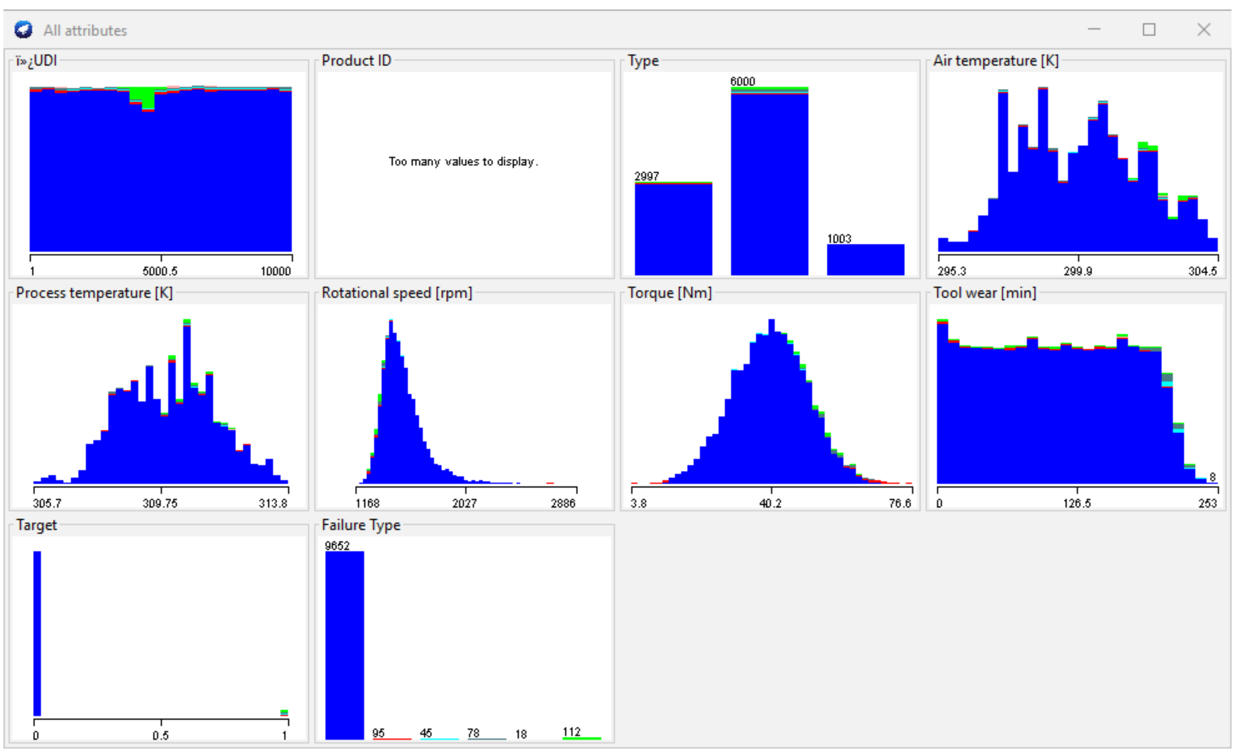
Important: The dataset contains **two target variables**:

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

- The dataset



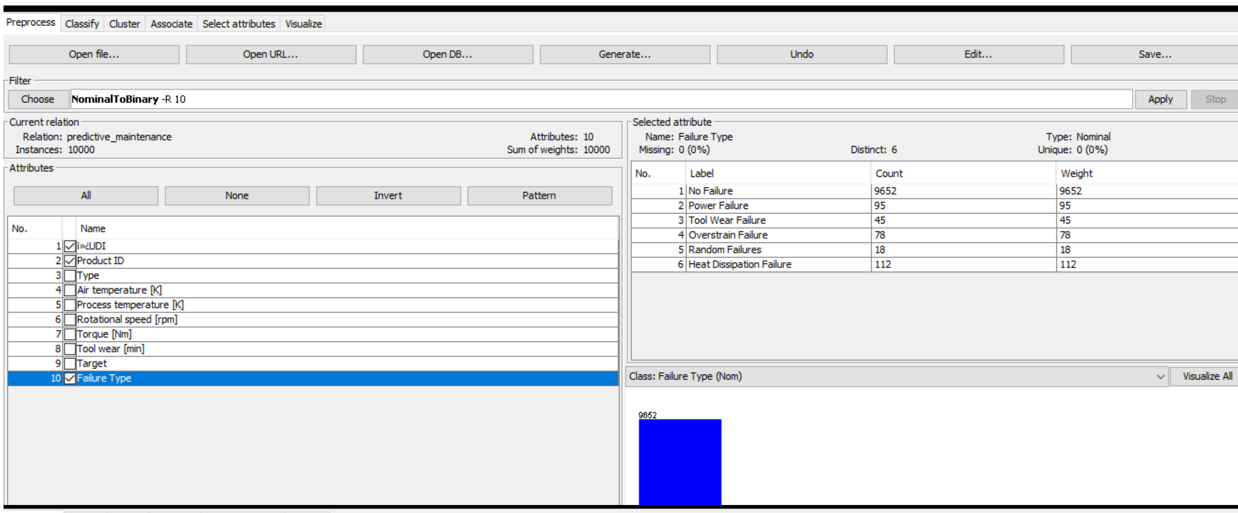
- Visualization

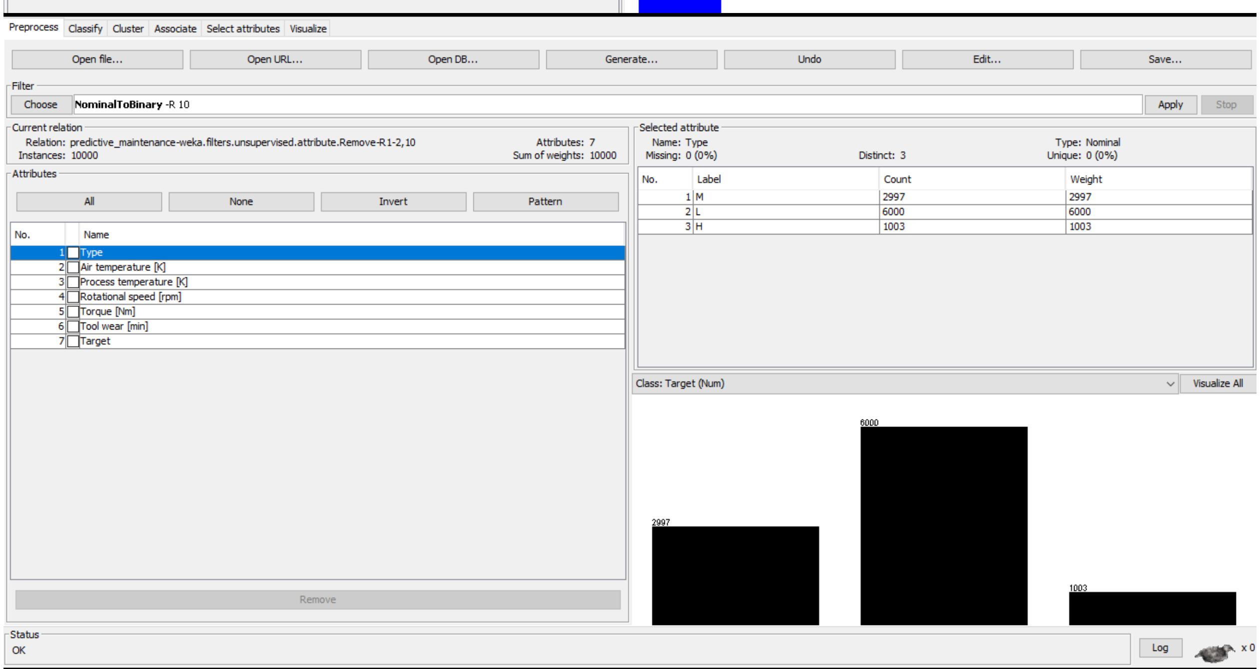


2)Data cleaning preparation:

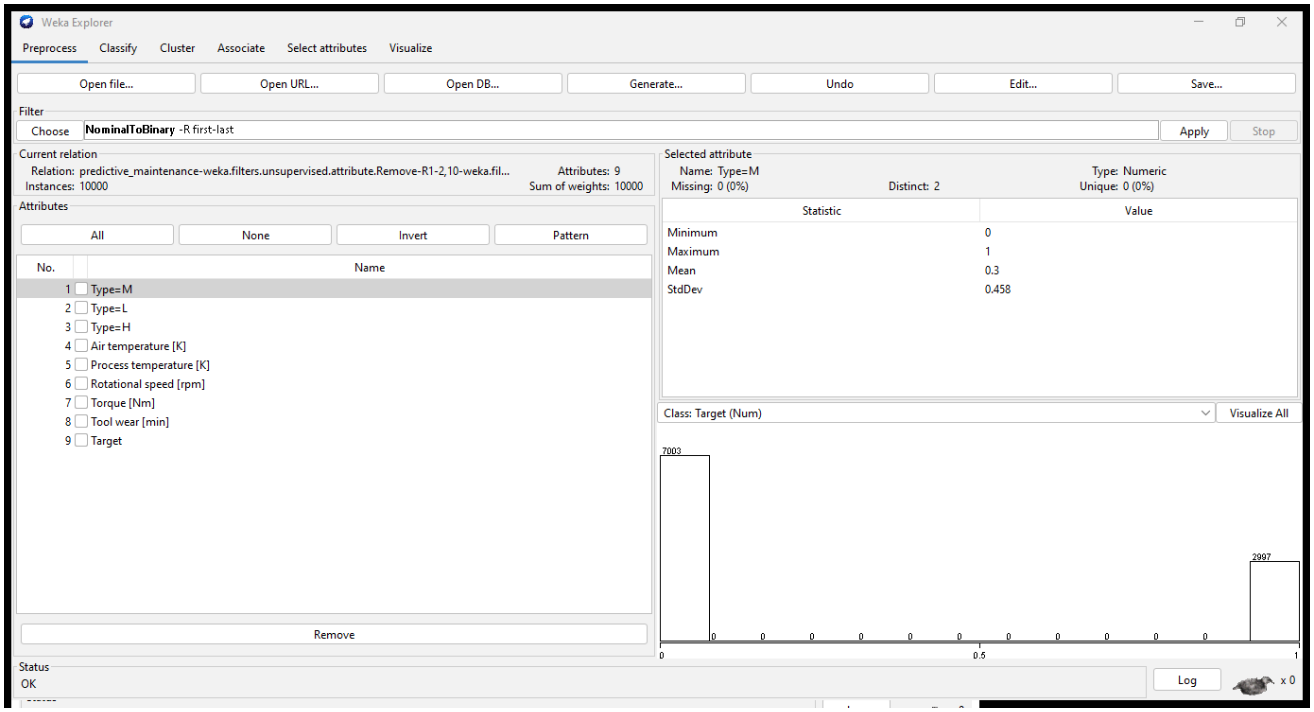
We applied some filter in order to clean and prepare the dataset for model training and evaluation:

- 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.

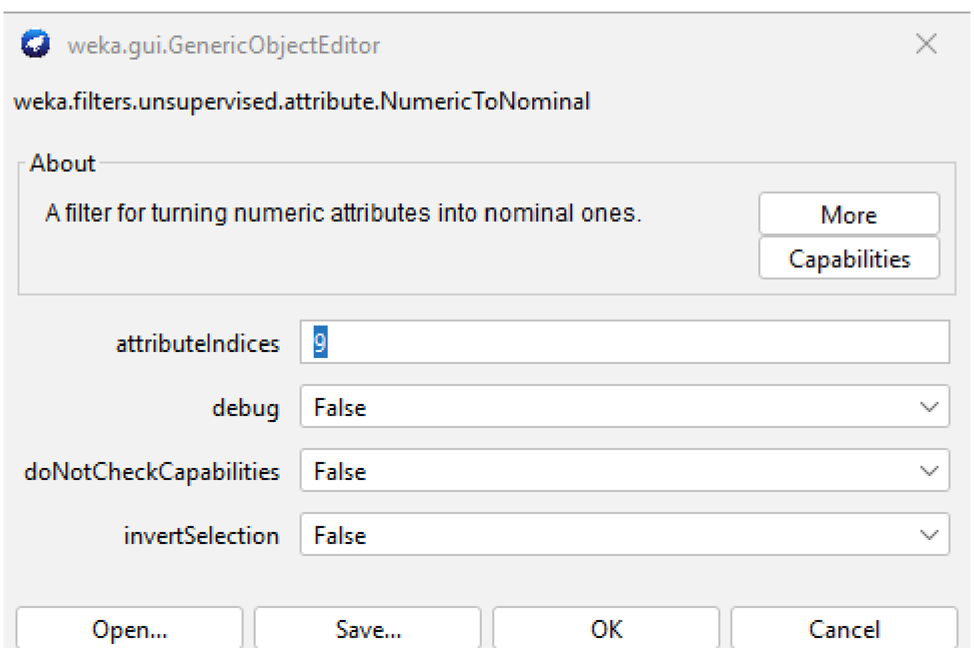


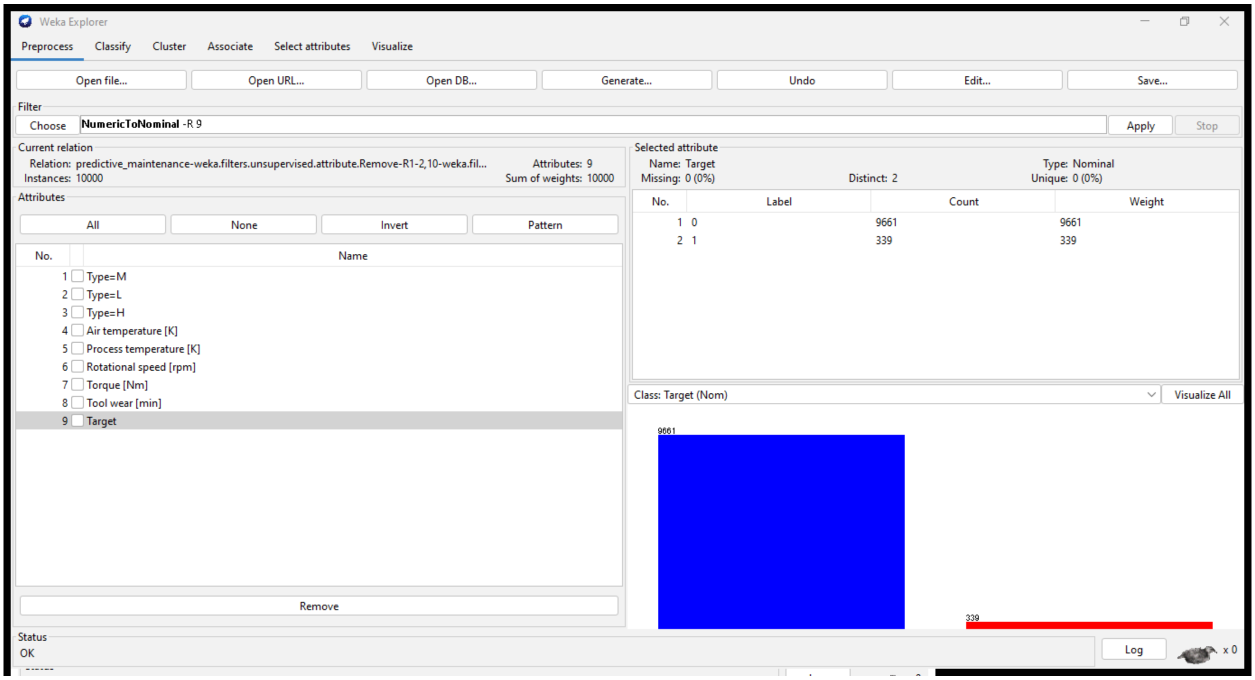


- We converted the "Type" column to binary to make it easier to handle. Additionally, some algorithms cannot work with categorical values directly and require numerical input for processing.

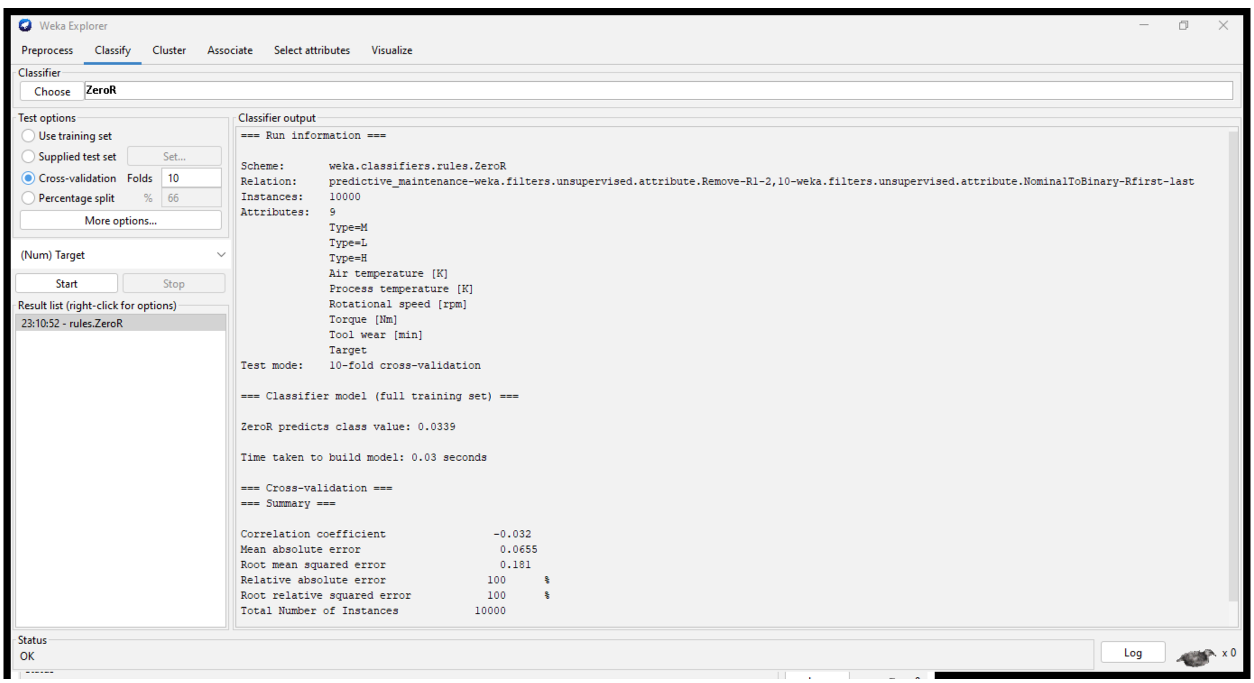


- The target column had numerical values, and since we want to use it for classification, we converted it to nominal.
  - We chose only the target column

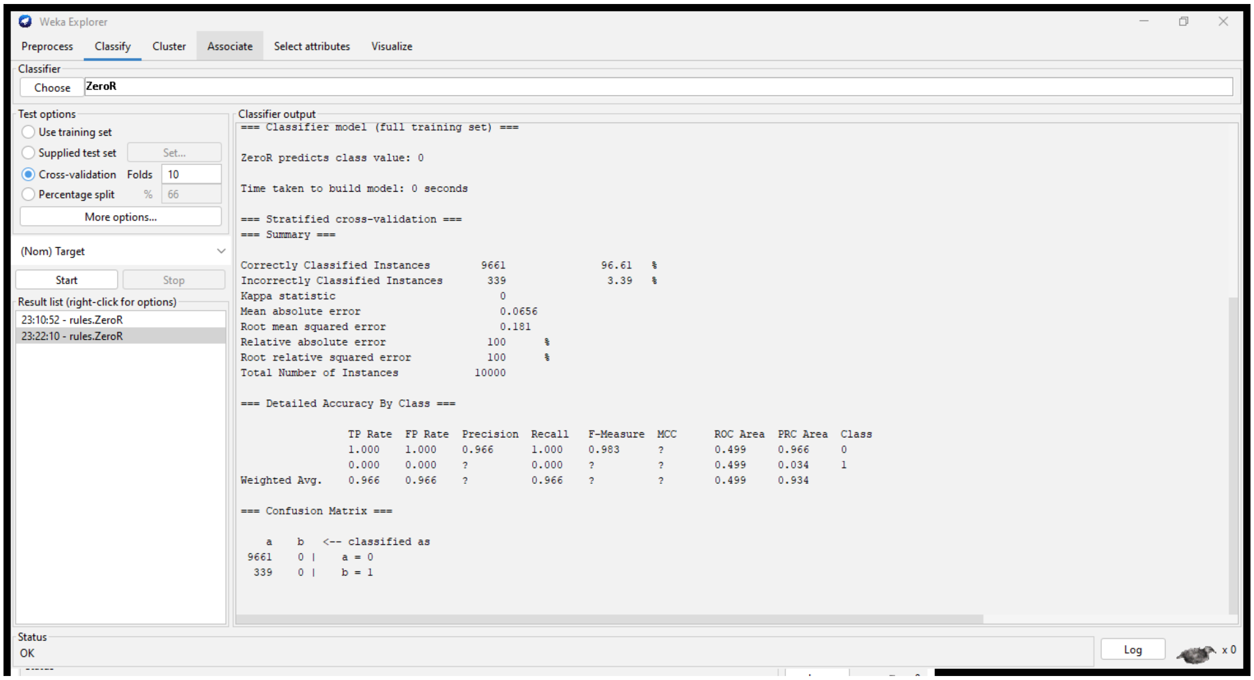




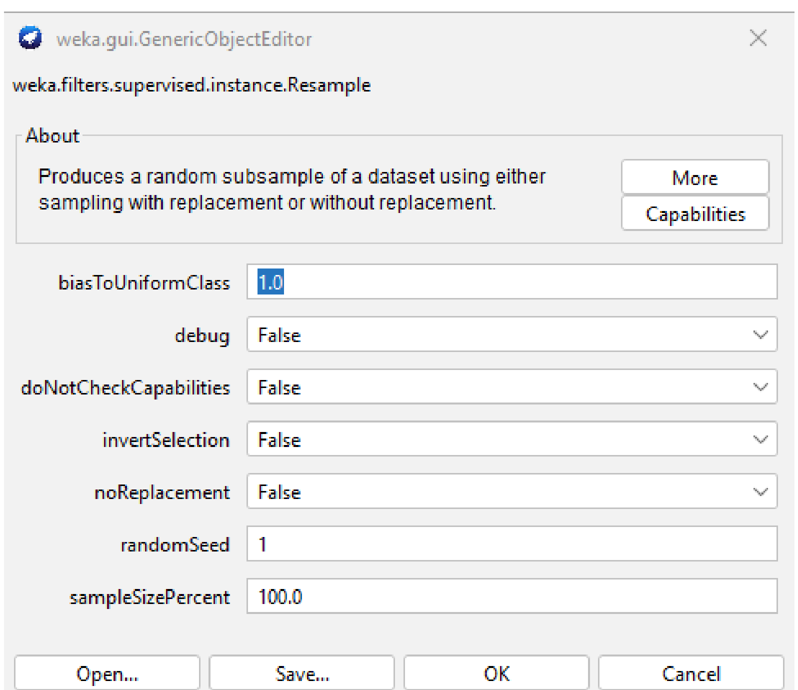
- The result before converting the column was treated as a regression problem, so the model predicted a number instead of a class.

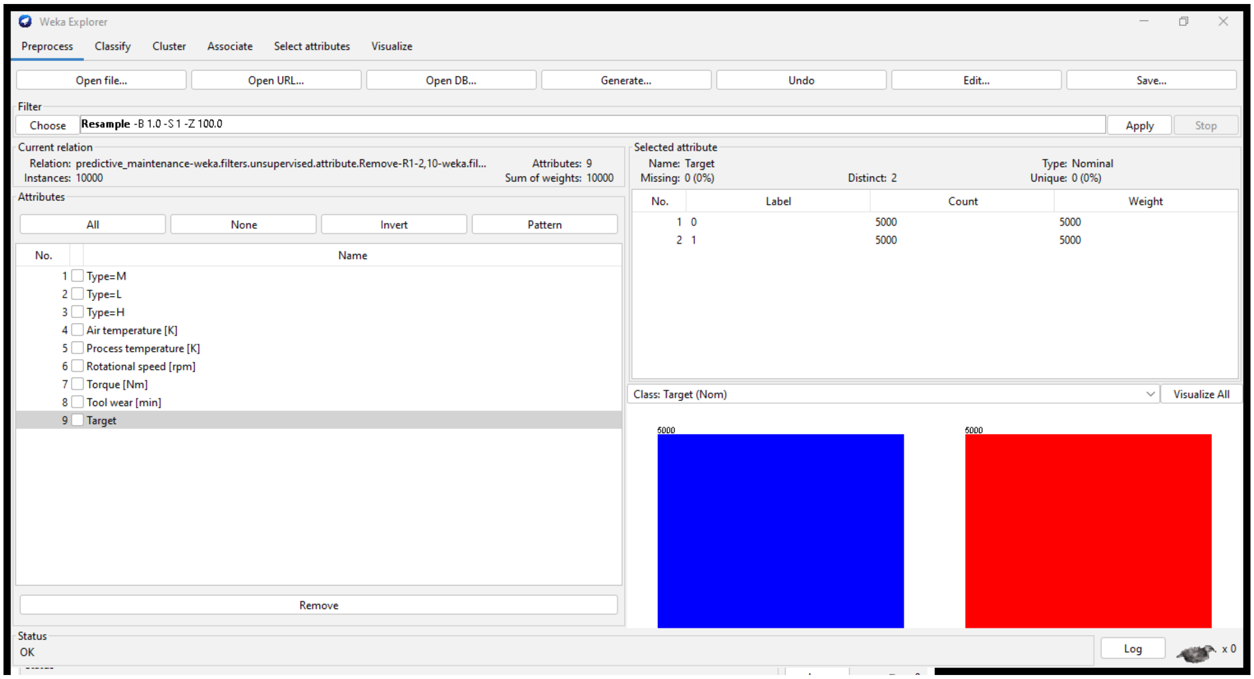


- The result after the conversion

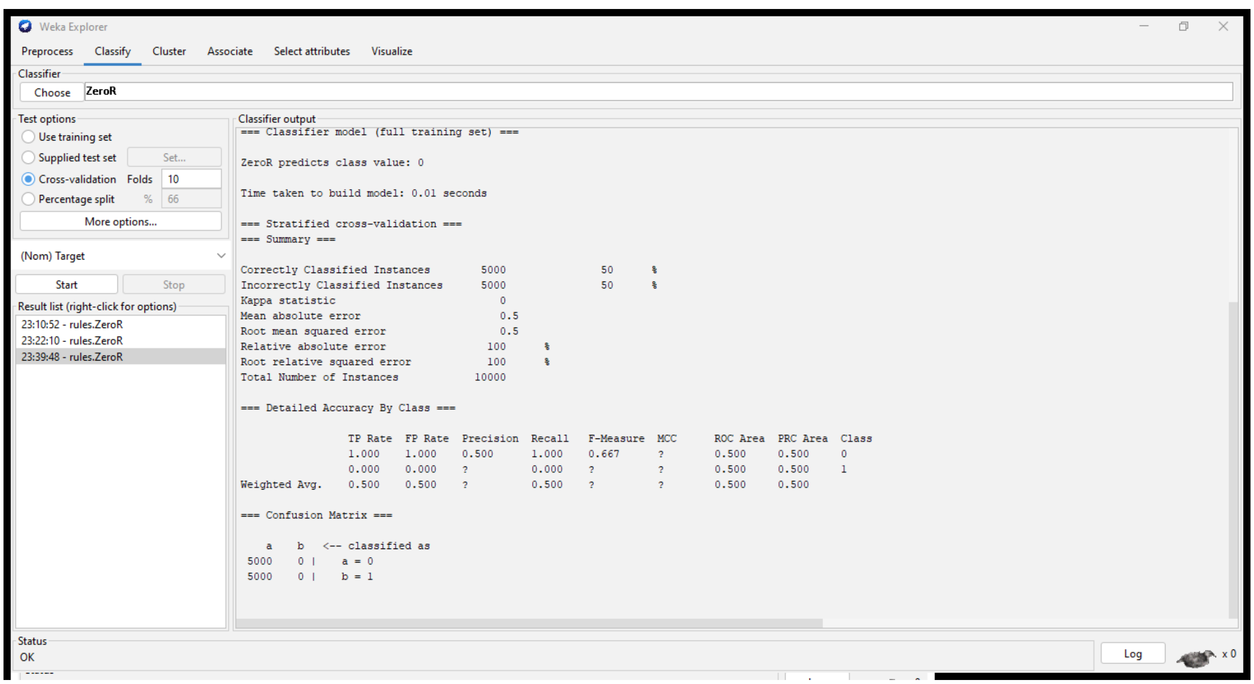


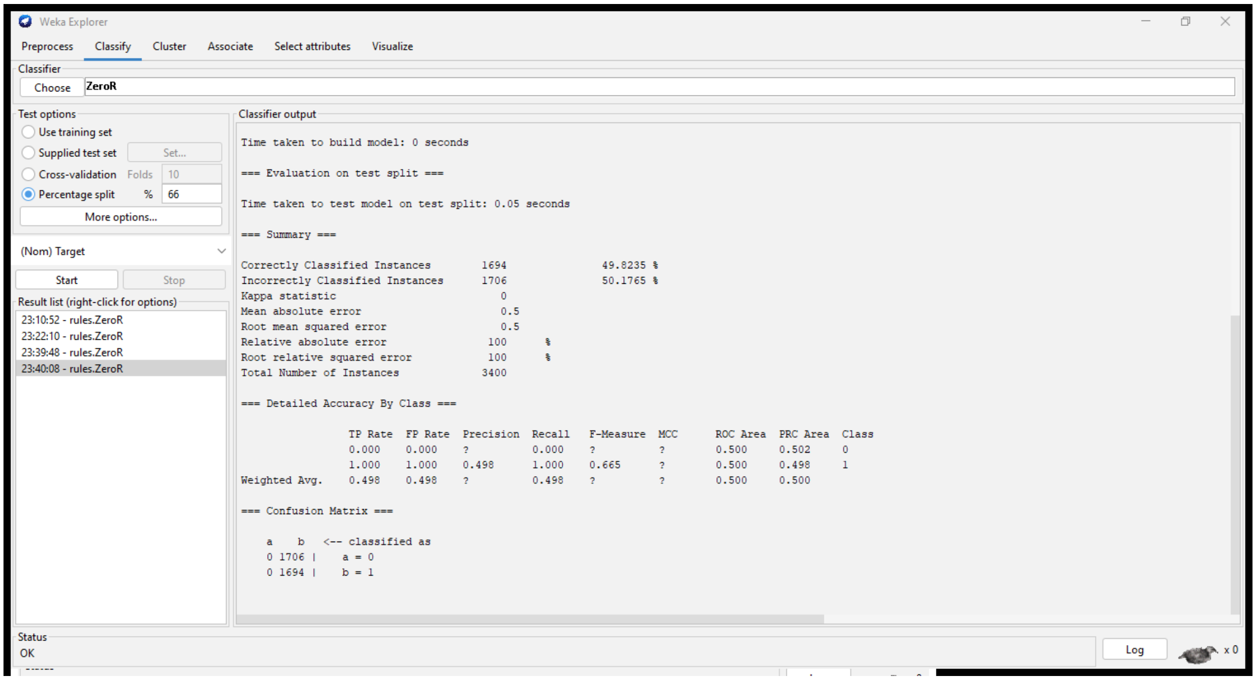
- As we can see, our data is imbalanced, and this causes a problem in classification. To solve this, we used the **Resample filter**, which helps make the data more balanced.
  - **Bias to Uniform Class = 1**, which means it will try to make the class distribution equal.
  - **Sample Size = 100%**, so 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.

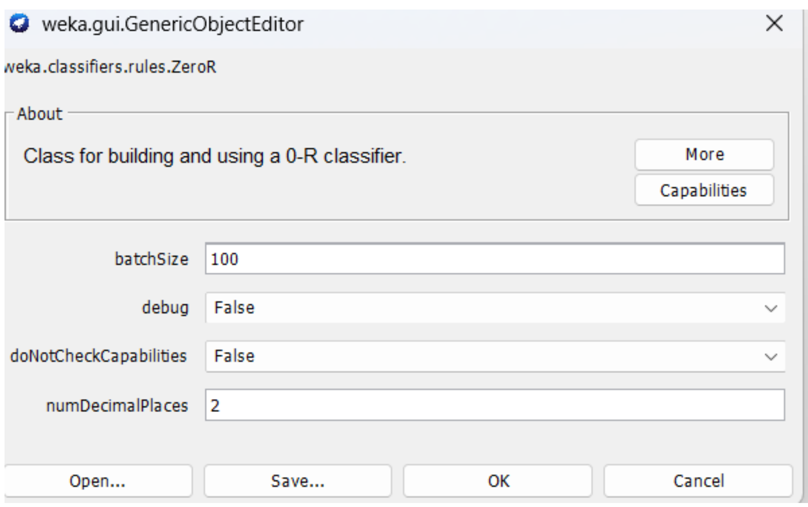
Machine learning algorithms and methods:

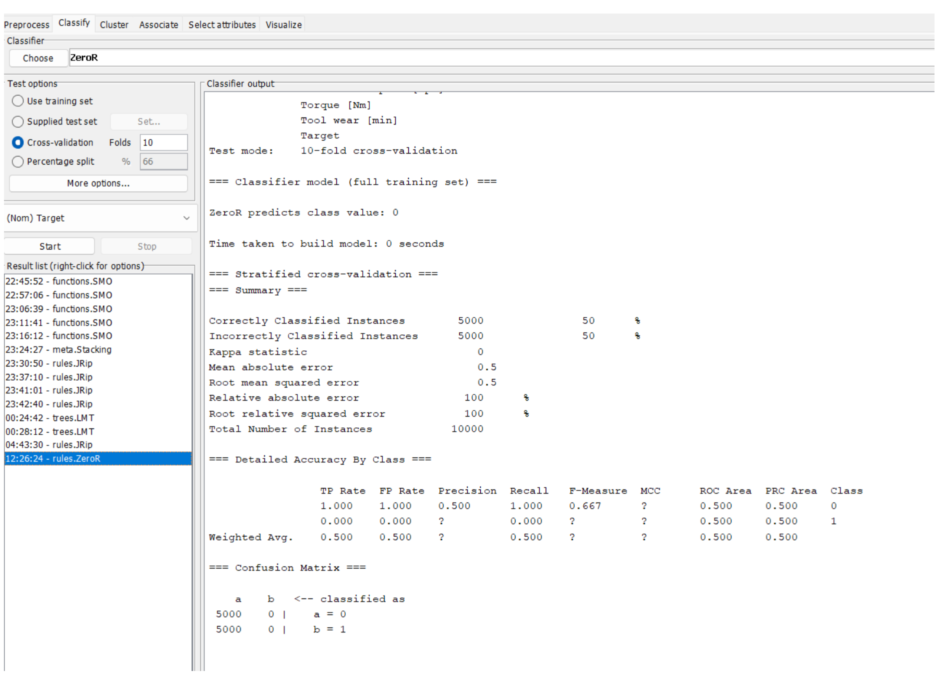
1) ZeroR – Zero Rule Classifier

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

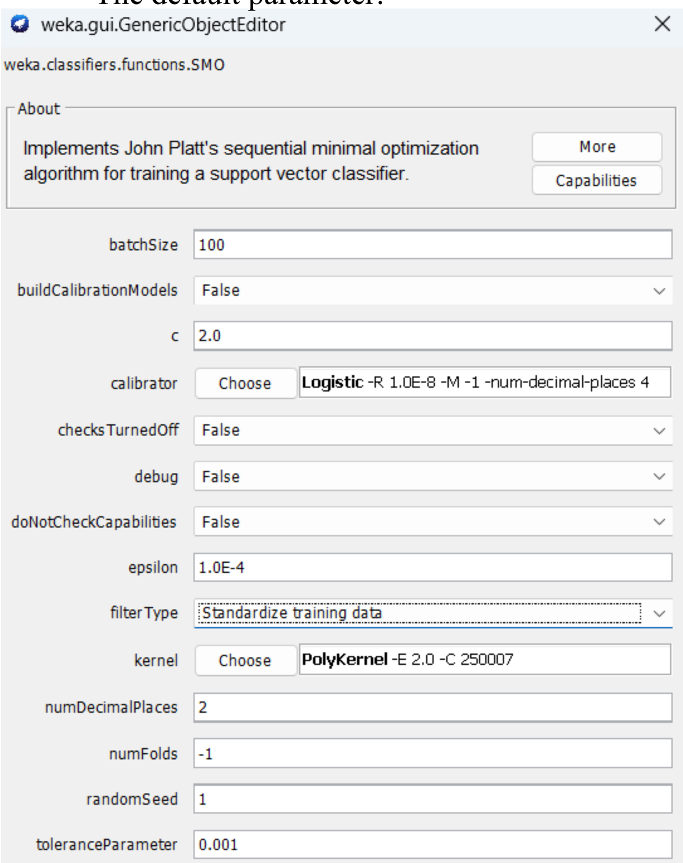


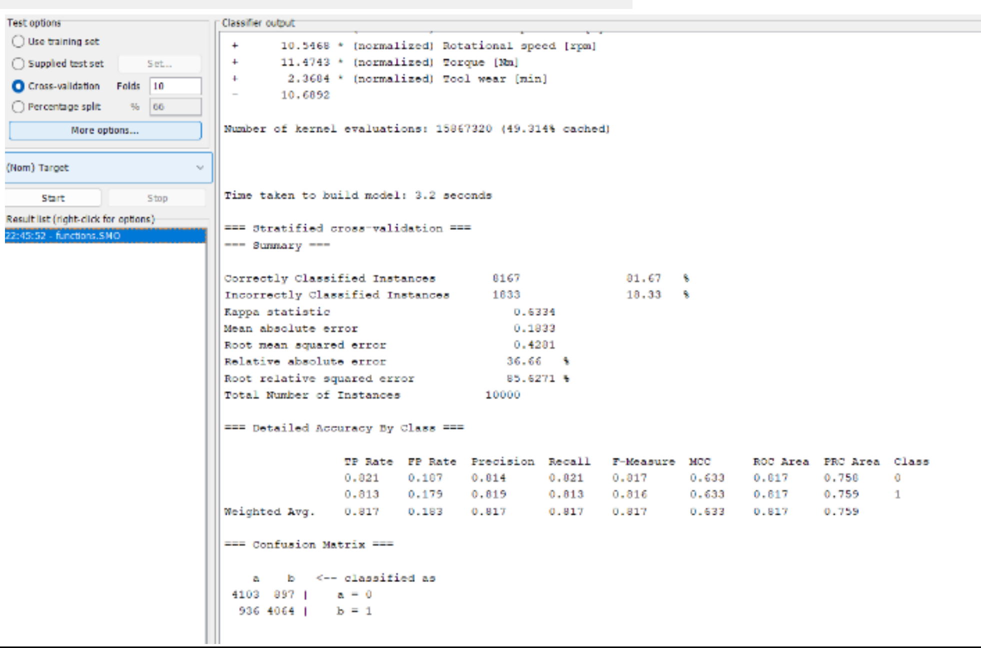


2) SMO - Sequential Minimal Optimization

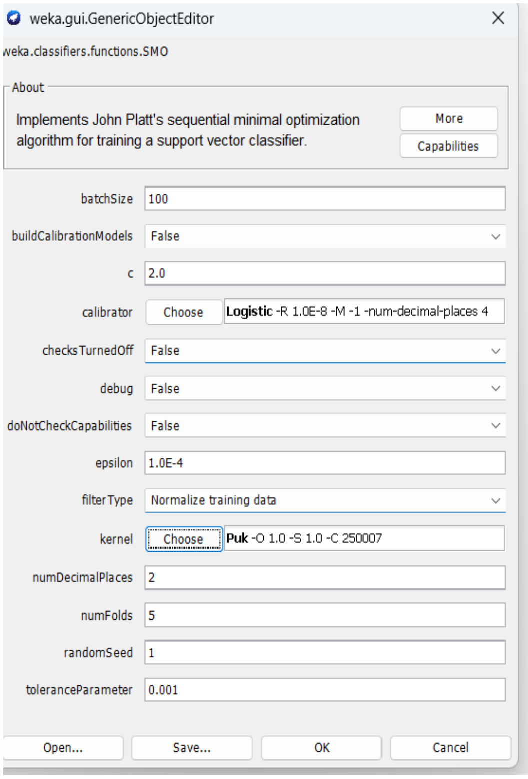
SMO trains an SVM model by breaking down the large optimization problem into a series of smaller problems involving only two variables at a time. Each small problem is solved analytically, without the need for complex iterative methods. This process is repeated until the entire model is trained efficiently

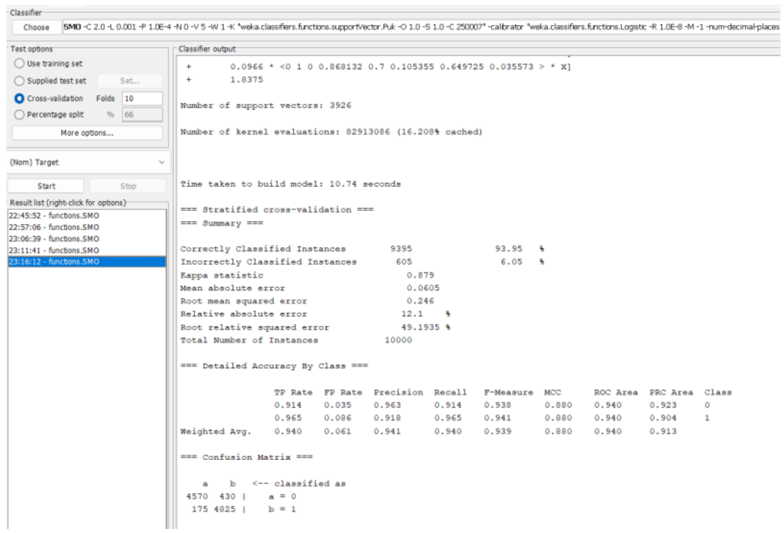
- The default parameter:

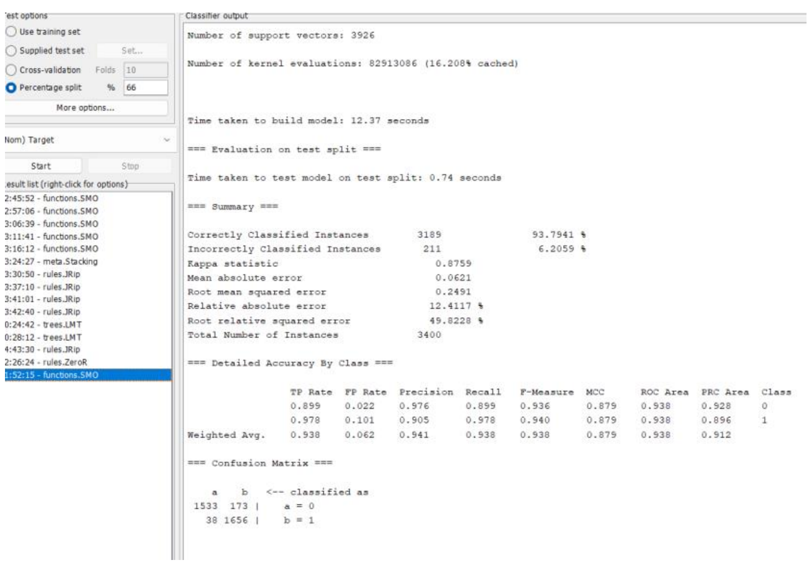




Best Accuracy Result:
SMO achieved its highest accuracy (**93.95%**) after changing the **kernel** to *PUK*, applying the **Normalize filter** to the training data, and **increasing the number of folds**





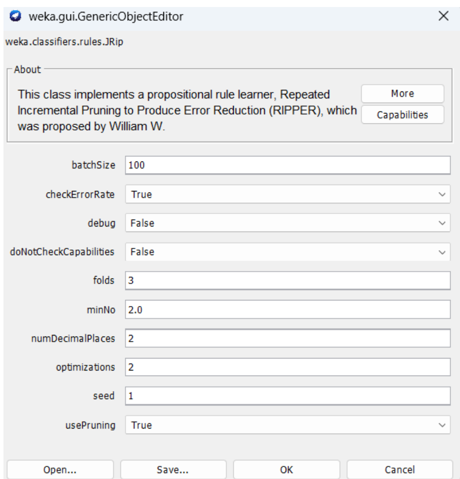


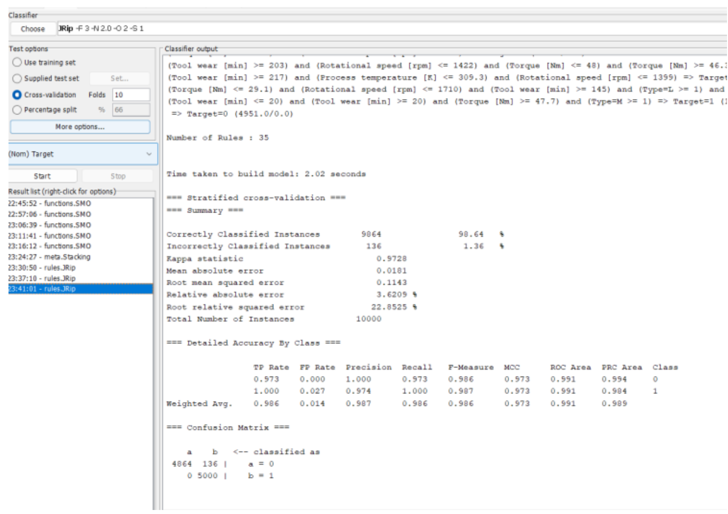
3) JRip – Java Repeated Incremental Pruning

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:

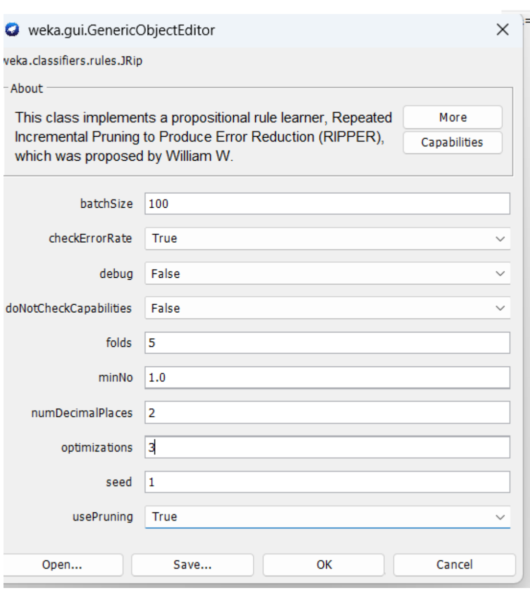
The default JRip parameters already achieved high accuracy (98.64%). However, I made a slight adjustment to one of the parameters to further improve the performance

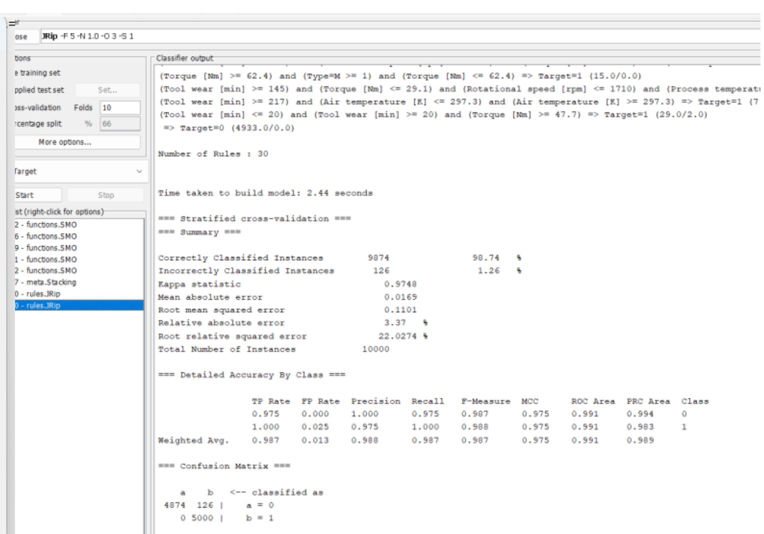


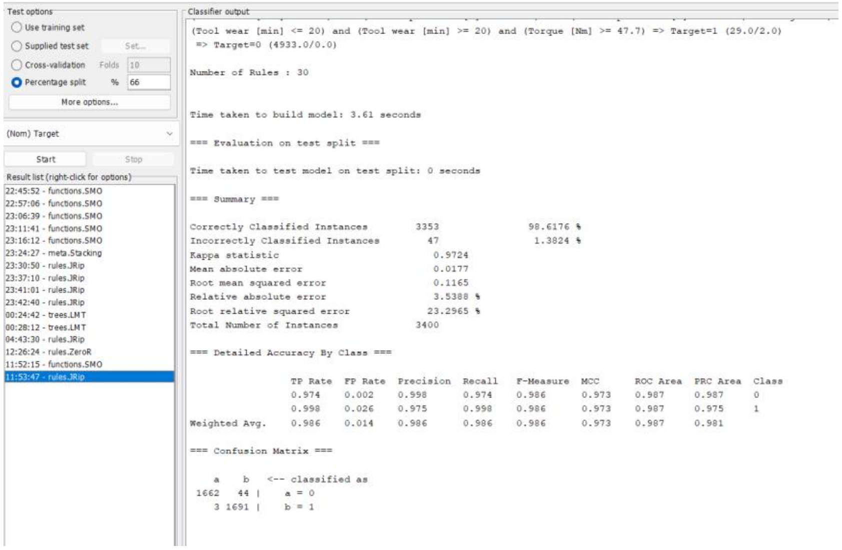


Best Accuracy Result:

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

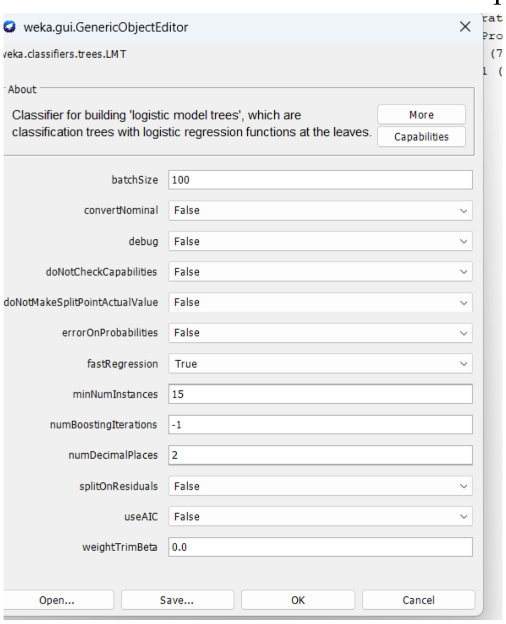


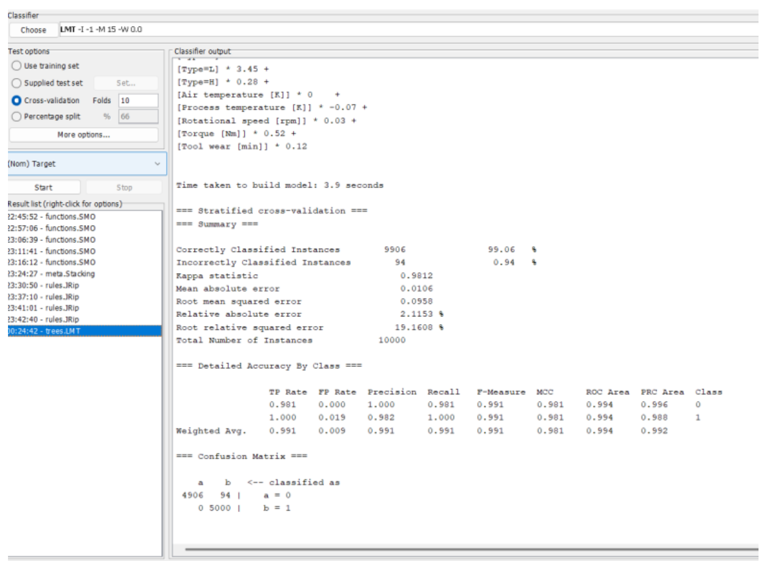




4) LMT – Logistic Model Tree

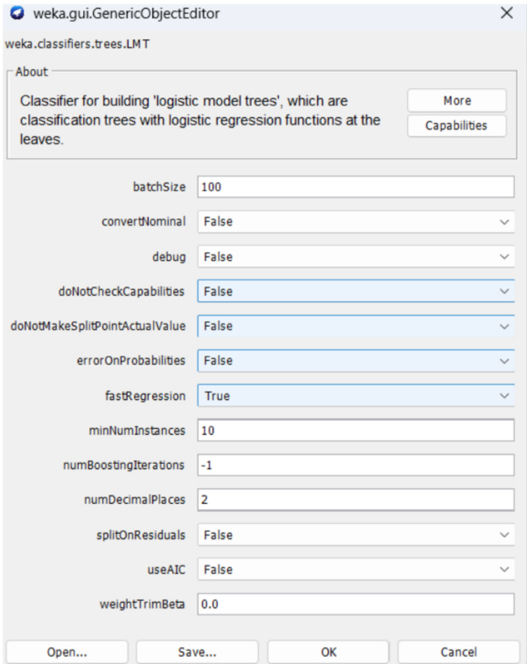
LMT is a hybrid classification algorithm that combines decision trees and logistic regression. It builds a tree where each leaf node contains a logistic regression model, allowing the algorithm to handle both non-linear and linear patterns in data effectively.

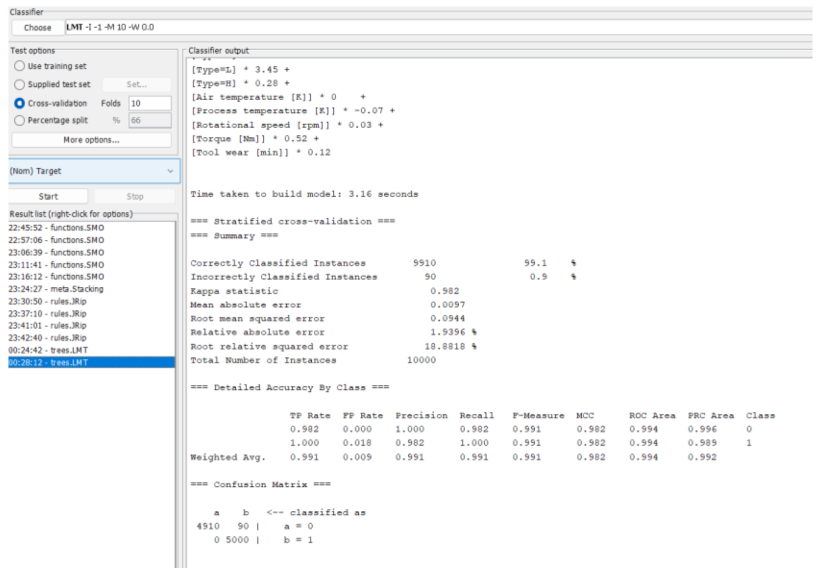


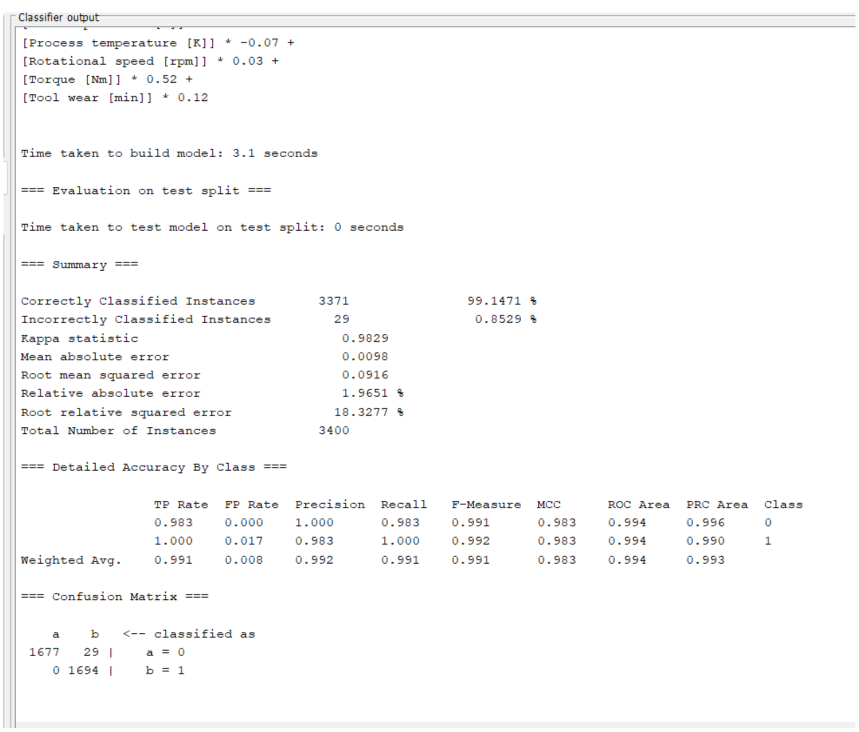


Best Accuracy:

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





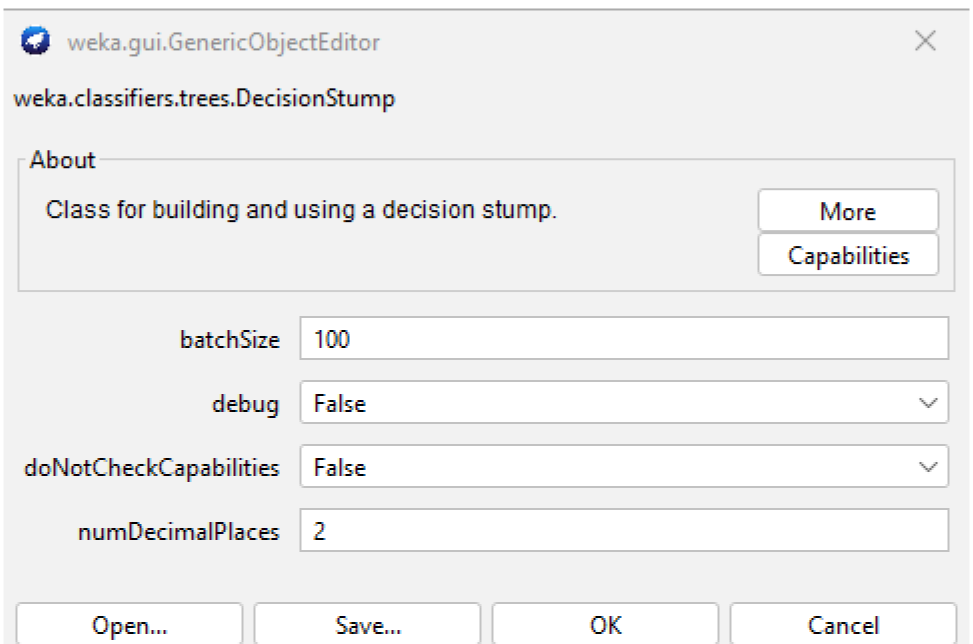


5) DecissionStump:

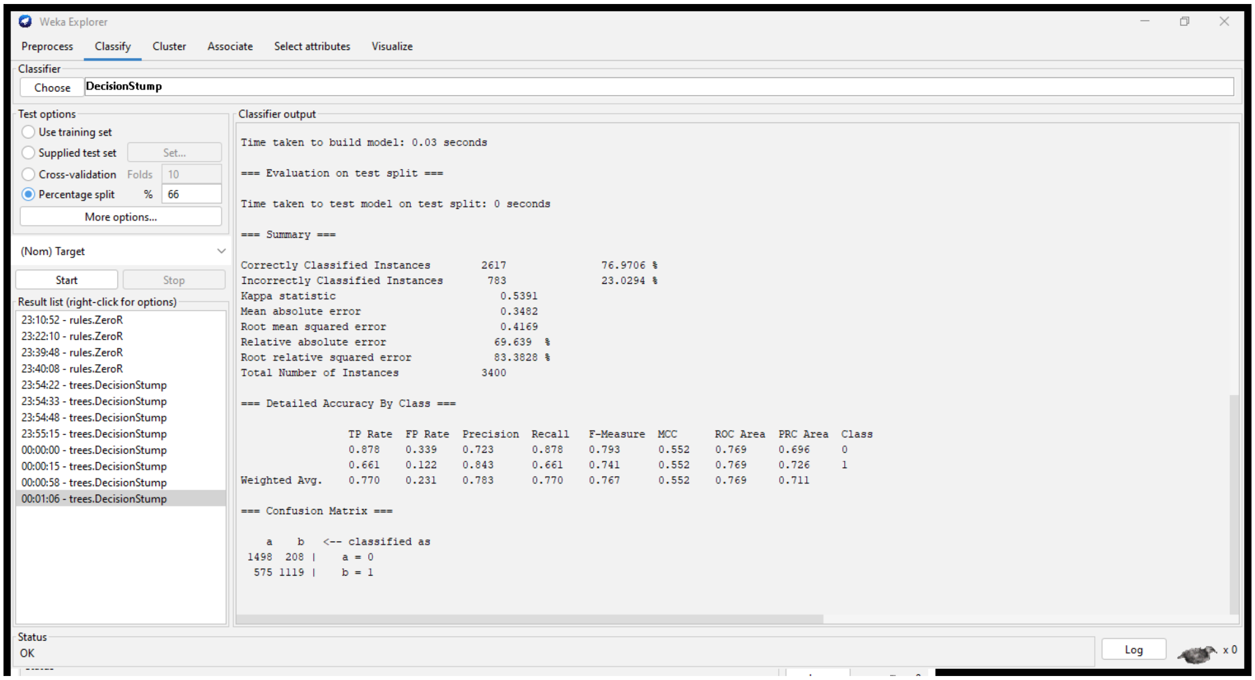
The Decision Stump focuses on only **one feature** at a time and finds a threshold that best separates the data.
In our data, it makes the decision based on the **'Rotational speed'** feature.

```
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:**



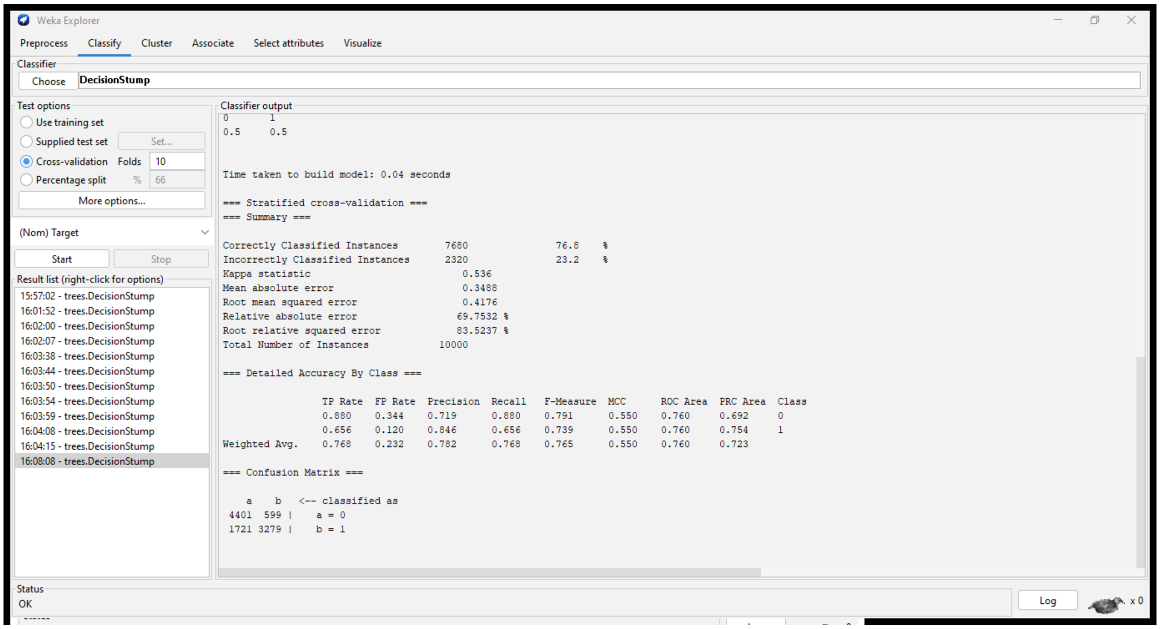
- **The result:**



The accuracy: 76.97%

The confusion matrix:

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



The accuracy: 76.8%

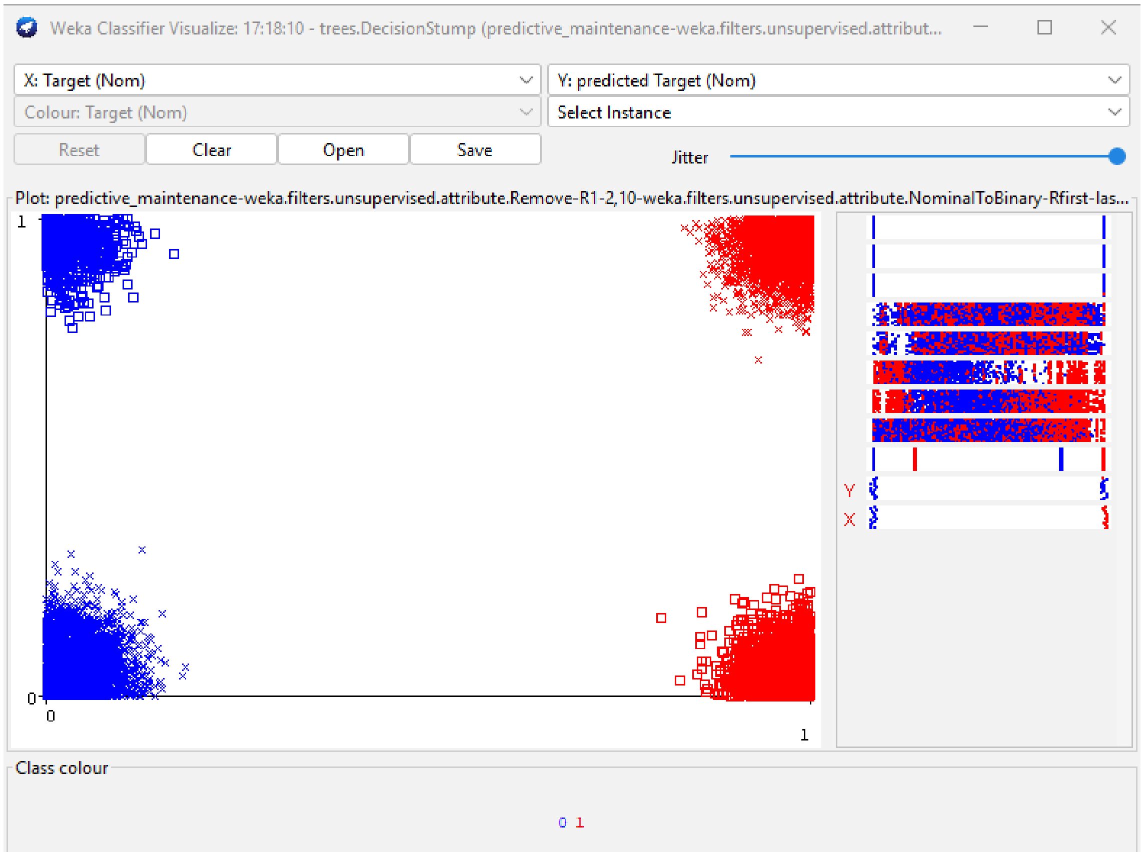
The confusion matrix:

	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.

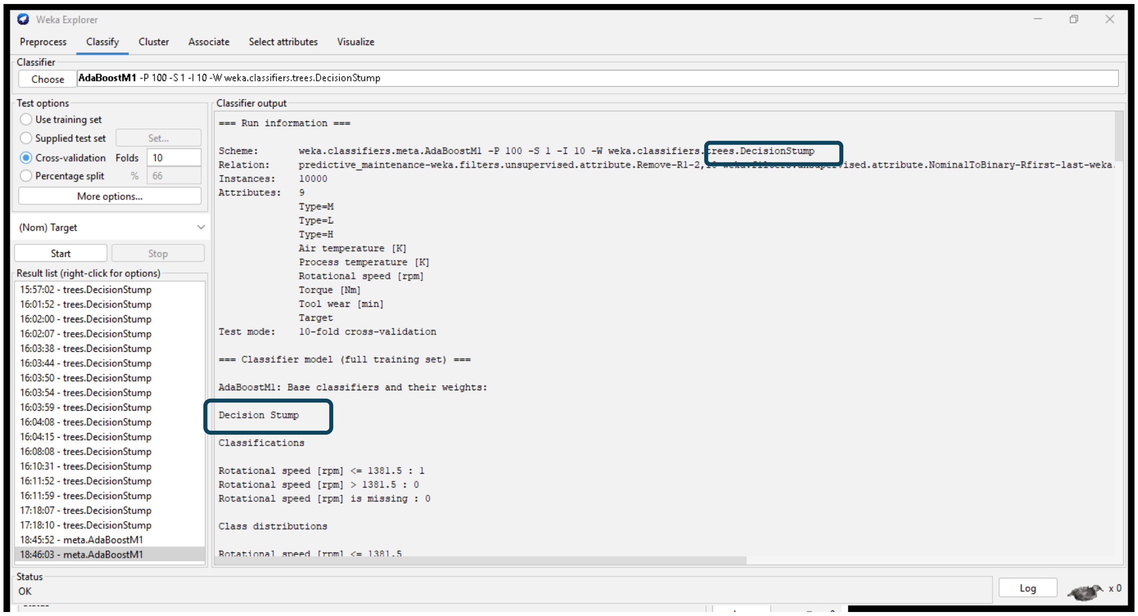
- **Visualize classifier Error:**



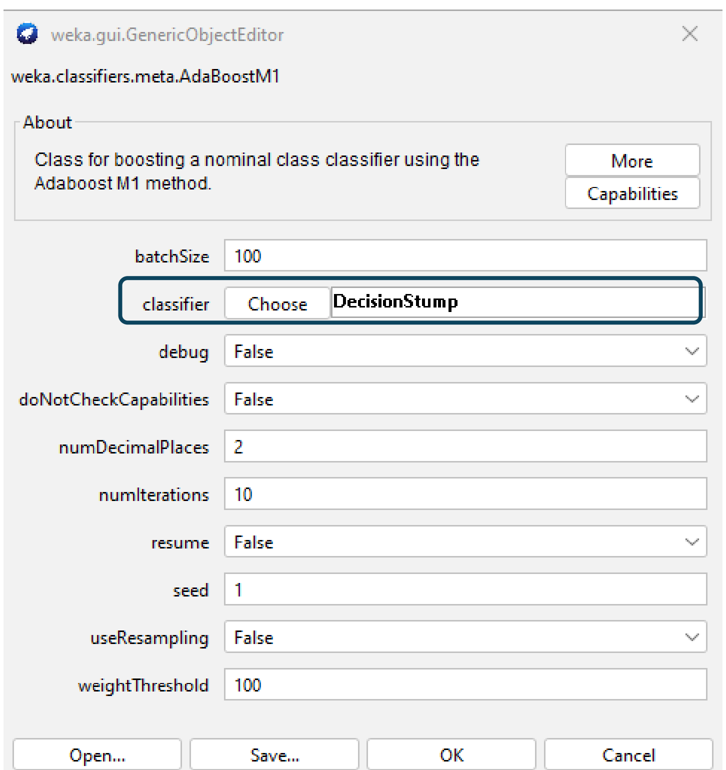
The **x**-axis represent the **true** values, while the **y**-axis represent the **predicted** values

6) AdaBoostM1:

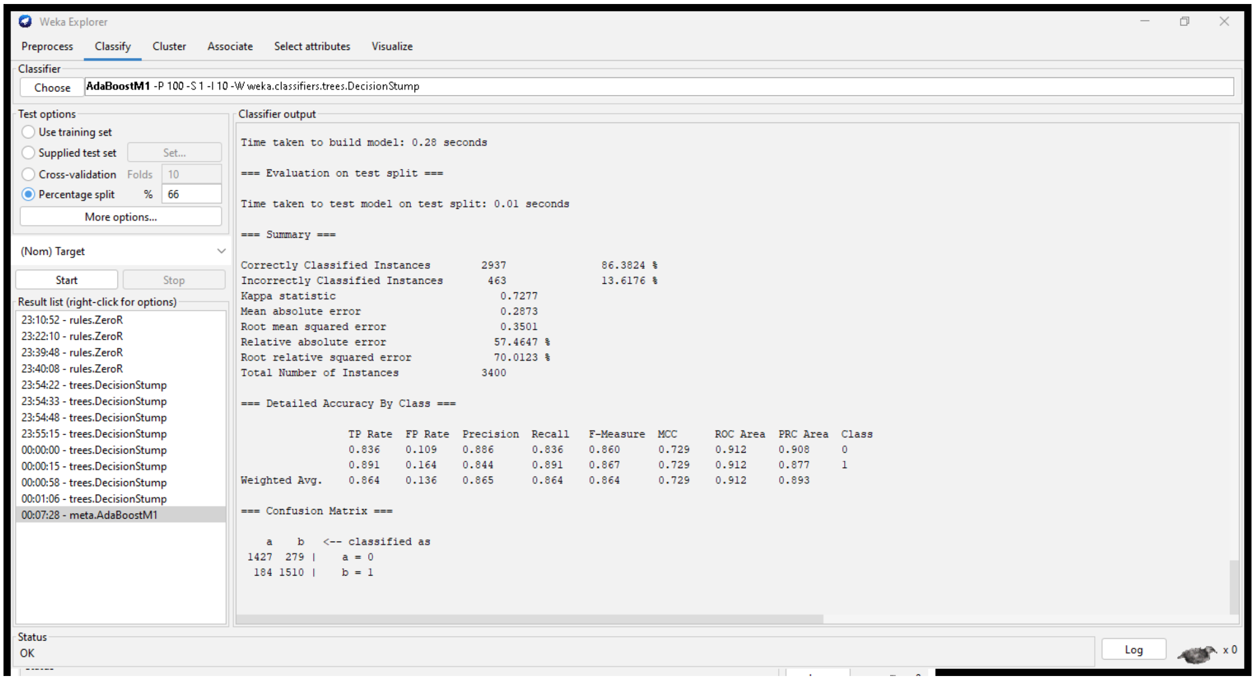
- The AdaBoostM1 combines multiple weak classifiers to create a strong classifier by adjusting the weights of training samples
- By default it use the **DecissionStump**



- **The default parameters**



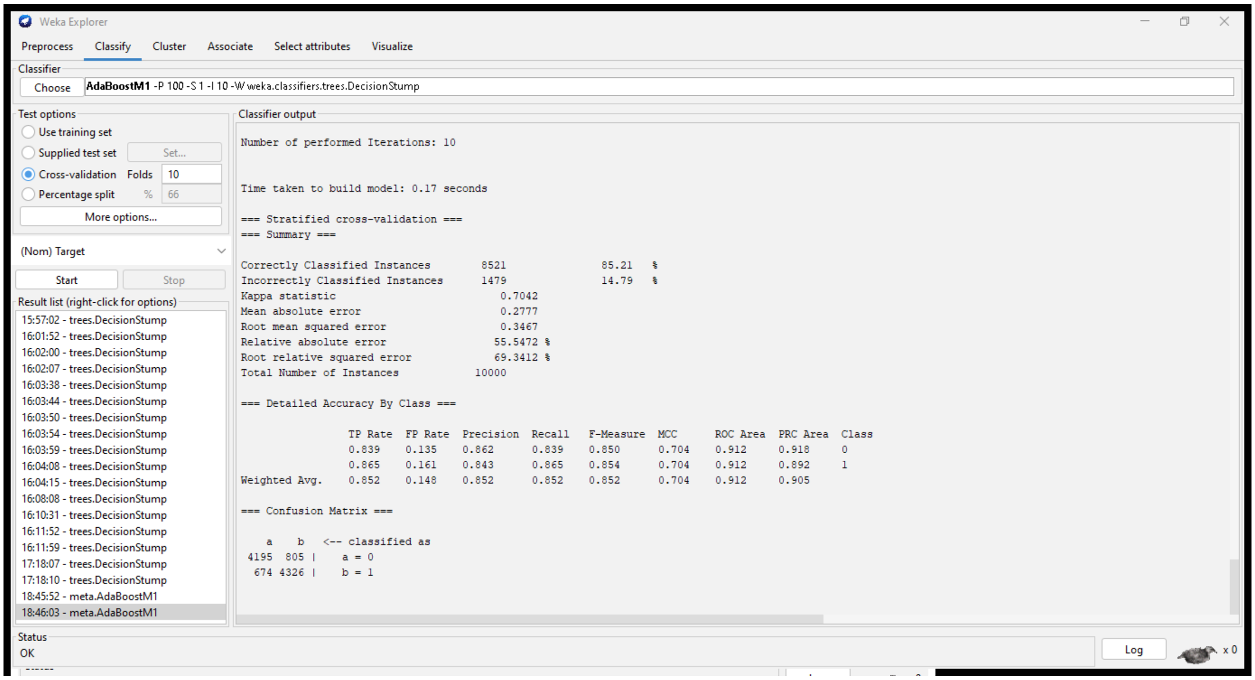
- **The result:**



The accuracy: 86.38%

The confusion matrix:

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

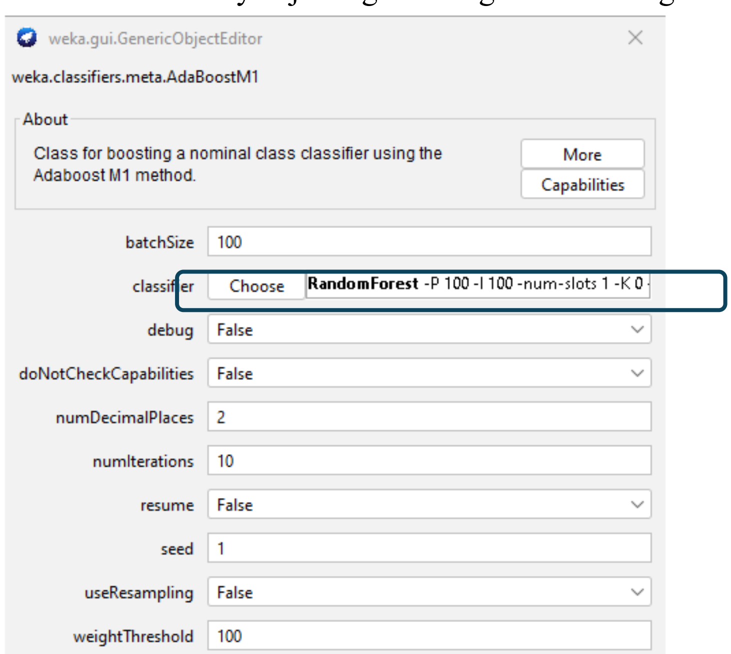


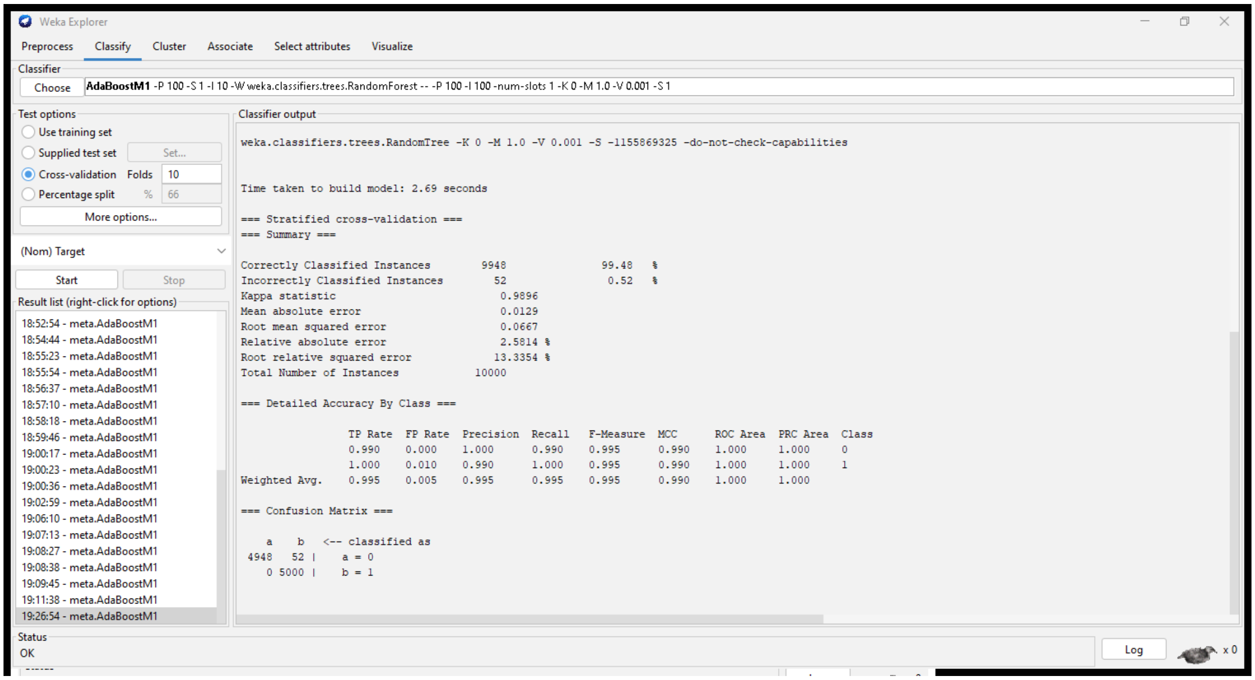
The confusion matrix:

	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.

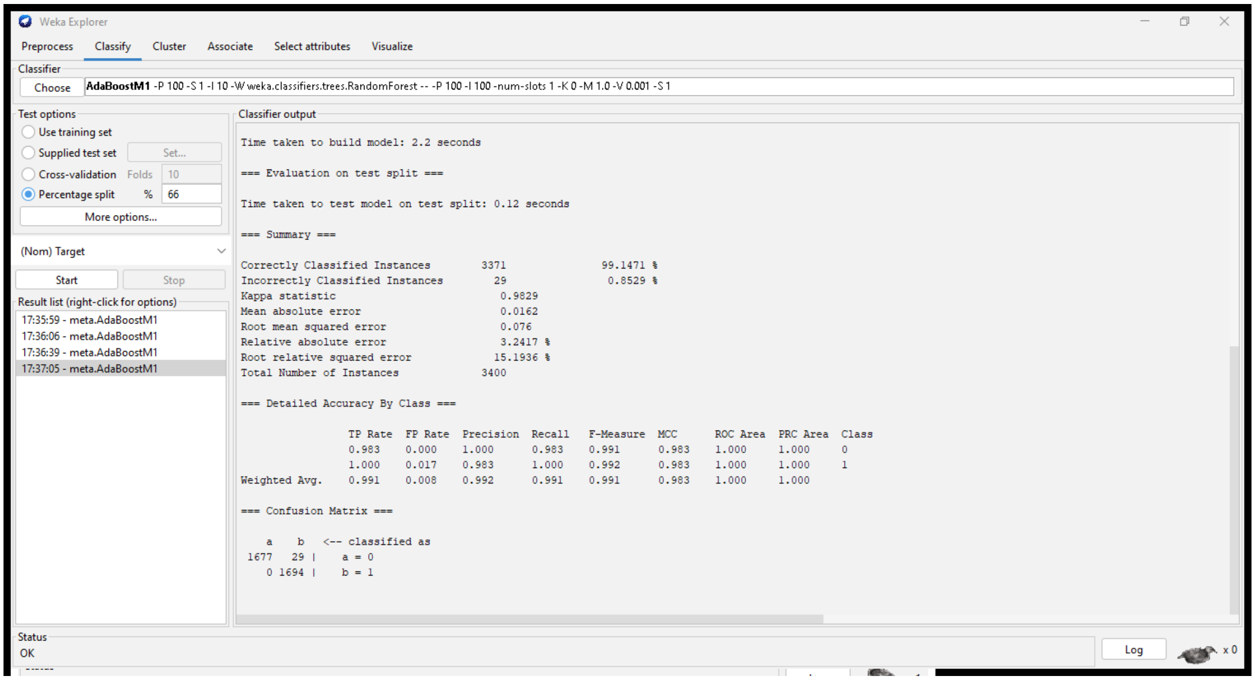




The accuracy: 99.48%

The confusion matrix:

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

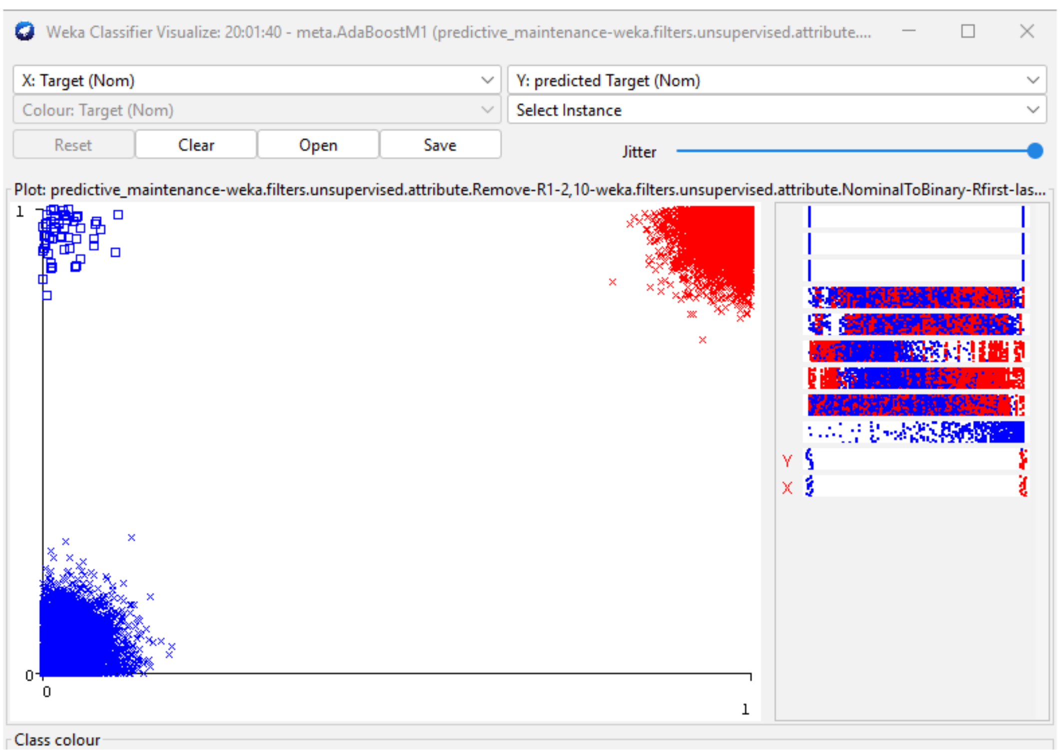


The accuracy: 99.14%

The confusion matrix:

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

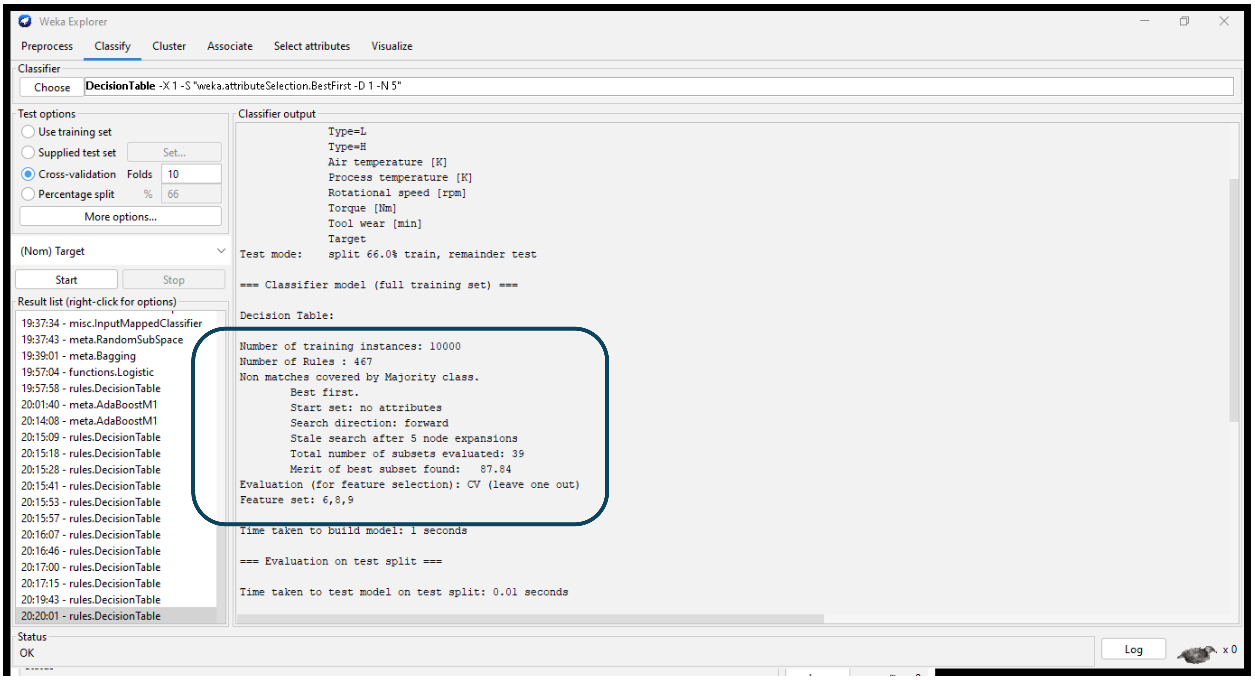
Visualize classifier Error:



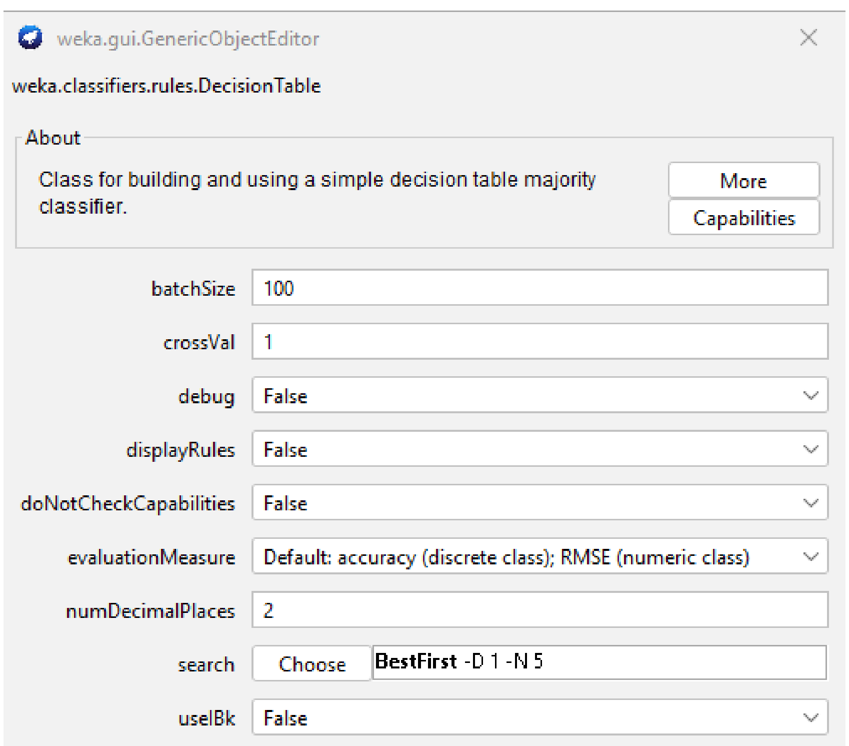
7) DecisionTable:

Class for building and using a simple decision table majority classifier

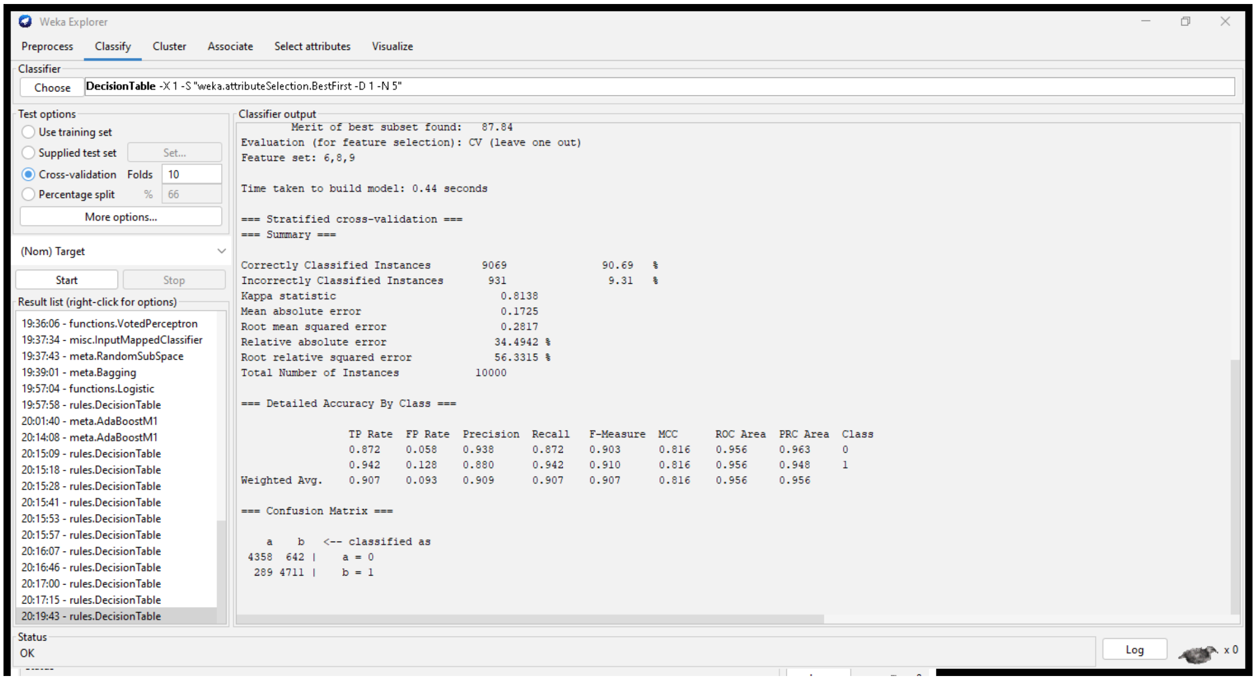
- Total **467 rules** were generated from the training data.
- If no rule matches, it assigns the **majority class**.
- Selected only **3 important features** (features 6, 8, and 9).



- **The default parameters:**



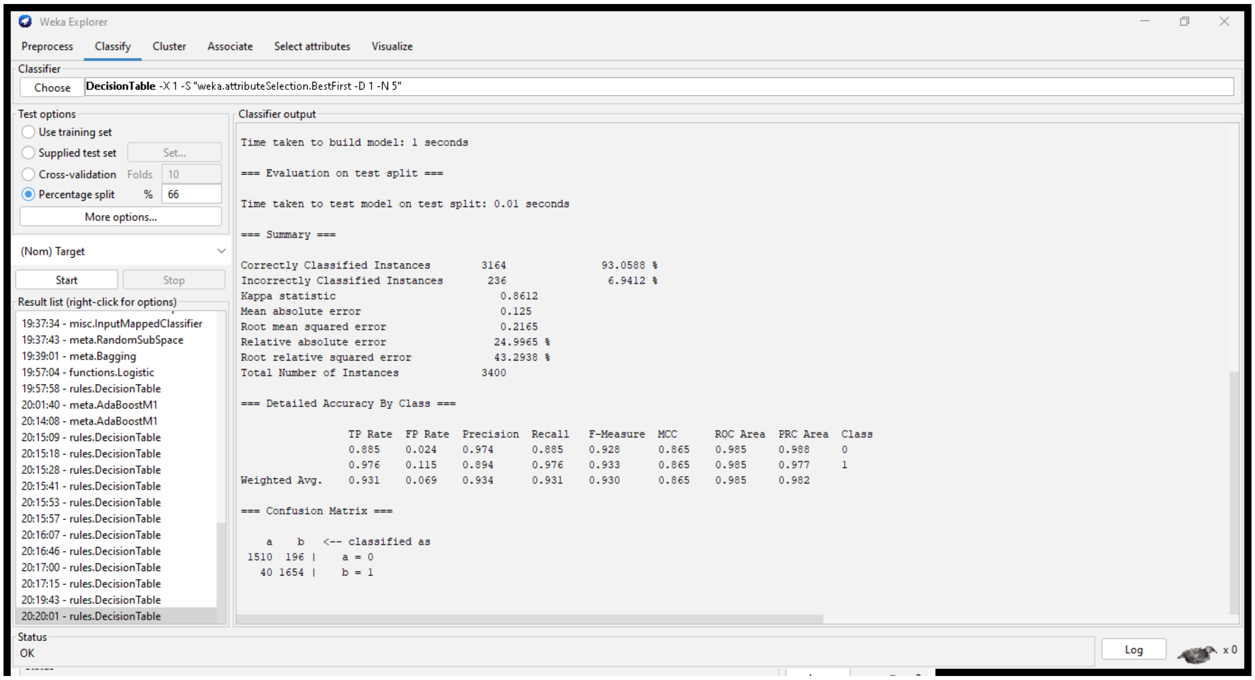
The result:



The accuracy: 90.69%

The confusion matrix:

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



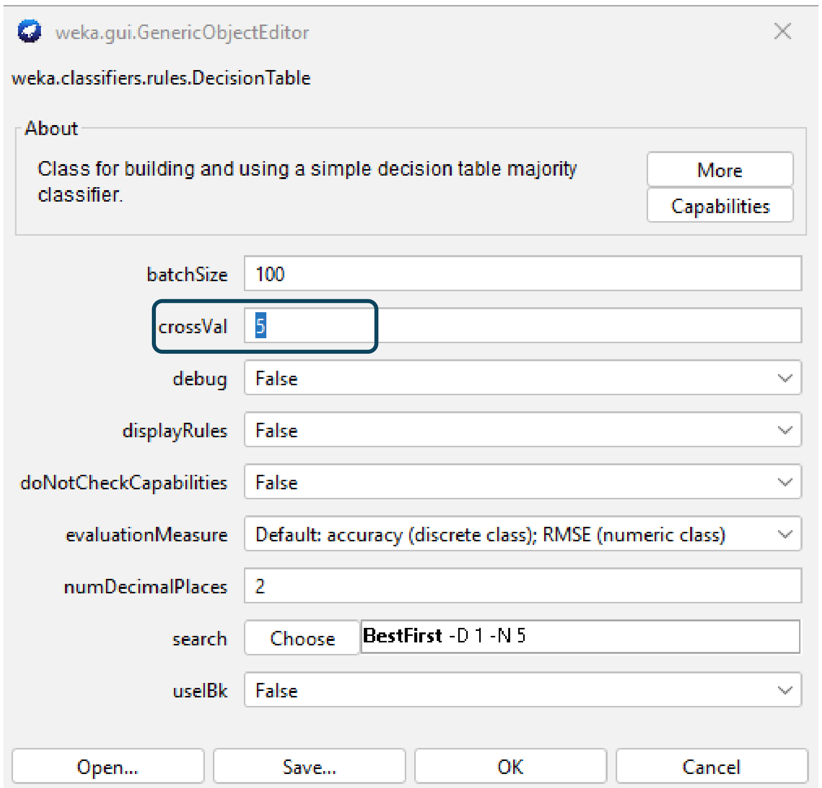
The accuracy: 93.05%

The confusion matrix:

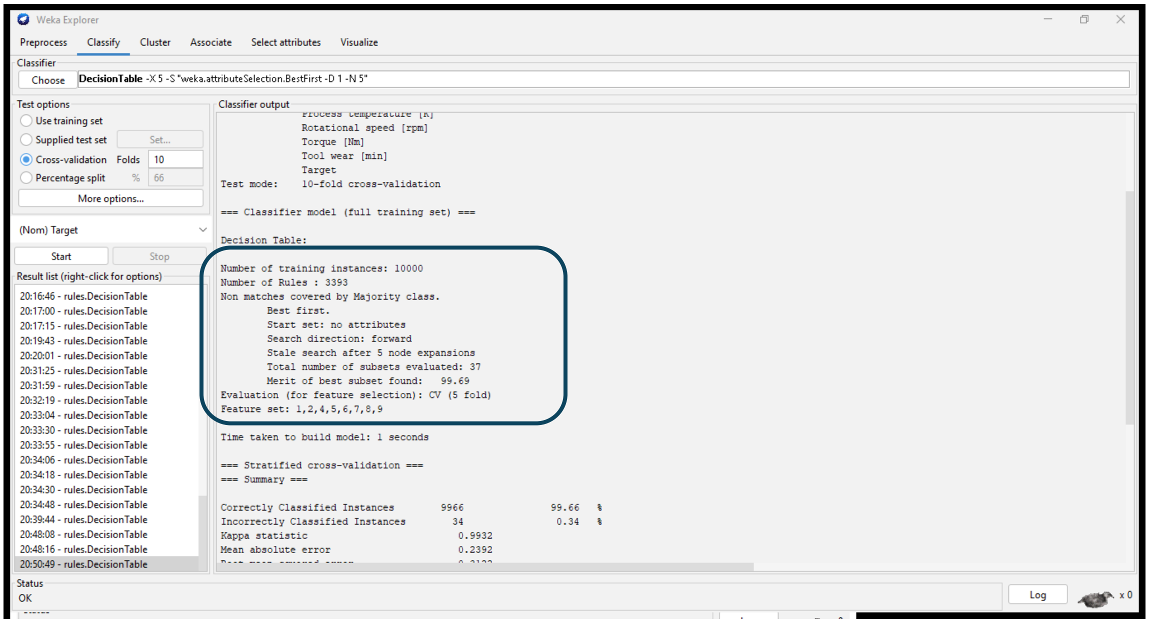
	Predicted 0	Predicted 1
Actual 0	**1510**	196
Actual 1	40	**1654**

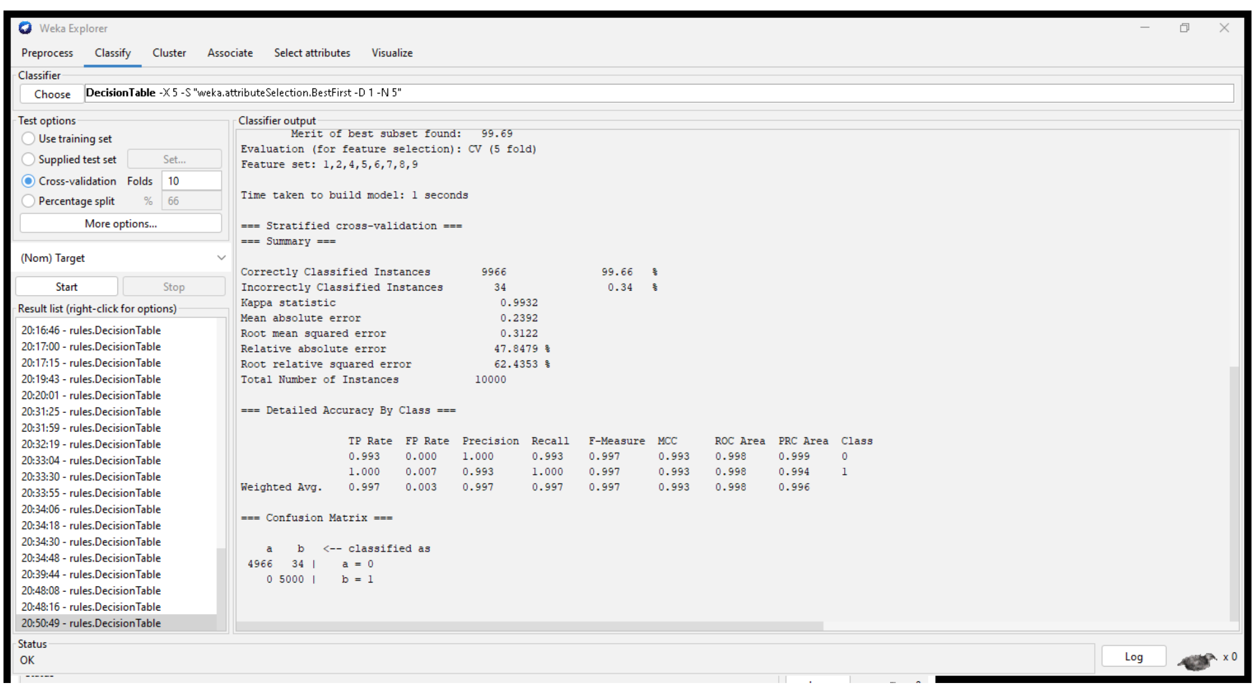
- Percentage spilt better than cross validation

- **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).

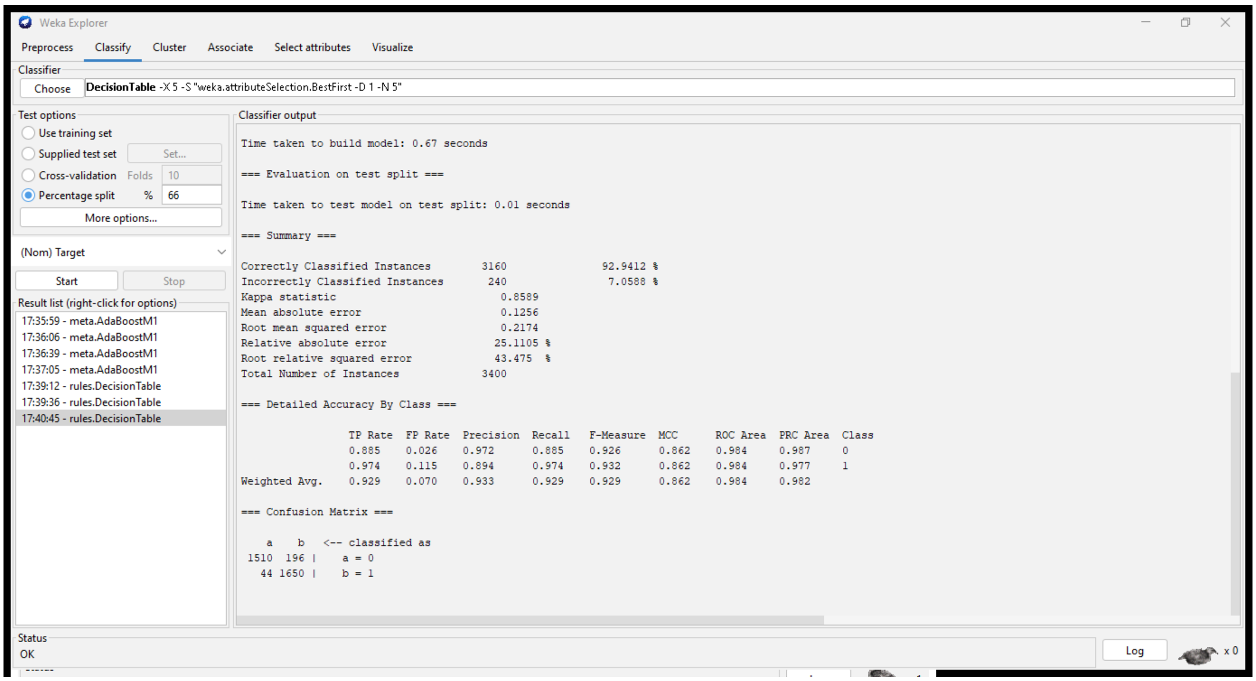




The accuracy: 99.66%

The confusion matrix:

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



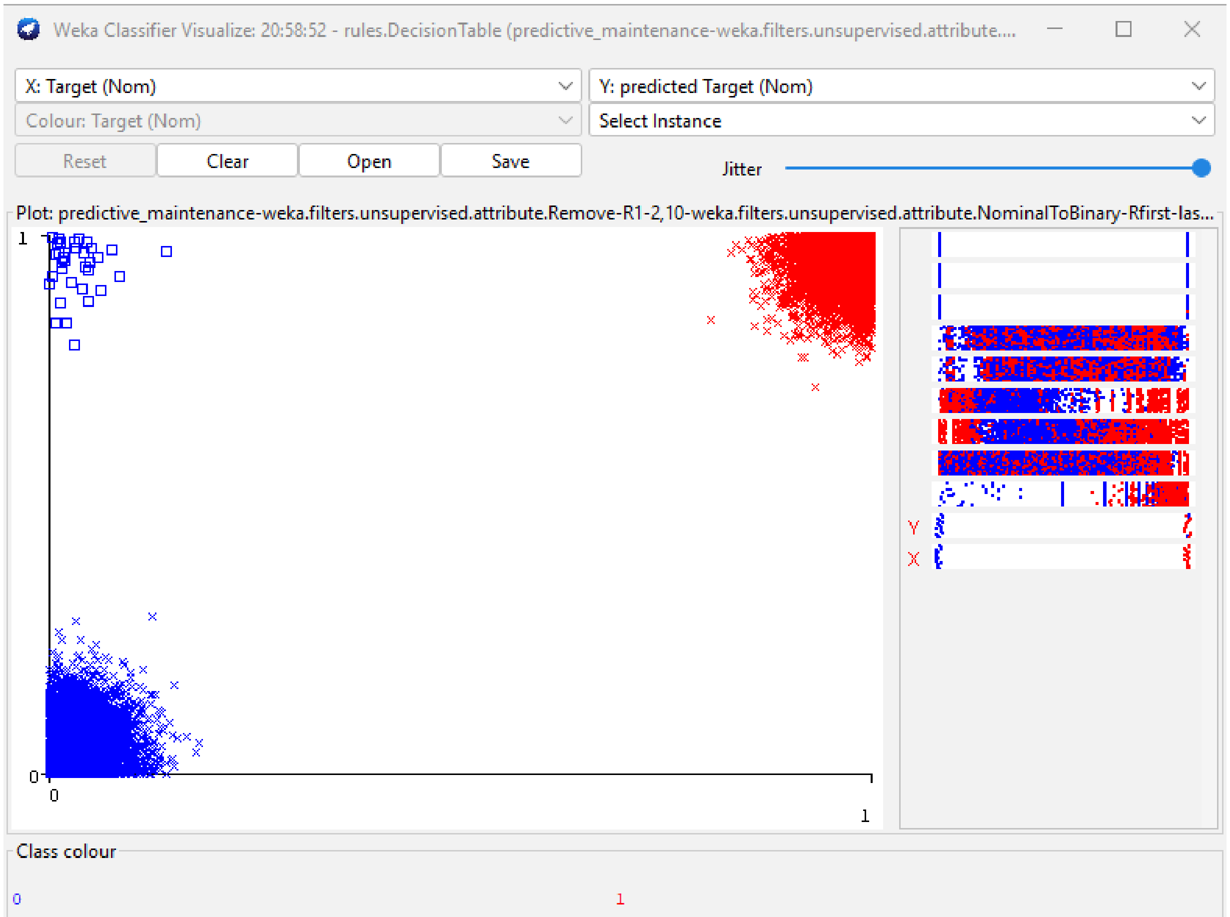
The accuracy: 92.94%

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%.**

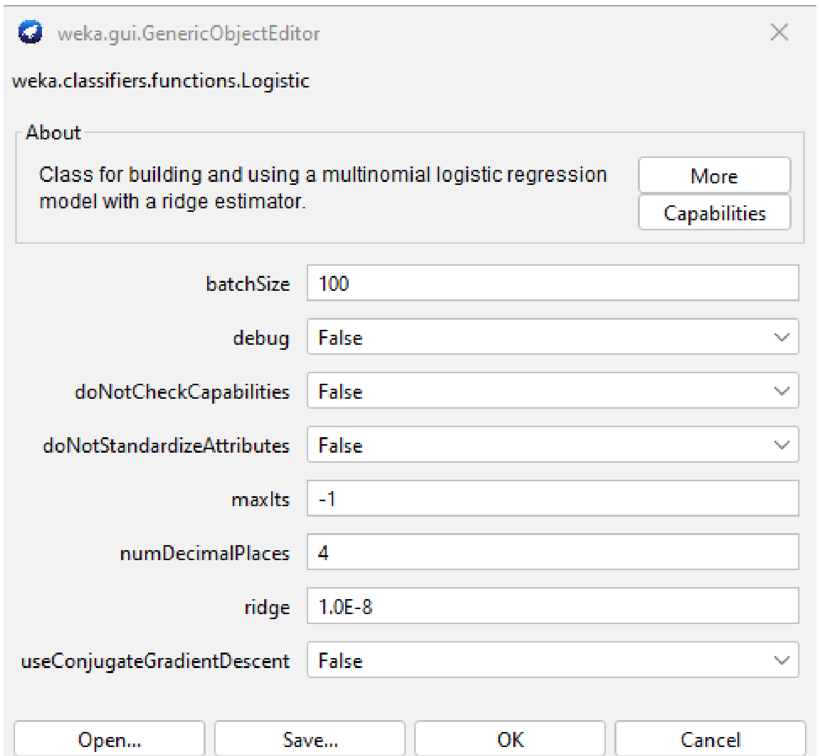
This indicates that the **DecisionTable** algorithm performed the **best** among the three.

8) Logistic Regression (LR)

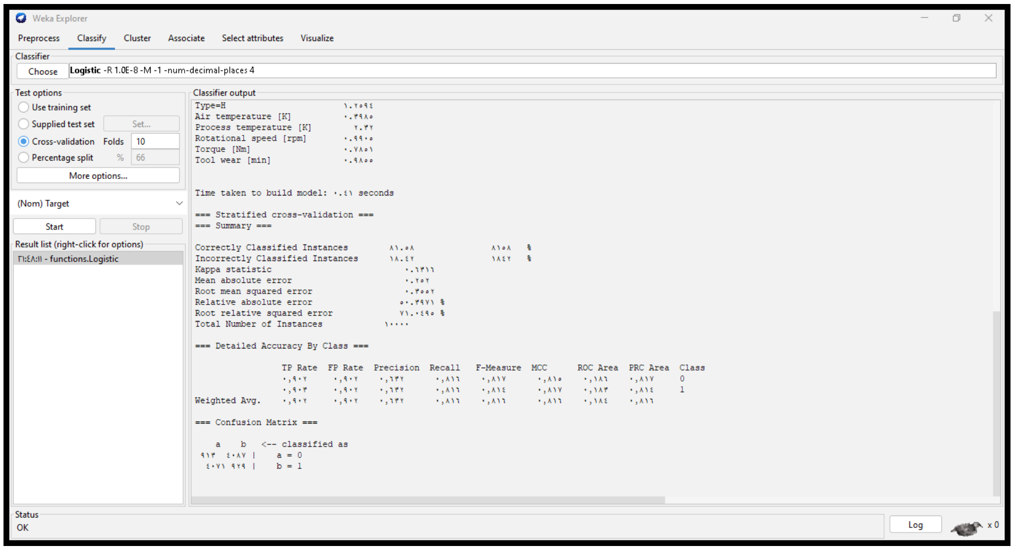
Logistic Regression is a classification method used to predict whether the outcome is 0 or 1 (such as success or failure). It aims to learn the relationship between the input features in order to estimate the probability that a given instance belongs to a particular class or not.

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:



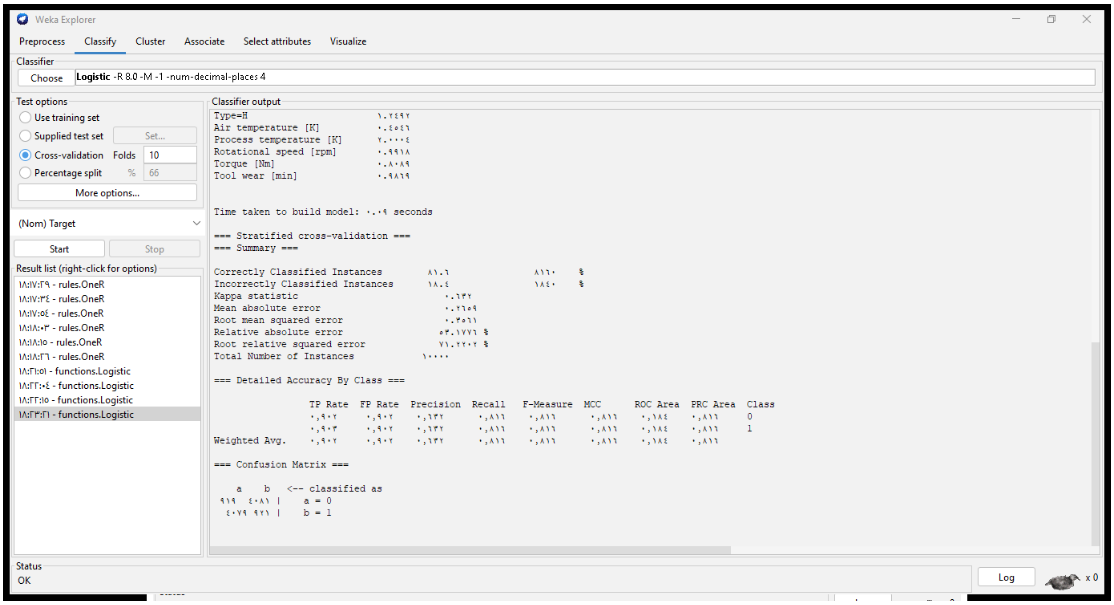
- The result



The accuracy: 81.58%

The confusion matrix:

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

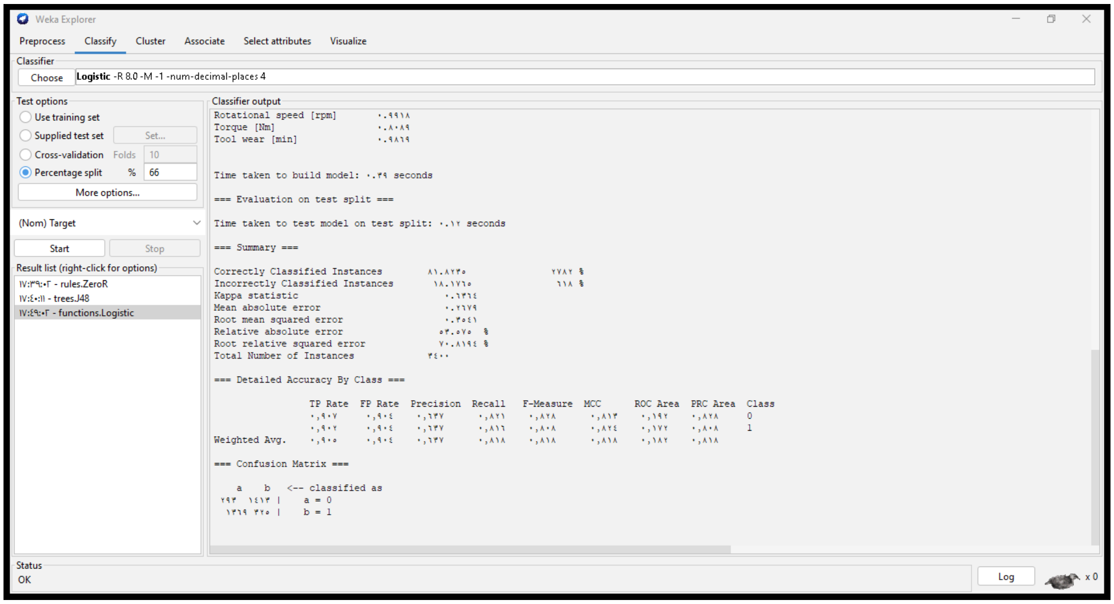


The accuracy : 81.6%

The confusion matrix:

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

- Percentage split



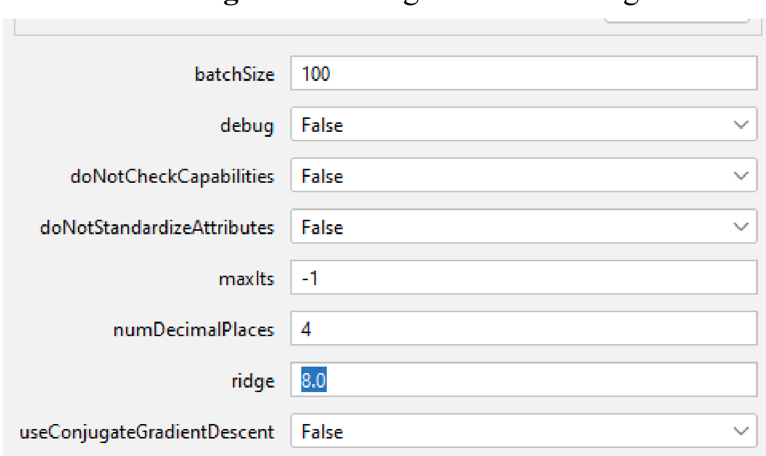
The accuracy : 81.8%

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

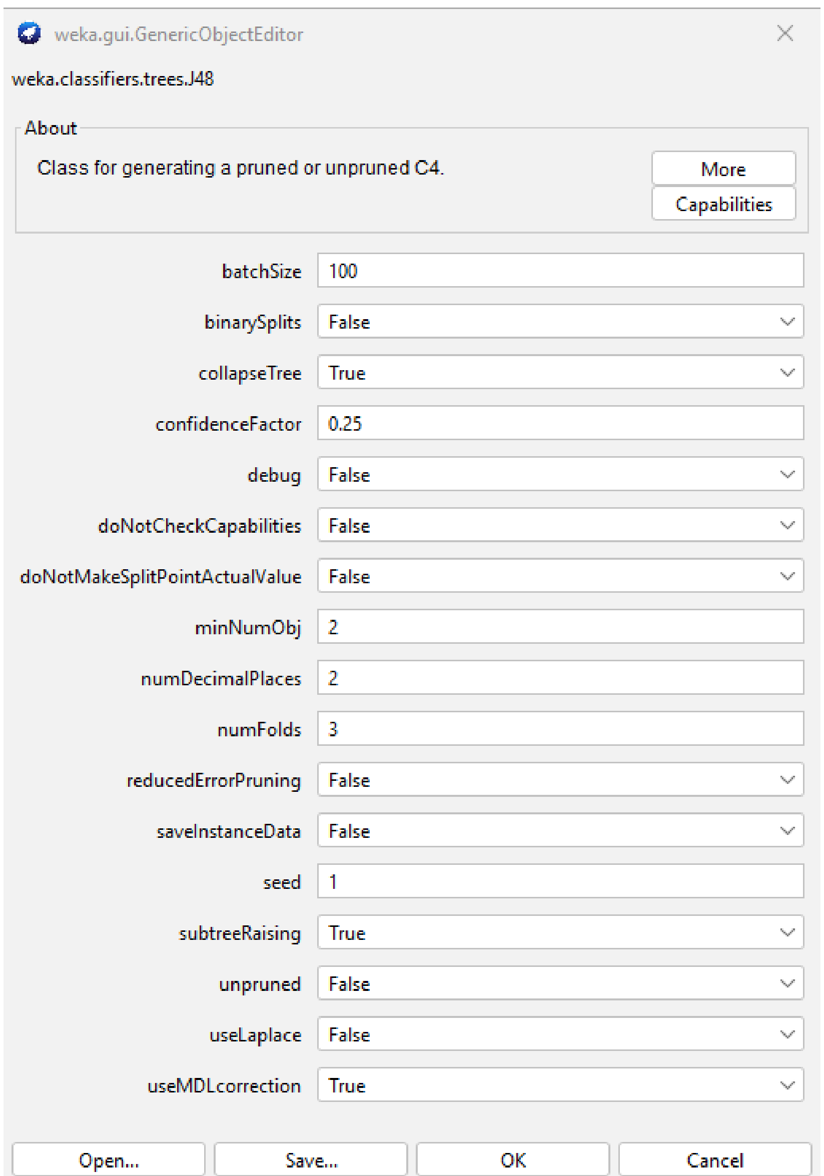
tree based on the attribute values of the dataset.

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

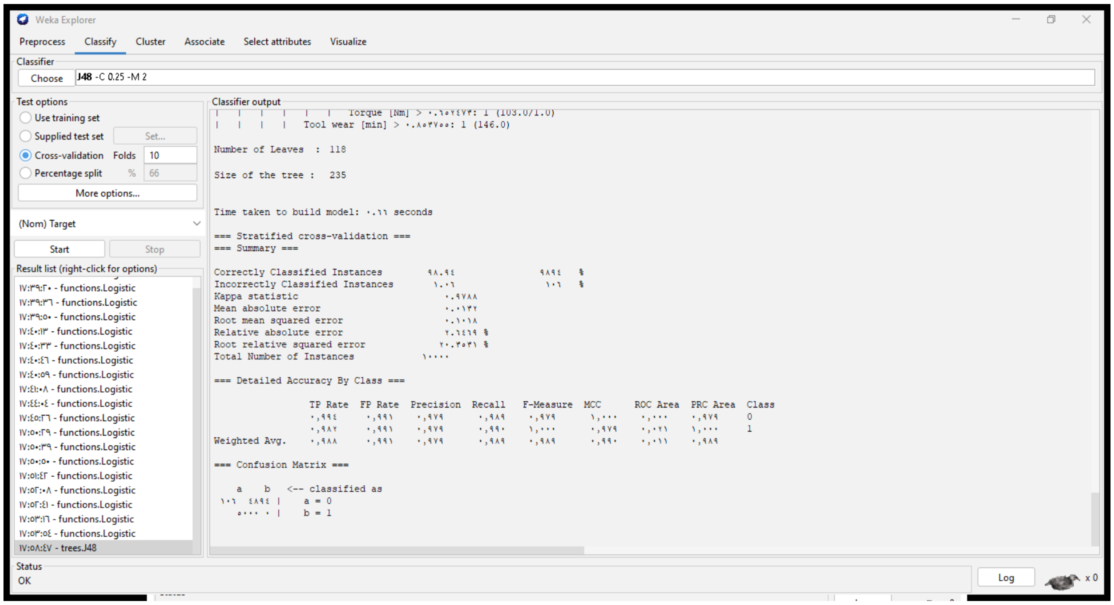
It supports pruning techniques to reduce overfitting and provides options for parameter tuning to optimize the model's accuracy.

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

- The default parametets:



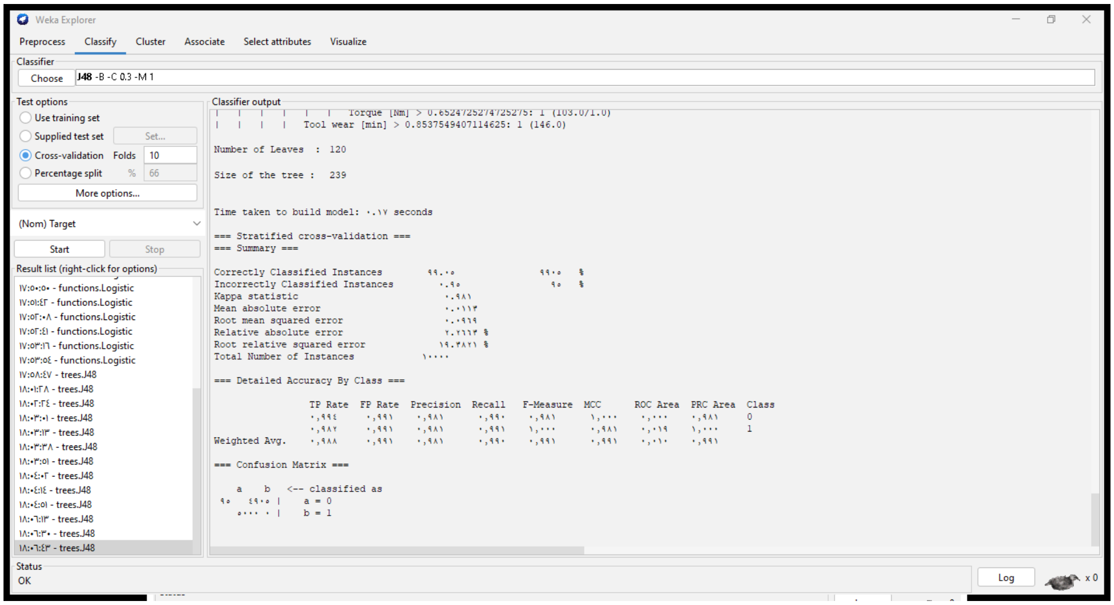
- The result



The accuracy: 98.94%

The confusion matrix:

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

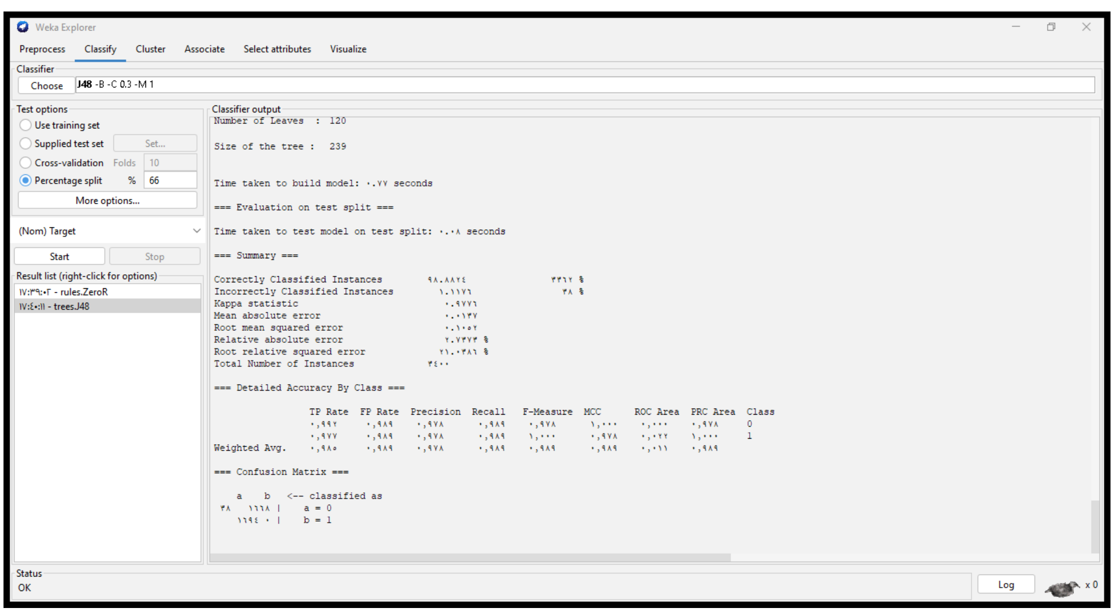


The accuracy : 99.5%

The confusion matrix:

	Predicted 0	Predicted 1
Actual 0	4905	95
Actual 1	0	5000

- Percentage split

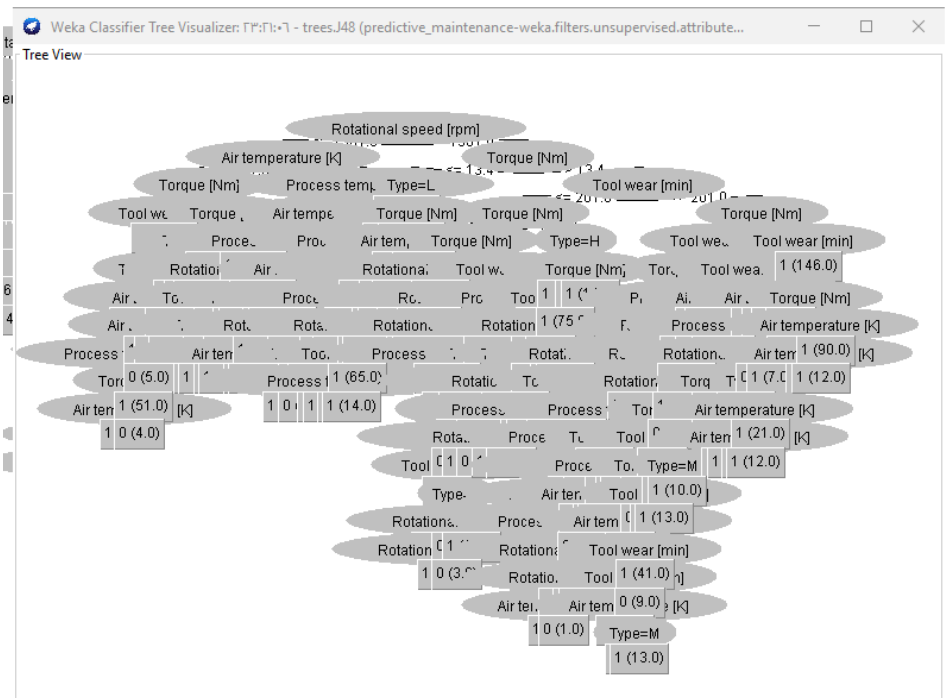


The accuracy : 98.88%

The confusion matrix:

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

- **Tree J48**



The algorithm with different parameters:

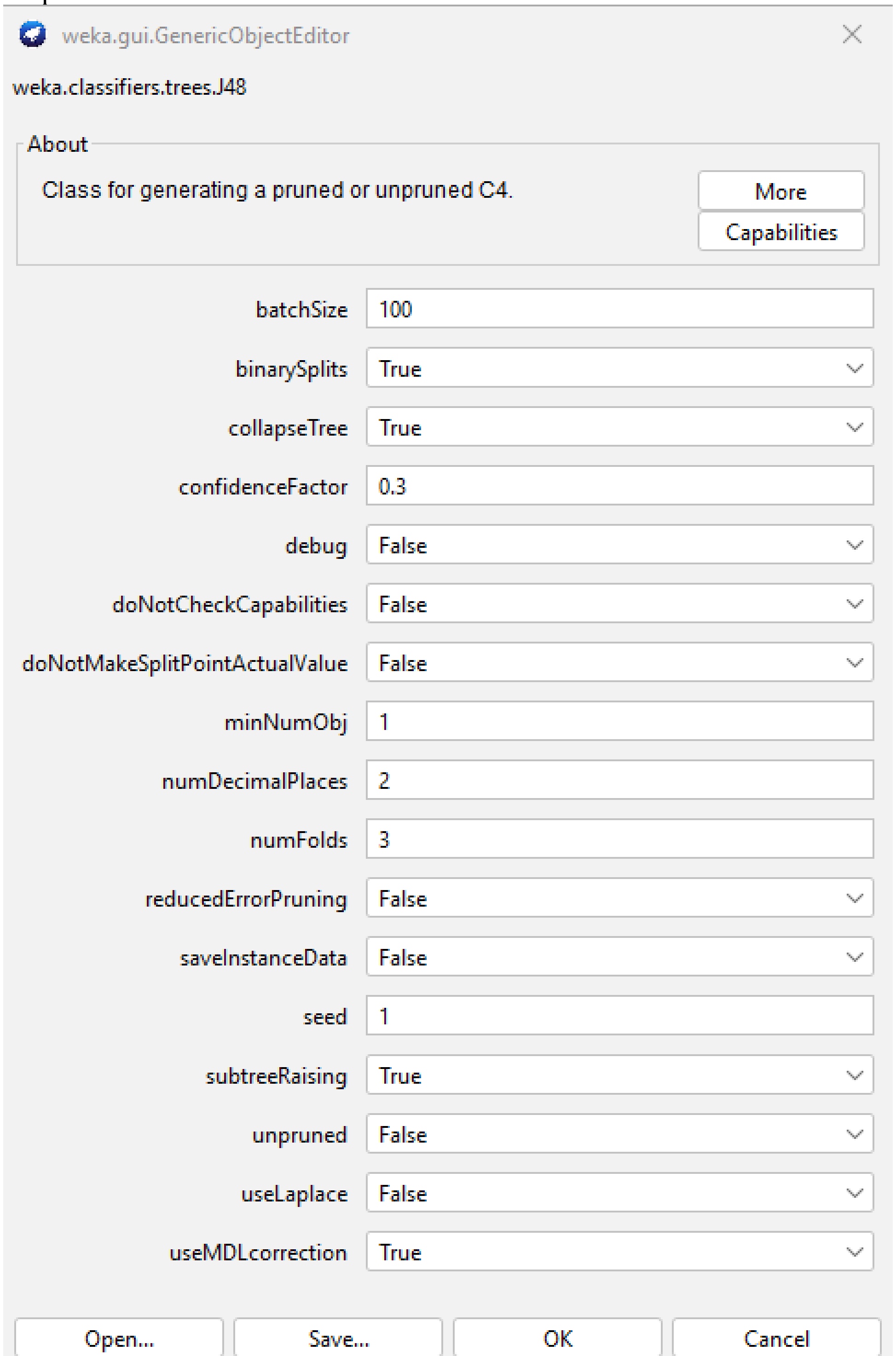
When I changed the binarySplits parameter (from false to true), the confidenceFactor (from 0.25 to 0.3), and the minNumObj (from 2 to 1), the algorithm improved performance."

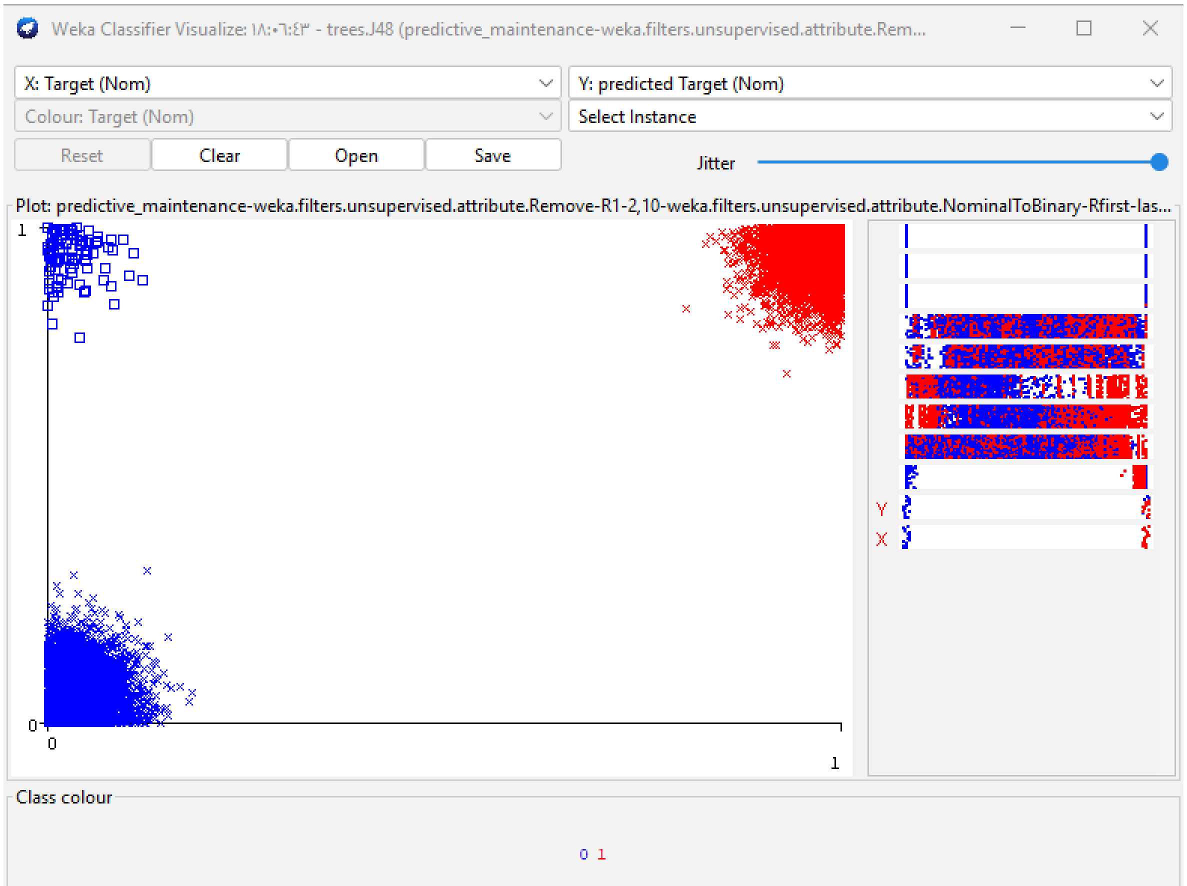
• **binarySplits**: Whether to use binary splits on nominal attributes when building the trees.

• **confidenceFactor**: The confidence factor used for pruning (smaller values incur more pruning).

• **minNumObj**: The minimum number of instances per leaf.

The parameter



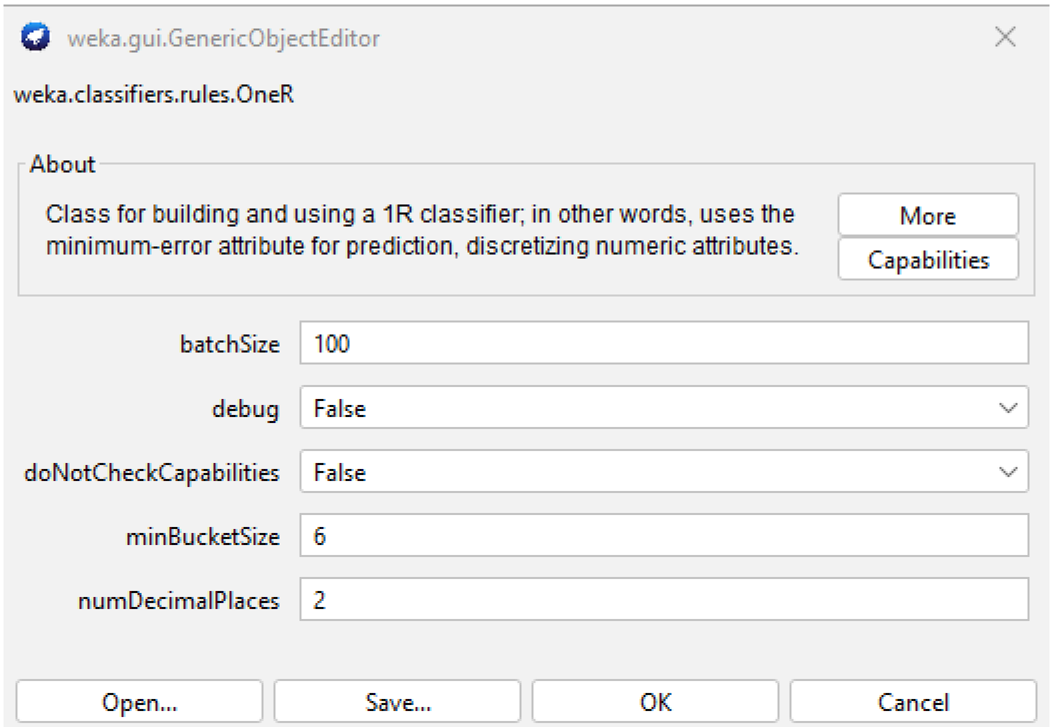

Weka Classifier Visualize: ١٨:٠٦:٤٣ - trees.J48 (predictive_maintenance-weka.filters.unsupervised.attribute.Rem...
X: Target (Nom)
Y: predicted Target (Nom)
Colour: Target (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: predictive_maintenance-weka.filters.unsupervised.attribute.Remove-R1-2,10-weka.filters.unsupervised.attribute.NominalToBinary-Rfirst-las...
1
0
0
1
Y
X
Class colour
0 1

10) OneR:

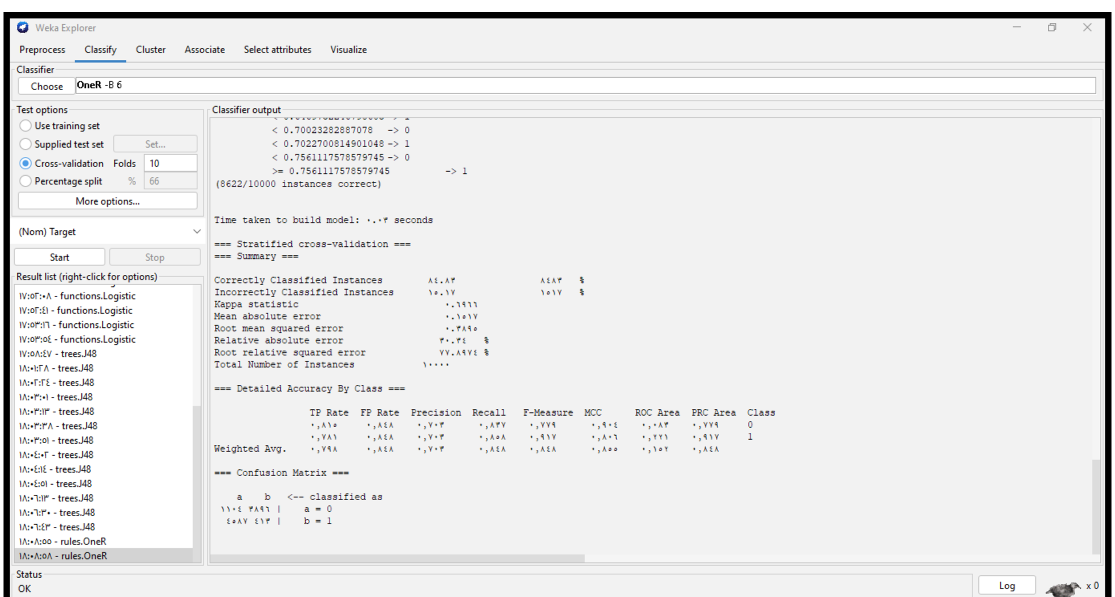
OneR is a simple classification algorithm that learns rules from a single attribute. It is characterized by its simplicity and ease of interpretation. The algorithm examines all attributes and selects the one that results in the lowest classification error.

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

- The default parameters:



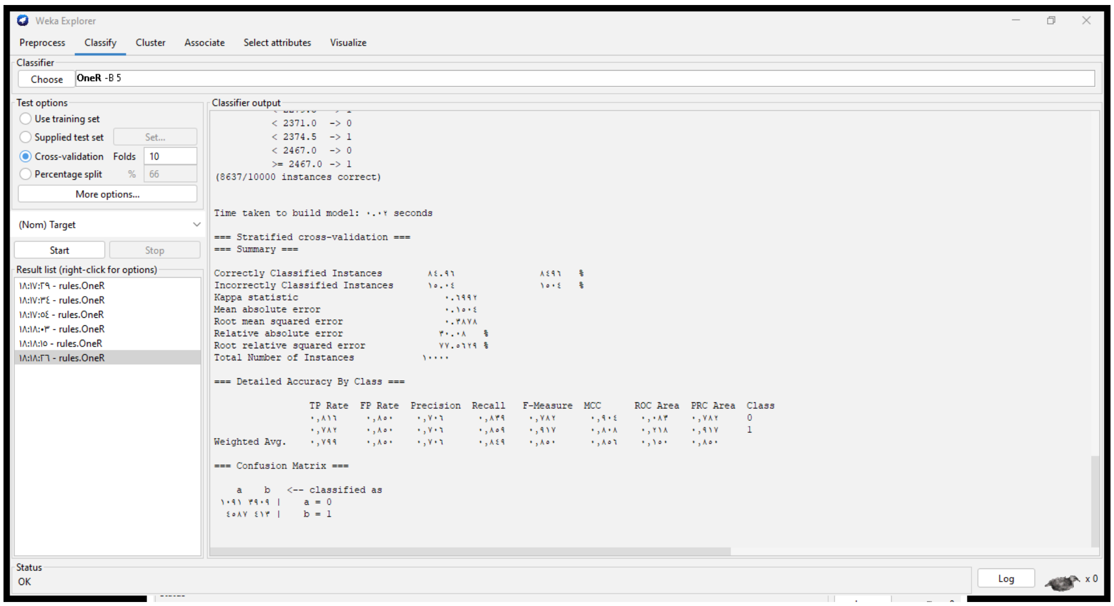
- The result



The accuracy : 84.83%

The confusion matrix:

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

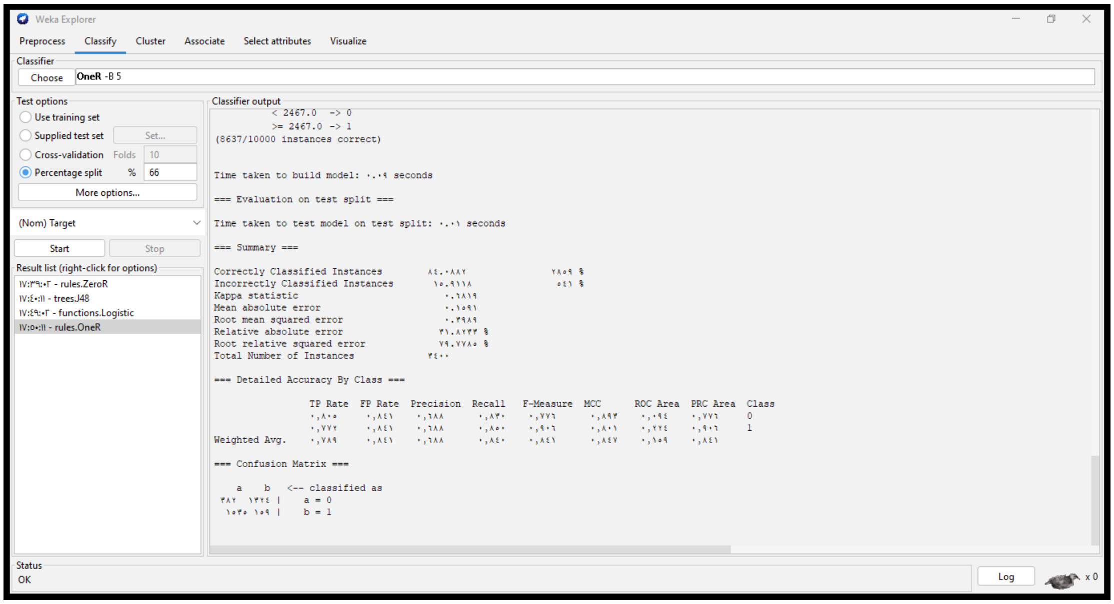


The accuracy : 85.96%

The confusion matrix:

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

- Percentage split



The accuracy : 84.08%

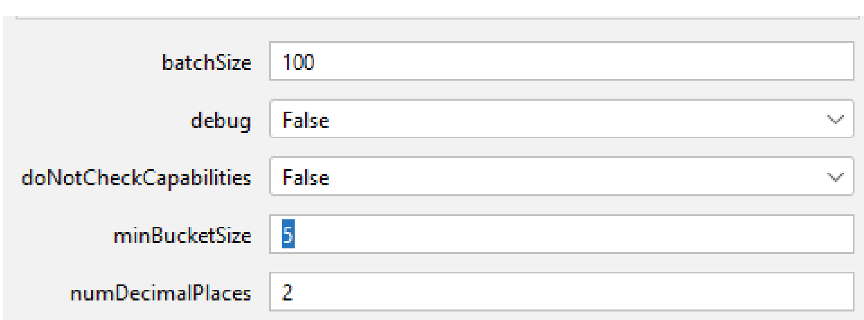
The confusion matrix:

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

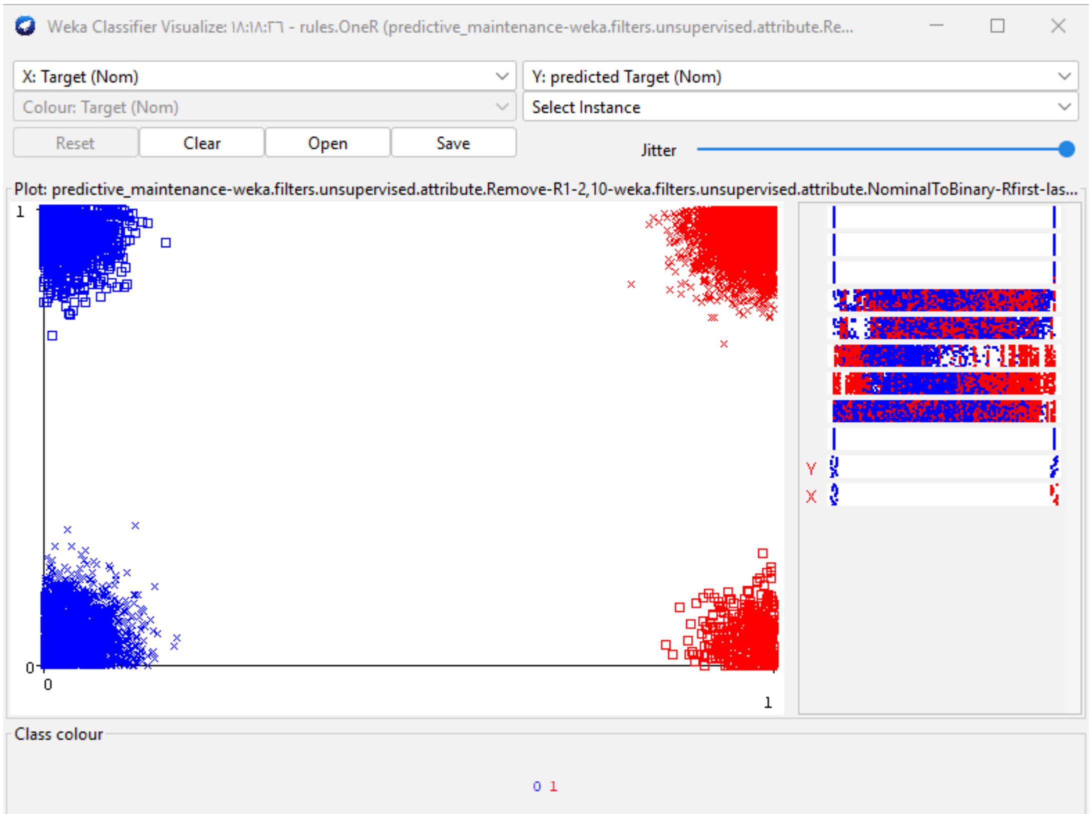
The algorithm with different parameters:

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.



- 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

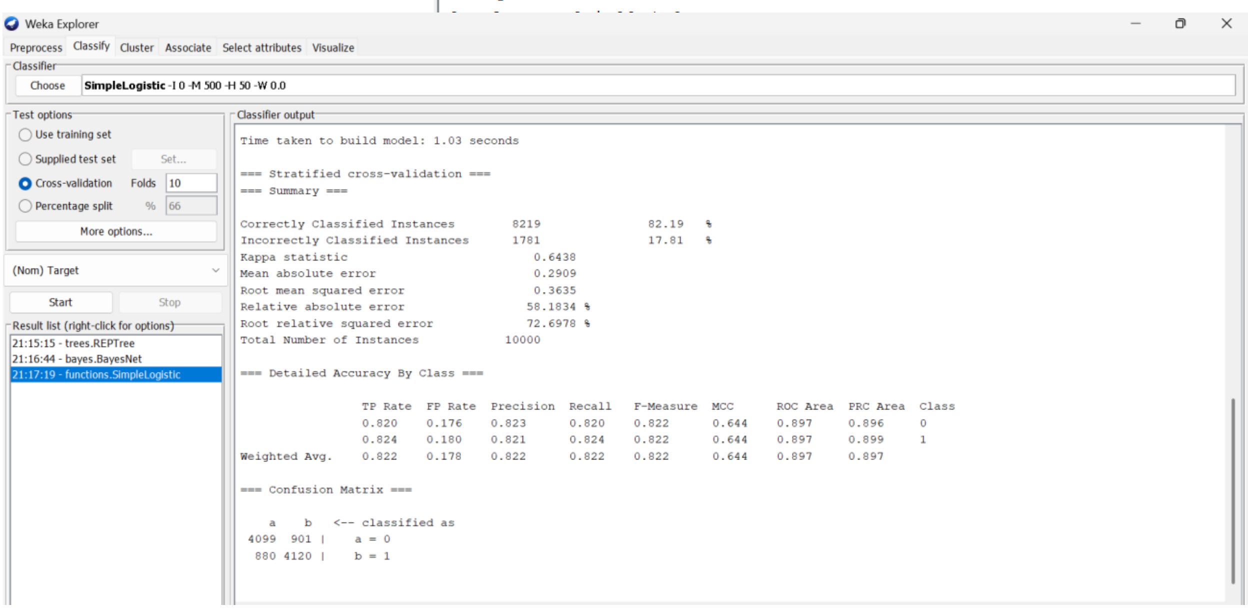
Focus on our data: Likely emphasizes Process temperature, Torque, and Tool wear

Most important features:

- Air temperature
- Torque
- Type L

```
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 +
```

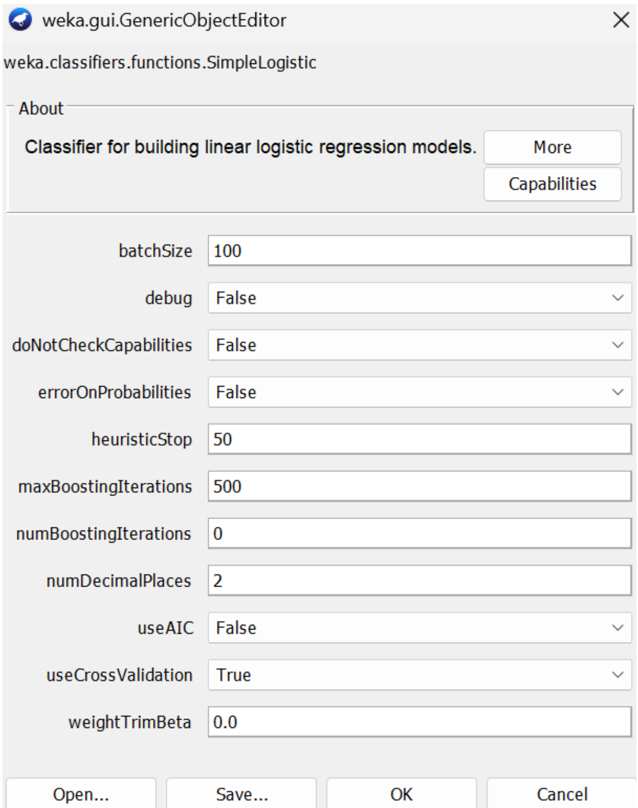
- The result:



The confusion matrix:

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

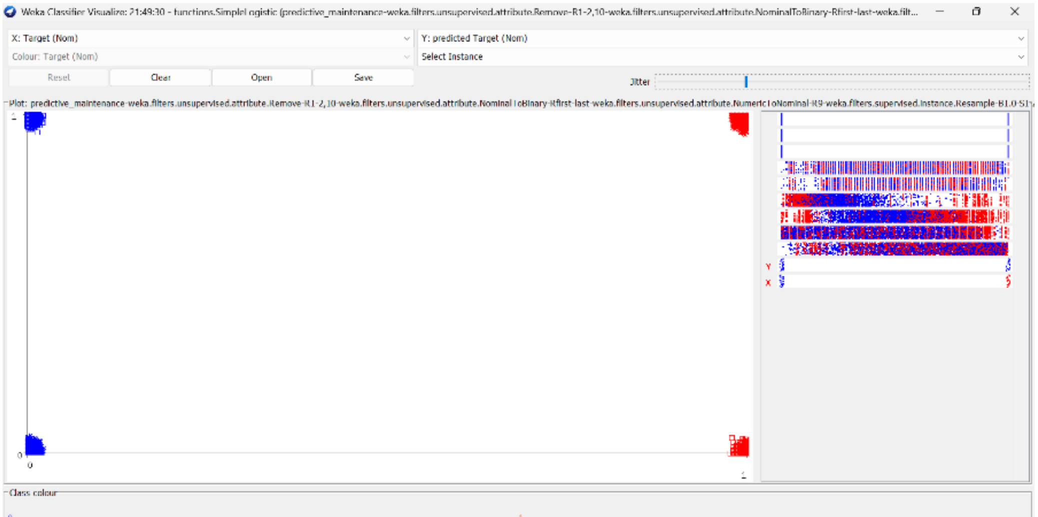
- **Default parameter**



- **Different parameters**

When trying to change the heuristicsStop 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

Tho 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
```

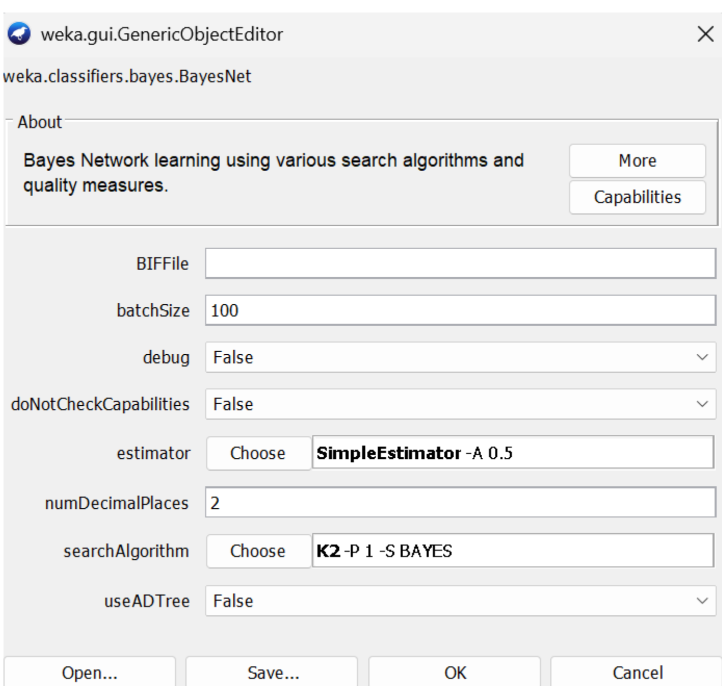
- The result:

```
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

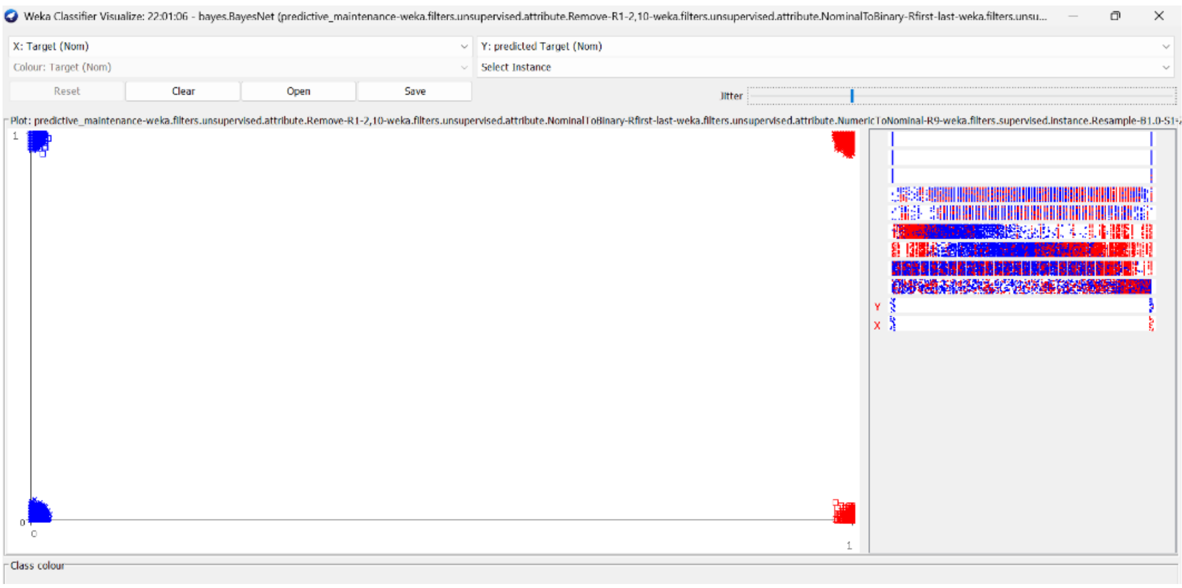
-Default parameters



-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.

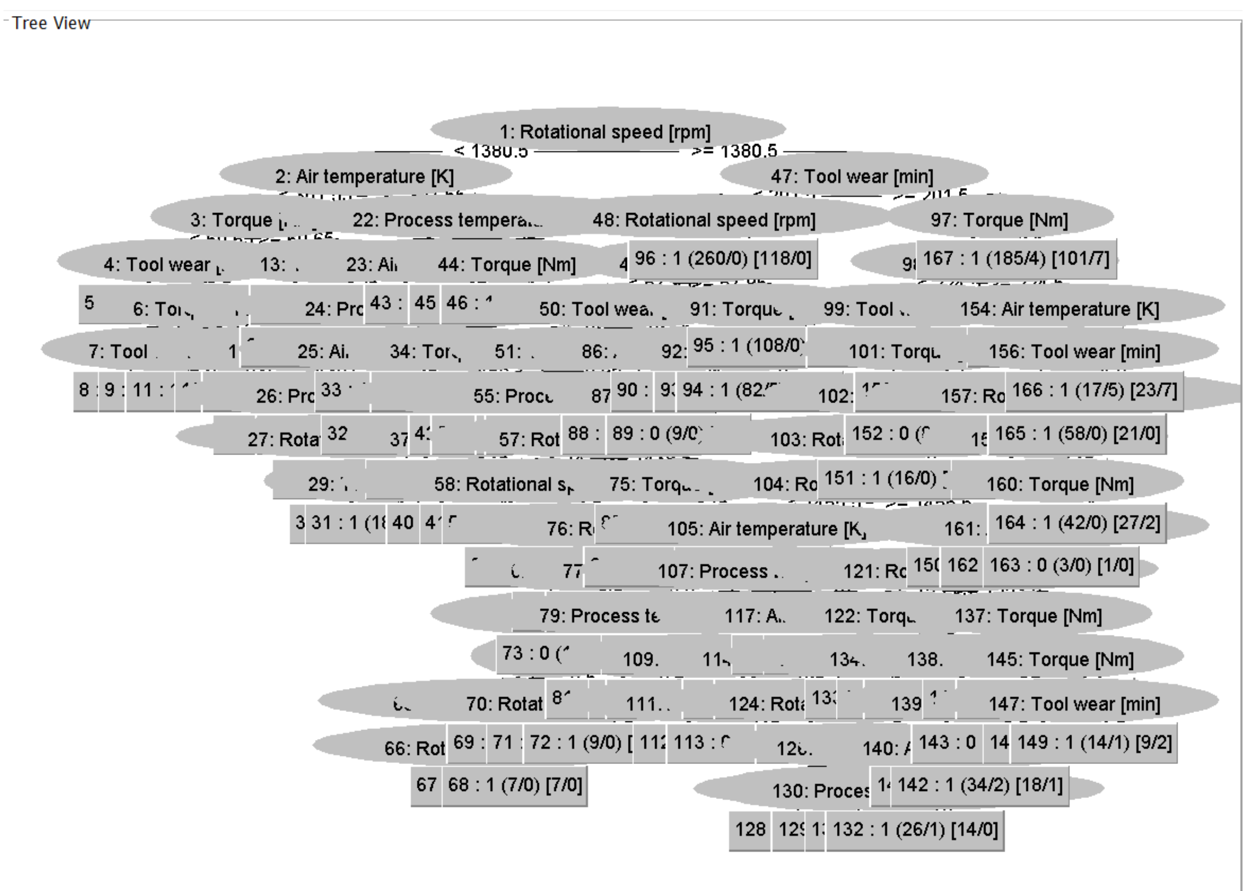
Focus on our data: Features used heavily:

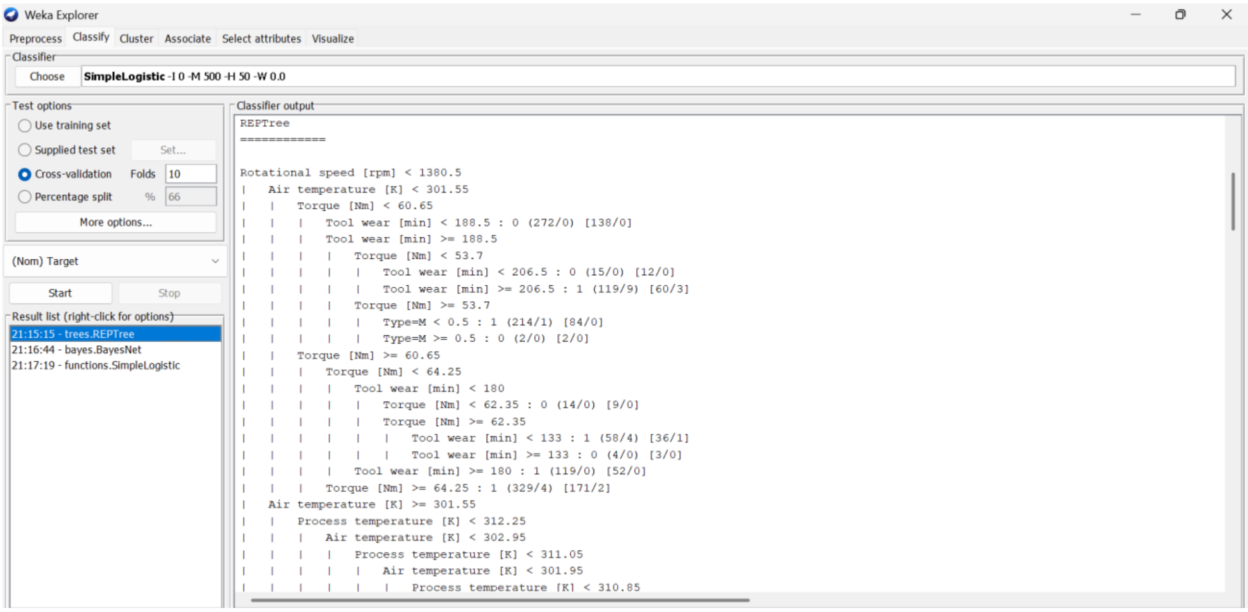
• Rotational speed and Tool wear are top decision nodes.

• 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.





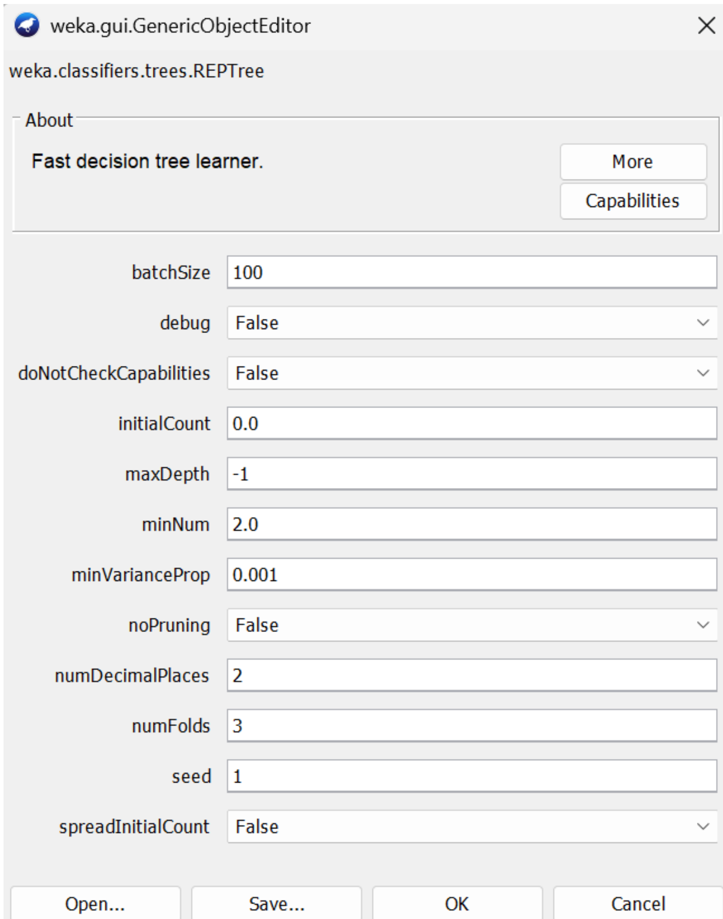
- The result:

```
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

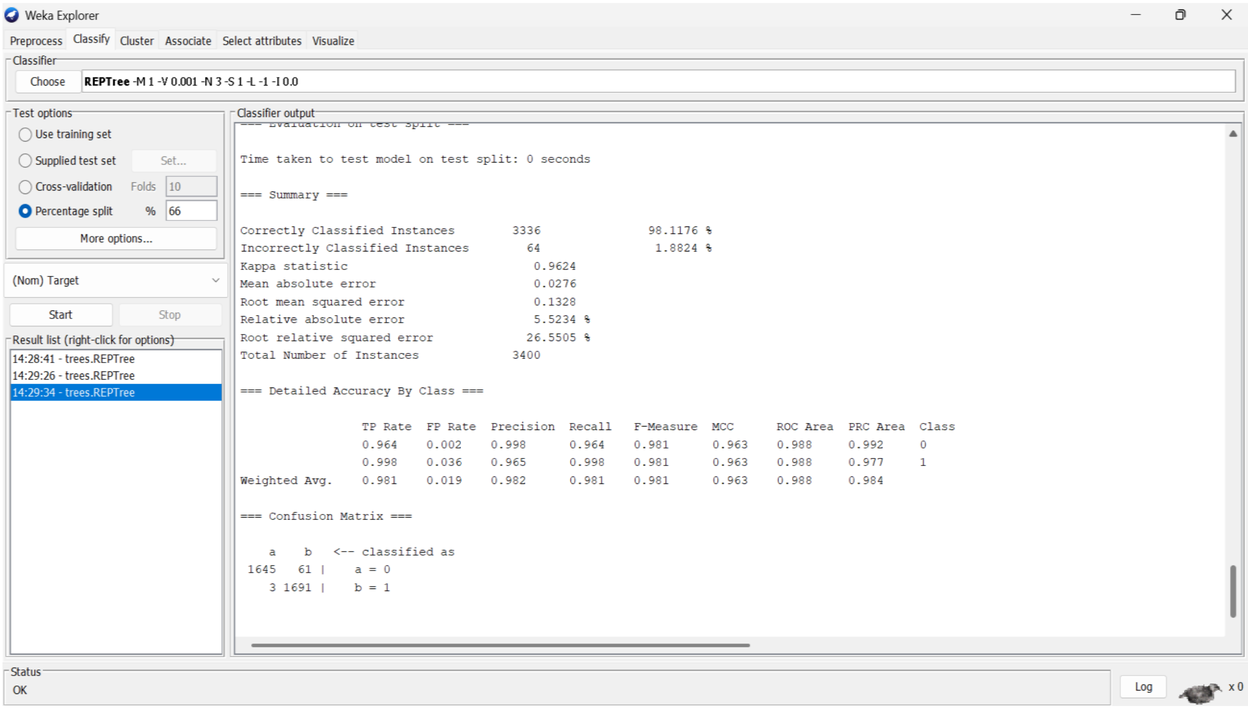


-Different parameter

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



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.

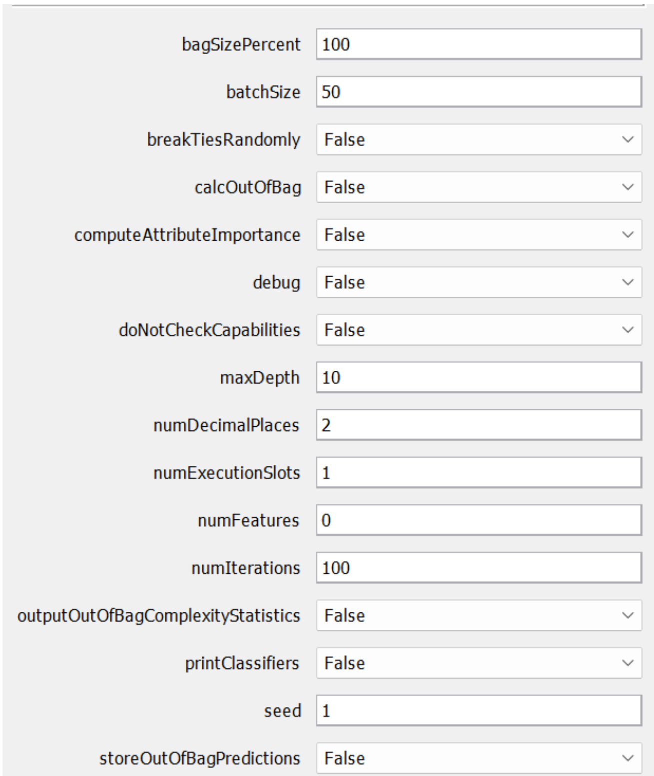
```
Classifier output
Time taken to build model: 1.27 seconds

=== 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**

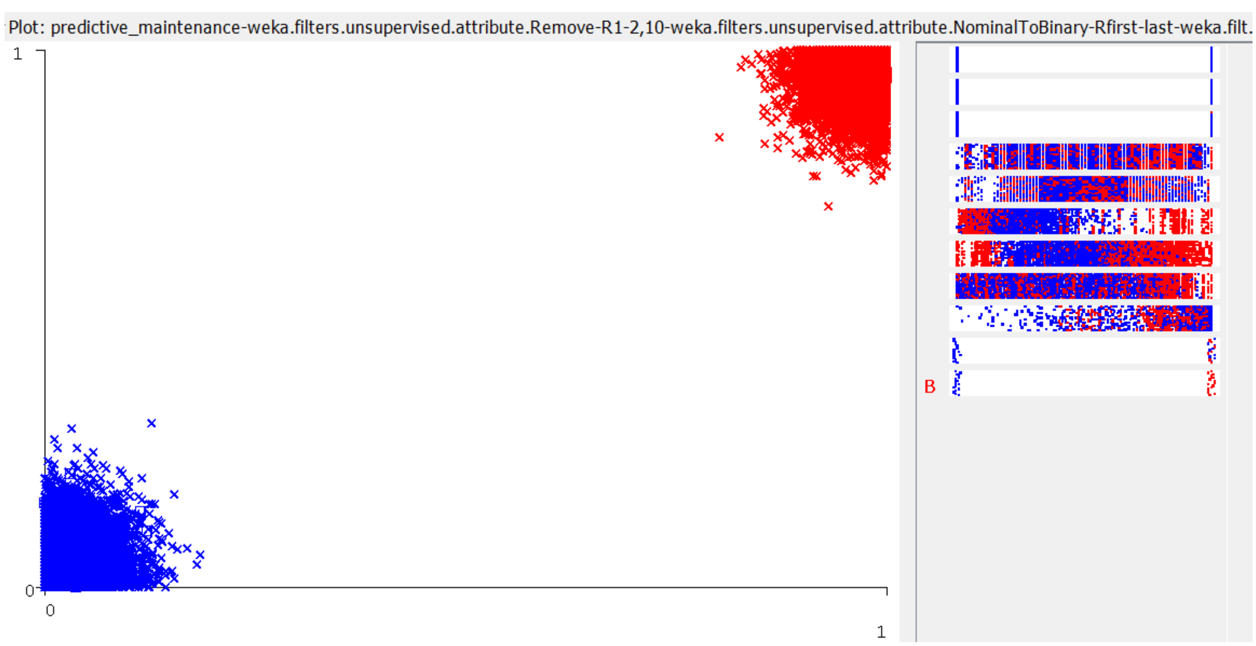
-The parameter



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

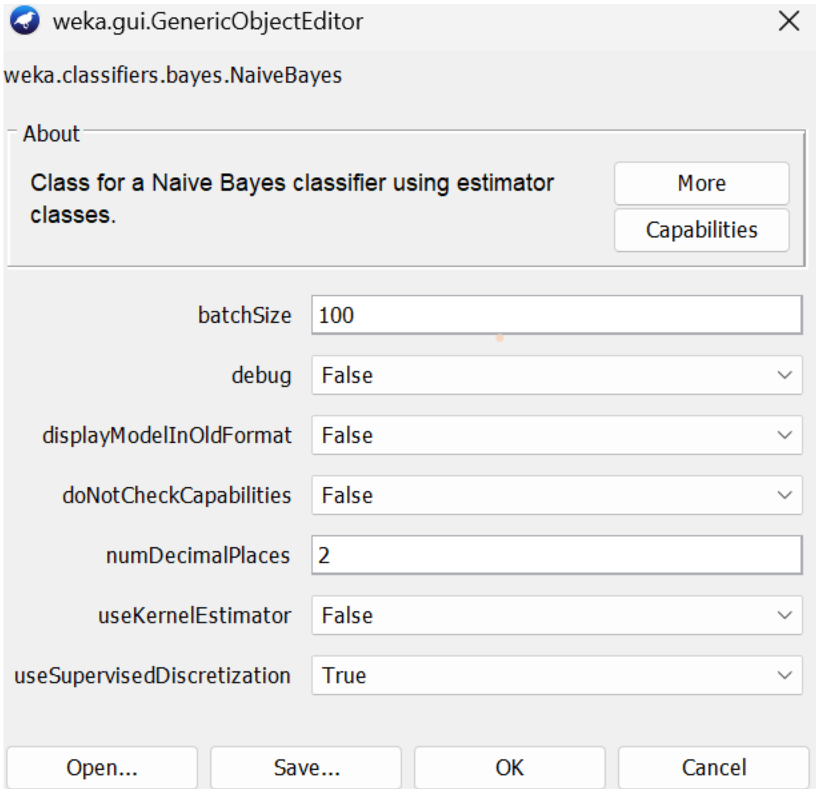
```
Time taken to build model: 0.13 seconds

=== 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:

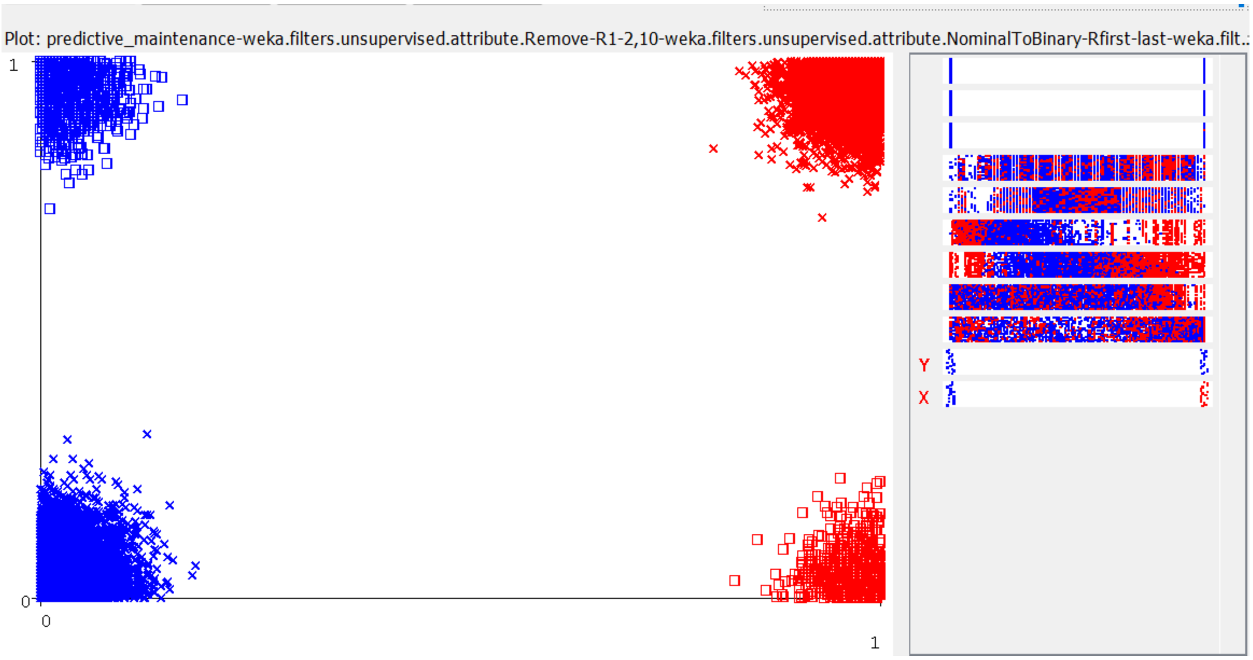
	Predicted 0	Predicted 1
Actual 0	**4299**	701
Actual 1	361	**4636**

- 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.

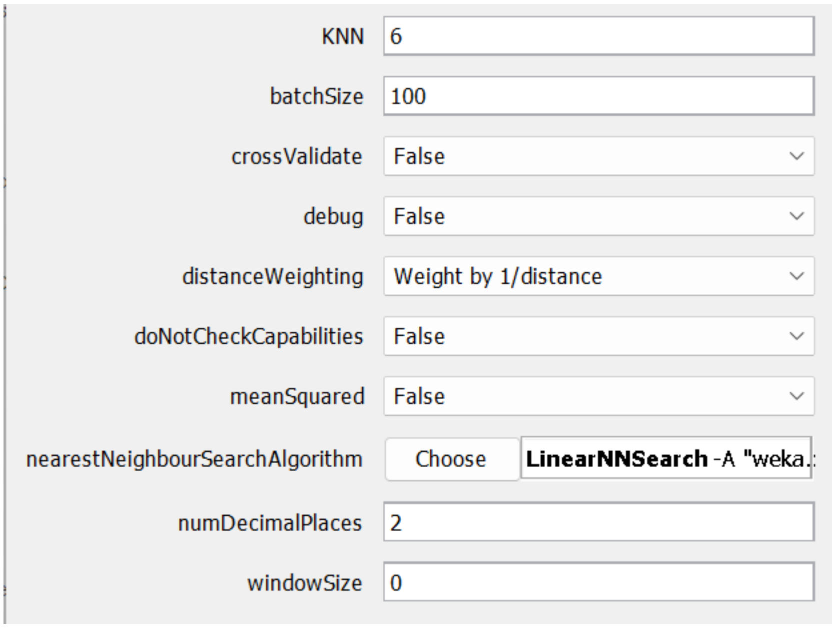
```
Classifier output

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:

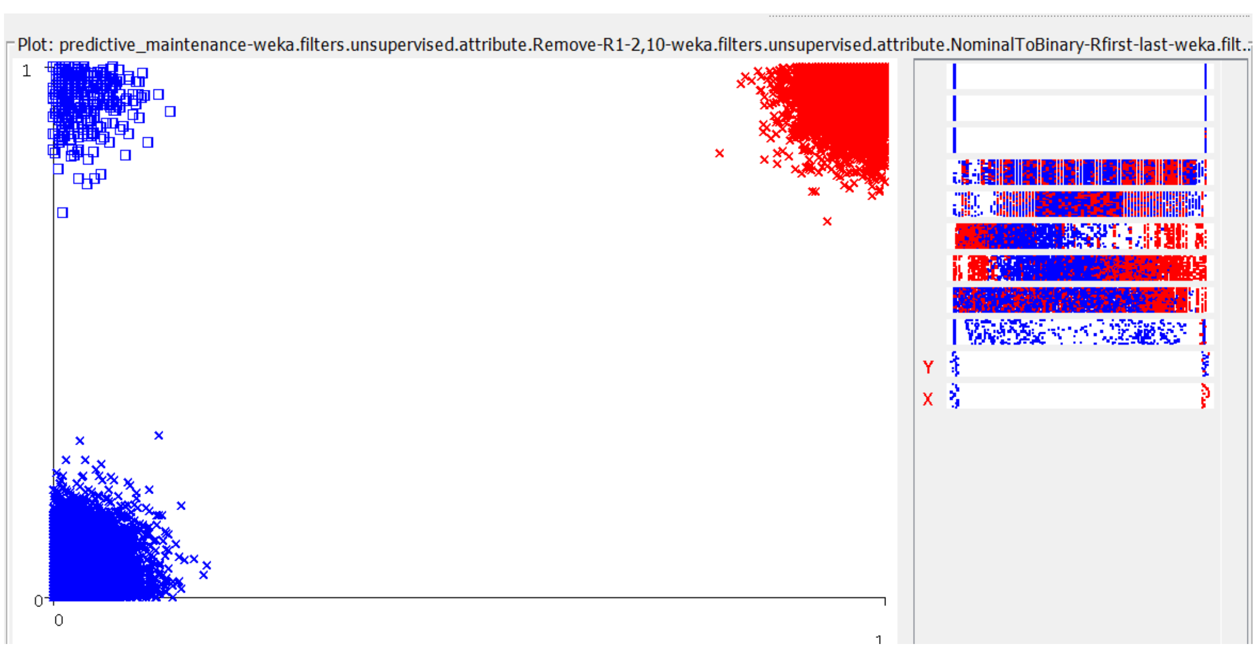
	Predicted 0	Predicted 1
Actual 0	**4692**	308
Actual 1	0	**5000**

- The parameter



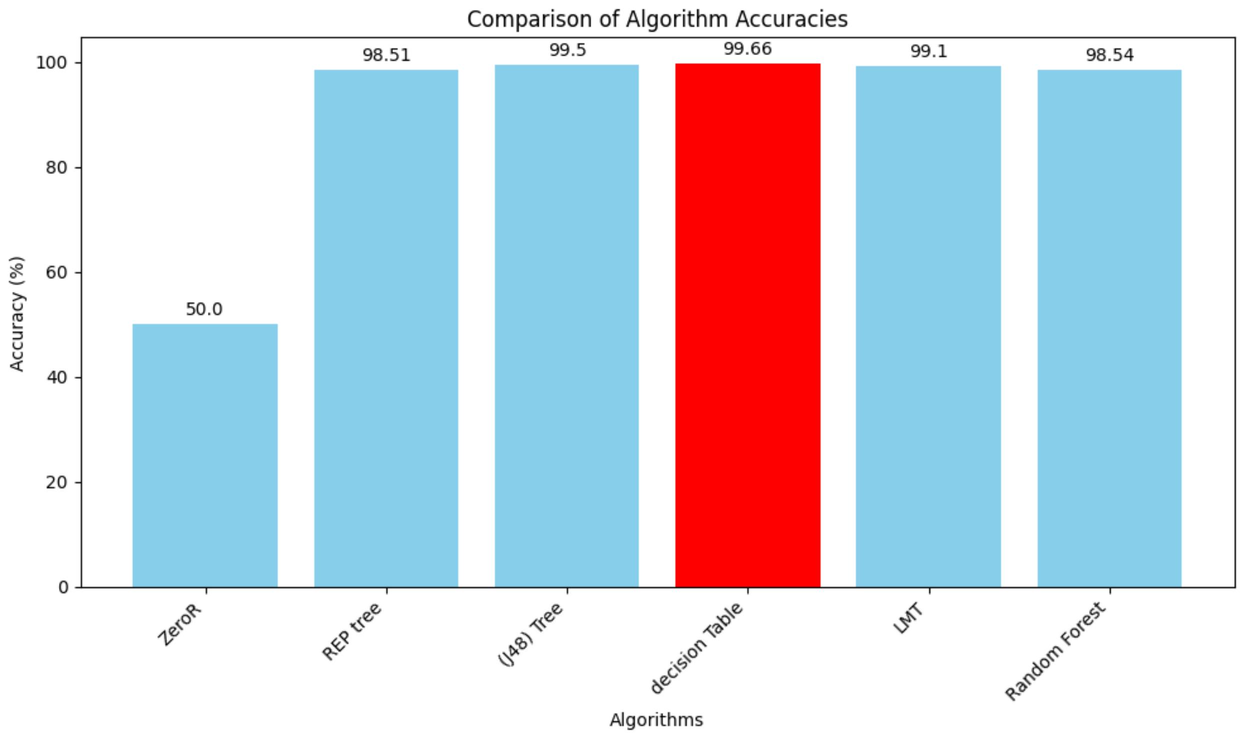
We set KNN to use 6 nearest neighbors and process data in batches of 100 without cross-validation. The model gives more weight to closer neighbors using a simple linear search method. Results are shown with two decimal places, and all neighbors are considered without any window limit. This setup keeps 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/*