

## WEKA Demo

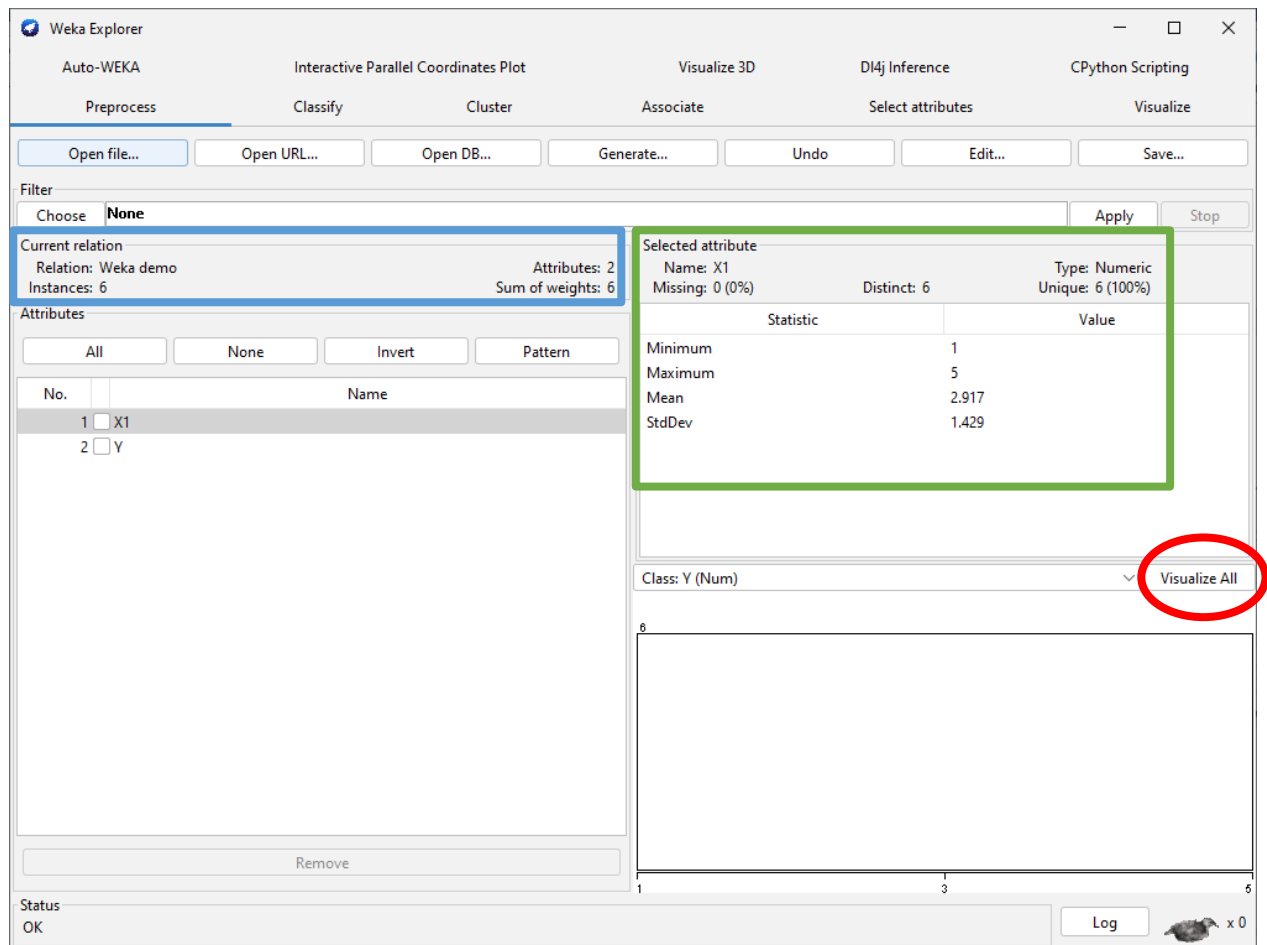
Welcome to the Weka Demo! Weka is a powerful and user-friendly data mining and machine learning tool that offers a comprehensive suite of algorithms for data analysis. Whether you're a beginner or an experienced data scientist, Weka provides an intuitive interface and a wide range of features to help you explore, preprocess, visualize, and model your data. With its extensive collection of algorithms and various evaluation methods, Weka allows you to experiment, compare, and build predictive models with ease. Join us in this demo as we showcase the capabilities of Weka and demonstrate how it can empower you in your data-driven endeavors.



Explorer allows you to effortlessly load datasets, perform data preprocessing tasks, explore data visually, apply machine learning algorithms, and evaluate their performance.

As you click on the Weka Explorer Tab, the Weka Explorer window opens up, offering an array of features designed to simplify your data exploration journey. In the Preprocess tab, we begin by loading a dataset. To do this, simply click on the **"Open File"** option, allowing you to select and import your desired dataset.

For this tutorial, we have chosen a random dataset, which you can observe in the following window. Our objective now is to create a new model using this dataset and explore various techniques to enhance its performance, ultimately achieving better predictive accuracy.



- Here, we have a current relation space that is giving us brief summary about the dataset loaded like in this dataset we have 2 attributes and 6 instances.
- Here, we have select attribute space that is giving us the brief summary about the attributes selected like we have been selected X1 and it is showing us its name, type missing values and brief statistics summary about the selected attribute.
- From the Visualize all option, we can see histogram plots of all attributes of our dataset that gives us idea about the distribution of each attribute.

Now, we will go to the classify tab to make a model.

## CASE 1: Linear Regression

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The 'LinearRegression' classifier is chosen, with parameters `-S 0 -R 1.0E-8 -num-decimal-places 4`. In the 'Test options' section, 'Use training set' is selected. A red box highlights the '(Num) Y' dropdown menu. Another red box highlights the 'Linear Regression Model' output, which shows the equation  $Y = 0.6286 * X1 + 1$ . A third red box highlights the 'Summary' section of the output, which includes the following statistics:

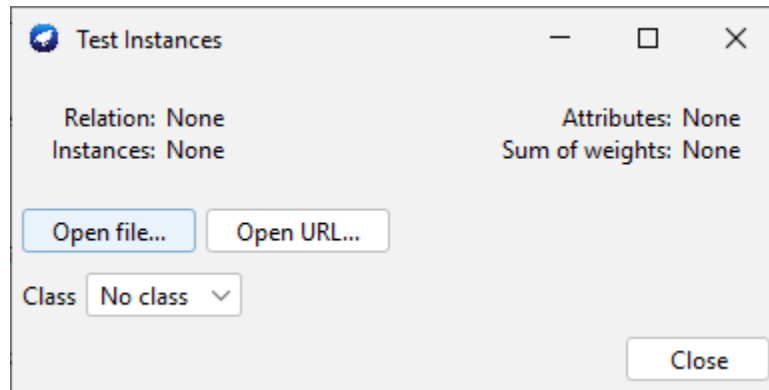
Summary	
Correlation coefficient	0.798
Mean absolute error	0.5143
Root mean squared error	0.6191
Relative absolute error	57.8571 %
Root relative squared error	60.2626 %
Total Number of Instances	6

Red arrows point from the '(Num) Y' dropdown to the text box and from the 'Linear Regression Model' output to the same text box.

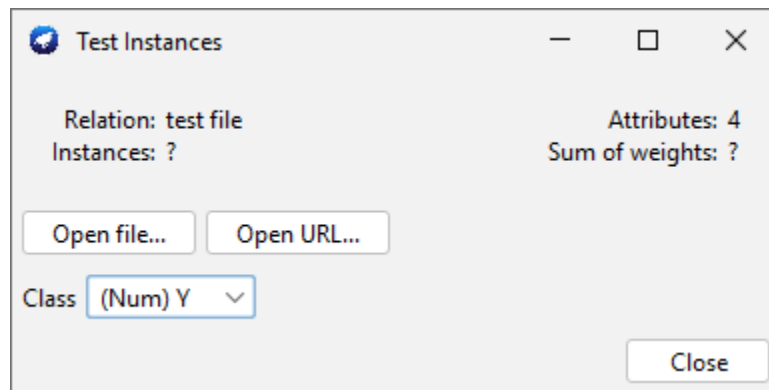
The attribute Y must be selected before making a model because it will be the subject attribute after making model.

In classify tab, choose **“Linear Regression”** model from the classifiers available and set test options to **“Use Training set”**.

From the summary of the Linear Regression model, we can see that the relative absolute error is 57.8571%. Now, we will test our model by supplying the test set to see how good our model is. To test our model, set test options to **“Supplied test set”** and click on the set to supply the test dataset. After clicking on the set the following window will appear.

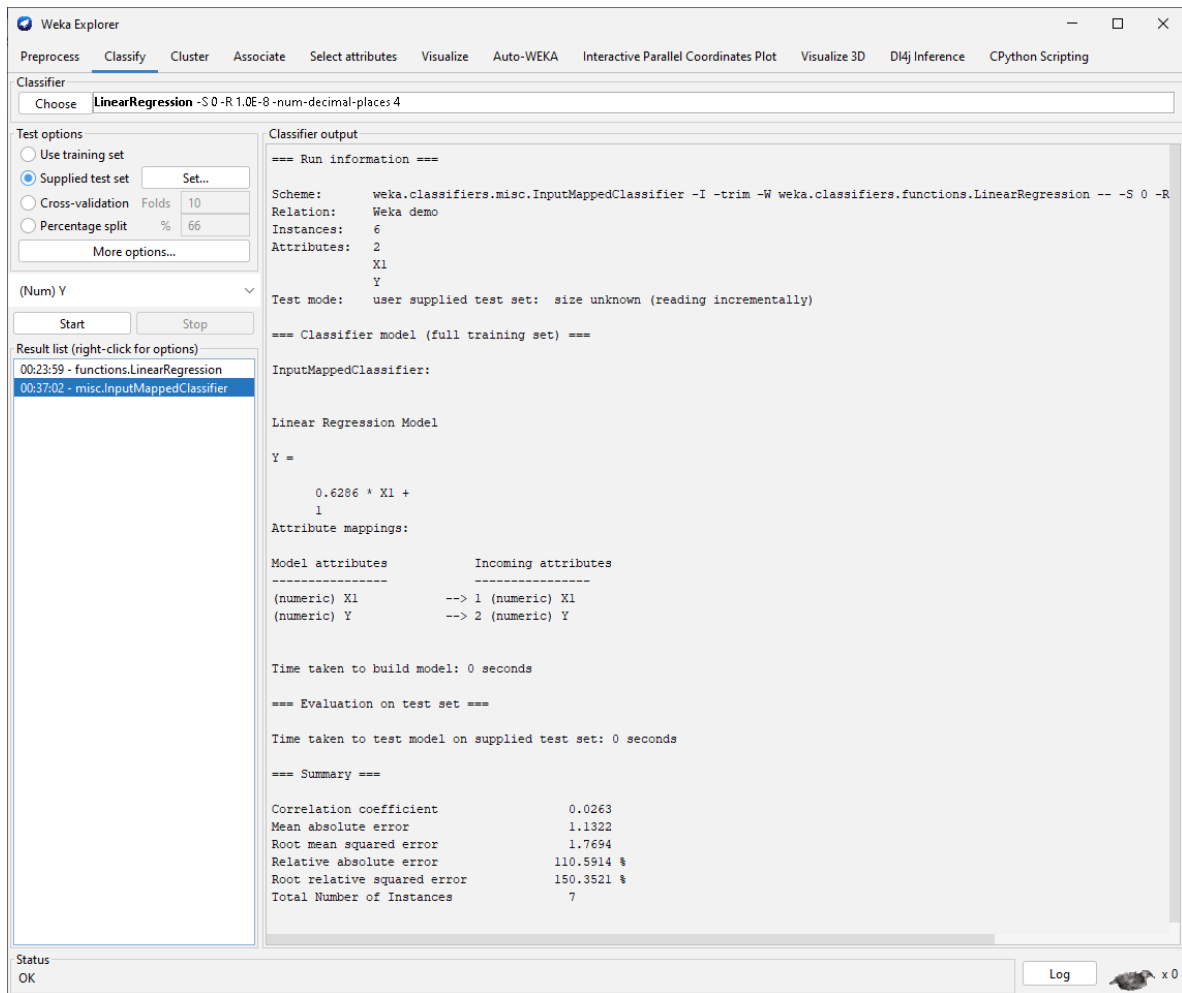


Click on the “**Open file**” and import test dataset. After importing the test dataset, it will look like:



Must remember to set Class to our subject attribute that is “**Y**” in our case and close it.

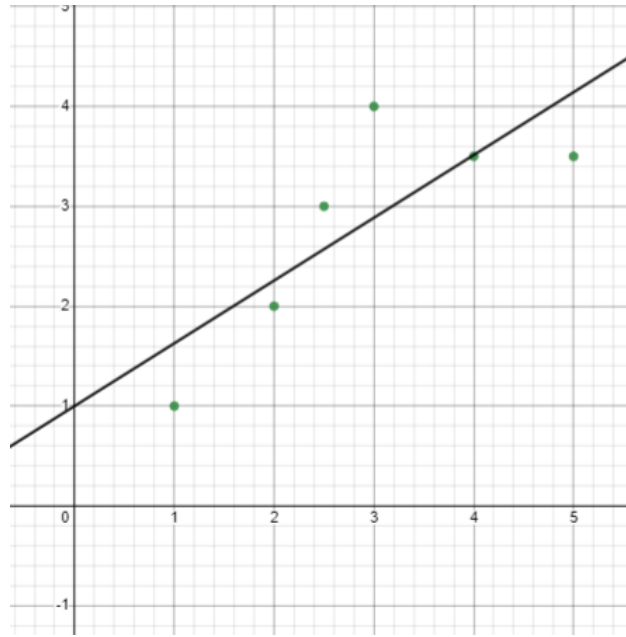
Now, click on the start button on Weka Explorer window to start testing our model.



In this way, we can train and test any model in WEKA.

We can see that both the train error and the test error are high because our model is very simple. This is the case of underfitting (see Appendix A). We need to add complexity in our model by increasing the order of our regression model.

I am using desmos (<https://www.desmos.com/calculator>) to visualize our current linear regression model versus the datapoints available in our dataset.



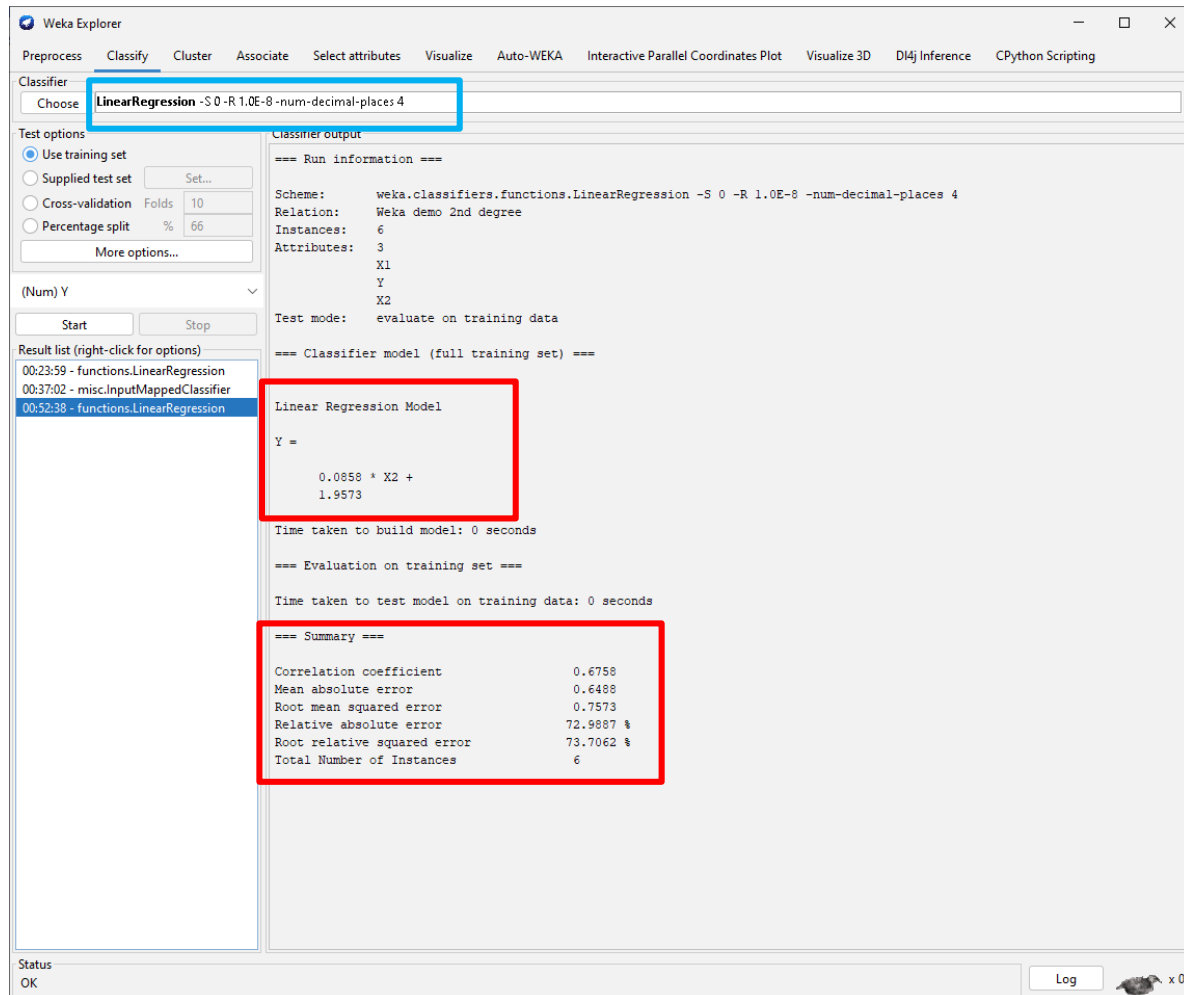
Here, black line is the linear regression line and the green points are the datapoints of our dataset. Now, we will try to adjust quadratic regression line in our dataset to see how it works.

## CASE 2: Quadratic Regression

Unfortunately, in weka we don't have any option available for quadratic regression. So, to apply quadratic regression we will add new attribute "X2" in our dataset that will be the square of "X1" attribute. Then, after applying linear regression we will automatically get quadratic regression.

Go to the Preprocess tab and import the updated dataset having three attributes one is X1, second is X2 and third is Y that is our target attribute.

Let's see what we get after applying linear regression on this updated dataset.



**This time Linear Regression model doesn't include X1 in our model and error also increase. Where did X1 goes???**

Actually, the weka automatically performs pre-processing on dataset before applying model on it. As we know, X2 is the square of X1 means they are closely correlated with each other. So, it takes only one attribute to reduce redundancy in our model. To get X1 in our model as well, we need disable these settings.

**To do this click on the Linear Regression, then following window will open.**

weka.gui.GenericObjectEditor

weka.classifiers.functions.LinearRegression

About

Class for using linear regression for prediction.

More

Capabilities

attributeSelectionMethod M5 method

batchSize 100

debug False

doNotCheckCapabilities False

eliminateColinearAttributes True

minimal False

numDecimalPlaces 4

outputAdditionalStats False

ridge 1.0E-8

useQRDecomposition False

Open... Save... OK Cancel

Change  
attributeSelectionMethod  
to **No attribute selection**  
and  
eliminateColinearAttributes  
to **False**

After making these changes, the updated model will be



Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize Auto-WEKA Interactive Parallel Coordinates Plot Visualize 3D DL4j Inference CPython Scripting

Classifier: Choose **LinearRegression -S 1 -C -R 1.0E-8 -num-decimal-places 4**

Test options

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

(Num) Y

Result list (right-click for options)

- 00:23:59 - functions.LinearRegression
- 00:37:02 - misc.InputMappedClassifier
- 00:52:38 - functions.LinearRegression
- 01:03:53 - functions.LinearRegression**

Classifier output

```

=== Run information ===

Scheme:      weka.classifiers.functions.LinearRegression -S 1 -C -R 1.0E-8 -num-decimal-places 4
Relation:    Weka demo 2nd degree
Instances:   6
Attributes:  3
              X1
              Y
              X2
Test mode:   evaluate on training data

=== Classifier model (full training set) ===

Linear Regression Model

Y =

    2.5604 * X1 +
   -0.3182 * X2 +
   -1.3864

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient      0.949
Mean absolute error         0.2506
Root mean squared error     0.3239
Relative absolute error     28.198 %
Root relative squared error 31.5281 %
Total Number of Instances   6
  
```

Status OK  x 0

Now, we can see that we have been get our desired model. Let's test it by supplying the test set and see what happens.

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize Auto-WEKA Interactive Parallel Coordinates Plot Visualize 3D D4j Inference CPython Scripting

Classifier

Choose LinearRegression -S 1 -C -R 1.0E-8 -num-decimal-places 4

Test options

☐ Use training set

☒ Supplied test set Set...

☐ Cross-validation Folds 10

☐ Percentage split % 66

More options...

(Num) Y

Start Stop

Result list (right-click for options)

00:23:59 - functions.LinearRegression

00:37:02 - misc.InputMappedClassifier

00:52:38 - functions.LinearRegression

01:03:53 - functions.LinearRegression

01:06:14 - misc.InputMappedClassifier

Classifier output

=== Run information ===

Scheme: weka.classifiers.misc.InputMappedClassifier -I -trim -W weka.classifiers.functions.LinearRegression -- -S 1 -C

Relation: Weka demo 2nd degree

Instances: 6

Attributes: 3

X1

Y

X2

Test mode: user supplied test set: size unknown (reading incrementally)

=== Classifier model (full training set) ===

InputMappedClassifier:

Linear Regression Model

Y =

$$2.5604 * X1 + -0.3182 * X2 + -1.3864$$

Attribute mappings:

Model attributes	Incoming attributes
(numeric) X1	--> 1 (numeric) X1
(numeric) Y	--> 2 (numeric) Y
(numeric) X2	--> 3 (numeric) X2

Time taken to build model: 0 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

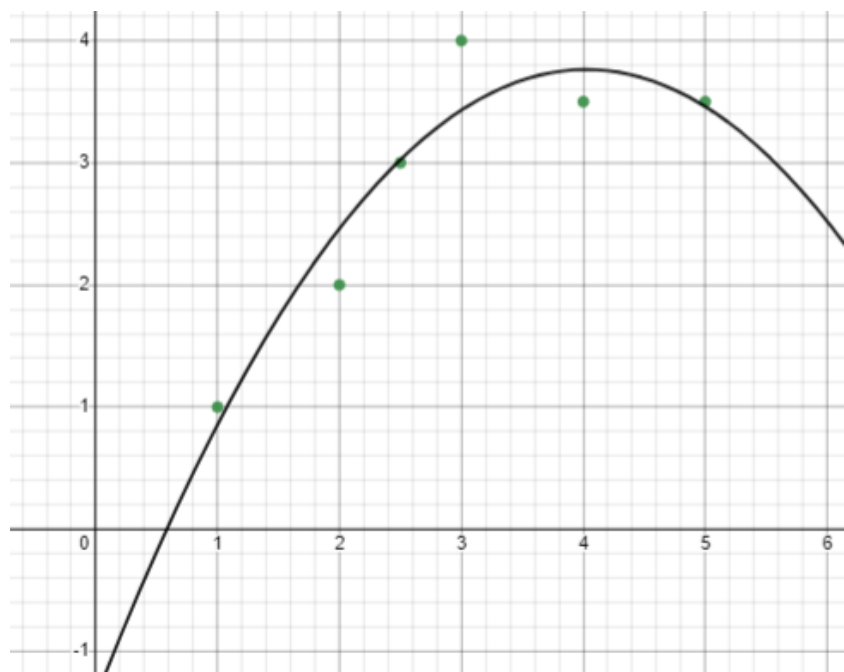
Correlation coefficient	0.927
Mean absolute error	0.3466
Root mean squared error	0.4538
Relative absolute error	33.8544 %
Root relative squared error	38.56 %
Total Number of Instances	7

Status OK

Log x 0

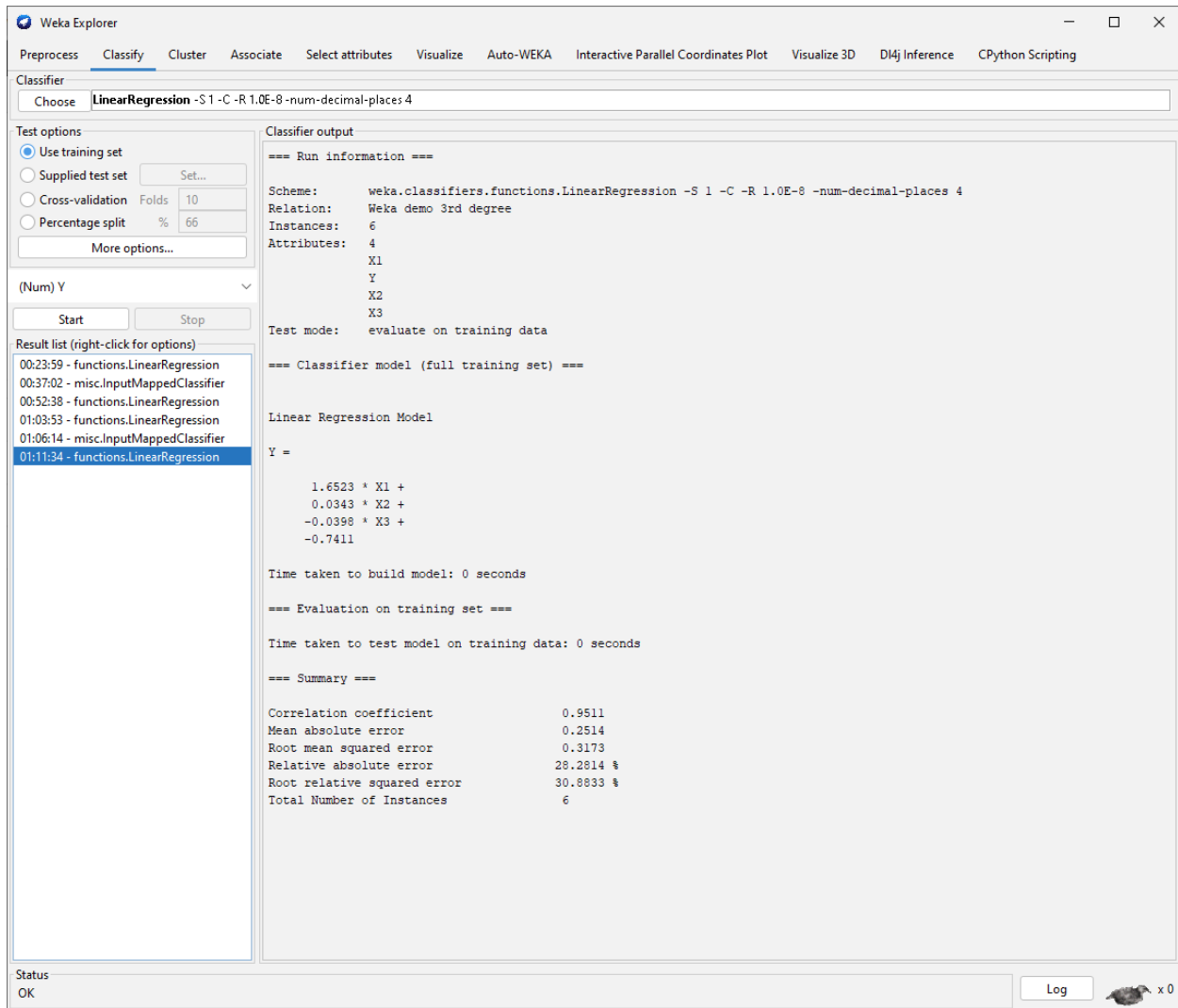
We can see that both the train error as well as the test error has been reduced by changing linear regression to a quadratic regression. Now, the question arises can we reduce error more?

Let's first of all plot this quadratic regression model on desmos to see how it fits in our dataset.



## CASE 3: Cubic Regression

Now, we will apply cubic regression to see can we increase the performance of our model. To do this, we will add another attribute in our dataset named X3 that is the cube of X1.



The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The classifier is set to 'LinearRegression' with parameters '-S 1 -C -R 1.0E-8 -num-decimal-places 4'. The test options are set to 'Use training set' with 'Cross-validation' (Folds: 10) and 'Percentage split' (%: 66). The result list on the left shows several entries, with the most recent one selected: '01:11:34 - functions.LinearRegression'. The classifier output on the right displays the following information:

```
=== Run information ===

Scheme:      weka.classifiers.functions.LinearRegression -S 1 -C -R 1.0E-8 -num-decimal-places 4
Relation:    Weka demo 3rd degree
Instances:    6
Attributes:  4
              X1
              Y
              X2
              X3

Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Linear Regression Model

Y =

    1.6523 * X1 +
    0.0343 * X2 +
   -0.0398 * X3 +
   -0.7411

Time taken to build model: 0 seconds

=== Evaluation on training set ===

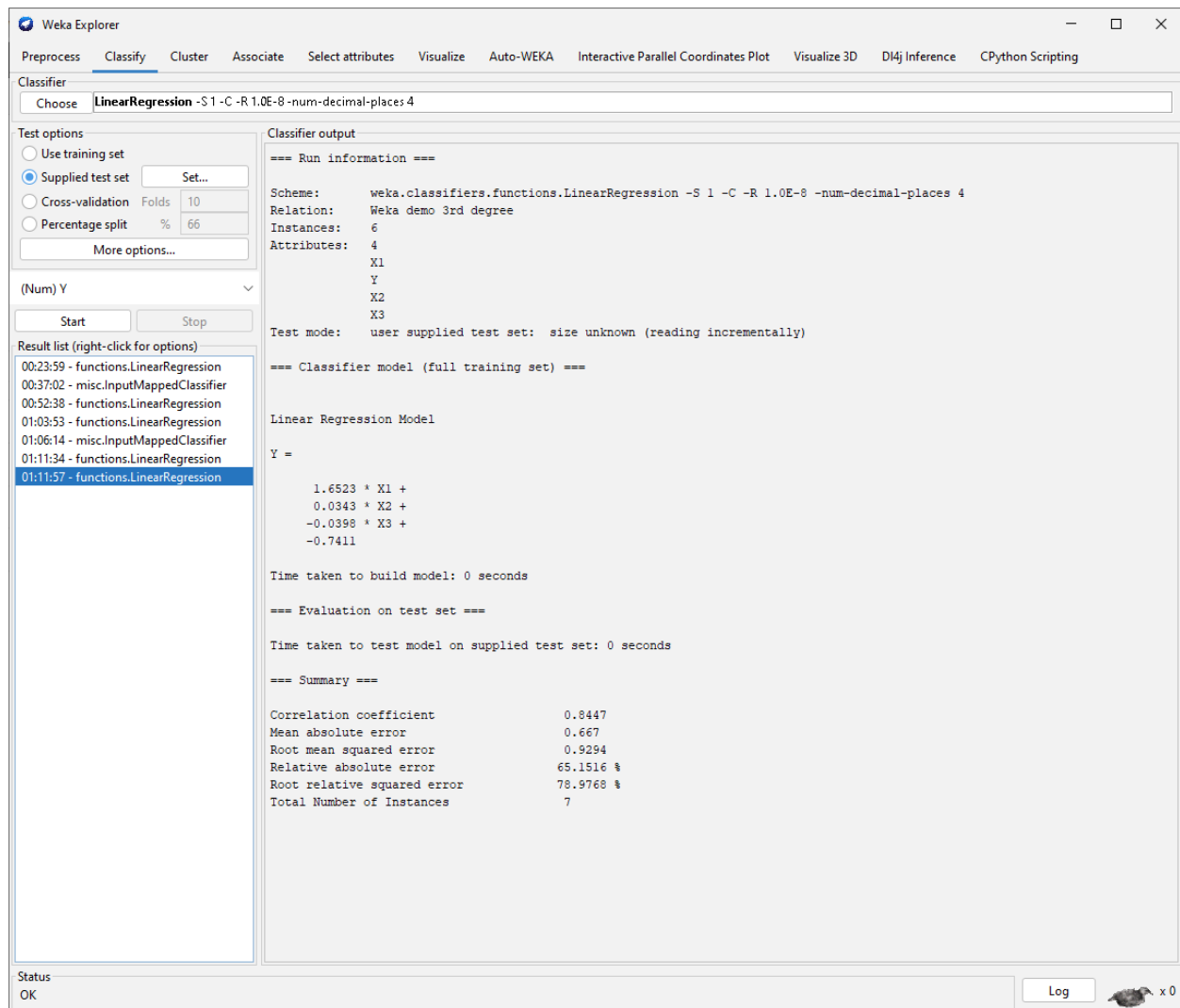
Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient      0.9511
Mean absolute error          0.2514
Root mean squared error     0.3173
Relative absolute error     28.2814 %
Root relative squared error 30.8833 %
Total Number of Instances    6
```

The status bar at the bottom shows 'Status OK' and a 'Log' button.

We can see that the train error almost remains the same.



But the test error has been increased. So, what causes the test error to increase.

Because, in case of cubic regression our model becomes very complex means it has been fitted very well on train dataset and doesn't perform well on the test dataset. This is the case of overfitting the model (see Appendix A).

## Conclusion:

The given summary presents the performance results of three regression models: Linear Regression, Quadratic Regression, and Cubic Regression.

CASE	Train Error	Test Error	Model
Linear Regression	57.8571%	110.5914%	Underfitting
Quadratic Regression	28.198%	33.8544%	Good
Cubic Regression	28.2814%	65.1516%	Overfitting

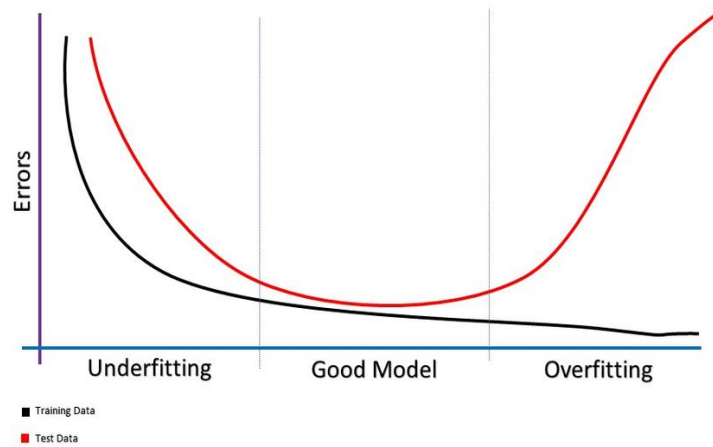
The Linear Regression model exhibits a high train error of 57.8571% and an even higher test error of 110.5914%. These values suggest that the model is underfitting the data, meaning it fails to capture the underlying relationships adequately. Consequently, its predictive performance on unseen data is poor.

In contrast, the Quadratic Regression model demonstrates significantly improved performance. It achieves a train error of 28.198% and a test error of 33.8544%. These values indicate that the model fits the data reasonably well, leading to good predictive accuracy on unseen instances.

However, the Cubic Regression model showcases a train error of 28.2814%, comparable to the Quadratic Regression model. Surprisingly, the test error jumps to 65.1516%, indicating a significant drop in performance. This suggests that the Cubic Regression model is overfitting the training data, capturing noise or outliers that hinder its generalization ability to unseen instances.

In summary, the Quadratic Regression model appears to strike the right balance between model complexity and generalization. It achieves relatively low errors on both the training and test datasets, indicating good predictive performance. Meanwhile, the Linear Regression model underfits the data, while the Cubic Regression model overfits, resulting in diminished performance on the test dataset.

## Appendix A



**Underfitting** occurs when a machine learning model is too simple and fails to capture the underlying patterns in the training data. It results in high training error and high test error. The model is unable to generalize well to unseen data and performs poorly on both the training and test sets.

**Overfitting** happens when a machine learning model is overly complex and learns the training data too well, including noise or random fluctuations. It leads to very low training error but high test error. The model memorizes the training data instead of capturing the underlying patterns, resulting in poor performance on new, unseen data.

**In summary:**

**Underfitting: High training error and high test error.**

**Overfitting: Very low training error and high test error.**