

# Package ‘LearnSL’

January 20, 2025

**Type** Package

**Title** Learn Supervised Classification Methods Through Examples and Code

**Version** 1.0.0

**Description** Supervised classification methods, which (if asked) can provide step-by-step explanations of the algorithms used, as described in PK Josephine et. al., (2021) <[doi:10.59176/kjcs.v1i1.1259](https://doi.org/10.59176/kjcs.v1i1.1259)>; and datasets to test them on, which highlight the strengths and weaknesses of each technique.

**URL** <https://github.com/ComiSeng/LearnSL>

**BugReports** <https://github.com/ComiSeng/LearnSL/issues>

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**Depends** R (>= 4.3.0)

**Imports** cli (>= 3.6.1)

**NeedsCompilation** no

**Author** Víctor Amador Padilla [aut, cre],  
Juan Jose Cuadrado Gallego [ctb]  
(<<https://orcid.org/0000-0001-8178-5556>>),  
Universidad de Alcalá [cph]

**Maintainer** Víctor Amador Padilla <[victor.amador@edu.uah.es](mailto:victor.amador@edu.uah.es)>

**Repository** CRAN

**Date/Publication** 2023-09-19 14:30:04 UTC

## Contents

act_method . . . . .	2
db1rl . . . . .	3
db2 . . . . .	4

db3 . . . . .	4
db_flowers . . . . .	5
db_per_and . . . . .	5
db_per_or . . . . .	6
db_per_xor . . . . .	6
db_tree_struct . . . . .	7
decision_tree . . . . .	7
knn . . . . .	9
multivariate_linear_regression . . . . .	10
perceptron . . . . .	11
polynomial_regression . . . . .	12
print.tree_struct . . . . .	13

<b>Index</b>	<b>14</b>
--------------	-----------

---

act_method	<i>Activation Function</i>
------------	----------------------------

---

## Description

Upon a received input, calculates the output based on the selected activation function

## Usage

```
act_method(method, x)
```

## Arguments

method	Activation function to be used. It must be one of "step", "sine", "tangent", "linear", "relu", "gelu" or "swish".
x	Input value to be used in the activation function.

## Details

Formulae used:

*step*

$$f(x) = \begin{cases} 0 & \text{if } x < \text{threshold} \\ 1 & \text{if } x \geq \text{threshold} \end{cases}$$

*sine*

$$f(x) = \sinh(x)$$

*tangent*

$$f(x) = \tanh(x)$$

*linear*

$x$

**relu**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

**gelu**

$$f(x) = \frac{1}{2} \cdot x \cdot \left( 1 + \tanh \left( \sqrt{\frac{2}{\pi}} \cdot (x + 0.044715 \cdot x^3) \right) \right)$$

**swish**

$$f(x) = \frac{x}{1 + e^{-x}}$$

**Value**

List with the weights of the inputs.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
act_method("step", 0.3)
act_method("gelu", 0.7)
```

---

 db1r1

*Test Database 1*


---

**Description**

Test Database 1

**Usage**

db1r1

**Format**

db1r1:

A data frame with 4 independent variables (first 4 columns, representing different line types). The last column is the independent variable.

---

db2

*Test Database 6*

---

**Description**

Test Database 6

**Usage**

db2

**Format**

db2:

A data frame with 3 independent variables (first 3 columns) and one independent variable (last column). It has information about vehicles.

---

db3

*Test Database 7*

---

**Description**

Test Database 7

**Usage**

db3

**Format**

db3:

A data frame with 3 independent variables (first 3 columns) and one independent variable (last column). It has information about vehicles. Similar to db2 but a little bit more complex.

---

db_flowers	<i>Test Database 5</i>
------------	------------------------

---

**Description**

Test Database 5

**Usage**

db\_flowers

**Format**

db\_flowers:

A data frame representing features of flowers. It has 4 independent variables (first 4 columns) and one independent variable (last column).

---

db_per_and	<i>Test Database 2</i>
------------	------------------------

---

**Description**

Test Database 2

**Usage**

db\_per\_and

**Format**

db\_per\_and:

A data frame with 3 independent variables (first 3 columns) and one independent variable (last column). It represents a 3 input "AND" logic gate.

---

db\_per\_or

*Test Database 3*

---

**Description**

Test Database 3

**Usage**

db\_per\_or

**Format**

db\_per\_or:

A data frame with 3 independent variables (first 3 columns) and one independent variable (last column). It represents a 3 input "OR" logic gate.

---

db\_per\_xor

*Test Database 4*

---

**Description**

Test Database 4

**Usage**

db\_per\_xor

**Format**

db\_per\_xor:

A data frame with 3 independent variables (first 3 columns) and one independent variable (last column). It represents a 3 input "XOR" logic gate.

---

db_tree_struct	<i>Test Database 8</i>
----------------	------------------------

---

**Description**

Test Database 8

**Usage**

db\_tree\_struct

**Format**

db\_tree\_struct:  
Decision tree structure. output of the decision\_tree() function "decision\_tree(db2, "VehicleType", 4, "gini")"

---

decision_tree	<i>Decision Tree</i>
---------------	----------------------

---

**Description**

This function creates a decision tree based of an example dataset, calculating the best classifier possible in each step. Only creates perfect divisions, this means, if the rule doesn't create a classified group, it is not considered. It is specifically designed for categorical values. Continues values are not recommended as they will be treated as categorical ones.

**Usage**

```
decision_tree(  
  data,  
  classy,  
  m,  
  method = "entropy",  
  details = FALSE,  
  waiting = TRUE  
)
```

**Arguments**

data	A data frame with already classified observations. Each column represents a parameter of the value. Each row is a different observation. The column names in the parameter "data" must not contain the sequence of characters " or ". As this is supposed to be a binary decision rules generator and not a binary decision tree generator, no tree structures are used, except for the information gain formulas.
------	--

classy	Name of the column we want the data to be classified by. the set of rules obtained will be calculated according to this.
m	Maximum numbers of child nodes each node can have.
method	The definition of Gain. It must be one of "Entropy", "Gini" or "Error".
details	Boolean value. If it is set to "TRUE" multiple clarifications and explanations are printed along the code
waiting	If TRUE while details = TRUE. The code will stop in each "block" of code and wait for the user to press "enter" to continue.

### Details

If data is not perfectly classifiable, the code will not finish.

Available information gain methods are:

**Entropy** The formula to calculate the entropy works as follows:  $p_i = - \sum f_i p_i \cdot \log 2 p_i$

**Gini** The formula to calculate gini works as follows:  $p_i = 1 - \sum f_i p_i^2$

**Error** The formula to calculate error works as follows:  $p_i = 1 - \max(f_i p_i)$

Once the impurity is calculated, the information gain is calculated as follows:

$$IG = I_{father} - \sum \frac{count(sonvalues)}{count(fathervalues)} \cdot I_{son}$$

### Value

Structure of the tree. List with a list per tree level. Each of these contains a list per level node, each of these contains a list with the node's filtered data, the node's id, the father's node id, the height that node is at, the variable it filters by, the value that variable is filtered by and the information gain of the division

### Author(s)

Víctor Amador Padilla, <victor.amador@edu.uah.es>

### Examples

```
# example code
decision_tree(db3, "VehicleType", 5, "entropy", details = TRUE, waiting = FALSE)
decision_tree(db2, "VehicleType", 4, "gini")
```



---

knn	<i>K-Nearest Neighbors</i>
-----	----------------------------

---

**Description**

This function applies knn algorithm to classify data.

**Usage**

```
knn(
  data,
  ClassLabel,
  p1,
  d_method = "euclidean",
  k,
  p = 3,
  details = FALSE,
  waiting = TRUE
)
```

**Arguments**

data	Data frame with already classified observations. Each column represents a parameter of the values. The last column contains the output, this means, the expected output when the other column values are inputs. Each row is a different observation.
ClassLabel	String containing the name of the column of the classes we want to classify
p1	Vector containing the parameters of the new value that we want to classify.
d_method	String with the name of the distance method that will be used. It must be one of "Euclidean", "Manhattan", "Cosine", "Chebyshev", "Minkowski", "Canberra", "Octile", "Hamming", "Binary" or "Jaccard". Where both "Hamming" and "Binary" use the same method, as it is known by both names.
k	Number of closest values that will be considered in order to classify the new value ("p1").
p	Exponent used in the Minkowski distance. 3 by default, otherwise if specified.
details	Boolean value. If it is set to "TRUE" multiple clarifications and explanations are printed along the code
waiting	If TRUE while details = TRUE. The code will stop in each "block" of code and wait for the user to press "enter" to continue.

**Value**

Value of the new classified example.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
knn(db_flowers,"ClassLabel", c(4.7, 1.2, 5.3, 2.1), "chebyshev", 4)
knn(db_flowers,"ClassLabel", c(4.7, 1.5, 5.3, 2.1), "chebyshev", 5)
knn(db_flowers,"ClassLabel", c(6.7, 1.5, 5.3, 2.1), "Euclidean", 2, details = TRUE, waiting = FALSE)
knn(db_per_or,"y", c(1,1,1), "Hamming", 3, details = TRUE, waiting = FALSE)
```

---

multivariate\_linear\_regression  
*Multivariate Linear Regression*

---

**Description**

Calculates and plots the linear regression of a given set of values. Being all of them independent values but one, which is the dependent value. It provides information about the process and intermediate values used to calculate the line equation.

**Usage**

```
multivariate_linear_regression(data, details = FALSE, waiting = TRUE)
```

**Arguments**

data	x*y data frame with already classified observations. Each column represents a parameter of the values (independent variable). The last column represents the classification value (dependent variable). Each row is a different observation.
details	Boolean value. If it is set to "TRUE" multiple clarifications and explanations are printed along the code
waiting	If TRUE while details = TRUE. The code will stop in each "block" of code and wait for the user to press "enter" to continue.

**Value**

List containing a list for each independent variable, each one contains, the variable name, the intercept and the slope.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
multivariate_linear_regression(db1rl)
```

perceptron

*Perceptron***Description**

Binary classification algorithm that learns to separate two classes of data points by finding an optimal decision boundary (hyper plane) in the feature space.

**Usage**

```
perceptron(
  training_data,
  to_clasify,
  activation_method,
  max_iter,
  learning_rate,
  details = FALSE,
  waiting = TRUE
)
```

**Arguments**

training_data	Data frame with already classified observations. Each column represents a parameter of the values. The last column contains the output, this means, the expected output when the other column values are inputs. Each row is a different observation. It works as training data.
to_clasify	Vector containing the parameters of the new value that we want to classify.
activation_method	Activation function to be used. It must be one of "step", "sine", "tangent", "linear", "relu", "gelu" or "swish".
max_iter	Maximum epoch during the training phase.
learning_rate	Value at which the perceptron will learn from previous epochs mistakes.
details	Boolean value. If it is set to "TRUE" multiple clarifications and explanations are printed along the code
waiting	If TRUE while details = TRUE. The code will stop in each "block" of code and wait for the user to press "enter" to continue.

**Details**

Functioning:

**Step 1** Generate a random weight for each independent variable.

**Step 2** Check if the weights classify correctly. If they do, go to step 4

**Step 3** Adjust weights based on the error between the expected output and the real output. If max\_iter is reached go to step 4. If not, go to step 2.

**Step 4** Return the weights and use them to classify the new value

**Value**

List with the weights of the inputs.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
perceptron(db_per_or, c(1, 1, 1), "gelu", 1000, 0.1)
perceptron(db_per_and, c(0,0,1), "swish", 1000, 0.1, TRUE, FALSE)
```

---

polynomial\_regression *Multivariate Polynomial Regression*

---

**Description**

Calculates and plots the polynomial regression of a given set of values. Being all of them independent values but one, which is the dependent value. It provides (if asked) information about the process and intermediate values used to calculate the line equation. The approximation depends entirely in the degree of the equations.

**Usage**

```
polynomial_regression(data, degree, details = FALSE, waiting = TRUE)
```

**Arguments**

data	x*y data frame with already classified observations. Each column represents a parameter of the values (independent variable). The last column represents the classification value (dependent variable). Each row is a different observation.
degree	Degree of the equations approximation.
details	Boolean value. If it is set to "TRUE" multiple clarifications and explanations are printed along the code
waiting	If TRUE while details = TRUE. The code will stop in each "block" of code and wait for the user to press "enter" to continue.

**Value**

List containing a list for each independent variable, each one contains the equation coefficients.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
polynomial_regression(db1r1,4, TRUE, FALSE)
polynomial_regression(db1r1,6)
```

---

`print.tree_struct`      *Print Tree Structure*

---

**Description**

This function prints the structure of a tree, generated by the `decision_tree` function.

**Usage**

```
## S3 method for class 'tree_struct'
print(x, ...)
```

**Arguments**

`x`                    The tree structure.  
`...`                 Extra useless parameters.

**Details**

It must receive a `tree_struct` data type.

**Value**

nothing.

**Author(s)**

Víctor Amador Padilla, <victor.amador@edu.uah.es>

**Examples**

```
# example code
print(db_tree_struct)
```

# Index

## \* datasets

- db1r1, 3
- db2, 4
- db3, 4
- db\_flowers, 5
- db\_per\_and, 5
- db\_per\_or, 6
- db\_per\_xor, 6
- db\_tree\_struct, 7

act\_method, 2

- db1r1, 3
- db2, 4
- db3, 4
- db\_flowers, 5
- db\_per\_and, 5
- db\_per\_or, 6
- db\_per\_xor, 6
- db\_tree\_struct, 7
- decision\_tree, 7

knn, 9

multivariate\_linear\_regression, 10

perceptron, 11

polynomial\_regression, 12

print.tree\_struct, 13