



# SDSC6004 Data Analytics and Data Mining II

Zijun Zhang

Parts of contents from textbook, The Elements of Statistical learning

# Regression: Definition

- I Given a collection of records (training set )
  - Each record is by characterized by a tuple  $(\mathbf{x}, y)$ , where  $\mathbf{x}$  is the attribute set and  $y$  is the numerical
- I Task:
  - Learn a model which predicts the value of  $y$  based on the attribute set  $\mathbf{x}$

# Basic statistical regression methods

- Linear regression

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j. \quad (3.1)$$

- Learning goal:

$$\beta = \operatorname{argmin}_{\beta} \sum (f(X, \beta) - y)$$

# Basic statistical regression methods

- Ridge

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}. \quad (3.41)$$

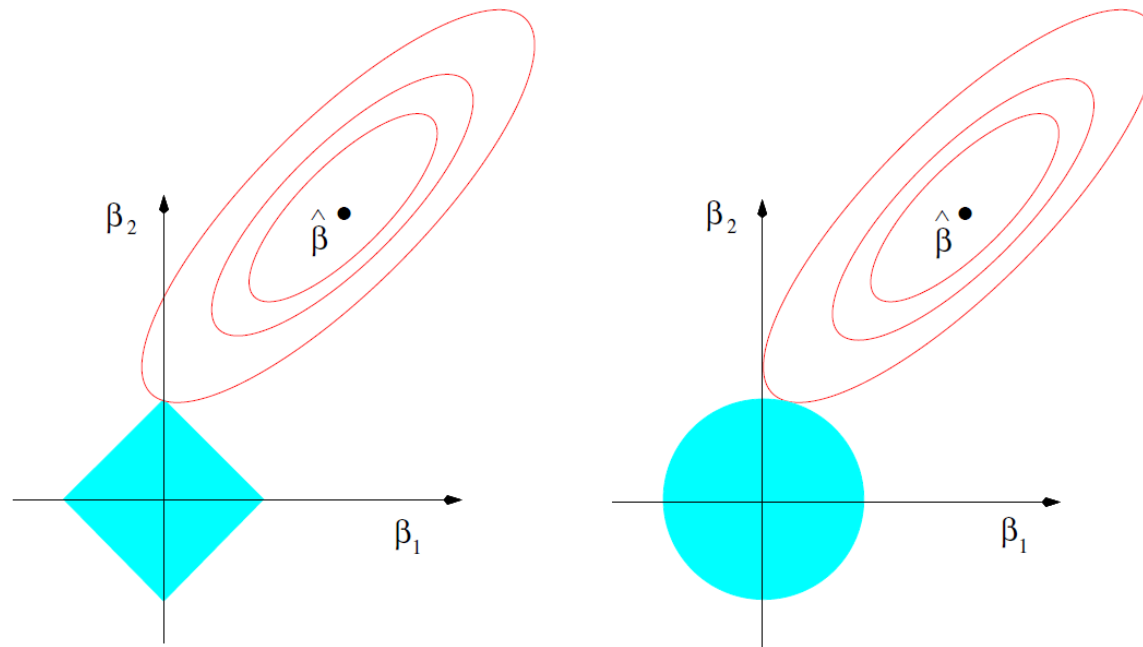
# Basic statistical regression methods

- LASSO

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}. \quad (3.52)$$



# Ridge v.s. Lasso



**FIGURE 3.11.** Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions  $|\beta_1| + |\beta_2| \leq t$  and  $\beta_1^2 + \beta_2^2 \leq t^2$ , respectively, while the red ellipses are the contours of the least squares error function.

# k Nearest Neighbor Regression

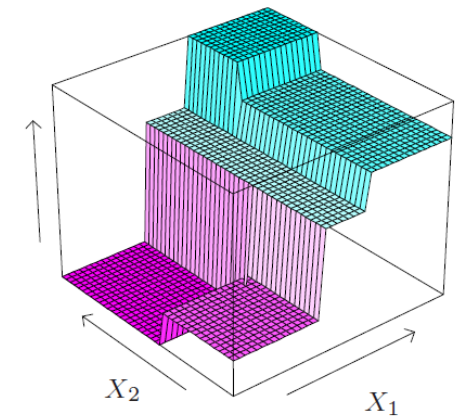
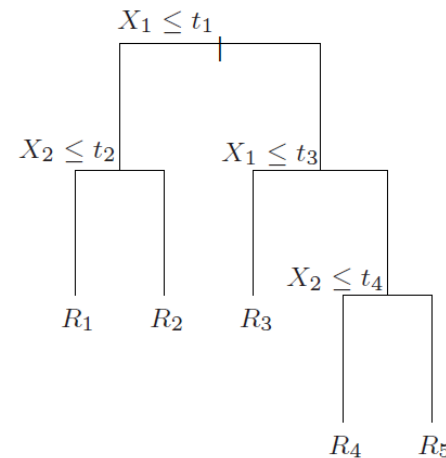
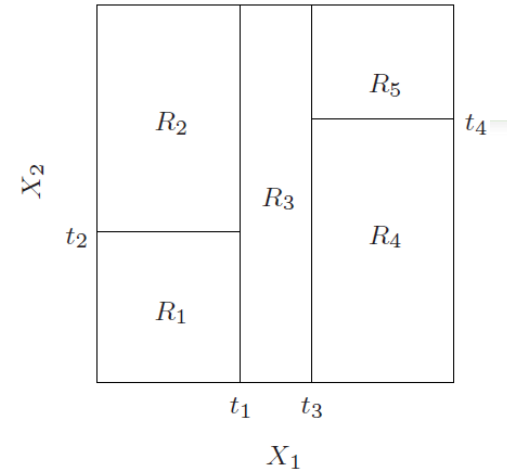
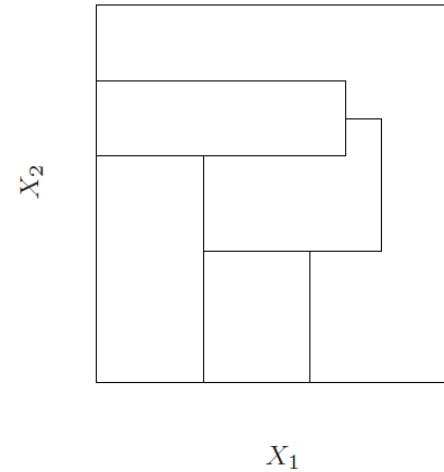
- Searching k nearest neighbors based on the value of k and the distance metric.
- Simply take the average of y of k nearest neighbors and use it as the estimated y of the target data.

$$\hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x)) \quad (6.1)$$

# Regression Trees

- Partition space of data
- Find the optimal split
- Taking the average of  $y$  of data points in nodes of tree as the final estimates

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m).$$





# Regression Trees

- A greedy split strategy and the split criteria

$$R_1(j, s) = \{X | X_j \leq s\} \text{ and } R_2(j, s) = \{X | X_j > s\}.$$

- which leads to

$$\min_{j, s} \left[ \min_{c_1} \sum_{x_i \in R_1(j, s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j, s)} (y_i - c_2)^2 \right].$$

# SVM Regression

- To find a linear function  $f(x)=x'\beta+b$
- with the minimal norm value  $(\beta'\beta)$ , which is formulated as

$$J(\beta)=1/2(\beta'\beta)$$

s.t.

$$\forall n:[y_n-(x_n'\beta+b)]\leq\varepsilon$$

It means to have all residuals less than  $\varepsilon$ .

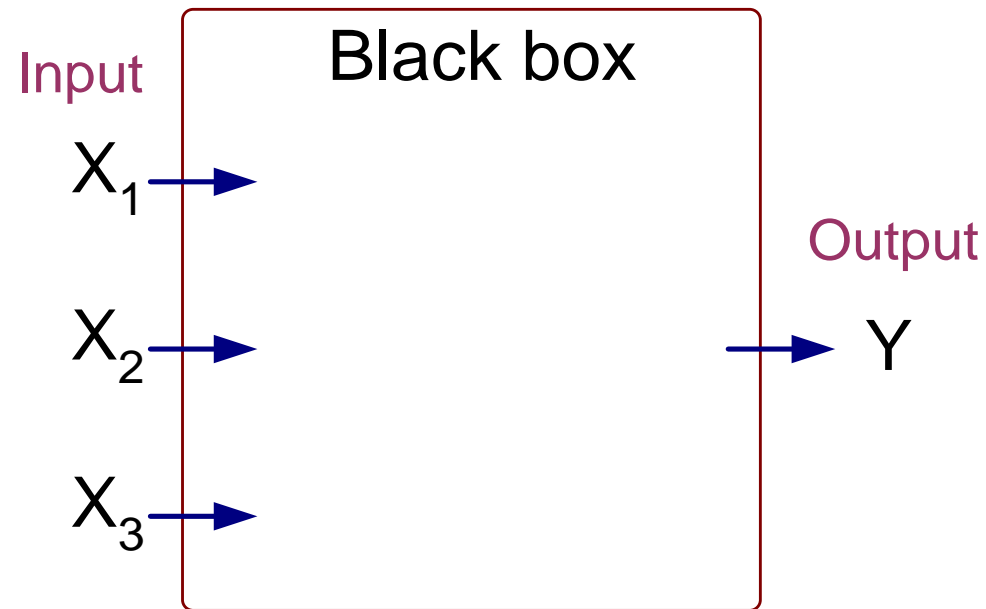
# SVM Regression

- It is possible that the previous idea  $f(x)$  might not exist
- We can introduce the slack variable  $\xi$
- We can use kernels to project  $x$  to other planes  $G(x_n, x)$ ,  
commonly considered kernels: Linear, Gaussian, Polynomial

# Artificial Neural Networks

- Start with classification examples

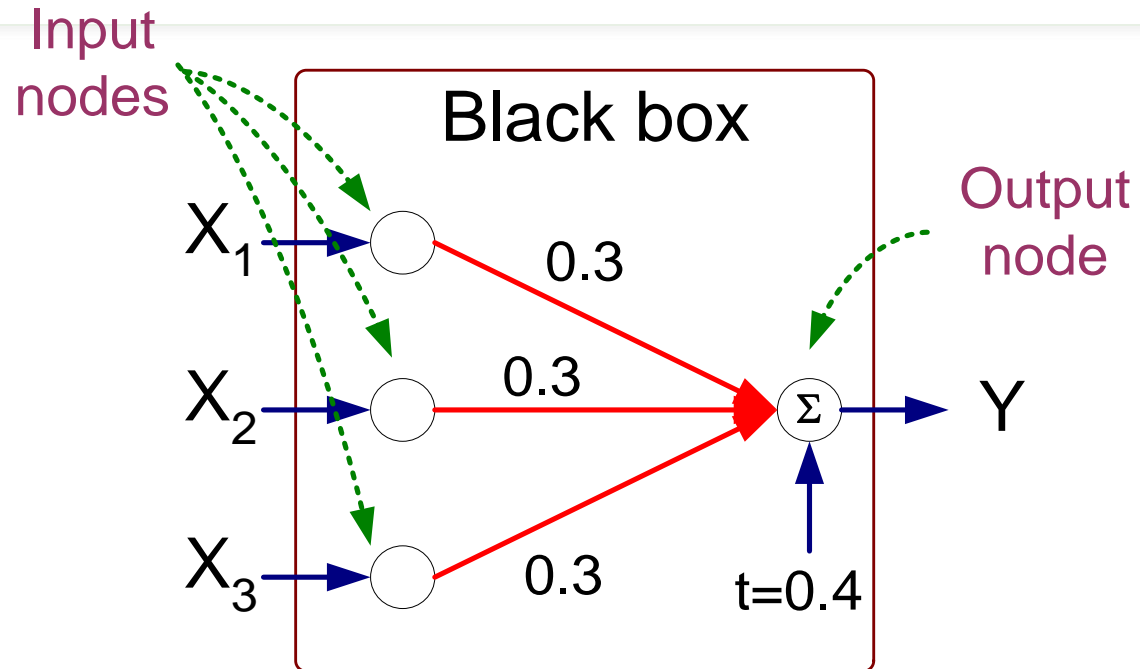
$X_1$	$X_2$	$X_3$	$Y$
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output  $Y$  is 1 if at least two of the three inputs are equal to 1.

# Artificial Neural Networks

$X_1$	$X_2$	$X_3$	$Y$
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

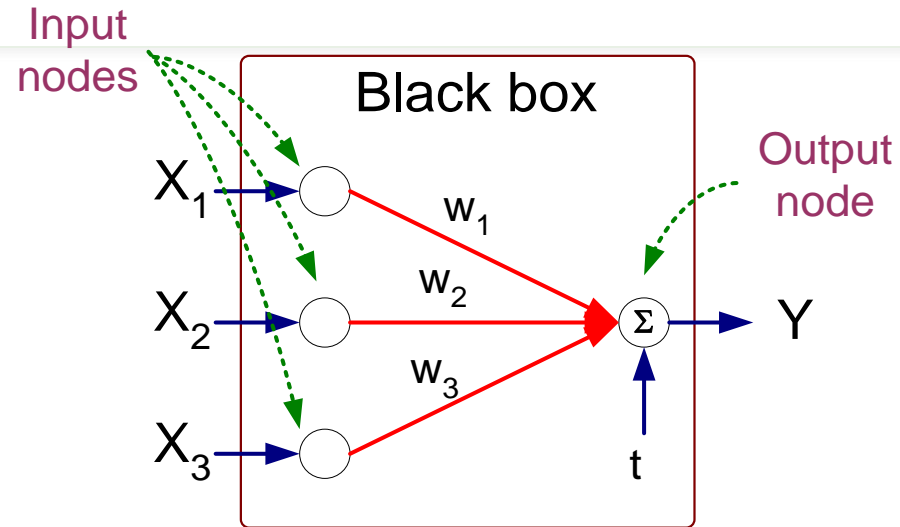


$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

# Artificial Neural Networks

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold  $t$

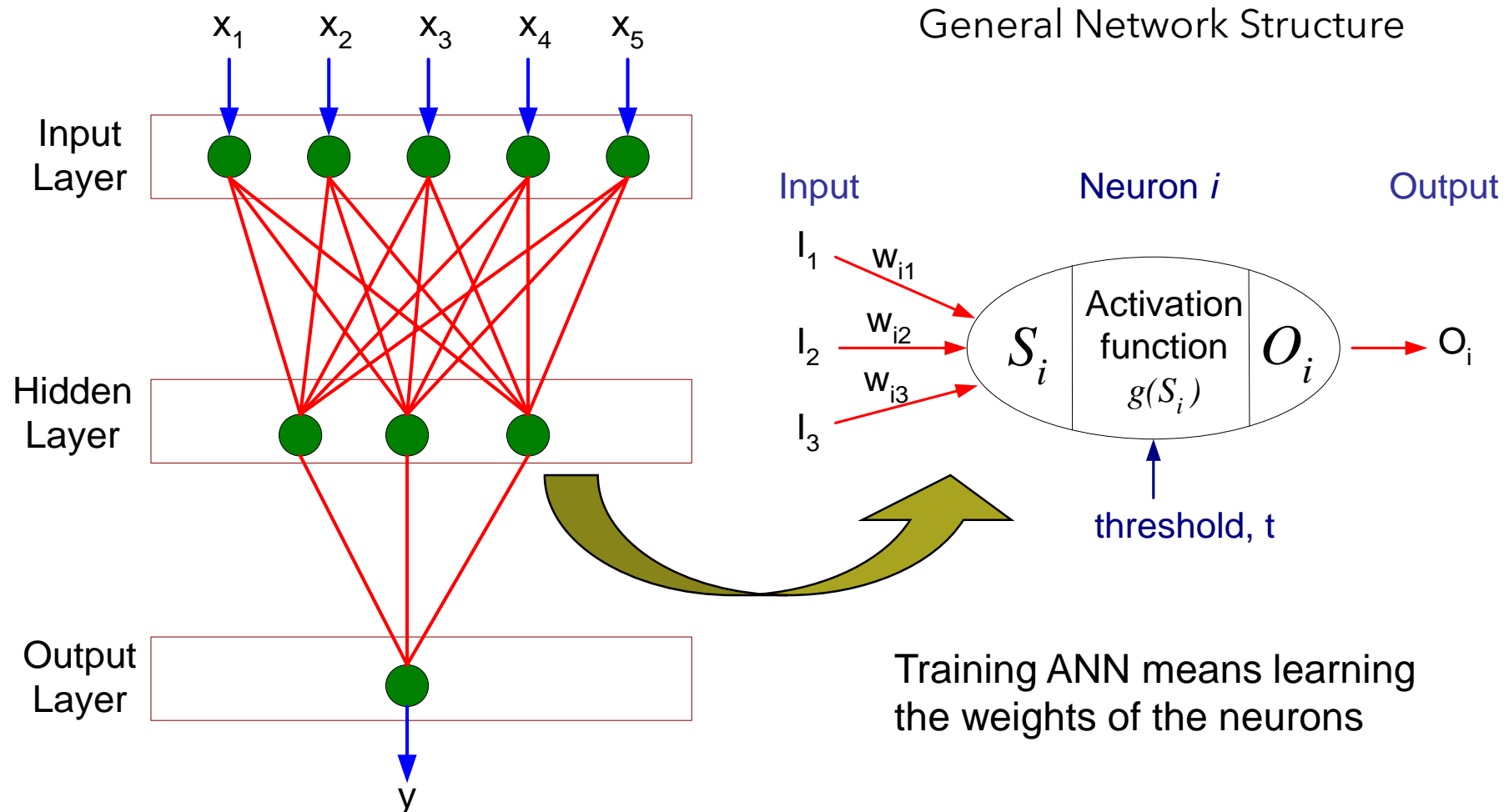


Perceptron Model

$$Y = \text{sign}\left(\sum_{i=1}^d w_i X_i - t\right)$$
$$= \text{sign}\left(\sum_{i=0}^d w_i X_i\right)$$



# Artificial Neural Networks

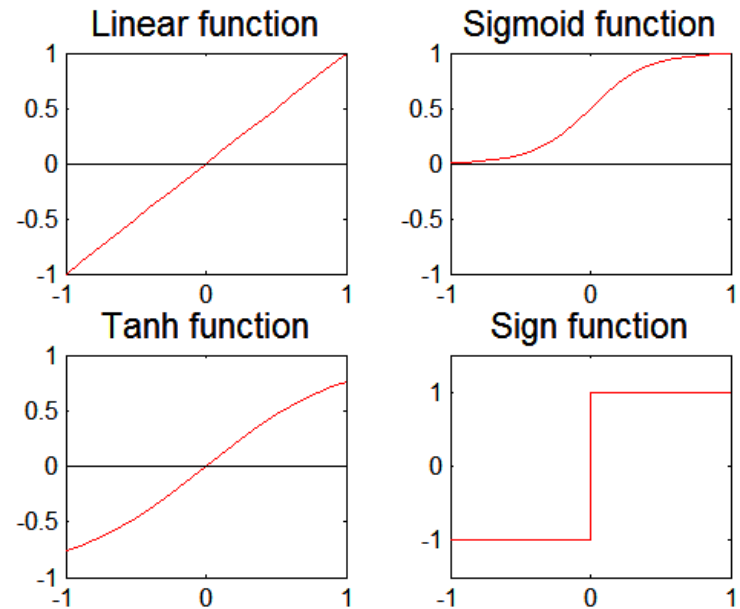


# Artificial Neural Networks

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f\left(\sum_i w_i X_i\right)$$



# Perceptron

- Single layer network
  - Contains only input and output nodes
- Activation function:  $f = \text{sign}(w \bullet x)$
- Applying model is straightforward

$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

- $X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = \text{sign}(0.2) = 1$

# Perceptron Learning Rule

- Initialize the weights ( $w_0, w_1, \dots, w_d$ )
- Repeat
  - For each training example ( $x_i, y_i$ )
    - Compute  $f(w, x_i)$
    - Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i$$

- Until stopping condition is met

# Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i ; \lambda : \text{learning rate}$$

- Intuition:

- Update weight based on error:  $e = [y_i - f(w^{(k)}, x_i)]$
- If  $y=f(x,w)$ ,  $e=0$ : no update needed
- If  $y>f(x,w)$ ,  $e=2$ : weight must be increased so that  $f(x,w)$  will increase
- If  $y<f(x,w)$ ,  $e=-2$ : weight must be decreased so that  $f(x,w)$  will decrease

# Example of Perceptron Learning

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i$$

$$Y = \text{sign}(\sum_{i=0}^d w_i X_i)$$

$$\lambda = 0.1$$

$X_1$	$X_2$	$X_3$	$Y$
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

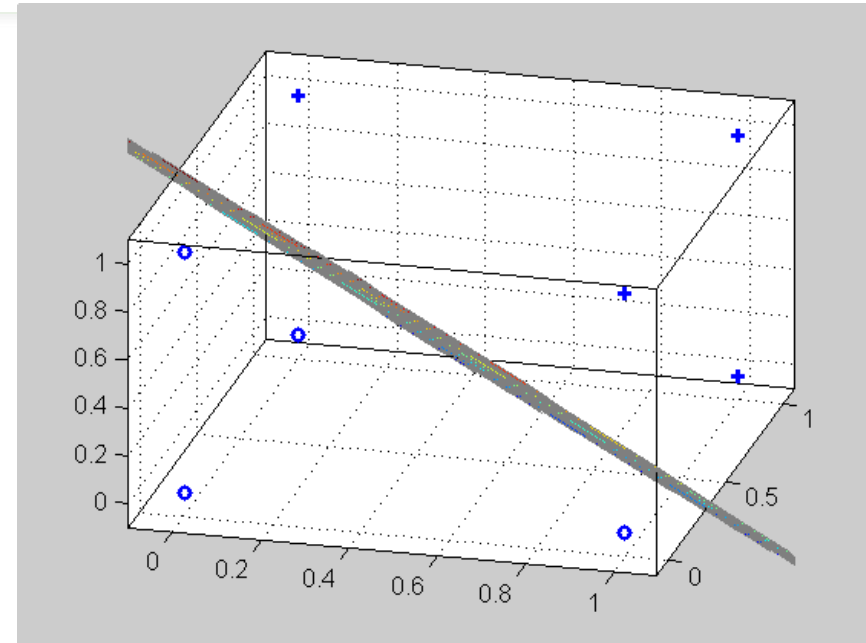
	$w_0$	$w_1$	$w_2$	$w_3$
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	$w_0$	$w_1$	$w_2$	$w_3$
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2



# Perceptron Learning Rule

- Since  $f(w, x)$  is a linear combination of input variables, decision boundary is linear



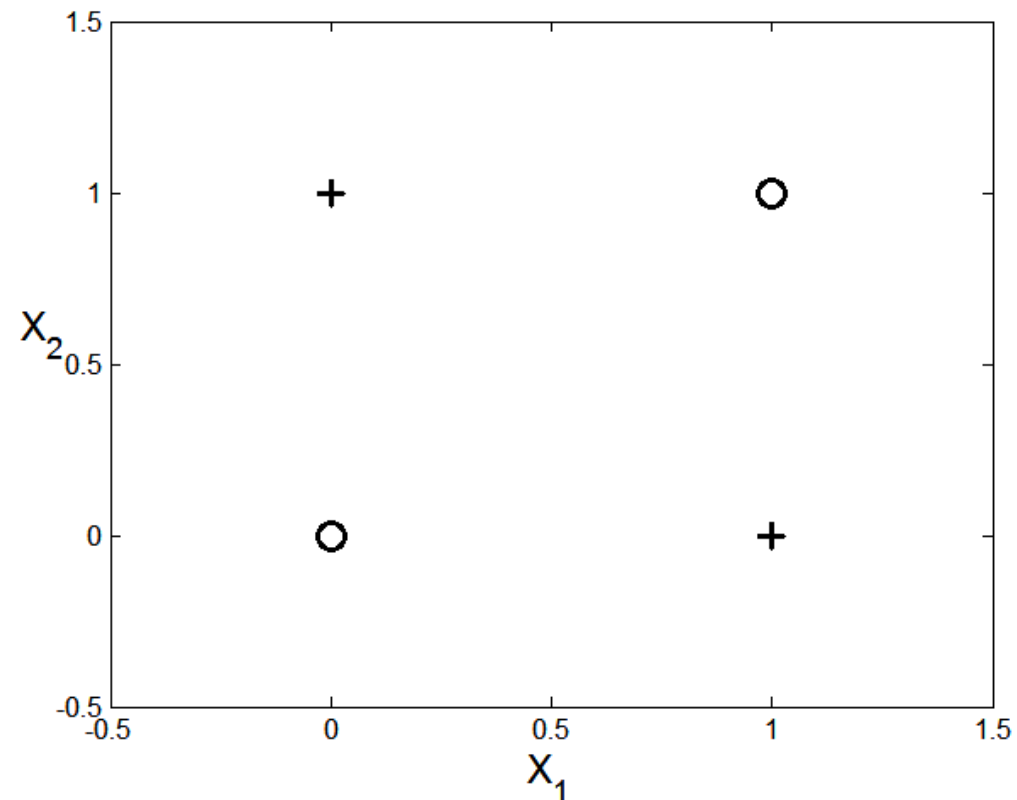
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

# Nonlinearly separable data

$$y = x_1 \oplus x_2$$

$x_1$	$x_2$	$y$
0	0	-1
1	0	1
0	1	1
1	1	-1

XOR Data

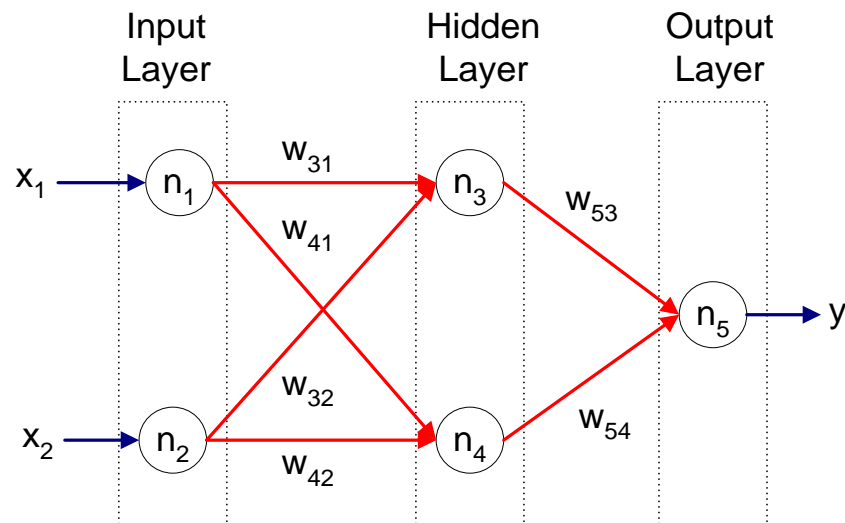


# Multilayer Perceptron

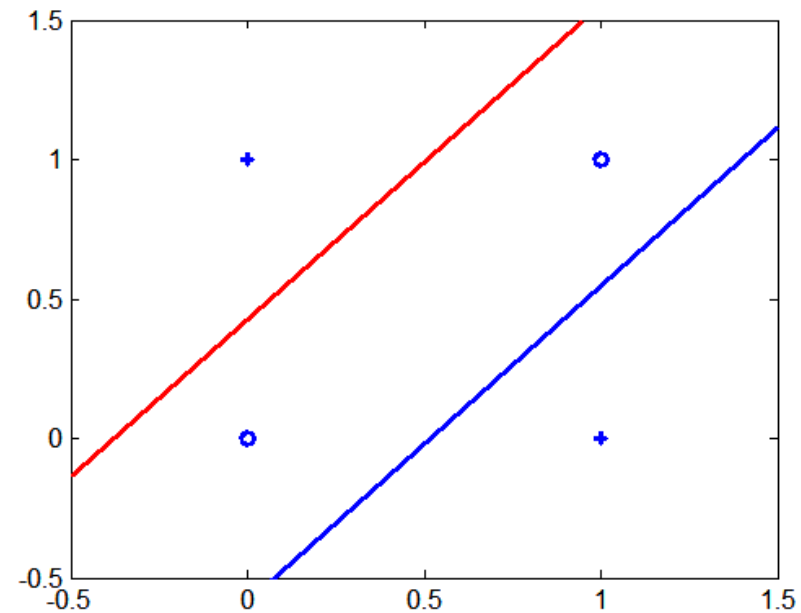
- Hidden layers
  - intermediary layers between input & output layers
- More general activation functions (sigmoid, linear, etc)

# Multilayer Perceptron

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



XOR Data



# Learning Multilayer Perceptron

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term  $e = y - f(w, x)$  and updates weights accordingly
    - Problem: how to determine the true value of  $y$  for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

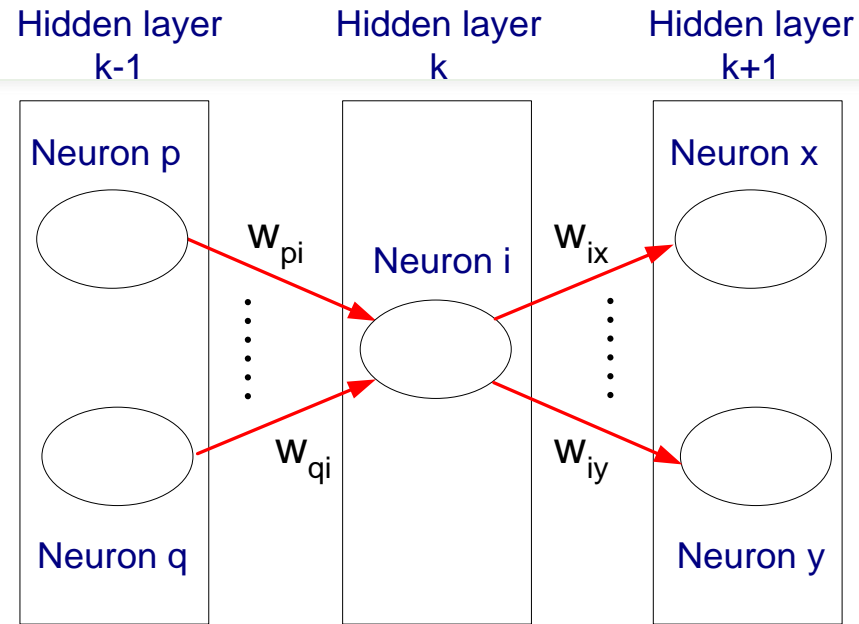
# Gradient Descent for Multilayer Perceptron

- Weight update:  $w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$
- Error function:  $E = \frac{1}{2} \sum_{i=1}^N \left( t_i - f\left(\sum_j w_j x_{ij}\right) \right)^2$
- Activation function  $f$  must be differentiable
- For sigmoid function:  $w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i (t_i - o_i) o_i (1 - o_i) x_{ij}$
- Stochastic gradient descent (update the weight immediately)



# Gradient Descent for Multilayer Perceptron

- For output neurons, weight update formula is the same as before (gradient descent for perceptron)



- For hidden neurons:

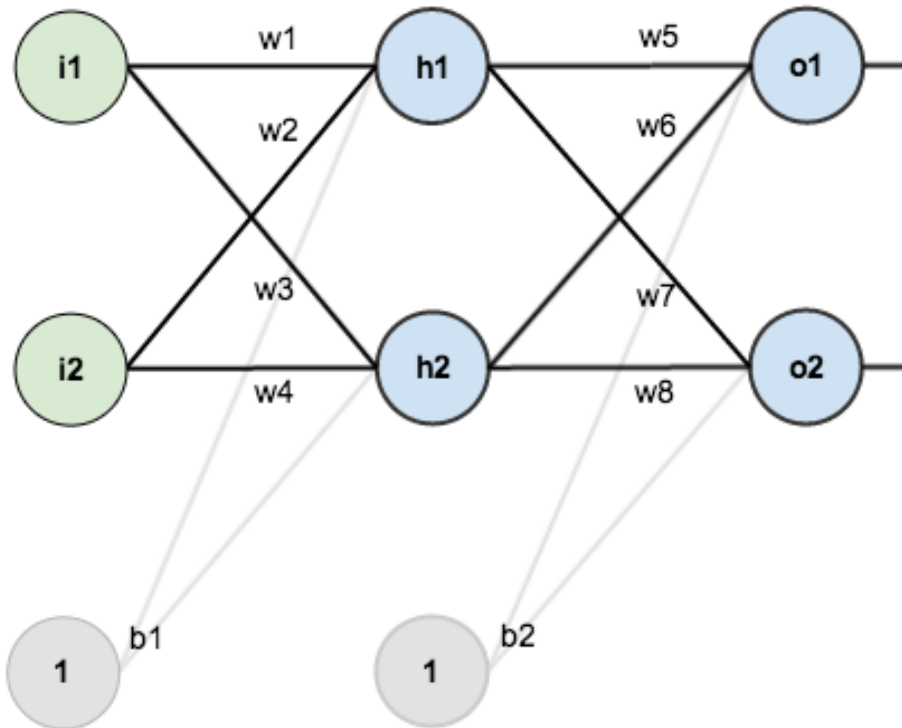
$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i (1 - o_i) \sum_{j \in \Phi_i} \delta_j w_{ij} x_{pi}$$

$$\text{Output neurons : } \delta_j = o_j (1 - o_j) (t_j - o_j)$$

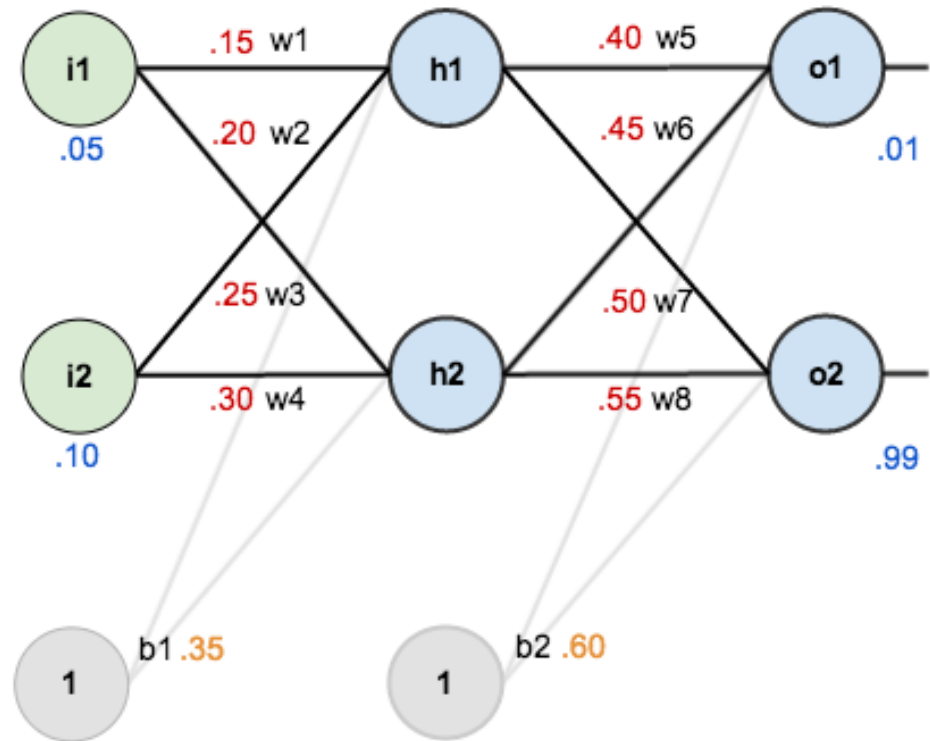
$$\text{Hidden neurons : } \delta_j = o_j (1 - o_j) \sum_{k \in \Phi_j} \delta_k w_{jk}$$

# Backpropagation

A nice example offered by Matt Mazur



Given inputs 0.05 and 0.1, make NN output 0.01 and 0.99



# Backpropagation

You want to make changes, first you need to check how it performs right now – Forward pass

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Activation function: Logistic

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

# Backpropagation

We next move to the output

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

Still activation function: Logistic

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

The forward pass completes

# Backpropagation

Computing error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

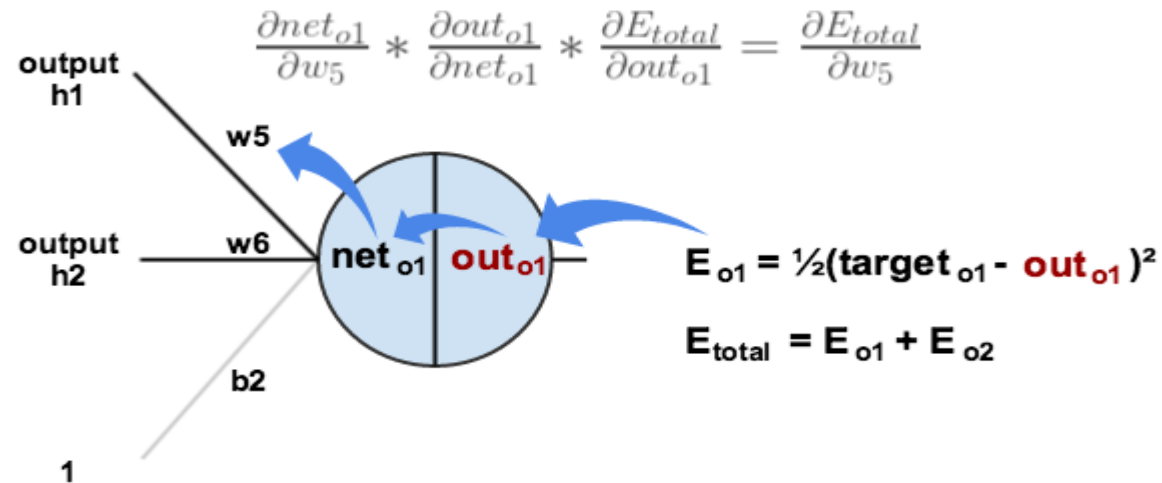
Now we know where we are!

# Backpropagation

Backwards pass - update weights of NN to make outputs closer to truths

Example: weight 5  $\frac{\partial E_{total}}{\partial w_5}$

A chain rule 
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$





# Backpropagation

Get the gradient step 1

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

# Backpropagation

Get the gradient step 2

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Get the gradient step 3

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

# Backpropagation

- Take them together

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

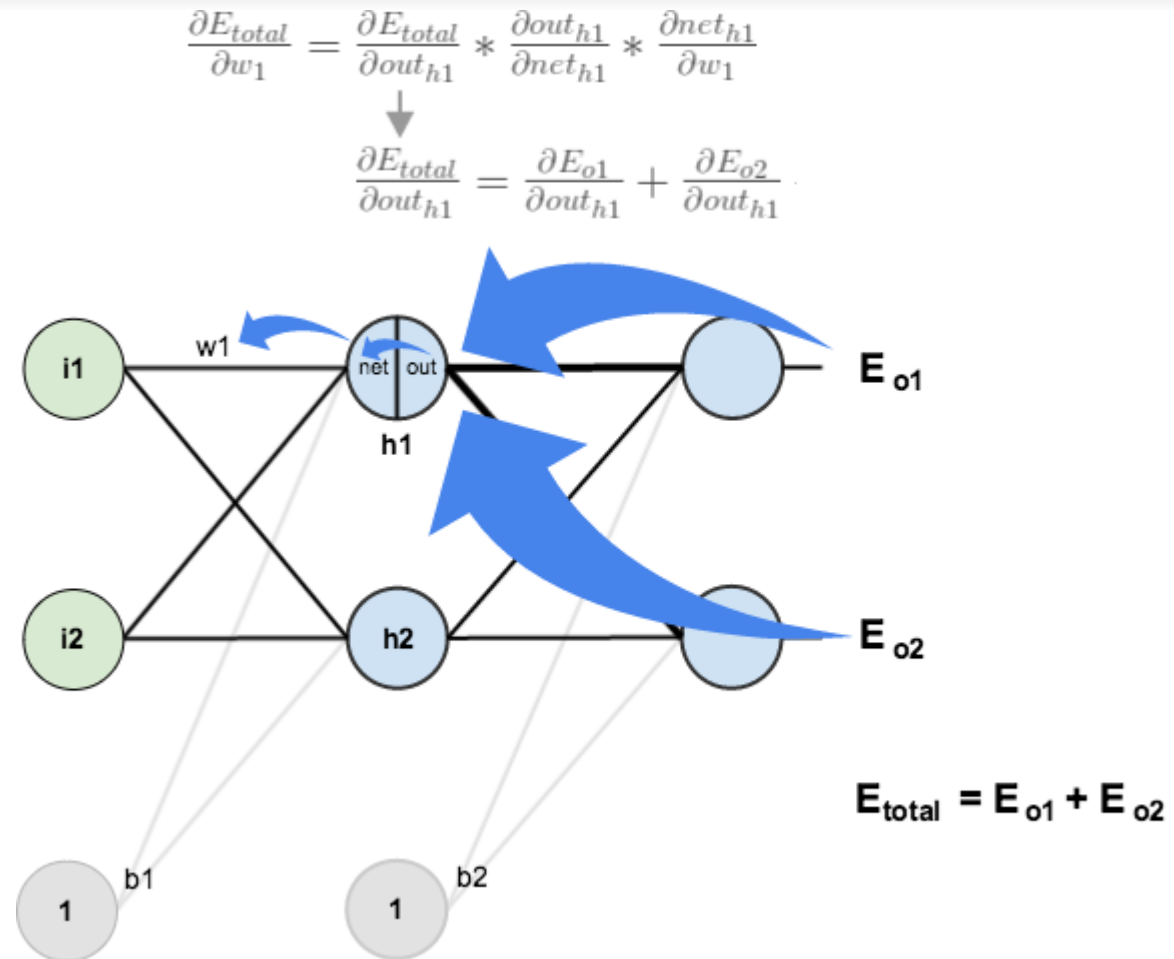
$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

The similar procedure can be applied to  $w_6$ ,  $w_7$ ,  $w_8$

# Backpropagation

- Hidden layer weight update



# Backpropagation

Starting with

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

# Backpropagation

- Similarly  $\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Next

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

# Backpropagation

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Put all together

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Similar process apply to  $w_2, w_3, w_4$

# Design Issues in ANN

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - $k$  or  $\log_2 k$  nodes for each categorical attribute with  $k$  values
- Number of nodes in output layer
  - One output for binary class problem
  - $k$  or  $\log_2 k$  nodes for  $k$ -class problem
- Number of nodes in hidden layer
- Initial weights and biases



# Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large
- Gradient descent may converge to local minimum
- Model building can be very time consuming, but testing can be very fast
- Can handle redundant attributes because weights are automatically learnt
- Sensitive to noise in training data
- Difficult to handle missing attributes



# SDSC 6004

---

Laboratory Section – Data Mining with Weka

---

# Why Weka?

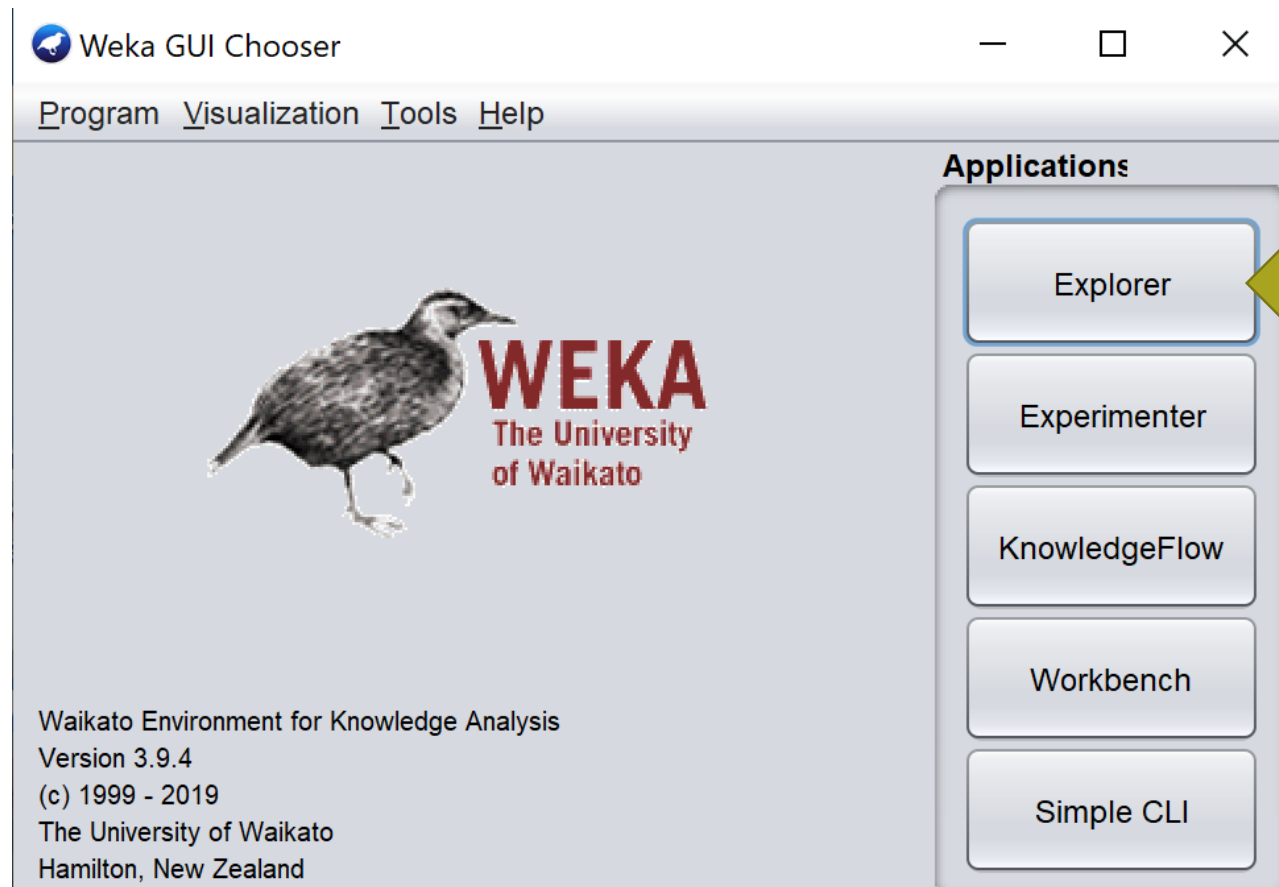
- A user-friendly free software
- It almost covers a full range of basic needs of data mining
- It offers classical algorithms in clustering, classification, and regression

# How to get Weka

Link: <https://www.cs.waikato.ac.nz/ml/weka/>

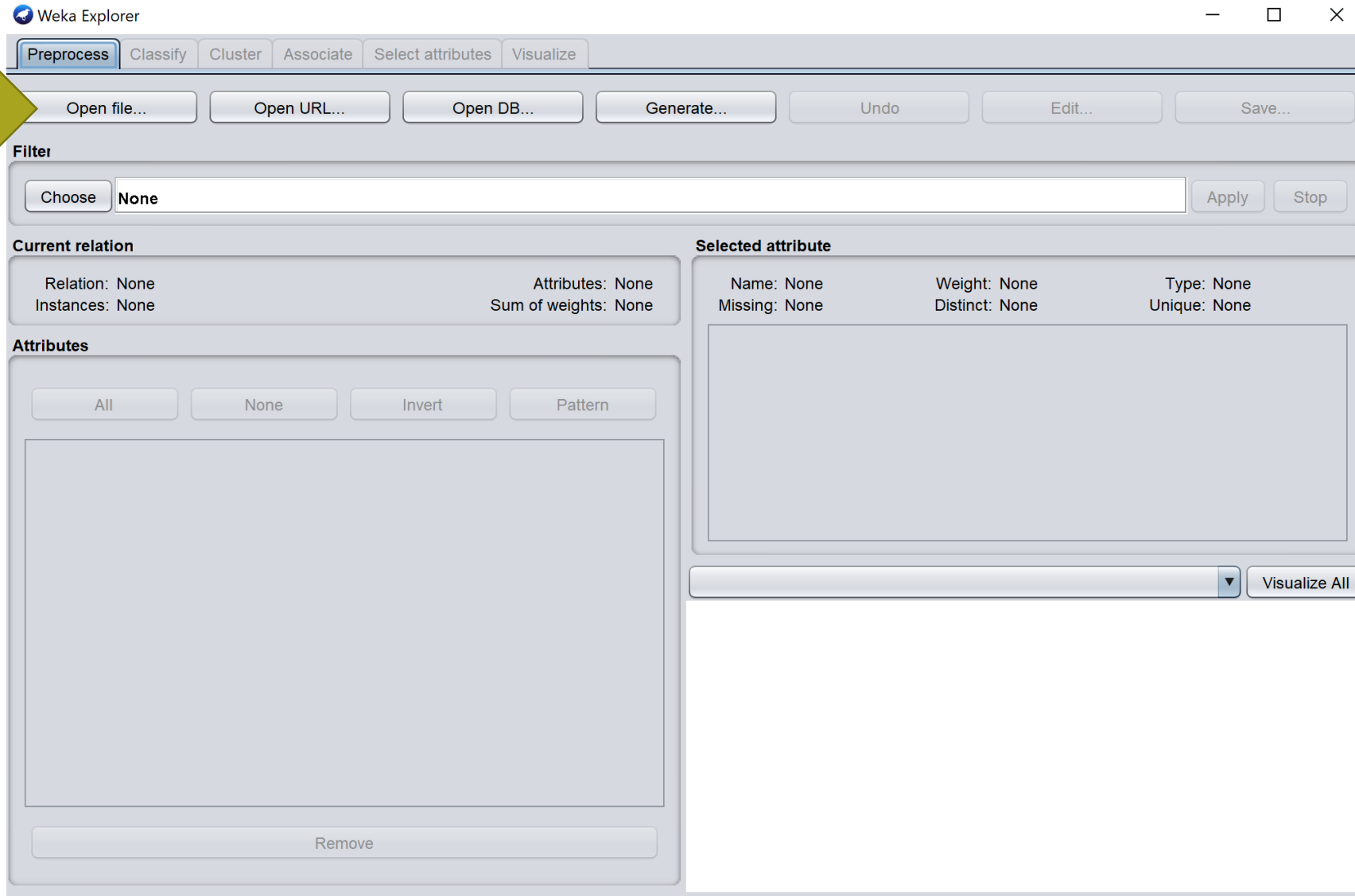
Select the correct version according to the specification of your operation systems, such as windows, Mac OS, Linux...

# Weka Interface



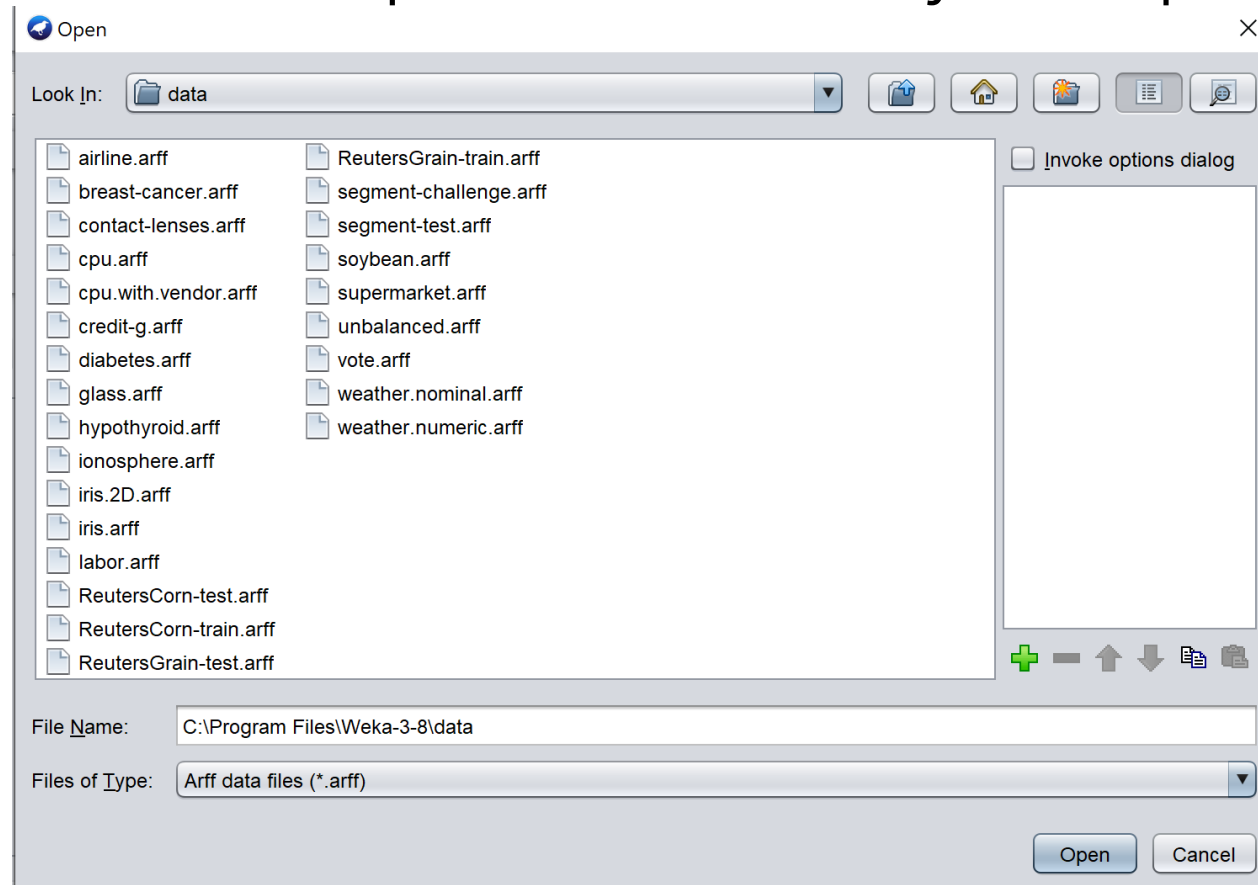
# Weka Interface

Loading  
data via  
Open File



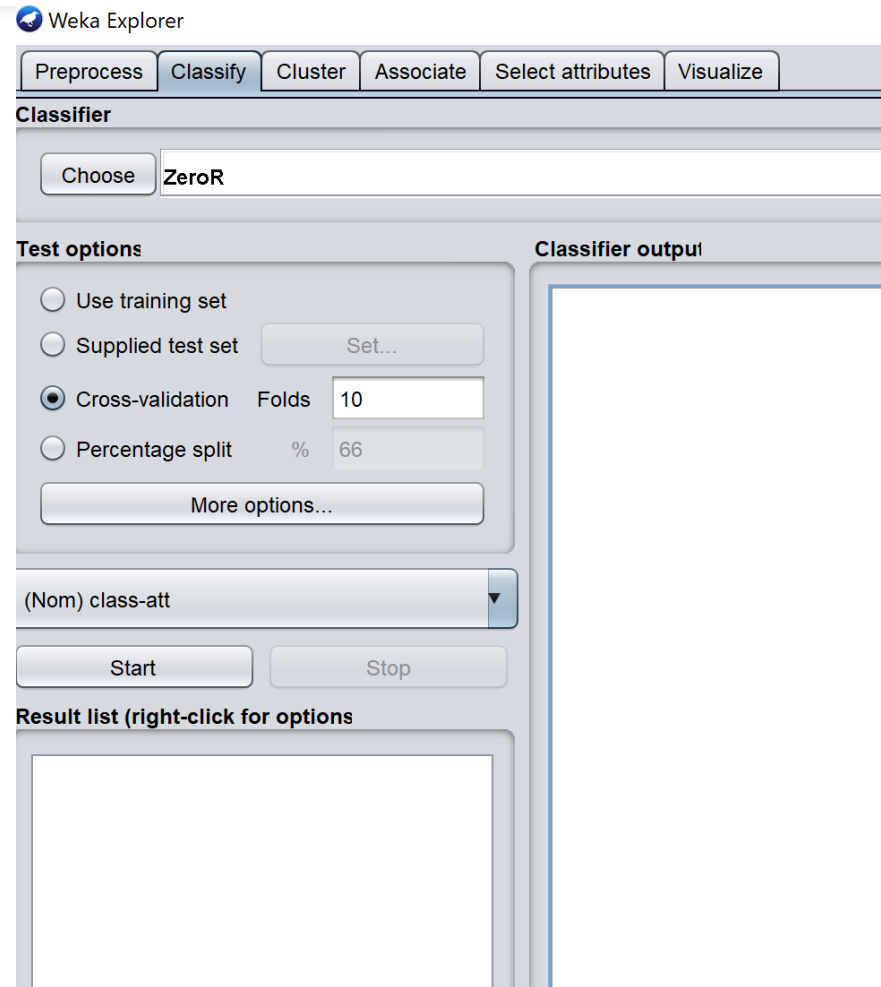
# Weka Default Datasets

Weka has some example datasets for you to practice:



# Weka Data Mining Options

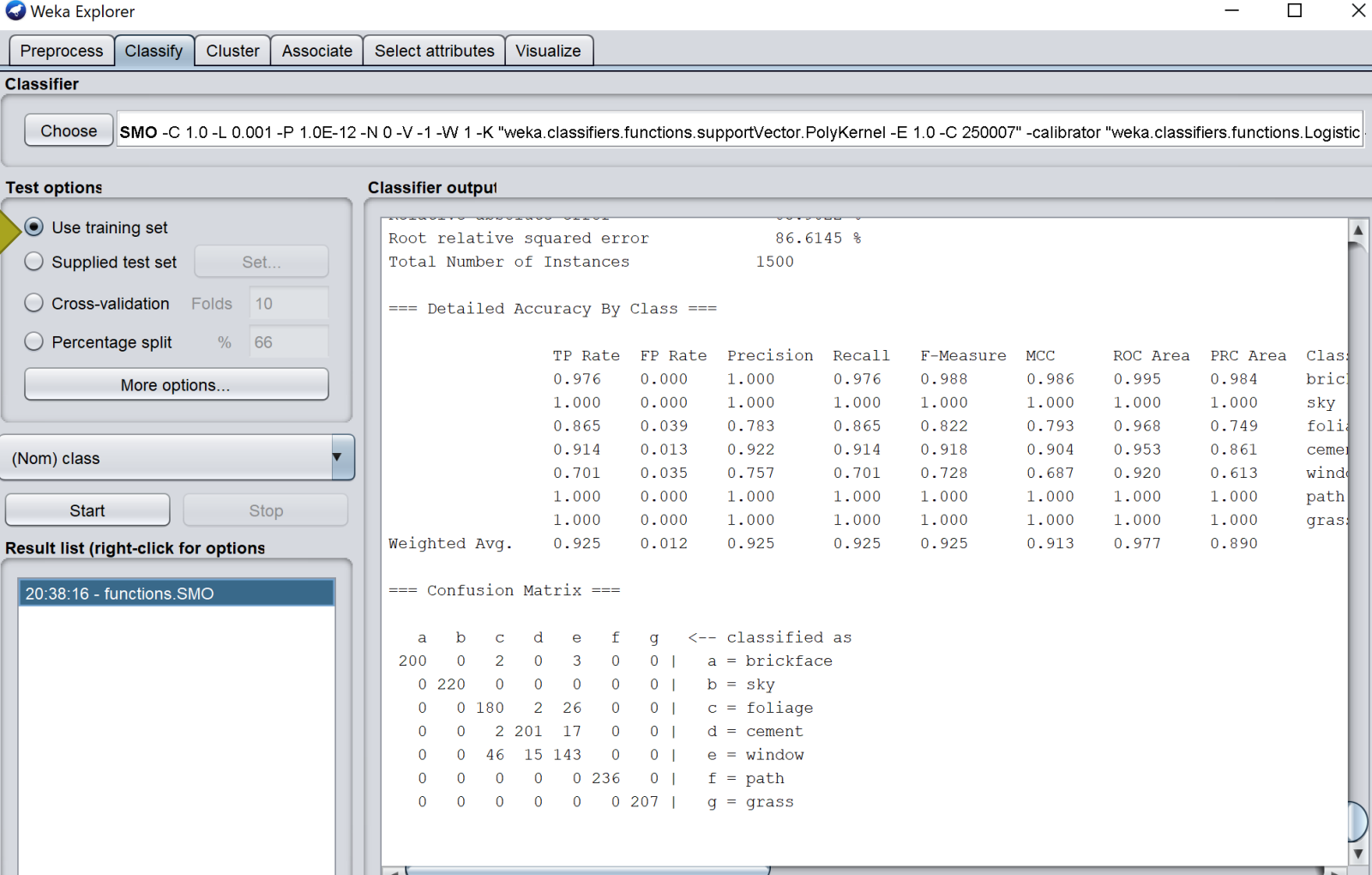
All classification and regression are done under the tag Classify





# Weka Data Mining Process

## Training



The Weka Explorer window shows the 'Classify' tab. The classifier selected is SMO with the following command: `SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic`

**Test options:**

- ☒ Use training set
- ☐ Supplied test set (Set...)
- ☐ Cross-validation (Folds: 10)
- ☐ Percentage split (%: 66)

**Classifier output:**

Root relative squared error: 86.6145 %  
Total Number of Instances: 1500

=== Detailed Accuracy By Class ===

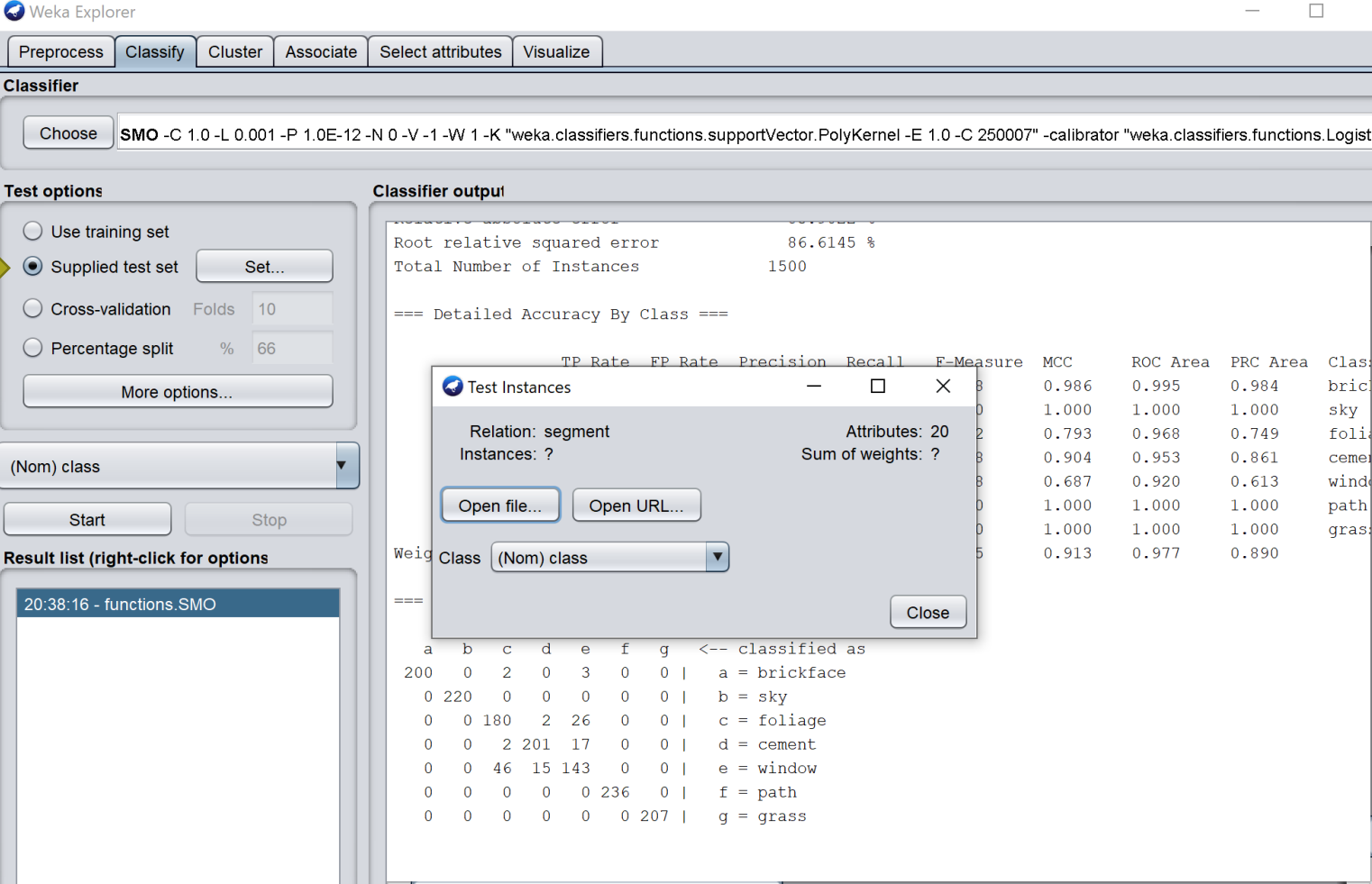
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.976	0.000	1.000	0.976	0.988	0.986	0.995	0.984	brick	
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	sky	
0.865	0.039	0.783	0.865	0.822	0.793	0.968	0.749	foliage	
0.914	0.013	0.922	0.914	0.918	0.904	0.953	0.861	cement	
0.701	0.035	0.757	0.701	0.728	0.687	0.920	0.613	window	
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	path	
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	grass	
Weighted Avg.	0.925	0.012	0.925	0.925	0.925	0.913	0.977	0.890	

=== Confusion Matrix ===

	a	b	c	d	e	f	g	<-- classified as
200	0	2	0	3	0	0	0	a = brickface
0	220	0	0	0	0	0	0	b = sky
0	0	180	2	26	0	0	0	c = foliage
0	0	2	201	17	0	0	0	d = cement
0	0	46	15	143	0	0	0	e = window
0	0	0	0	0	236	0	0	f = path
0	0	0	0	0	0	207	0	g = grass

# Weka Data Mining Process

## Testing



**Classifier**

Choose **SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic"**

**Test options**

☐ Use training set  
☒ Supplied test set **Set...**  
☐ Cross-validation Folds **10**  
☐ Percentage split % **66**  
**More options...**

**Classifier output**

Root relative squared error 86.6145 %  
Total Number of Instances 1500

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0	0.986	0.995	0.984	0.984	0.985	0.986	0.995	0.984	brick
1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	sky
2	0.793	0.968	0.749	0.793	0.771	0.793	0.968	0.749	foliage
3	0.904	0.953	0.861	0.904	0.882	0.904	0.953	0.861	cement
4	0.687	0.920	0.613	0.687	0.648	0.687	0.920	0.613	window
5	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	path
6	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	grass
7	0.913	0.977	0.890	0.913	0.901	0.913	0.977	0.890	grass

**Test Instances**

Relation: segment  
Instances: ?  
Attributes: 20  
Sum of weights: ?

**Open file...** **Open URL...**

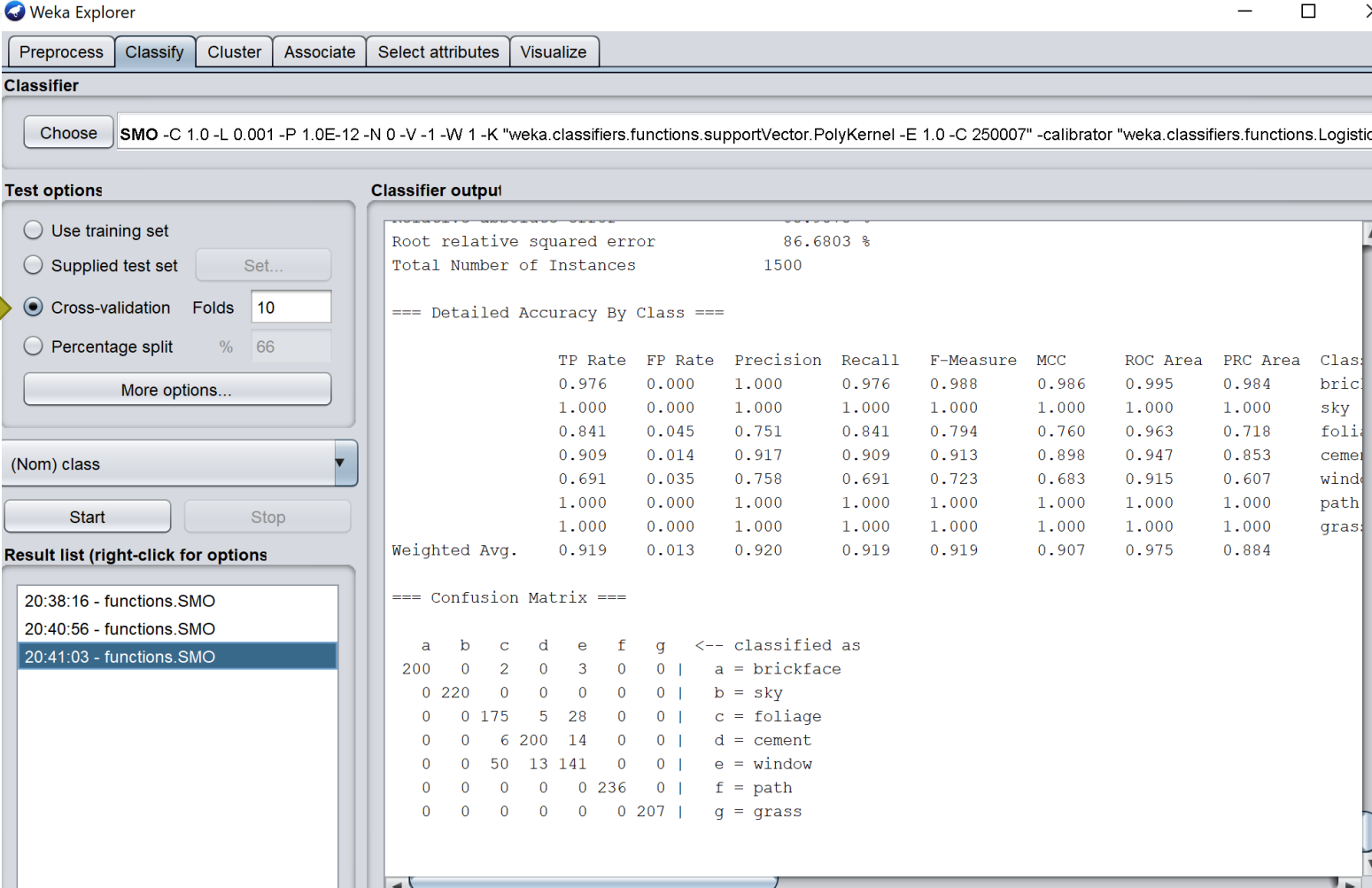
**Close**

**Result list (right-click for options)**

20:38:16 - functions.SMO

# Weka Data Mining Process

10-fold cross  
validation



The image shows the Weka Explorer interface with the 'Classify' tab selected. The classifier chosen is SMO with parameters: -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic".

**Test options:**

- ☐ Use training set
- ☐ Supplied test set (Set...)
- ☒ Cross-validation (Folds: 10)
- ☐ Percentage split (%: 66)

**Classifier output:**

Root relative squared error: 86.6803 %  
Total Number of Instances: 1500

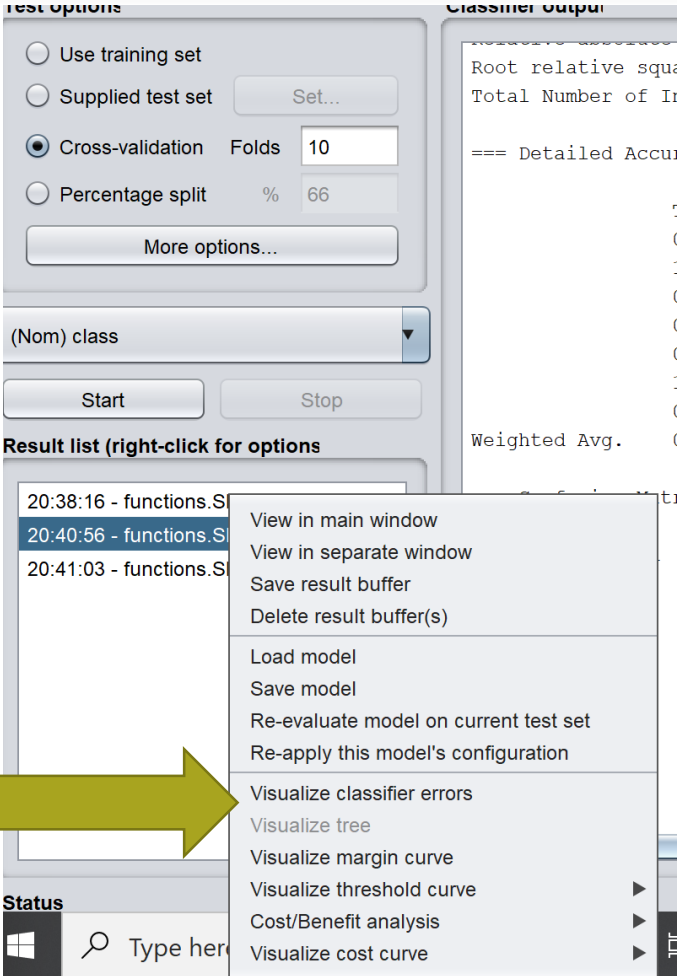
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.976	0.000	1.000	0.976	0.988	0.986	0.995	0.984	brick
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	sky
	0.841	0.045	0.751	0.841	0.794	0.760	0.963	0.718	foliage
	0.909	0.014	0.917	0.909	0.913	0.898	0.947	0.853	cement
	0.691	0.035	0.758	0.691	0.723	0.683	0.915	0.607	window
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	path
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	grass
Weighted Avg.	0.919	0.013	0.920	0.919	0.919	0.907	0.975	0.884	

=== Confusion Matrix ===

	a	b	c	d	e	f	g	<-- classified as
200	0	2	0	3	0	0	0	a = brickface
0	220	0	0	0	0	0	0	b = sky
0	0	175	5	28	0	0	0	c = foliage
0	0	6	200	14	0	0	0	d = cement
0	0	50	13	141	0	0	0	e = window
0	0	0	0	0	236	0	0	f = path
0	0	0	0	0	0	0	207	g = grass

# Weka Data Mining Results



The screenshot shows the 'Test Options' dialog box in Weka. The 'Cross-validation' option is selected with 10 folds. Below the dialog, the 'Result list' is visible, showing three entries: '20:38:16 - functions.S...', '20:40:56 - functions.S...', and '20:41:03 - functions.S...'. A right-click context menu is open over the '20:40:56' entry, listing various actions such as 'View in main window', 'Visualize classifier errors', and 'Visualize tree'. A large green arrow points from the left towards the context menu.

**Test Options**

- ☐ Use training set
- ☐ Supplied test set (Set...)
- ☒ Cross-validation Folds: 10
- ☐ Percentage split %: 66
- More options...

(Nom) class

Start Stop

**Result list (right-click for options)**

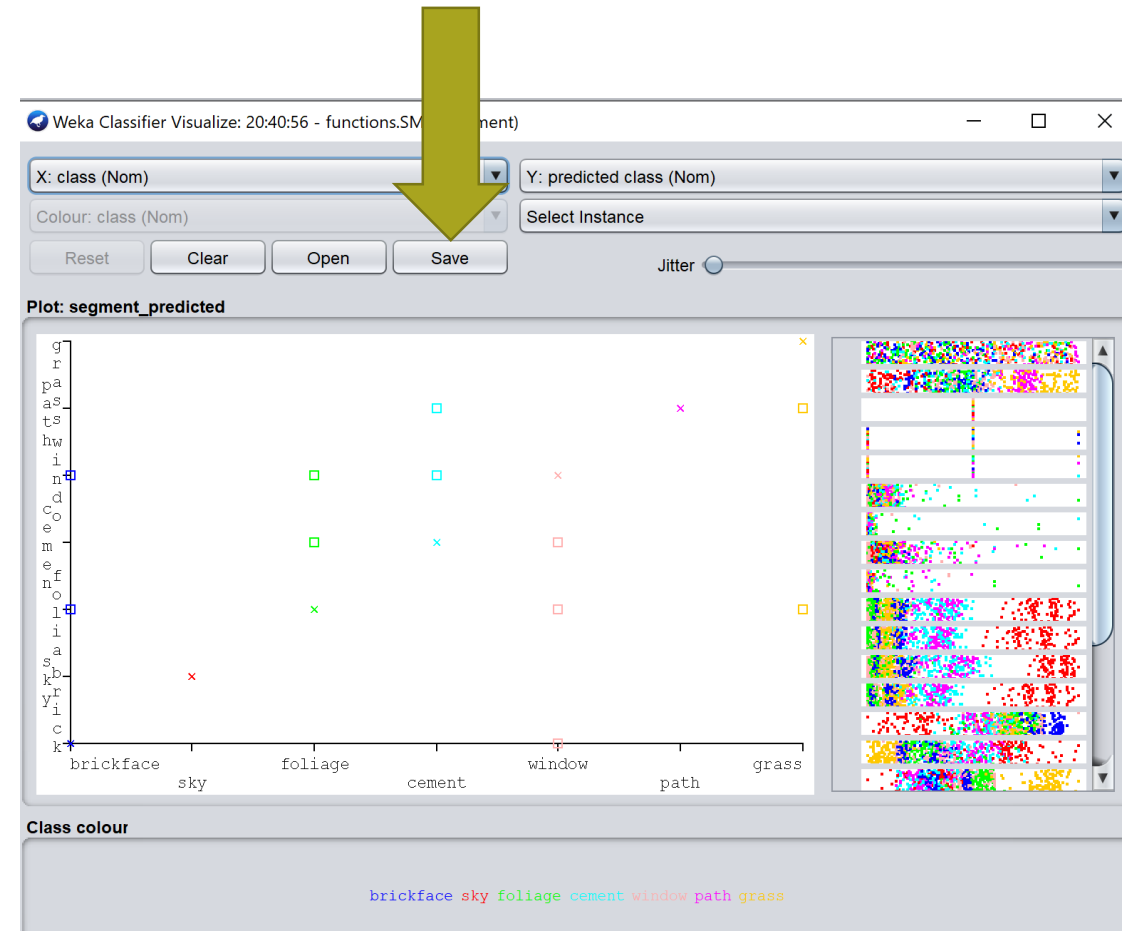
- 20:38:16 - functions.S...
- 20:40:56 - functions.S...
- 20:41:03 - functions.S...

**Context Menu:**

- View in main window
- View in separate window
- Save result buffer
- Delete result buffer(s)
- Load model
- Save model
- Re-evaluate model on current test set
- Re-apply this model's configuration
- Visualize classifier errors
- Visualize tree
- Visualize margin curve
- Visualize threshold curve
- Cost/Benefit analysis
- Visualize cost curve

**Status**

Type here





# Weka Data Mining

- Clustering...
- Feature selection...

# Weka Data Mining

Q&A