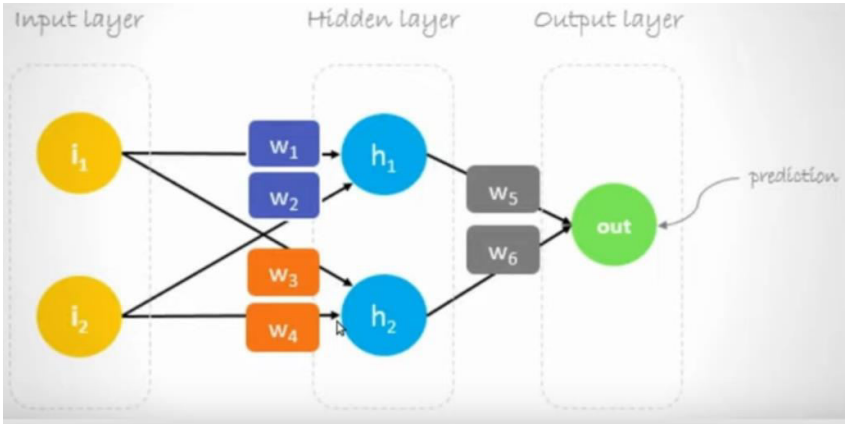
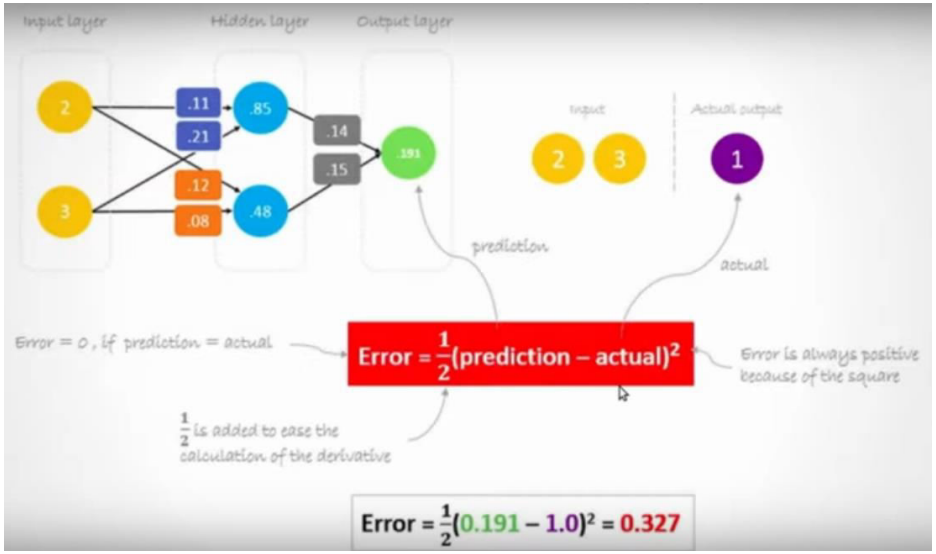
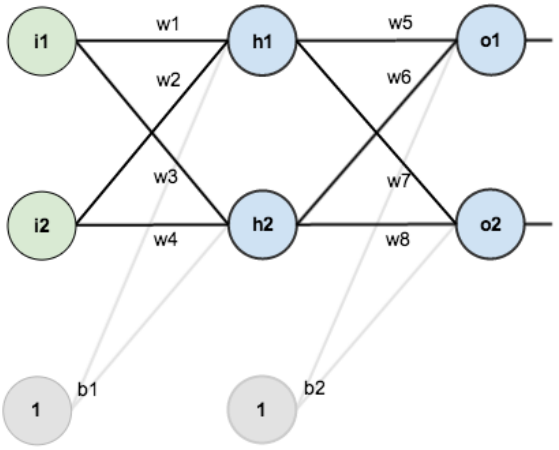
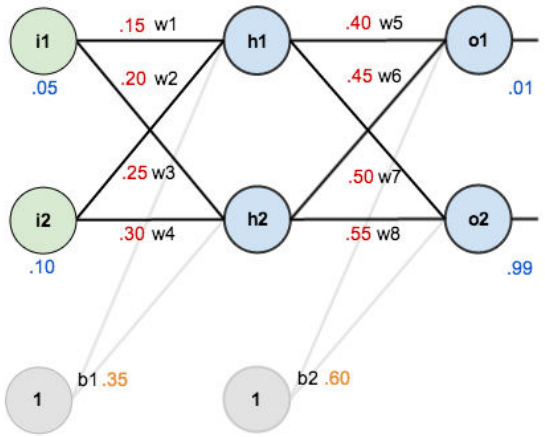
Backpropagation Example





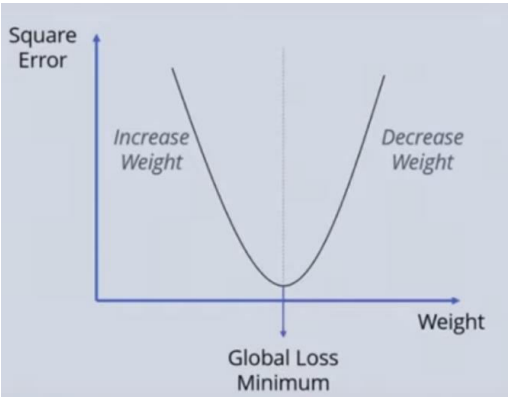


initial weights, the biases, and training inputs/outputs:

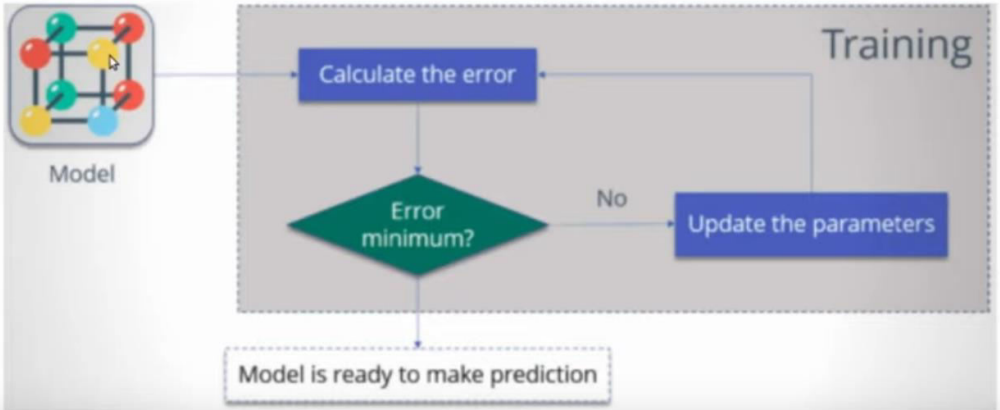


GRADIENT DESCENT

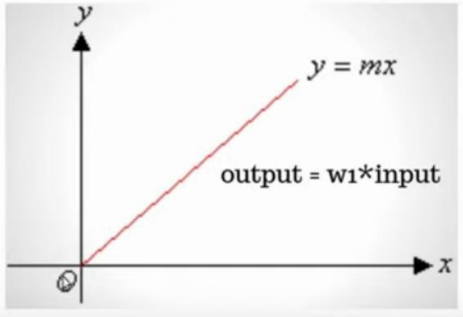
- Update the weights using gradient descent.
- Gradient descent is used for finding the minimum of a function.
- In our case we want to minimize the error function.

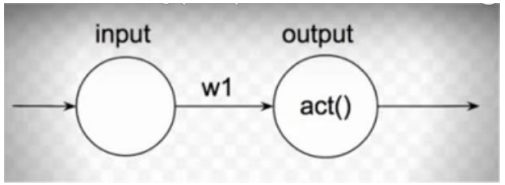


OVERVIEW:

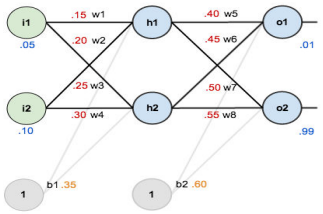


- y = w1 * x
- Output of bias is **w_b**.
- w_b = 1 * b1





The Forward Pass



calculate the total net input for h1

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

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

output of h1:

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

$$out_{h2} = 0.596884378$$

i1=0.05, i2=0.10

w1=0.15, w2=0.20
w3=0.25, w4=0.30
b1=0.3

w5=0.4. w6=0.45
w7=0.5, w8=0.55
b2=0.6

o1=0.01, o2=0.99

repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

output for O1:

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

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

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

$$out_{o2} = 0.772928465$$

now calculate the error for each output neuron using the squared error function and sum them to get the total error:

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

For example, the target output for O1 is 0.01 but the neural network output 0.75136507, therefore its error is:

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

Repeating this process for O2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

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

The Backwards Pass

- Update the weights using gradient descent.
- So that they cause the predicted output to be closer to actual output.
- Thereby minimizing the error for each output neuron and the network as a whole.

The Backwards Pass

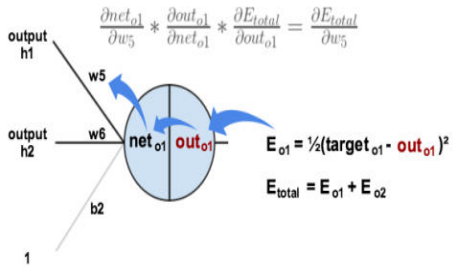
Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

By applying the chain rule we know that:

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



First, how much does the total error change with respect to the output?

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

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

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

$-(target - out)$ is sometimes expressed as $out - target$

When we take the partial derivative of the total error with respect to out_{o1} the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative of a constant which is zero.

Next, how much does the output of $o1$ change with respect to its total net input?

The partial derivative of the logistic function is the output multiplied by 1 minus the output:

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

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

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x},$$
$$\frac{d}{dx}f(x) = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = f(x)(1 - f(x)).$$

Finally, how much does the total net input of $o1$ change with respect to w_5

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

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

Putting it all together:

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

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

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

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

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

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

We can repeat this process to get the new weights w_6, w_7, and w_8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

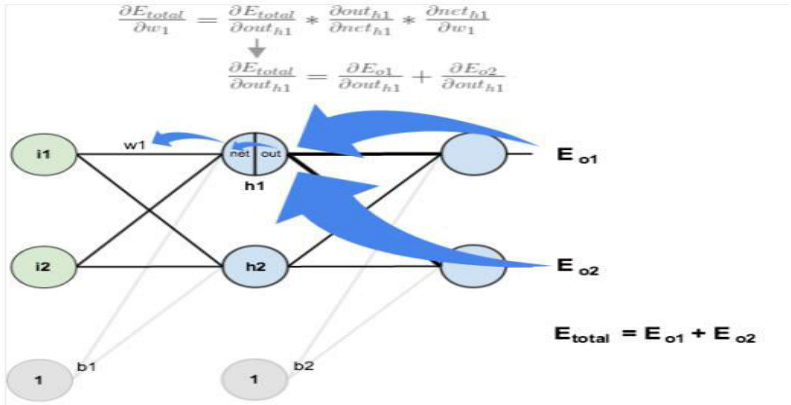
We perform the actual updates in the neural network after we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4.

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

Visually:



We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

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

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

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

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

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

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

Plugging them in:

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

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

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

Therefore:

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

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

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

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

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

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

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

Putting it all together:

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

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

You might also see this written as:

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \delta_o * w_{ho}) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

We can now update w_1:

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

Repeating this for w_2, w_3, and w_4

$w_2^+ = 0.19956143$

$w_3^+ = 0.24975114$

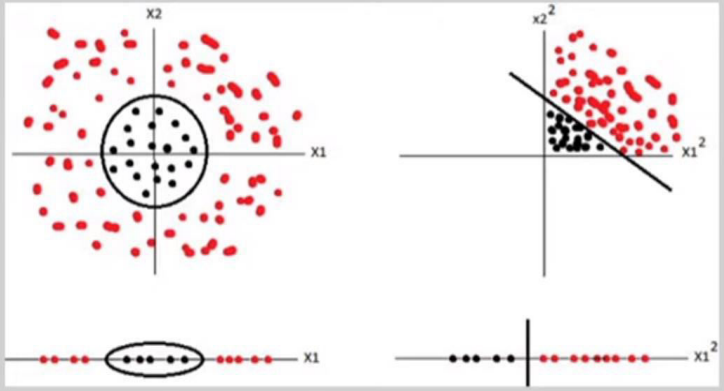
$w_4^+ = 0.29950229$

- Finally, we've updated all of our weights!
- When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109.
- After this first round of backpropagation, the total error is now down to 0.291027924.
- It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085.

Support Vector Machines (SVM): Non-Linear, Kernel Functions

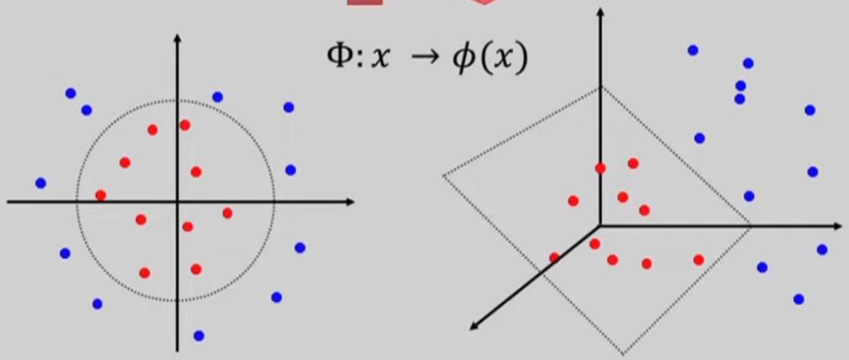
Non-linear SVMs: Feature Space

$\Phi: x \rightarrow \phi(x)$



Non-linear SVMs: Feature Space

$\Phi: x \rightarrow \phi(x)$



This slide is from *www.iro.umontreal.ca/~pift6080/documents/papers/svm_tutorial.ppt*

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- We only use the dot product of feature vectors in both the training and test.
- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(x_a, x_b) = \phi(x_a).\phi(x_b)$$

Kernel Example

2-dimensional vectors $\bar{x} = [x_1 x_2]$

let $K(x_i, x_j) = (1 + x_i.x_j)^2$

We need to show that $K(x_i, x_j) = \phi(x_i).\phi(x_j)$

$K(x_i,x_j) = (1 + x_ix_j)^2$,

$= 1 + x_{i1}^2x_{j1}^2 + 2\,x_{i1}x_{j1}\,x_{i2}x_{j2} + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$

$= [1\ \ x_{i1}^2\ \sqrt{2}\,x_{i1}x_{i2}\ \ x_{i2}^2\ \sqrt{2}x_{i1}\ \sqrt{2}x_{i2}].[1\ \ x_{j1}^2\ \sqrt{2}\,x_{j1}x_{j2}\ \ x_{j2}^2\ \sqrt{2}x_{j1}\ \sqrt{2}x_{j2}]$

$= \phi(x_i).\phi(x_j)$,

where $\phi(x) = [1\ \ x_1^2\ \sqrt{2}\,x_1x_2\ \ x_2^2\ \sqrt{2}x_1\ \sqrt{2}x_2]$

Commonly-used kernel functions

- Linear kernel: $K(x_i, x_j) = x_i.x_j$
- Polynomial of power p:

$$K(x_i, x_j) = (1 + x_i.x_j)^p$$

- Gaussian (radial-basis function):

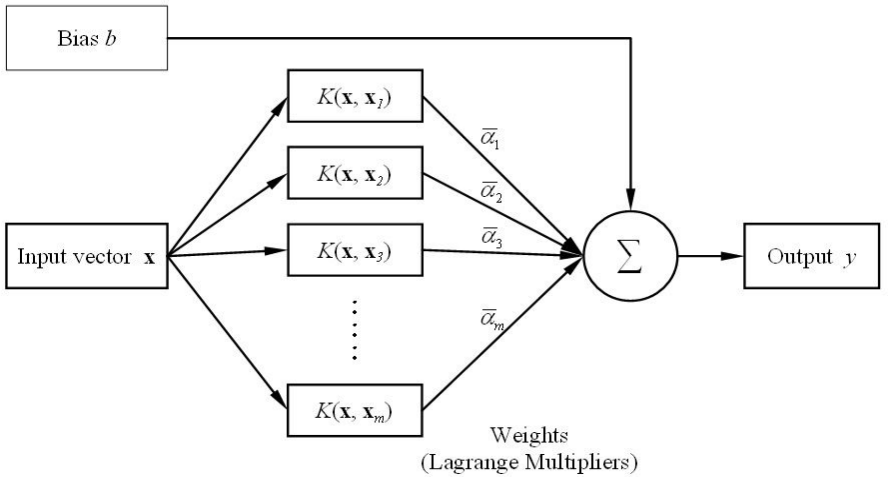
$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

- Sigmoid

$$K(x_i, x_j) = \tanh(\beta_0 x_i.x_j + \beta_1)$$

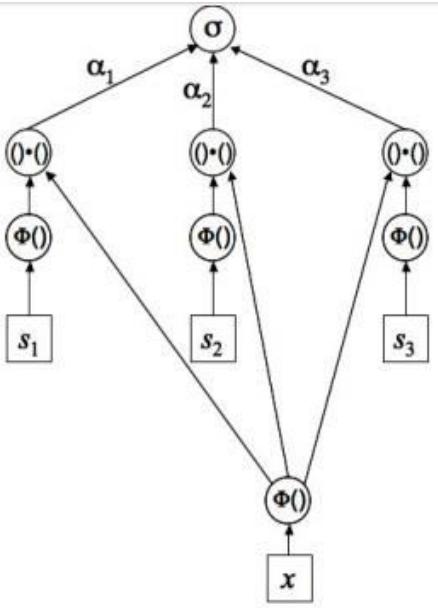
In general, functions that satisfy *Mercer's condition* can be kernel functions.

Architecture



$$y = f(\mathbf{x}) = \sum_{k=1}^{m} \alpha_k \cdot K(\mathbf{x}, \mathbf{x}_k) + b$$

SVM Architecture



$$y = f(\mathbf{x}) = \sum_{k=1}^{m} \overline{\alpha}_k \cdot K(\mathbf{x}, \mathbf{x}_k) + b$$

SVM-Example

Nonlinear function

SVM – Linearly Inseparable data - Example

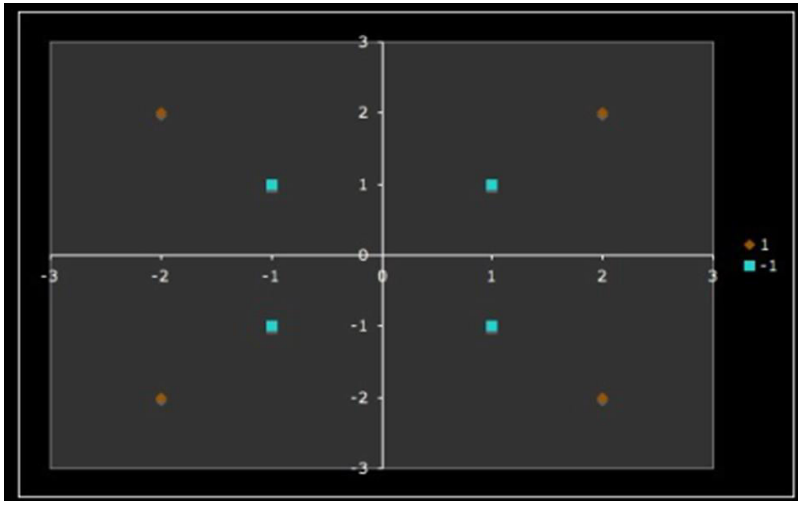
Now suppose instead that we are given the following positively labeled data points in $\Re^2$:

$$\left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 2 \end{pmatrix} \right\}$$

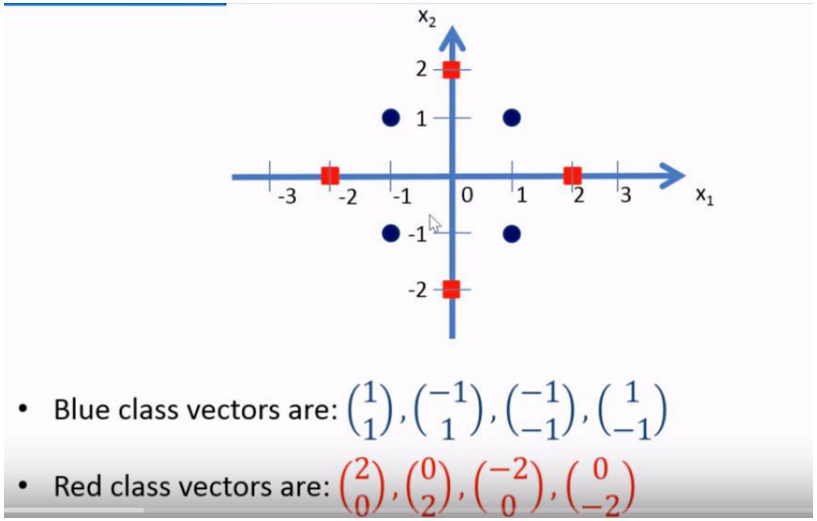
and the following negatively labeled data points in $\Re^2$ (see Figure 5):

$$\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\}$$

Nonlinearly separable sample data points - Example



Example-2



- Blue class vectors are: $\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}, \begin{pmatrix}-1\\-1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix}$
- Red class vectors are: $\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}-2\\0\end{pmatrix}, \begin{pmatrix}0\\-2\end{pmatrix}$

- Here we need to find a non-linear mapping function Φ which can transform these data in to a new feature space where a seperating hyperplane can be found.
- Let us consider the following mapping function.

- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{cases}\begin{pmatrix}6-x_1+(x_1-x_2)^2\\6-x_2+(x_1-x_2)^2\end{pmatrix} & \text{if } \sqrt{x_1^2+x_2^2} \geq 2\\ \begin{pmatrix}x_1\\x_2\end{pmatrix} & \text{otherwise}\end{cases}$

- Now let us transform the blue and red calss vectors using the non-linear mapping function Φ .

- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{cases}\begin{pmatrix}6-x_1+(x_1-x_2)^2\\6-x_2+(x_1-x_2)^2\end{pmatrix} & \text{if } \sqrt{x_1^2+x_2^2} \geq 2\\ \begin{pmatrix}x_1\\x_2\end{pmatrix} & \text{otherwise}\end{cases}$

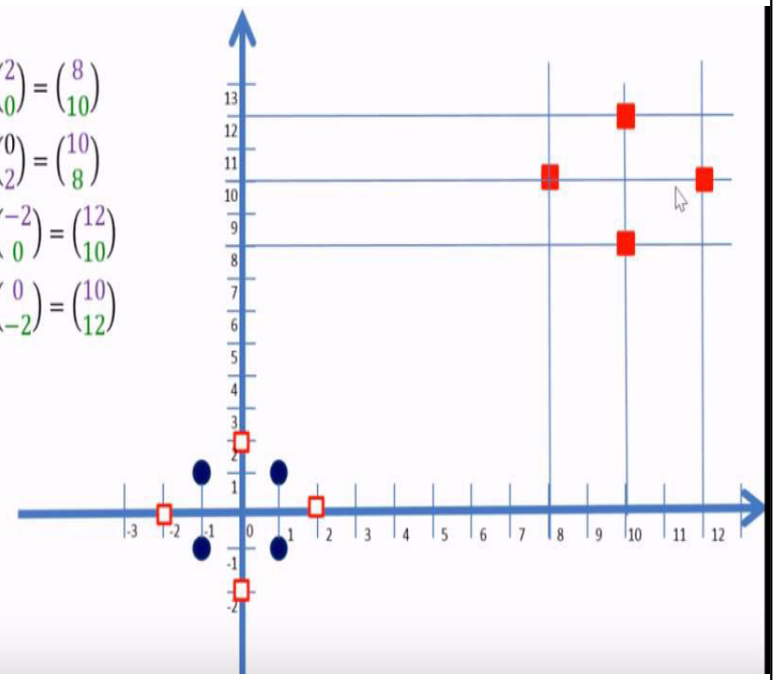
- Blue class vectors are: $\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}, \begin{pmatrix}-1\\-1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix}$ no change since $\sqrt{x_1^2+x_2^2} < 2$ for all the vectors

- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{cases}\begin{pmatrix}6-x_1+(x_1-x_2)^2\\6-x_2+(x_1-x_2)^2\end{pmatrix} & \text{if } \sqrt{x_1^2+x_2^2} \geq 2\\ \begin{pmatrix}x_1\\x_2\end{pmatrix} & \text{otherwise}\end{cases}$

- Let us take Red class vectors : $\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}-2\\0\end{pmatrix}, \begin{pmatrix}0\\-2\end{pmatrix}$

- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \Phi\begin{pmatrix}2\\0\end{pmatrix} = \begin{pmatrix}6-2+(2-0)^2\\6-0+(2-0)^2\end{pmatrix} = \begin{pmatrix}8\\10\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \Phi\begin{pmatrix}0\\2\end{pmatrix} = \begin{pmatrix}6-0+(0-2)^2\\6-2+(0-2)^2\end{pmatrix} = \begin{pmatrix}10\\8\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \Phi\begin{pmatrix}-2\\0\end{pmatrix} = \begin{pmatrix}6+2+(-2-0)^2\\6-0+(-2-0)^2\end{pmatrix} = \begin{pmatrix}12\\10\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \Phi\begin{pmatrix}0\\-2\end{pmatrix} = \begin{pmatrix}6-0+(0+2)^2\\6+2+(0+2)^2\end{pmatrix} = \begin{pmatrix}10\\12\end{pmatrix}$

- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}2\\0\end{pmatrix}=\begin{pmatrix}8\\10\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}10\\8\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}-2\\0\end{pmatrix}=\begin{pmatrix}12\\10\end{pmatrix}$
- $\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}0\\-2\end{pmatrix}=\begin{pmatrix}10\\12\end{pmatrix}$

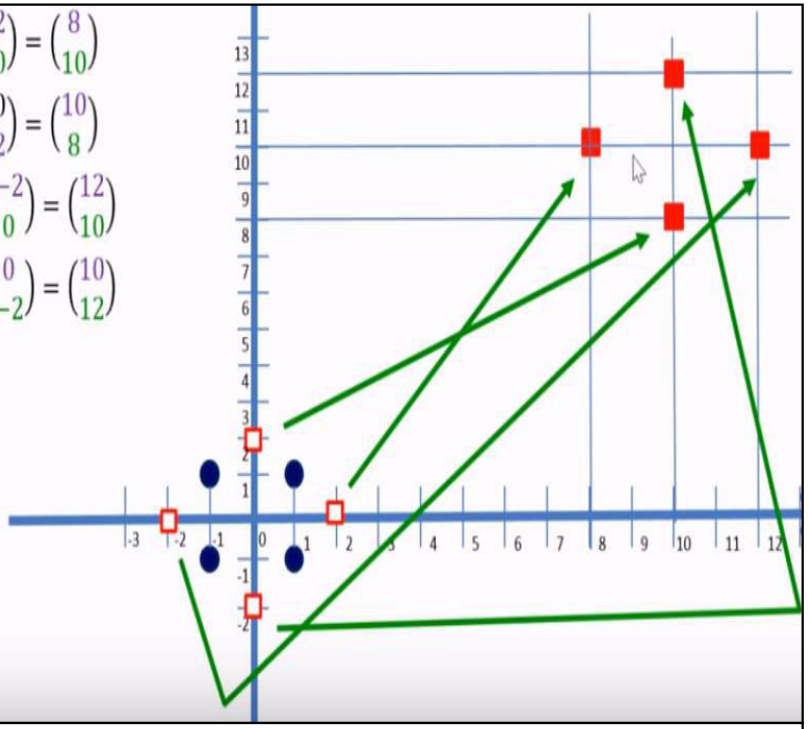


$\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}2\\0\end{pmatrix}=\begin{pmatrix}8\\10\end{pmatrix}$

$\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}10\\8\end{pmatrix}$

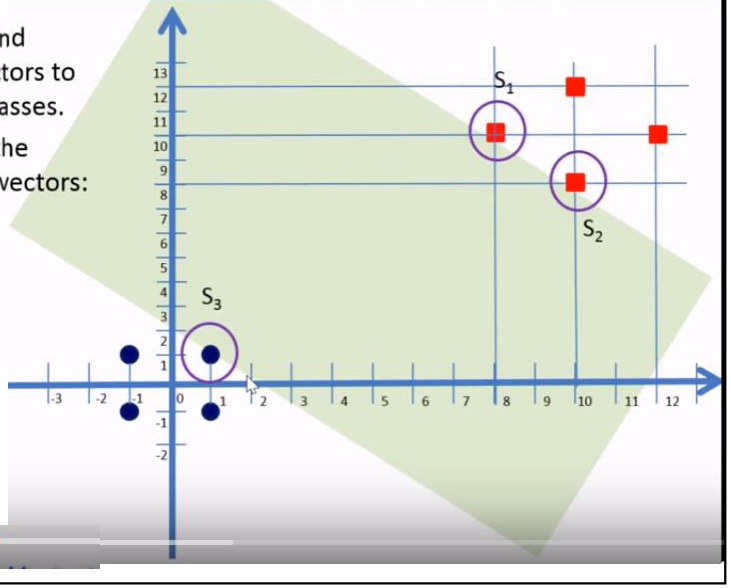
$\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}-2\\0\end{pmatrix}=\begin{pmatrix}12\\10\end{pmatrix}$

$\Phi\begin{pmatrix}x_1\\x_2\end{pmatrix}=\Phi\begin{pmatrix}0\\-2\end{pmatrix}=\begin{pmatrix}10\\12\end{pmatrix}$



Selection of Support Vectors

- Now our task is to find suitable support vectors to classify these two classes.
- Here we will select the following 3 support vectors:
- $S_1=\begin{pmatrix}8\\10\end{pmatrix}$,
- $S_2=\begin{pmatrix}10\\8\end{pmatrix}$,
- and $S_3=\begin{pmatrix}1\\1\end{pmatrix}$



Augmented Support Vector

- Here we will use vectors augmented with a 1 as a bias input, and for clarity we will differentiate these with an over-tilde. That is:

$$S_1=\begin{pmatrix}8\\10\end{pmatrix} \qquad \widetilde{S_1}=\begin{pmatrix}8\\10\\1\end{pmatrix}$$

$$S_2=\begin{pmatrix}10\\8\end{pmatrix} \qquad \widetilde{S_2}=\begin{pmatrix}10\\8\\1\end{pmatrix}$$

$$S_3=\begin{pmatrix}1\\1\end{pmatrix} \qquad \widetilde{S_3}=\begin{pmatrix}1\\1\\1\end{pmatrix}$$

Find Parameters by Substituting Support vectors

- Now we need to find 3 parameters α_1, α_2, and α_3 based on the following 3 linear equations:

$$\alpha_1 \widetilde{S_1}.\widetilde{S_1} + \alpha_2 \widetilde{S_2}.\widetilde{S_1} + \alpha_3 \widetilde{S_3}.\widetilde{S_1} = +1 \ \ (+ve\ class)$$

$$\alpha_1 \widetilde{S_1}.\widetilde{S_2} + \alpha_2 \widetilde{S_2}.\widetilde{S_2} + \alpha_3 \widetilde{S_3}.\widetilde{S_2} = +1 \ (+ve\ class)$$

$$\alpha_1 \widetilde{S_1}.\widetilde{S_3} + \alpha_2 \widetilde{S_2}.\widetilde{S_3} + \alpha_3 \widetilde{S_3}.\widetilde{S_3} = -1 \ \ (-ve\ class)$$

- Let's substitute the values for $\widetilde{S_1}$, $\widetilde{S_2}$ and $\widetilde{S_3}$ in the above equations.

- Let's substitute the values for $\widetilde{S_1}$, $\widetilde{S_2}$ and $\widetilde{S_3}$ in the above equations.

$$\widetilde{S_1} = \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} \quad \widetilde{S_2} = \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} \quad \widetilde{S_3} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\alpha_1 \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix}.\begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix}.\begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} = +1$$

$$\alpha_1 \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix}.\begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix}.\begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} = +1$$

$$\alpha_1 \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix}.\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix}.\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = -1$$

After simplification we get:

$$165\ \alpha_1 + 161\ \alpha_2 + 19\ \alpha_3 = -1$$

$$161\ \alpha_1 + 165\ \alpha_2 + 19\ \alpha_3 = -1$$

$$19\ \alpha_1 + 19\ \alpha_2 + 3\ \alpha_3 = +1$$

Simplifying the above 3 simultaneous equations we get: $\alpha_1 = \alpha_2 = 0.859$ and $\alpha_3 = -1.4219$.

Discriminating Hyper plane

- The hyper plane that discriminates the positive class from the negative class is given by:

$$\widetilde{w} = \sum_i \alpha_i \widetilde{S_i}$$

- Substituting the values we get:

$$\widetilde{w} = \alpha_1 \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\widetilde{w} = (0.0859).\begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} + (0.0859).\begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} + (-1.4219).\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.1243 \\ 0.1243 \\ -1.2501 \end{pmatrix}$$

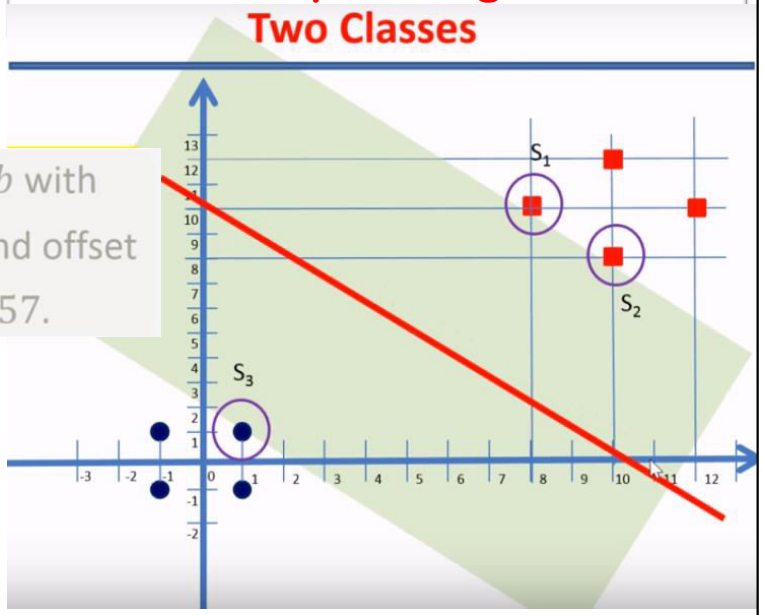
- Our vectors are augmented with a bias.
- Hence we can equate the entry in $\widetilde{w}$ as the hyper plane with an offset b.
- Therefore the separating hyper plane equation

$y = wx + b$ with $w = \begin{pmatrix} 0.125/0.125 \\ 0.125/0.125 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

and an offset $b = -\frac{1.250}{0.125} = -10.057.$

Separating Two Classes

$y = wx + b$ with $w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and offset $b = -10.057.$



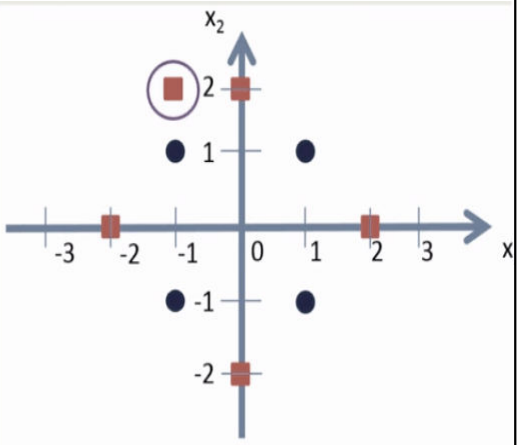
Let's consider a classification example here.

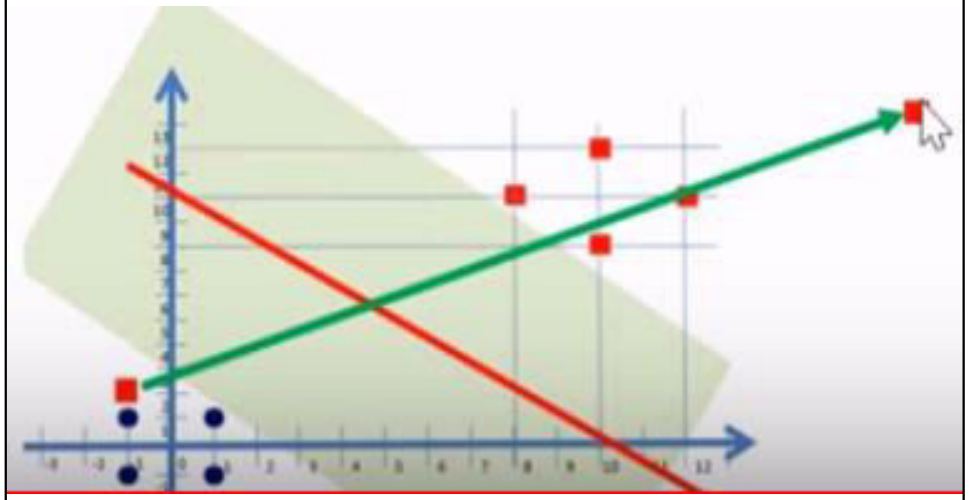
Let's classify the point $(x_1,x_2)=(-1,2)$.

$\Phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \Phi\begin{pmatrix} -1 \\ 2 \end{pmatrix} =$

$\begin{pmatrix} 6+1+(-1-2)^2 \\ 6-2+(-1-2)^2 \end{pmatrix} = \begin{pmatrix} 16 \\ 13 \end{pmatrix}$

$w.\Phi(x) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 16 \\ 13 \end{pmatrix} = 29 > 10$





Non-Linear SVM – Example (cont'd)

Our goal, again, is to discover a separating hyperplane that accurately discriminates the two classes. Of course, it is obvious that no such hyperplane exists in the input space (that is, in the space in which the original input data live). Therefore, we must use a nonlinear SVM (that is, one whose mapping function Φ is a nonlinear mapping from input space into some feature space). Define

$$\Phi_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} 4 - x_2 + |x_1 - x_2| \\ 4 - x_1 + |x_1 - x_2| \end{pmatrix} & \text{if } \sqrt{x_1^2 + x_2^2} > 2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \text{otherwise} \end{cases}$$

Example (cont'd)

we can see how Φ transforms our data before

the dot products are performed. Therefore, we can rewrite the data in feature space as

$$\left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 6 \\ 2 \end{pmatrix}, \begin{pmatrix} 6 \\ 6 \end{pmatrix}, \begin{pmatrix} 2 \\ 6 \end{pmatrix} \right\}$$

for the positive examples and

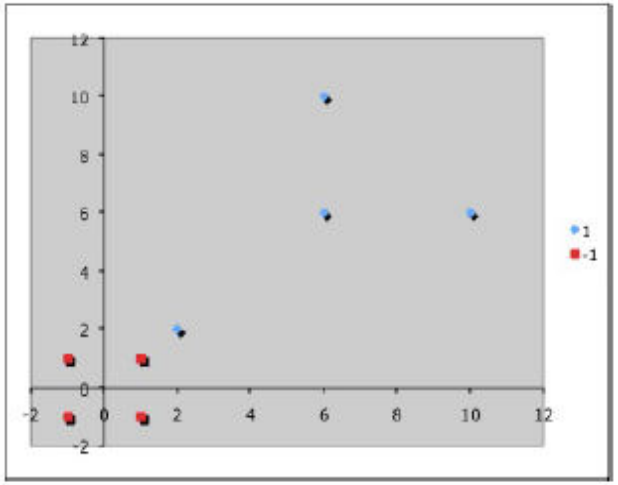
$$\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\}$$

for the negative examples. Now we can once again easily identify the support vectors

$$\left\{ s_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, s_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}$$

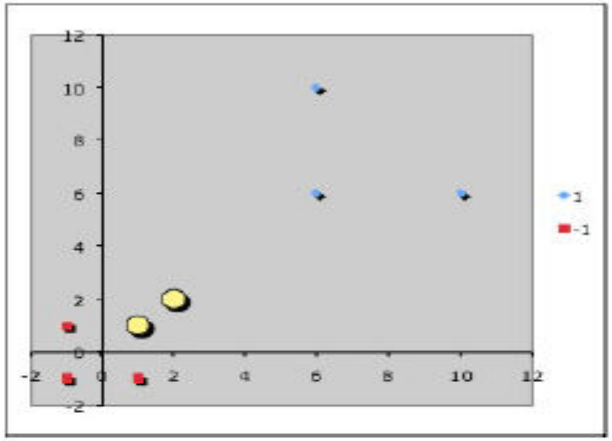
We again use vectors augmented with a 1 as a bias input and will differentiate them as before. Now given the [augmented] support vectors, we must again find values for the α_i.

Data represented in feature space - Example (cont'd)



Data represented in feature space - Example (cont'd)

- The two support vectors (in feature space) are marked as yellow circles.



Example (cont'd)

The augmented support vectors are

$$\tilde{s_1} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \tilde{s_2} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

The constraints are

$$\begin{aligned} \alpha_1 \Phi_1(s_1) \cdot \Phi_1(s_1) + \alpha_2 \Phi_1(s_2) \cdot \Phi_1(s_1) &= -1 \\ \alpha_1 \Phi_1(s_1) \cdot \Phi_1(s_2) + \alpha_2 \Phi_1(s_2) \cdot \Phi_1(s_2) &= +1 \end{aligned}$$

$$\begin{aligned} \alpha_1 \tilde{s_1} \cdot \tilde{s_1} + \alpha_2 \tilde{s_2} \cdot \tilde{s_1} &= -1 \\ \alpha_1 \tilde{s_1} \cdot \tilde{s_2} + \alpha_2 \tilde{s_2} \cdot \tilde{s_2} &= +1 \end{aligned}$$

Now, computing the dot products results in

$$\begin{aligned} 3\alpha_1 + 5\alpha_2 &= -1 \\ 5\alpha_1 + 9\alpha_2 &= +1 \end{aligned}$$

And the solution to this system of equations is $\alpha_1 = -7$ and $\alpha_2 = 4$.

Example (cont'd)

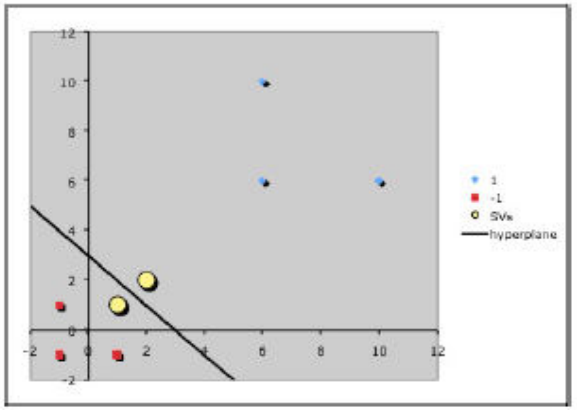
Finally, the discriminating hyperplane in input space that corresponds to these α.

$$\begin{aligned} \tilde{w} &= \sum_i \alpha_i \tilde{s}_i \\ &= -7 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 1 \\ -3 \end{pmatrix} \end{aligned}$$

giving us the separating hyperplane equation $y = wx + b$ with $w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $b = -3$. Plotting the line gives the expected decision surface.

Hyperplane - Example (cont'd)

- The discriminating hyperplane corresponding to the values α_1 = -7 and α_2 = 4



Support Vector Machine (SVM)

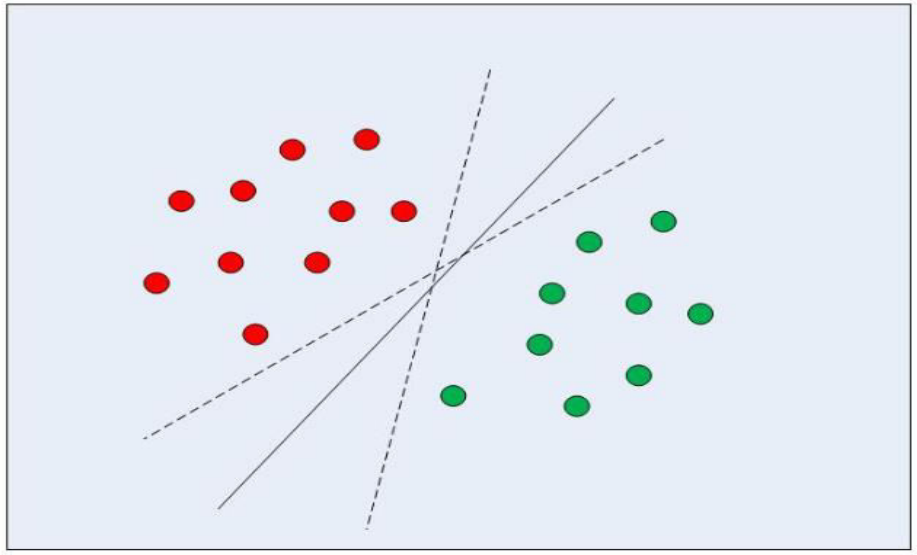
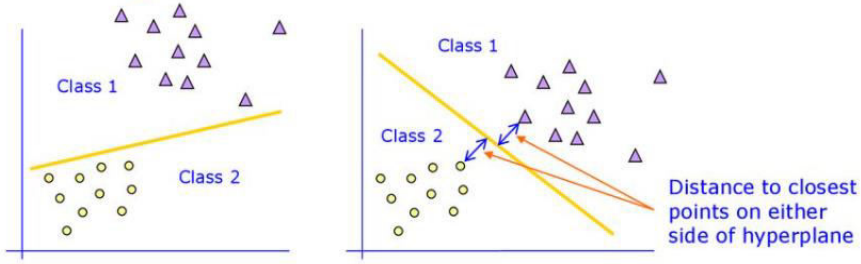


Fig 1: Multiple Decision Boundaries

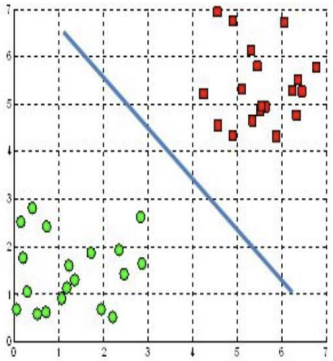
SVM Design Objective

- Find the hyperplane that maximizes the margin

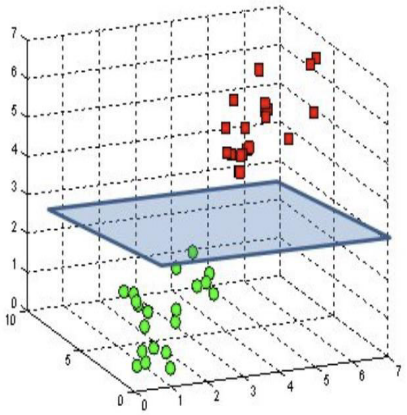


Hyperplanes in 2D and 3D feature space

A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



Simple SVM

- A support vector machine (SVM) is a type of supervised machine learning classification algorithm.
- SVMs were introduced initially in 1960s and were later refined in 1990s.
- In case of linearly separable data in two dimensions, a typical machine learning algorithm tries to find a boundary that divides the data in such a way that the misclassification error can be minimized.
- There can be several boundaries that correctly divide the data points. The two dashed lines as well as one solid line classify the data correctly.

Simple SVM

- SVM differs from the other classification algorithms in the way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes.
- An SVM doesn't merely find a decision boundary; it finds the most optimal decision boundary.
- The most optimal decision boundary is the one which has maximum margin from the nearest points of all the classes.
- SVMs are one of the most robust prediction methods, being based on **statistical learning frameworks** or VC theory

Hypothesis Space

- Our hypothesis space is the space of functions

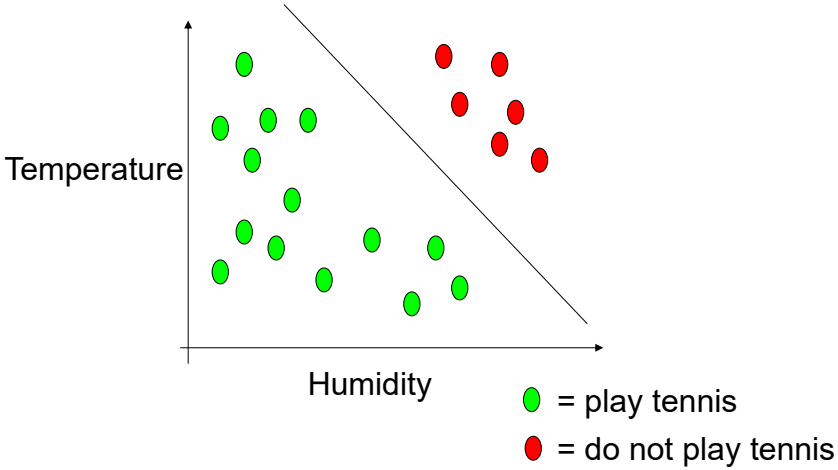
$$f(X, W, w_0) = \text{sign}(W \cdot X + w_0)$$

- Similar to Perceptron, but now we want to maximize the margins from the separating hyperplane to the nearest positive and negative data points.
- Find the maximum margin hyperplane for the given training set.

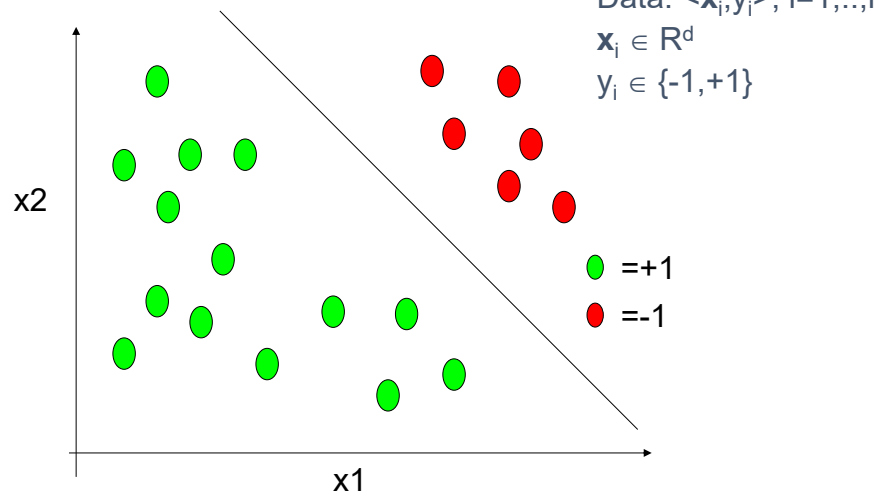
Simple SVM

- The nearest points from the decision boundary that maximize the distance between the decision boundary and the points are called support vectors.
- The decision boundary in case of support vector machines is called the maximum margin classifier, or the maximum margin hyper plane.
- There is complex mathematics involved behind finding the support vectors, **calculating the margin between decision boundary and the support vectors and maximizing this margin.**

Tennis example

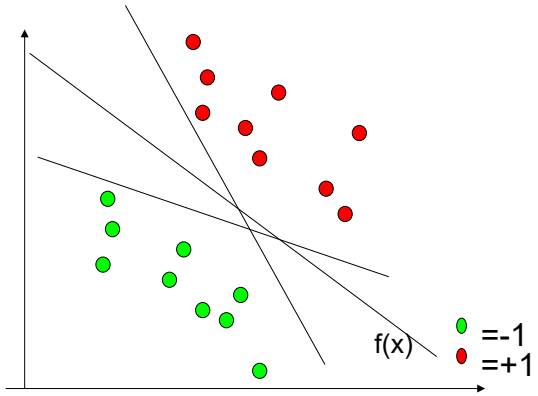


Linear Support Vector Machines



Data: $<\mathbf{x}_i, y_i>$, i=1,..,l
$\mathbf{x}_i \in R^d$
$y_i \in \{-1,+1\}$

Linear SVM 2



Data: $<\mathbf{x}_i, y_i>$, i=1,..,l
$\mathbf{x}_i \in R^d$
$y_i \in \{-1,+1\}$

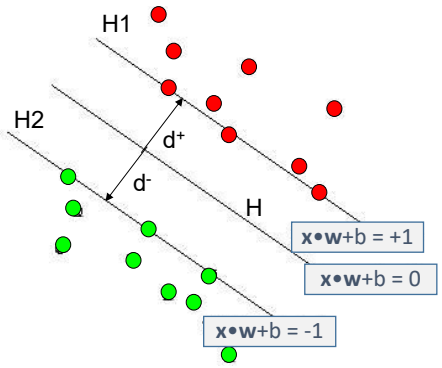
All hyperplanes in R^d are parameterize by a vector ($\mathbf{w}$) and a constant b. Can be expressed as $\mathbf{w}\bullet\mathbf{x}+b=0$ (remember the equation for a hyperplane from algebra!)

Our aim is to find such a hyperplane f(x)=sign($\mathbf{w}\bullet\mathbf{x}+b$), that correctly classify our data.

Definitions

Define the hyperplane H such tl
$\mathbf{x}_i\bullet\mathbf{w}+b \geq +1$ when $y_i = +1$
$\mathbf{x}_i\bullet\mathbf{w}+b \leq -1$ when $y_i = -1$

H1 and H2 are the planes:
H1: $\mathbf{x}_i\bullet\mathbf{w}+b = +1$
H2: $\mathbf{x}_i\bullet\mathbf{w}+b = -1$
The points on the planes H1 and H2 are the Support Vectors

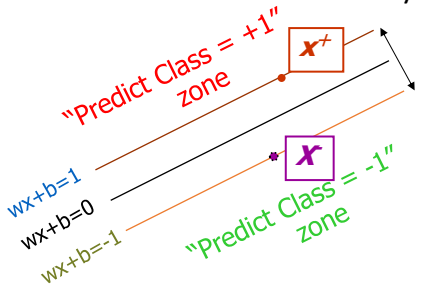


d+ = the shortest distance to the closest positive point

d- = the shortest distance to the closest negative point

The margin of a separating hyperplane is $d^+ + d^-$.

Linear SVM Mathematically



M=Margin Width

What we know:
- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{w} \cdot (\boldsymbol{x}^+ - \boldsymbol{x}^-) = 2$

$$M = \frac{(x^+ - x^-)\cdot w}{|w|} = \frac{2}{|w|}$$

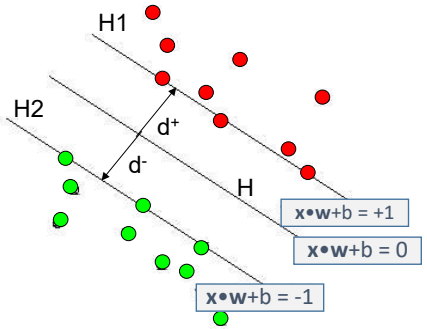
Maximizing the margin

We want a classifier with as big margin as possible.

Recall the distance from a point(x_0,y_0) to a line:
Ax+By+c = 0 is|A x_0 +B y_0 +c|/sqrt(A^2+B^2)

The distance between H and H1 is:
|**w•x**+b|/||w||=1/||w||

The distance between H1 and H2 is: 2/||w||



In order to maximize the margin, we need to minimize ||w||. With the condition that there are no datapoints between H1 and H2:

x_i•**w**+b ≥ +1 when y_i =+1
x_i•**w**+b ≤ -1 when y_i =-1 } **Can be combined into yi(x_i•w+b) ≥ 1**

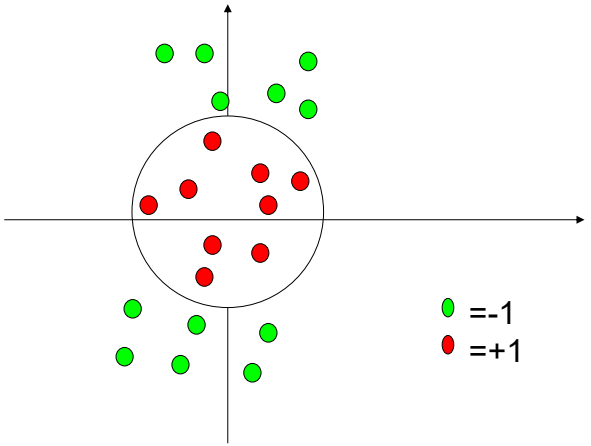
$y = \boldsymbol{w}^T\boldsymbol{x}$

where $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$ and $\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$ are *augmented* vectors to include also the bias weight(coefficient) and input.

During testing, with given weights, ***w***, for input ***x***, we compute the output *y*. To implement a given task, we need to *learn* the weights ***w***, the parameters of the system, such that correct outputs are generated given the inputs. When d = 1 and x is fed from the environment through an input unit, we have

$y = wx + w_0$ w_0 = b in the above equation x_i•**w**+b

Problems with linear SVM



What if the decision function is not a linear?

SVM Applications

- **SVM has been used successfully in many real-world problems**
  - **text (and hypertext) categorization**
  - **image classification**
  - **bioinformatics (Protein classification, Cancer classification)**
  - **hand-written character recognition**

SVM –Example

Linear Functions

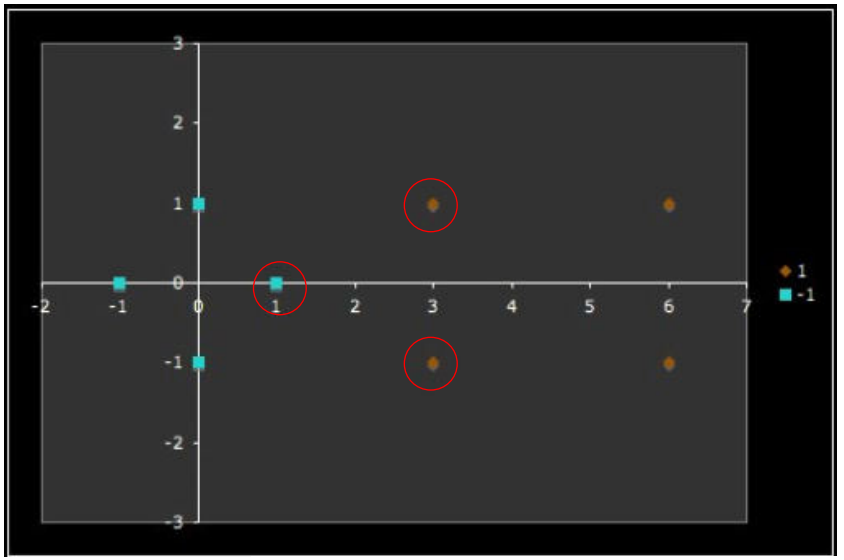
SVM by Example –Linearly separable data

Suppose we are given the following positively labeled data points in $\Re^2$:

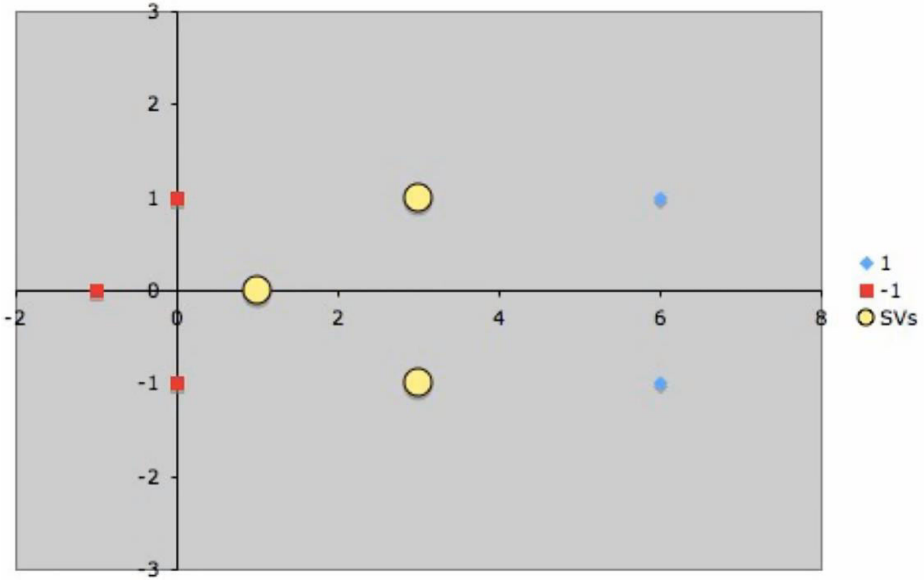
$$\left\{\begin{pmatrix}3\\1\end{pmatrix},\begin{pmatrix}3\\-1\end{pmatrix},\begin{pmatrix}6\\1\end{pmatrix},\begin{pmatrix}6\\-1\end{pmatrix}\right\}$$

and the following negatively labeled data points in $\Re^2$

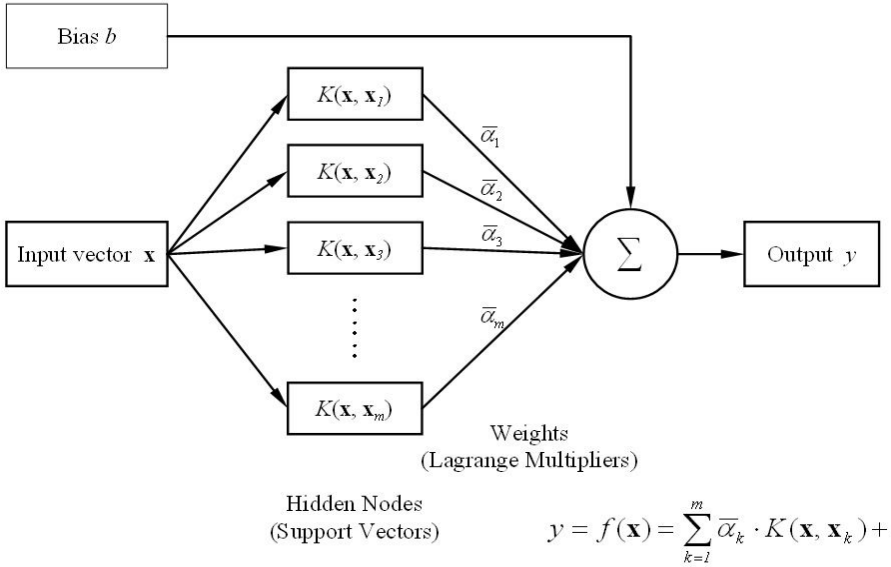
$$\left\{\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\-1\end{pmatrix},\begin{pmatrix}-1\\0\end{pmatrix}\right\}$$



Support Vectors are $\left\{s_1=\begin{pmatrix}1\\0\end{pmatrix}, s_2=\begin{pmatrix}3\\1\end{pmatrix}, s_3=\begin{pmatrix}3\\-1\end{pmatrix}\right\}$

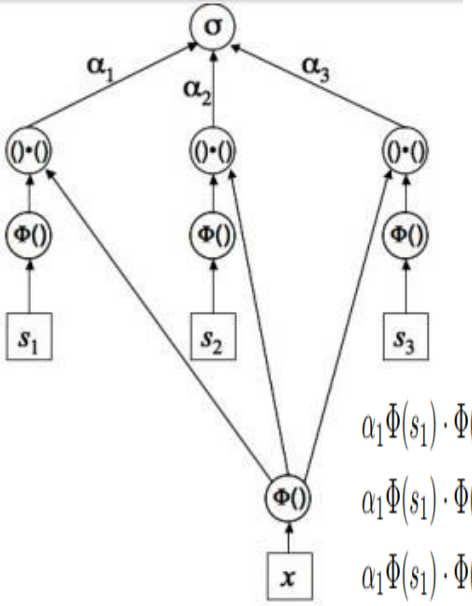


Architecture



$$y = f(\mathbf{x}) = \sum_{k=1}^{m} \overline{\alpha}_k \cdot K(\mathbf{x}, \mathbf{x}_k) + b$$

SVM Architecture



$$y = f(\mathbf{x}) = \sum_{k=1}^{m} \overline{\alpha}_k \cdot K(\mathbf{x}, \mathbf{x}_k) + b$$

$$\begin{aligned}
\alpha_1 \Phi(s_1)\cdot\Phi(s_1) + \alpha_2 \Phi(s_2)\cdot\Phi(s_1) + \alpha_3 \Phi(s_3)\cdot\Phi(s_1) &= -1 \\
\alpha_1 \Phi(s_1)\cdot\Phi(s_2) + \alpha_2 \Phi(s_2)\cdot\Phi(s_2) + \alpha_3 \Phi(s_3)\cdot\Phi(s_2) &= +1 \\
\alpha_1 \Phi(s_1)\cdot\Phi(s_3) + \alpha_2 \Phi(s_2)\cdot\Phi(s_3) + \alpha_3 \Phi(s_3)\cdot\Phi(s_3) &= +1
\end{aligned}$$

Three support vectors without bias term are represented as follows

$$\left\{ s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right\}$$

Three support vectors with bias term are represented as follows:

$$\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad \tilde{s}_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \quad \tilde{s}_3 = \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$$

let $\Phi() = I$

$$y = f(\mathbf{x}) = \sum_{k=1}^{m} \overline{\alpha}_k \cdot K(\mathbf{x}, \mathbf{x}_k) + b$$

$$\begin{aligned}
\alpha_1 \tilde{s}_1\cdot\tilde{s}_1 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_1 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_1 &= -1 \\
\alpha_1 \tilde{s}_1\cdot\tilde{s}_2 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_2 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_2 &= +1 \\
\alpha_1 \tilde{s}_1\cdot\tilde{s}_3 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_3 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_3 &= +1
\end{aligned}$$

$$\begin{aligned}
2\alpha_1 + 4\alpha_2 + 4\alpha_3 &= -1 \\
4\alpha_1 + 11\alpha_2 + 9\alpha_3 &= +1 \\
4\alpha_1 + 9\alpha_2 + 11\alpha_3 &= +1
\end{aligned}$$

Support Vector Machine - Linear Example Solved

$$\begin{aligned}
\alpha_1 \tilde{s}_1\cdot\tilde{s}_1 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_1 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_1 &= -1 \quad & \alpha_1(1+0+1)+\alpha_2(3+0+1)+\alpha_3(3+0+1) &= -1 \\
\alpha_1 \tilde{s}_1\cdot\tilde{s}_2 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_2 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_2 &= +1 \quad & \alpha_1(3+0+1)+\alpha_2(9+1+1)+\alpha_3(9-1+1) &= 1 \\
\alpha_1 \tilde{s}_1\cdot\tilde{s}_3 + \alpha_2 \tilde{s}_2\cdot\tilde{s}_3 + \alpha_3 \tilde{s}_3\cdot\tilde{s}_3 &= +1 \quad & \alpha_1(3+0+1)+\alpha_2(9-1+1)+\alpha_3(9+1+1) &= 1
\end{aligned}$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} = 1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} = 1$$

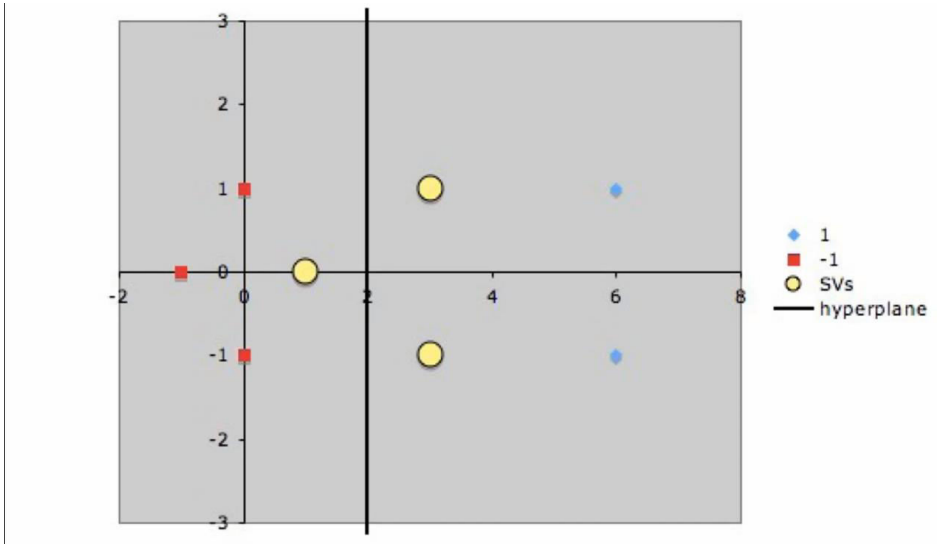
$$\begin{aligned}
\alpha_1 &= -3.5 \\
\alpha_2 &= 0.75 \\
\alpha_3 &= 0.75
\end{aligned}$$

By solving $\alpha_1 = -3.5$, $\alpha_2 = 0.75$ and $\alpha_3 = 0.75$

$$\begin{aligned}
\tilde{w} &= \sum_i \alpha_i \tilde{s}_i \\
&= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix}
\end{aligned}$$

Hyperplane Equation is $y = wx + b$ with $w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $b = -2$

Hyperplane



Show me who your friends are, and I will tell you what you are.

Vladimir Lenin

SVM Example - 2

- Construct a linear discriminant with :
  - Positive samples : (1,1) , (1,-1), (2,1), (2,-1)
  - Negative samples : (4,0) , (6,0), (5,1), (5,-1)

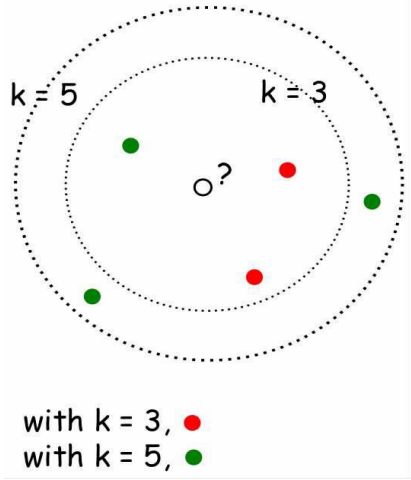
K Nearest Neighbour Classifier

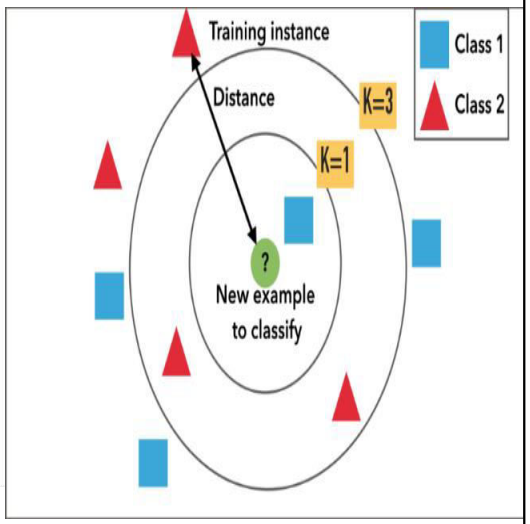
Eager Learners vs Lazy Learners

- Eager learners, when given a set of training tuples, will construct a **generalization model** before receiving new (e.g., test) tuples to classify.
- Lazy learners simply stores data (or does only a little minor processing) and waits until it is given a test tuple.
- Lazy learners store the training tuples or "instances," they are also referred to as instance based learners, even though all learning is essentially based on instances.
- Lazy learner: less time in training but more in predicting.

-k- Nearest Neighbor Classifier

-case based Classifier





What is k- NN??

- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).
- KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

K????

When K is small, we are restraining the region of a given prediction and forcing our classifier to be "more blind" to the overall distribution.

A small value for K provides the most flexible fit, which will have **low bias but high variance**.

Larger values of K will have smoother decision boundaries which means **lower variance** but **increased bias**.

Remarks!!

- Similarity Function Based.
- Choose an odd value of k for 2 class problem.
- k must not be multiple of number of classes.

Closeness

- The Euclidean distance between two points or tuples, say,

X1 = (x11,x12,...,x1n) and X2 =(x21,x22,...,x2n), is

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}.$$

- Min-max normalization can be used to transform a value v of a numeric attribute A to v0 in the range [0,1] by computing

$$v' = \frac{v - min_A}{max_A - min_A},$$

How to determine a good value for k?

- Starting with k = 1, we use a test set to estimate the error rate of the classifier.
- The k value that gives the minimum error rate may be selected.

KNN Algorithm and Example

Distance Measures

- Euclidean distance

$$d(g_1, g_2) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Manhattan distance

$$d(g_1, g_2) = \sum_{i=1}^{n} |(x_i - y_i)|$$

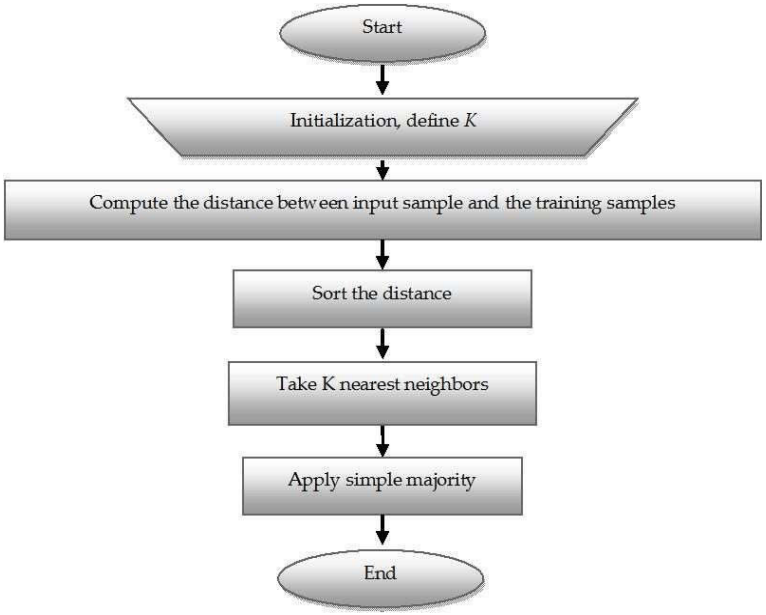
- Minkowski distance

$$d(g_1, g_2) = \sqrt[m]{\sum_{i=1}^{n} (x_i - y_i)^m}$$

Which distance measure to use?

We use standard Euclidean Distance as it treats each feature as equally important.

KNN Classifier Algorithm



The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Example

We have data from the questionnaires survey and objective testing with two attributes (acid durability and strength) to classify whether a special paper **tissue is good or not.** Here are four training samples :

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that passes the laboratory test with X1 = 3 and X2 = 7. Guess the classification of this new tissue.

- Step 1 : **Initialize and Define k.**

  Lets say, k = 3

  (Always choose k as an odd number if the number of attributes is even to avoid a tie in the class prediction)

- Step 2 : **Compute the distance between input sample and Training sample**

  - Co-ordinate of the input sample is (3,7).

  - Instead of calculating the Euclidean distance, we calculate the Squared Euclidean distance.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance
7	7	$(7-3)^2 + (7-7)^2 = \mathbf{16}$
7	4	$(7-3)^2 + (4-7)^2 = \mathbf{25}$
3	4	$(3-3)^2 + (4-7)^2 = \mathbf{09}$
1	4	$(1-3)^2 + (4-7)^2 = \mathbf{13}$

Step 3 : **Sort the distance and determine the nearest neighbours based of the Kth minimum distance :**

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?
7	7	16	**3**	**Yes**
7	4	25	**4**	**No**
3	4	09	**1**	**Yes**
1	4	13	**2**	**Yes**

Step 4 : **Take 3-Nearest Neighbours:**

Gather the category Y of the nearest neighbours.

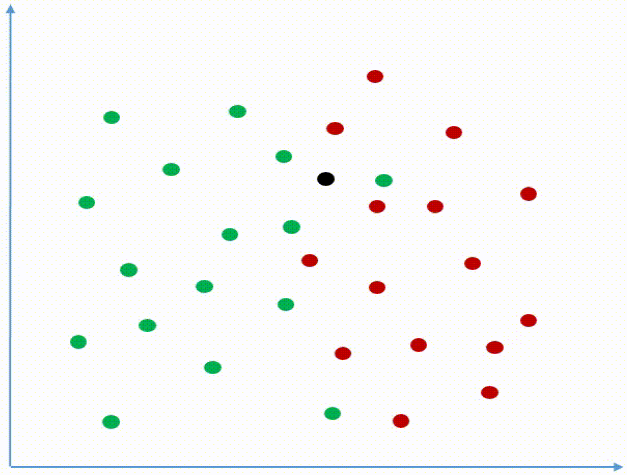
X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?	Y = Category of the nearest neighbour
7	7	16	3	**Yes**	**Bad**
7	4	25	4	**No**	-
3	4	09	1	**Yes**	**Good**
1	4	13	2	**Yes**	**Good**

- Step 5 : **Apply simple majority**

- Use simple majority of the category of the nearest neighbours as the prediction value of the query instance.

- We have 2 “good” and 1 “bad”. Thus we conclude that the new paper tissue that passes the laboratory test with X1 = 3 and X2 = 7 is included in the “good” category.

K-Nearest Neighbors Classification



machinelearningknowledge.ai

Pros:

No assumptions about data — useful, for example, for nonlinear data
Simple algorithm — to explain and understand/interpret
There's no need to build a model, tune several parameters
Versatile — useful for classification or regression

Cons:

Computationally expensive — because the algorithm stores all of the training data
High memory requirement
Stores all (or almost all) of the training data
Prediction stage might be slow (with big N)
Sensitive to irrelevant features and the scale of the data

Applications of KNN Classifier

- Used in classification
- Used to get missing values
- Used in pattern recognition
- Used in gene expression
- Used in protein-protein prediction
- Used to get 3D structure of protein
- Used to measure document similarity

Comparison of various classifiers

Algorithm	Features	Limitations
C4.5 Algorithm	- Models built can be easily interpreted - Easy to implement - Can use both discrete and continuous values - Deals with noise	- Small variation in data can lead to different decision trees - Does not work very well on small training dataset - Over-fitting
ID3 Algorithm	- It produces more accuracy than C4.5 - Detection rate is increased and space consumption is reduced	- Requires large searching time - Sometimes it may generate very long rules which are difficult to prune - Requires large amount of memory to store tree
K-Nearest Neighbour Algorithm	- Classes need not be linearly separable - Zero cost of the learning process - Sometimes it is robust with regard to noisy training data - Well suited for multimodal	- Time to find the nearest neighbours in a large training dataset can be excessive - It is sensitive to noisy or irrelevant attributes - Performance of the algorithm depends on the number of dimensions used

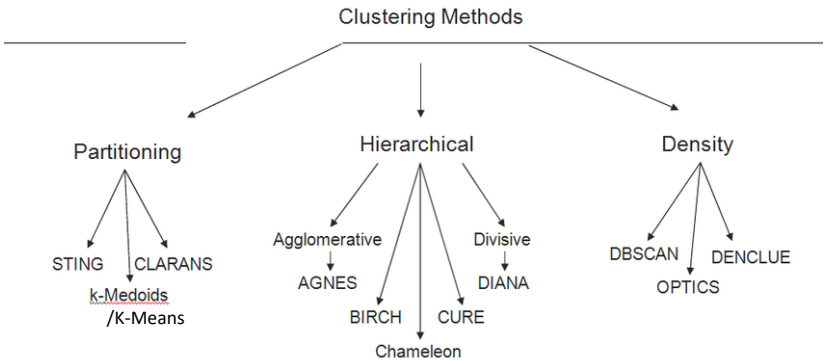
Naïve Bayes Algorithm	- Simple to implement - Great computational efficiency and classification rate - It predicts accurate results for most of the classification and prediction problems	- The precision of the algorithm decreases if the amount of data is less - For obtaining good results, it requires a very large number of records
Support vector machine Algorithm	- High accuracy - Work well even if the data is not linearly separable in the base feature space	- Speed and size requirement both in training and testing is more - High complexity and extensive memory requirements for classification in many cases
Artificial Neural Networks Algorithm	- It is easy to use with few parameters to adjust - A neural network learns and reprogramming is not needed. - Easy to implement - Applicable to a wide range of problems in real life.	- Requires high processing time if neural network is large - Difficult to know how many neurons and layers are necessary - Learning can be slow

Conclusion

- KNN is what we call *lazy learning (vs. eager learning)*
- Conceptually simple, easy to understand and explain
- Very flexible decision boundaries
- Not much learning at all!
- It can be hard to find a good distance measure
- Irrelevant features and noise can be very detrimental
- Typically can not handle more than a few dozen attributes
- Computational cost: requires a lot computation

Clustering

Cluster can be defined as "**A way of grouping the data points into different clusters**, consisting of similar data points.



K-Means Clustering

K-Means Clustering-

- K-Means clustering is an unsupervised iterative clustering technique.
- It partitions the given data set into k predefined distinct clusters.
- A cluster is defined as a collection of data points exhibiting certain similarities.

Partitions the data set such that-

- Each data point belongs to a cluster with the nearest mean.
- Data points belonging to one cluster have high degree of similarity.
- Data points belonging to different clusters have high degree of dissimilarity.

K-Means Clustering Algorithm-

K-Means Clustering Algorithm involves the following steps-

Step-01:

•Choose the number of clusters K.

Step-02:

•Randomly select any K data points as cluster centers.
•Select cluster centers in such a way that they are as farther as possible from each other.

Step-03:

•Calculate the distance between each data point and each cluster center.
•The distance may be calculated either by using given distance function or by using euclidean distance formula.

Step-04:

•Assign each data point to some cluster.
•A data point is assigned to that cluster whose center is nearest to that data point.

Step-05:

•Re-compute the center of newly formed clusters.

Advantages-

K-Means Clustering Algorithm offers the following advantages-

Point-01:

It is relatively efficient with time complexity O(n*k*t) where-
•n = number of instances
•k = number of clusters
•t = number of iterations

isadvantages-

K-Means Clustering Algorithm has the following disadvantages-
•It requires to specify the number of clusters (k) in advance.
•It can not handle noisy data and outliers.
•It is not suitable to identify clusters with non-convex shapes.

Cluster the following eight points (with (x, y) representing locations) into three clusters:

A1(2, 10), A2(2, 5), A3(8, 4), A4(5, 8), A5(7, 5), A6(6, 4), A7(1, 2), A8(4, 9)

Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2).

The distance function between two points a = (x1, y1) and b = (x2, y2) is defined as-

$$P(a, b) = |x2 – x1| + |y2 – y1|$$

Use K-Means Algorithm to find the three cluster centers after the second iteration.

Iteration-01:

•We calculate the distance of each point from each of the center of the three clusters.
•The distance is calculated by using the given distance function.

The following illustration shows the calculation of distance between point A1(2, 10) and each of the center of the three clusters-

Calculating Distance Between A1(2, 10) and C1(2, 10)-

P(A1, C1)
= |x2 – x1| + |y2 – y1|
= |2 – 2| + |10 – 10|
= 0

Calculating Distance Between A1(2, 10) and C2(5, 8)-

P(A1, C2)
= |x2 – x1| + |y2 – y1|
= |5 – 2| + |8 – 10|
= 3 + 2
= 5

Calculating Distance Between A1(2, 10) and C3(1, 2)-

P(A1, C3)
= |x2 – x1| + |y2 – y1|
= |1 – 2| + |2 – 10|
= 1 + 8
= 9

In the similar manner, we calculate the distance of other points from each of the center of the three clusters.

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (5, 8) of Cluster-02	Distance from center (1, 2) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	5	9	C1
A2(2, 5)	5	6	4	C3
A3(8, 4)	12	7	9	C2
A4(5, 8)	5	0	10	C2
A5(7, 5)	10	5	9	C2
A6(6, 4)	10	5	7	C2
A7(1, 2)	9	10	0	C3
A8(4, 9)	3	2	10	C2

From here, New clusters are-

Cluster-01:
First cluster contains points-
•A1(2, 10)

Cluster-02:
Second cluster contains points-
•A3(8, 4)
•A4(5, 8)
•A5(7, 5)
•A6(6, 4)
•A8(4, 9)

Cluster-03:
Third cluster contains points-
•A2(2, 5)
•A7(1, 2)

Now,
•We re-compute the new cluster clusters.
•The new cluster center is computed by taking mean of all the points contained in that cluster.

For Cluster-01:
•We have only one point A1(2, 10) in Cluster-01.
•So, cluster center remains the same.

For Cluster-02:
Center of Cluster-02
= ((8 + 5 + 7 + 6 + 4)/5, (4 + 8 + 5 + 4 + 9)/5)
= (6, 6)

For Cluster-03:
Center of Cluster-03
= ((2 + 1)/2, (5 + 2)/2)
= (1.5, 3.5)

This is completion of Iteration-01.

Iteration-02:

•We calculate the distance of each point from each of the center of the three clusters.
•The distance is calculated by using the given distance function.

The following illustration shows the calculation of distance between point A1(2, 10) and each of the center of the three clusters-

Calculating Distance Between A1(2, 10) and C1(2, 10)-
P(A1, C1)
= |x2 – x1| + |y2 – y1|
= |2 – 2| + |10 – 10|
= 0

Calculating Distance Between A1(2, 10) and C2(6, 6)-
P(A1, C2)
= |x2 – x1| + |y2 – y1|
= |6 – 2| + |6 – 10|
= 4 + 4
= 8

Calculating Distance Between A1(2, 10) and C3(1.5, 3.5)-
P(A1, C3)
= |x2 – x1| + |y2 – y1|
= |1.5 – 2| + |3.5 – 10|
= 0.5 + 6.5
= 7

In the similar manner, we calculate the distance of other points from each of the center of the three clusters.

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (6, 6) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	8	7	C1
A2(2, 5)	5	5	2	C3
A3(8, 4)	12	4	7	C2
A4(5, 8)	5	3	8	C2
A5(7, 5)	10	2	7	C2
A6(6, 4)	10	2	5	C2
A7(1, 2)	9	9	2	C3
A8(4, 9)	3	5	8	C1

From here, New clusters are-

Cluster-01:

First cluster contains points-

•A1(2, 10)

•A8(4, 9)

Cluster-02:

Second cluster contains points-

•A3(8, 4)

•A4(5, 8)

•A5(7, 5)

•A6(6, 4)

Cluster-03:

Third cluster contains points-

•A2(2, 5)

•A7(1, 2)

Now,

•We re-compute the new cluster clusters.

•The new cluster center is computed by taking mean of all the points contained in that cluster.

For Cluster-01:

Center of Cluster-01

= ((2 + 4)/2, (10 + 9)/2)

= (3, 9.5)

For Cluster-02:

Center of Cluster-02

= ((8 + 5 + 7 + 6)/4, (4 + 8 + 5 + 4)/4)

= (6.5, 5.25)

For Cluster-03:

Center of Cluster-03

= ((2 + 1)/2, (5 + 2)/2)

= (1.5, 3.5)

This is completion of Iteration-02.

After second iteration, the center of the three clusters are-

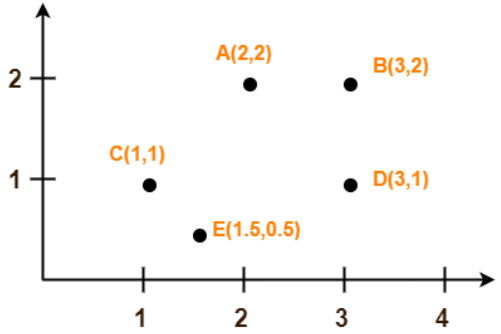
•C1(3, 9.5)

•C2(6.5, 5.25)

•C3(1.5, 3.5)

Problem-2

Use K-Means Algorithm to create two clusters-



Module 5
Unsupervised Learning

K Mode Clustering

K Modes Clustering

k-MODES CLUSTERING ALGORITHM

The *k*-modes clustering algorithm is an extension of *k*-means clustering algorithm. The *k*-means algorithm is the most widely used centre based partitional clustering algorithm. Huang extends the *k*-means clustering algorithm to *k*-modes clustering algorithm to group the categorical data · The modifications done in the *k*-means are (i) using a simple matching dissimilarity measure for categorical objects, (ii) replacing means of clusters by modes, and (iii) using a frequency-based method to update the modes.

Let X, x_{11}, x_{12},....x_{nm} be the data set consists of n number of objects with m number of attributes. The main objective of the *k*-modes clustering algorithm is to group the data objects X into K clusters by minimize the cost function Eq.(1).

K Modes Clustering

Input: Data objects X, Number of clusters K.

Step 1: Randomly select the K initial modes from the data objects such that $C_j, j = 1,2,\ldots,K$

Step 2: Find the matching dissimilarity between the each K initial cluster modes and each data objects using the Eq.(2).

Step 3: Evaluate the fitness using the Eq.(1)

Step 4: Find the minimum mode values in each data object i.e. finding the objects nearest to the initial cluster modes.

Step 5: Assign the data objects to the nearest cluster centroid modes.

Step 6: Update the modes by apply the frequency based method on newly formed clusters.

Step 7: Recalculate the similarity between the data objects and the updated modes.

Step 8: Repeat the step 4 and step 5 until no changes in the cluster ship of data objects.

Output: Clustered data objects.

Fig.1. *k*-modes algorithm

K Modes Clustering

$$P(W,Q) = \sum_{l=1}^{k}\sum_{i=1}^{n} w_{il} d_{sim}(x_i, q_l) \quad (1)$$

where, w_{il} is an $N \times K$ matrix where each element belongs to 0 or 1. N is the total number data objects and K is the number of clusters. $d_{sim}(x_i,q_l)$ is the simple dissimilarity measure and it is defined in the following Eq.(2).

$$d_{sim}(x_i, q_i) = \sum_{j=1}^{m} \delta(x_{ij}, z_{lj}) \quad (2)$$

where, $\delta(x_{ij}, q_{lj})$ is calculated using the following Eq.(3)

$$\delta(x_{ij}, z_{lj}) = \begin{cases} 1 & if\ x_{ij} = z_{lj} \\ 0 & if\ x_{ij} \neq z_{lj} \end{cases} \quad (3)$$

The *k*-modes clustering algorithm is described in Fig.1.

K-Modes Clustering – Example 1

- Consider the given dataset:

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
2	AB	BB	AB	AA	AB	BB
3	AA	AB	AA	AB	AA	AB
4	BB	AA	BB	AB	AA	BB
5	AB	AA	AB	BB	BB	BB
6	AA	AB	BB	AA	AB	BB
7	BB	BB	AA	AB	AA	AB
8	AB	AB	AA	AB	BB	AB

- **Let K be 2**
- Let tuples 1 and 5 be the initial centroids (chosen randomly) of the two clusters respectively.
- Centroids:

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
5	AB	AA	AB	BB	BB	BB

Example 1 (cont'd)

- Let us compute the distance from each tuple to the two cluster centroids as given below by d(X,Y):

$$d(X,Y) = \sum_{j=1}^{m} \delta(x_j, y_j)$$

where

$$\delta(x_j, y_j) = \begin{cases} 0, & x_j = y_j \\ 1, & x_j \neq y_j \end{cases}$$

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6	Distance to Cluster1	Distance to Cluster2	Cluster Assignment
1	AA	BB	AB	AA	AB	AB	0	5	C1
2	AB	BB	AB	AA	AB	BB	2	3	C1
3	AA	AB	AA	AB	AA	AB	4	6	C1
4	BB	AA	BB	AB	AA	BB	5	4	C2
5	AB	AA	AB	BB	BB	BB	5	0	C2
6	AA	AB	BB	AA	AB	BB	3	5	C1
7	BB	BB	AA	AB	AA	AB	4	6	C1
8	AB	AB	AA	AB	BB	AB	5	4	C2

Centroids

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
5	AB	AA	AB	BB	BB	BB

- The distance from the tuple to the closest cluster is given in red color. Hence
- **Tuples 1,2,3,6 and 7 will fall in cluster 1.**
- **Tuples 4,5 and 8 will fall in cluster 2.**

Example 1 (cont'd – new centroid1)

- Let us compute the distance from each tuple to the two cluster centroids as given below by d(X,Y):
- Cluster 1 given below:

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
2	AB	BB	AB	AA	AB	BB
3	AA	AB	AA	AB	AA	AB
6	AA	AB	BB	AA	AB	BB
7	BB	BB	AA	AB	AA	AB
mode	AA	BB	AA	AA	AB	AB

New centroid 1

Centroids

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
5	AB	AA	AB	BB	BB	BB

- Note: x2 - 2 values are equal frequency, hence chosen as a different from old centroid

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
New C1	AA	BB	AA	AA	AB	AB
5	AB	AA	AB	BB	BB	BB

Example 1 (cont'd – new centroid2)

- Let us compute the distance from each tuple to the two cluster centroids as given below by d(X,Y):

$$d(X,Y) = \sum_{j=1}^{m} \delta(x_j, y_j)$$

where

$$\delta(x_j, y_j) = \begin{cases} 0, & x_j = y_j \\ 1, & x_j \neq y_j \end{cases}$$

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
4	BB	AA	BB	AB	AA	BB
5	AB	AA	AB	BB	BB	BB
8	AB	AB	AA	AB	BB	AB
mode	AB	AA	BB	AB	BB	BB

New centroid 2

Centroids

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AB	AA	AB	AB
5	AB	AA	AB	BB	BB	BB

- Mode –major frequency value among the feature

  Note: x3- all are equal , hence chosen as a different from old centroid
- New Centroid of new clusters

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AA	AA	AB	AB
5	AB	AA	BB	AB	BB	BB

Example 1 (cont'd)

- Let the centroids (modes) of two clusters be updated with reference to the tuples currently assigned to the clusters.
- New Centroids:

Cluster No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AA	AA	AB	AB
2	AB	AA	BB	AB	BB	BB

- Let us compute the updated distance from each tuple to the two cluster centroids as given below:

Tuple No.	X_1	X_2	X_3	X_4	X_5	X_6	Distance to Cluster1	Distance to Cluster2	Cluster Assignment
1	AA	BB	AB	AA	AB	AB	1	6	C1
2	AB	BB	AB	AA	AB	BB	3	4	C1
3	AA	AB	AA	AB	AA	AB	3	5	C1
4	BB	AA	BB	AB	AA	BB	6	2	C2
5	AB	AA	AB	BB	BB	BB	6	2	C2
6	AA	AB	BB	AA	AB	BB	3	4	C1
7	BB	BB	AA	AB	AA	AB	3	5	C1
8	AB	AB	AA	AB	BB	AB	4	3	C2

Example 1 (cont'd)

- By following the updated distance , updated Clusters are given below:
  - Cluster 1 = { 1,2,3,6 and 7}
  - Cluster 2 ={ 4,5 and 8}
- Let the centroids of two clusters be updated with reference to the tuples currently assigned to the clusters.
- New Centroids:

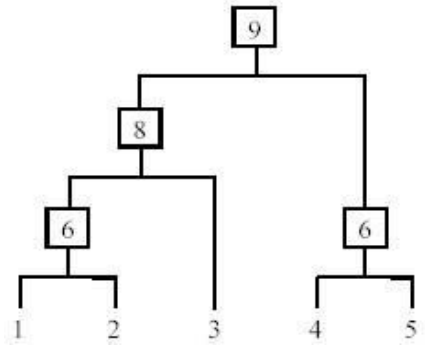
Cluster No.	X_1	X_2	X_3	X_4	X_5	X_6
1	AA	BB	AA	AA	AB	AB
2	AB	AA	BB	AB	BB	BB

- The current state of the two centroids is same as the previous centroids. Hence the membership of the two clusters will not change anymore.
- **Conclusion:**
  - Cluster 1 = { 1,2,3,6 and 7}
  - Cluster 2 ={ 4,5 and 8}

Hierarchical Clustering

Hierarchical Clustering

- Produce a nested sequence of clusters, a tree, also called Dendrogram.



Types of hierarchical clustering

- Agglomerative (bottom up) clustering: It builds the dendrogram (tree) from the bottom level, and
  - merges the most similar (or nearest) pair of clusters
  - stops when all the data points are merged into a single cluster (i.e., the root cluster).
- Divisive (top down) clustering: It starts with all data points in one cluster, the root.
  - Splits the root into a set of child clusters. Each child cluster is recursively divided further
  - stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

Agglomerative clustering

It is more popular then divisive methods.

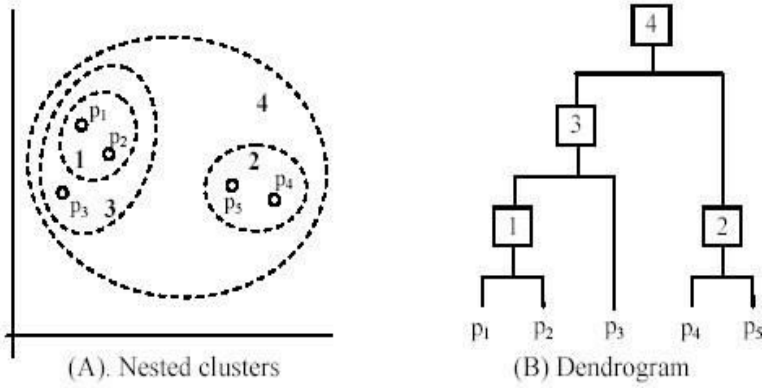
- At the beginning, each data point forms a cluster (also called a node).
- Merge nodes/clusters that have the least distance.
- Go on merging
- Eventually all nodes belong to one cluster

Agglomerative clustering algorithm

Algorithm Agglomerative(D)

1 Make each data point in the data set D a cluster,
2 Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in D$;
2 **repeat**
3 find two clusters that are nearest to each other;
4 merge the two clusters form a new cluster c;
5 compute the distance from c to all other clusters;
12 **until** there is only one cluster left

An example: working of the algorithm



(A). Nested clusters

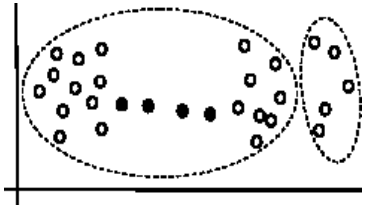
(B) Dendrogram

Measuring the distance of two clusters

- A few ways to measure distances of two clusters.
- Results in different variations of the algorithm.
  - Single link
  - Complete link
  - Average link
  - Centroids

Single link method

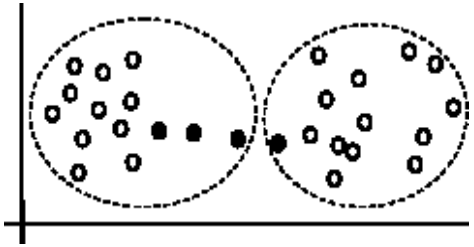
- The distance between two clusters is the distance between two closest data points in the two clusters, one data point from each cluster.
- It can find arbitrarily shaped clusters, but
  - It may cause the undesirable "chain effect" by noisy points



Two natural clusters are split into two

Complete link method

- The distance between two clusters is the distance of two furthest data points in the two clusters.
- It is sensitive to outliers because they are far away



Average link and centroid methods

- Average link: A compromise between
  - the sensitivity of complete-link clustering to outliers and the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.
  - In this method, the distance between two clusters is the average distance of all pair-wise distances between the data points in two clusters.
- Centroid method: In this method, the distance between two clusters is the distance between their centroids

The complexity

- All the algorithms are at least $O(n^2)$. n is the number of data points.
- Single link can be done in $O(n^2)$.
- Complete and average links can be done in $O(n^2 \log n)$.
- Due the complexity, hard to use for large data sets.
  - Sampling
  - Scale-up methods (e.g., BIRCH).

Problem :

- Find the Clusters using Single, Complete, Average Link Technique. Use Euclidian distance and draw the dendrogram

	X	Y
P1	0.40	0.53
P2	0.22	0.38
P3	0.35	0.32
P4	0.26	0.19
P5	0.08	0.41
P6	0.45	0.30

Step : 1

Calculate Euclidean distance, create the distance matrix.

Distance [(x,y), (a,b)] = $\sqrt{(x-a)^2+(x-b)^2}$

Distance (P1,P2) = $\sqrt{(0.40-0.22)^2+(0.53-0.38)^2}$

(0.40,0.53), (0.22,0.38) = $\sqrt{(0.18)^2+(0.15)^2}$

= $\sqrt{0.0324+0.0225}$

= $\sqrt{0.0549}$

= 0.23

- The Distance matrix is calculated for all the Points and shown below

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

- The Diagonal is Zero and same value will be obtained in the Upper and Lower Bounds.
- Take Lower bound for the Updating the Clusters

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

Step:2

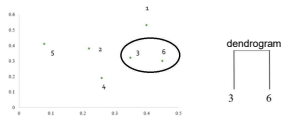
Find the Minimum Element from the Cluster Matrix

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.24	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

- The Minimum element obtained is (P3,P6)

Step:3 Cluster Formation

- The First Cluster Formed is (P3,P6)
- Represent in dendrogram or in a Graph



Step 4:

- Update the Distance Matrix by using any of the three techniques asked
- Single Link : Minimum(Distance)
- Complete Link : Maximum(Distance)
- Average Link : Average (Distance)

Step:5

- **Using Single Link Method**

To update the distance matrix MIN[dist(P3,P6),P1)]

MIN(dist(P3,P1), (P6,P1))

= min[(0.22,0.23)]

= 0.22

To update the distance matrix MIN[dist(P3,P6),P2)]

MIN(dist(P3,P2), (P6,P2))

= min[(0.15,0.25)]

= 0.15

- To update the distance matrix MIN[dist(P3,P6),P4)]
- MIN(dist(P3,P4), (P6,P4))
  = min[(0.15,0.22)]
  = 0.15
- To update the distance matrix MIN[dist(P3,P6),P5)]
- MIN(dist(P3,P5), (P6,P5))
  = min[(0.28,0.39)]
  = 0.28

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

The updated distance matrix for cluster P3, P6

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.23	0			
P3,P6	0.22	0.15	0		
P4	0.37	0.20	0.15	0	
P5	0.34	0.14	0.28	0.29	0

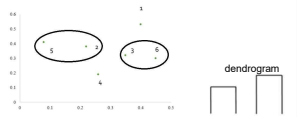
After First Cluster – Distance Matrix

The distance matrix is

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.23	0			
P3,P6	0.22	0.15	0		
P4	0.37	0.20	0.15	0	
P5	0.34	0.14	0.28	0.29	0

- Find the Minimum value in the lower bound

Cluster Formation



Update the Distance Matrix to find the New Cluster

To update the distance matrix MIN[dist(P2,P5),P1)]
MIN[dist(P2,P1), (P5,P1)]
= min[(0.23,0.34)]
= 0.23
To update the distance matrix MIN[dist(P2,P5),(P3,P6)]
MIN[dist(P2,(P3,P6)), (P5,(P3,P6))]
= min[(0.15,0.28)]
= 0.15

To update the distance matrix MIN[dist(P2,P5),P4)]
MIN[dist(P2,P4), (P5,P4)]
= min[(0.20,0.29)]
= 0.20

	P1	P2,P5	P3,P6	P4
P1	0			
P2,P5	0.23	0		
P3,P6	0.22	0.15	0	
P4	0.37	0.20	0.15	0

Choose the First Sequence minimum value to form the cluster

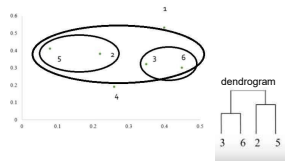
The distance matrix is

	P1	P2,P5	P3,P6	P4
P1	0			
P2,P5	0.23	0		
P3,P6	0.22	0.15	0	
P4	0.37	0.20	0.15	0

The distance matrix for cluster (P3,P6), (P2,P5)

	P1	P2,P5	P3,P6	P4
P1	0			
P2,P5	0.23	0		
P3,P6	0.22	0.15	0	
P4	0.37	0.20	0.15	0

Cluster Formation



Update the Distance Matrix to find the New Cluster

To update the distance matrix MIN[dist((P2,P5),(P3,P6)),P1]

MIN[dist((P2,P5),P1), ((P3,P6),P1)]

= min[(0.23,0.22)]

= 0.22

Update the Distance Matrix to find the New Cluster

To update the distance matrix MIN[dist((P2,P5),(P3,P6)),P4]

MIN[dist((P2,P5),P4), ((P3,P6),P4)]

= min[(0.20,0.15)]

= 0.15

The updated distance matrix for cluster P2,P5,P3,P6

	P1	P2,P5,P3,P6	P4
P1	0		
P2,P5,P3,P6	0.22	0	
P4	0.37	0.15	0

Choose the First Sequence minimum value to form the cluster

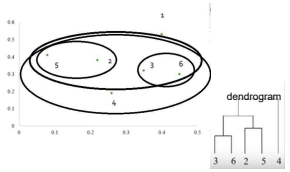
The distance matrix is

	P1	P2,P5,P3,P6	P4
P1	0		
P2,P5,P3,P6	0.22	0	
P4	0.37	0.15	0

The distance matrix is

	P1	P2,P5,P3,P6	P4
P1	0		
P2,P5,P3,P6	0.22	0	
P4	0.37	0.15	0

Cluster Formation



Update the Distance Matrix to find the New Cluster

To update the distance matrix MIN[dist(P2,P5,P3,P6),P4]

MIN[dist((P2,P5,P3,P6),P1), (P4,P1)]

= min[(0.22,0.37)]

= 0.22

COMPLETE LINK

- The Distance matrix is calculated for all the Points and shown below

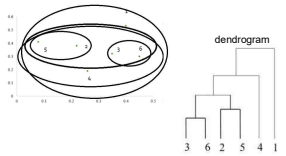
The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

The updated distance matrix for cluster P2,P5,P3,P6,P4

	P1	P2,P5,P3,P6,P4
P1	0	
P2,P5,P3,P6,P4	0.22	0

Final Clustering



- The Diagonal is Zero and same value will be obtained in the Upper and Lower Bounds.
- Take Lower bound for the Updating the Clusters

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

Step:2

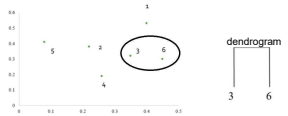
Find the Minimum Element from the Cluster Matrix

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.24	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

- The Minimum element obtained is (P3,P6)

Step:3 Cluster Formation

- The First Cluster Formed is (P3,P6)
- Represent in dendrogram or in a Graph



Update Matrix using Complete Link Method

To update the distance matrix MAX[dist(P3,P6),P1)]

MAX(dist(P3,P1), (P6,P1))

= MAX[(0.22,0.23)]

= 0.23

To update the distance matrix MAX[dist(P3,P6),P2)]

MAX(dist(P3,P2), (P6,P2))

= MAX[(0.15,0.25)]

= 0.25

To update the distance matrix MAX[dist(P3,P6),P4)]

MAX(dist(P3,P4), (P6,P4))

= MAX[(0.15,0.22)]

= 0.22

To update the distance matrix MAX[dist(P3,P6),P5)]

MAX(dist(P3,P5), (P6,P5))

= MAX[(0.28,0.39)]

= 0.39

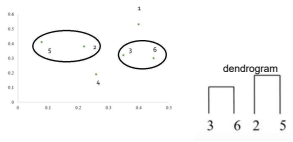
The updated distance matrix for cluster P3,P6

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.23	0			
P3,P6	0.23	0.25	0		
P4	0.37	0.20	0.22	0	
P5	0.34	0.14	0.39	0.29	0

The distance matrix

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.23	0			
P3,P6	0.23	0.25	0		
P4	0.37	0.20	0.22	0	
P5	0.34	0.14	0.39	0.29	0

Cluster Formation



The distance matrix , cluster (P2,P5)

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.23	0			
P3,P6	0.23	0.25	0		
P4	0.37	0.20	0.22	0	
P5	0.34	0.14	0.39	0.29	0

To update the distance matrix MAX[dist(P2,P5),P1)]

MAX[dist(P2,P1), (P5,P1)]

= MAX[(0.23,0.34)]

= 0.34

To update the distance matrix MAX[dist(P2,P5),(P3,P6)]

MAX[dist(P2,(P3,P6)), (P5,(P3,P6))]

= MAX[(0.25,0.39)]

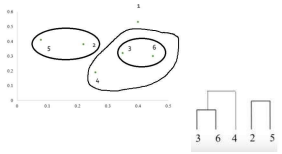
= 0.39

To update the distance matrix MAX[dist(P2,P5),P4)]

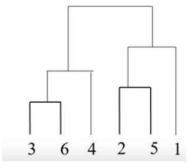
MAX[dist(P2,P4), (P5,P4)]

= MAX[(0.20,0.29)]

= 0.29



Final Dendrogram



The updated distance matrix for cluster (P2,P5)

	P1	P2,P5	P3,P6	P4
P1	0			
P2,P5	0.34	0		
P3,P6	0.23	0.39	0	
P4	0.37	0.29	0.22	0

The distance matrix is

	P1	P2,P5	P3,P6	P4
P1	0			
P2,P5	0.34	0		
P3,P6	0.23	0.39	0	
P4	0.37	0.29	0.22	0

Average Link

- The Distance matrix is calculated for all the Points and shown below

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

- The Diagonal is Zero and same value will be obtained in the Upper and Lower Bounds.
- Take Lower bound for the Updating the Clusters

The distance matrix is

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.23	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

Step:2
Find the Minimum Element from the Cluster Matrix

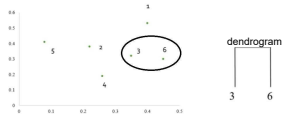
	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.24	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	0.11	0.22	0.39	0

- The Minimum element obtained is (P3,P6)

25

Step:3 Cluster Formation

- The First Cluster Formed is (P3,P6)
- Represent in dendrogram or in a Graph



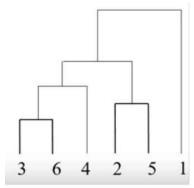
Update Matrix using Complete Link Method

The distance matrix is, AVG[dist(P3,P6),P1]

dist((P3,P6),P1) = ½ (dist(P3,P1)+dist(P6,P1)
= ½ (0.22+0.23)
= ½ (0.45)
= 0.23

26

Final Dendrogram



27

Distance Measures

- The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.
- Similar points are "close."
- Two major classes of distance measure:
  1. *Euclidean*
  2. *Non-Euclidean*

Euclidean Vs. Non-Euclidean

- A *Euclidean space* has some number of real-valued dimensions and "dense" points.
  - There is a notion of "average" of two points.
  - A *Euclidean distance* is based on the locations of points in such a space.
- A *Non-Euclidean distance* is based on properties of points, but not their "location" in a space.

Axioms of a Distance Measure

- *d* is a *distance measure* if it is a function from pairs of points to real numbers such that:
  1. d(x,y) $\geq$ 0.
  2. d(x,y) = 0 iff x = y.
  3. d(x,y) = d(y,x).
  4. d(x,y) $\leq$ d(x,z) + d(z,y) (*triangle inequality*).

Some Euclidean Distances

- L_2 *norm* : d(x,y) = square root of the sum of the squares of the differences between *x* and *y* in each dimension.
  - The most common notion of "distance."
- L_1 *norm* : sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.

Another Euclidean Distance

- L_∞ *norm* : d(x,y) = the maximum of the differences between *x* and *y* in any dimension.
- Note: L∞-norm, which is the limit as r approaches infinity of the Lr-norm. As r gets larger, only the dimension with the largest difference matters, so formally, the L∞-norm is defined as the maximum of |xi −yi| over all dimensions i.

example

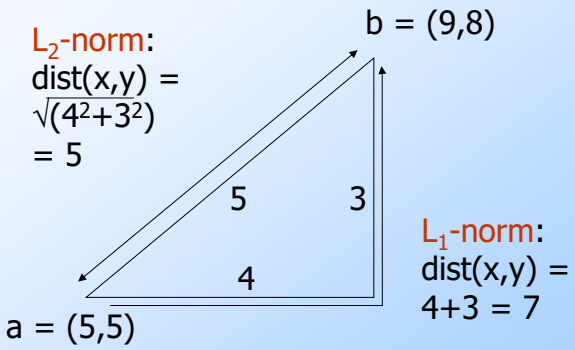
Consider the two-dimensional Euclidean space (the customary plane) and the points (2,7) and (6,4).

The **L2-norm** gives a distance of p(2−6)2 + (7−4)2 = √42 + 32 = 5.
square the distance in each dimension, sum the squares, and take the positive square root

The **L1-norm** gives a distance of |2−6|+|7−4| = 4 + 3 = 7.
sum of the magnitudes of the differences in each dimension

The **L∞-norm** gives a distance of max(|2−6|,|7−4|) = max(4,3) = 4.
Max. of the magnitudes of the differences in each dimension

Examples of Euclidean Distances



Non-Euclidean Distances

- *Jaccard distance* for sets = 1 minus Jaccard similarity.
- *Cosine distance* = angle between vectors from the origin to the points in question.
- *Edit distance* = number of inserts and deletes to change one string into another.
- *Hamming Distance* = number of positions in which bit vectors differ.

M_{11} represents the total number of attributes where *A* and *B* both have a value of 1.

M_{01} represents the total number of attributes where the attribute of *A* is 0 and the attribute of *B* is 1.

M_{10} represents the total number of attributes where the attribute of *A* is 1 and the attribute of *B* is 0.

M_{00} represents the total number of attributes where *A* and *B* both have a value of 0.

Each attribute must fall into one of these four categories, meaning that

$$M_{11} + M_{01} + M_{10} + M_{00} = n.$$

The Jaccard similarity coefficient, J, is given as

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}.$$

The Jaccard distance, d_J, is given as

$$d_J = \frac{M_{01} + M_{10}}{M_{01} + M_{10} + M_{11}} = 1 - J.$$

		A	
		0	1
B	0	M_{00}	M_{10}
	1	M_{01}	M_{11}

Jaccard Distance for Sets (Bit-Vectors)

- Example: p_1 = 10111; p_2 = 10011.
- Size of intersection = 3;
- size of union = 4,
- Jaccard similarity (not distance) = 3/4.
- d(x,y) = 1 – (Jaccard similarity) = 1/4.

Cosine Distance

- Think of a point as a vector from the origin (0,0,…,0) to its location.
- Two points' vectors make an angle, whose cosine is the normalized dot-product of the vectors: $p_1.p_2/|p_2||p_1|$.
  - Example: p_1 = 00111; p_2 = 10011.
  - $p_1.p_2$ = 2; $|p_1|$ = $|p_2|$ = $\sqrt{3}$.
  - cos(θ) = 2/3; θ is about 48 degrees.

Let our two vectors be x = [1,2,−1] and = [2,1,1].

The dot product x.y is 1×2 + 2×1 + (−1)×1 = 3.

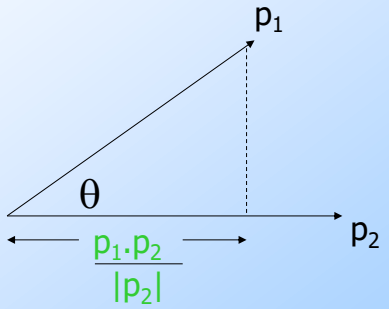
The L2-norm of both vectors is √6.

For example, x has L2-normp12 + 22 + (−1)2 =√6.

Thus, the cosine of the angle between x and y is 3/(√6√6) or 1/2.

The angle whose cosine is 1/2 is 60 degrees, so that is the cosine distance between x and y.

Cosine-Measure Diagram



d (p_1, p_2) = θ = arccos($p_1.p_2/|p_2||p_1|$)

Edit Distance

- The *edit distance* of two strings is the number of inserts and deletes of characters needed to turn one into the other. Equivalently:
- d(x,y) = |x| + |y| - 2|LCS(x,y)|.
  - LCS = *longest common subsequence* = any longest string obtained both by deleting from *x* and deleting from *y*.

Example: LCS

- *x* = *abcde* ; *y* = *bcduve*.
- Turn *x* into *y* by deleting *a*, then inserting *u* and *v* after *d*.
  - Edit distance = 3.
- Or, LCS(x,y) = *bcde*.
- Note: |x| + |y| - 2|LCS(x,y)| = 5 + 6 –2*4 = 3 = edit distance.

Hamming Distance

- *Hamming distance* is the number of positions in which bit-vectors differ.
- Example: p_1 = 10101; p_2 = 10011.
- d(p_1, p_2) = 2 because the bit-vectors differ in the 3rd and 4th positions.

correlation-based distances

1. Pearson correlation distance:

$$d_{cor}(x,y) = 1 - \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Spearman correlation distance

The spearman correlation method computes the correlation between the rank of x and the rank of y variables.

$$d_{spear}(x,y) = 1 - \frac{\sum_{i=1}^{n}(x_i' - \bar{x'})(y_i' - \bar{y'})}{\sqrt{\sum_{i=1}^{n}(x_i' - \bar{x'})^2 \sum_{i=1}^{n}(y_i' - \bar{y'})^2}}$$

Where $x_i' = rank(x_i)$ and $y_i' = rank(y)$.

KOHONEN SELF ORGANIZING MAPS (KSOM)

History of Kohonen SOM

- Developed in 1982 by Tuevo Kohonen, a professor emeritus of the Academy of Finland
- Professor Kohonen worked on auto-associative memory during the 70s and 80s and in 1982 he presented his self-organizing map algorithm

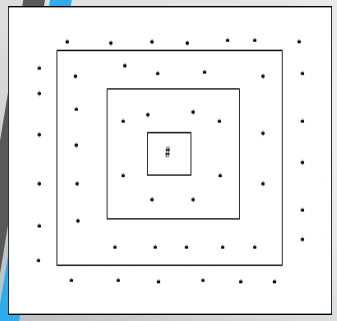
  His idea on Kohonen SOM only became famous much later in 1988 when he presented a paper on "The Neural Phonetic Typewriter" on IEEE computer that his work became widely known
- Since then many excellent papers and books have been made on SOM

What are self organizing maps?

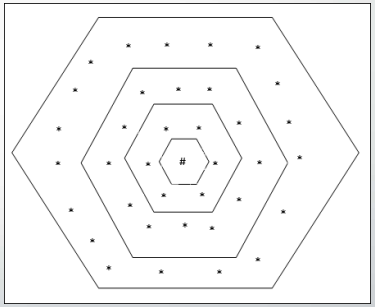
- Are aptly named "Self-Organizing" because no supervision is required.
- SOMs learn on their own through unsupervised competitive learning.
- They attempt to *map* their weights to conform to the given input data.

Suppose we have some pattern of arbitrary dimensions, however, we need them in one dimension or two dimensions.
Then the process of feature mapping would be very useful to convert the wide pattern space into a typical feature space.
Now, the question arises why do we require self-organizing feature map?
The reason is, along with the capability to convert the arbitrary dimensions into 1-D or 2-D, it must also have the ability to preserve the neighbor topology.

Neighbor Topologies in Kohonen SOM



Rectangular Grid Topology
This topology has 24 nodes in the distance-2 grid, 16 nodes in the distance-1 grid, and 8 nodes in the distance-0 grid, which means the difference between each rectangular grid is 8 nodes. The winning unit is indicated by #.



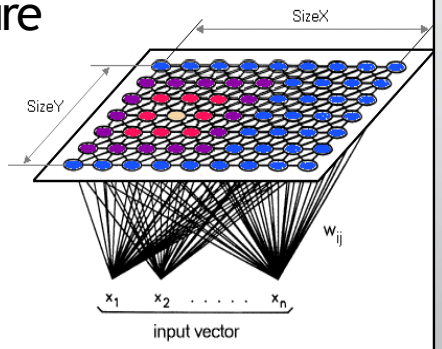
Hexagonal Grid Topology
This topology has 18 nodes in the distance-2 grid, 12 nodes in the distance-1 grid, and 6 nodes in the distance-0 grid, which means the difference between each **Hexagonal** grid is 6 nodes. The winning unit is indicated by #.

What are self organizing maps?

- *Thus SOMs are neural networks that employ unsupervised learning methods, mapping their weights to conform to the given input data with a goal of representing multidimensional data in an easier and understandable form for the human eye.* (pragmatic value of representing complex data)

The Architecture

- Training a SOM requires no target vector.
- A SOM learns to classify the training data without any external supervision.



- Made up of an input nodes and computational nodes.
- Each computational node is connected to each input node to form a lattice.
- There are no interconnections among the computational nodes.
- The number of input nodes is determined by the dimensions of the input vector.

Representing Data

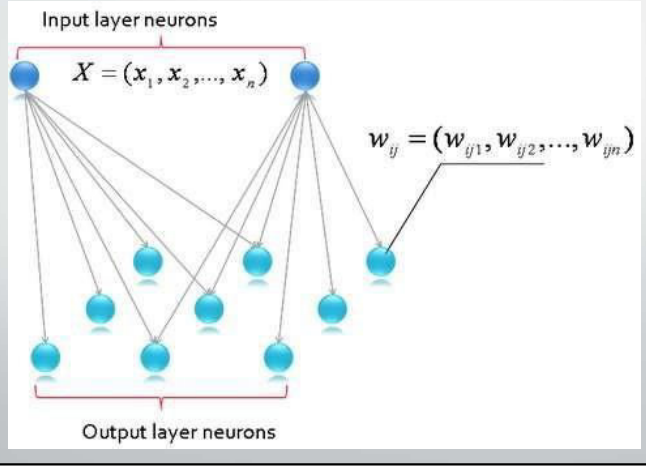
- Weight vectors are of the same dimension as the input vectors. If the training data consists of vectors, *V*, of *n* dimensions:

$$V_1, V_2, V_3 \ldots V_n$$

- Then each node will contain a corresponding weight vector *W*, of *n* dimensions:

$$W_1, W_2, W_3 \ldots W_n$$

A sample SOM



Terms used inSOMs

- **vector quantization** - This is a data compression technique. SOMs provide a way of representing multidimensional data in a much lower dimensional space; typically one or two dimensions

Kohonen Self-Organizing Maps (SOM)

Algorithm:

Step 0. Initialise weights w_{ij} (randomly)
Set topological neighbourhood parameters
Set learning rate parameters

Step 1 While stopping condition is false, do steps 2-8.

Step 2 For each i/p vector **x**, do steps 3-5

Step 3 For each *j*, compute: $D(J) = \sum_i (w_{ij} - x_i)^2$

Step 4 Find index *J* such that *D(J)* is a minimum

Step 5 For all units *j* within a specified neighbourhood of *J*, and for all *i*:
$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$

Step 6 Update learning rate.

Step 7 Reduce radius of topological neighbourhood at specified time

Step 8 Test stopping condition

The SOM Algorithm

The aim is to learn a ***feature map*** from the spatially ***continuous input space***, in which our input vectors live, to the low dimensional spatially ***discrete output space***, which is formed by arranging the computational neurons into a grid.

The stages of the SOM algorithm that achieves this can be summarised as follows:

1. ***Initialization*** – Choose random values for the initial weight vectors $\mathbf{w}_j$.
2. ***Sampling*** – Draw a sample training input vector **x** from the input space.
3. ***Matching*** – Find the winning neuron $I(\mathbf{x})$ that has weight vector closest to the input vector, i.e. the minimum value of $d_j(\mathbf{x}) = \sum_{i=1}^{D} (x_i - w_{ji})^2$.
4. ***Updating*** – Apply the weight update equation $\Delta w_{ji} = \eta(t)\, T_{j,I(\mathbf{x})}(t)\, (x_i - w_{ji})$ where $T_{j,I(\mathbf{x})}(t)$ is a Gaussian neighbourhood and $\eta(t)$ is the learning rate.
5. ***Continuation*** – keep returning to step 2 until the feature map stops changing.

EXPLANATION: **How Kohonen SOMs work**

The SOM Algorithm

- The Self-Organizing Map algorithm can be broken up into 6 steps
- **1). Each node's weights are initialized.**
- **2). A vector is chosen at random from the set of training data and presented to the network.**

EXPLANATION: **The SOM Algorithm…**

3). Every node in the network is examined to calculate which ones' weights are most like the input vector. The winning node is commonly known as the *Best Matching Unit* (BMU).

Best Matching Unit is a technique which calculates the distance from each weight to the sample vector, by running through all weight vectors. The weight with the shortest distance is the winner. There are numerous ways to determine the distance, however, the most commonly used method is the *Euclidean Distance*

4). The radius of the neighbourhood of the BMU is calculated. This value starts large. Typically it is set to be the radius of the network, diminishing each time- step.

EXPLANATION: **The SOM Algorithm…**

- **5). Any nodes found within the radius of the BMU, calculated in 4), are adjusted to make them more like the input vector (Equation 3a, 3b). The closer a node is to the BMU, the more its' weights are altered**
- **6). Repeat 2) for N iterations.**

Computation of scores

- The Function for calculating the score inclusion for an output node is known as $\sqrt{\sum_i (ni - wij)\ 2}$
- Thus to calculate the score for inclusion with output node i:

Computation of scores…

- To calculate the score for inclusion with output node j:

$$\sqrt{(0.4-0.3)^2 + (0.7-0.6)^2} = 0.141$$

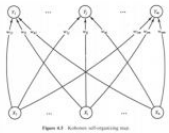
The Winning Node

- **Node j** becomes the winning node since it has the lowest score.
- This implies that its weight vector values are similar to the input values of the presented instance.
- i.e. The value of **node j** is closest to the input vector.
- As a result, the weight vectors associated with the winning node are adjusted so as to reward the node for winning the instance.

Kohonen Self-Organizing Maps (SOM)

Applications and Examples:

- The SOM have had may applications:
  - Computer generated music (Kohonen 1989)
  - Traveling salesman problem
- Examples: A Kohonen SOM to cluster 4 vectors:
  - Let the 4 vectors to be clustered be:



s(1) = (1, 1, 0, 0) s(2) = (0, 0, 0, 1)

s(3) = (1, 0, 0, 0) s(4) = (0, 0, 1, 1)

- The maximum number of clusters to be formed is m=2 (4 input nodes, 2 output nodes)

Suppose the learning rate is $\alpha(0)=0.6$ and $\alpha(t+1) = 0.5\,\alpha(t)$

- With only 2 clusters available, the neighborhood of node *J* (step 4) is set so that only one cluster update its weights at each step (i.e. $R=0$)

Kohonen Self-Organizing Maps (SOM)

Examples:

- Step 0 Initial weight matrix

$$W = \begin{bmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{bmatrix}$$

Initial radius: $R=0$

Initial learning rate: $\alpha(0)=0.6$

- Step 1 Begin training
  - Step 2 For s(1) = (1, 1, 0, 0) , do steps 3-5
    - Step 3

$$D(1) = (.2-1)^2 + (.6-1)^2 + (.5-0)^2 + (.9-0)^2 = 1.86$$

$$D(2) = (.8-1)^2 + (.4-1)^2 + (.7-0)^2 + (.3-0)^2 = 0.98$$

Kohonen Self-Organizing Maps (SOM)

Examples:

- Step 4 The i/p vector is closest to o/p node 2, so J=2
- Step 5 The weights on the winning unit are updated

$$w_{i2}(new) = w_{i2}(old) + 0.6[x_i - w_{i2}(old)]$$
$$0.4w_{i2}(old) + 0.6x_i$$

This gives the weight matrix:

$$W = \begin{bmatrix} .2 & .92 \\ .6 & .76 \\ .5 & .28 \\ .9 & .12 \end{bmatrix}$$

This set of weights has not been modified.

Kohonen Self-Organizing Maps (SOM)

Examples:

- Step 2 For s(2) = (0, 0, 0, 1) , do steps 3-5
- Step 3

$$D(1) = (.2-0)^2 + (.6-0)^2 + (.5-0)^2 + (.9-1)^2 = 0.66$$
$$D(2) = (.92-0)^2 + (.76-0)^2 + (.28-0)^2 + (.12-1)^2 = 2.2768$$

- Step 4 The i/p vector is closest to o/p node 1, so J=1
- Step 5 The weights on the winning unit are updated

This gives the weight matrix:

$$W = \begin{bmatrix} .08 & .92 \\ .24 & .76 \\ .20 & .28 \\ .96 & .12 \end{bmatrix}$$

This set of weights has not been modified.

Kohonen Self-Organizing Maps (SOM)

Examples:

- Step 2 For s(3) = (1, 0, 0, 0) , do steps 3-5
- Step 3

$$D(1) = (.08-1)^2 + (.24-0)^2 + (.2-0)^2 + (.96-0)^2 = 1.8656$$
$$D(2) = (.92-1)^2 + (.76-0)^2 + (.28-0)^2 + (.12-0)^2 = 0.6768$$

- Step 4 The i/p vector is closest to o/p node 2, so J=2
- Step 5 The weights on the winning unit are updated

This gives the weight matrix:

$$W = \begin{bmatrix} .08 & .968 \\ .24 & .304 \\ .20 & .112 \\ .96 & .048 \end{bmatrix}$$

This set of weights has not been modified.

Kohonen Self-Organizing Maps (SOM)

Examples:

- Step 2 For s(4) = (0, 0, 1, 1) , do steps 3-5
- Step 3

$$D(1) = (.08-0)^2 + (.24-0)^2 + (.2-1)^2 + (.96-1)^2 = 0.7056$$
$$D(2) = (.968-0)^2 + (.304-0)^2 + (.112-1)^2 + (.048-1)^2 = 2.724$$

- Step 4 The i/p vector is closest to o/p node 1, so J=1
- Step 5 The weights on the winning unit are updated

This gives the weight matrix:

$$W = \begin{bmatrix} .032 & .968 \\ .096 & .304 \\ .680 & .112 \\ .984 & .048 \end{bmatrix}$$

This set of weights has not been modified.

- Step 6 Reduce the learning rate $\alpha = 0.5(0.6) = 0.3$

Kohonen Self-Organizing Maps (SOM)

Examples:

- The weights update equations are now

$$w_{ij}(new) = w_{ij}(old) + 0.3[x_i - w_{ij}(old)]$$
$$0.7w_{ij}(old) + 0.3x_i$$

- The weight matrix after the 2nd epoch of training is:

$$W = \begin{bmatrix} .016 & .980 \\ .047 & .360 \\ .630 & .055 \\ .999 & .024 \end{bmatrix}$$

This set of weights has not been modified.

Modifying the learning rate and after 100 iterations (epochs), the weight matrix appears to be converging to:

$$W = \begin{bmatrix} .0 & 1.0 \\ .0 & 0.5 \\ .5 & 0.0 \\ 1.0 & 0.0 \end{bmatrix}$$

The 1st column is the average of the 2 vects in cluster 1, and the 2nd col is the average of the 2 vects in cluster 2

Concluding the tests

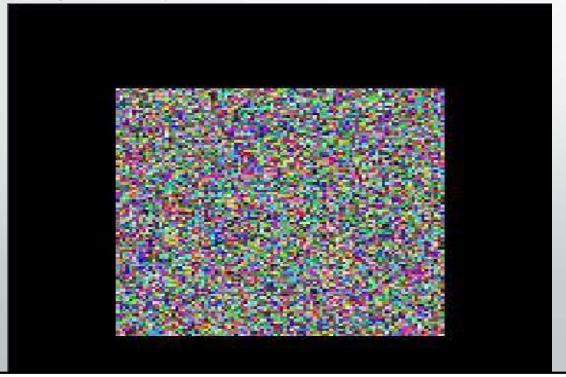
- Both of these are decreased linearly over the span of several iterations and terminates after instance classifications do not vary from one iteration to the next
- Finally the clusters formed by the training or test data are analysed in order to determine what has been discovered

NEIGHBORHOOD ADJUSMENTS

- After adjusting the weights of the winning node, the neighbourhood nodes also have their weights adjusted using the same formula
- A neighbourhood is typified by a square grid with the centre of the grid containing the winning node.
- The size of the neighbourhood as well as the learning rate r is specified when training begins

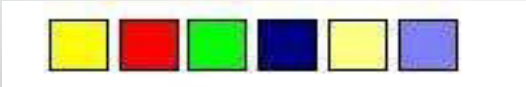
ILLUSTRATION: A Color Classifier

- Problem: Group and represent the primary colors and their corresponding shades on a two dimensional plane.



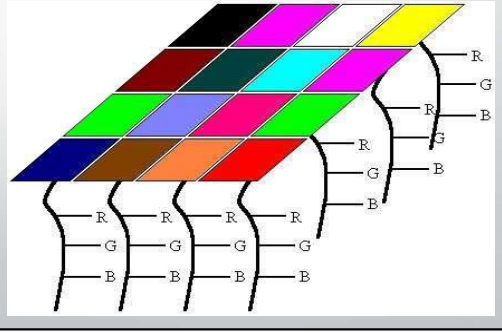
A Color classifier: Sample Data

- The colors are represented in their RGB values to form 3-dimensional vectors.



A Color classifier: **Node Weighting**

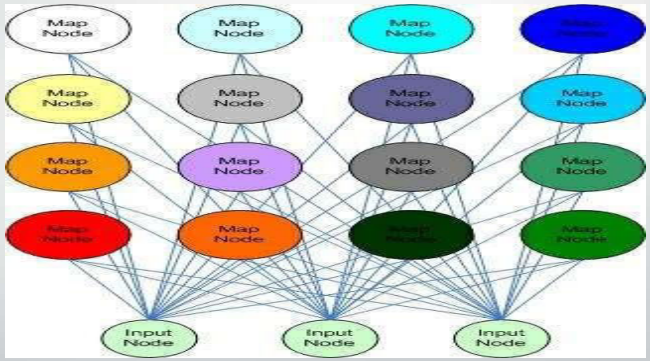
- Each node is characterized by:
  - Data of the same dimensions as the sample vectors
  - An X, Y position



A Color Classifier: **The algorithm**

Initialize Map
Radius = d
Learning rate = r
For **1** to **N** iterations
Randomly select a sample
Get best matching unit
Scale neighbors
Adjust **d**, **r** appropriately
End for

A Color classifier: The Layout



A Color classifier: **Getting a winner**

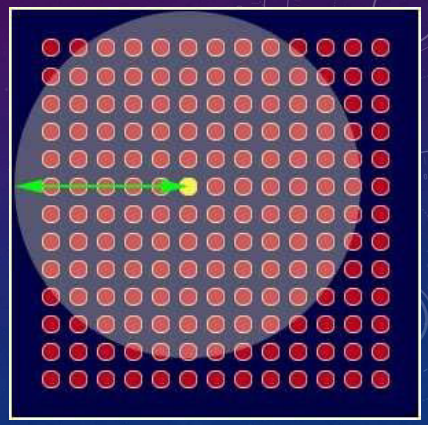
- Go through all the weight vectors and calculate the Euclidean distance from each weight to the chosen sample vector
- Assume the RGB values are represented by the values 0 – 6 depending on their intensity.
  - i.e. Red = (6, 0, 0); Green = (0, 6, 0); Blue = (0, 0, 6)

A Color classifier: **Getting a winner…**

- If we have colour green as the sample input instance, a probable node representing the colour light green (3,6,3) will be closer to green than red.
- Light green = Sqrt((3-0)^2+(6-6)^2+(3-0)^2) = 4.24
  Red = Sqrt((6-0)^2+(0-6)^2+(0-0)^2) = 8.49

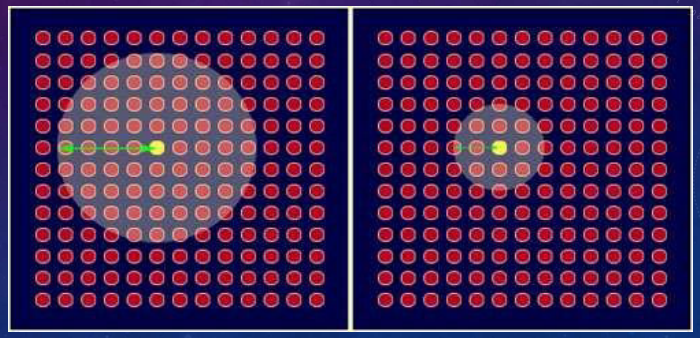
A COLOR CLASSIFIER: DETERMINING THE NEIGHBORHOOD

- Since a node has an X – Y position, it's neighbors can be easily determined based on their radial distance from the BMU coordinates.



A COLOR CLASSIFIER: DETERMINING THE NEIGHBORHOOD…

- The area of the neighbourhood shrinks over time with each iteration.



A Color classifier: **Learning**

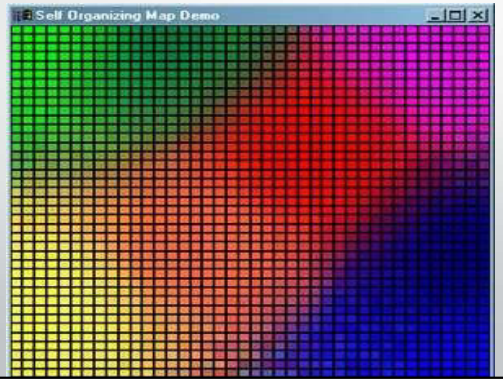
- Every node within the BMU's neighbourhood (including the BMU) has its weight vector adjusted according to a pre-determined equation
- The learning rate is decayed over time.

A Color classifier: **Learning**

- The effect of learning should be proportional to the distance a node is from the BMU.
- A Gaussian function can be used to achieve this, whereby the closest neighbors are adjusted the most to be more like the input vector.

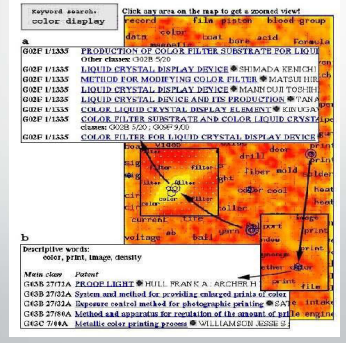
A Color classifier: **Sample output**

- Classification on a 40 x 40 SOM



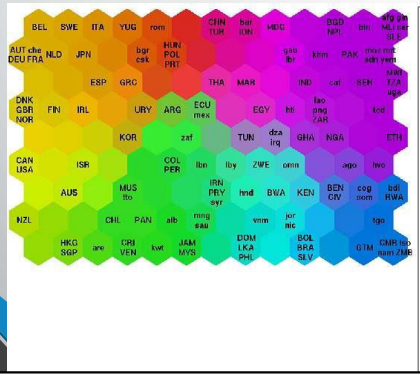
Current Applications

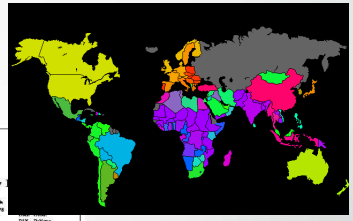
- WEBSOM: Organization of a Massive Document Collection



Current Applications...

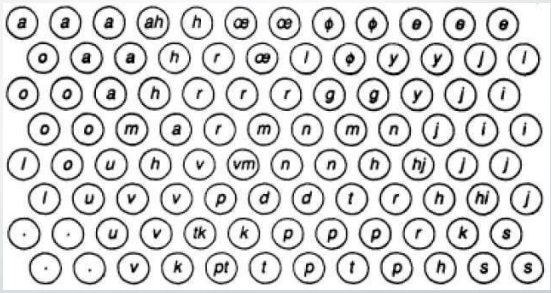
- Classifying World Poverty





Current Applications...

- Phonetic Typewriter



Ensembles of Classifiers

Different Classifiers

- Performance
  - None of the classifiers is perfect
  - Complementary
    - Examples which are not correctly classified by one classifier may be correctly classified by the other classifiers
- Potential Improvements?
  - Utilize the complementary property

Ensembles of Classifiers

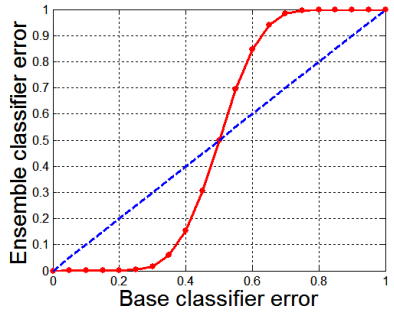
- Idea
  - Combine the classifiers to improve the performance
- Ensembles of Classifiers
  - Combine the classification results from different classifiers to produce the final output
    - Unweighted voting
    - Weighted voting

Ensembles of Classifiers

- **Basic idea** is to learn a set of classifiers (experts) and to allow them to vote.
- **Advantage:** improvement in predictive accuracy.
- **Disadvantage:** it is difficult to understand an ensemble of classifiers.

Why Majority Voting works?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume errors made by classifiers are uncorrelated
  - Probability that the ensemble classifier makes a wrong prediction:



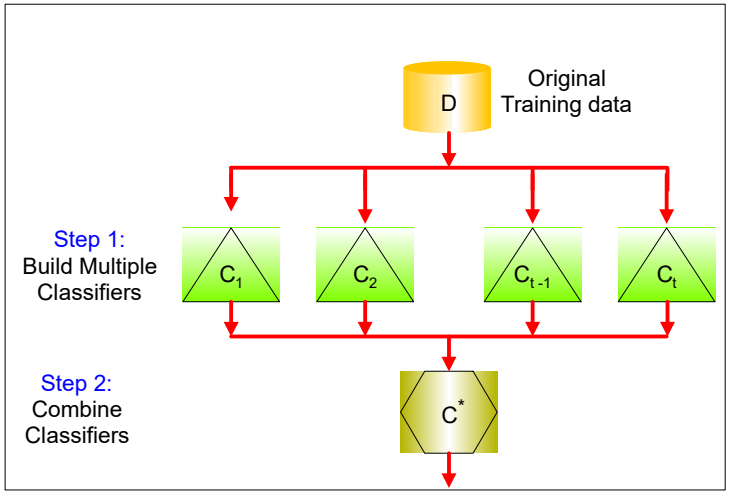
$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Methods for Independently Constructing Ensembles

One way to force a learning algorithm to construct multiple hypotheses is to run the algorithm several times and provide it with somewhat different data in each run. This idea is used in the following methods:

- ***Majority Voting***
- ***Bagging***
- ***Randomness Injection***
- ***Feature-Selection Ensembles***
- ***Error-Correcting Output Coding***

Majority Vote



Bagging

- Employs simplest way of combining predictions that belong to the same type.
- Combining can be realized with voting or averaging
- Each model receives equal weight
- "Idealized" version of bagging:
  - Sample several training sets of size *n* (instead of just having one training set of size *n*)
  - Build a classifier for each training set
  - Combine the classifier's predictions
- This improves performance in almost all cases if learning scheme is *unstable* (i.e. decision trees)

Why does bagging work?

- Bagging reduces variance by voting/ averaging, thus reducing the overall expected error
  - In the case of classification there are pathological situations where the overall error might increase
  - Usually, the more classifiers the better

Bagging classifiers

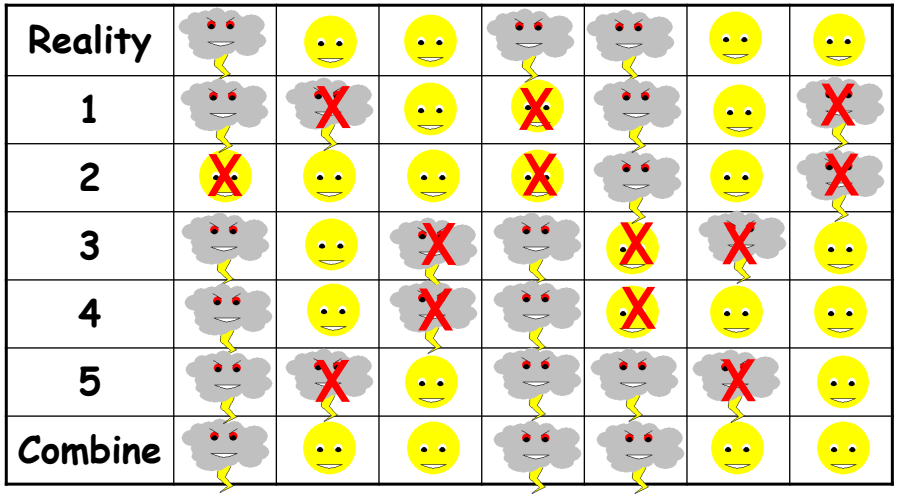
Classifier generation

```
Let n be the size of the training set.
For each of t iterations:
  Sample n instances with replacement from the
  training set.
  Apply the learning algorithm to the sample.
  Store the resulting classifier.
```

classification

```
For each of the t classifiers:
  Predict class of instance using classifier.
Return class that was predicted most often.
```

Example: Weather Forecast



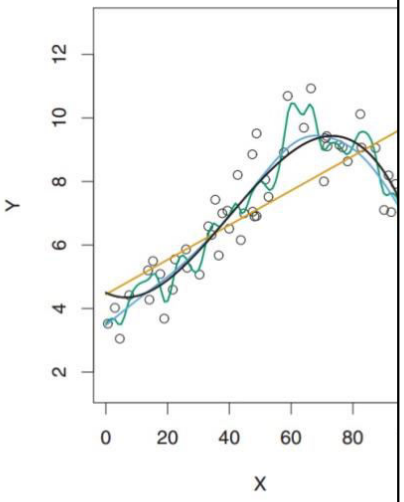
Bias-Variance Tradeoff

IIMB तेजस्वि नावधीत

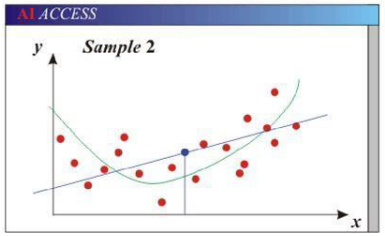
Expected test MSE

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon).$$

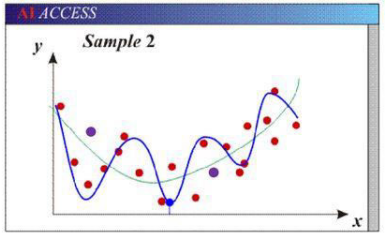
- Bias
  - Error that is introduced by approximating a complicated relationship, by a much simpler model.
  - Difference between the truth and what you expect to learn
  - Underfitting
- Variance
  - Amount by which model would change if we estimated it using a different training data.
  - If a model has high variance then small changes in the training data can result in large changes in the model.
  - Overfitting



Bias-Variance Trade-off



- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).



- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

Purple dots = possible test points

Red dots = training data (all that we see before we ship off our model!)

Green curve = true underlying model

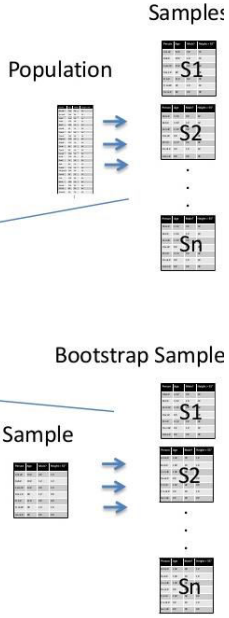
Blue curve = our predicted model/fit

Adapted from D. Hoiem

Bagging

IIMB तेजस्वि नावधीतमस्तु

- **Problem:** Decision tree have low bias & suffer from high variance
- **Goal:** Reduce variance of decision trees
- **Hint:** Given set of *n* independent observations *Z1, . . . , Zn*, each with variance *σ2*, the variance of the mean of the observations is given by *σ2/n*.
  - In other words, *averaging a set of observations reduces variance*.
- **Theoretically:** Take multiple independent samples S' from the population
  - Fit "bushy"/deep decision trees on each S1,S2…. Sn
    - Trees are grown deep and are not pruned
  - Variance reduces linearly & Bias remain unchanged
- **Practically:** We only have one sample/training set & not the population.
  - So take bootstrap samples i.e. multiple samples from the single sample with replacement
  - Variance reduces sub-linearly & Bias often increase slightly because bootstrap samples are correlated.
- **Final Classifier:** Average of predictions for regression or majority vote for classification.
  - High Variance introduced by deep decision trees are mitigated by averaging predictions from each decision trees.



Bootstrap sampling

Original Sample	1	2	3	4	5	6	7	8	9	10
Bootstrap Sample 1	6	3	1	4	10	2	4	4	5	1
Bootstrap Sample 2	5	8	5	1	7	4	6	8	8	8
Bootstrap Sample 3	5	10	9	8	4	4	9	10	6	7
Bootstrap Sample 4	1	5	4	4	8	10	8	2	7	2
Bootstrap Sample 5	5	6	1	3	2	5	8	3	3	3

Bootstrap sample should have same sample size as the original sample.

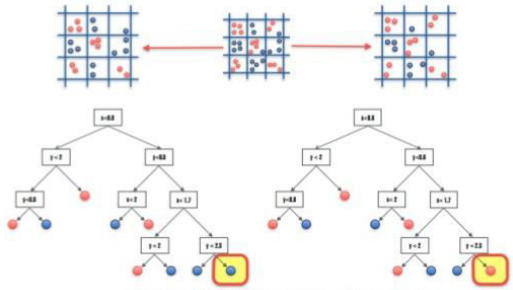
Bootstrap sample on an average uses only 2/3 of the data in the original sample

With replacement results in repetition of values

6

Random Forest

- **Problem:** Bagging still have relatively high variance
- **Goal:** Reduce variance of Bagging
- **Solution:** Along with sampling of data in Bagging, take samples of features also!
  - In other words, in building a random forest, at each split in the tree, the use only a random subset of features instead of all the features.
  - This de-correlates the trees.
  - Its mathematically proved that $\sqrt{Number\ of\ predictors}$ is a good approximate value for predictor subset size (mtry/max_features).
- **Evaluation:** A bootstrap sample uses only approximately 2/3 of the observations of original sample.
  - Remaining training data (OOB) are used to estimate error and variable importance



Different Learners: Bagging

- Bagging = “bootstrap aggregation”
  - Bootstrap: draw N items from X with replacement
- Want “unstable” learners
  - Unstable: high variance
  - Decision trees and ANNs are unstable
  - K-NN is stable
- Bagging
  - Train L learners on L bootstrap samples
    - Combine outputs by voting

Different Learners: Boosting

- Boosting: train next learner on mistakes made by previous learner(s)
- Want “weak” learners
  - Weak: P(correct) > 50%, but not necessarily by a lot
  - Idea: solve easy problems with simple model
  - Save complex model for hard problems

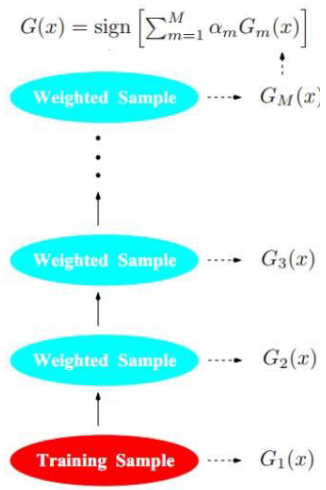
Original Boosting

1. Split data X into {X1, X2, X3}
2. Train L1 on X1
   - Test L1 on X2
3. Train L2 on L1's mistakes on X2 (plus some right)
   - Test L1 and L2 on X3
4. Train L3 on disagreements between L1 and L2

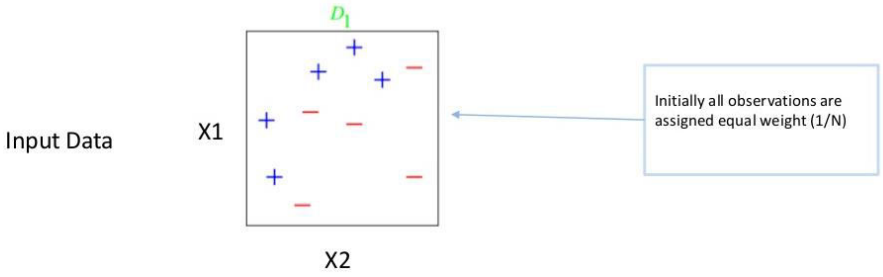
- Testing: apply L1 and L2; if disagree, use L3
- Drawback: need large X

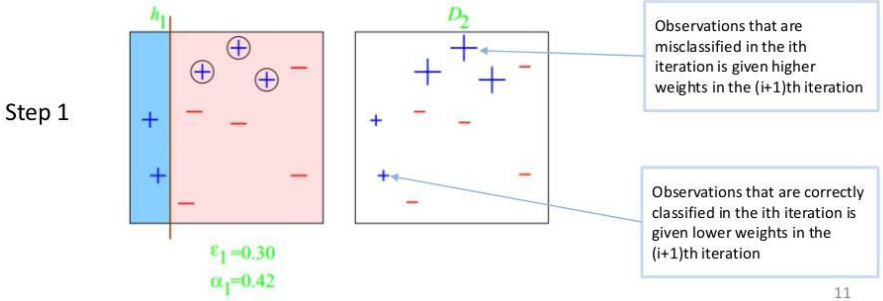
Boosting

- Intuition: Ensemble many "weak" classifiers (typically decision trees) to produce a final "strong" classifier
  - Weak classifier → Error rate is only slightly better than random guessing.
- Boosting is a Forward Stagewise Additive model
- Boosting sequentially apply the weak classifiers one by one to repeatedly reweighted versions of the data.
- Each new weak learner in the sequence tries to correct the misclassification/error made by the previous weak learners.
  - Initially all of the weights are set to Wi = 1/N
  - For each successive step the observation weights are individually modified and a new weak learner is fitted on the reweighted observations.
  - At step m, those observations that were misclassified by the classifier Gm−1(x) induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly.
- Final "strong" classifier is based on weighted vote of weak classifiers

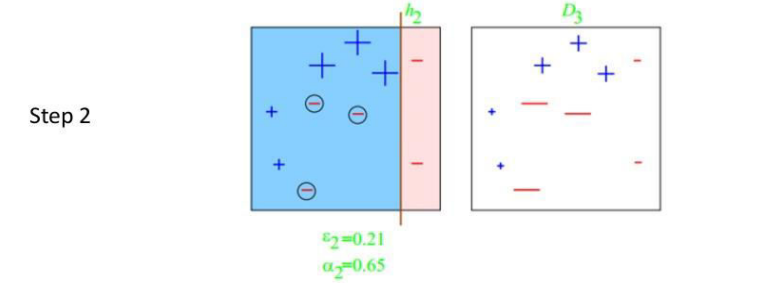


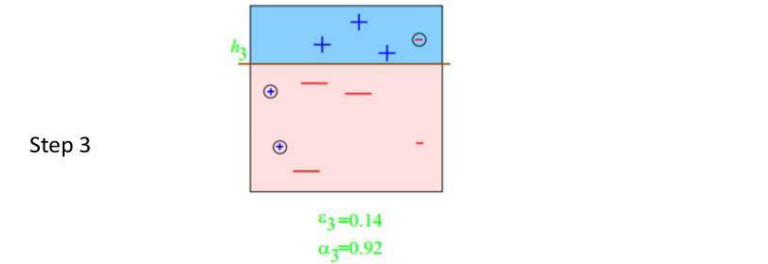
AdaBoost – Illustration



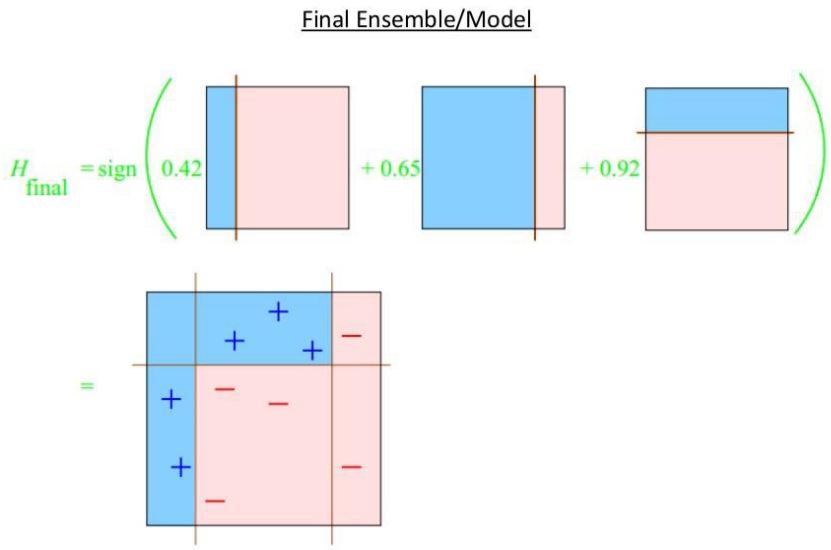


AdaBoost – Illustration





AdaBoost – Illustration



Methods for Coordinated Construction of Ensembles

The key idea is to learn complementary classifiers so that instance classification is realized by taking an weighted sum of the classifiers. This idea is used in two methods:

- Boosting
- Stacking.

Boosting

- Also uses voting/averaging but models are weighted according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm

AdaBoost.M1

classifier generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Learn a classifier from weighted dataset.
  Compute error e of classifier on weighted dataset.
  If e equal to zero, or e greater or equal to 0.5:
    Terminate classifier generation.
  For each instance in dataset:
    If instance classified correctly by classifier:
       Multiply weight of instance by e / (1 - e).
  Normalize weight of all instances.
```

classification

```
Assign weight of zero to all classes.
For each of the t classifiers:
  Add -log(e / (1 - e)) to weight of class predicted
  by the classifier.
Return class with highest weight.
```

Remarks on Boosting

- Boosting can be applied without weights using re-sampling with probability determined by weights;
- Boosting decreases exponentially the training error in the number of iterations;
- Boosting works well if base classifiers are not too complex and their error doesn't become too large too quickly!
- Boosting reduces the bias component of the error of simple classifiers!

Some Practical Advices

•If the classifier is unstable (high variance), then apply bagging!

•If the classifier is stable and simple (high bias) then apply boosting!

•If the classifier is stable and complex then apply randomization injection!

•If you have many classes and a binary classifier then try error-correcting codes! If it does not work then use a complex binary classifier!