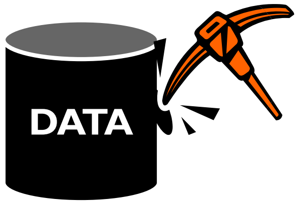
National College of Ireland



Data & Web Mining

Perceptrons and Support Vector Machines

Dr. Jason Roche
Jason.roche@ncirl.ie

10. Perceptrons and SVMs

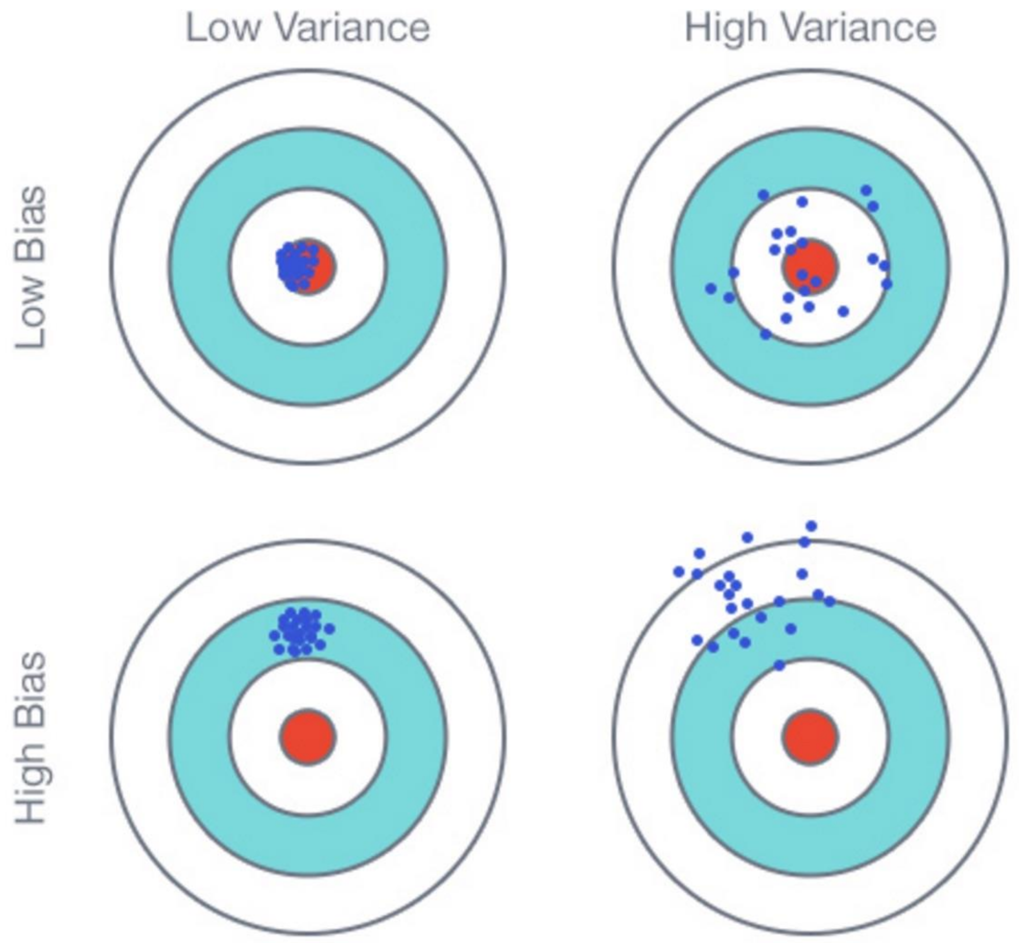
10.1 Introduction

- Introduction

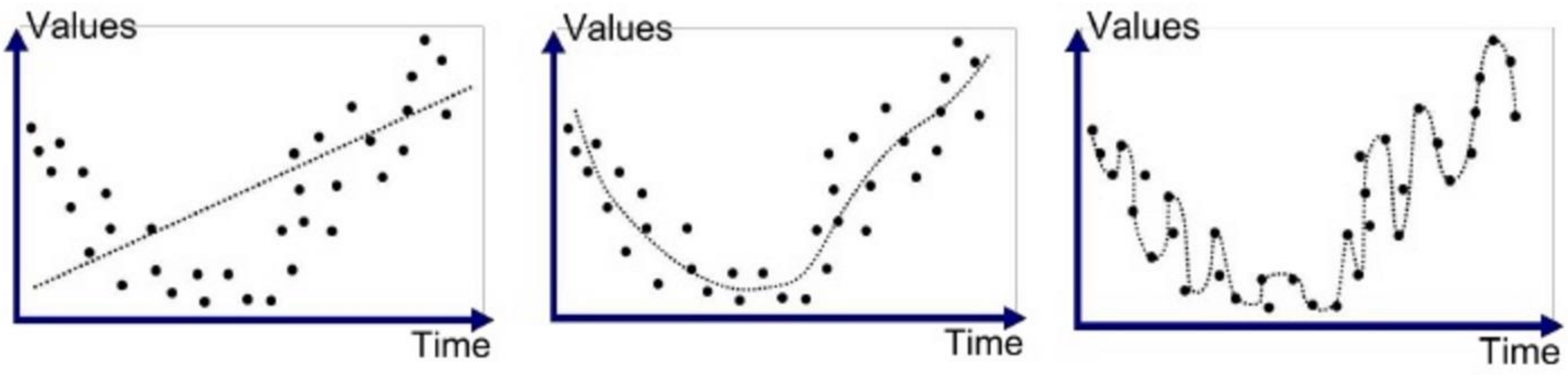
  - We will look at

    - Generalization
    - VC Dimension
    - Perceptrons
    - Classification with Support Vector Machines
    - Domain Usage
    - Kernel Methods
    - Examples

10.1 Error = bias + variance



10.1 How well is ouor model doing ?



high bias
low variance

medium bias
medium variance

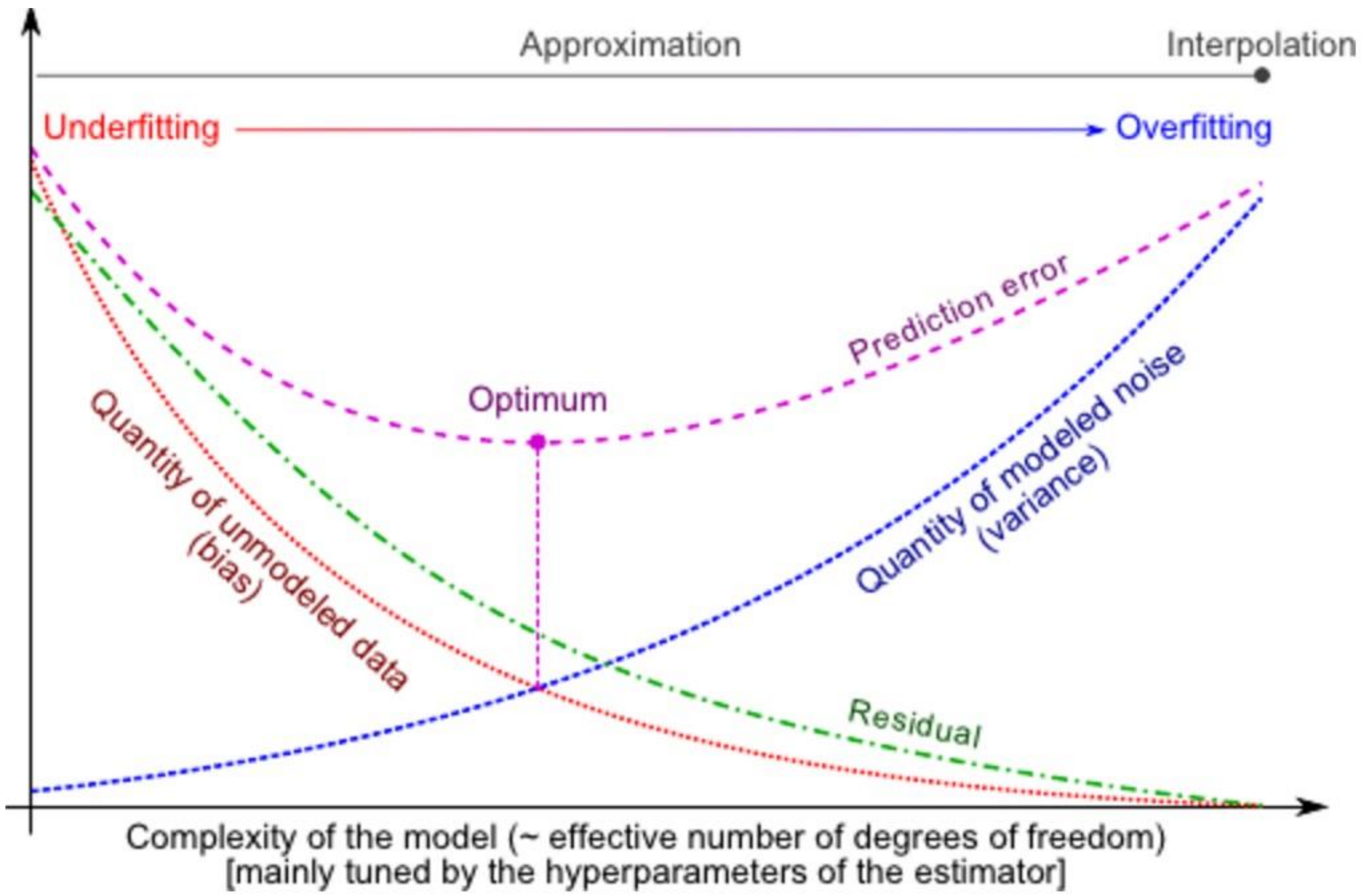
low bias
high variance

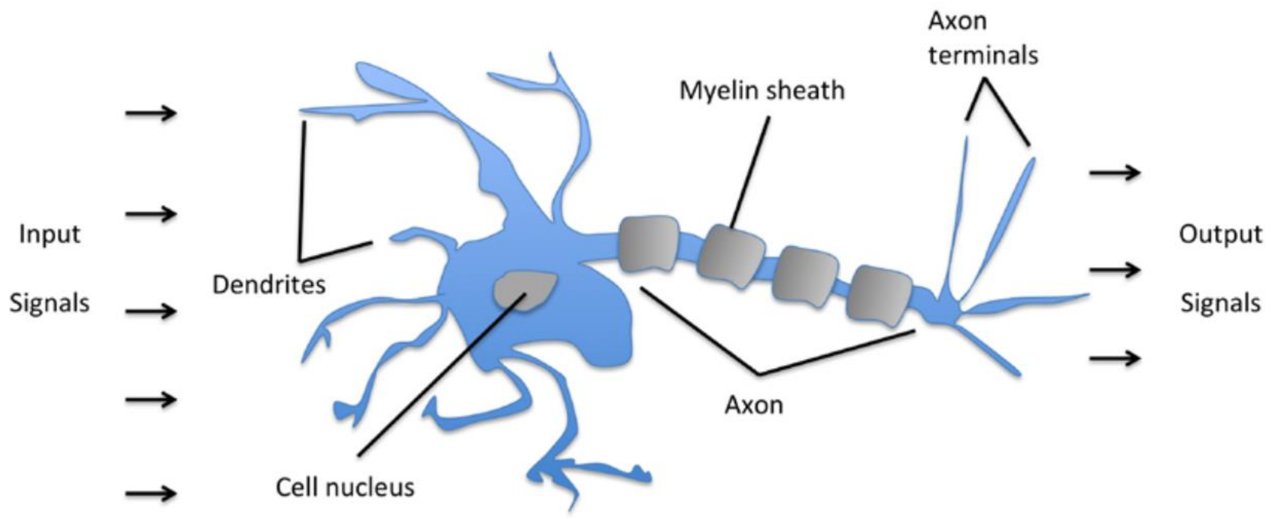
Error = bias + variance

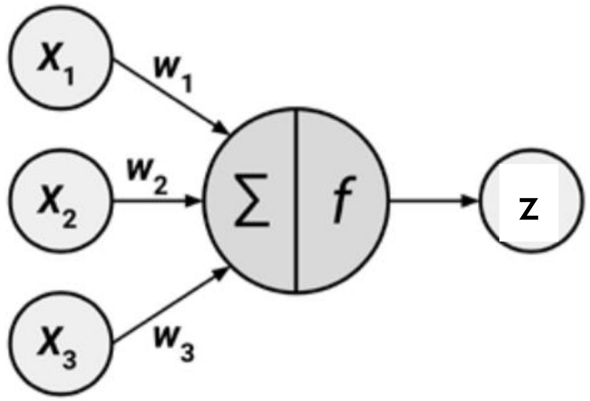
Joannes Vermorel, 2009-04-19, www.lokad.com

National College of Ireland

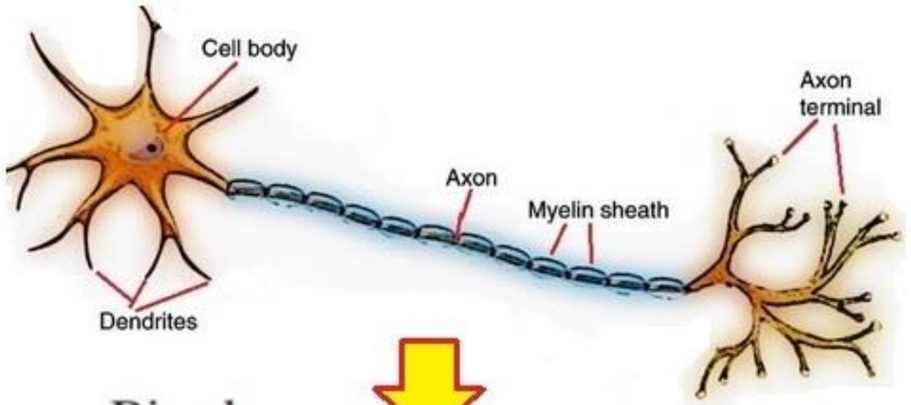
10.1 Introduction

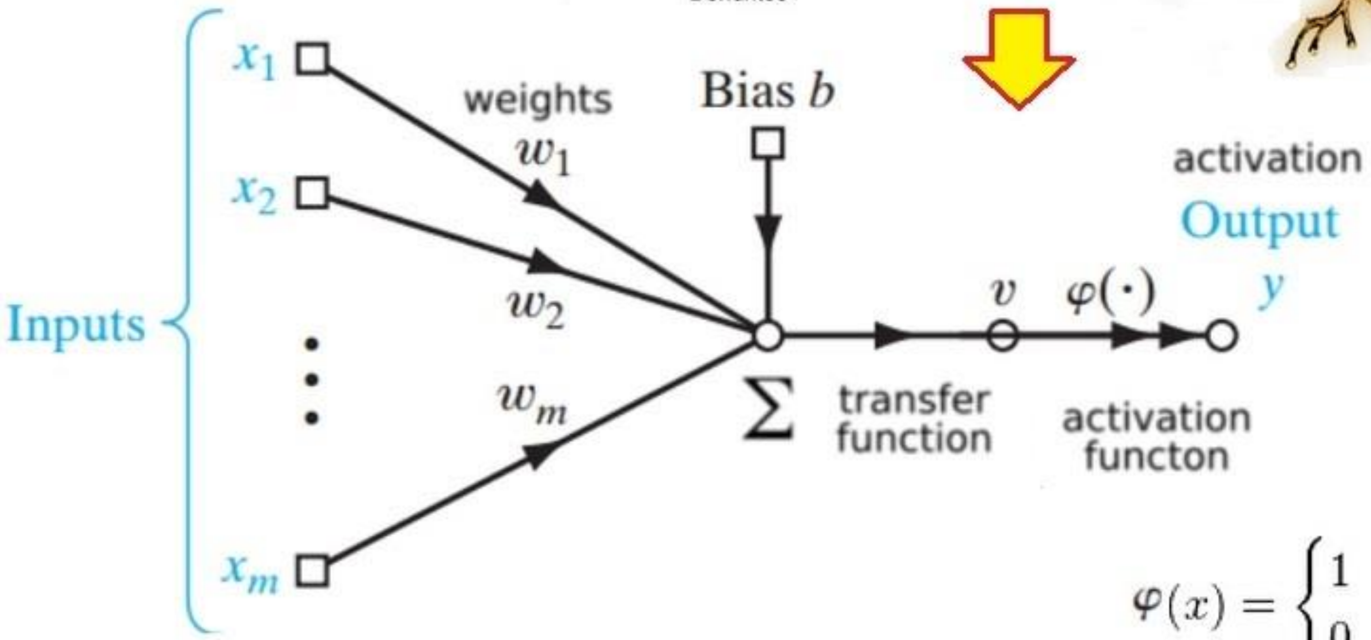




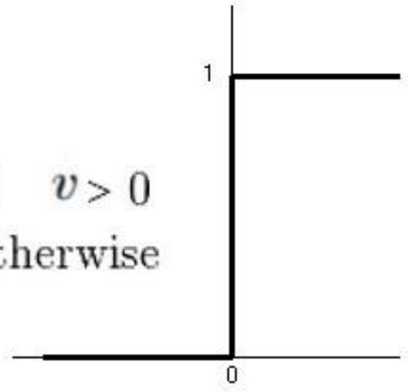


X is a vector representing the input signals. W is a vector representing the weights. The result of the dot product is compared to a threshold value Θ to determine the class label “1” or “-1” depending on threshold function Φ(z)





$$\varphi(x) = \begin{cases} 1 & \text{if} \quad v > 0 \\ 0 & \text{otherwise} \end{cases}$$

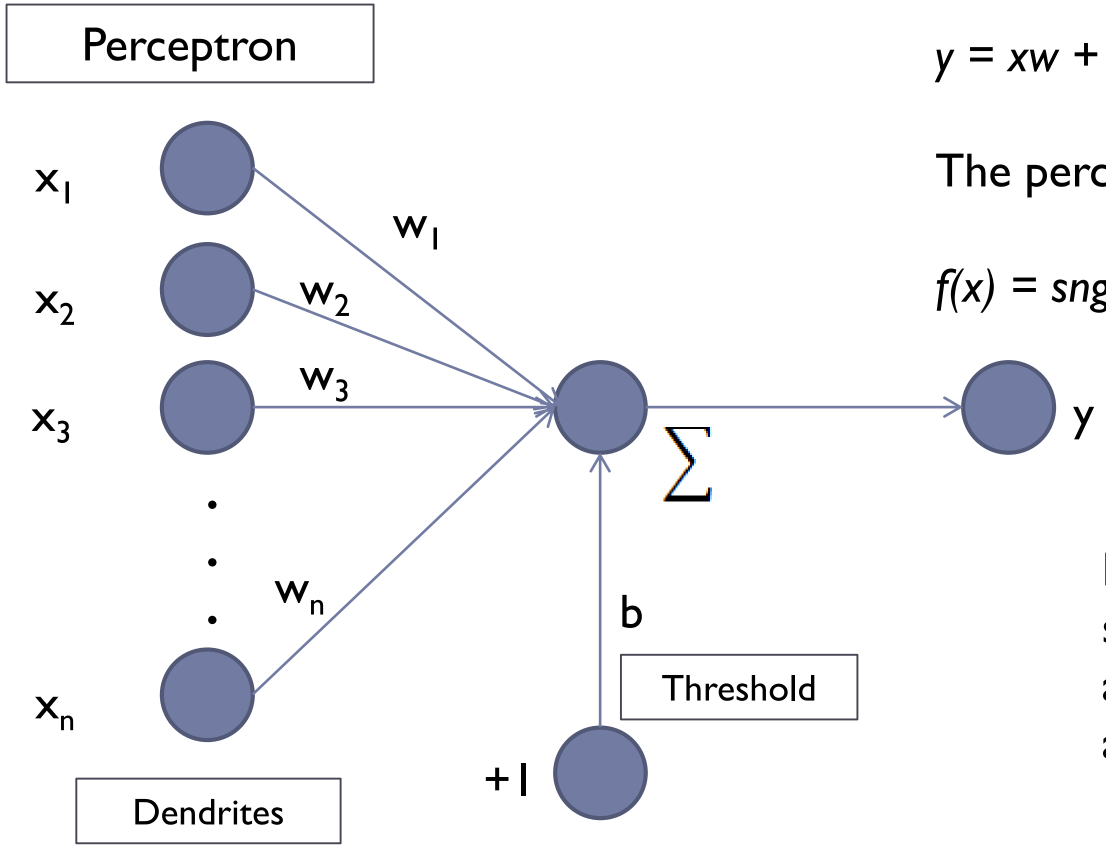


$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_m \end{bmatrix} \quad v = \sum_{j=1}^{m} x_j w_j + \mathrm{b} = \mathbf{w}^T \mathbf{x}$$

$$\mathrm{b} = w_0 x_0 \qquad x_0 = 1$$

10.3 Perceptrons

- Perceptrons



Output can be modeled by a Hyperplane

$y = x_1w_1 + x_2w_2 + \dots x_nw_n \; b = 0$

$y = xw + b = 0$

The perceptron classification function is:

$f(x) = sng(y) = sng(xw + b)$

Learning algorithm will find suitable values for the vector and the scalar and will determine a hyperplane

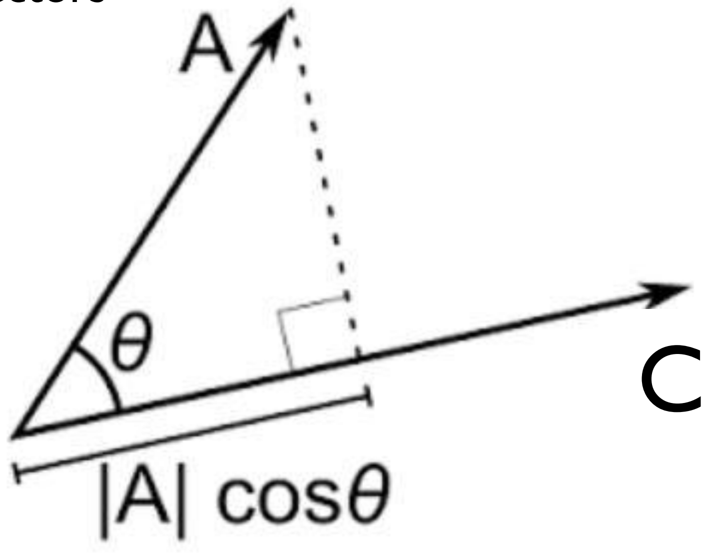
10.3 Perceptrons: Dot product

Consider two points A and C represented by the vectors drawn from the origin to A and C .

They both have a magnitude and a direction.

But how similar are they ?

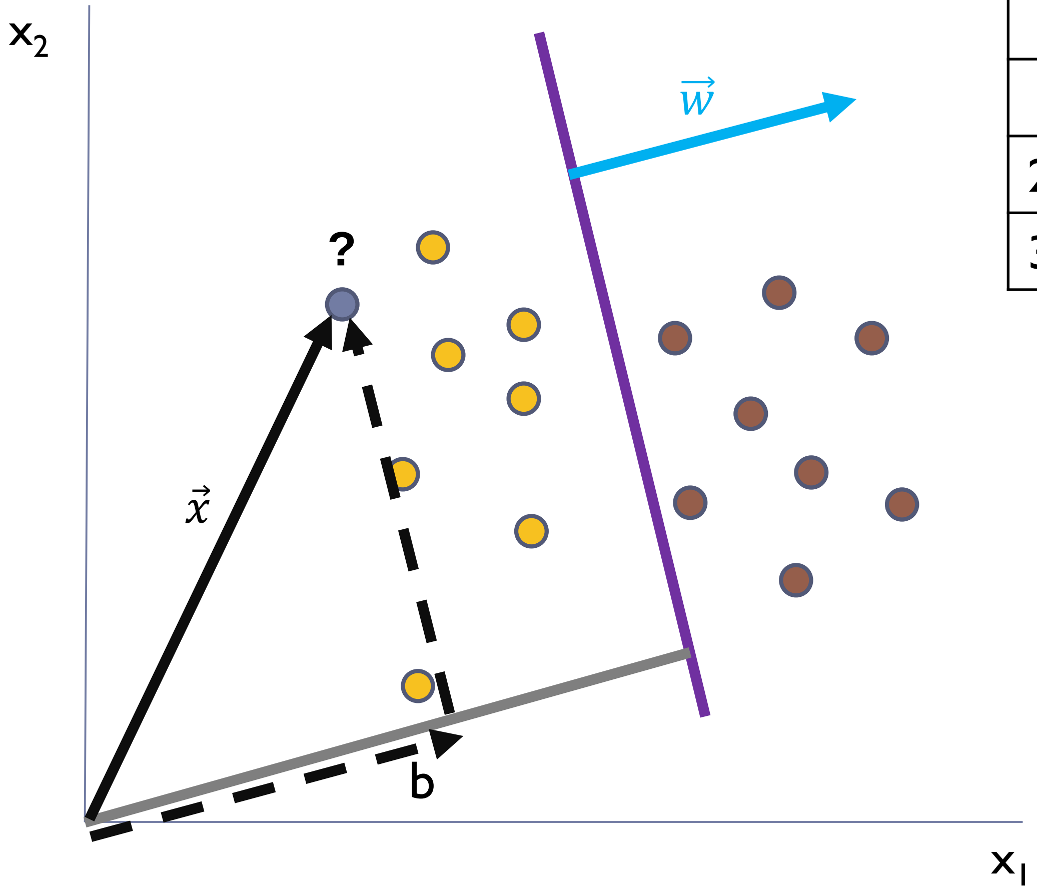
To consider how similar the vector A is to the vector C, lets project A onto C and thus considering the length (or magnitude) of the component of A in the direction of C.



Note: this length of A in the direction of C will not be a vector but will be a single number or scalar.

We will use this magnitude of A in the direction of the weight vector to compare against the bias value to see which side of the hyperplane (i.e. decision boundary) the point falls on.

10.3 Perceptrons: Lets classify a point



id	x1	x2	Class (A/B)
1	2	6	A
2	7	8	B
3	1	5	?

● **Class A**

● **Class B**

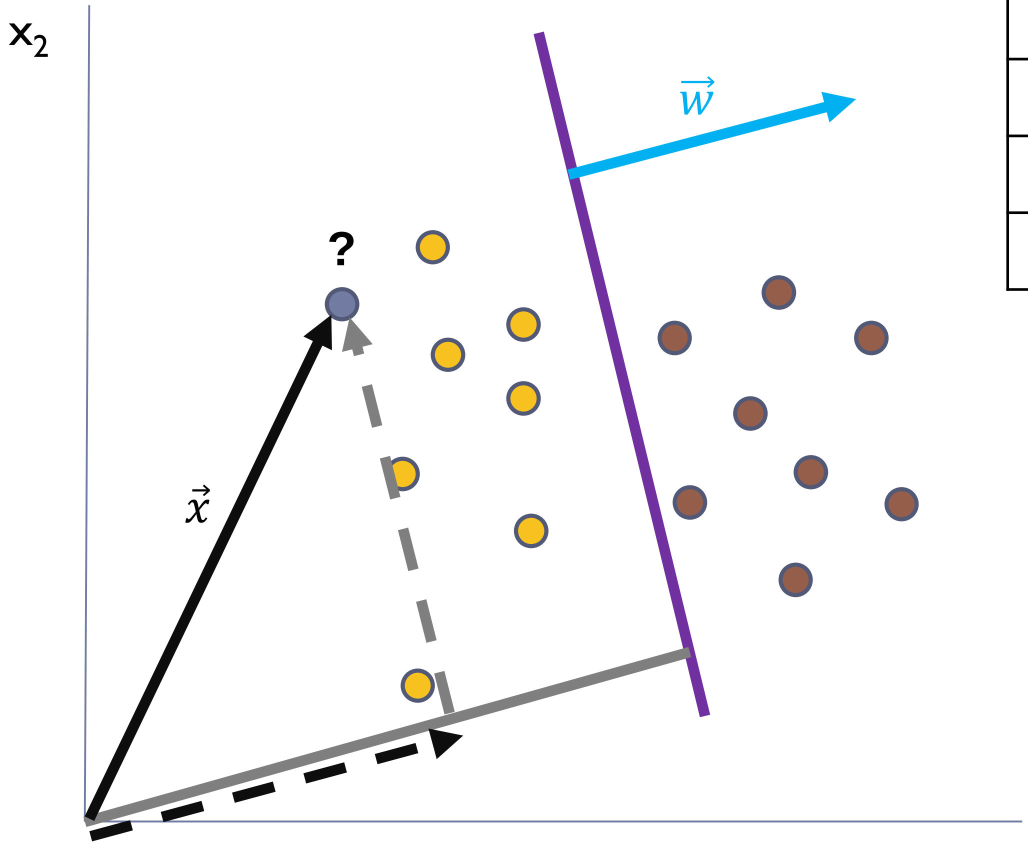
$\vec{w}$ Is the weight vector with components:

$\vec{x}$ Is the instance vector with components:

b Is the perpendicular distance of the hyperplane from the origin.

| Is the hyperplane defined by b and w

10.3 Perceptrons: Lets classify a point



id	x1	x2	Class (A/B)
1	2	6	A
2	7	8	B
3	1	5	?

● Class A

● Class B

$\vec{w}$ Is the weight vector with components:

$\vec{x}$ Is the instance vector with components:

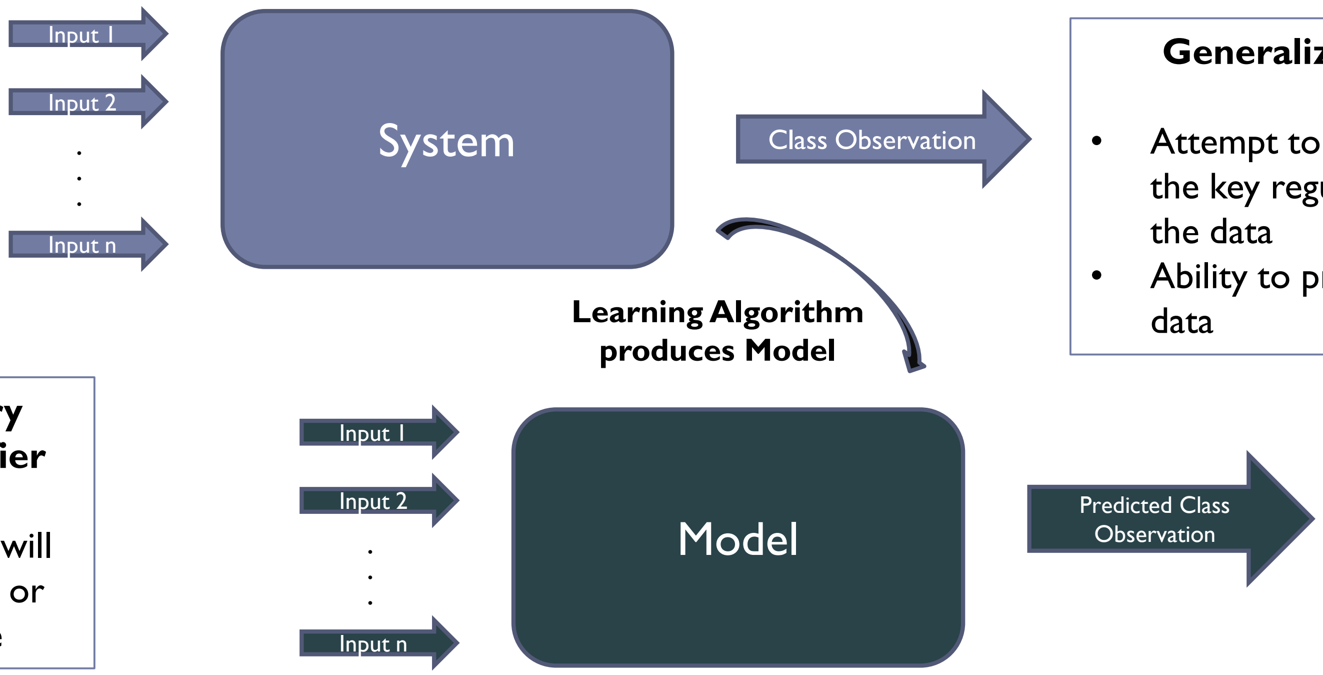
b Is the perpendicular distance of the hyperplane from the origin.

▮ Is the hyperplane defined by b and w

$< \vec{x}\ \vec{w} >$ = (x1*w1 + x2*w2) is clearly < b and so out instance is classified as type **B**

10.2 Generalization

- Classification / Inductive Learning
  - Build model of a system
    - Use observations of how the system has behaved given certain inputs
    - Predict how system will behave for a previously unobserved set of input



Generalization

- Attempt to capture the key regularities in the data
- Ability to predict new data

Binary Classifier

Output will be True or False

10.2 Generalization

- Generalization
  - Some thoughts and questions …

A model's degree of generalization is good if it predicts new data well

A more complex model will have a better chance of classifying most (or even all) of the training set instances?

Can we compute a model's expected error rate?

Is there any way to get a measure or bounds regarding a model's ability to generalize?

Is a model's expected error rate related to the error rate obtained when applying the classification rules to the training data?

10.2 Generalization

- Learning process
  - Let's say we have some dataset **(x, y)**, where
    - **x** is a vector of values – feature vector; these can be nominal, ordinal, numerical etc.
    - **y** is a label that classifies the value for **x**
    - We'll call this a training data set for now
  - The objective of machine learning is to discover some function such that:
    - **y = f(x)**
  - There are several settings here:
    - y is a number – typically this is a regression problem
    - y is a boolean value – this is a binary classification problem
    - y is a member of some finite set – this is a multi-class classification problem
    - y is a member of an infinite set – this is a (typically) decision tree problem

National College of Ireland

10.2 Generalization

Learning process

- We can compute the error rate of the model with respect to the set of training data → this is called the ***empirical error rate*** and is obtained through computing the ratio

$$\frac{number\ of\ instances\ classified\ incorrectly}{number\ of\ instances\ in\ the\ training\ data\ sample}$$

- Since the model is optimised on the training data we would expect the observed error rate associated with the training set to be more optimistic than the true error rate.

- The ***true error rate*** of a model will be unknown
  - We assume the existence of an unknown distribution function $P(\boldsymbol{x}, y)$

- The difference between the true error rate and the empirical error rate will depend on the complexity of the model and the distribution $P(\boldsymbol{x}, y)$

We can get an **upper-bound** on the **true error rate** though!

10.2 Generalization

VC Dimension

- To calculate the upper-bound (called the ***generalization bound***) we need to take account of the model's complexity

- Vapnik (1995) introduced a measure of complexity called the

Vapnik-Chervonenkis Dimension
or
VC Dimension

- Generalization Bound is independent of $P(\boldsymbol{x}, y)$
- The VC dimension is used in computing the **capacity** of learning algorithms – this capacity determines the generalization reachable during learning

10.2 Generalization

Understanding VC Dimension

- The idea behind supervised Machine Learning is to learn some functions from a set of training examples.

- Let's say that
  - $T = \{x_1, x_2, ..., x_n\}$ represents the set of training instances
  - We can view these instances as data points that are associated with some discrete values from a set $C = \{c_1, c_2, ..., c_m\}$
  - We are attempting to find a function $f: X \rightarrow C$ where T is a subset of X.
  - The function f can be learned by using an algorithm which can generate only a small subset of all possible functions.

10.2 Generalization

- Understanding VC Dimension

  - f is the target function

  - The algorithm derives a function h ∈ H (the set of all possible hypotheses (functions) derivable by the algorithm)

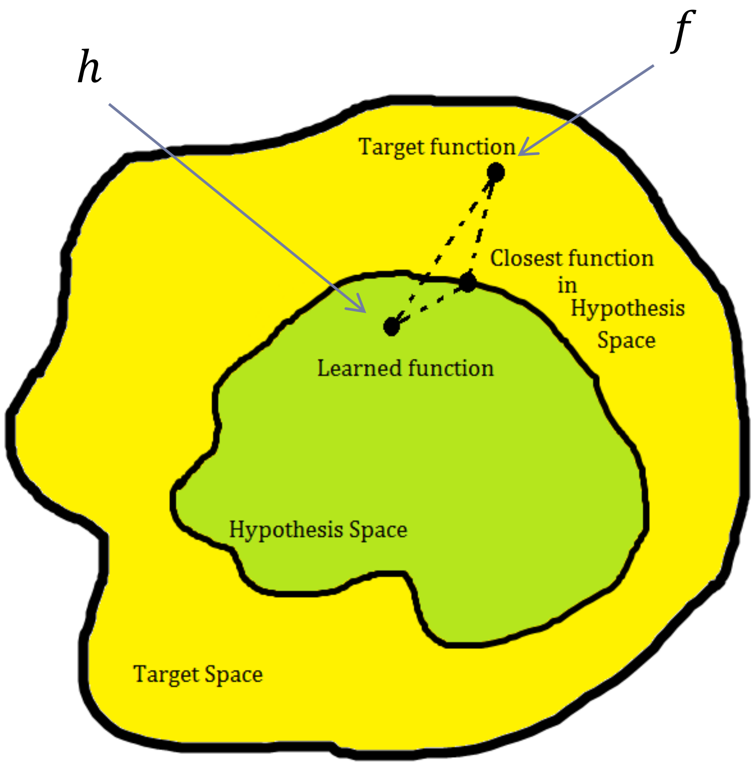
  - It is unlikely that f = h
    - A measure of the degree of dissimilarity is provided for by the error rate

National College of Ireland

10.2 Generalization

- Understanding VC Dimension

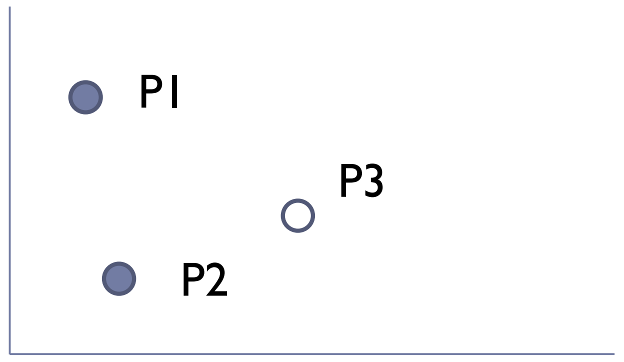
  - The Hypothesis Space represents the set of all possible functions that the learning algorithm could create
    - i.e., the set of all possible models a particular learning algorithm could create
  - It will generally not be possible for the learning algorithm to generate a function that is identical to the targe function
    - and impossible if the target function lies outside the Hypothesis Space



10.2 Generalization

- Shattering
  - Consider a set of N points in the 2D x-y plane
  - Then there are 2^N ways to assign binary class labels to these points

P1 = (x1, y1)
P2 = (x2, y2)
P3 = (x3, y3)



	P1	P2	P3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

National College of Ireland

10.2 Generalization

- Shattering

  - Consider a class of functions H (a particular Hypothesis Space)

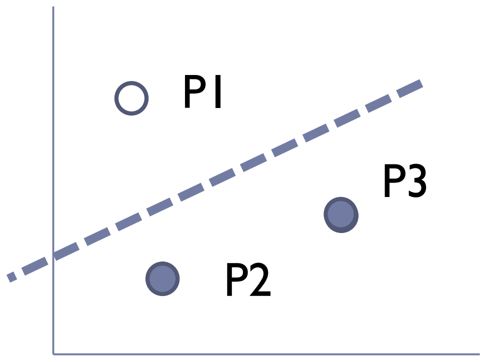
A set of N points is said to be ***shattered*** by H if and only if for all the 2^N label permutations there exists a function h ∈ H such that when we use h as a classification rule then there are no training errors

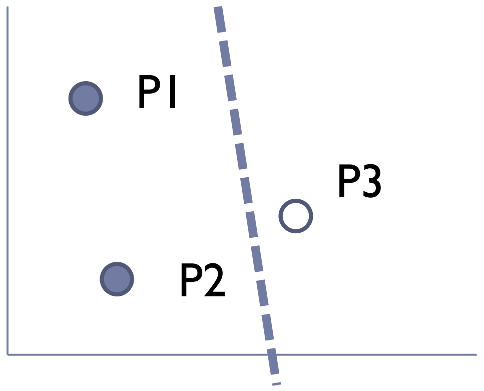
The largest number of points that a function class can shatter is the ***VC Dimension***

10.2 Generalization

- Shattering

  - Let's consider as a class of functions H = {h | h is a straight line}
  - What is the VC Dimension of H in this case?
  - What is the maximum number of points in the plane that a straight line could successfully classify?





P1 = (x1, y1)
P2 = (x2, y2)
P3 = (x3, y3)

National College of Ireland

10.2 Generalization

- Understanding VC Dimension
  - Different classes of functions have different capacity measures in separating data instances
    - Some can separate more data instances → greater generalization
  - A function from a high capacity class of functions will separate with greater ease any particular training set

May lead to overfitting though!

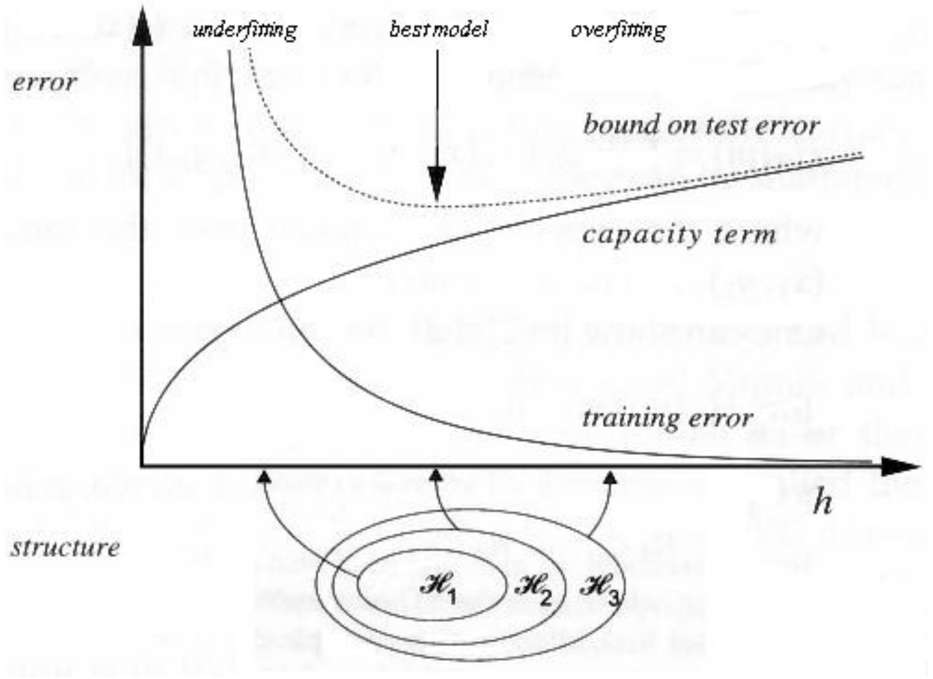
  - If a function from a low capacity class of functions successfully separates any particular training set then it will probably also separate a set of test data

May lead to underfitting though!

10.2 Generalization

- Structural Risk Minimization

  - Inductive principle for model selection used for learning from finite training data sets
  - Describes a general model of capacity control
  - Provides a trade-off between hypothesis space complexity (the VC dimension of approximating functions) and the quality of fitting the training data (empirical error)
  - Best model is the model whose sum of empirical risk and VC confidence is minimal



http://www.svms.org/srm/

10.2 Generalization

- Generalization and Support Vector Machines
  - We have seen that a 1D line can shatter 3 points on the 2D plane
  - Similarly, it can be shown that a 2D plane can shatter 4 points in a 3D cube

•

•

•

  - A (n-1)D plane (i.e., a hyperplane) can shatter (n+1) points in a nD hypercube
  - Support Vector Machines rely on such classes of linear functions

10.3 Perceptrons

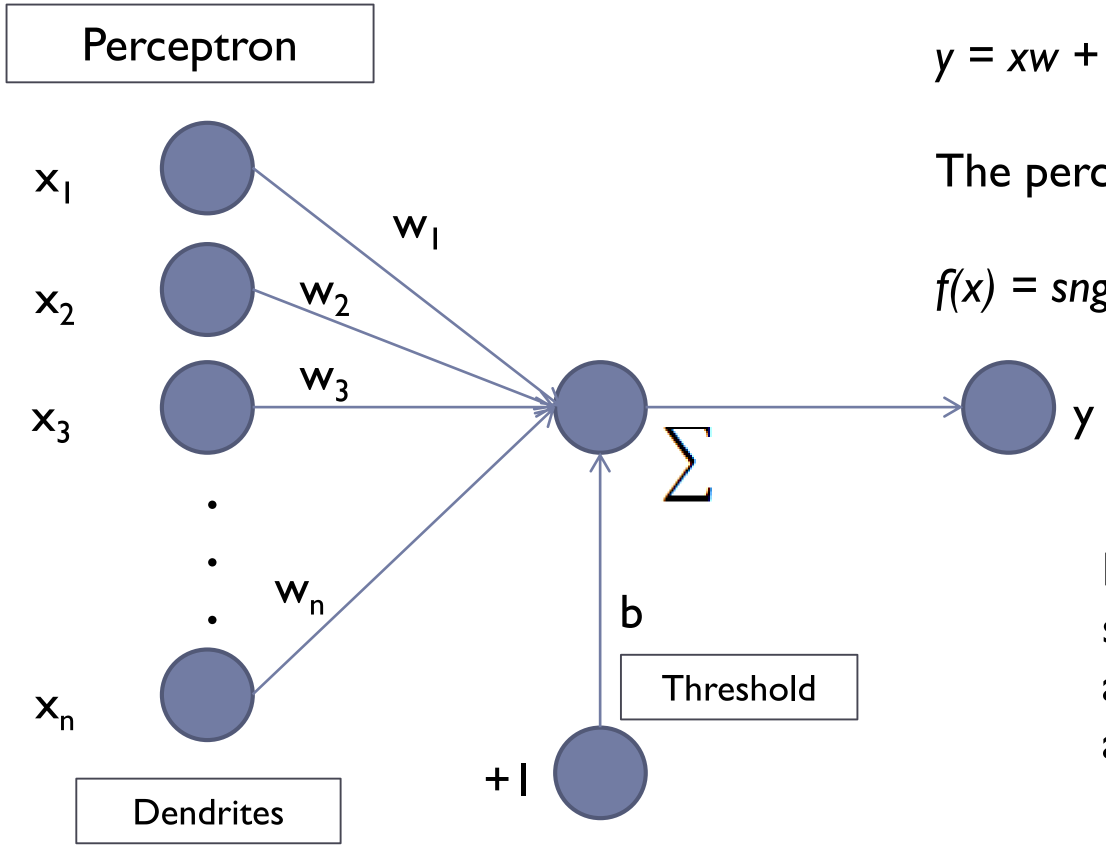
- Perceptrons
  - The simplest SVM - a linear binary classifier
    - read is class A or is not class A
  - Learns linear functions (i.e., hyperplanes) as a means to separate and differentiate between a large number of data instances
  - Given an input vector x = $[x_1, x_2, ..., x_n]$
  - Associated with a vector of weights w = $[w_1, w_2, ..., w_n]$
  - A peceptron uses a threshold value θ and assigns
    - +1 if $w.x > \theta$
    - -1 if $w.x < \theta$
  - Where $w.x = \theta$ is considered "wrong" – such cases sit on the decision boundary

10.3 Perceptrons

- Perceptrons
  - The weight vector w defines a hyperplane of dimension d–1
    - Where d is the set of points x such that w.x = θ
  - Points on the positive side of the hyperplane are classified +1; points on the negative side -1
  - If there are multiple hyperplanes possible a perception will converge to one that separates the given training data
    - Danger of over fitting!
  - A perceptron can <u>ONLY</u> classify linearly separable data
    - If data is not linearly separable: infinite loop that doesn't converge

10.3 Perceptrons

- Perceptrons



Output can be modeled by a Hyperplane

$y = x_1w_1 + x_2w_2 + \ldots\ x_nw_n\ b = 0$

$y = xw + b = 0$

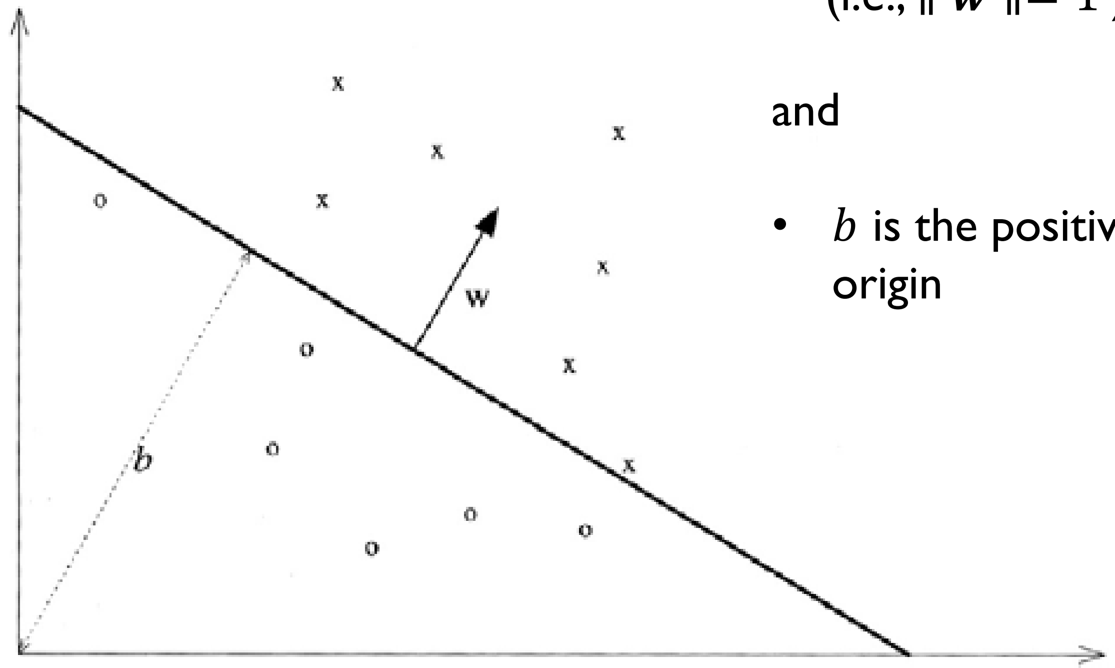
The perceptron classification function is:

$f(x) = sng(y) = sng(xw + b)$

Learning algorithm will find suitable values for the vector and the scalar and will determine a hyperplane

10.3 Perceptrons

- Perceptrons



Here we assume

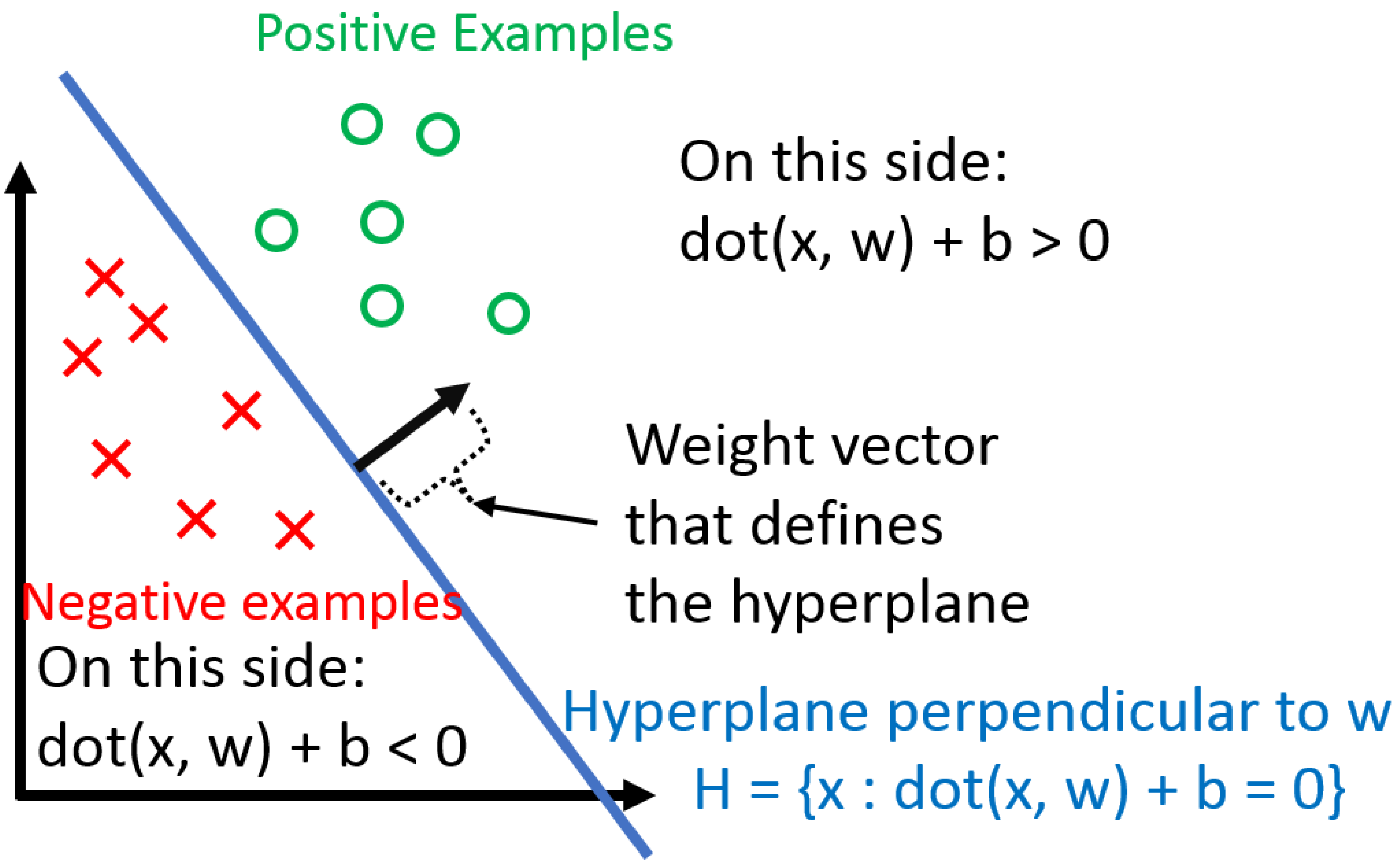
- the length of $\boldsymbol{w} = 1$ (i.e., $\| \boldsymbol{w} \| = 1$)

and

- b is the positive distance from the origin

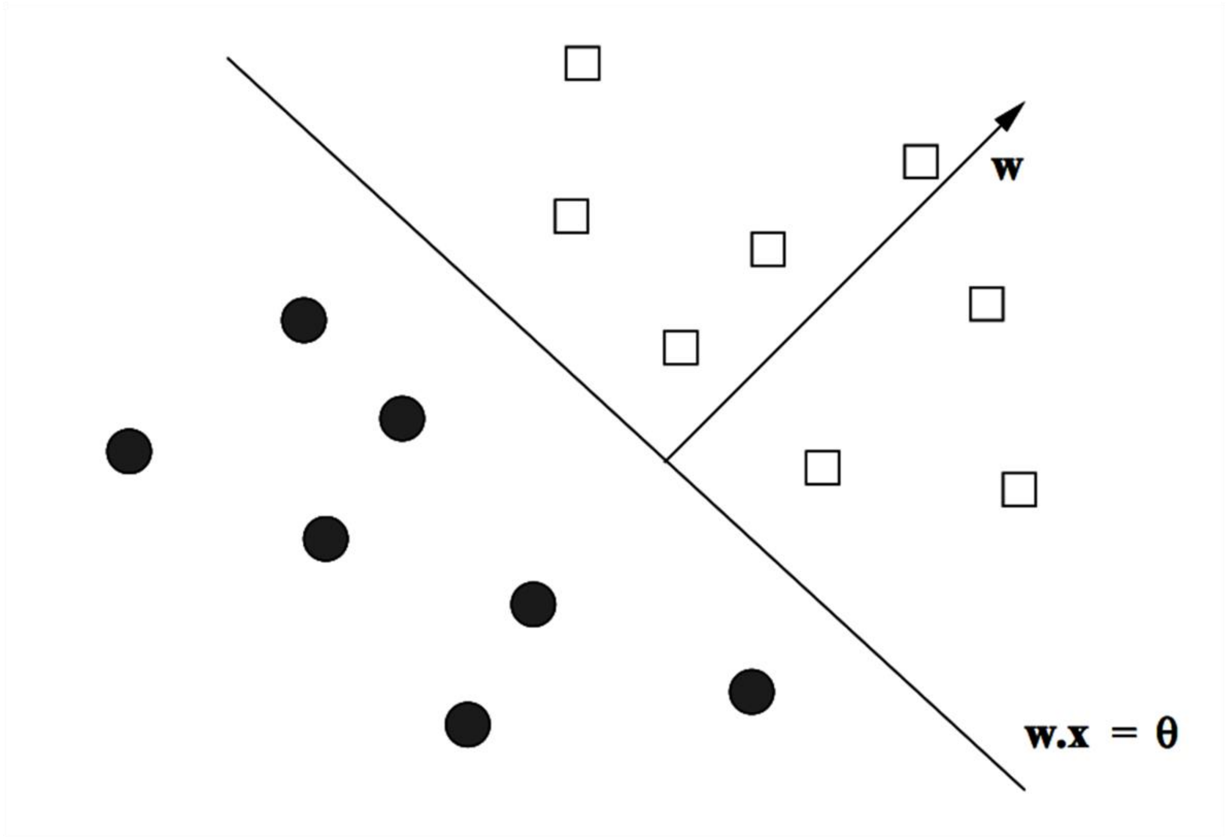
(Cristianini, 2000)

10.3 Perceptrons



10.3 Perceptrons

- Perceptrons



10.3 Perceptrons

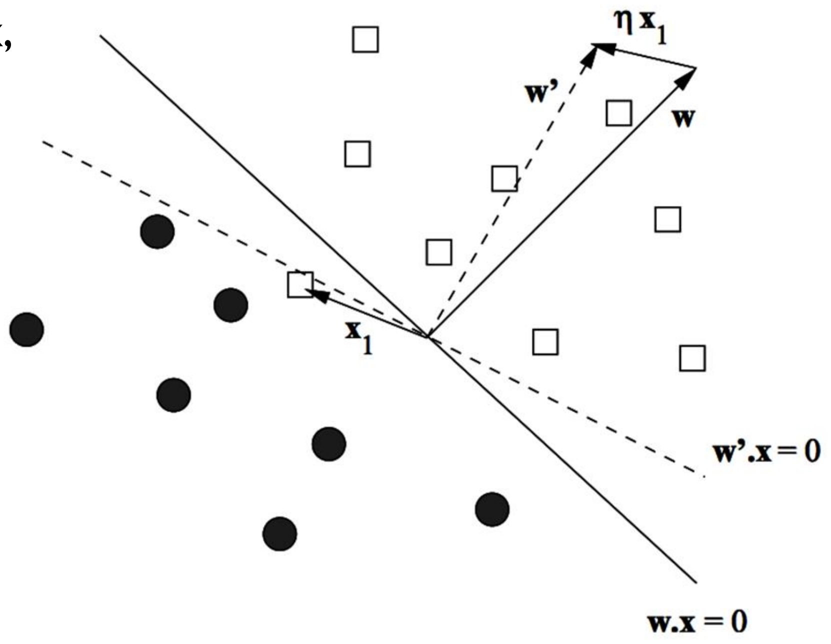
- Training a perceptron

  - Let us first consider the simplest case: $\theta = 0$
    - Simple assumption if we don't know what the threshold should be
  - Essentially, we try to find a weight vector such that:
    - All feature vectors with y = +1 are on the positive side of the hyperplane
    - All feature vectors with y = -1 are on the negative side of the hyperplane
  - Set up:
    - Initialise all weights to 0
    - Pick a learning rate $\boldsymbol{\eta}$ – a small positive real number; it controls the speed of convergence.
      - Too small and convergence is slow, too big and convergence is too volatile (and again slow – if it converges at all)

National College of Ireland

10.3 Perceptrons

- Training a perceptron
  - Consider each training example t = (x, y) individually, and
    1. Let y' = w.x
    2. If y' = y: Do nothing t is properly classified
    3. If y' != y: Replace w with w + ηyx,
       - i.e. adjust w in the direction of x
  - Moving w towards x shifts the hyperplane



National College of Ireland

10.3 Perceptrons

- Example: spam detection
  - The training (x, y) set consists of 0's and 1's indicating the presence (x_i = 1) or absence (x_i = 0) of a word in an email
  - y = +1 if the email is known to be spam and -1 otherwise
  - We're going to use a learning rate η = ½
  - For simplification, we use only 5 keywords: "and", "Viagra", "the", "of", "Nigeria"
  - Initially, w is all 0's

	and	Viagra	the	of	Nigeria	y
a	1	1	0	1	1	+1
b	0	0	1	1	0	-1
c	0	1	1	0	0	+1
d	1	0	0	1	0	-1
e	1	0	1	0	1	+1
f	1	0	1	1	0	-1

10.3 Perceptrons

	and	Viagra	the	of	Nigeria	y
a	1	1	0	1	1	+1
b	0	0	1	1	0	-1
c	0	1	1	0	0	+1
d	1	0	0	1	0	-1
e	1	0	1	0	1	+1
f	1	0	1	1	0	-1

- Example: spam detection
  - Start with a: [1,1,0,1,1]; y=+1
    - w.a = 0 – this is wrong, so we move w in the direction of a
    - w = w + ηya
    - w = w + ½ * 1 * a
    - w = [0,0,0,0,0] + [½, ½, 0, ½, ½]
    - w = [½, ½, 0, ½, ½]
  - Now consider b: [0,0,1,1,0]; y=-1
    - w.b = [½, ½, 0, ½, ½] . [0,0,1,1,0] = ½
    - b is misclassified so we move w again
    - w = w + ½ * -1 * b
    - w = [½, ½, 0, ½, ½] – [0, 0, ½, ½, 0] = [½, ½, -½, 0, ½]

10.3 Perceptrons

	and	Viagra	the	of	Nigeria	y
a	1	1	0	1	1	+1
b	0	0	1	1	0	-1
c	0	1	1	0	0	+1
d	1	0	0	1	0	-1
e	1	0	1	0	1	+1
f	1	0	1	1	0	-1

- Example: spam detection
  - Now consider c: [0,1,1,0,0]; y=+1
    - w.c = [½, ½, -½, 0, ½] . [0, 1, 1, 0, 0] = 0; move again
    - w = w + ½ * 1 * c = [½, ½, -½, 0, ½] + [0, ½, ½, 0, 0] = [½, 1, 0, 0, ½]
  - Now consider d: [1,0,0,1,0]; y=-1
    - w.d = [½, 1, 0, 0, ½] . [1, 0, 0, 1, 0] = 1; move again
    - w = [½, 1, 0, 0, ½] – [½, 0, 0, ½, 0] = [0, 1, 0, -½, ½]
  - Now consider e: [1,0,1,0,1]; y=+1
    - w.e = [0, 1, 0, -½, ½] . [1, 0, 1, 0, 1] = ½; that's fine, do nothing
  - Now consider f: [1,0,1,1,0]; y=-1
    - w.f = [0, 1, 0, -½, ½] . [1, 0, 1, 1, 0] = -½; that's fine too
  - Let's check: w.a = 1; w.b = -½; w.c = 1; w.d = -½
    - This looks good, but what does it mean?

National College of Ireland

10.3 Perceptrons

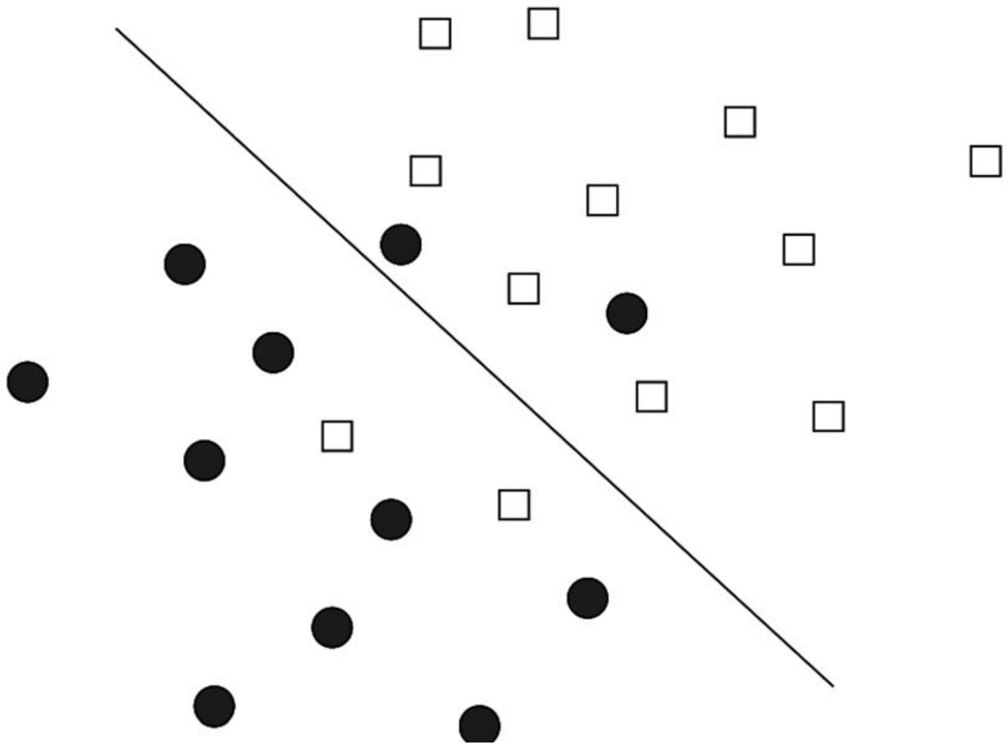
- Example: spam detection

	and	Viagra	the	of	Nigeria	y
a	1	1	0	1	1	+1
b	0	0	1	1	0	-1
c	0	1	1	0	0	+1
d	1	0	0	1	0	-1
e	1	0	1	0	1	+1
f	1	0	1	1	0	-1

  - For, w = [0, 1, 0, -½, ½]
    - The words "and" and "the" are considered neutral
    - The words "Viagra" and "Nigeria" are considered indicative of spam
    - The word "of" is considered indicative of not spam
  - In this case, we terminated at a viable solution using training data only.
  - Whilst the training data is linearly separable, test data may not be.
  - Therefore, we can use stopping criteria like:
    - Terminate after k rounds
    - Terminate when the number of misclassified instances stops changing
    - …

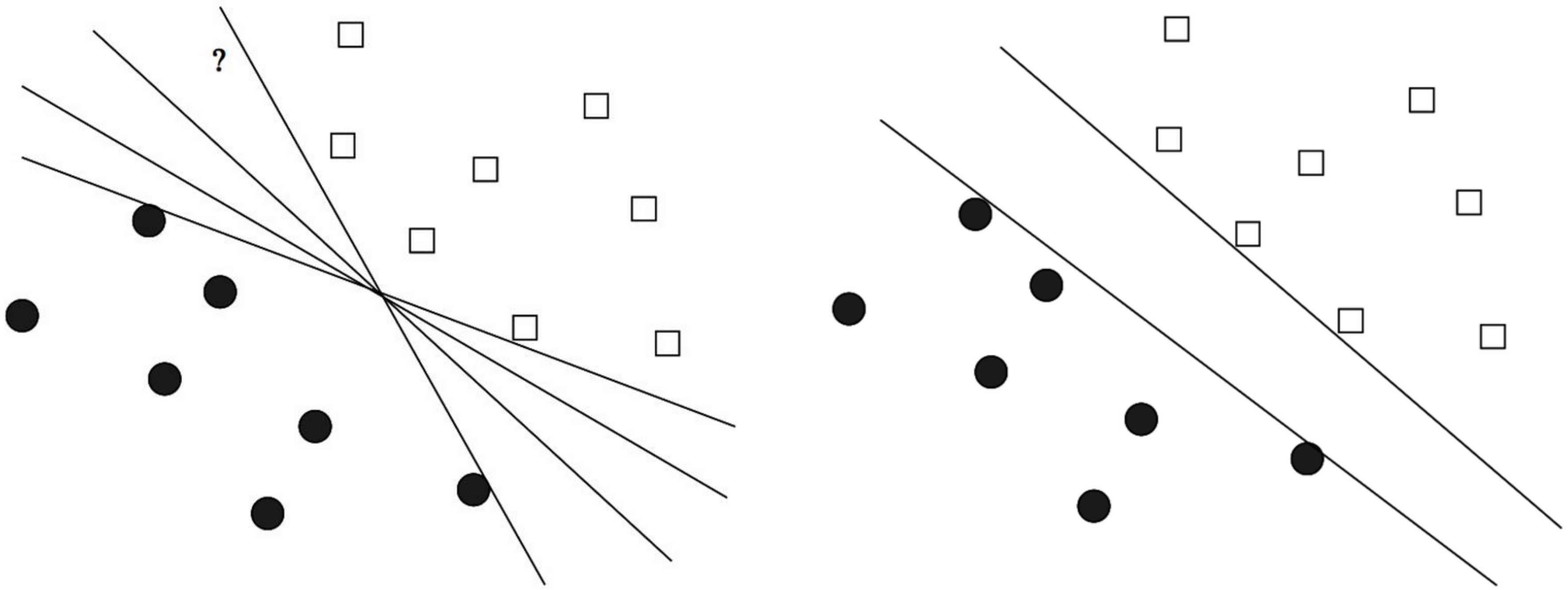
10.3 Perceptrons

- Limitations
  - Lack of linear separability



10.3 Perceptrons

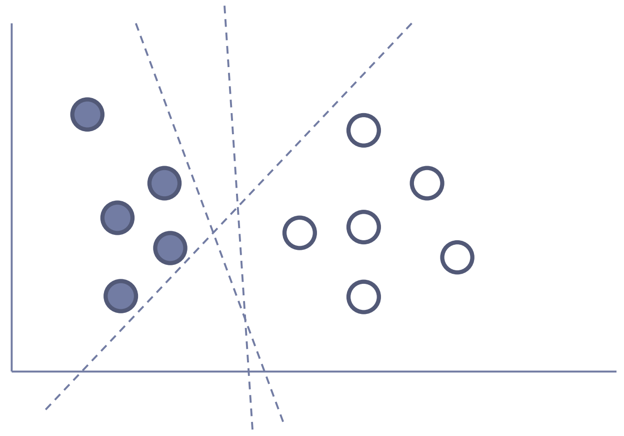
- Limitations
  - Multiple solutions for a hyperplane



10.4 Support Vector Machines

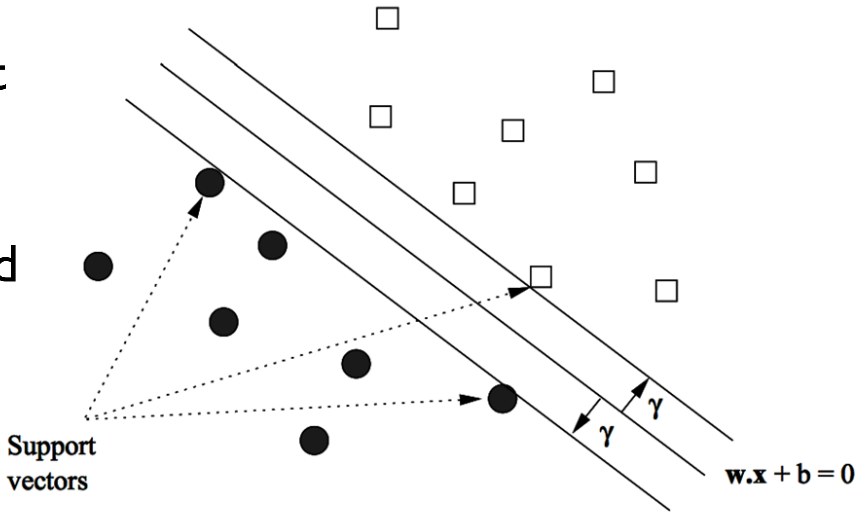
- SVMs
  - Support Vector Machines extend the idea associated with a Perceptron to find a certain hyperplane for classification purposes, but there is only 1 solution to a SVM:
    - A hyperplane that maximises the distance (margin) between the points of the training set
  - SVMs can also handle data instances that are not immediately lineraly seperable
  - Consider the graph

How many ways are there to separate the class instances using lines (i.e hyperplanes)?



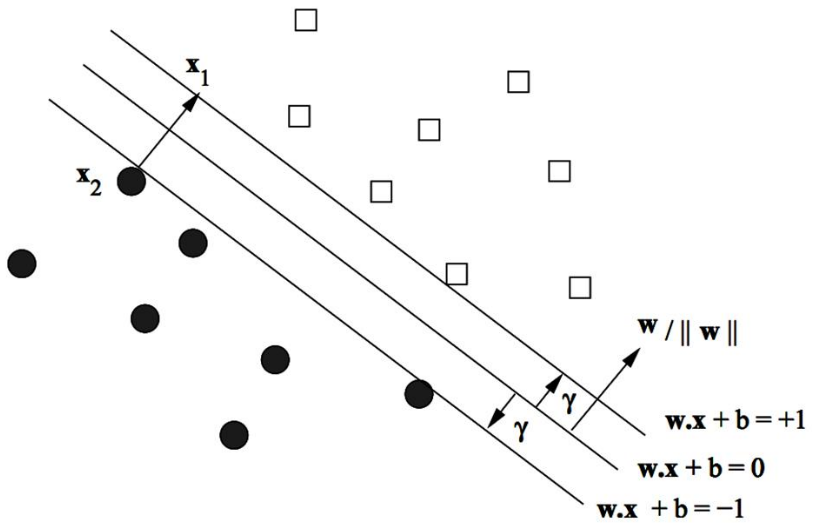
10.4 Support Vector Machines

- SVMs
  - SVM is a **linear learning system** for binary classification
  - Given (x, y), find a linear function on the form $f(x) = x \cdot w + b$ that maximises γ
  - An input vector x is assigned to the positive class if $f(x) \geq 0$ and to the negative class otherwise.
  - Typically d + 1 support vectors
  - Maximising the margin means that points close(r) to the hyperplane in the test dataset have a better chance of being correctly classified



10.4 Support Vector Machines

- Objective of a SVM
  - SVM maximises the margin between positive and negative data points.
  - Maximise γ as the multiple of the unit vector w / ||w||
    - ||w|| is the Frobenius norm (the square root of the sum of squares of the components in w)
  - Given a training set T = $\{(x_1,y_1), \ldots,(x_n,y_n)\}$ minimise ||w|| by varying w and b such that:
    - $\underbrace{y_i(w.x_i + b)}_{\text{Functional Margin}} \geq 1$



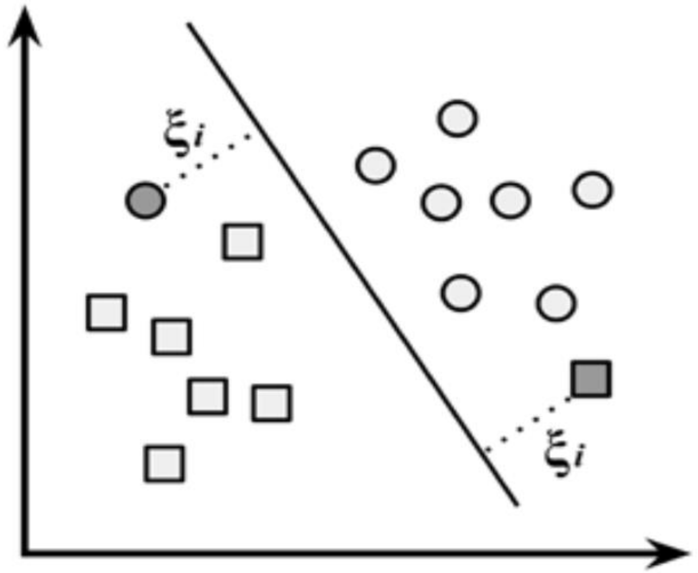
Non linearly separable data

The solution to this problem is the use of a **slack variable**, which creates a soft margin that allows some points to fall on the incorrect side of the margin.

A cost value (denoted as C) is applied to all points that violate the constraints, and rather than finding the maximum margin, the algorithm attempts to minimize the total cost.

$$\min \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$s.t.\quad y_i(\vec{w}\cdot\vec{x}_i - b) \geq 1 - \xi_i, \forall \vec{x}_i, \xi_i \geq 0$$



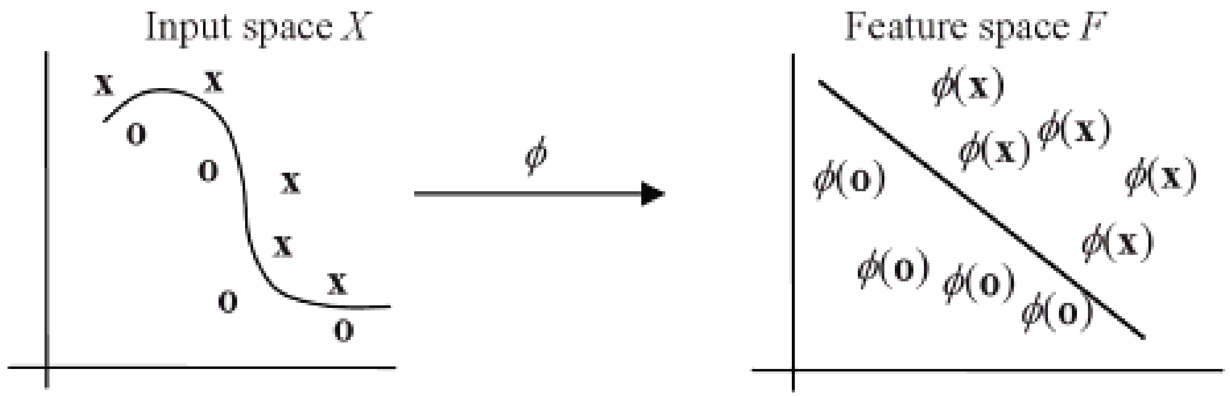
The figure illustrates two points falling on the wrong side of the line with the corresponding slack terms

10.4 Support Vector Machines

- SVM – Kernel Methods
  - In the **Input Space** X the decision boundary is non-linear.
  - To create a classifier in this case we can use the same techniques as with the linear case – however, we must transform the input data from the original space into another space called a **Feature Space** F by using a non-linear mapping ϕ.

$$\phi: X \rightarrow F \quad \text{with } \boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$$

  - The Feature Space will typically have many more dimensions. The same linear SVM method can then be applied.



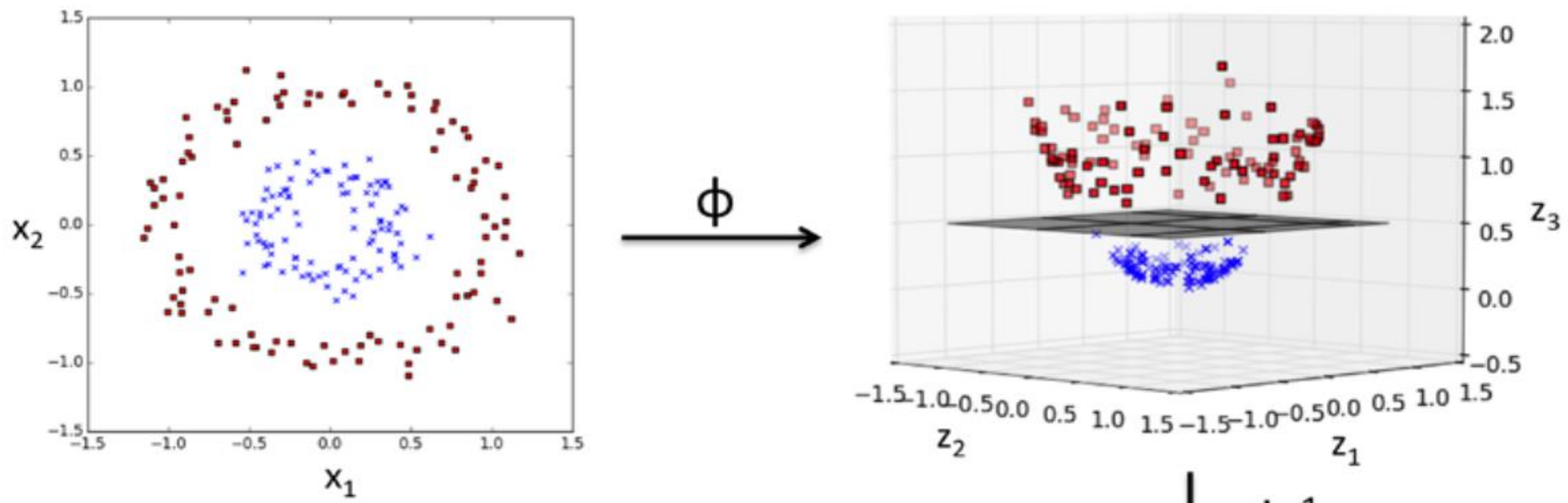
10.4 Support Vector Machines

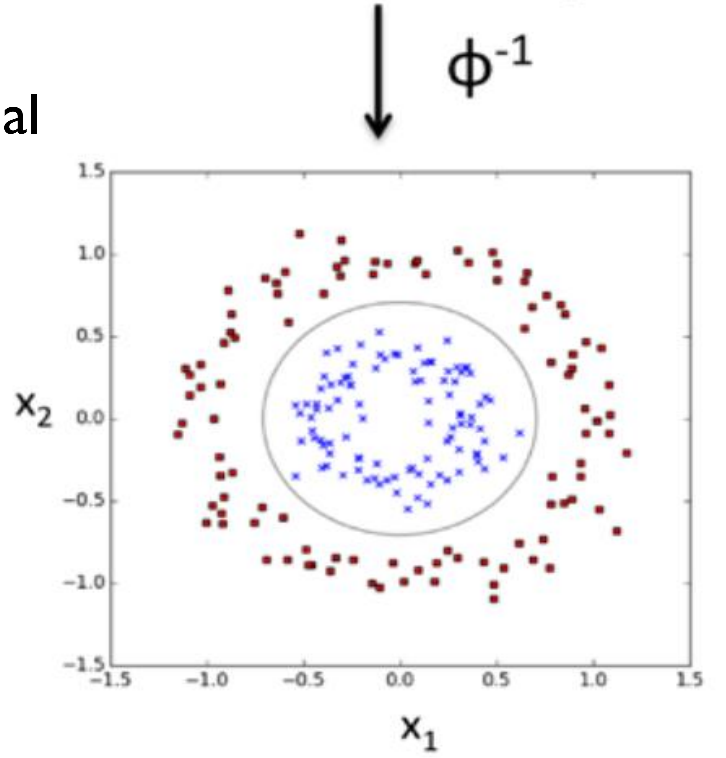
- SVM – Kernel Methods
  - Suppose the input space is 2-dimensional and we choose the following mapping:

$$(x_1, x_2) \longmapsto (x_1{}^2, x_2{}^2, \sqrt{2}x_1x_2)$$

  - Then the training example ((2, 3), -1) would map to ((4, 9, 8.5), -1)

  - Problem is that we may encounter the **curse of dimensionality**.
  - Explicit calculations can be avoided though by using **kernel functions.**





Create nonlinear combinations of the original features to project them onto a higher dimensional space via a mapping function ϕ( ) where it becomes linearly separable.

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

To solve a nonlinear problem using an SVM, we transform the training data onto a higher dimensional feature space via a mapping function $\phi(\cdot)$ and train a linear SVM model to classify the data in this new feature space. Then we can use the same mapping function $\phi(\cdot)$ to transform new, unseen data to classify it using the linear SVM model.

However, one problem with this mapping approach is that the construction of the new features is computationally very expensive, especially if we are dealing with high-dimensional data. This is where the so-called kernel trick comes into play.

The Kernel Trick [Kim http://bit.ly/1JlwFmE]

It turns out that the SVM has no need to explicitly work in the higher dimensional space at training or testing time. One can show that during training, the optimization problem only uses the training examples to compute pair-wise dot products: $< \overrightarrow{x_i}, \overrightarrow{x_j} >$ where $\overrightarrow{x_i}$, $\overrightarrow{x_j} \in \mathbb{R}^N$

Why is this significant?

It turns out that there exist functions that, given two vectors $\vec{v}$ and $\vec{w}$ in $\mathbb{R}^N$,the function **implicitly** computes the dot product between $\vec{v}$ and $\vec{w}$ in a higher-dimensional space $\mathbb{R}^M$ **without explicitly transforming** $\vec{v}$ and $\vec{w}$ to $\mathbb{R}^M$. Such functions are called **kernel** functions $\mathrm{K}(\vec{v}, \vec{w})$

The Kernel Trick [Kim http://bit.ly/1JlwFmE]

The implications are twofold:

[1] By using a kernel $\mathrm{K}(\vec{v}, \vec{w})$, we can implicitly transform datasets to a higher-dimensional $\mathbb{R}^M$ using no extra memory, and with a minimal effect on computation time.

The only effect on computation is the extra time required to compute $\mathrm{K}(\vec{v}, \vec{w})$. Depending on $\mathrm{K}(\vec{v}, \vec{w})$, this can be minimal.

[2] By virtue of [1], we can efficiently learn nonlinear decision boundaries for SVMs simply by **replacing all dot products** in the SVM computation with $\mathrm{K}(\vec{x_i}, \vec{x_j})$

10.5 Summary

- Summary

  - SVMs: supervised linear learning method to 2-class problems with numeric data
  - Objective: maximise margin between 2 classes
  - Non-separable problems are handled by mapping the vector into a higher dimensional non-linear feature space
  - Kernel functions are used to simplify calculations in the feature space
  - Limitations:
    1. Only work in real-valued spaces. We need to convert values to numeric if this is not the case
    2. Only 2 classes are permissible
    3. Hyperplanes are hard for users to understand – hence we tend to use SVMs in cases where human understanding is not required

10.5 Summary

- Summary
  - Introduction
  - Generalization
  - Perceptrons
  - Support Vector Machines

- References

  - Schölkopf, B. (2001). SVM and Kernel Methods, Neural Information Processing Systems conference.
  - Cristianini, N. and Shawe-Taylor, J. (2000),. An Introduction to Support Vector Machines. Cambridge University Press, Cambridge, UK.
  - Burges, C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 2:121-167.
  - Liu B.(2011), Web Data Mining, Springer.
  - Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). Mining of massive datasets. Cambridge University Press.