# COMS 4771
# Perceptron and Kernelization

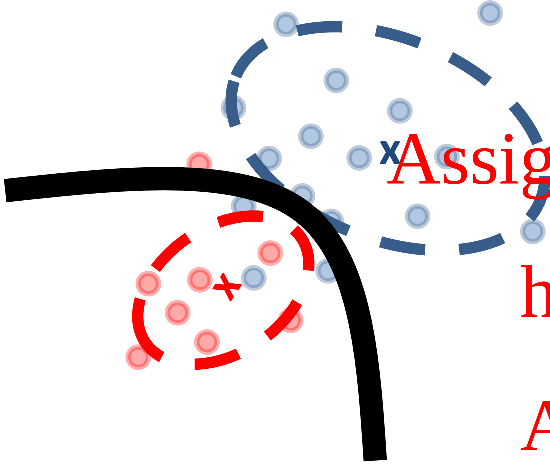


Nakul Verma

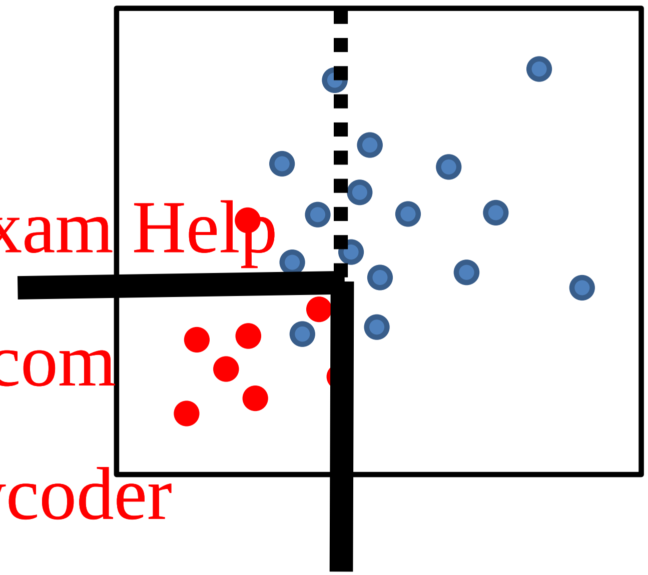
# Last time...

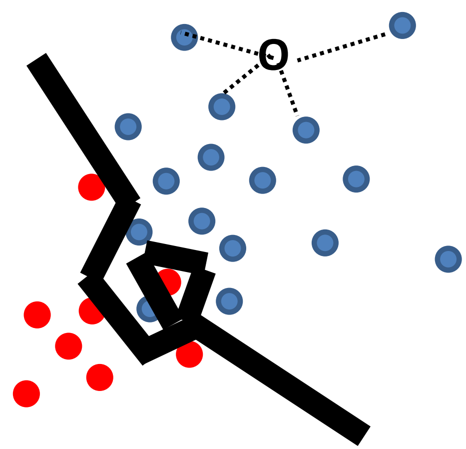
- Generative vs. Discriminative Classifiers
- Nearest Neighbor (NN) classification
- Optimality of *k*-NN
- Coping with drawbacks of *k*-NN
- Decision Trees
- The notion of overfitting in machine learning

# A Closer Look Classification



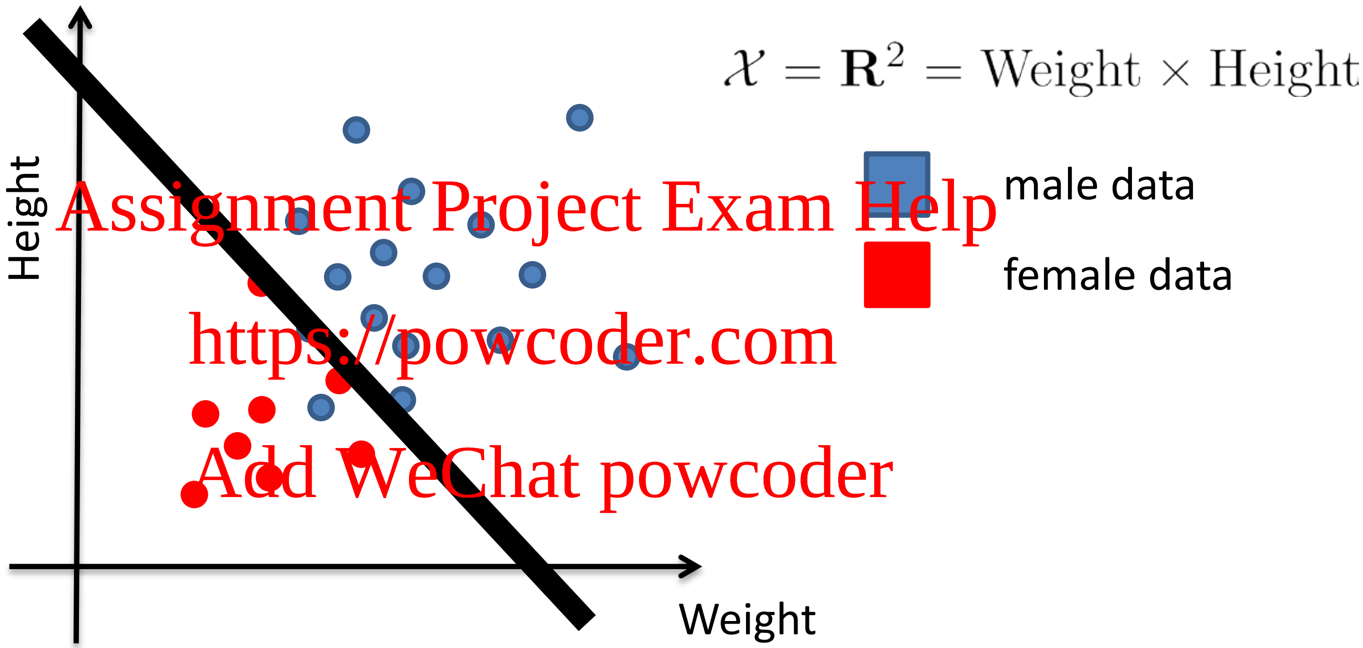





*Knowing the boundary is enough for classification*

# Linear Decision Boundary






*Assume binary classification y= {-1,+1}*

*(What happens in multi-class case?)*

# Learning Linear Decision Boundaries

*g* = decision boundary

*d=1 case:* $g(x) = w_1 x + w_0 = 0$



*general:* $g(\vec{x}) = \vec{w} \cdot \vec{x} + w_0 = 0$





*f* = linear classifier $$f(\vec{x}) := \begin{cases} +1 & \text{if } g(\vec{x}) \geq 0 \\ -1 & \text{if } g(\vec{x}) < 0 \end{cases}$$

$$= \text{sign}(\vec{w} \cdot \vec{x} + w_0)$$

*# of parameters to learn in* $\boldsymbol{R}^d$*?*

# Dealing with $w_0$

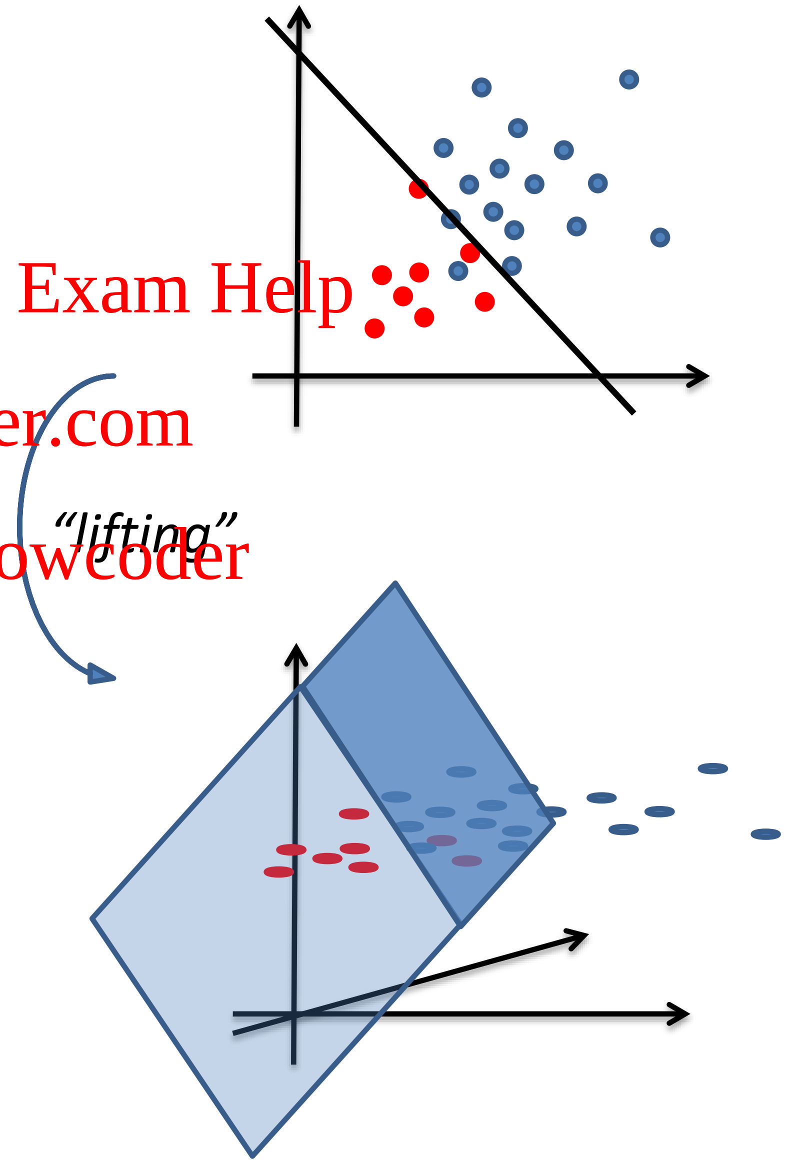
$$g(\vec{x}) = \vec{w} \cdot \vec{x} + w_0$$

$$= \underbrace{\begin{bmatrix} \vec{w} \\ w_0 \end{bmatrix}}_{\vec{w}'} \cdot \underbrace{\begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}}_{\vec{x}'}$$

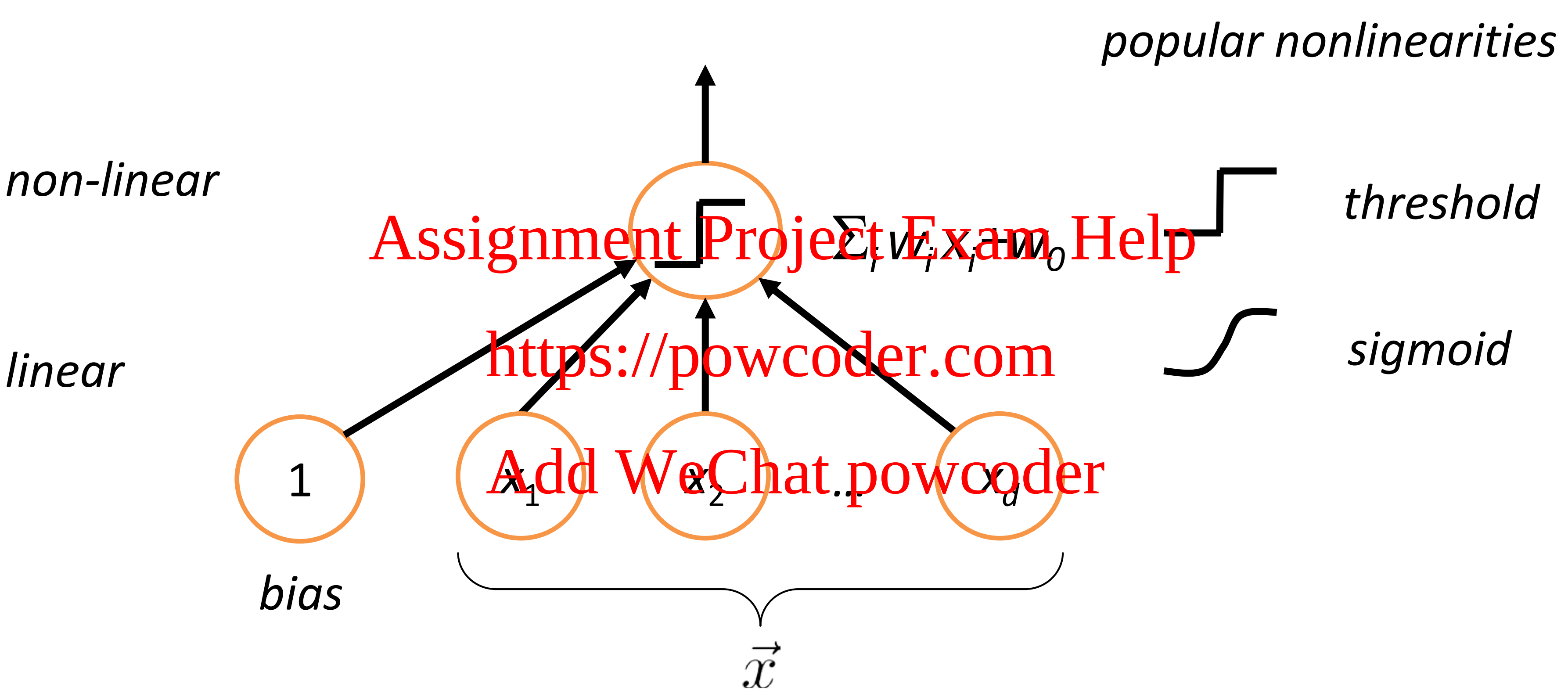
*bias*

$$g(\vec{x}') = \vec{w}' \cdot \vec{x}'$$

*homogeneous*



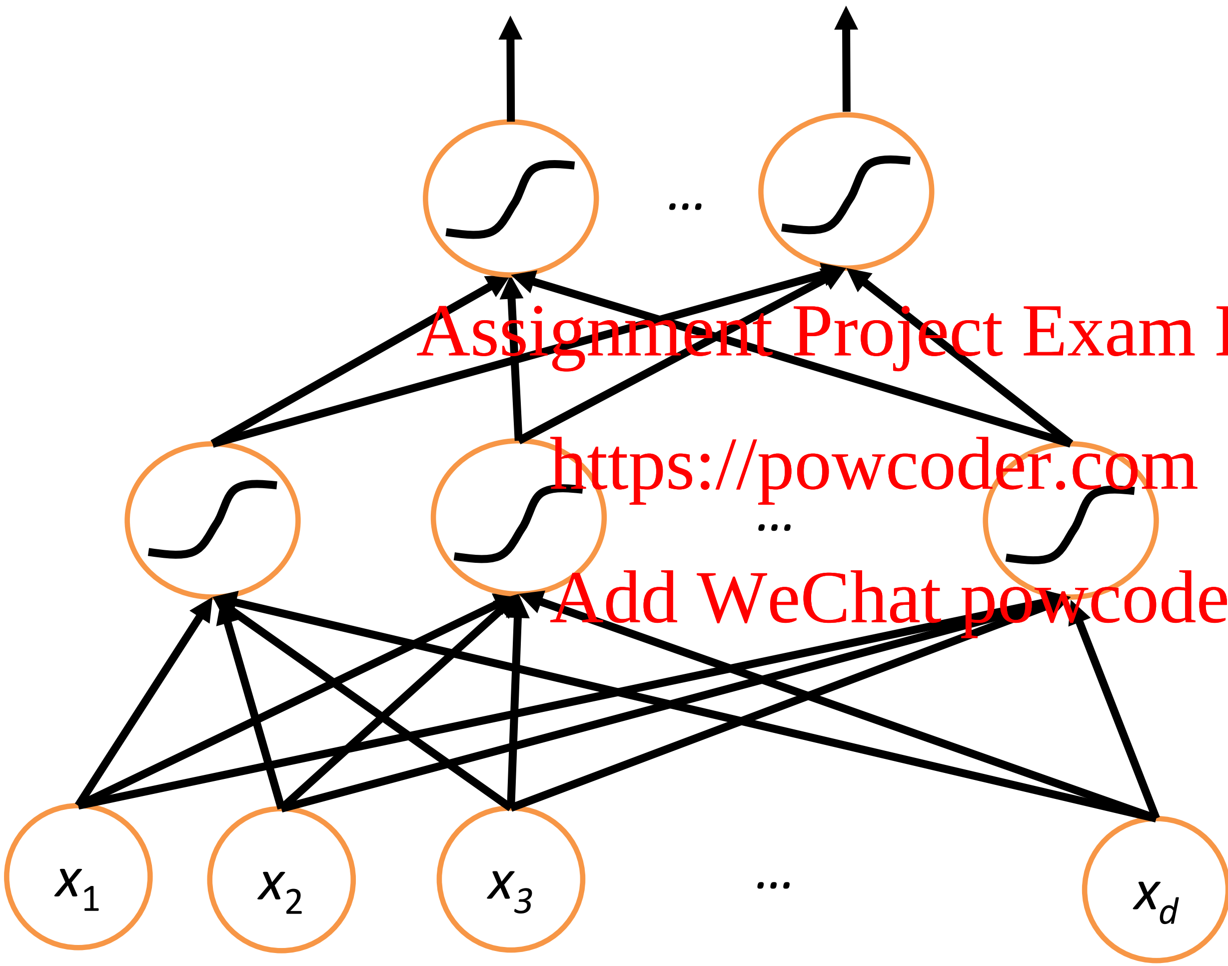
# The Linear Classifier






*A basic computational unit in a neuron*

# Can Be Combined to Make a Network






*An artificial neural network*

*Amazing fact:*

***Can approximate any smooth function!***

# How to Learn the Weights?

Given labeled training data (bias included): $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots (\vec{x}_n, y_n)$

Want: $\vec{w}$, which **minimizes** the training error, i.e.

$$\arg\min_{\vec{w}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\text{sign}(\vec{w} \cdot \vec{x}_i) \neq y_i]$$



$$= \arg\min_{\vec{w}} \sum_{\substack{x_i \\ \text{s.t. } y_i = +1}} \mathbf{1}[\vec{x}_i \cdot \vec{w} < 0] + \sum_{\substack{x_i \\ \text{s.t. } y_i = -1}} \mathbf{1}[\vec{x}_i \cdot \vec{w} \geq 0]$$



*How do we* ***minimize****?*

- Cannot use the standard technique (take derivate and examine the stationary points). Why?

Unfortunately: *NP-hard to solve or even approximate!*

# Finding Weights (Relaxed Assumptions)



Can we approximate the weights if we make reasonable assumptions?

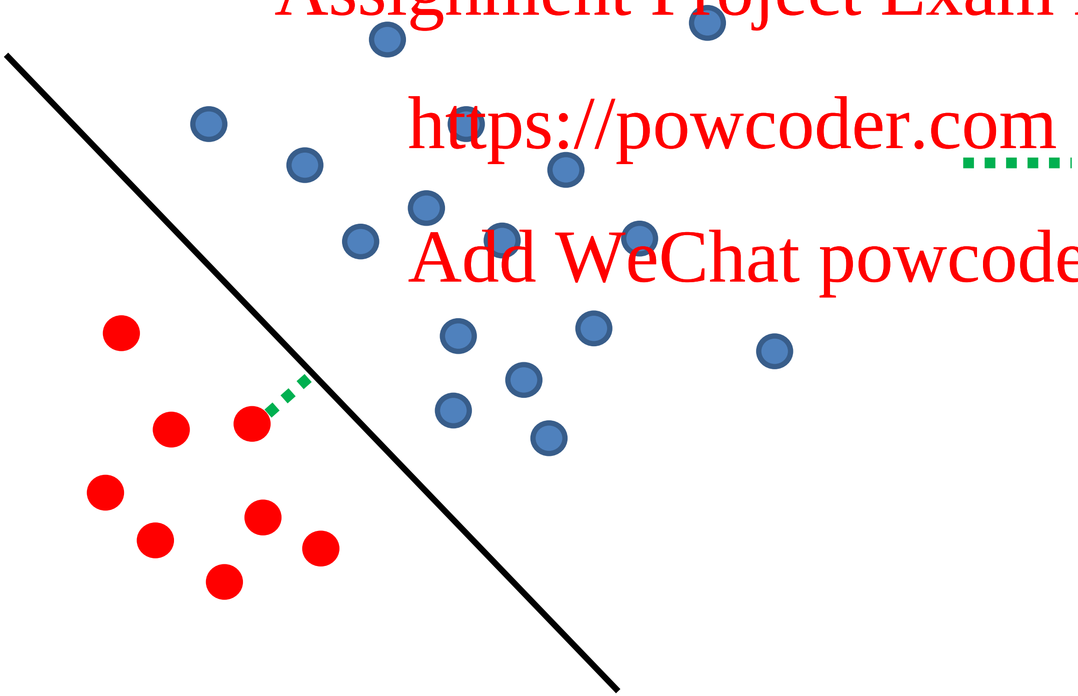


*What if the training data is linearly separable?*

# Linear Separability

Say there is a **linear** decision boundary which can **perfectly separate** the training data





*distance of the closest point to the boundary (margin $\gamma$)*

# Finding Weights

Given: labeled training data S = $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots (\vec{x}_n, y_n)$

Want to determine: is there a $\vec{w}$ which satisfies $y_i(\vec{w} \cdot \vec{x}_i) \geq 0$ (for all *i*)



*i.e., is the training data linearly separable?*



Since there are d+1 variables and |S| constraints, it is possible to solve efficiently it via a (constraint) optimization program. (How?)



*Can find it in a much **simpler** way!*

# The Perceptron Algorithm

Given: labelled training data S = $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots (\vec{x}_n, y_n)$

Initialize $\vec{w}^{(0)}$ = 0

For t = 1,2,3,…



If exists $(\vec{x}, y) \in S$ s.t. $\text{sign}(\vec{w}^{(t-1)} \cdot \vec{x}) \neq y$

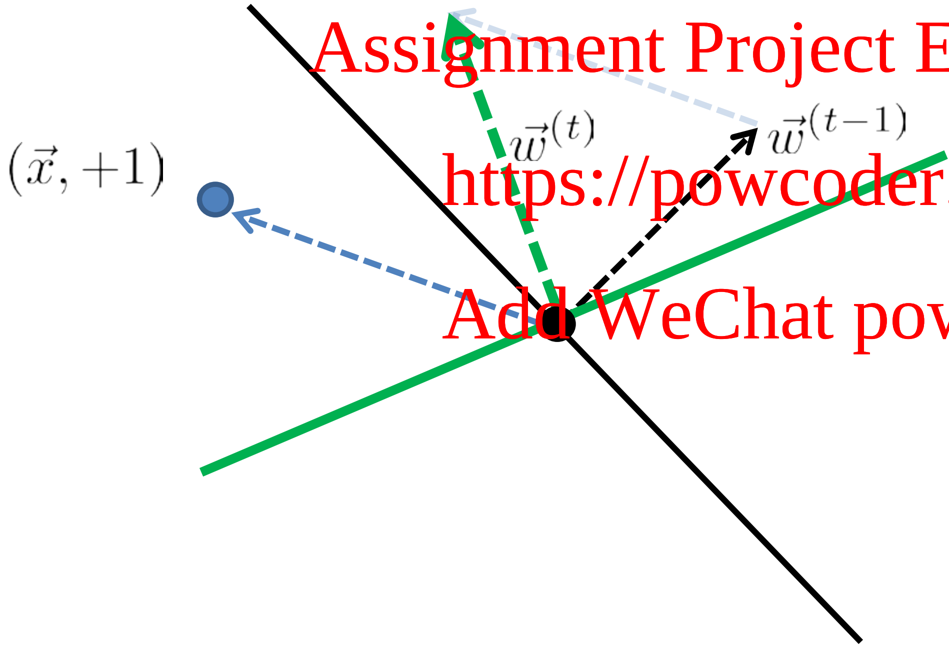


$$\vec{w}^{(t)} \leftarrow \begin{cases} \vec{w}^{(t-1)} + \vec{x} & \text{if } y = +1 \\ \vec{w}^{(t-1)} - \vec{x} & \text{if } y = -1 \end{cases} = \vec{w}^{(t-1)} + y\vec{x}$$



(terminate when no such training sample exists)

# Perceptron Algorithm: Geometry

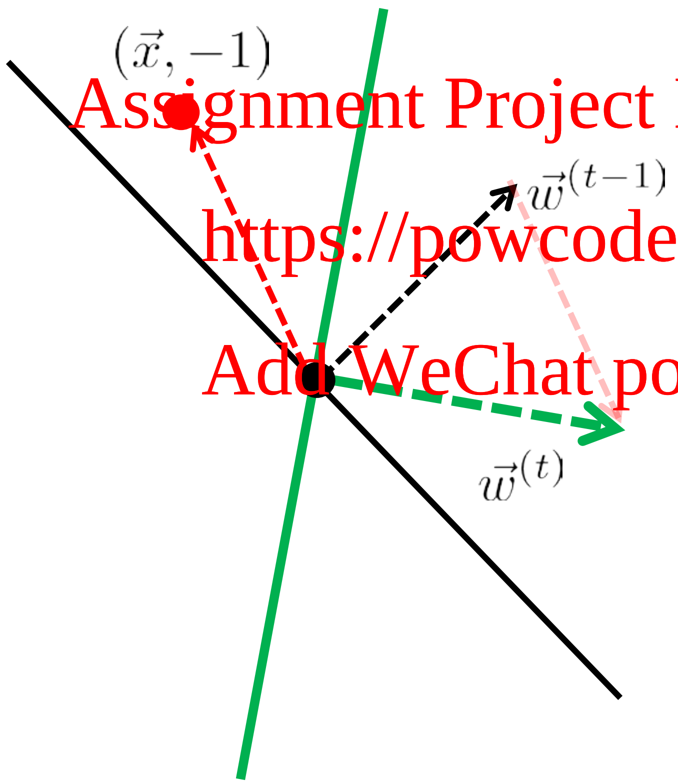



$$\text{sign}(\vec{w}^{(t-1)} \cdot \vec{x}) \neq +1$$

$$\vec{w}^{(t)} \leftarrow \vec{w}^{(t-1)} + \vec{x}$$

$$\text{sign}(\vec{w}^{t} \cdot \vec{x}) = +1$$

# Perceptron Algorithm: Geometry




$$\text{sign}(\vec{w}^{(t-1)} \cdot \vec{x}) \neq -1$$

$$\vec{w}^{(t)} \leftarrow \vec{w}^{(t-1)} - \vec{x}$$

$$\text{sign}(\vec{w}^{t} \cdot \vec{x}) = -1$$

# The Perceptron Algorithm

Input: labelled training data S = $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots (\vec{x}_n, y_n)$

Initialize $\vec{w}^{(0)}$ = 0

For t = 1,2,3,…



If exists $(\vec{x}, y) \in S$ s.t. $\text{sign}(\vec{w}^{(t-1)} \cdot \vec{x}) \neq y$



$$\vec{w}^{(t)} \leftarrow \begin{cases} \vec{w}^{(t-1)} + \vec{x} & \text{if } y = +1 \\ \vec{w}^{(t-1)} - \vec{x} & \text{if } y = -1 \end{cases} = \vec{w}^{(t-1)} + y\vec{x}$$



(terminate when no such training sample exists)

***Question:*** *Does the perceptron algorithm terminates? If so, when?*

# Perceptron Algorithm: Guarantee

**Theorem (Perceptron mistake bound):**

Assume there is a (unit length) $\vec{w}^*$ that can separate the training sample S with margin $\gamma$

Let R = $\max_{\vec{x} \in S} ||\vec{x}||$



Then, the perceptron algorithm will make at most $T := \left(\frac{R}{\gamma}\right)^2$ mistakes.





*Thus, the algorithm will terminate in T rounds!*

*umm… but what about the generalization or the test error?*

# Proof

Key quantity to analyze:

How far is $\vec{w}^{(t)}$ from $\vec{w}^*$ ?

Suppose the perceptron algorithm makes a mistake in iteration *t*, then



$$
\begin{aligned}
\vec{w}^{(t)} \cdot \vec{w}^* &= (\vec{w}^{(t-1)} + y\vec{x}) \cdot \vec{w}^* \\
&\geq \vec{w}^{(t-1)} \cdot \vec{w}^* + \gamma
\end{aligned}
$$

$$
\begin{aligned}
\|\vec{w}^{(t)}\|^2 &= \|\vec{w}^{(t-1)} + y\vec{x}\|^2 \\
&= \|\vec{w}^{(t-1)}\|^2 + 2y(\vec{w}^{(t-1)} \cdot \vec{x}) + \|y\vec{x}\|^2 \\
&\leq \|\vec{w}^{(t-1)}\|^2 + R^2
\end{aligned}
$$

# Proof (contd.)

for all iterations *t*

$$\vec{w}^{(t)} \cdot \vec{w}^{*} \geq \vec{w}^{(t-1)} \cdot \vec{w}^{*} + \gamma$$

$$\|\vec{w}^{(t)}\|^2 \leq \|\vec{w}^{(t-1)}\|^2 + R^2$$



So, after T rounds



$$T\gamma \leq \vec{w}^{(T)} \cdot \vec{w}^{*} \leq \|\vec{w}^{(T)}\|\|\vec{w}^{*}\| \leq R\sqrt{T}$$

Therefore: $$T \leq \left(\frac{R}{\gamma}\right)^2$$

$\blacksquare$

# What Good is a Mistake Bound?

- It's an upper bound on the number of mistakes made by an ***online algorithm*** on an **arbitrary sequence** of examples



*i.e. no i.i.d. assumption and not loading all the data at once!*



- Online algorithms with small mistake bounds can be used to develop classifiers with **good generalization error**!

# Other Simple Variants on the Perceptron

Voted perceptron

Average perceptron



Winnow

…

# Linear Classification

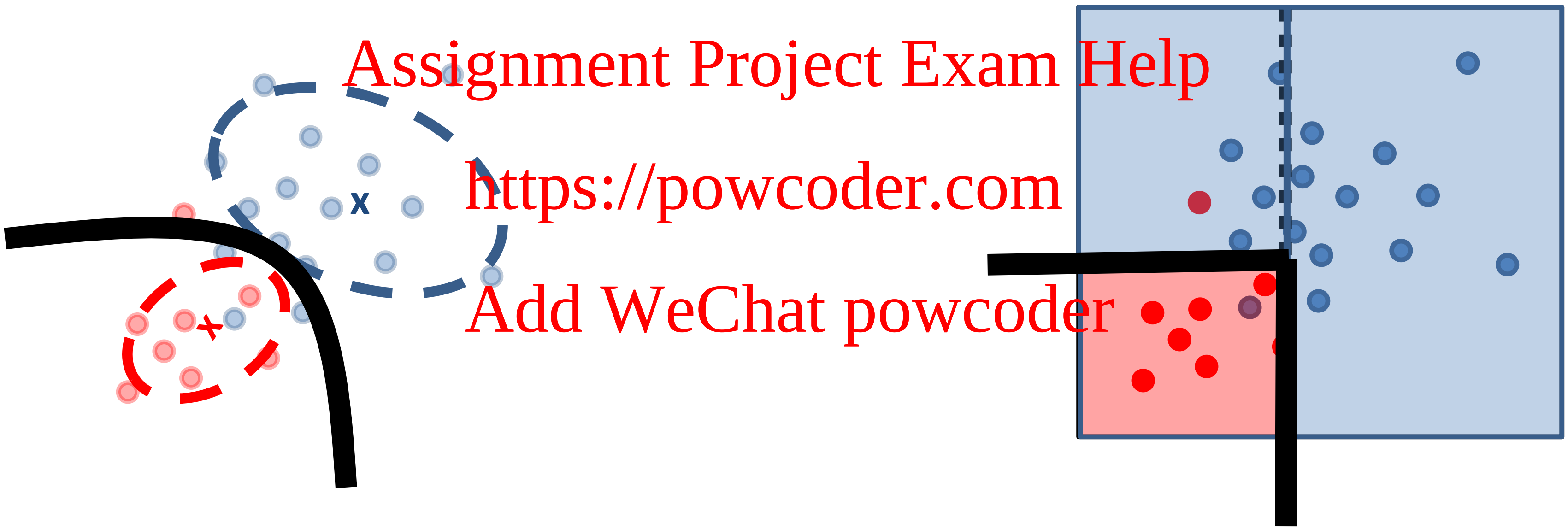


Linear classification simple,

but… *when is real-data (even approximately) linearly separable?*

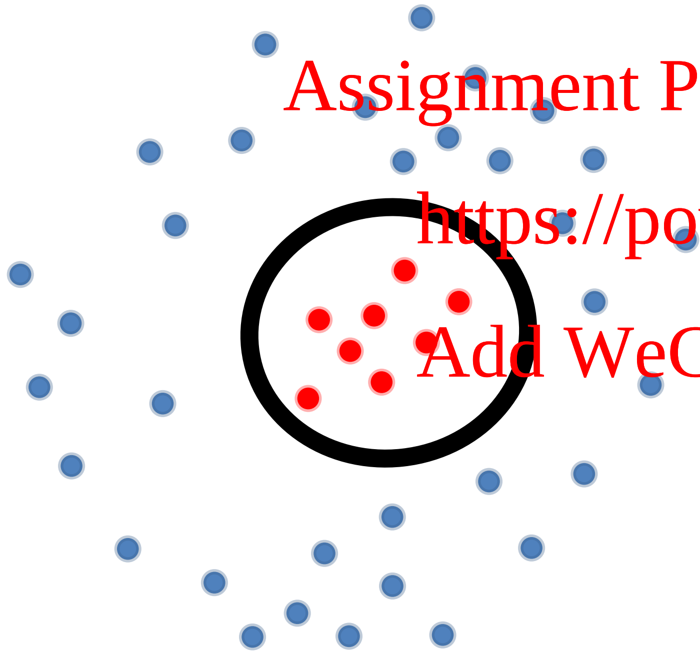
# What about non-linear decision boundaries?

Non linear decision boundaries are common:



# Generalizing Linear Classification

Suppose we have the following training data:





*d=2 case:*

$$g(\vec{x}) = w_1 x_1^2 + w_2 x_2^2 + w_0$$

*say, the decision boundary is some sort of ellipse*

*e.g. circle of radius r:*

$$w_1 = 1$$
$$w_2 = 1$$
$$w_0 = -r^2$$

separable via a circular decision boundary

$\boxed{\textit{not linear in } \vec{x} \textit{ !}}$

# But g *is* Linear in *some* Space!

$$g(\vec{x}) = w_1 \boxed{x_1^2} + w_2 \boxed{x_2^2} + w_0$$

*non linear in $x_1$ & $x_2$*

$$= w_1 \boxed{\chi_1} + w_2 \boxed{\chi_2} + w_0$$

*linear in $\chi_1$ & $\chi_2$ !*



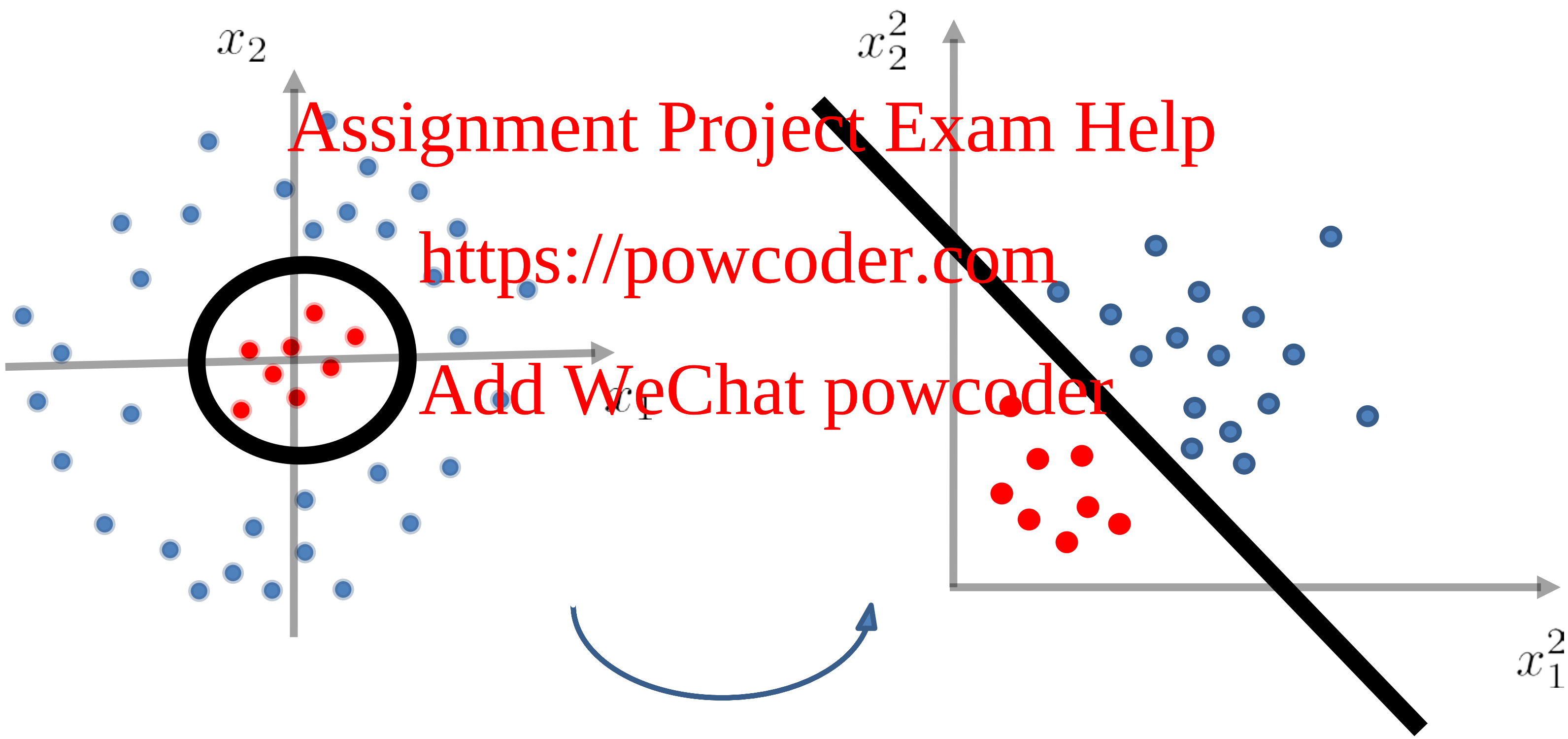
So if we apply a feature transformation on our data:



$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2)$$

Then *g* becomes linear in $\phi$ - transformed feature space!

# Feature Transformation Geometrically






$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2)$$

# Feature Transform for Quadratic Boundaries

**R**$^2$ *case: (generic quadratic boundary)*

$$g(\vec{x}) = w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + w_4 x_1 + w_5 x_2 + w_0$$

$$= \sum_{p+q \leq 2} w^{p,q} \; x_1^p x_2^q$$



feature transformation:



$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2, x_1, x_2, 1)$$



**R**$^d$ *case: (generic quadratic boundary)*

$$g(\vec{x}) = \sum_{i,j=1}^{d} \sum_{p+q \leq 2} w_{i,j}^{p,q} \; x_i^p x_j^q$$

*This captures all pairwise interactions between variables*

feature transformation:

$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2, \ldots, x_d^2, x_1 x_2, \ldots, x_{d-1} x_d, x_1, x_2, \ldots, x_d, 1)$$

# Data is Linearly Separable in some Space!

**Theorem:**

Given *n* distinct points *S* = $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$



there exists a feature transform such that for *any* labelling of *S* is linearly separable in the transformed space!



*(feature transforms are sometimes called the Kernel transforms)*

***the proof is almost trivial!***

# Proof

Given $n$ points, consider the mapping into $\mathbf{R}^n$:

$$\phi(\vec{x}_i) \mapsto \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$



*(zero in all coordinates except in coordinate i)*





Then, the decision boundary induced by linear weighting $\vec{w}^* = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$ perfectly separates the input data!

■

# Transforming the Data into Kernel Space

Pros:

Any problem becomes **linearly separable**!



Cons:



What about computation? Generic kernel transform is typically $\Omega(n)$



*Some useful kernel transforms map the input space into **infinite dimensional space**!*

What about model complexity?

*Generalization performance typically degrades with model complexity*

# The Kernel Trick (to Deal with Computation)

Explicitly working in generic Kernel space $\phi(\vec{x}_i)$ takes time $\Omega$(n)

But the dot product between two data points in kernel space can be computed relatively quickly



$$\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$$ *can compute fast*



**Examples:**

- quadratic kernel transform for data in **R**$^d$



*explicit transform* $O(d^2)$

$$\vec{x} \mapsto (x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, 1)$$

*dot products* *O(d)* $(1 + \vec{x}_i \cdot \vec{x}_j)^2$

- RBF (radial basis function) kernel transform for data in **R**$^d$

*explicit transform* infinite dimension! $\vec{x} \mapsto \left(\exp(-\|\vec{x} - \alpha\|^2)\right)_{\alpha \in \mathbb{R}^d}$

dot products *O(d)* $\exp(-\|\vec{x}_i - \vec{x}_j\|^2)$

# The Kernel Trick

The trick is to perform classification in such a way that it **only accesses the data** in terms of dot products (so it can be done quicker)

**Example:** the `kernel Perceptron'



*Recall:* $\vec{w}^{(t)} \leftarrow \vec{w}^{(t-1)} + y\vec{x}$



*Equivalently* $\vec{w} = \sum_{k=1}^{n} \alpha_k y_k \vec{x}_k$ *$\alpha_k$ = # of times mistake was made on $x_k$*



*Thus, classification becomes*

$$f(\vec{x}) := \mathrm{sign}(\vec{w} \cdot \vec{x}) \;= \mathrm{sign}\Big(\vec{x} \cdot \sum_{k=1}^{n} \alpha_k y_k \vec{x}_k\Big) \;= \mathrm{sign}\Big(\sum_{k=1}^{n} \alpha_k y_k (\vec{x}_k \cdot \vec{x})\Big)$$

*Only accessing data in terms of dot products!*

# The Kernel Trick: for Perceptron

*classification in original space:*

$$f(\vec{x}) = \text{sign}\Big(\sum_{k=1}^{n} \alpha_k y_k (\vec{x}_k \cdot \vec{x})\Big)$$

*If we were working in the transformed Kernel space, it would have been*

$$f(\phi(\vec{x})) = \text{sign}\Big(\sum_{k=1}^{n} \alpha_k y_k (\phi(\vec{x}_k) \cdot \phi(\vec{x}))\Big)$$



**Algorithm:**

Initialize $\vec{\alpha}$ = 0

For t = 1,2,3,…, T

If exists $(\vec{x}_i, y_i) \in S$ s.t. $\text{sign}\Big(\sum_{k=1}^{n} \alpha_k y_k (\phi(\vec{x}_k) \cdot \phi(\vec{x}_i))\Big) \neq y_i$

$\alpha_i \leftarrow \alpha_i + 1$

*implicitly working in non-linear kernel space!*

# The Kernel Trick: Significance



$$\sum_{k=1}^{n} \alpha_k y_k \left(\phi(\vec{x}_k) \cdot \phi(\vec{x})\right)$$

*dot products are a measure of similarity*





***Can be replaced by any user-defined measure of similarity!***

*So, we can work in any user-defined non-linear space* ***implicitly*** ***without*** *the potentially heavy computational cost*

# What We Learned…

- Decision boundaries for classification
- Linear decision boundary (linear classification)
- The Perceptron algorithm
- Mistake bound for the perceptron
- Generalizing to non-linear boundaries (via Kernel space)
- Problems become linear in Kernel space
- The Kernel trick to speed up computation

# Questions?

# Next time…

Support Vector Machines (SVMs)!