Non-linear Models-I

CS771: Introduction to Machine Learning

Purushottam Kar

Outline of discussion coming up

- Introduction to Kernels
  - Why/when are kernels used?
  - What are kernels?
  - How are kernels used?
- Using kernels to perform
  - Supervised learning tasks: classification
- Next lecture:
  - Supervised learning tasks: regression
  - Data Modelling tasks: clustering, dimensionality reduction
- How PML and FA techniques change in order to use kernels

Why/when are kernels used?

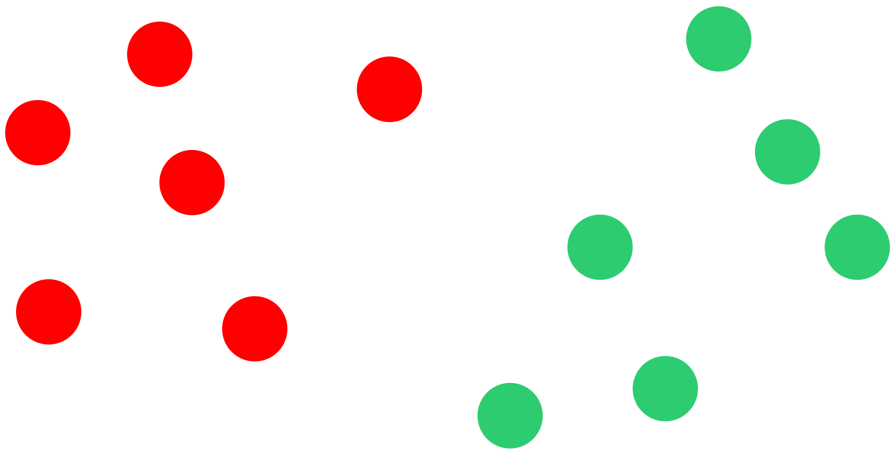
Why/when are kernels used?

Why/when are kernels used?

- Whenever linear models poorly fit your data

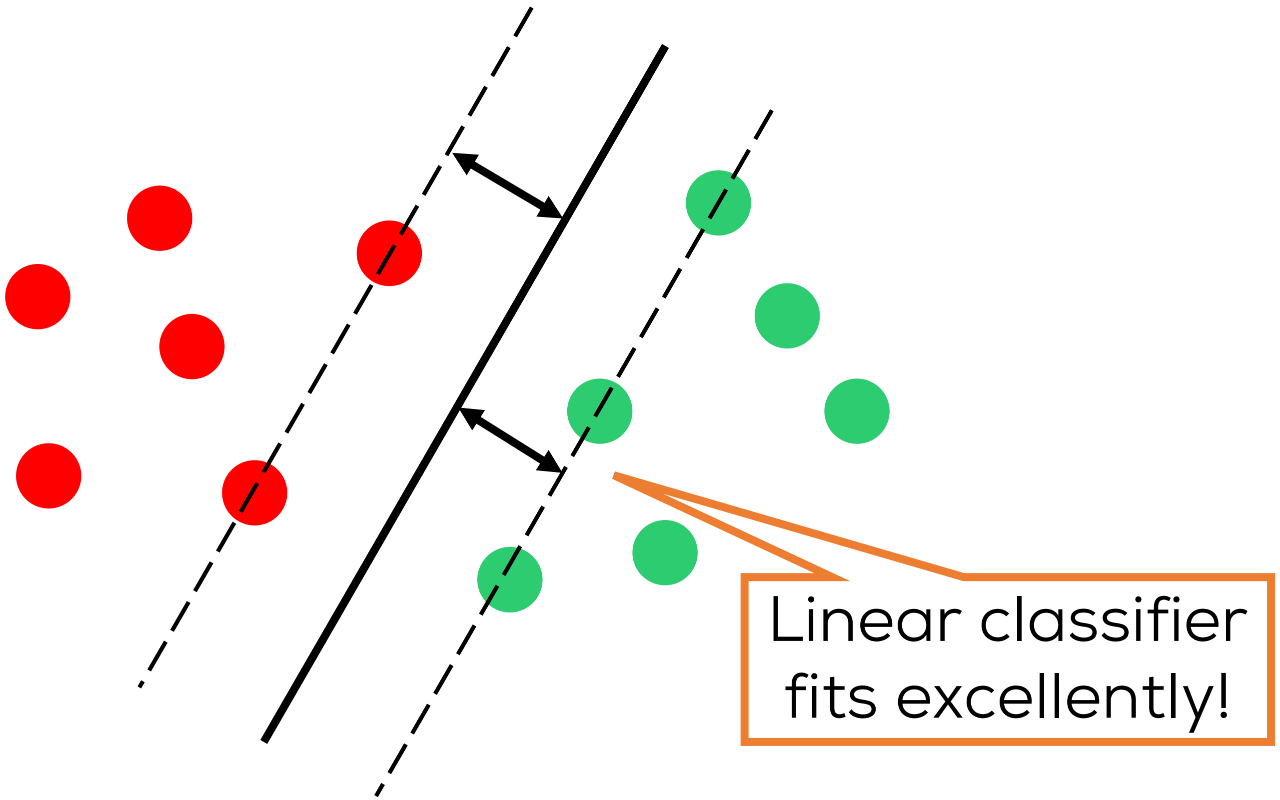
Why/when are kernels used?

- Whenever linear models poorly fit your data



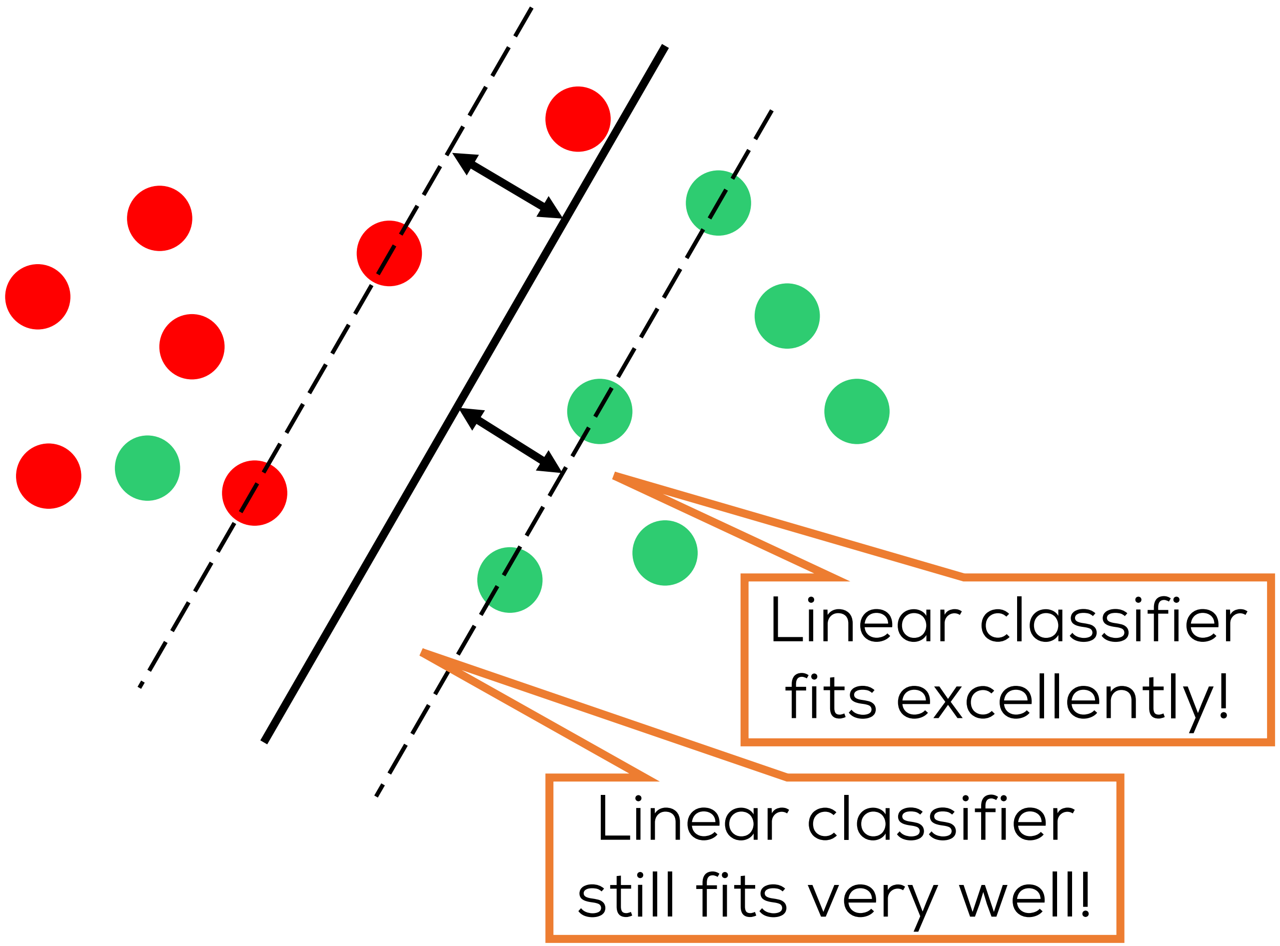
Why/when are kernels used?

- Whenever linear models poorly fit your data



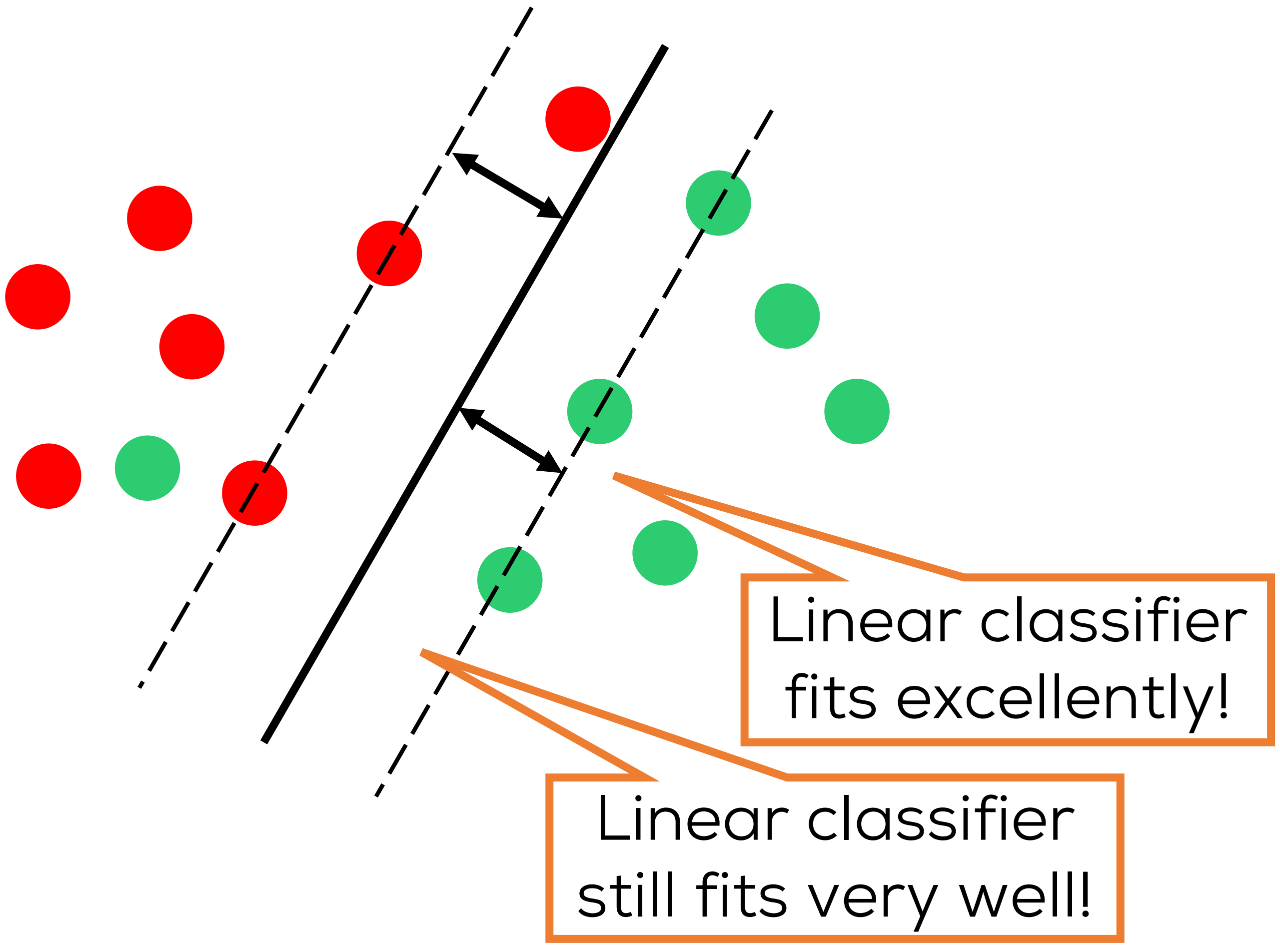
Why/when are kernels used?

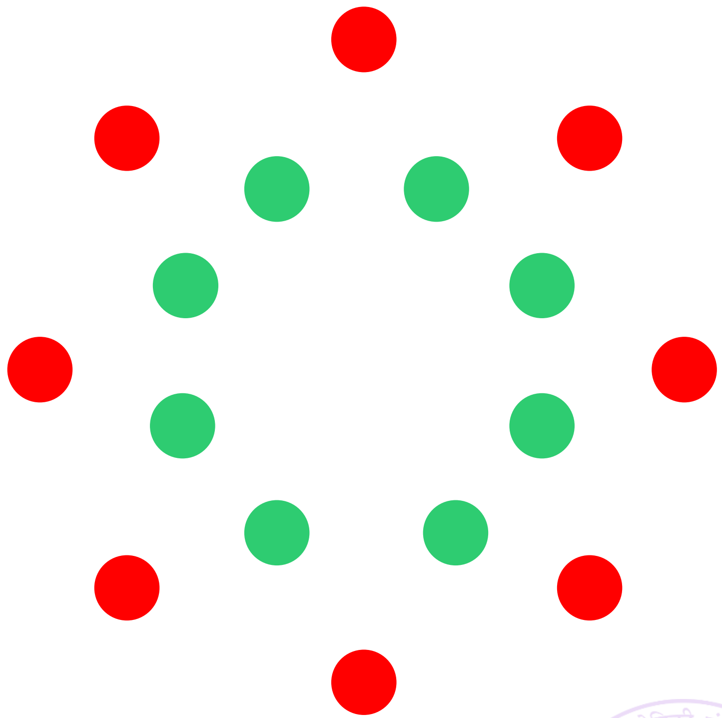
- Whenever linear models poorly fit your data



Why/when are kernels used?

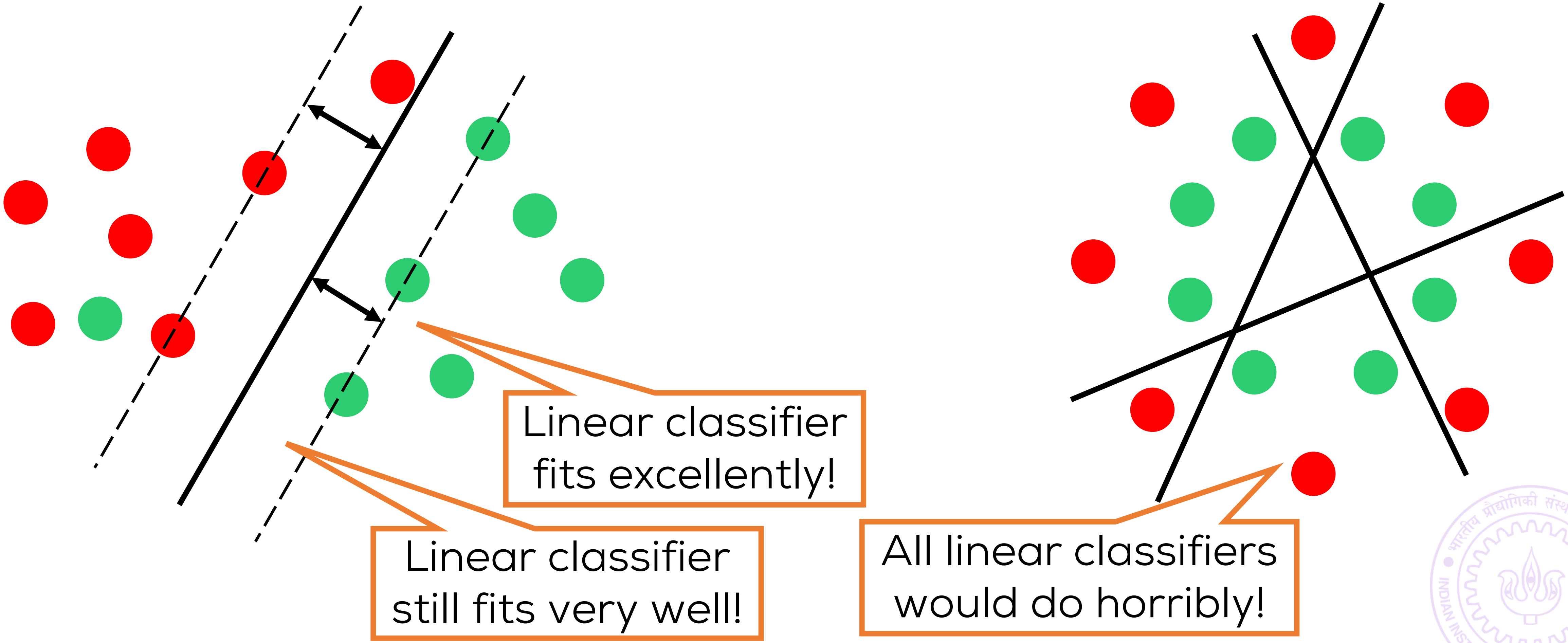
- Whenever linear models poorly fit your data





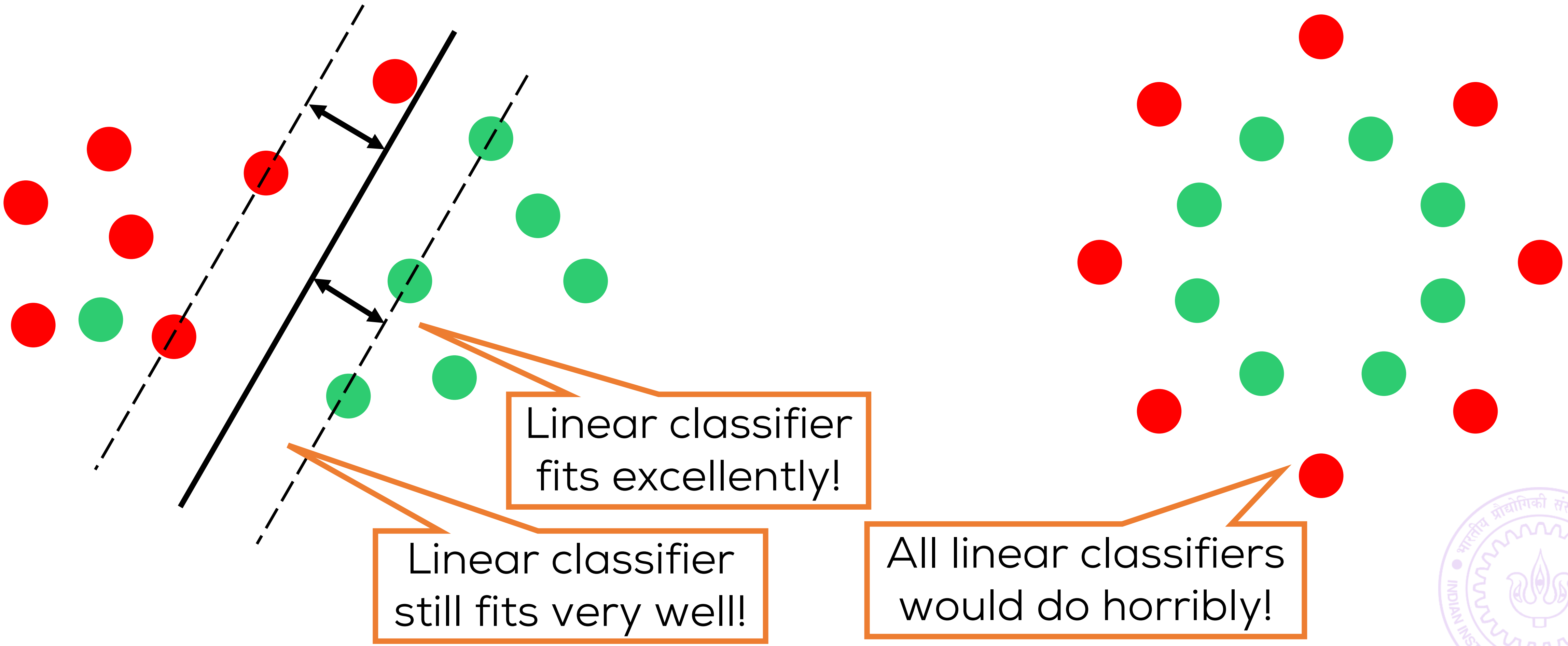
Why/when are kernels used?

- Whenever linear models poorly fit your data



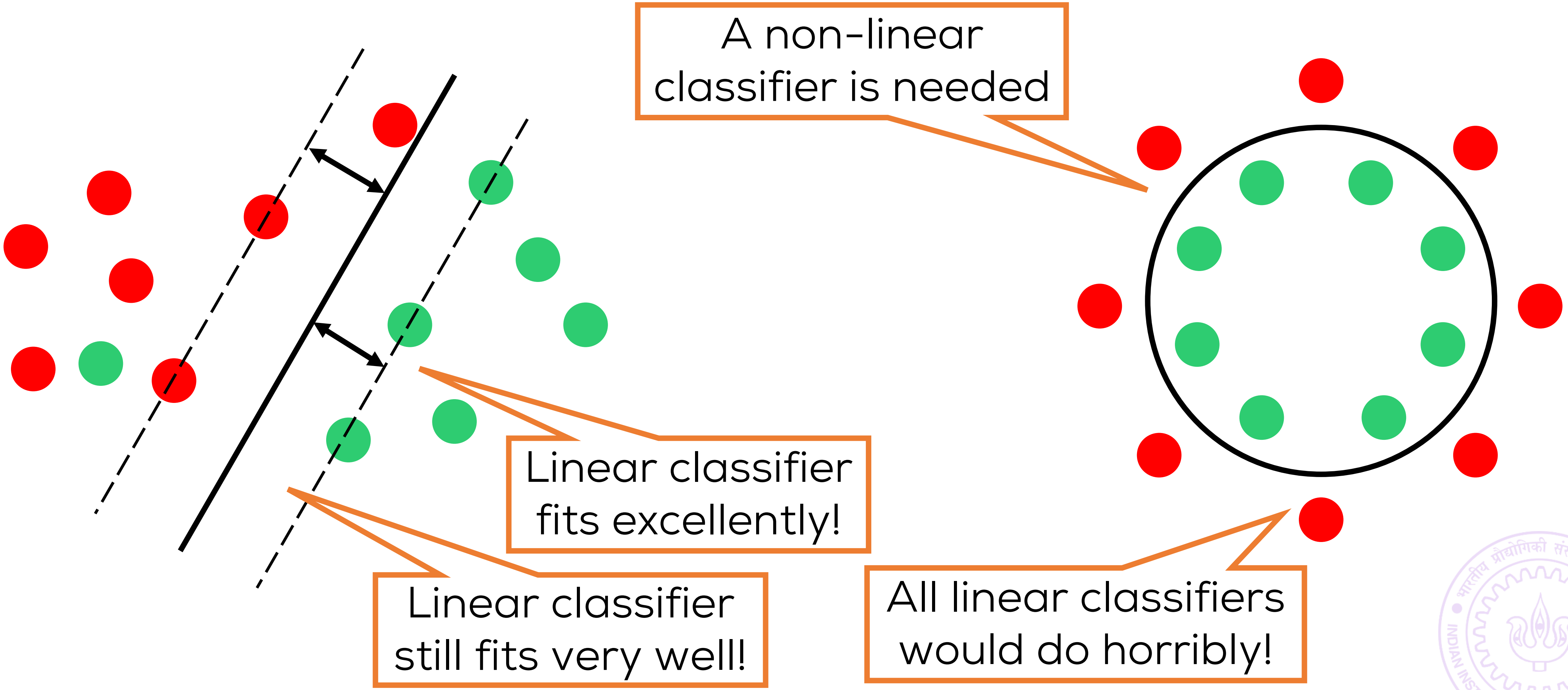
Why/when are kernels used?

- Whenever linear models poorly fit your data



Why/when are kernels used?

- Whenever linear models poorly fit your data



Why/when are kernels used?

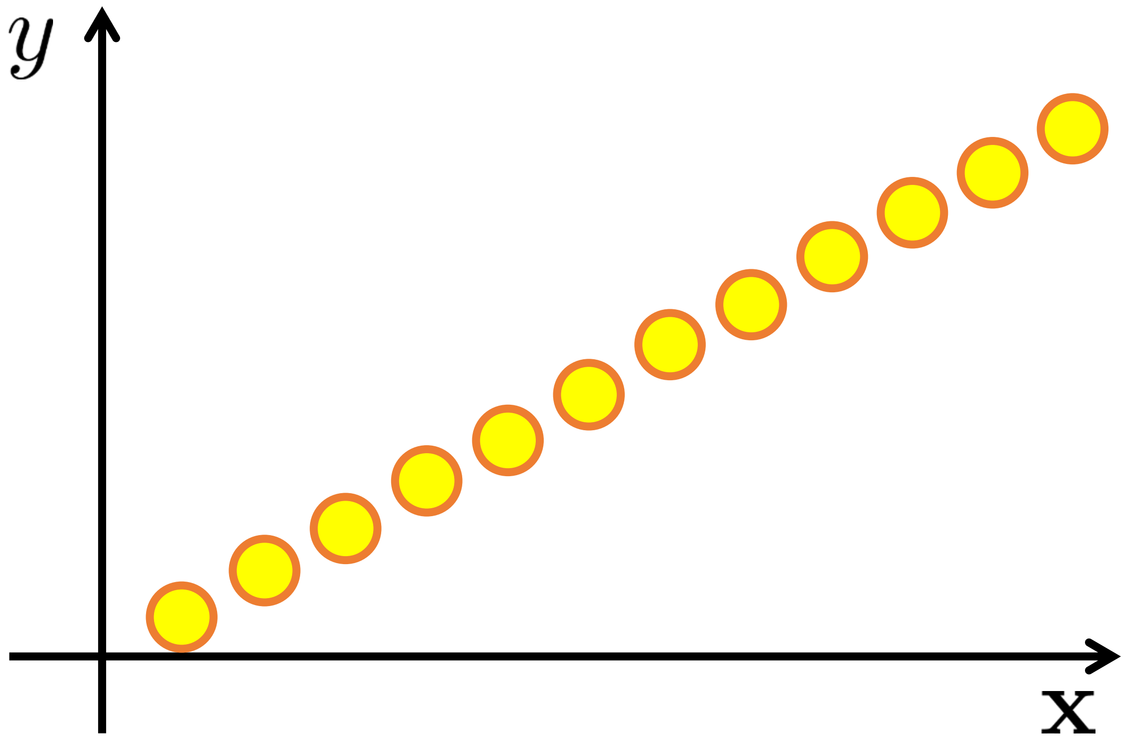
- Whenever linear models poorly fit your data

Why/when are kernels used?

- Whenever linear models poorly fit your data

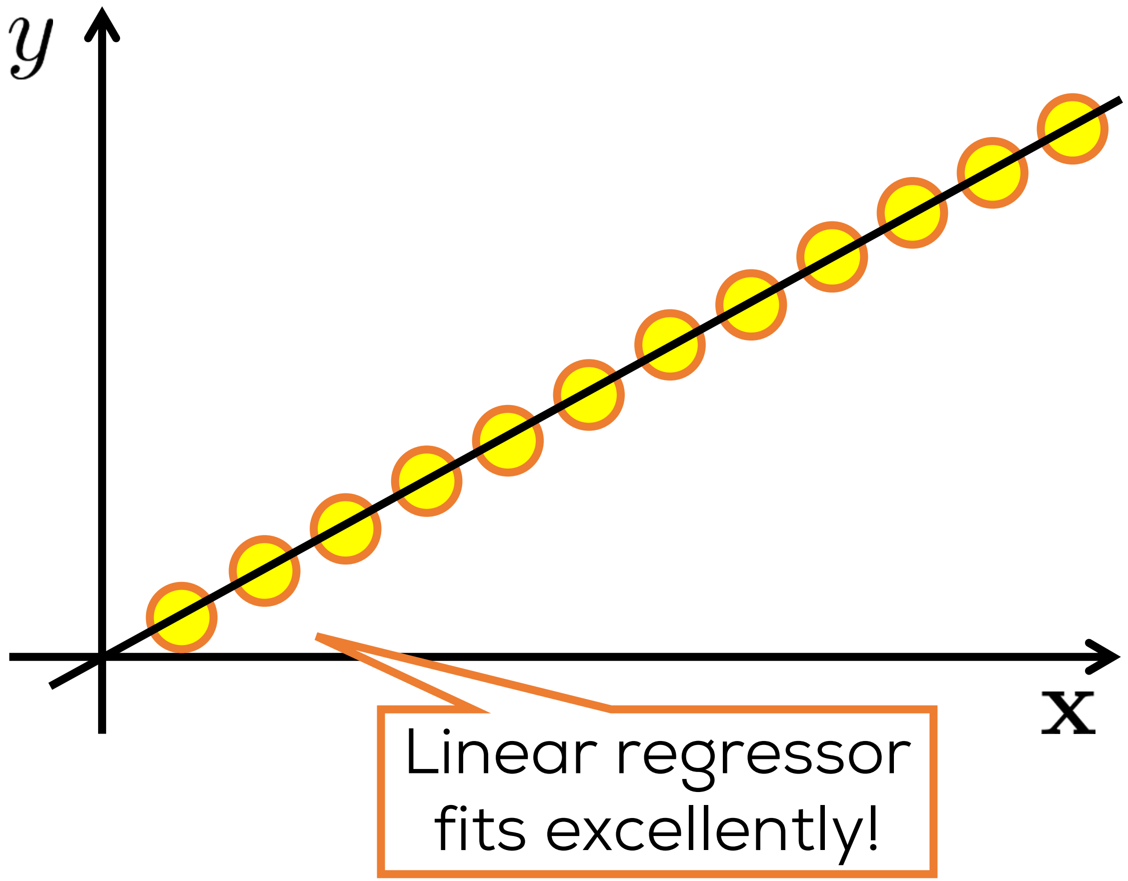
Why/when are kernels used?

- Whenever linear models poorly fit your data



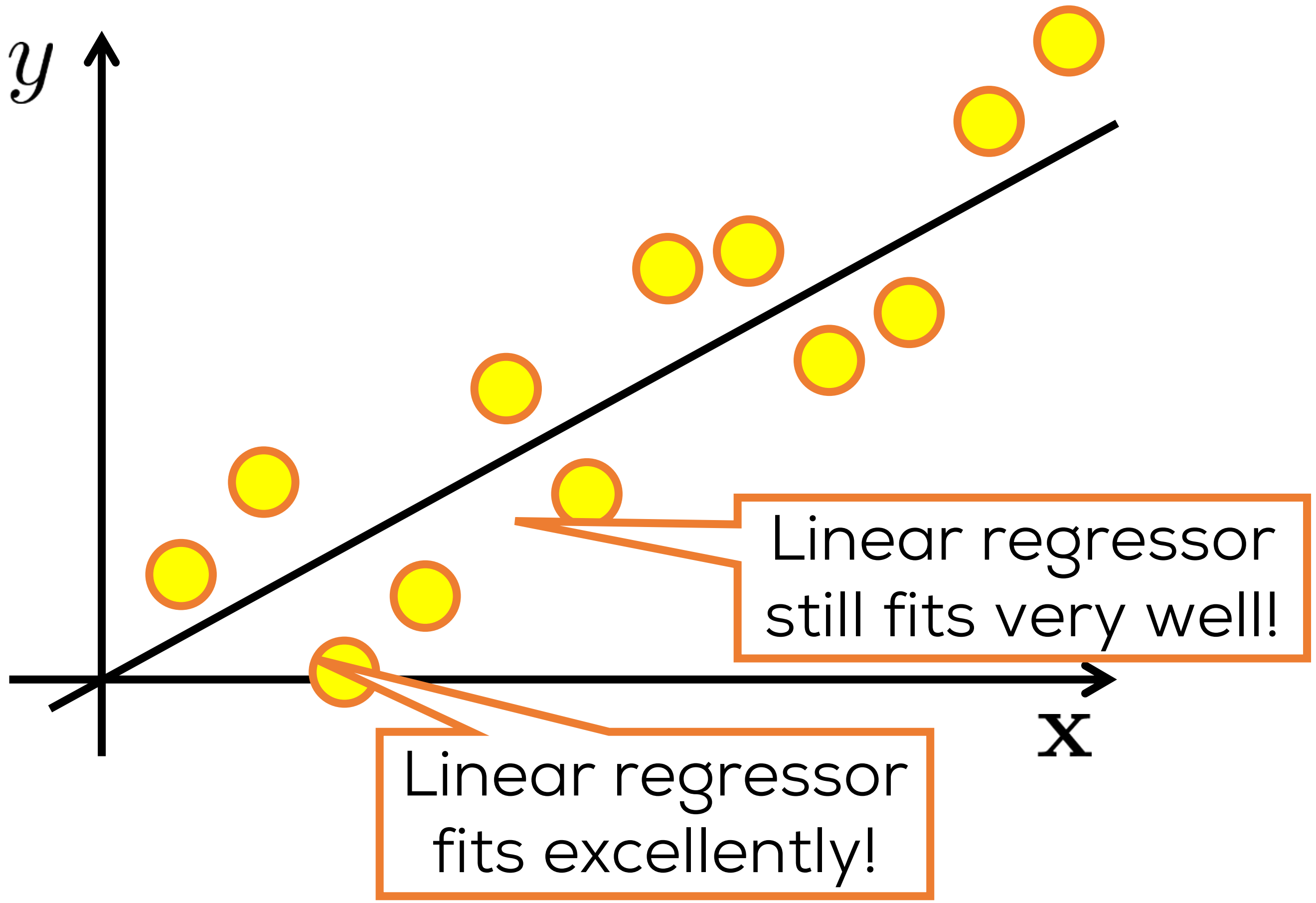
Why/when are kernels used?

- Whenever linear models poorly fit your data



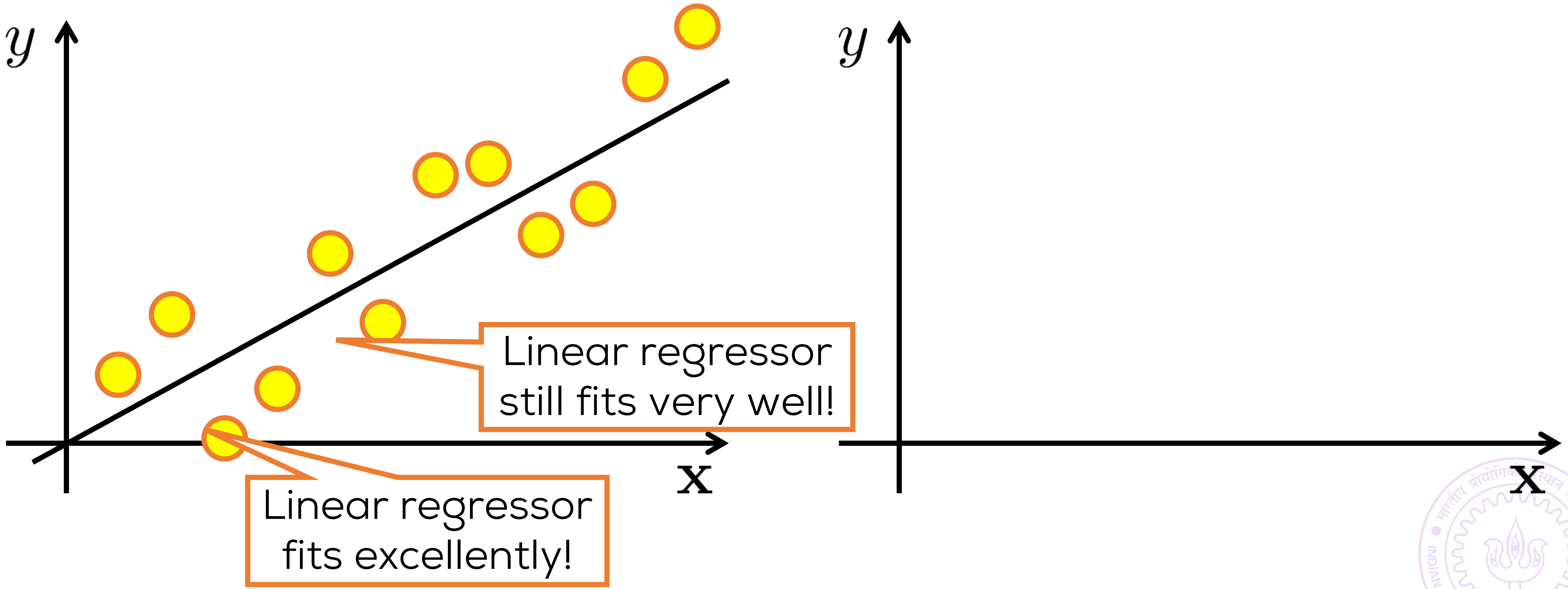
Why/when are kernels used?

- Whenever linear models poorly fit your data



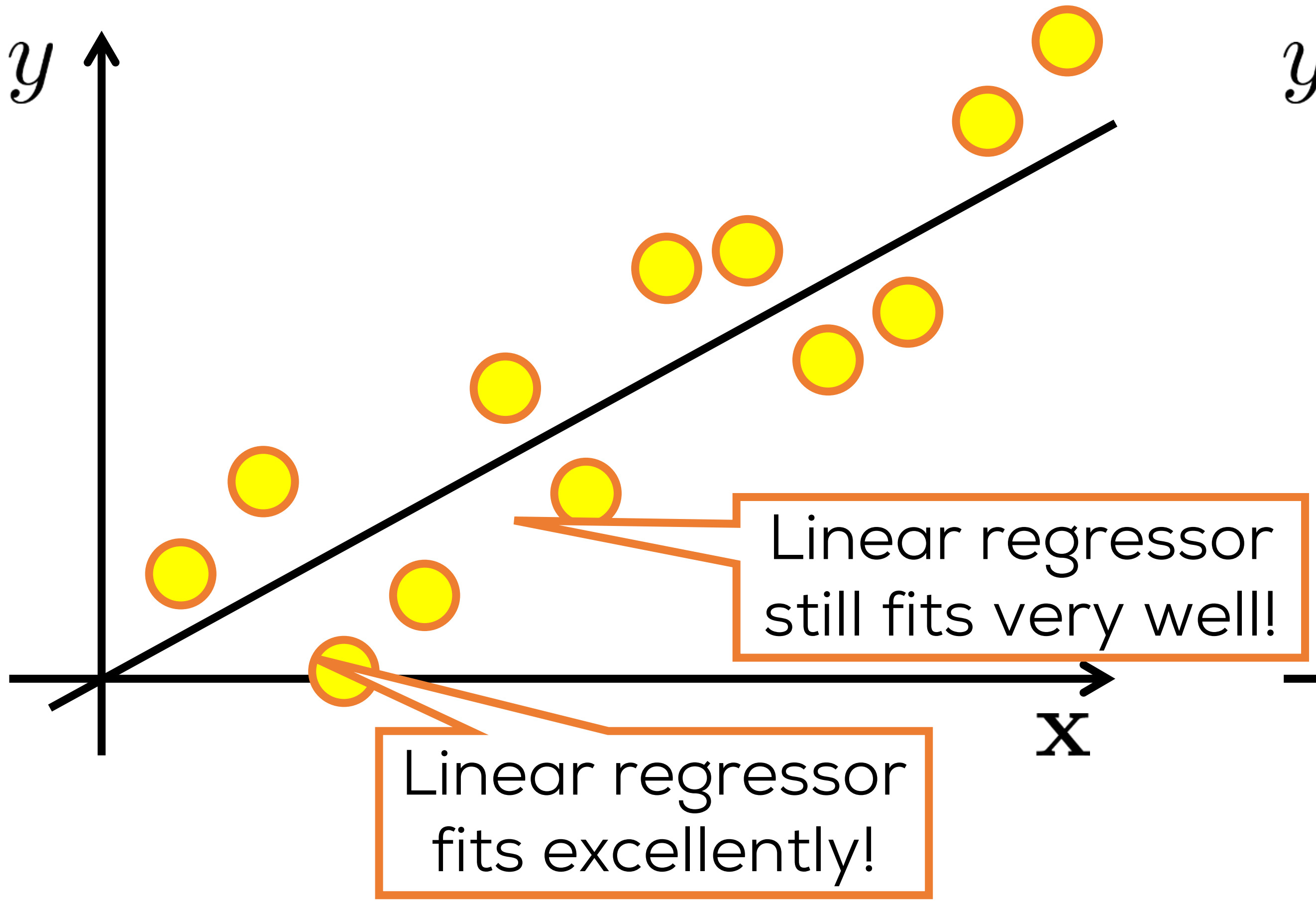
Why/when are kernels used?

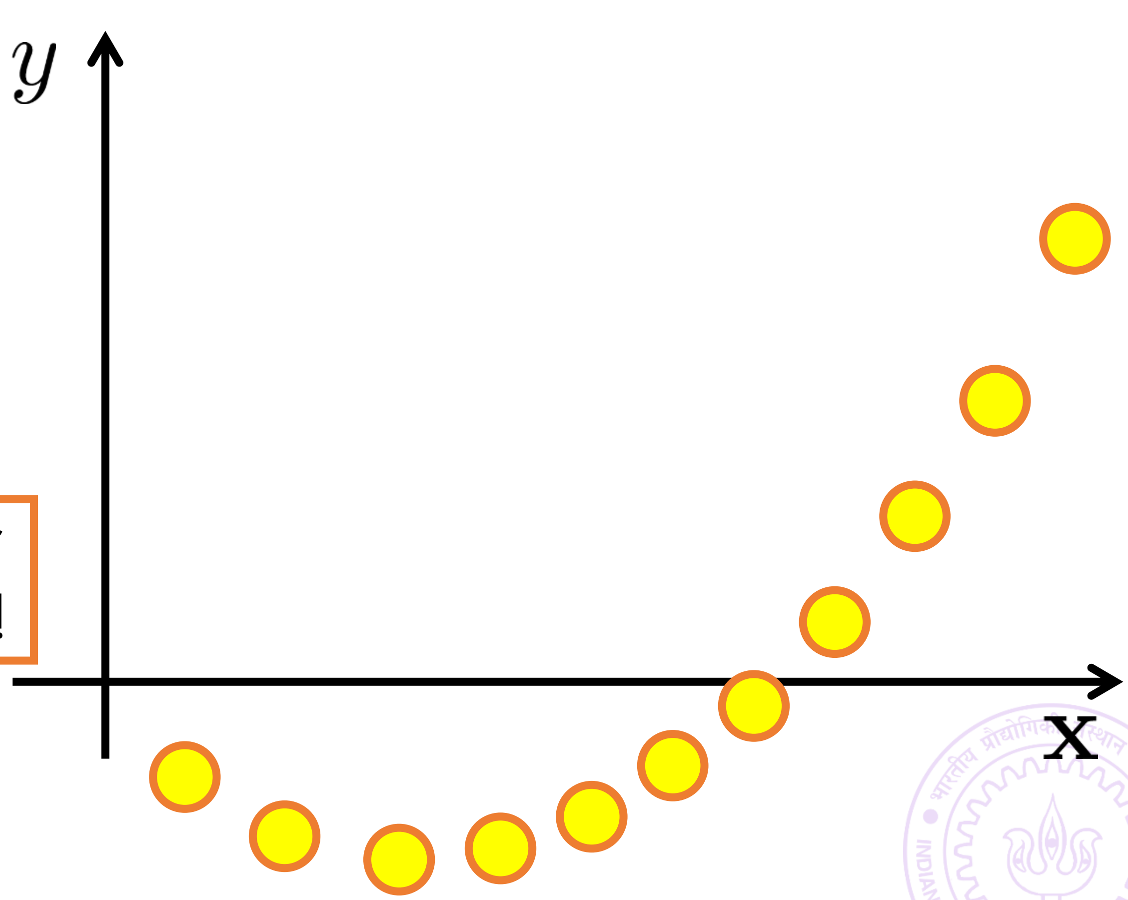
- Whenever linear models poorly fit your data



Why/when are kernels used?

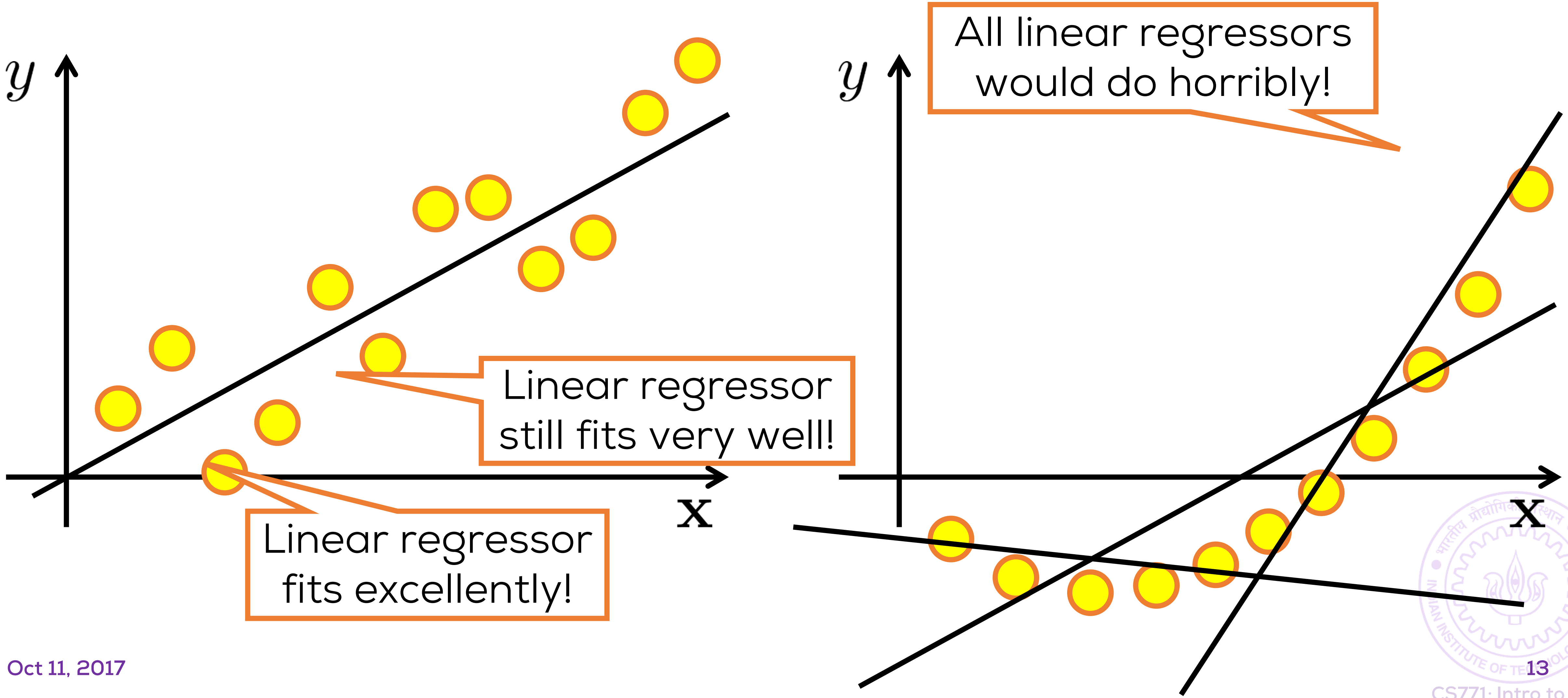
- Whenever linear models poorly fit your data





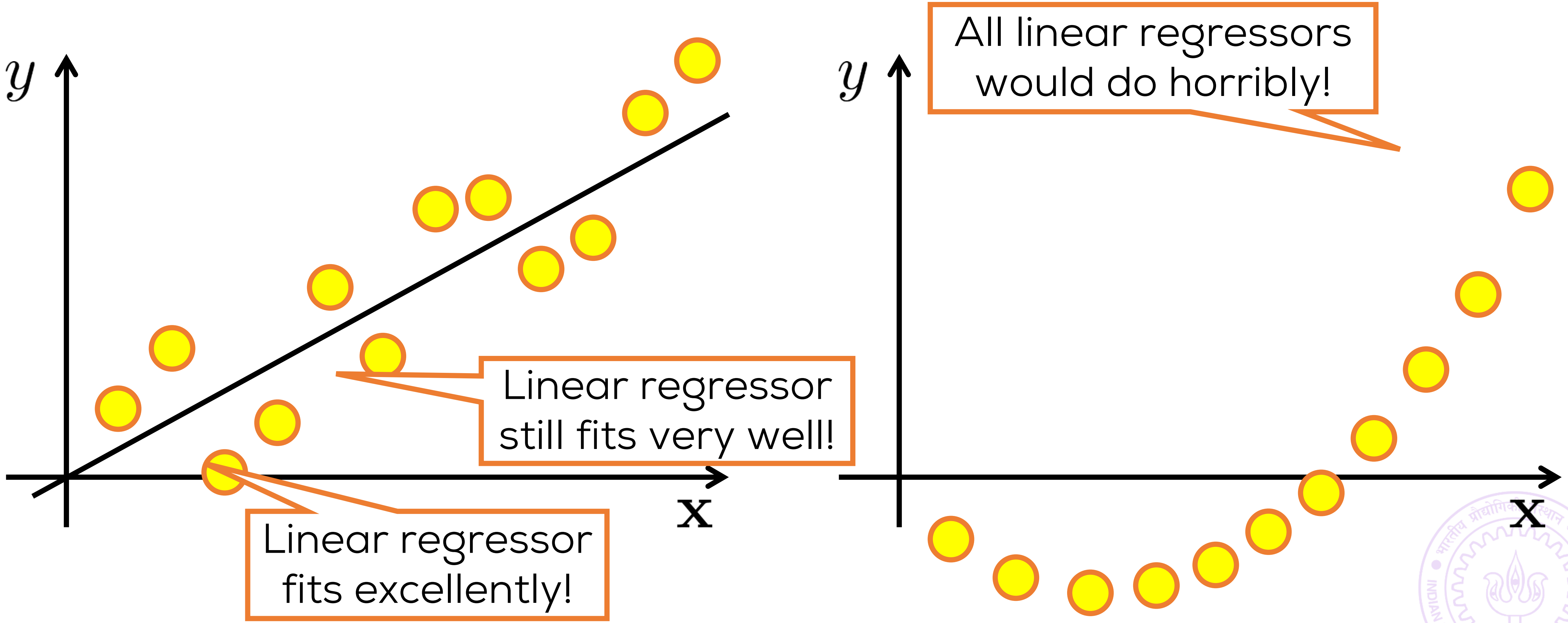
Why/when are kernels used?

- Whenever linear models poorly fit your data



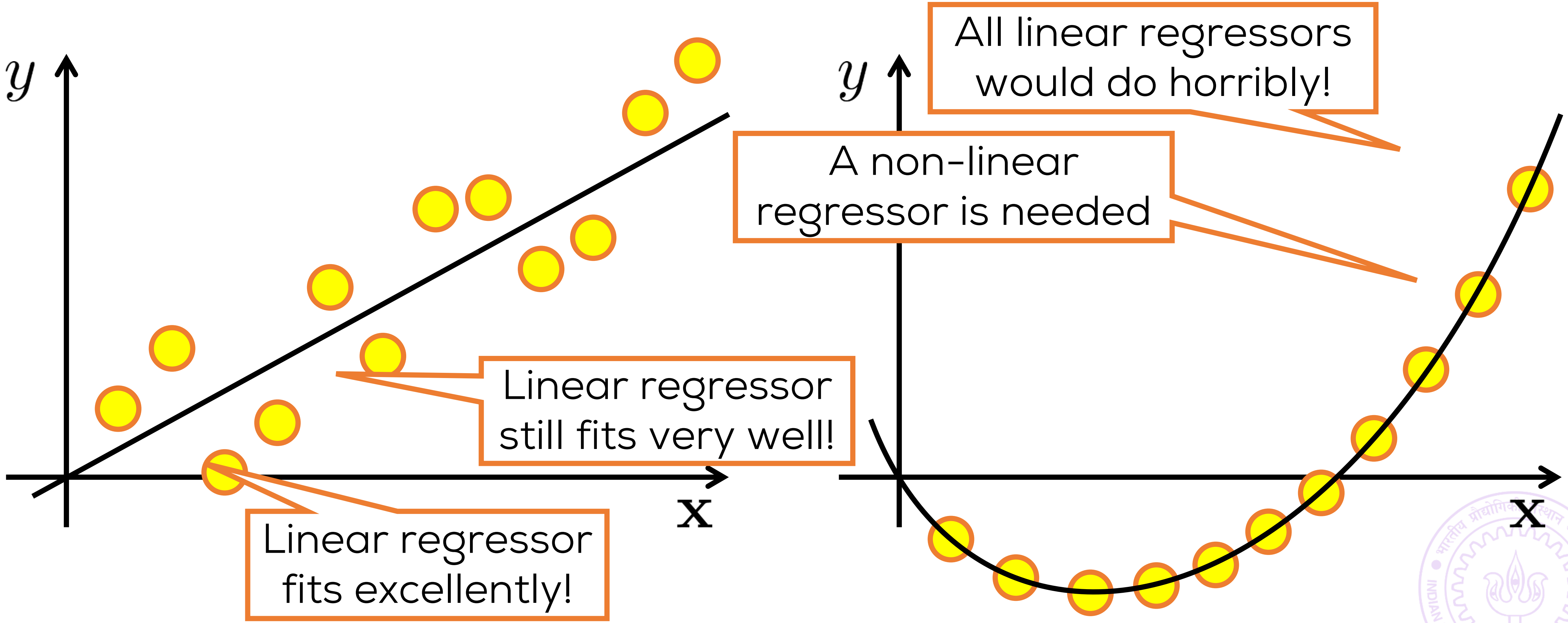
Why/when are kernels used?

- Whenever linear models poorly fit your data



Why/when are kernels used?

- Whenever linear models poorly fit your data

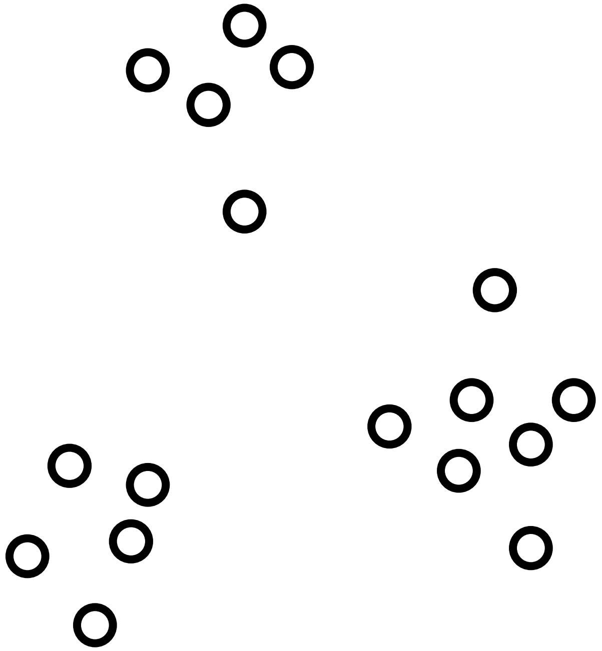


Why/when are kernels used?

- Whenever linear models poorly fit your data

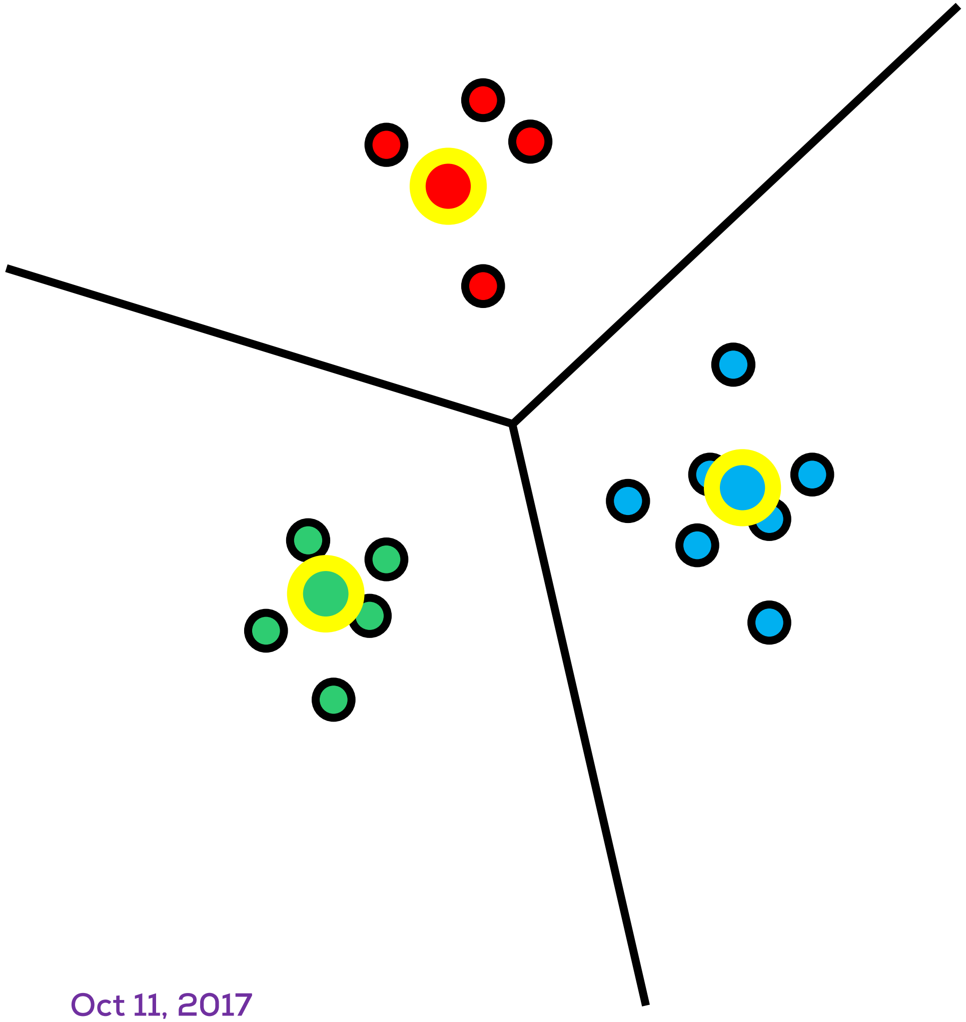
Why/when are kernels used?

- Whenever linear models poorly fit your data



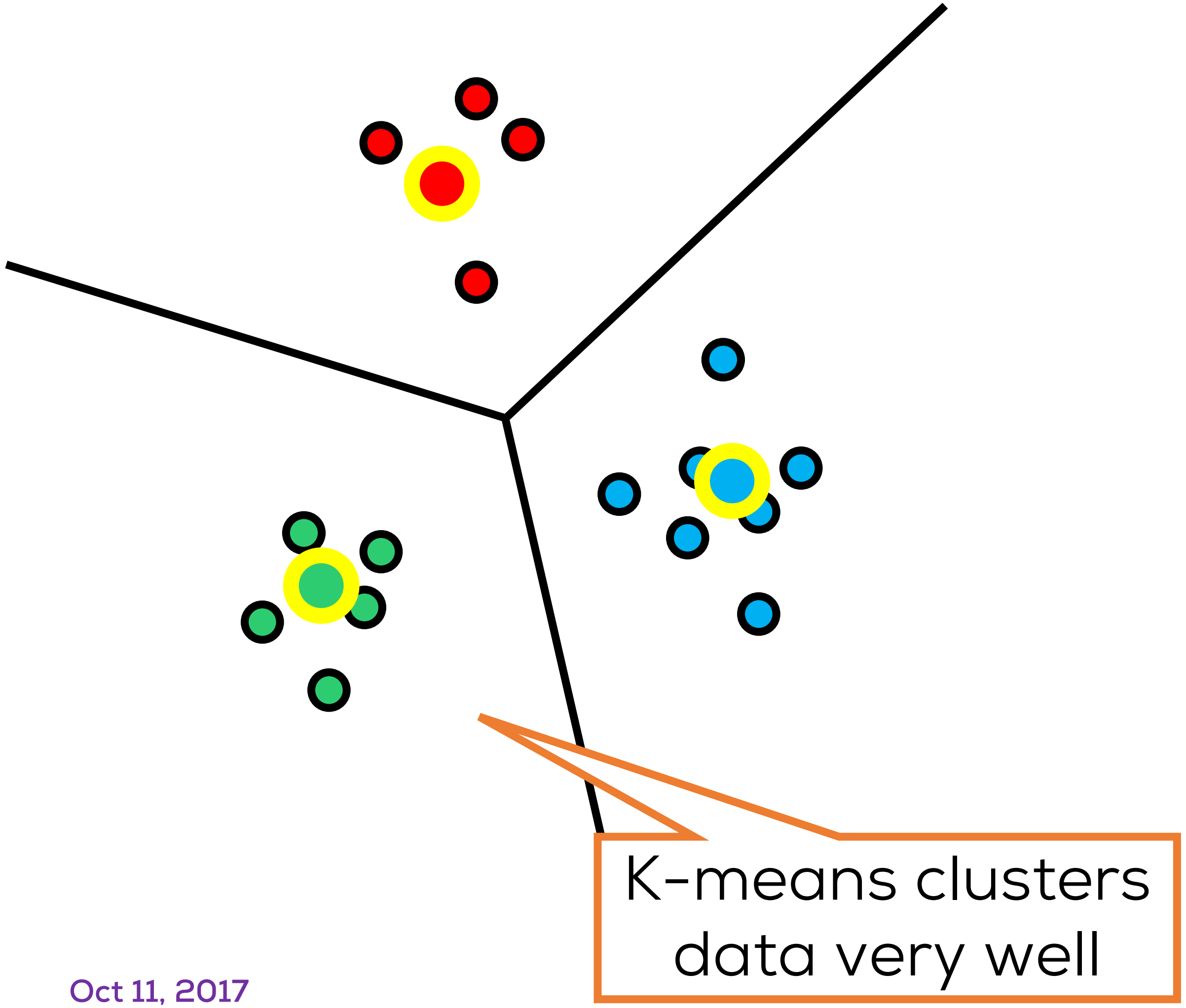
Why/when are kernels used?

- Whenever linear models poorly fit your data



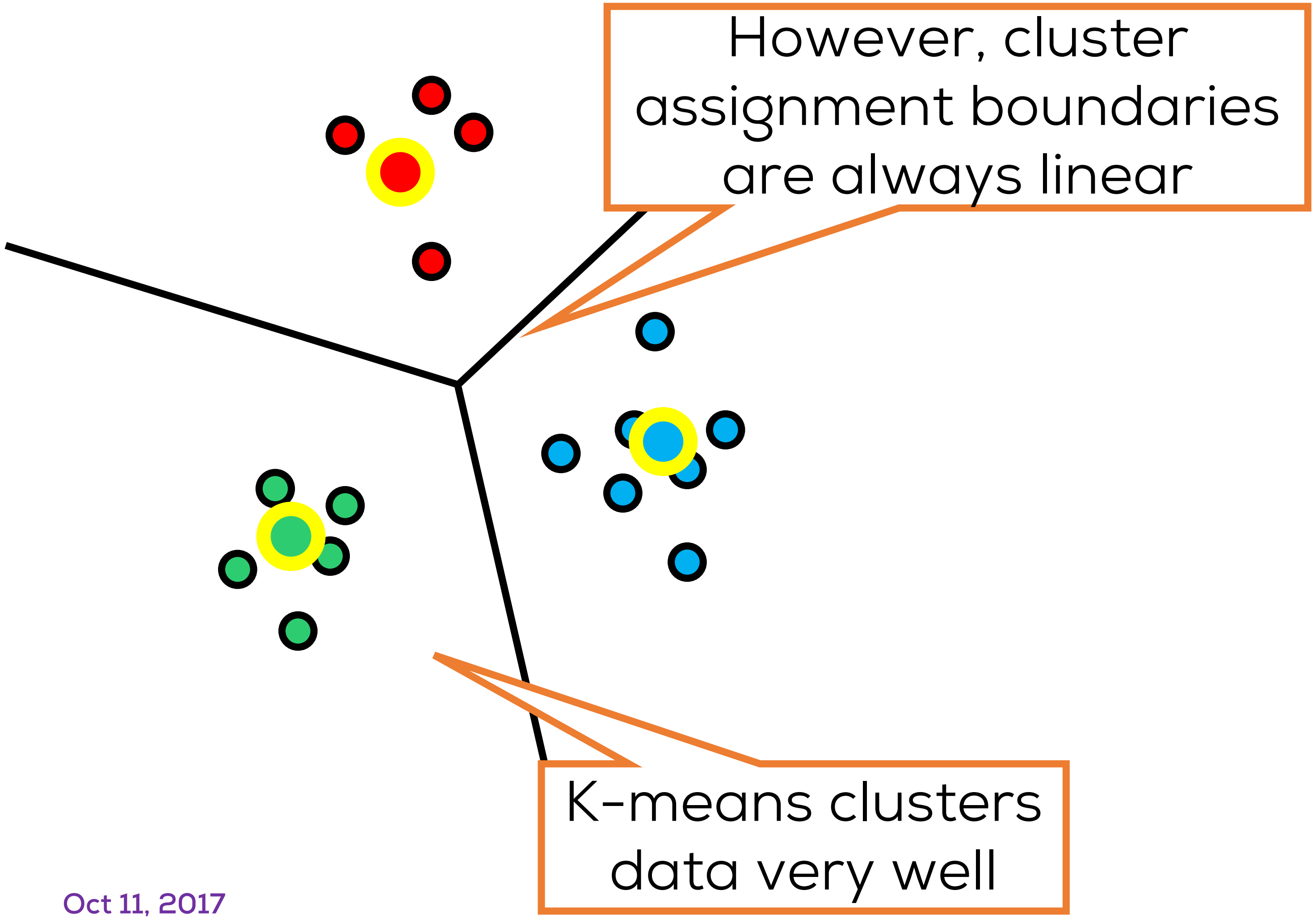
Why/when are kernels used?

- Whenever linear models poorly fit your data



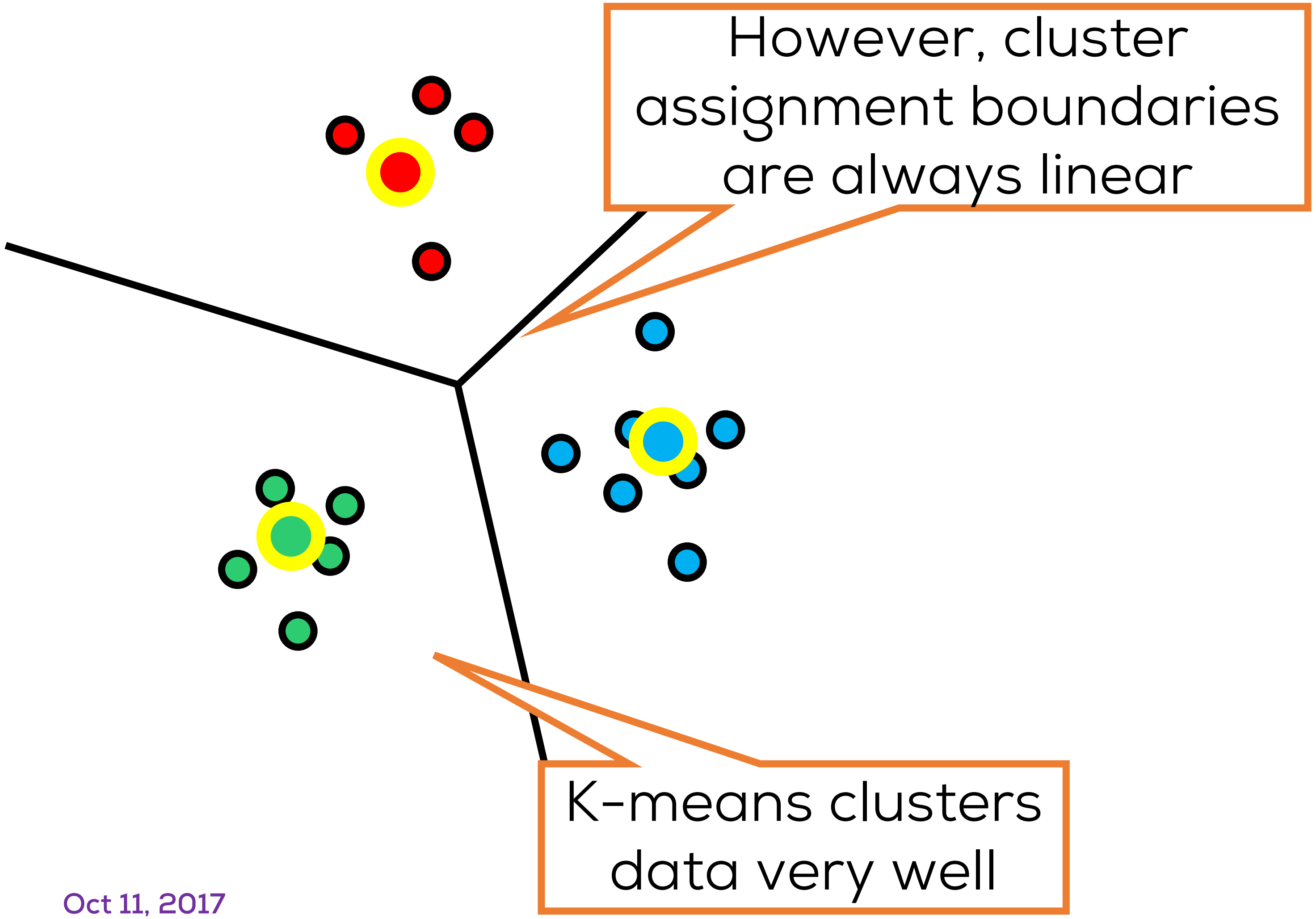
Why/when are kernels used?

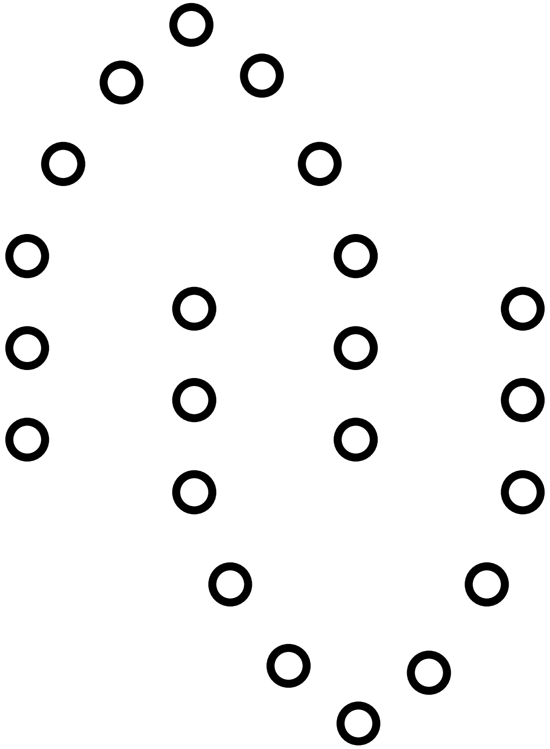
- Whenever linear models poorly fit your data



Why/when are kernels used?

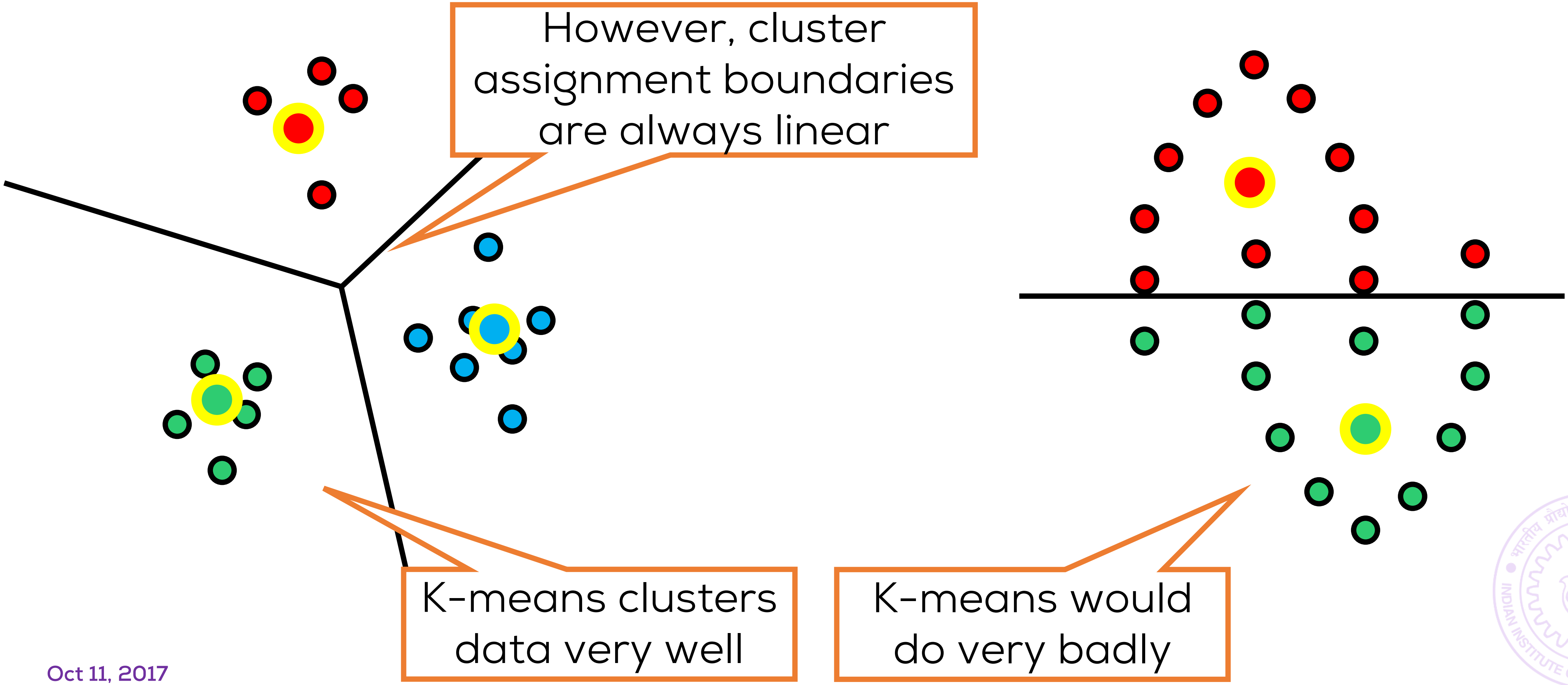
- Whenever linear models poorly fit your data





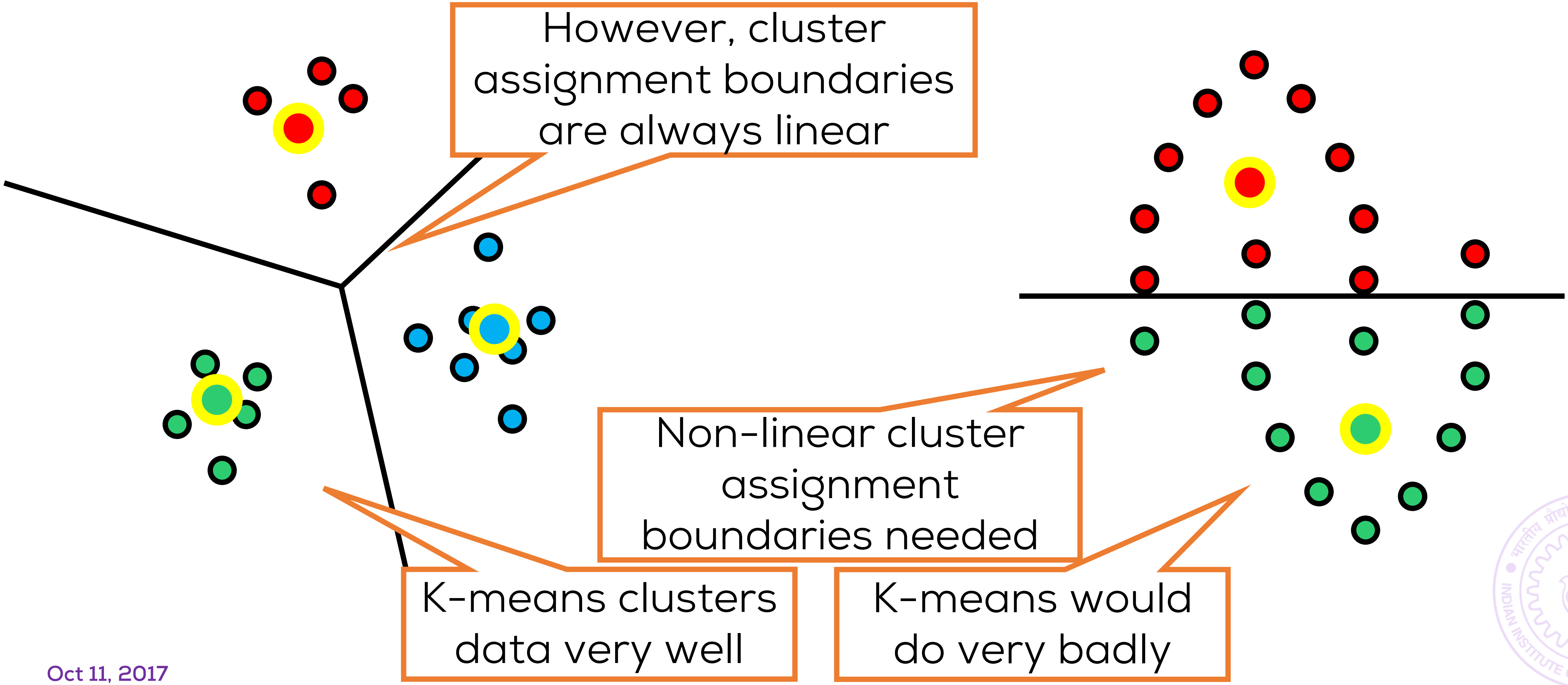
Why/when are kernels used?

- Whenever linear models poorly fit your data



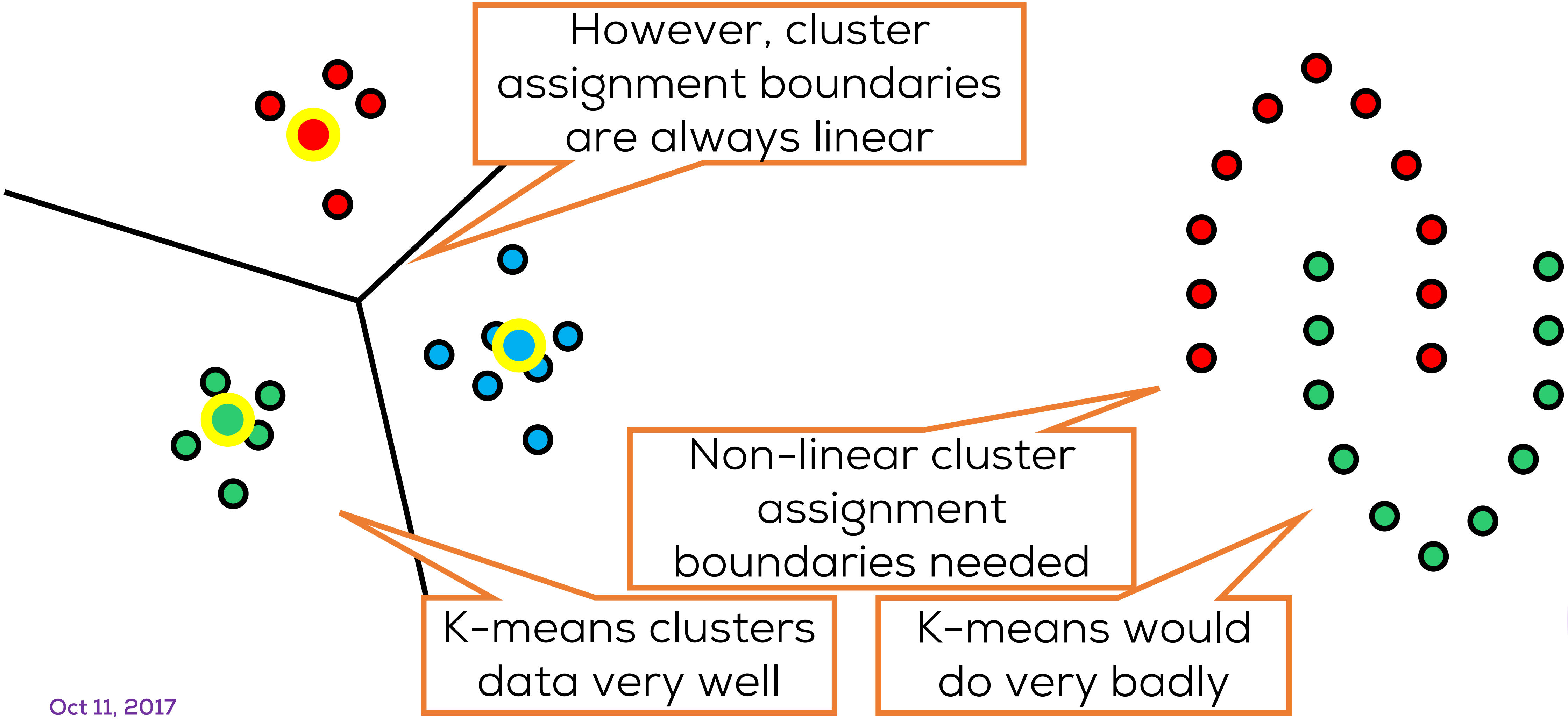
Why/when are kernels used?

- Whenever linear models poorly fit your data



Why/when are kernels used?

- Whenever linear models poorly fit your data

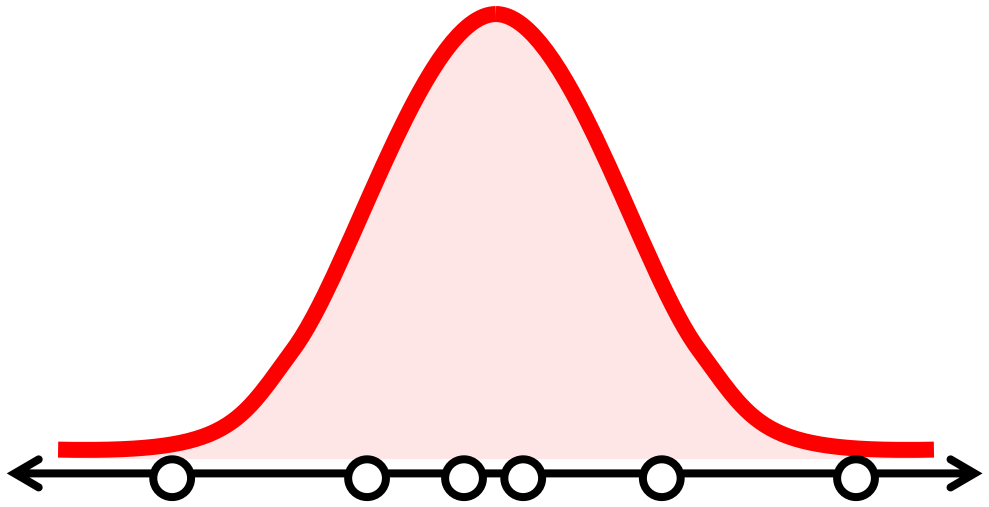


Why/when are kernels used?

- Whenever linear models poorly fit your data

Why/when are kernels used?

- Whenever linear models poorly fit your data

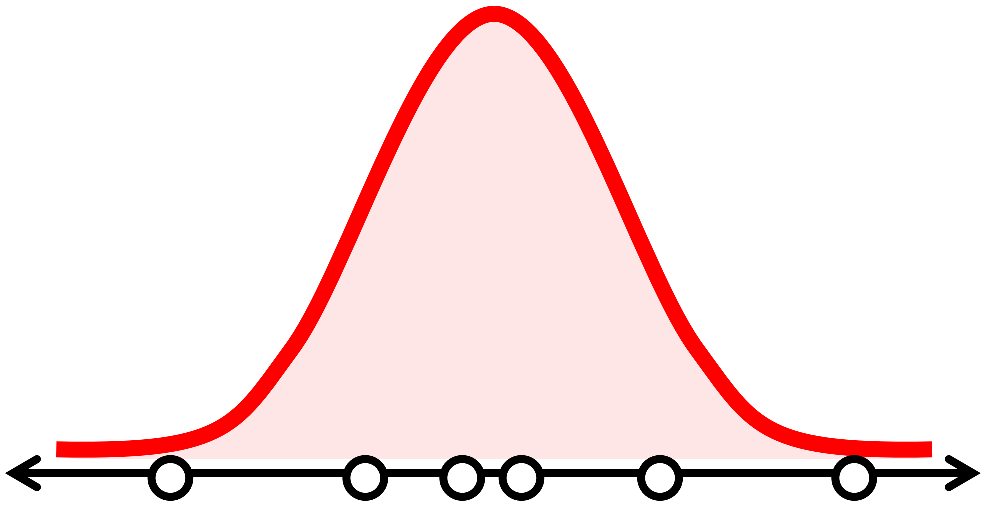


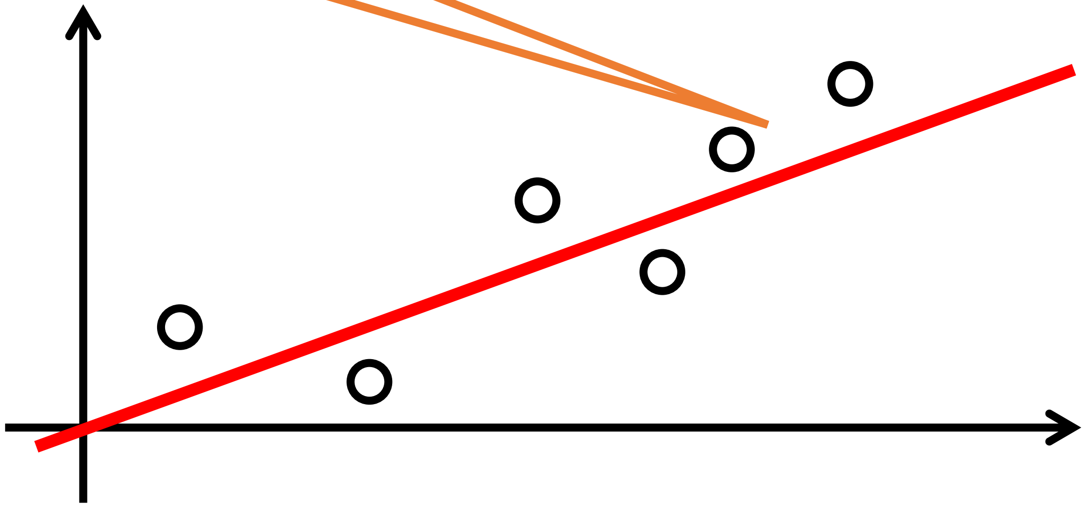
Why/when are kernel

- Whenever linear models poorly fit your da

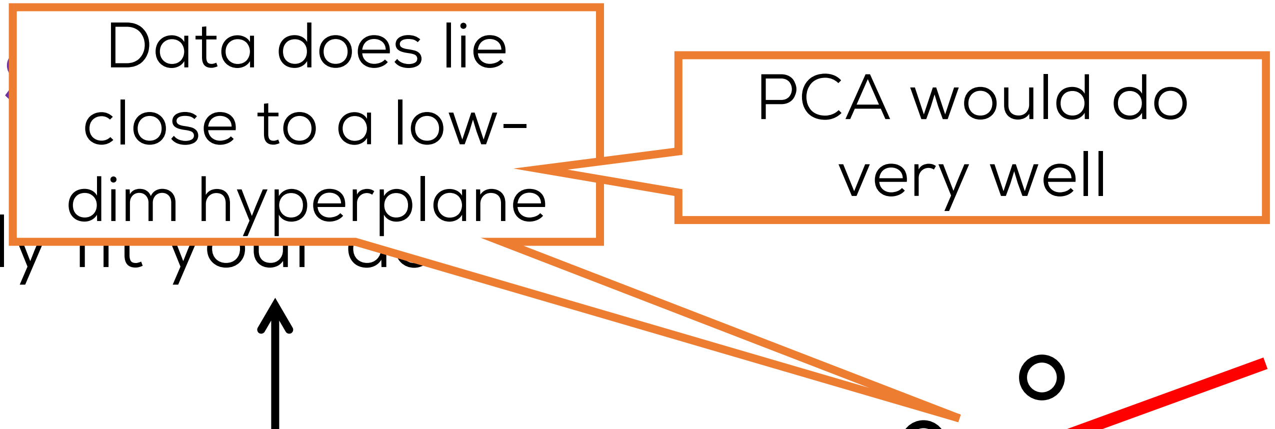
Data does lie close to a low-dim hyperplane

PCA would do very well

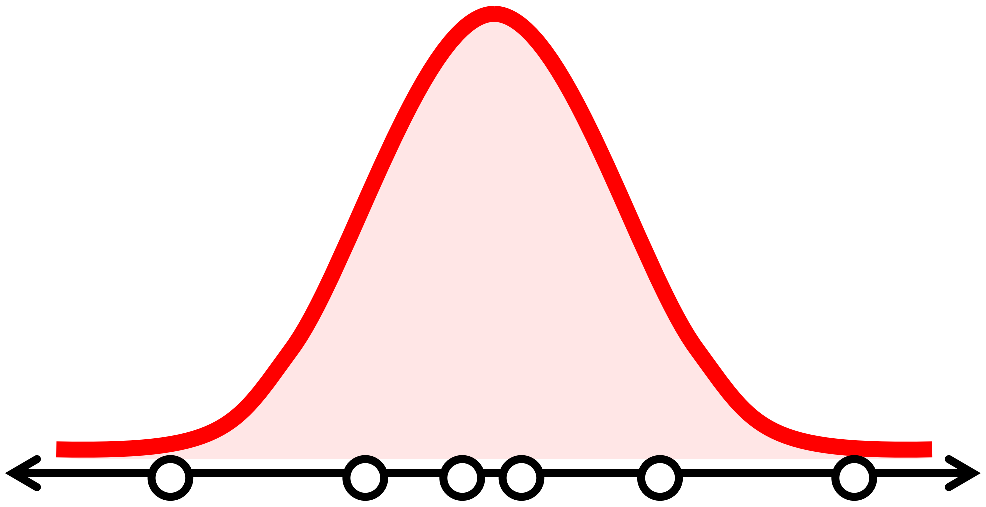


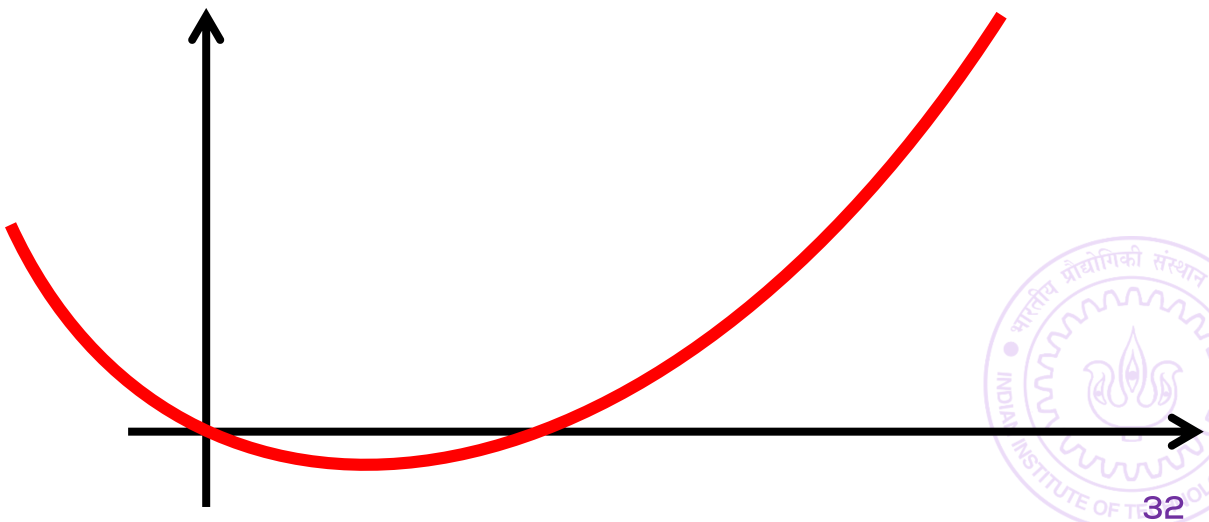


Why/when are kernel



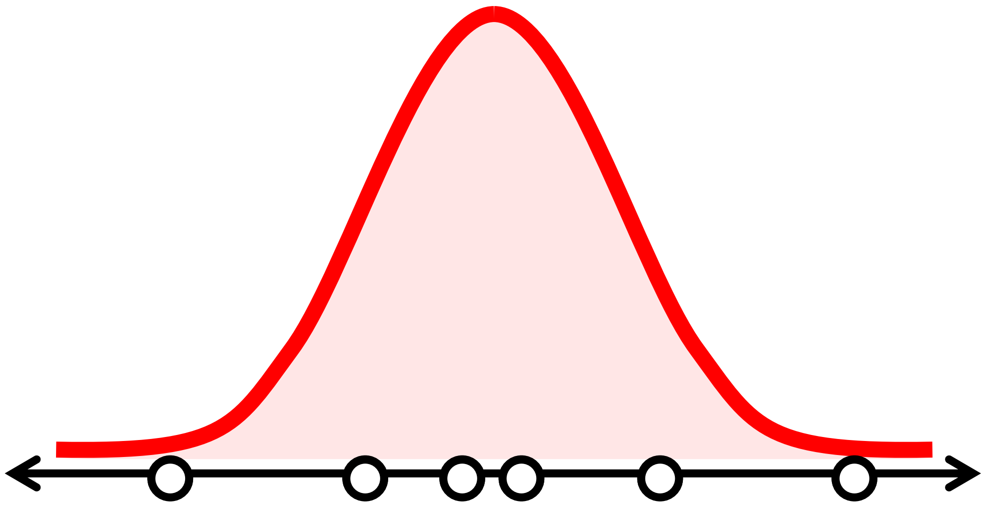
- Whenever linear models poorly fit your da

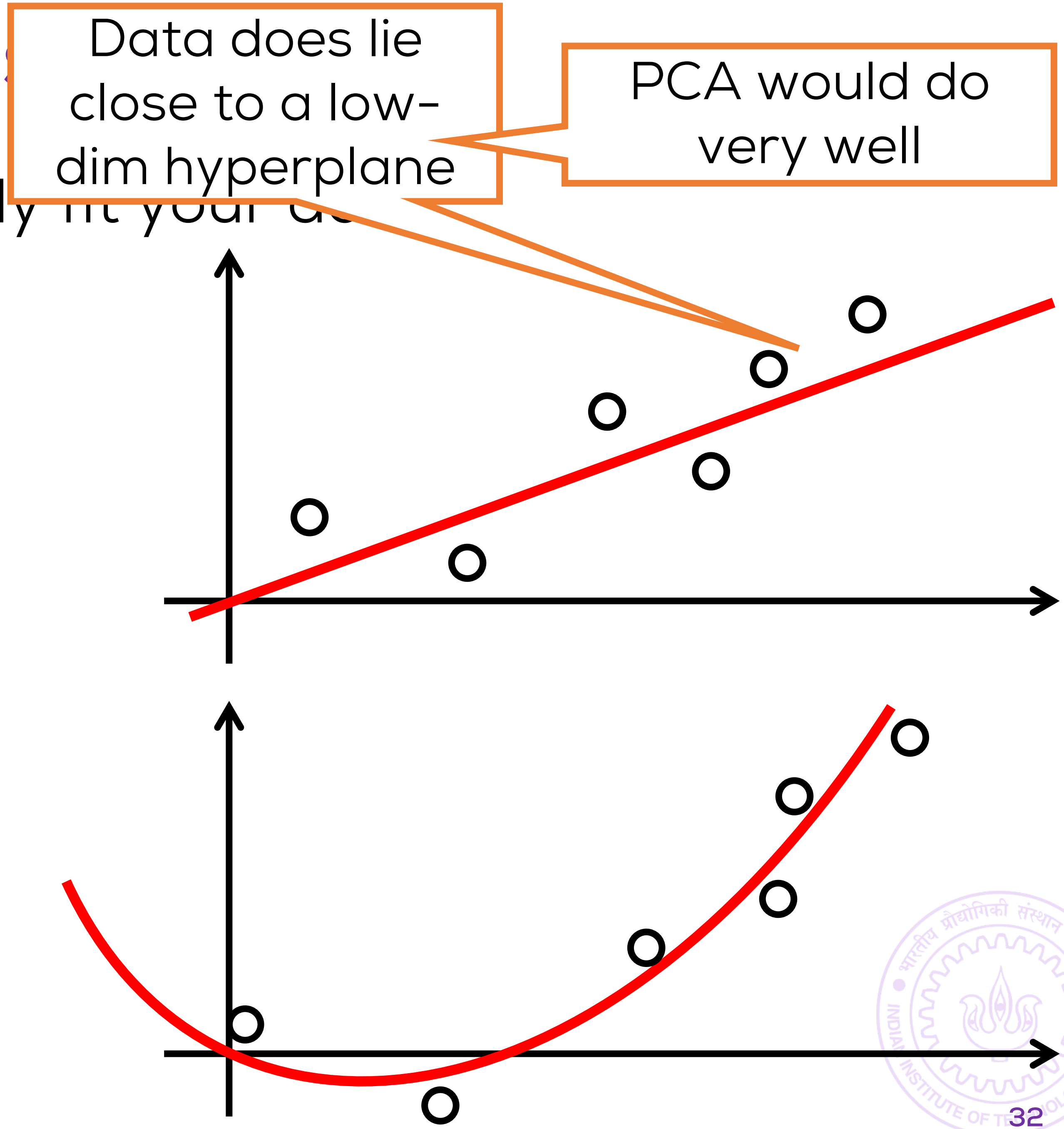




Why/when are kernels

- Whenever linear models poorly fit your da



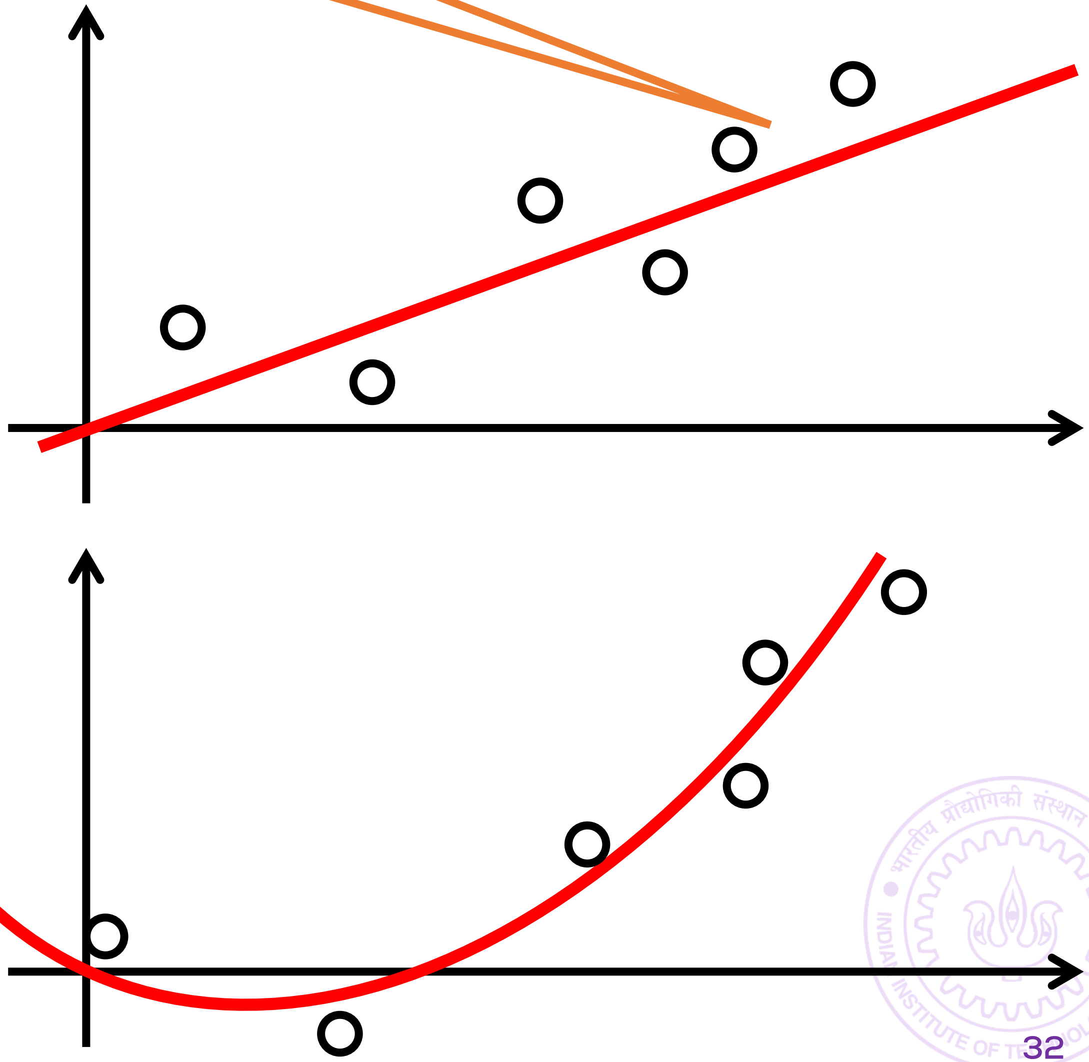


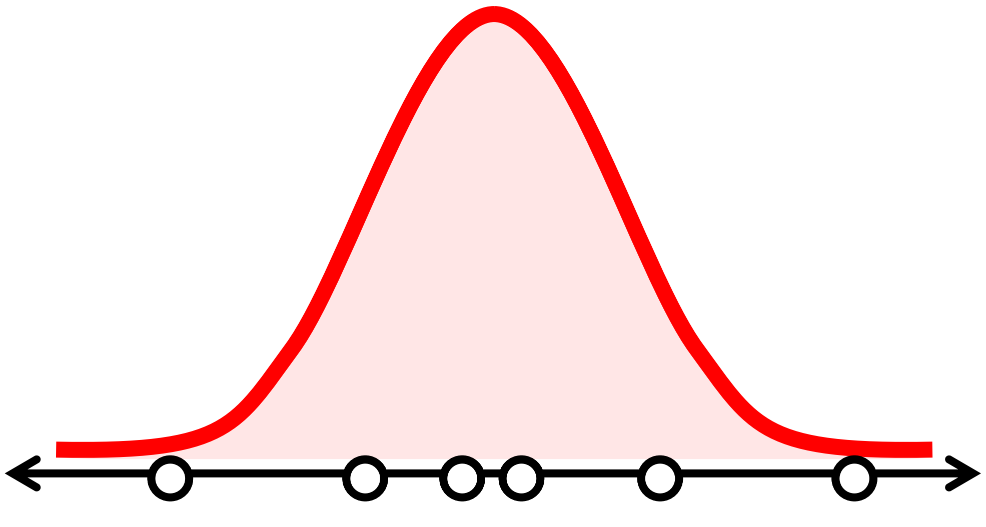
Why/when are kernel

- Whenever linear models poorly fit your da

Data does lie close to a low-dim hyperplane

PCA would do very well





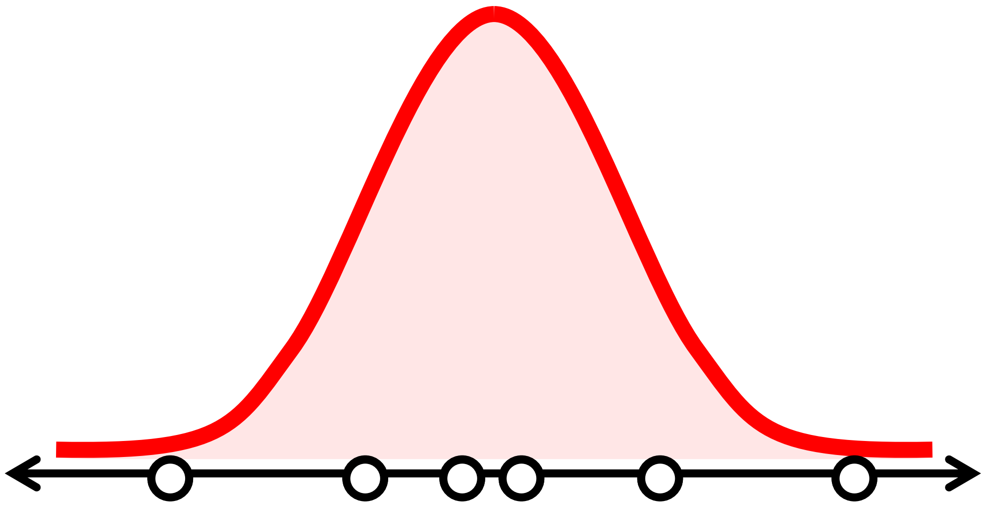
No low-dim hyperplane approximates data

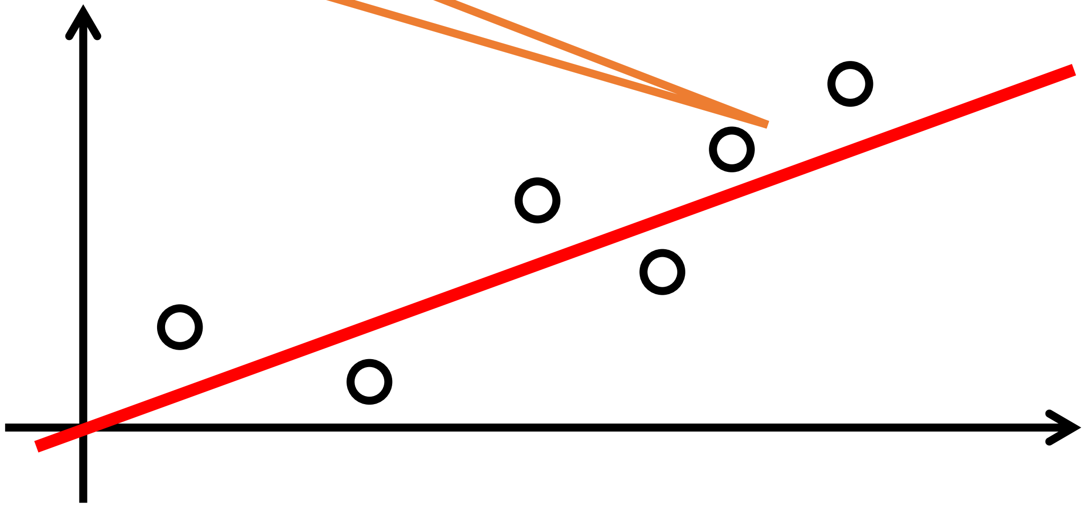
Why/when are kernel

- Whenever linear models poorly fit your da

Data does lie close to a low-dim hyperplane

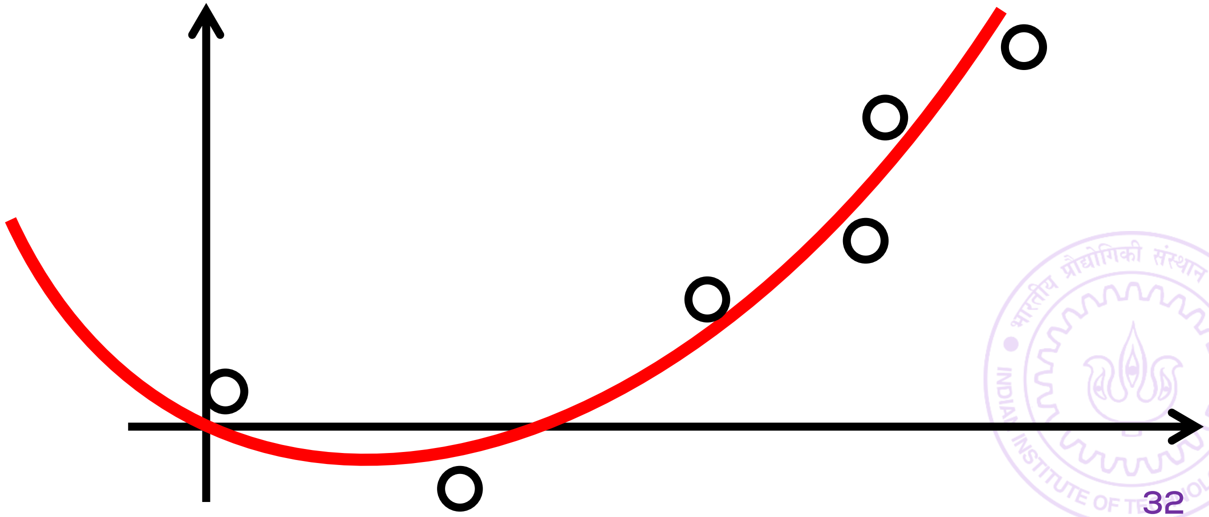
PCA would do very well





No low-dim hyperplane approximates data

Data actually close to a smooth low-dim surface

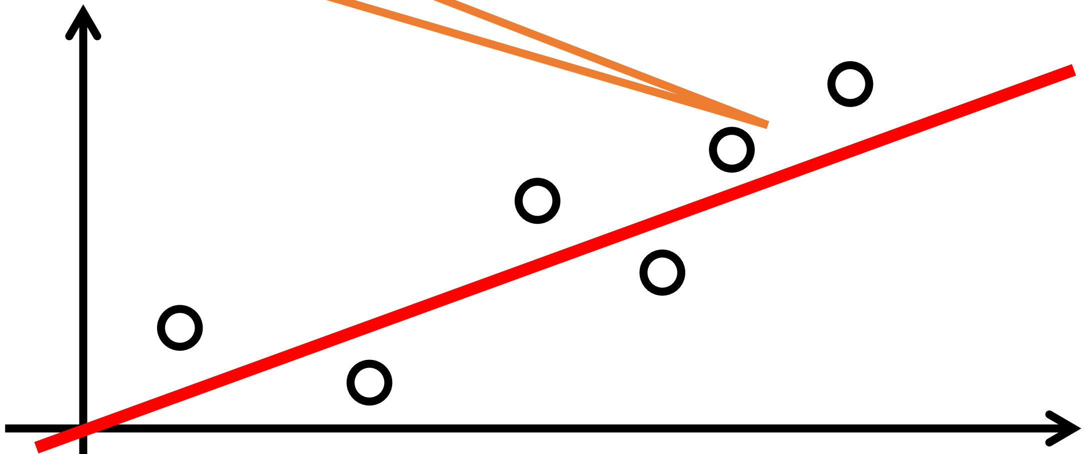


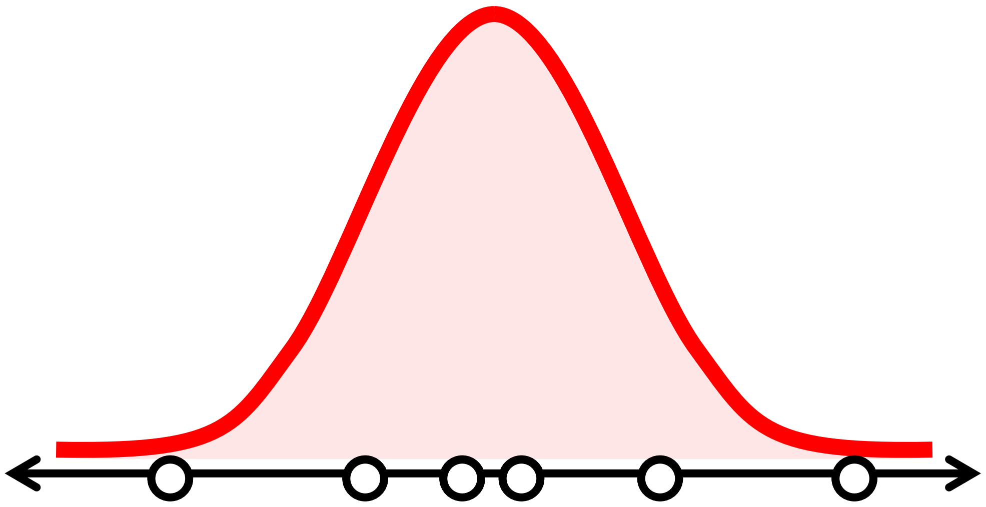
Why/when are kernel

- Whenever linear models poorly fit your da

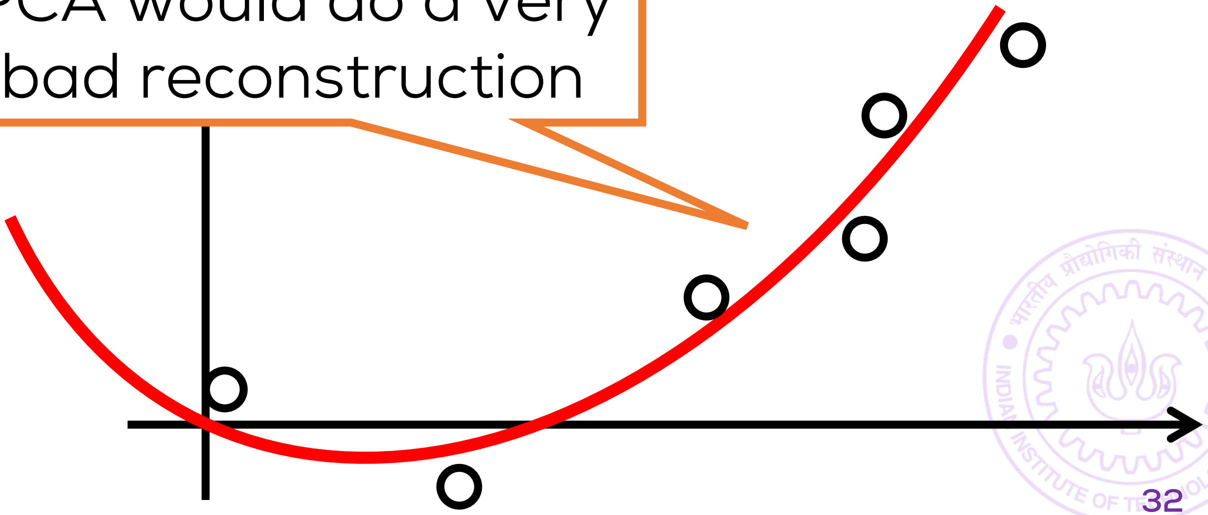
Data does lie close to a low-dim hyperplane

PCA would do very well





PCA would do a very bad reconstruction



No low-dim hyperplane approximates data

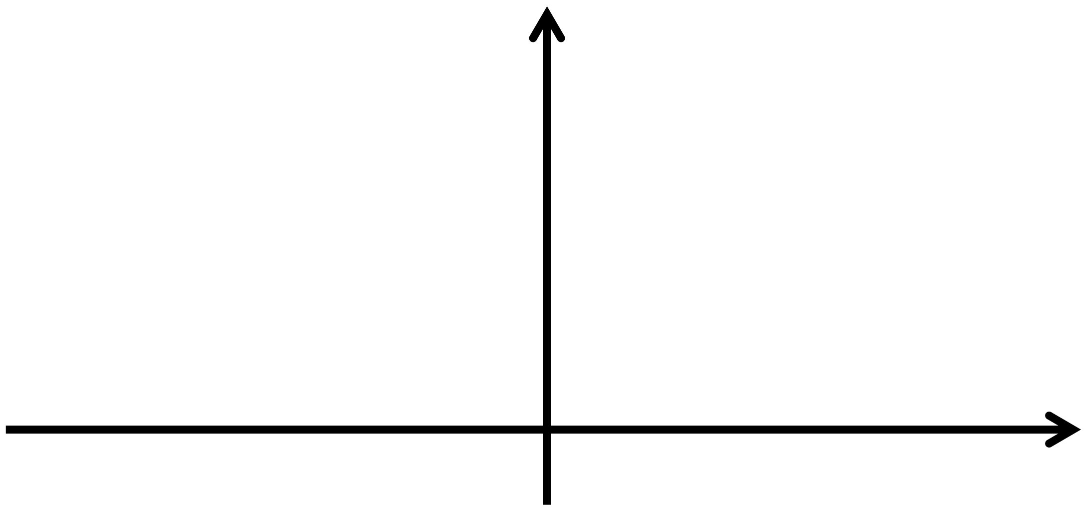
Data actually close to a smooth low-dim surface

How to learn non-linear functions

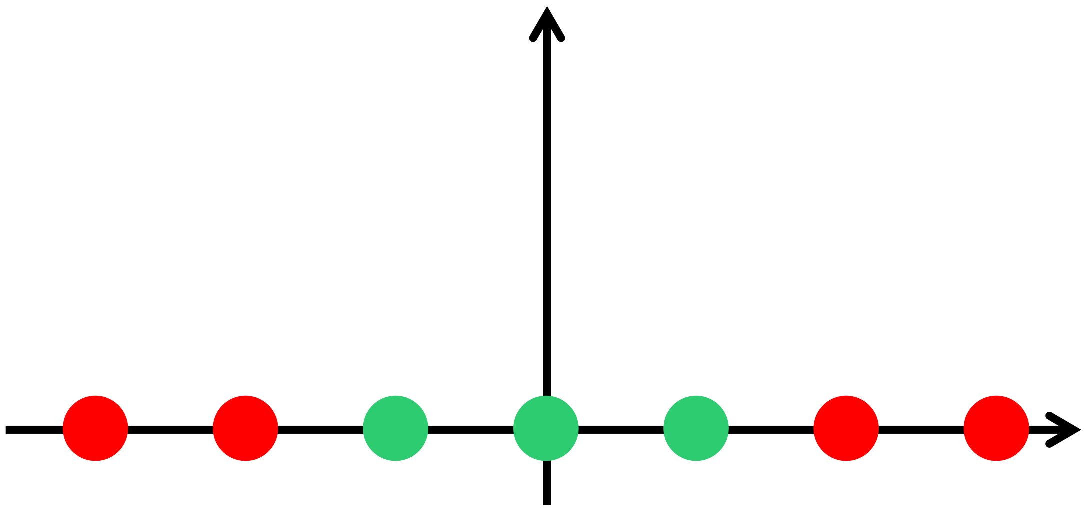
- The reductionist mind: I already know how to learn
  - Linear classifiers: perceptron, SVM, logistic regression
  - Linear regressors: least squares, ridge regression
  - Linear dimensionality reducers: PCA, PPCA
- Why waste all that effort?
- Can I have an easy way so that my algorithms for learning linear models continue to work but end up learning non-linear funcs?
- Yes, through magical objects called kernels!
- Before going into the math (yes there is a bit of it I'm afraid ☹) let us see what is really going on

A toy example: non-linear classification

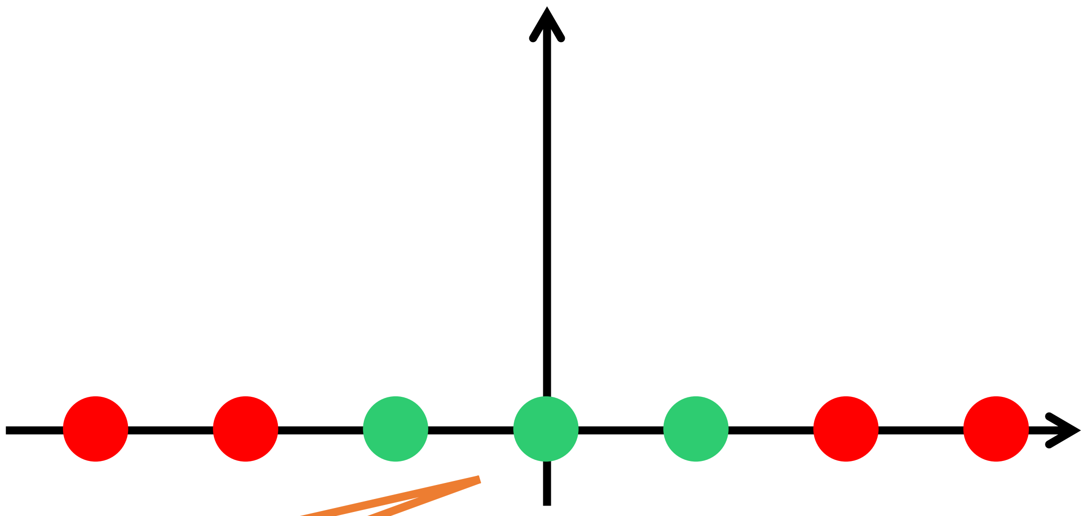
A toy example: non-linear classification



A toy example: non-linear classification

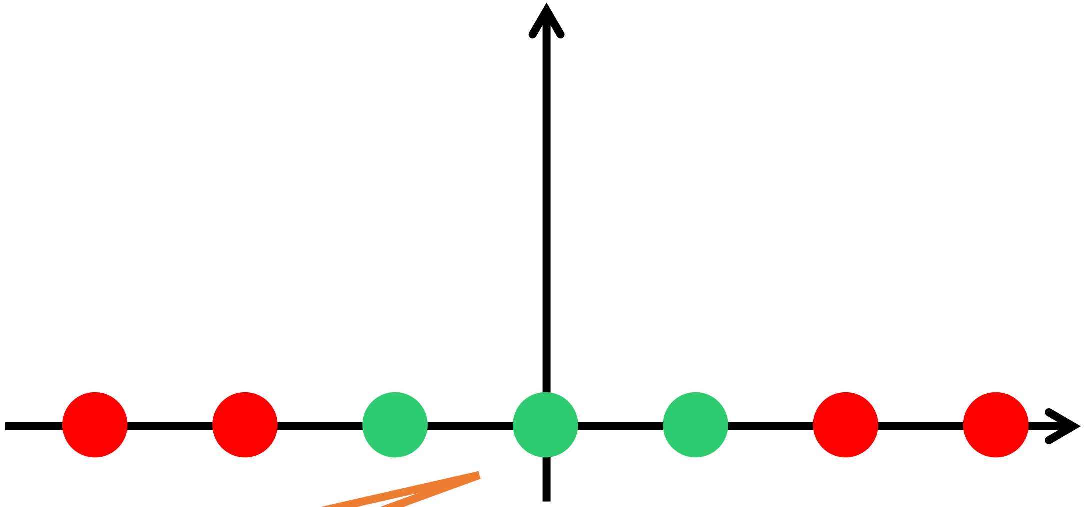


A toy example: non-linear classification



Points on the real line

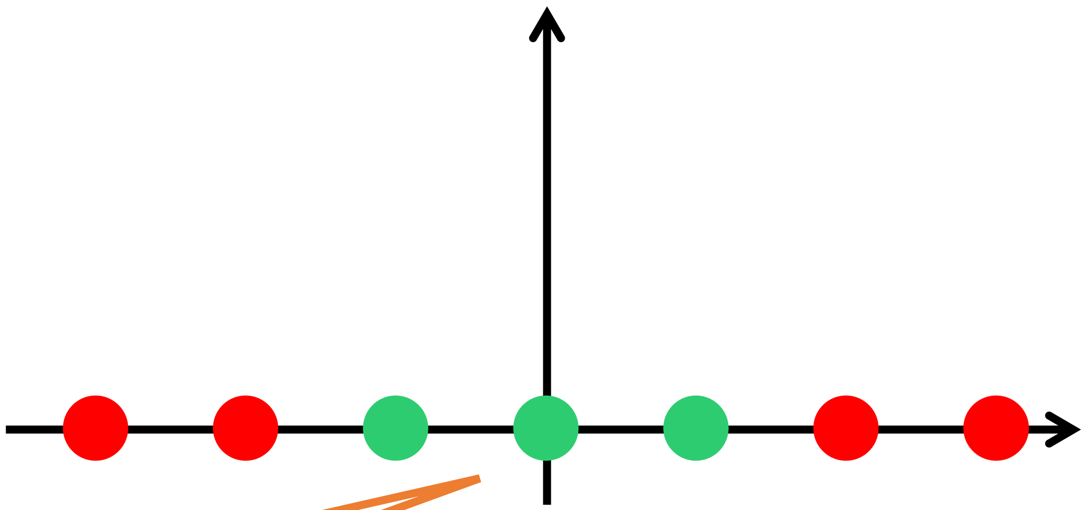
A toy example: non-linear classification



Points on the real line

No linear classfier can separate them ☹

A toy example: non-linear classification

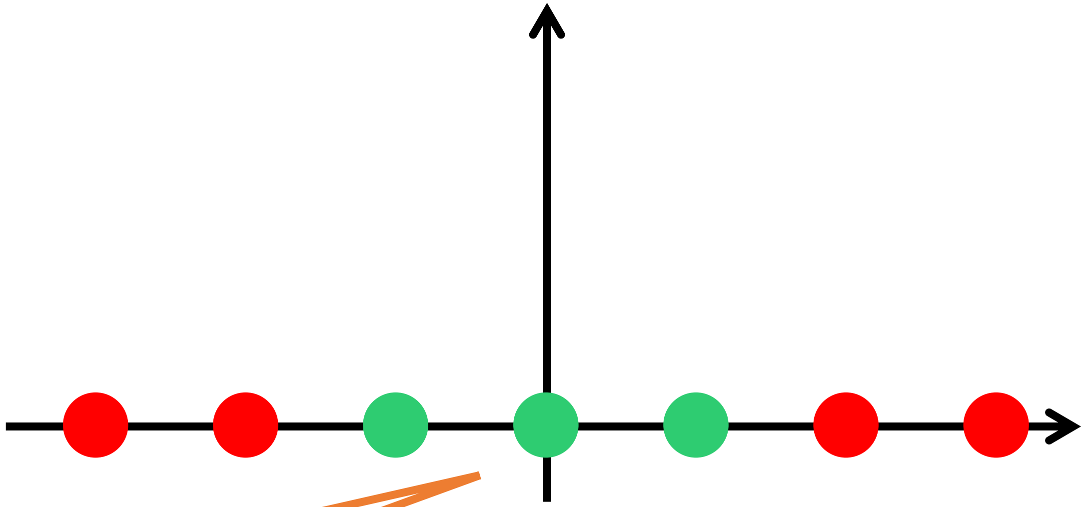


Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

No linear classfier can separate them ☹

A toy example: non-linear classification



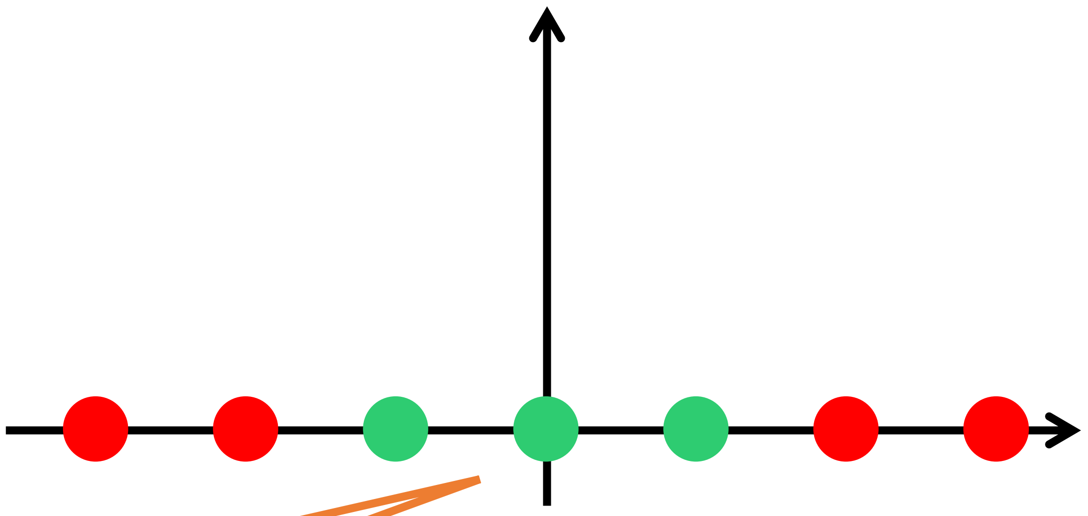
Points on the real line

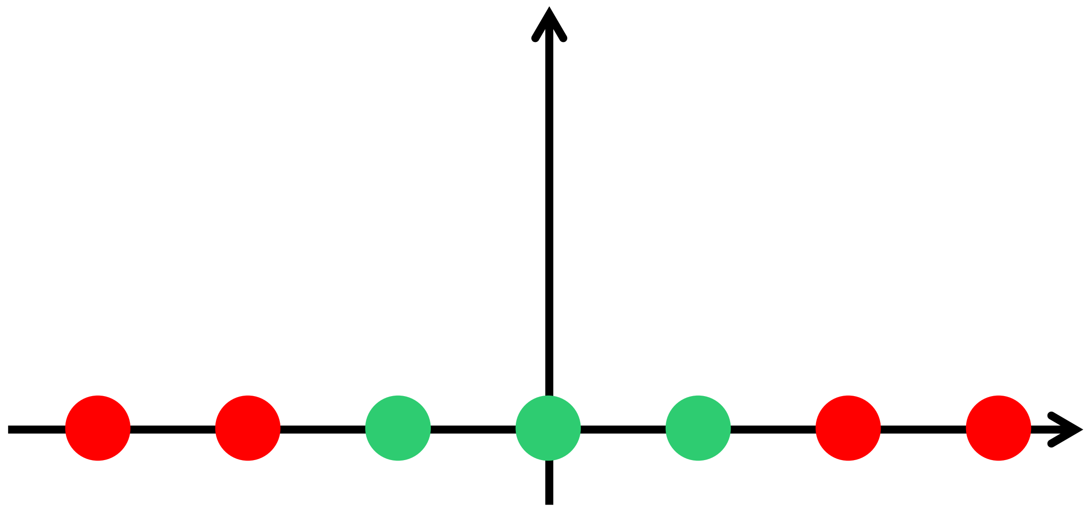
$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

No linear classfier can separate them ☹

A toy example: non-linear classification





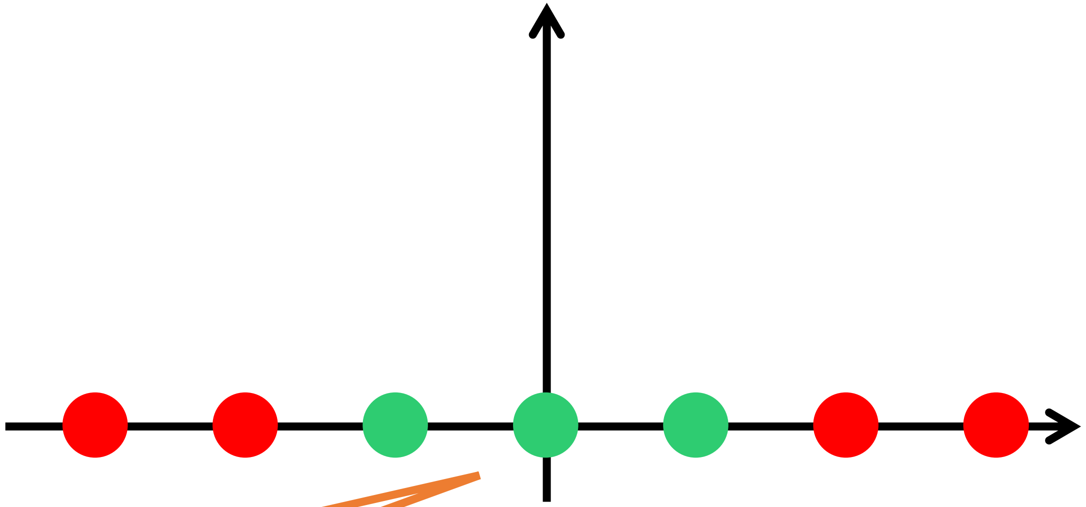
Points on the real line

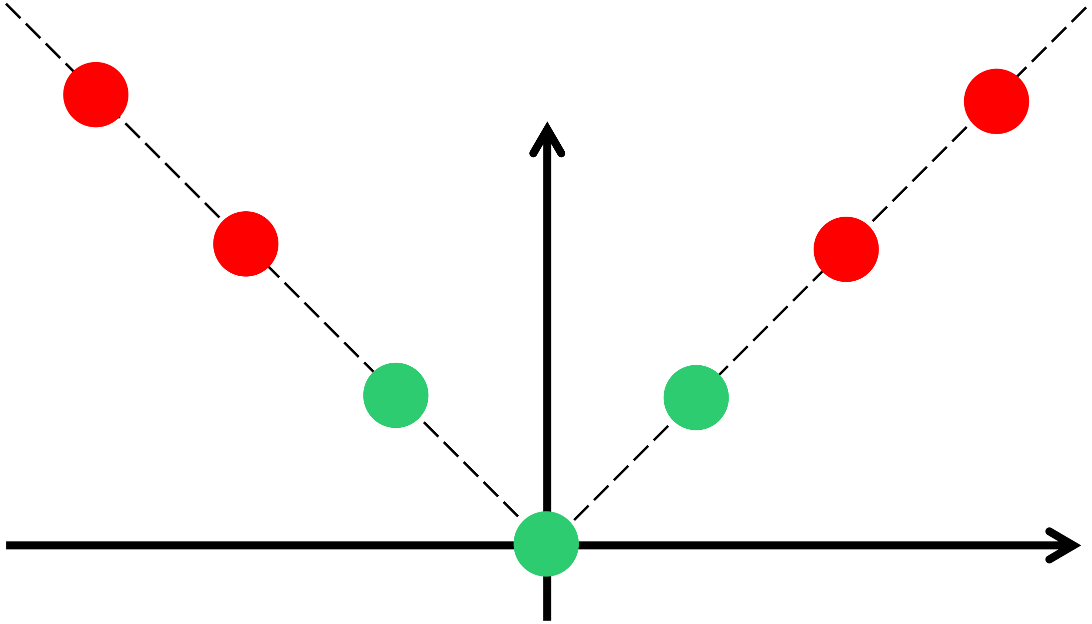
$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

No linear classfier can separate them ☹

A toy example: non-linear classification





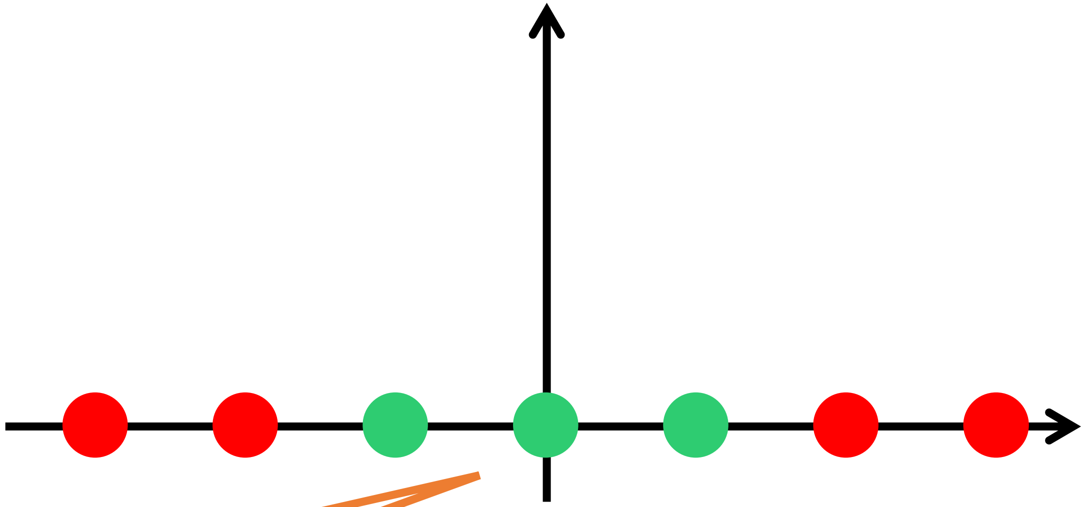
Points on the real line

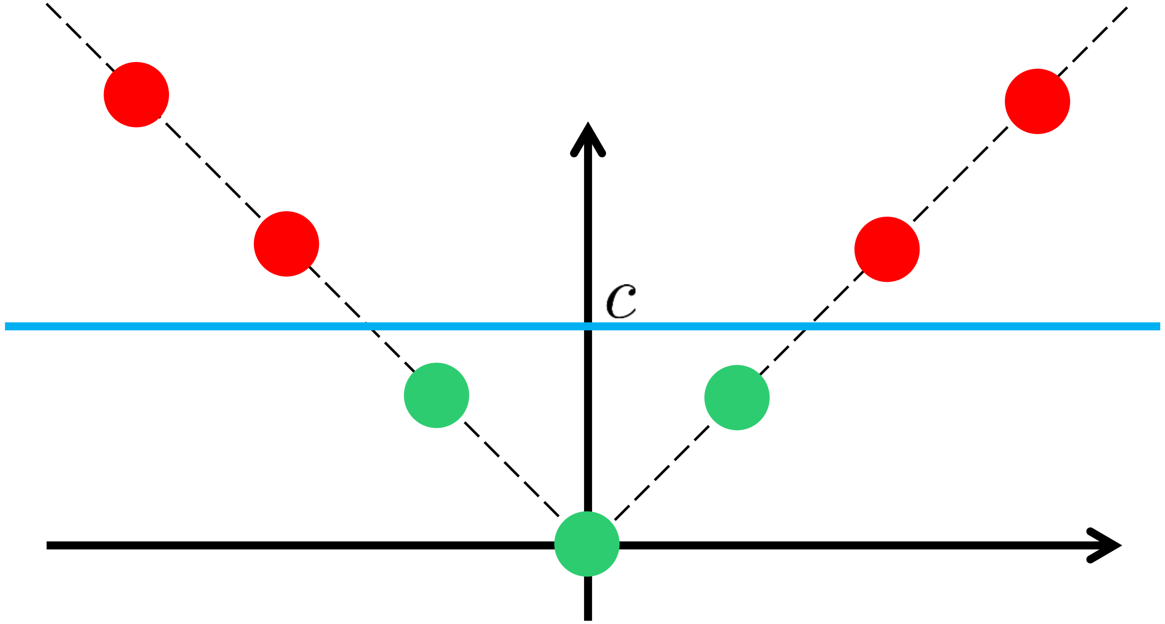
$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

No linear classfier can separate them ☹

A toy example: non-linear classification





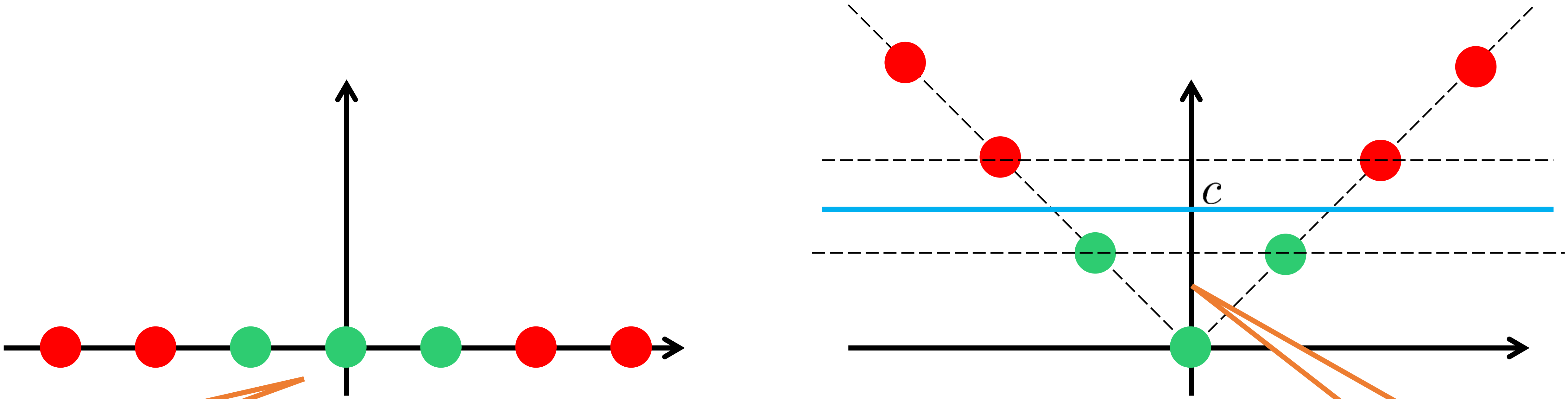
Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

No linear classfier can separate them ☹

A toy example: non-linear classification



Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

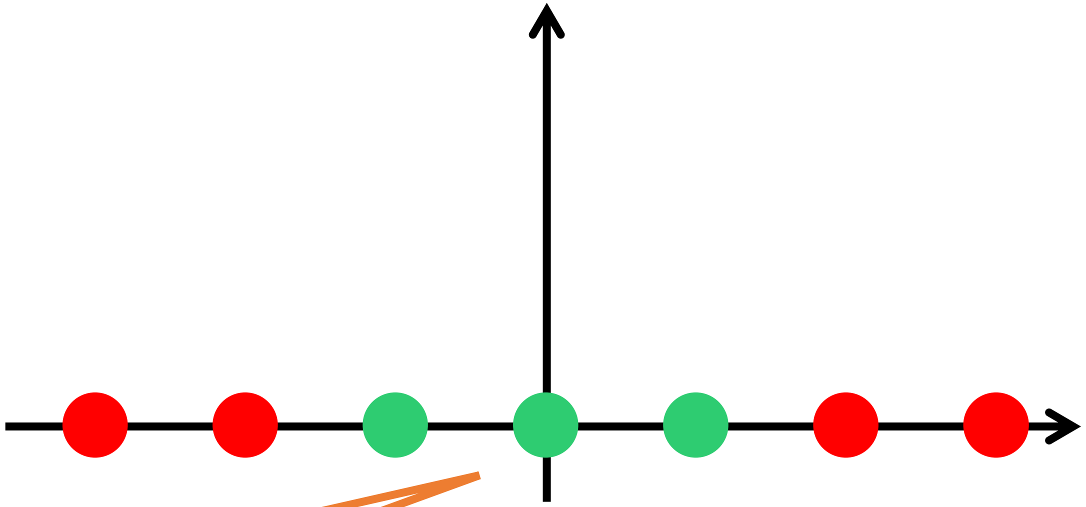
$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

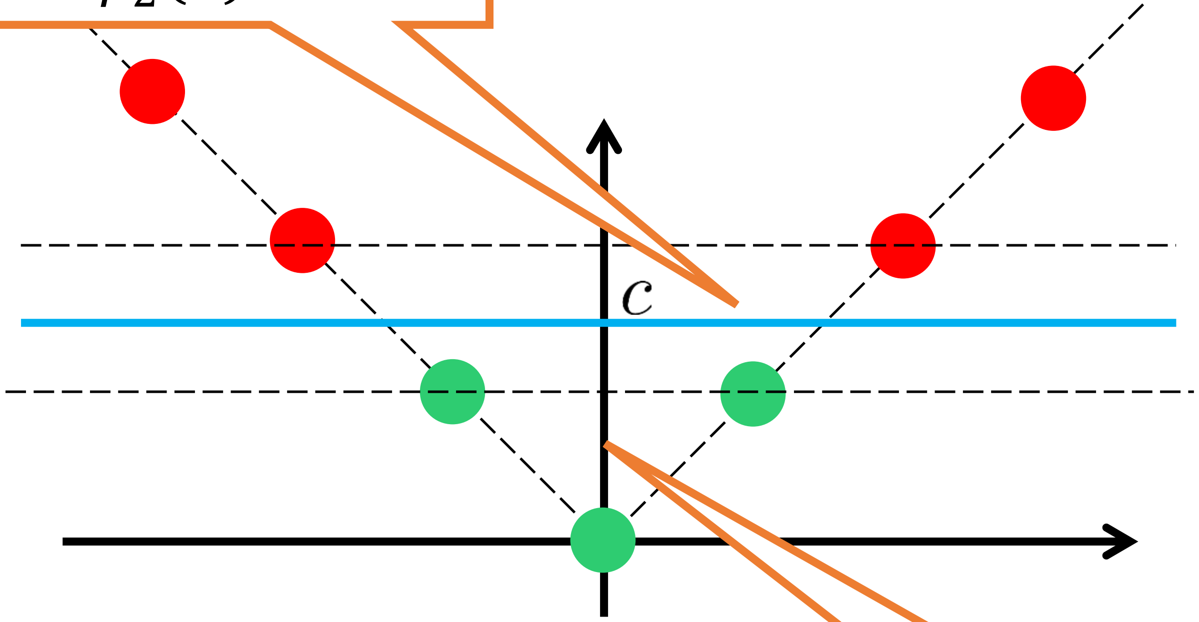
Not just classifiable but with a margin ☺

No linear classfier can separate them ☹

A toy example: non-line cation

This classifier is $\phi_2(x) = c$





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

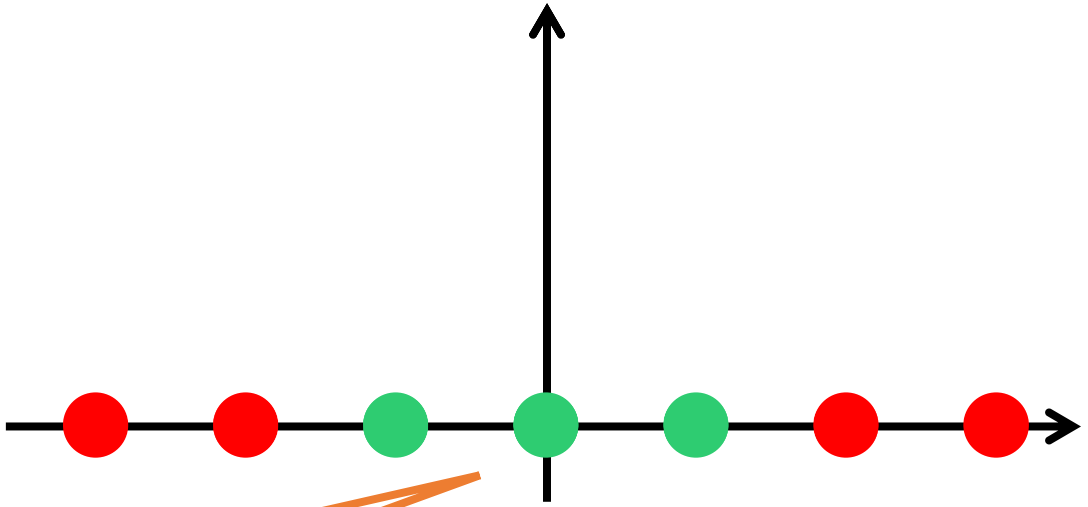
Not just classifiable but with a margin ☺

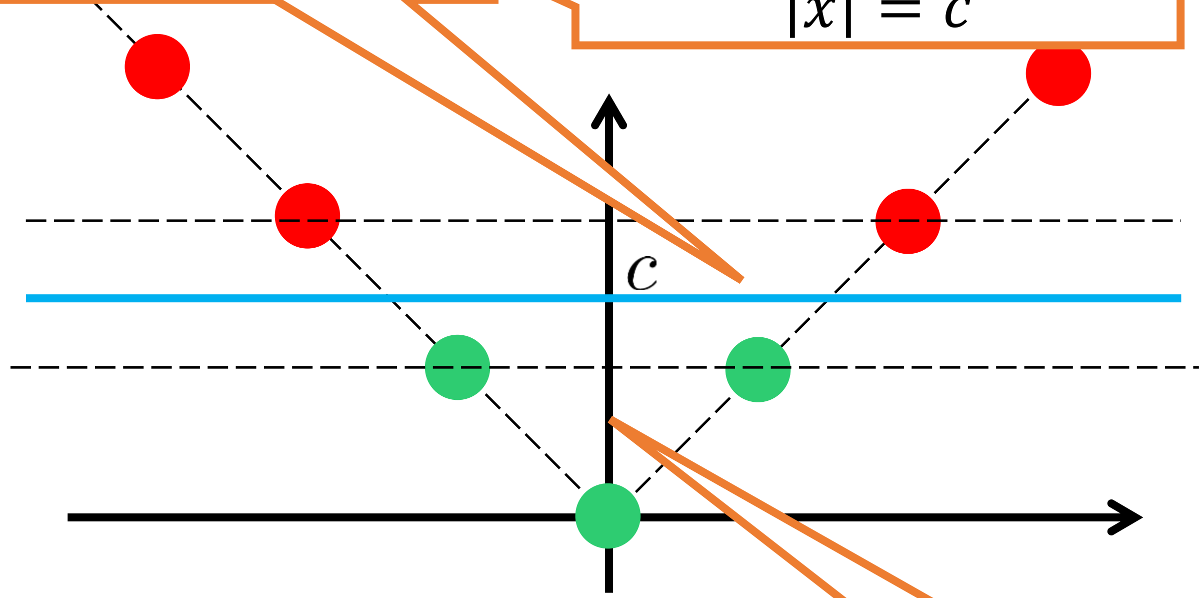
No linear classfier can separate them ☹

A toy example: non-line… ca…

This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

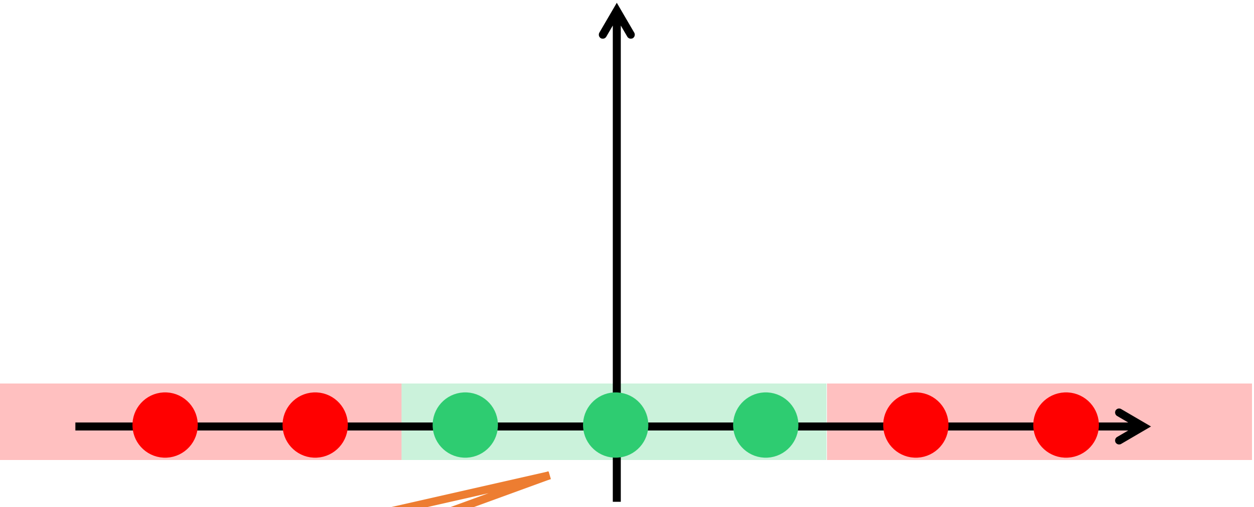
Not just classifiable but with a margin ☺

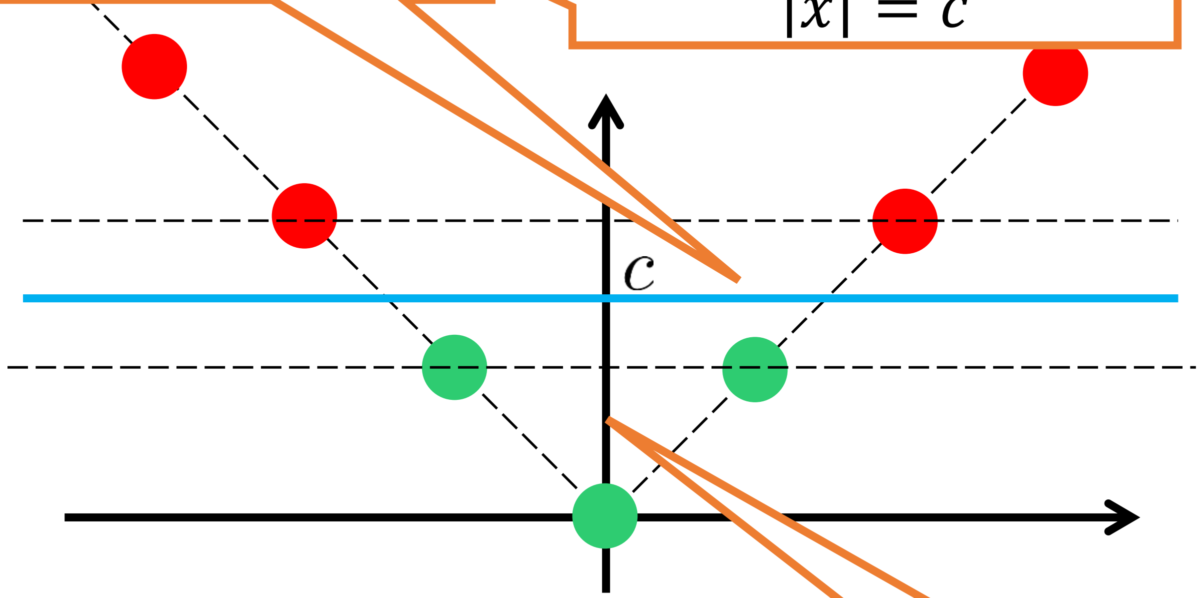
No linear classfier can separate them ☹

A toy example: non-line ... ca

This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

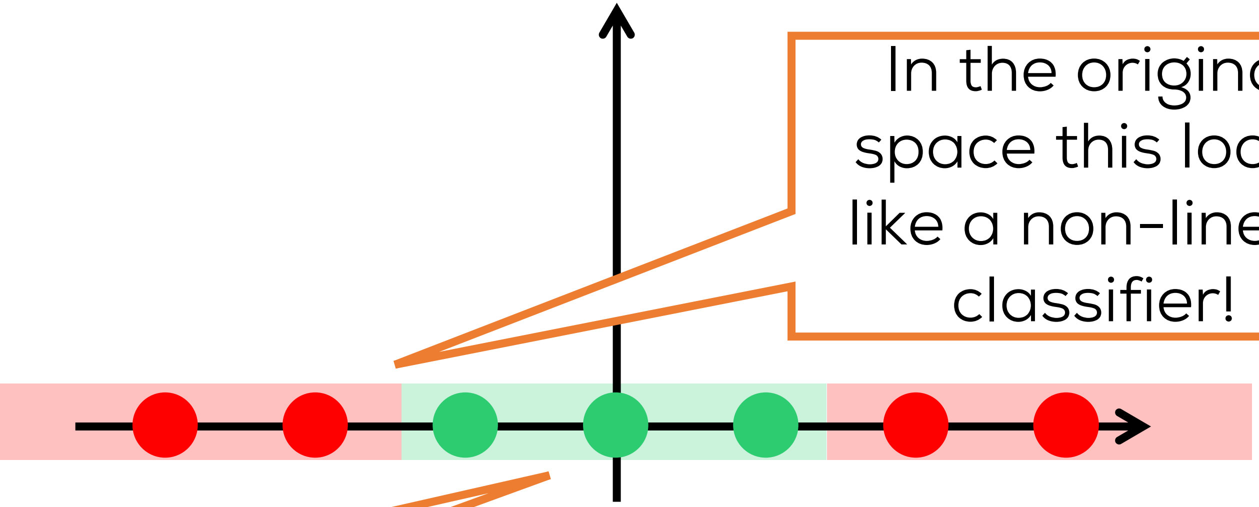
Not just classifiable but with a margin ☺

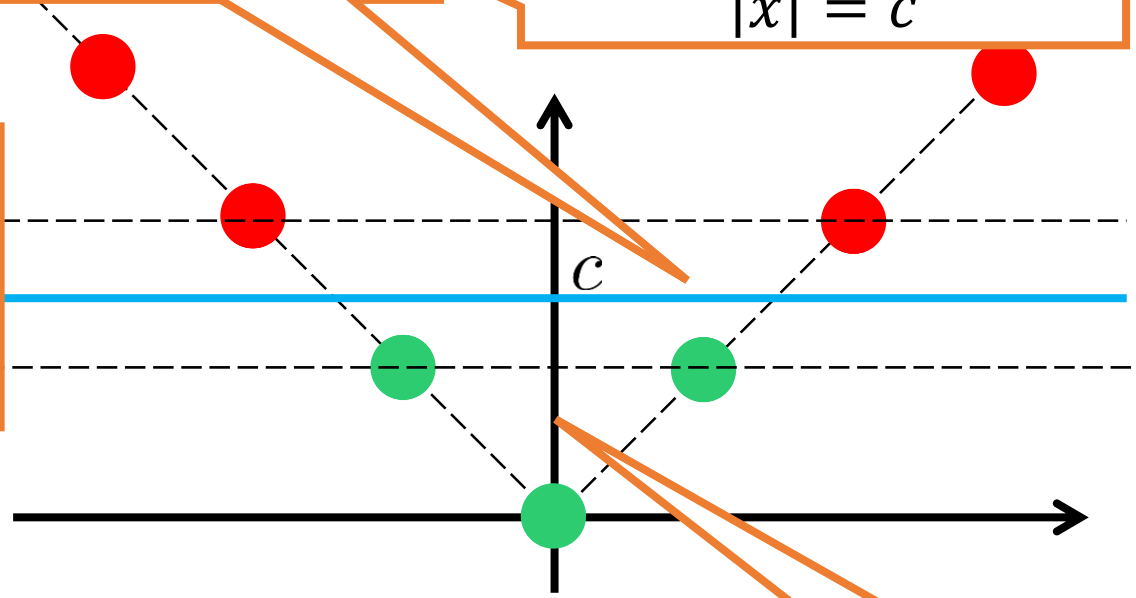
No linear classfier can separate them ☹

A toy example: non-line… ca…

This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$





In the original space this looks like a non-linear classifier!

Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

Not just classifiable but with a margin ☺

No linear classfier can separate them ☹

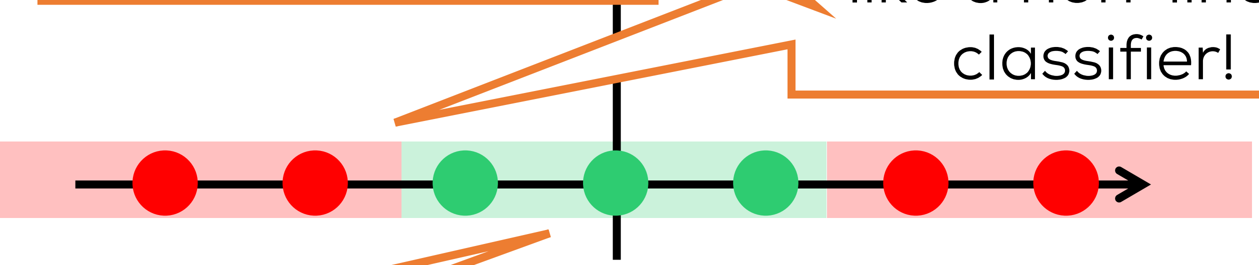
A toy example: non-line… ca…

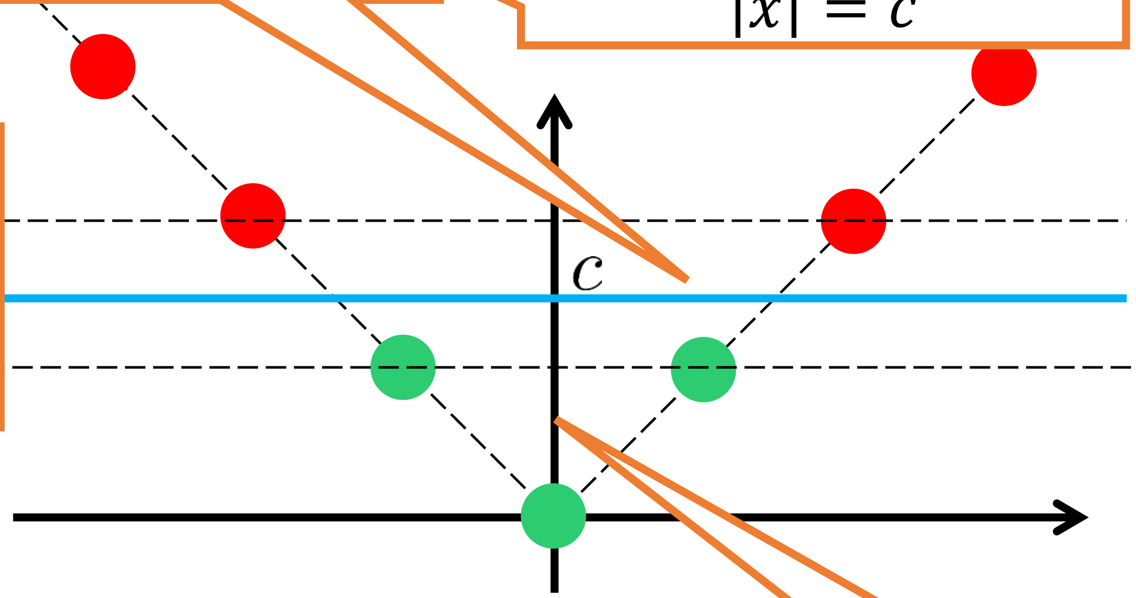
This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$

How did a linear classifier in $\mathbb{R}^2$ impose non-linear classification in $\mathbb{R}$

In the original space this looks like a non-linear classifier!





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

Not just classifiable but with a margin ☺

No linear classfier can separate them ☹

A toy example: non-line... ca...

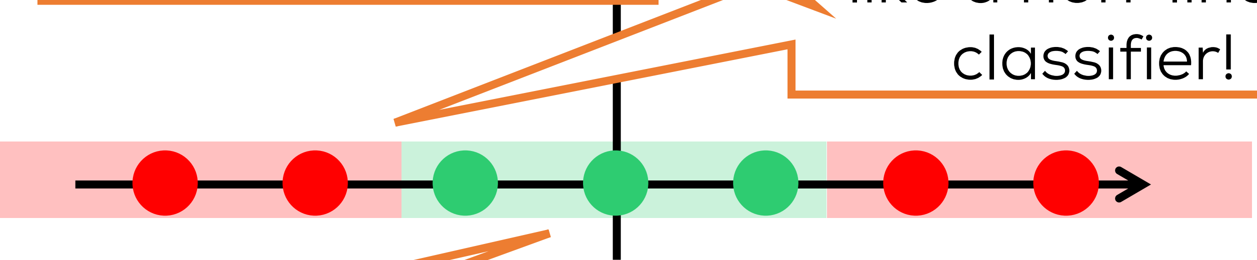
Where did non-linearity come from?

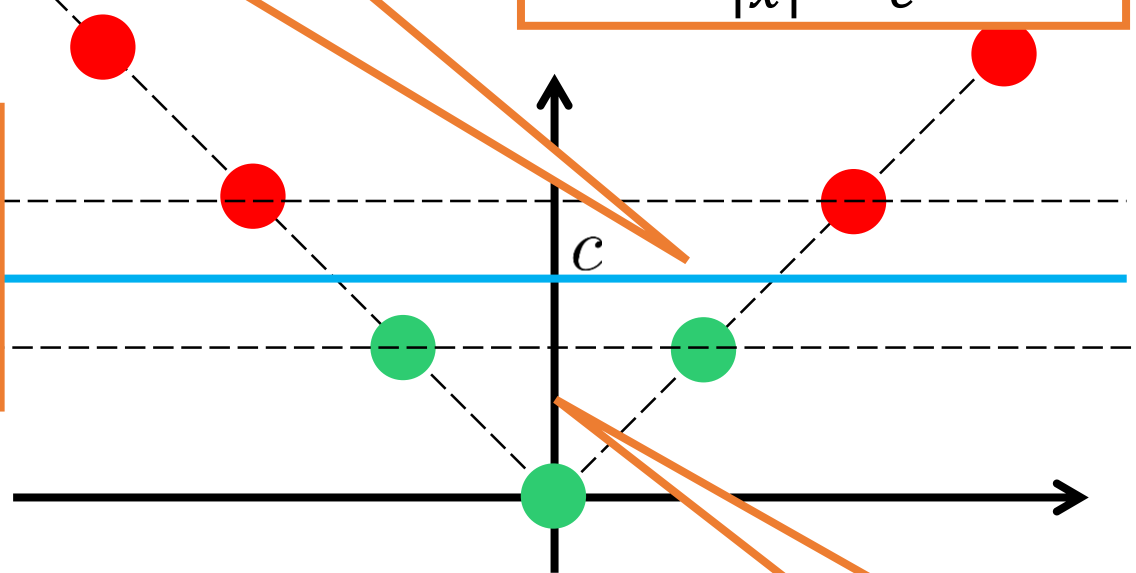
How did a linear classifier in $\mathbb{R}^2$ impose non-linear classification in $\mathbb{R}$

In the original space this looks like a non-linear classifier!

This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

Not just classifiable but with a margin ☺

No linear classfier can separate them ☹

A toy example: non-line… ca…

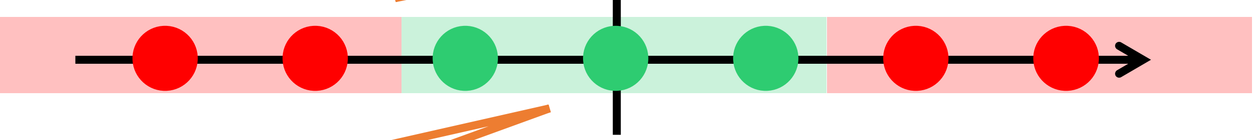
How did a linear classifier in $\mathbb{R}^2$ impose non-linear classification in $\mathbb{R}$

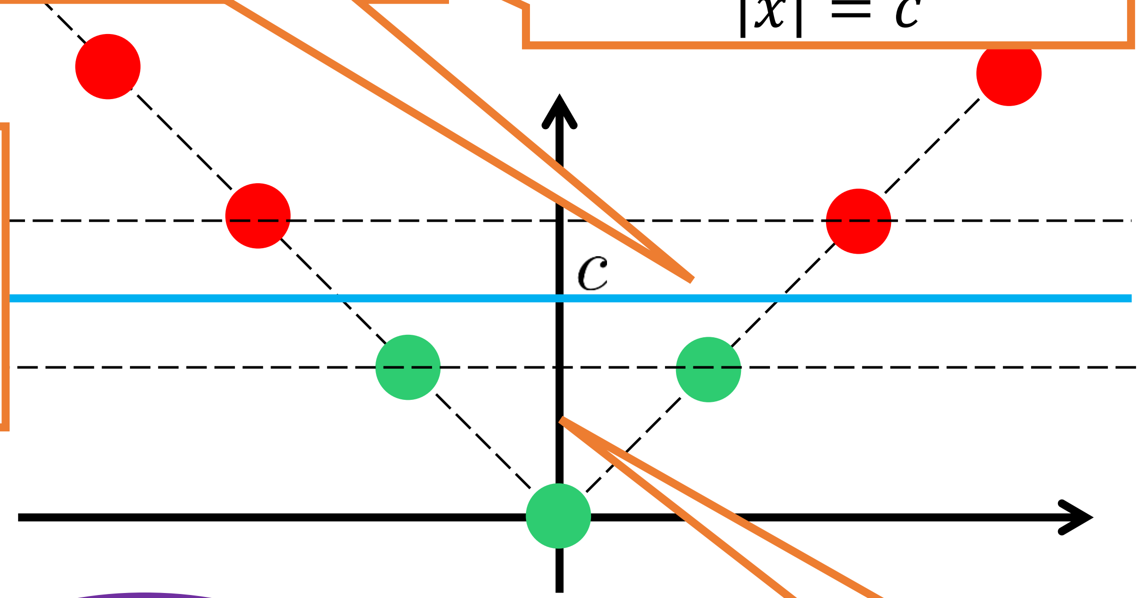
Where did non-linearity come from?

This classifier is $\phi_2(x) = c$

But $\phi_2(x) = |x|$ so it is really doing $|x| = c$

In the original space this looks like a non-linear classifier!





Points on the real line

$$\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$$

$$\phi(x) = [\phi_1(x), \phi_2(x)], \phi_1(x) = x, \phi_2(x) = |x|$$

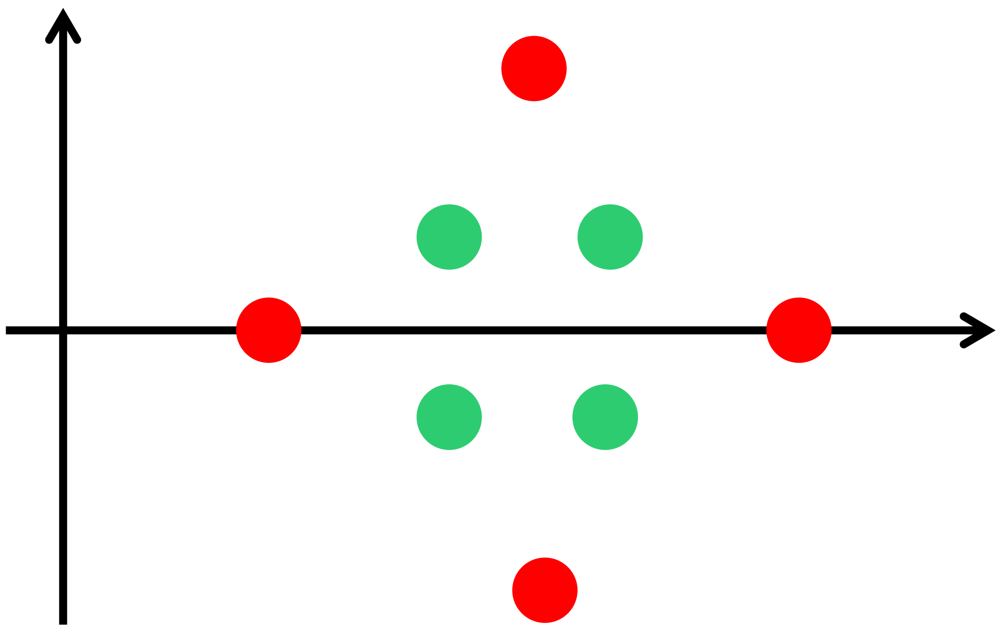
Not just classifiable but with a margin ☺

No linear classfier can separate them ☹

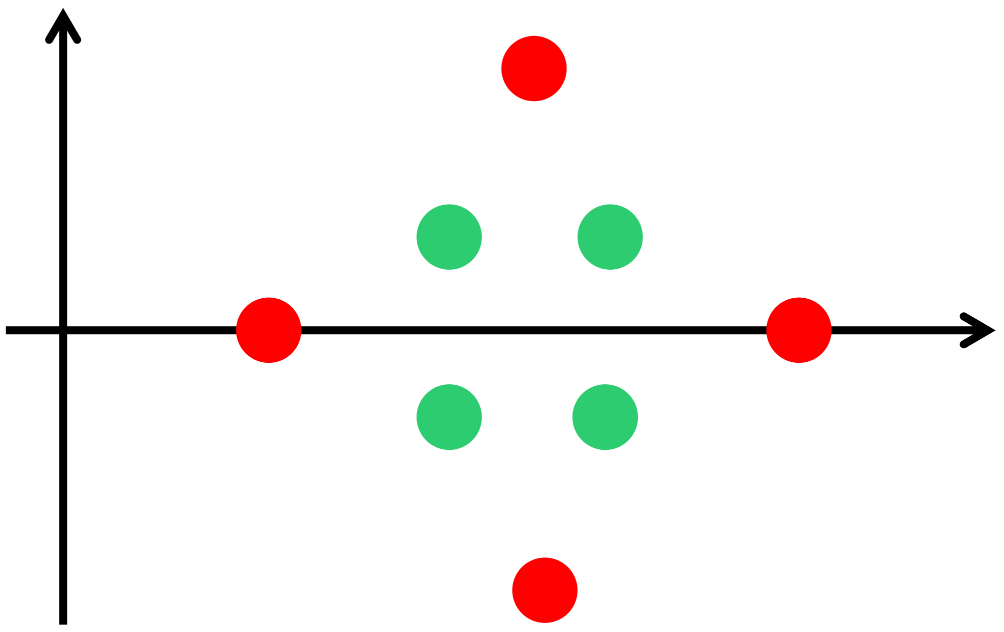
The mapping ϕ is non-linear! Not of the form $\phi(x) = Wx$ for some $W \in \mathbb{R}^{2\times1}$

A toy example: non-linear classification

A toy example: non-linear classification

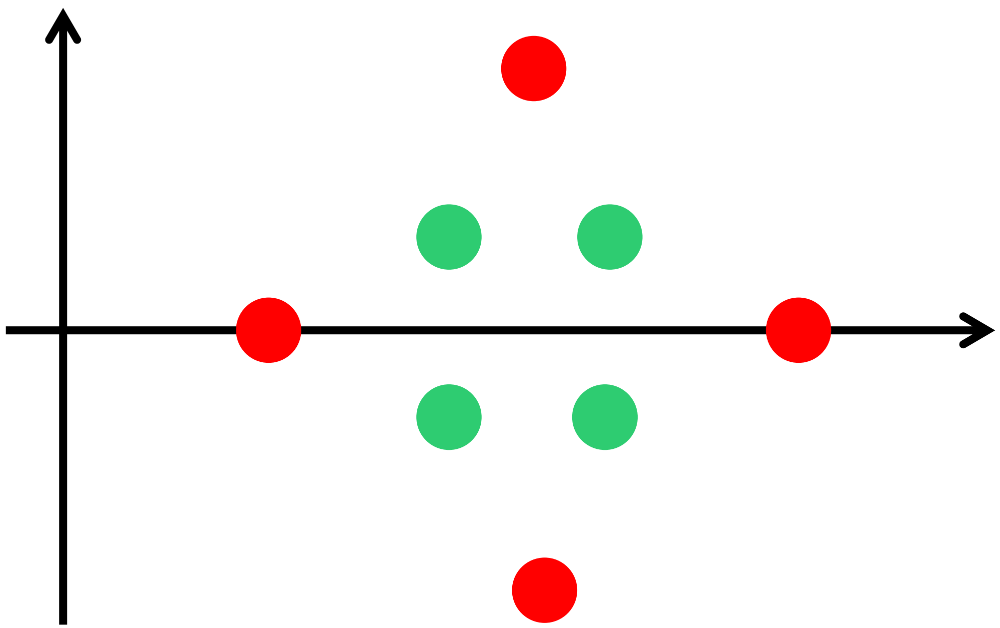


A toy example: non-linear classification



$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x, x^2, y^2] \in \mathbb{R}^3$$

A toy example: non-linear classification



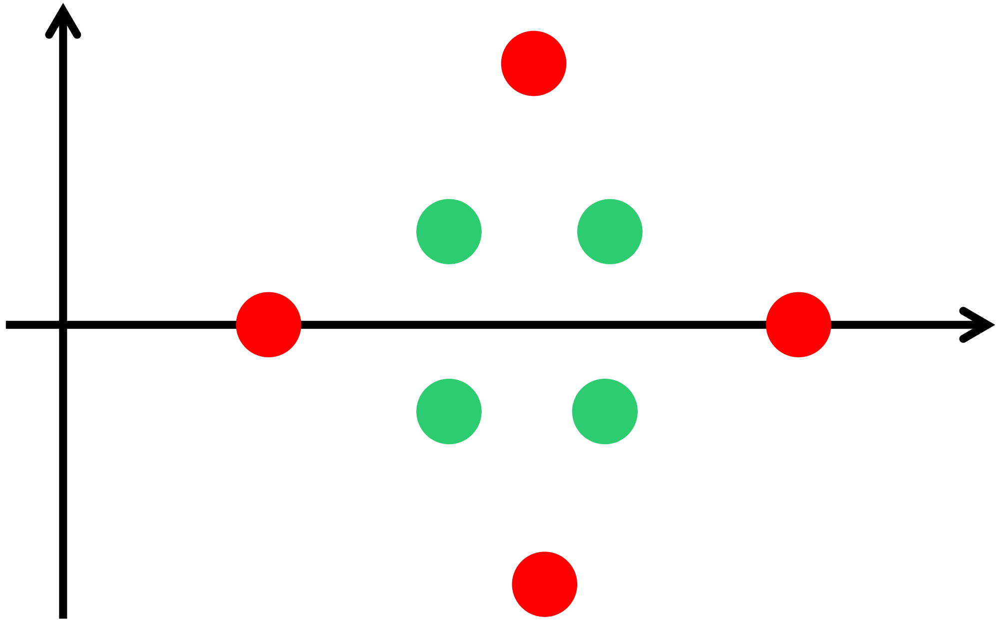
A linear function on this 3-D vector looks like

$\langle \mathbf{w}, \phi(x, y) \rangle$

$\mathbf{w}_1 \cdot x + \mathbf{w}_2 \cdot x^2 + \mathbf{w}_3 \cdot y^2$

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x, x^2, y^2] \in \mathbb{R}^3$$

A toy example: non-l on



Hmm … so a linear function in the 3D space can learn the decision boundary

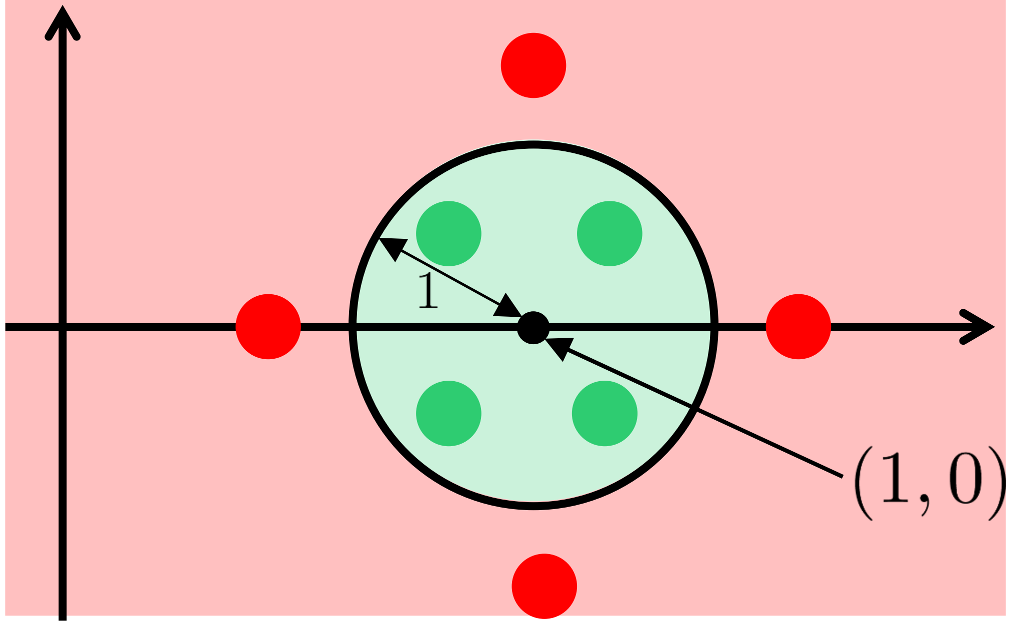
$$-2x + x^2 + y^2$$
$$= (x-1)^2 + y^2 - 1$$

A linear function on this 3-D vector looks like

$$\langle \mathbf{w}, \phi(x,y) \rangle$$
$$\mathbf{w}_1 \cdot x + \mathbf{w}_2 \cdot x^2 + \mathbf{w}_3 \cdot y^2$$

$$\mathbb{R}^2 \ni (x,y) \mapsto \phi(x,y) = [x, x^2, y^2] \in \mathbb{R}^3$$

A toy example: non-l… …on



Hmm … so a linear function in the 3D space can learn the decision boundary

$-2x + x^2 + y^2$
$= (x-1)^2 + y^2 - 1$

A linear function on this 3-D vector looks like

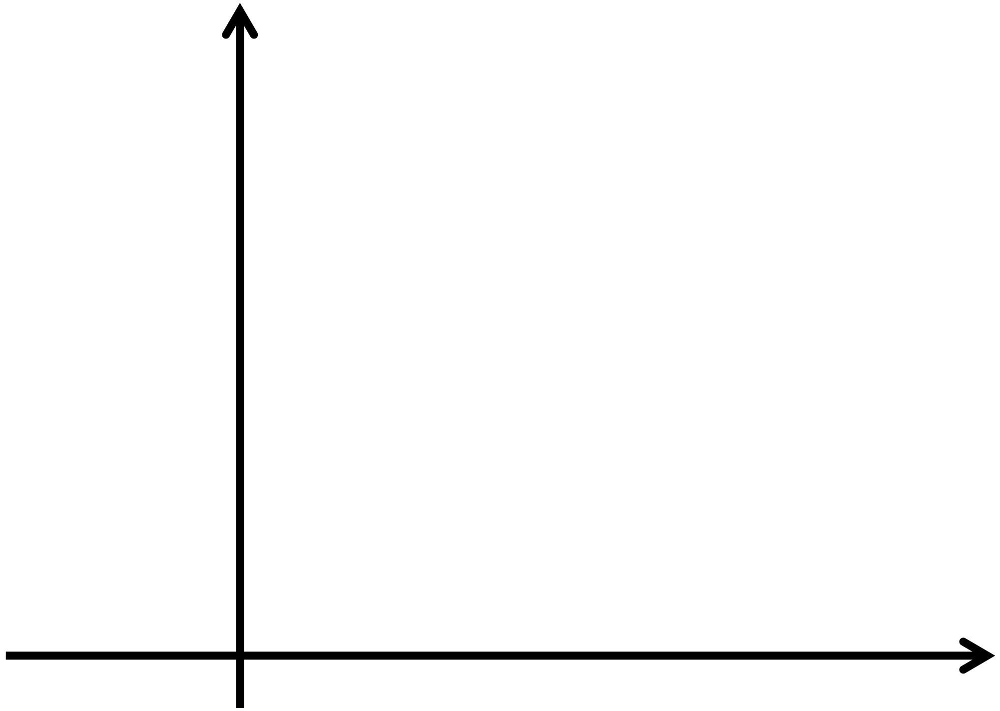
$\langle \mathbf{w}, \phi(x, y) \rangle$

$\mathbf{w}_1 \cdot x + \mathbf{w}_2 \cdot x^2 + \mathbf{w}_3 \cdot y^2$

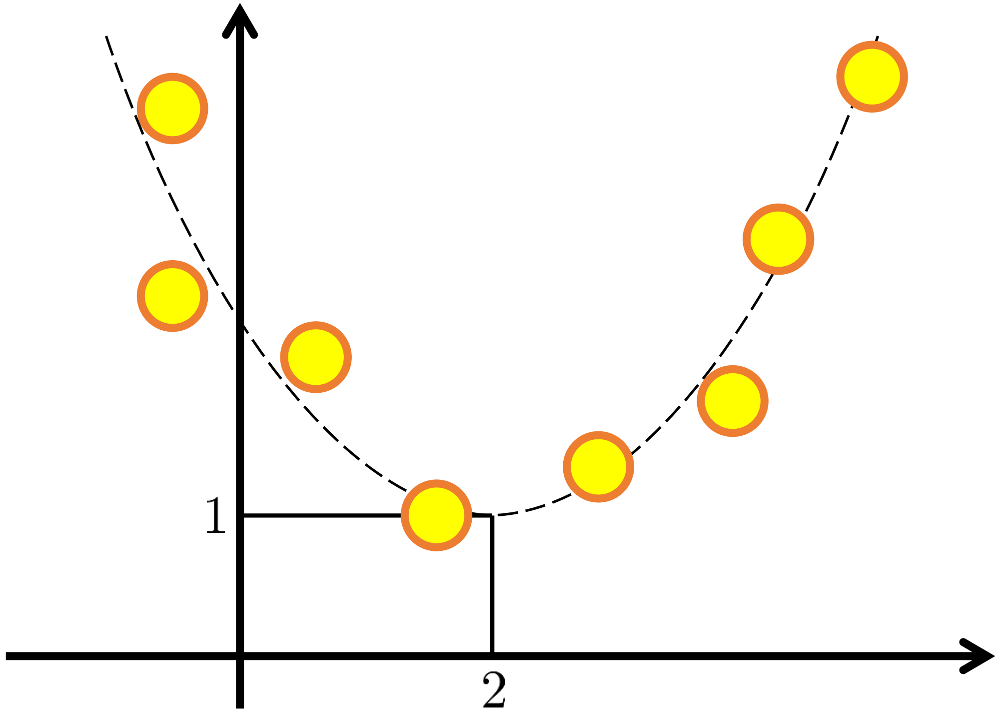
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x, x^2, y^2] \in \mathbb{R}^3$$

A toy example: non-linear regression

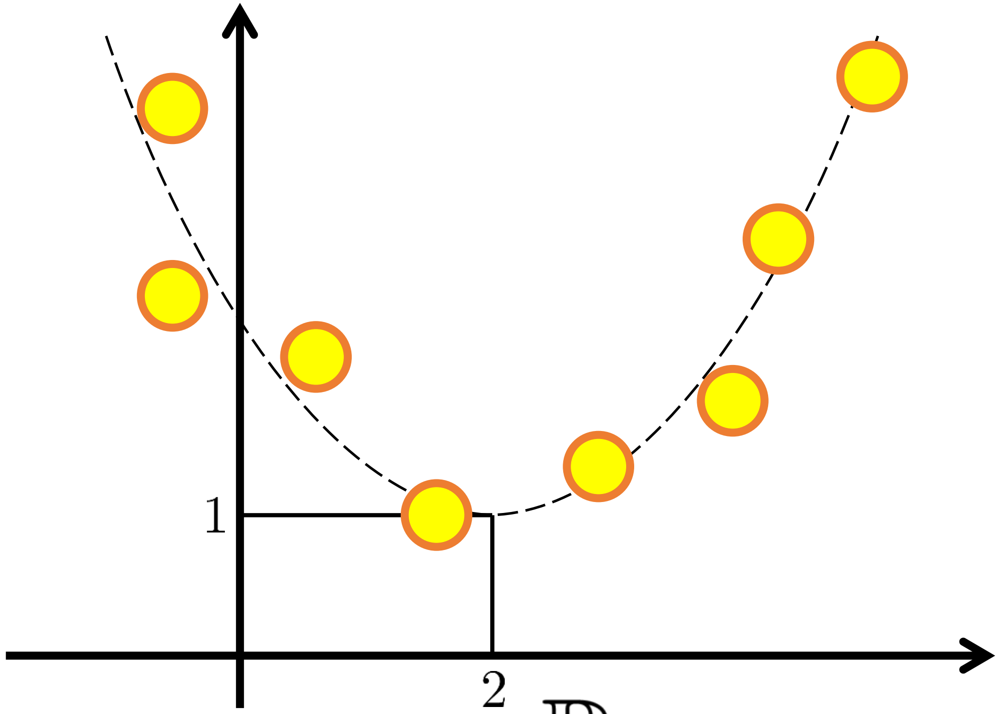
A toy example: non-linear regression



A toy example: non-linear regression

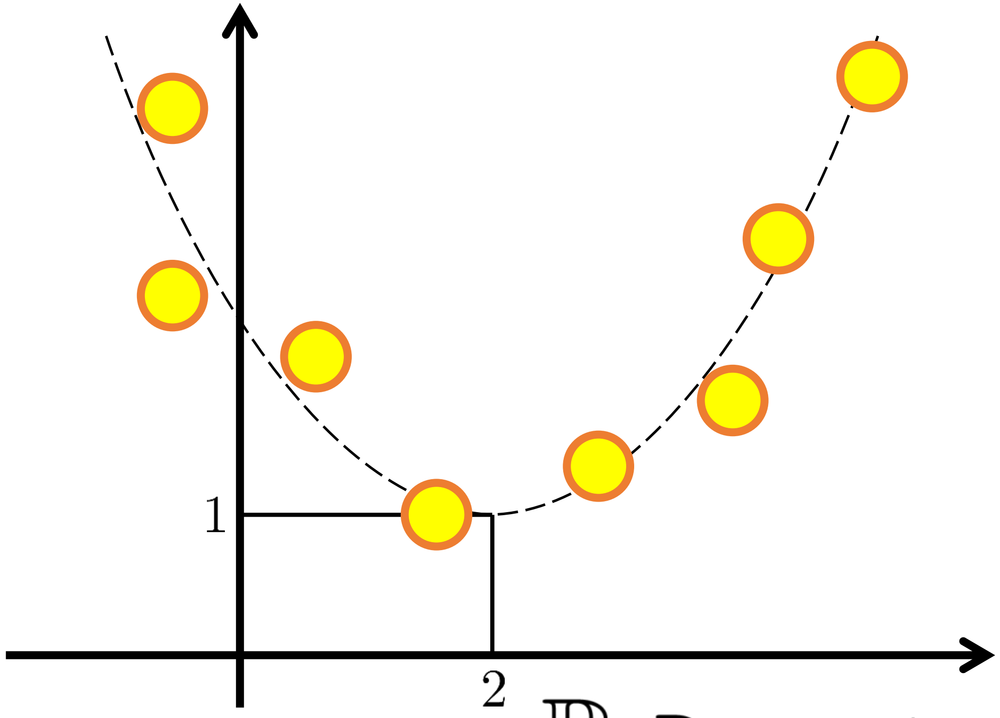


A toy example: non-linear regression



$$\mathbb{R} \ni x \mapsto \phi(x) = [1, x, x^2] \in \mathbb{R}^3$$

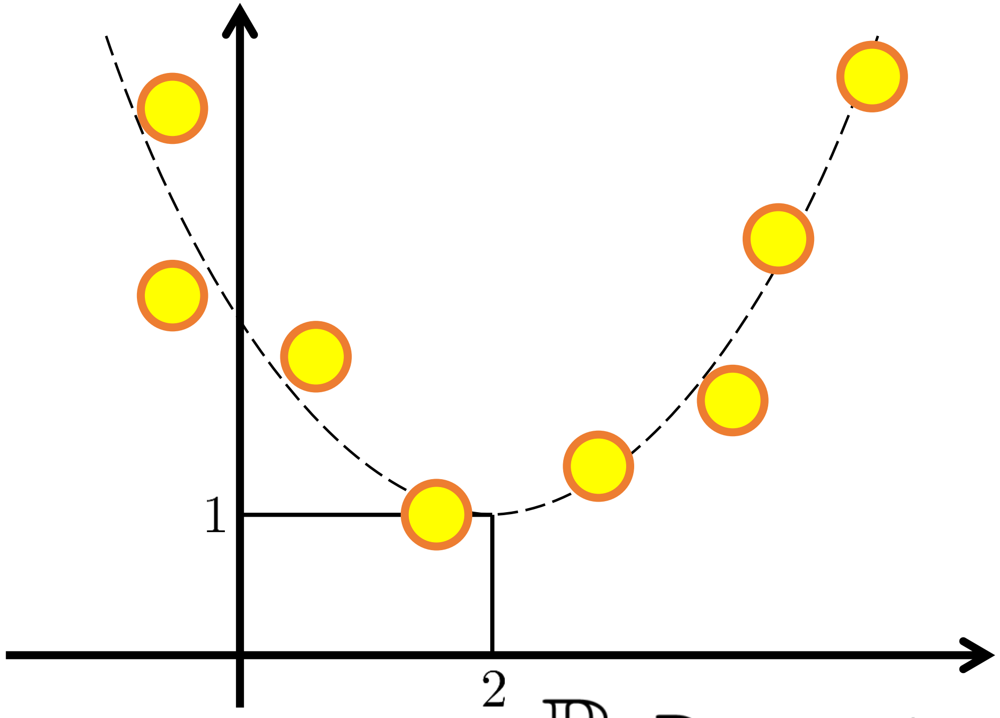
A toy example: non-linear regression



A function of the form
$(x-a)^2+c$
can be written as $\langle \mathbf{w}, \phi(x) \rangle$
for some vector $\mathbf{w}$

$$\mathbb{R} \ni x \mapsto \phi(x) = [1, x, x^2] \in \mathbb{R}^3$$

A toy example: non-linear regression

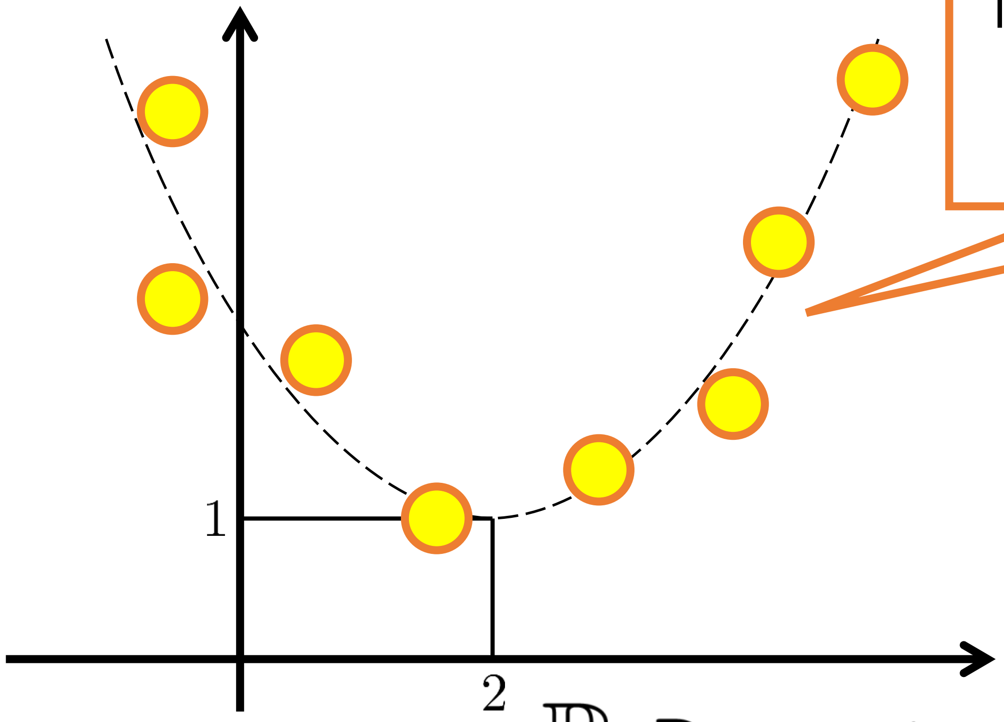


A function of the form
$(x-a)^2 + c$
can be written as $\langle \mathbf{w}, \phi(x) \rangle$
for some vector $\mathbf{w}$

$$\mathbb{R} \ni x \mapsto \phi(x) = [1, x, x^2] \in \mathbb{R}^3$$

Exercise!

A toy example: non-linear regression



The non-linear regressor
$y = (x-2)^2 + 1$
closely fits data

A function of the form
$(x-a)^2 + c$
can be written as $\langle \mathbf{w}, \phi(x) \rangle$
for some vector $\mathbf{w}$

$$\mathbb{R} \ni x \mapsto \phi(x) = [1, x, x^2] \in \mathbb{R}^3$$

Exercise!

Linear learning but non-linear effects

- Okay, so you are telling me that if I project data to a higher dimensional space in a non-linear manner, two things happen
  - There is a good chance a linear model will fit the data nicely in the higher dimensional space
  - If I learn a linear model up there, it may still induce non-linear effects down here since the projection was non-linear
  - And that non-linear model will obviously fit data well too
- However, calculating $\phi(x)$ for all data points is too much work
- If I really wanted to work all that hard why would I have wanted to reuse linear learning algorithms in the first place!!
- Maybe now I really need to learn about kernels

What are kernels?

Kernels vs Distances

$$K(\mathbf{x}, \mathbf{y}) \qquad\qquad d(\mathbf{x}, \mathbf{y})$$

Kernels

- Give measures of similarity
- High value ⇒ Similar points
- Example: wait a bit
- Nice similarity functions satisfy the *Mercer's theorem*
- Can be used for kNN (define neigh. using K instead of d) and much, much more … wait a bit

Distances

- Give measures of dissimilarity
- High value ⇒ Different points
- Example: Euclidean norm
- Nice distance functions satisfy metric or norm properties
- Can be used for classfn, regressn, multi-label via kNN, clustering via k-means

The *nice* Mercer Kernel

- For starters, let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ be unit real norm valued vectors
- An intuitive notion of similarity between them is dot product $\langle \mathbf{x}, \mathbf{y} \rangle$
- Dot product is highest when $\mathbf{x} = \mathbf{y}$. Then we have $\langle \mathbf{x}, \mathbf{y} \rangle = 1$
- Dot product is smallest when $\mathbf{x} = -\mathbf{y}$. Then we have $\langle \mathbf{x}, \mathbf{y} \rangle = -1$
- Mercer kernels are* nice kernels that extend this intuition

*Some technical details missing

The *nice* Mercer kernel

- Given any object set $\mathcal{X}$ (need not be vectors, may be set of images, video, strings, genome sequences), a similarity function
$$K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
is called a Mercer kernel if there exists a projection $\phi: \mathcal{X} \to \mathcal{H}$ such that for all $x, y \in \mathcal{X}$ we have $K(x, y) = \langle \phi(x), \phi(y) \rangle$
- ϕ often called feature map, embedding as well
- $\mathcal{H}$ can be $\mathbb{R}^D$ for some large/moderate D
- $\mathcal{H}$ can even be infinite dimensional
- In general $\mathcal{H}$ has to be a *Hilbert space*
- Technical details aside, Hilbert spaces are real vector spaces (possibly infinite dimensional) which have an inner (dot) product defined on them.

Examples of Kernels

- Can define kernels whenever have a sound notion of similarity
- All the following are Mercer kernels
- When $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are vectors
  - Linear kernel $K_{\text{lin}}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$
  - Quadratic kernel $K_{\text{quad}}(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$
  - Polynomial kernel $K_{\text{poly}}(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^p$, $c \geq 0, p \in \mathbb{N}$
  - Gaussian kernel $K_{\text{gauss}}(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \cdot \|\mathbf{x} - \mathbf{y}\|_2^2)$
  - Laplacian kernel $K_{\text{lap}}(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \cdot \|\mathbf{x} - \mathbf{y}\|_1)$
- p, γ need to be tuned. Large p, γ can cause overfitting
- Notice all are notions of similarity (assume $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$)
  - $K(\mathbf{x}, \mathbf{y})$ is largest when $\mathbf{x} = \mathbf{y}$ and smallest when $\mathbf{x} = -\mathbf{y}$

$\boxed{\text{Exercise!}}$

Really … all those kernels are Mercer?

- Yup. It is trivial for the linear kernel. Just use $\phi(\mathbf{x}) = \mathbf{x}$
- For the quadratic kernel, note that

$$(\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2 = 1 + 2 \cdot \langle \mathbf{x}, \mathbf{y} \rangle + \sum_{i,j}^{d} \mathbf{x}_i \mathbf{x}_j \mathbf{y}_i \mathbf{y}_j$$

$= \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$, where $\phi(\mathbf{x}) = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \phi_2(\mathbf{x})] \in \mathbb{R}^{d^2+d+1}$

- $\phi_0(\mathbf{x}) = [1] \in \mathbb{R}^1$
- $\phi_1(\mathbf{x}) = \sqrt{2} \cdot \mathbf{x} \in \mathbb{R}^d$
- $\phi_2(\mathbf{x}) = [\mathbf{x}_1\mathbf{x}_1, \mathbf{x}_1\mathbf{x}_2, \mathbf{x}_1\mathbf{x}_3, \ldots, \mathbf{x}_1\mathbf{x}_d, \mathbf{x}_2\mathbf{x}_1, \mathbf{x}_2\mathbf{x}_2, \ldots, \mathbf{x}_d\mathbf{x}_d] \in \mathbb{R}^{d^2}$
- For the polynomial kernel , the construction is similar.

Really … all those kernels are Mercer?

- The quadratic kernel is named so for a reason.
- Notice the feature map corresponding to K_{quad}
$$\phi(\mathbf{x}) = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \phi_2(\mathbf{x})]$$
- A linear function over this map for some $\mathbf{w} \in \mathbb{R}^{d^2+d+1}$
$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$
corresponds to a quadratic function over $\mathbf{x}$
- Verify this yourself when $\mathbf{x} \in \mathbb{R}$ $(d = 1)$ for sake of simplicity
- If we use the quadratic kernel, using just a linear model learning algorithm in the space $\mathbb{R}^{d^2+d+1}$, we can learn any quadratic function over the data in the original space $\mathbb{R}^d$
- A polynomial kernel of degree p similarly allows learning of degree p polynomial functions over original data

Really … all those kernels are Mercer?

- For Gaussian/Laplacian kernels, situation more intense
- The only feature maps ϕ that give rise to these kernels as $K_{\text{gauss/lap}}(\mathbf{x},\mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$ are infinite dimensional!

$$
\begin{aligned}
K_{\text{gauss}}(\mathbf{x},\mathbf{y}) &= \exp(-\gamma \cdot \|\mathbf{x}-\mathbf{y}\|_2^2) \\
&= \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\exp(-\gamma \cdot \|\mathbf{y}\|_2^2)\exp(2\gamma \cdot \langle \mathbf{x},\mathbf{y} \rangle) \\
&= \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\exp(-\gamma \cdot \|\mathbf{y}\|_2^2)\sum_{i=0}^{\infty}\frac{(2\gamma)^i}{i!} \cdot \langle \mathbf{x},\mathbf{y} \rangle^i
\end{aligned}
$$

- Wow, Gaussian kernel is an infinite linear combination of polynomial kernels of all orders!!

Really … all those kernels are Mercer?

- $K_{\text{gauss}}(\mathbf{x},\mathbf{y}) = \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\exp(-\gamma \cdot \|\mathbf{y}\|_2^2)\sum_{i=0}^{\infty}\frac{(2\gamma)^i}{i!}\cdot\langle\mathbf{x},\mathbf{y}\rangle^i$
- If ϕ_i is the feature map corresponding to the poly kernel $\langle\mathbf{x},\mathbf{y}\rangle^i$, then a possible feature map for Gaussian kernel is

$$\phi_{\text{gauss}}(\mathbf{x}) = \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\cdot\left[\frac{\sqrt{(2\gamma)^i}}{\sqrt{i!}}\cdot\phi_i(\mathbf{x})\right]_{i=1,2,\ldots,\infty} \in \mathcal{H}$$

- Learning a linear function over $\mathcal{H}$ amounts to learning an infinite-degree polynomial over $\mathbf{x}$
- These are very powerful kernels, often called *universal kernels*
- Using these kernels, theoretically speaking, one can learn *any* function over data*

*Some technical details missing

Really … all those kernels are Mercer?

- $K_{\text{gauss}}(\mathbf{x},\mathbf{y}) = \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\exp(-\gamma \cdot \|\mathbf{y}\|_2^2)\sum_{i=0}^{\infty}\frac{(2\gamma)^i}{i!}\cdot\langle\mathbf{x},\mathbf{y}\rangle^i$
- If ϕ_i is the feature map corresponding to the poly kernel $\langle\mathbf{x},\mathbf{y}\rangle^i$, then a possible feature map for Gaussian kernel is

$$\phi_{\text{gauss}}(\mathbf{x}) = \exp(-\gamma \cdot \|\mathbf{x}\|_2^2)\cdot\left[\frac{\sqrt{(2\gamma)^i}}{\sqrt{i!}}\cdot\phi_i(\mathbf{x})\right]_{i=1,2,\ldots,\infty} \in \mathcal{H}$$

- Learning a linear function over $\mathcal{H}$ amounts to learning an infinite-degree polynomial over $\mathbf{x}$
- These are very powerful kernels, often
- Using these kernels, theoretically speak function over data*

"possible"? You mean there can be more feature maps for the same kernel? Yes!

*Some technical details missing

Examples of Kernels contd …

- When $X, Y \subseteq \mathcal{U}$ are sets
  - Intersection kernel $K_{\text{int}}(X, Y) = |X \cap Y|$
  - Norm. Int. kernel $K_{\text{int-n}}(X, Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}$ (notice that $K_{\text{int-n}} \in (0,1)$)
  - This is just the linear kernel in disguise and hence Mercer
- When $\mathbf{x}, \mathbf{y}$ are strings/documents of words from a dictionary Σ
  - Let dictionary $\Sigma = \{w_1, w_2, \ldots, w_d\}$ have d words in it
  - Let $c_i(x)$ be the count of word i in string $\mathbf{x}$
  - Intersection kernel $K_{\text{int}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \min\{c_i(\mathbf{x}), c_i(\mathbf{y})\}$
  - Norm. Int. kernel $K_{\text{int-n}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \frac{\min\{c_i(\mathbf{x}), c_i(\mathbf{y})\}}{\sqrt{c_i(\mathbf{x}) \cdot c_i(\mathbf{y})}}$ (define $\frac{0}{0} = 0$)
- All of the above are Mercer kernels! Notice all encode similarity

Exercise!

Examples of Kernels

- Many, many other possibilities
- N-gram, substring, Fisher kernels between two strings
- Random walk kernels between two graphs
- Subtree, convolutional kernels between two trees
- …
- But what if I need to construct a kernel over images/videos/FB?

How to construct new Mercer kernels

- **Method 1**: operations on existing kernels
- If K_1, K_2 are existing Mercer kernels then
  - $K_3 = c_1 \cdot K_1 + c_2 \cdot K_2$ is also a Mercer kernel if $c_1, c_2 \geq 0$
  $$K_3(\mathbf{x}, \mathbf{y}) = c_1 \cdot K_1(\mathbf{x}, \mathbf{y}) + c_2 \cdot K_2(\mathbf{x}, \mathbf{y})$$
  - $K_4 = K_1 \cdot K_2$ is also a nice kernel
  $$K_4(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) \cdot K_2(\mathbf{x}, \mathbf{y})$$
- **Method 2**: find a new feature construction for the data
- Find a way to represent data x as a vector $\phi_{\text{new}}(x) \in \mathbb{R}^d$
$$K_{\text{new}}(x, y) = \langle \phi_{\text{new}}(x), \phi_{\text{new}}(y) \rangle$$
- **Method 3**: mix and match
- Take new data representation $\phi_{\text{new}}(x) \in \mathbb{R}^d$ and an old kernel K_{old}
$$K_{\text{newer}}(x, y) = K_{\text{old}}\big(\phi_{\text{new}}(x), \phi_{\text{new}}(y)\big)$$

Examples of Kernels

- Many, many other possibilities
- N-gram, substring, Fisher kernels between two strings
- Random walk kernels between two graphs
- Subtree, convolutional kernels between two trees

- …
- But what if I need to construct a kernel over images/videos/FB?
- Pyramid kernel used in vision … combination of Int kernels
- Okay, but what is so special or *nice* about Mercer kernels?

Linear learning but non-linear effects

- Okay, so you are telling me that if I project data to a higher dimensional space in a non-linear manner, two things happen
  - There is a good chance a linear model will fit the data nicely in the higher dimensional space
  - If I learn a linear model up there, it may still induce non-linear effects down here since the projection was non-linear
- However, calculating $\phi(x)$ for all data points is too much work
- If I really wanted to work all that hard why would I have wanted to reuse linear learning algorithms in the first place!!
- Need to learn about kernels first

Kernel SVMs

Executing an SVM in original space $\mathbb{R}^d$

Primal Formulation

- $\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\cdot\|\mathbf{w}\|_2^2$
  s.t. $1 - y^i\langle\mathbf{w},\mathbf{x}^i\rangle \leq 0$
- Have seen GD/SGD methods to solve this ☺

Dual Formulation

- $\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} \sum_{i=1}^n \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i,j}\boldsymbol{\alpha}_i\boldsymbol{\alpha}_j y^i y^j\langle\mathbf{x}^i,\mathbf{x}^j\rangle$
  s.t. $\alpha_i \geq 0$
- Have seen SCD methods to solve this ☺
- Can recover $\mathbf{w} = \sum_{i=1}^n \boldsymbol{\alpha}_i y^i \boldsymbol{x}^i$

Executing an SVM in original space $\mathbb{R}^d$

Primal Formulation

- $\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\cdot\|\mathbf{w}\|_2^2$
  s.t. $1 - y^i\langle\mathbf{w},\mathbf{x}^i\rangle \leq 0$
- Have seen GD/SGD methods to solve this ☺

Dual Formulation

- $\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} \sum_{i=1}^n \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i,j}\boldsymbol{\alpha}_i\boldsymbol{\alpha}_j y^i y^j\langle\mathbf{x}^i,\mathbf{x}^j\rangle$
  s.t. $\alpha_i \geq 0$
- Have seen SCD methods to solve this ☺
- Can recover $\mathbf{w} = \sum_{i=1}^n \boldsymbol{\alpha}_i y^i \boldsymbol{x}^i$

But my problem does horribly with linear classifiers. I really need non-linearity

Hmm … okay. Choose a nice feature map/projection $\phi(\cdot): \mathbb{R}^d \to \mathcal{H}$ and solve the SVM in the space of $\mathcal{H}$

Executing an SVM in $\mathcal{H}$

Primal Formulation

- $\min_{\mathbf{w}\in\mathcal{H}} \frac{1}{2} \cdot \|\mathbf{w}\|_2^2$
  s.t. $1 - y^i\langle\mathbf{w}, \phi(\mathbf{x}^i)\rangle \leq 0$
- But I chose $\mathcal{H}$ to be really large dimensional
- A single step of GD/SGD will take enormous time to run ☹

Executing an SVM in $\mathcal{H}$

Primal Formulation

- $\min_{\mathbf{w} \in \mathcal{H}} \frac{1}{2} \cdot \|\mathbf{w}\|_2^2$
  s.t. $1 - y^i \langle \mathbf{w}, \phi(\mathbf{x}^i) \rangle \leq 0$
- But I chose $\mathcal{H}$ to be really large dimensional
- A single step of GD/SGD will take enormous time to run ☹

Dual Formulation

- $\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \sum_{i=1}^n \boldsymbol{\alpha}_i - \frac{1}{2} \sum_{i,j} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle$
  s.t. $\alpha_i \geq 0$
- Wow … this still has only n variables
- But what about $\langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle$?
- If only I had a way to compute this fast 😐
- Kernels !!!
- Choose a kernel K and take $\phi = \phi_K$ to be its feature map
- We will get $\langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle = K(\boldsymbol{x}^i, \boldsymbol{x}^j)$

Executing an SVM in $\mathcal{H}$

Primal Formulation

- $\min_{\mathbf{w} \in \mathcal{H}} \frac{1}{2} \cdot \|\mathbf{w}\|_2^2$
  s.t. $1 - y^i \langle \mathbf{w}, \phi(\mathbf{x}^i) \rangle \leq 0$
- But I chose $\mathcal{H}$ to be really large dimensional
- A single step of GD/SGD will take enormous time to run ☹
- Can now choose $\mathcal{H}$ to be infinite-dim as well ☺
- How do I recover $\mathbf{w}$ from $\boldsymbol{\alpha}$? Uh oh …

Dual Formulation

- $\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \sum_{i=1}^n \boldsymbol{\alpha}_i - \frac{1}{2} \sum_{i,j} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle$
  s.t. $\alpha_i \geq 0$
- Wow … this still has only n variables
- But what about $\langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle$?
- If only I had a way to compute this fast 😐
- Kernels !!!
- Choose a kernel K and take $\phi = \phi_K$ to be its feature map
- We will get $\langle \phi(\mathbf{x}^i), \phi(\mathbf{x}^j) \rangle = K(\boldsymbol{x}^i, \boldsymbol{x}^j)$

Using an SVM learnt in $\mathcal{H}$

- **Exercise**: take the Gaussian kernel and show how a single step of SCD on the dual can be executed in $\mathcal{O}(nd)$ time
- Note: d is not the dimensionality of $\mathcal{H}$ which may be infinite dim
- So we can argue that SCD is really still taking $\mathcal{O}(n)$ time ☺
- But once we get a dual solution $\boldsymbol{\alpha}$ how do we get a classifier?
- Getting $\mathbf{w} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i y^i \phi(\boldsymbol{x}^i)$ pointless since $\mathbf{w}$ is infinite/large dim.
- Instead, exploit the fact the we will only need to calculate

$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

  to classify a test point $\mathbf{x}$
- $\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \langle \sum_{i=1}^{n} \boldsymbol{\alpha}_i y^i \phi(\boldsymbol{x}^i), \phi(\mathbf{x}) \rangle = \sum_{i=1}^{n} \boldsymbol{\alpha}_i y^i \langle \phi(\boldsymbol{x}^i), \phi(\mathbf{x}) \rangle$
  $= \sum_{i=1}^{n} \boldsymbol{\alpha}_i y^i K(\mathbf{x}^i, \mathbf{x})$

Using an SVM learnt in $\mathcal{H}$

Outliers, redundant points should have $\boldsymbol{\alpha}_i \approx 0$

$\boldsymbol{\alpha}_i$ indicates how useful the data point $(\mathbf{x}^i, y^i)$ is while voting

If $\mathbf{x}^i$ "similar" to $\mathbf{x}$ i.e. $K(\mathbf{x}^i, \mathbf{x})$ is large, y^i influences predicted label more – nice!!

- Note: d is not the dime…
- So we can argue that SC… really still taking…
- But once we get a c… we get a classifier?

Note that this looks like soft weighted k-NN

- Getting $\mathbf{w} = \sum_{i=1}^n \boldsymbol{\alpha}_i y$… $\mathbf{w}$ is infinite/large dim.
- Instead, exploit the f… the we w… late

$\langle \mathbf{w},$…

to classify a test point $\mathbf{x}$

Need to perform $\tilde{n}$ kernel evaluations to predict on a test point

- $\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \langle \sum_{i=1}^n \boldsymbol{\alpha}_i y^i \phi(\mathbf{x}^i), \phi(\mathbf{x}) \rangle = \sum_{i=1}^n \boldsymbol{\alpha}_i y^i \langle \phi$…

$= \sum_{i=1}^n \boldsymbol{\alpha}_i y^i K(\mathbf{x}^i, \mathbf{x})$

$\tilde{n}$ is the number of support vectors which have $\boldsymbol{\alpha}_i \neq 0$

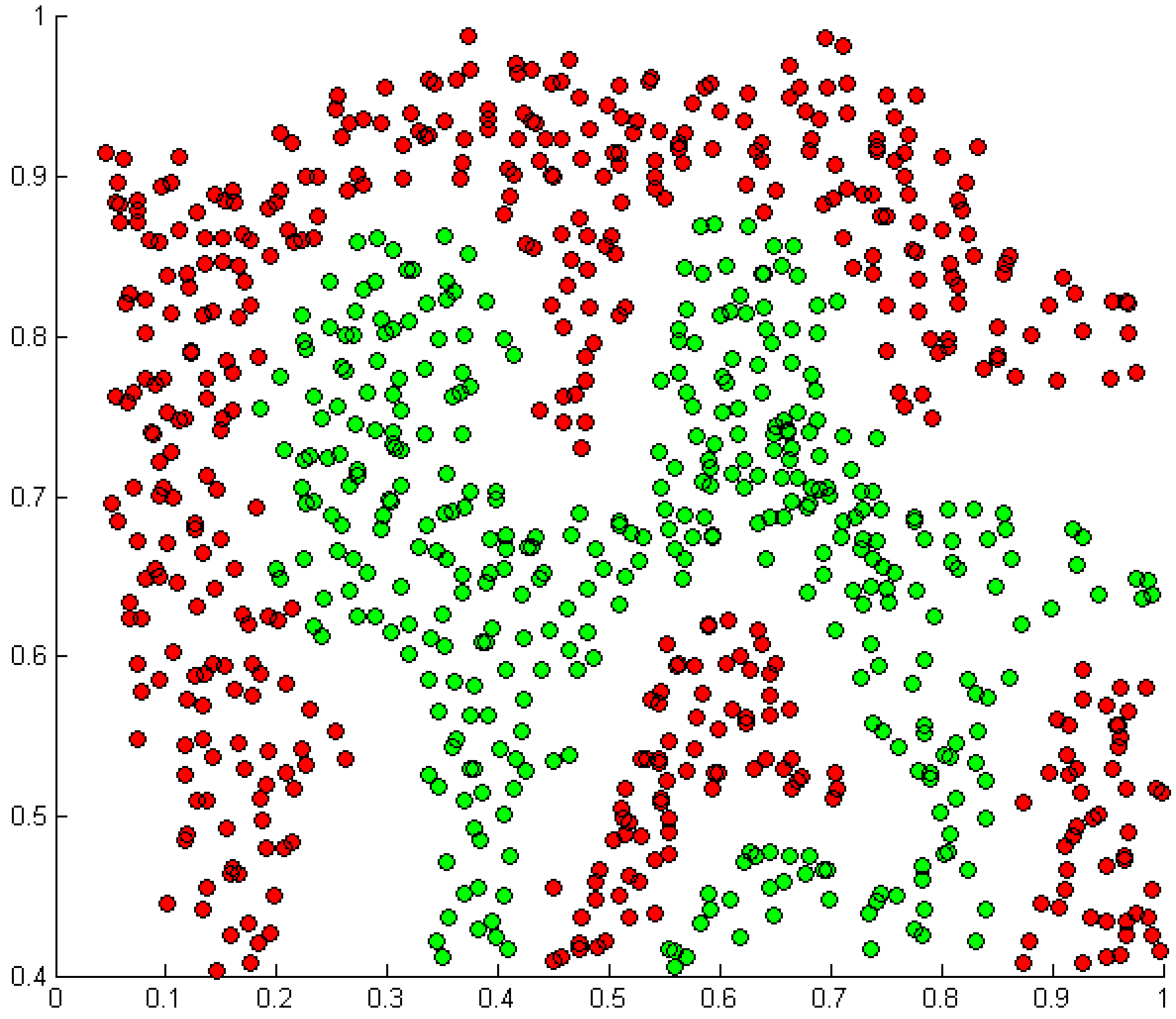
Some Pros and Cons

- Kernels allow us to learn powerful non-linear functions
- Use the libsvm library (even scikit-learn uses it internally)
- But they are more expensive to work with
- More expensive at training time -- $\mathcal{O}(nd)$ time for one SCD step
- More expensive to store learnt model –– need to store all support vectors -- $\mathcal{O}(\tilde{n}d)$ storage
- More expensive at prediction time -- $\mathcal{O}(\tilde{n}d)$ time to predict
- Linear SVMs took $\mathcal{O}(n)$ time for one SCD step, $\mathcal{O}(d)$ storage (simply store $\mathbf{w}$), $\mathcal{O}(d)$ time to predict (simply calculate $\langle \mathbf{w}, \mathbf{x} \rangle$)
- But … they are powerful and beautiful … !!!

Example

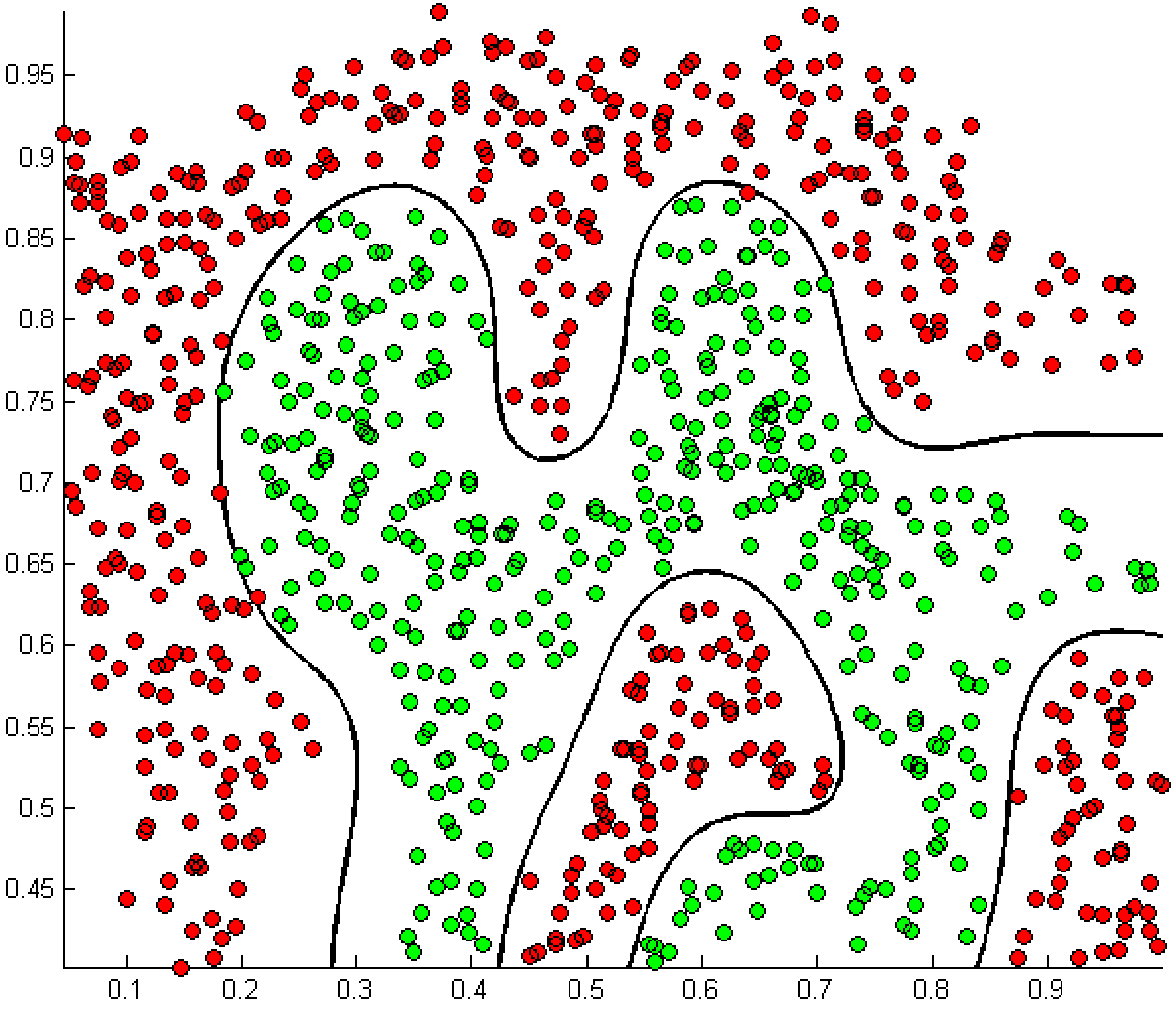
http://openclassroom.stanford.edu

Example



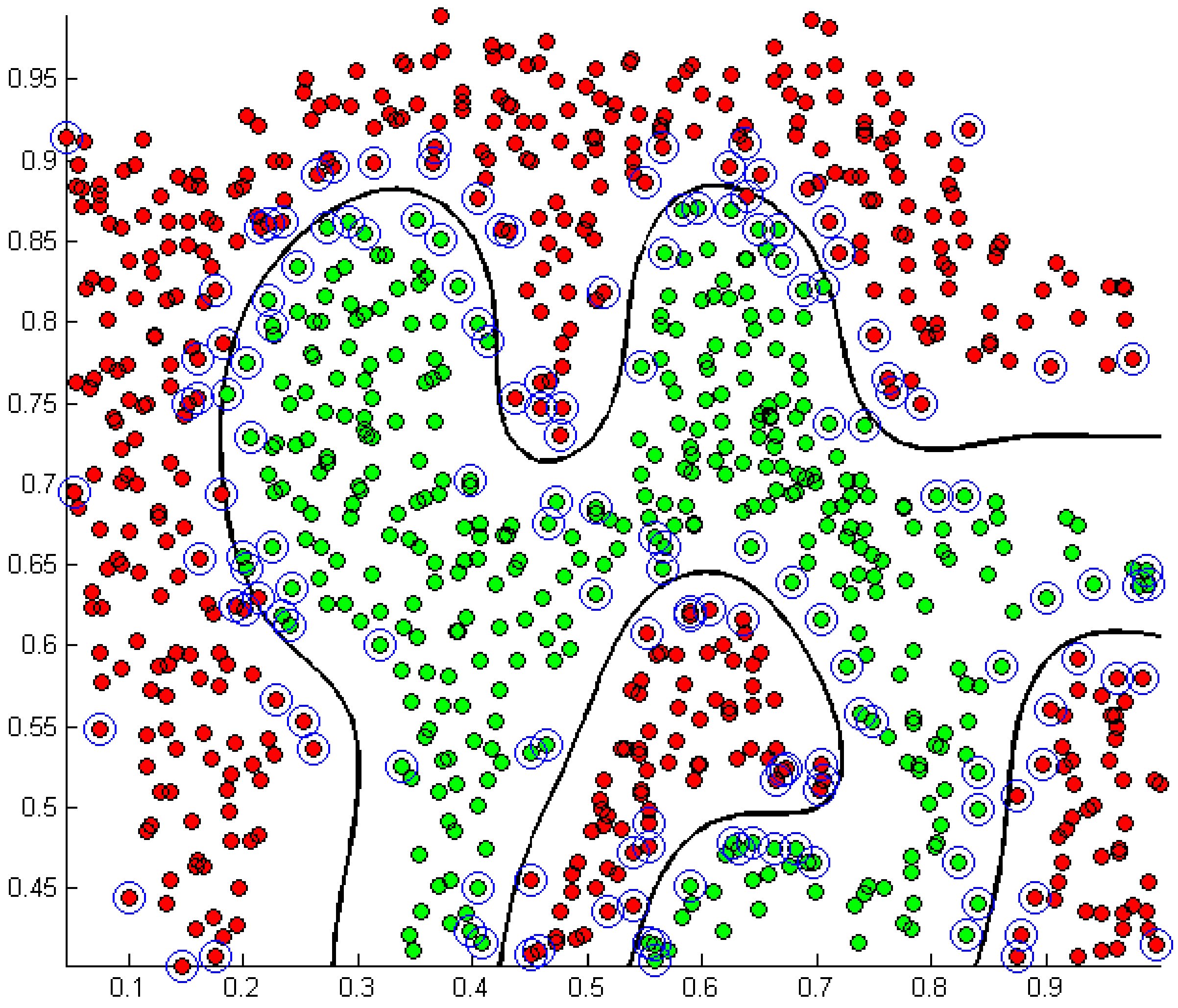
http://openclassroom.stanford.edu

Example



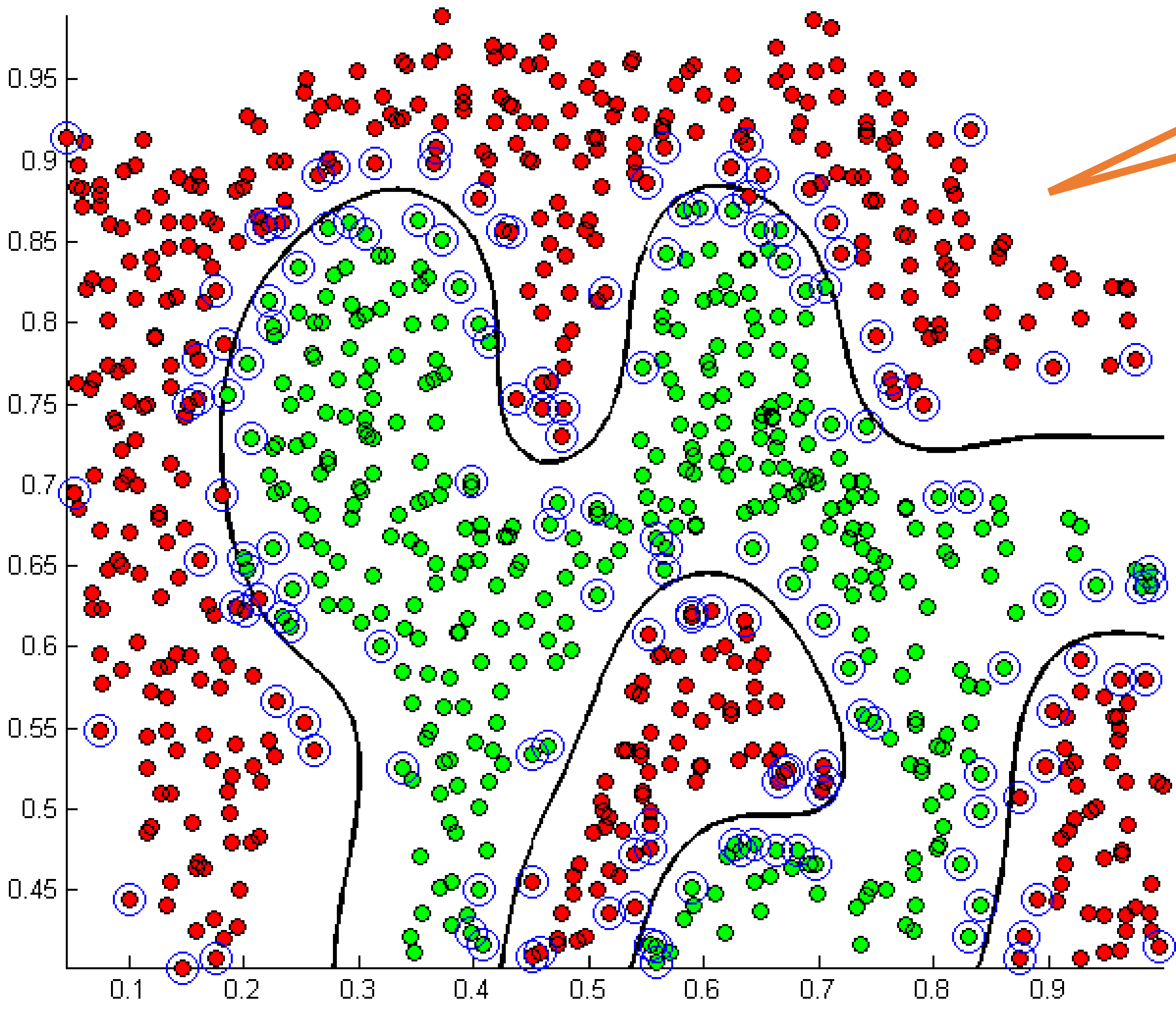
http://openclassroom.stanford.edu

Example



http://openclassroom.stanford.edu

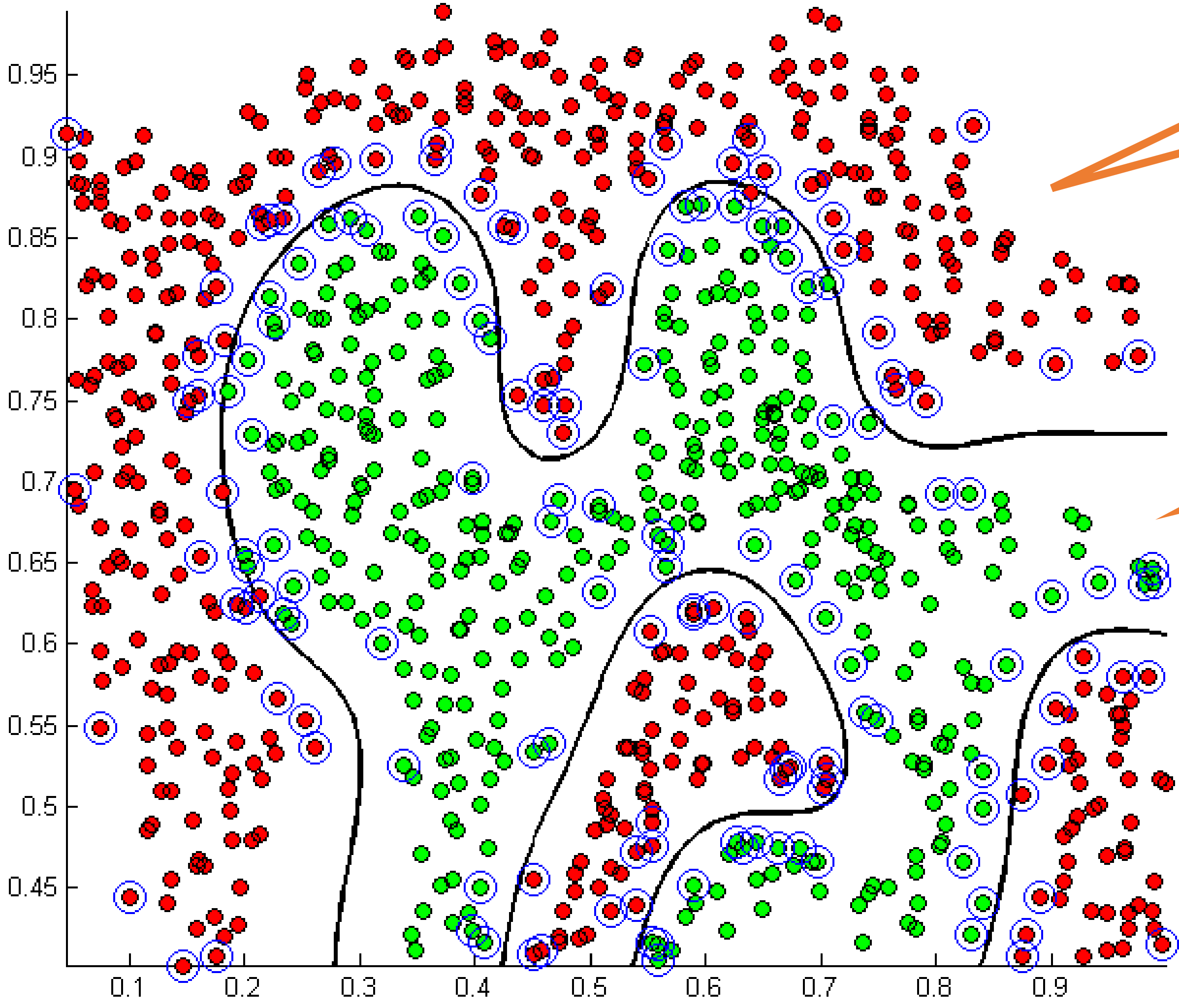
Example



Gaussian kernel with $\gamma = 100$, 850 points, 150 SVs

http://openclassroom.stanford.edu

Example



Gaussian kernel with $\gamma = 100$, 850 points, 150 SVs

Most SVs close to the decision boundary

http://openclassroom.stanford.edu

Please give your Feedback

http://tinyurl.com/ml17-18afb