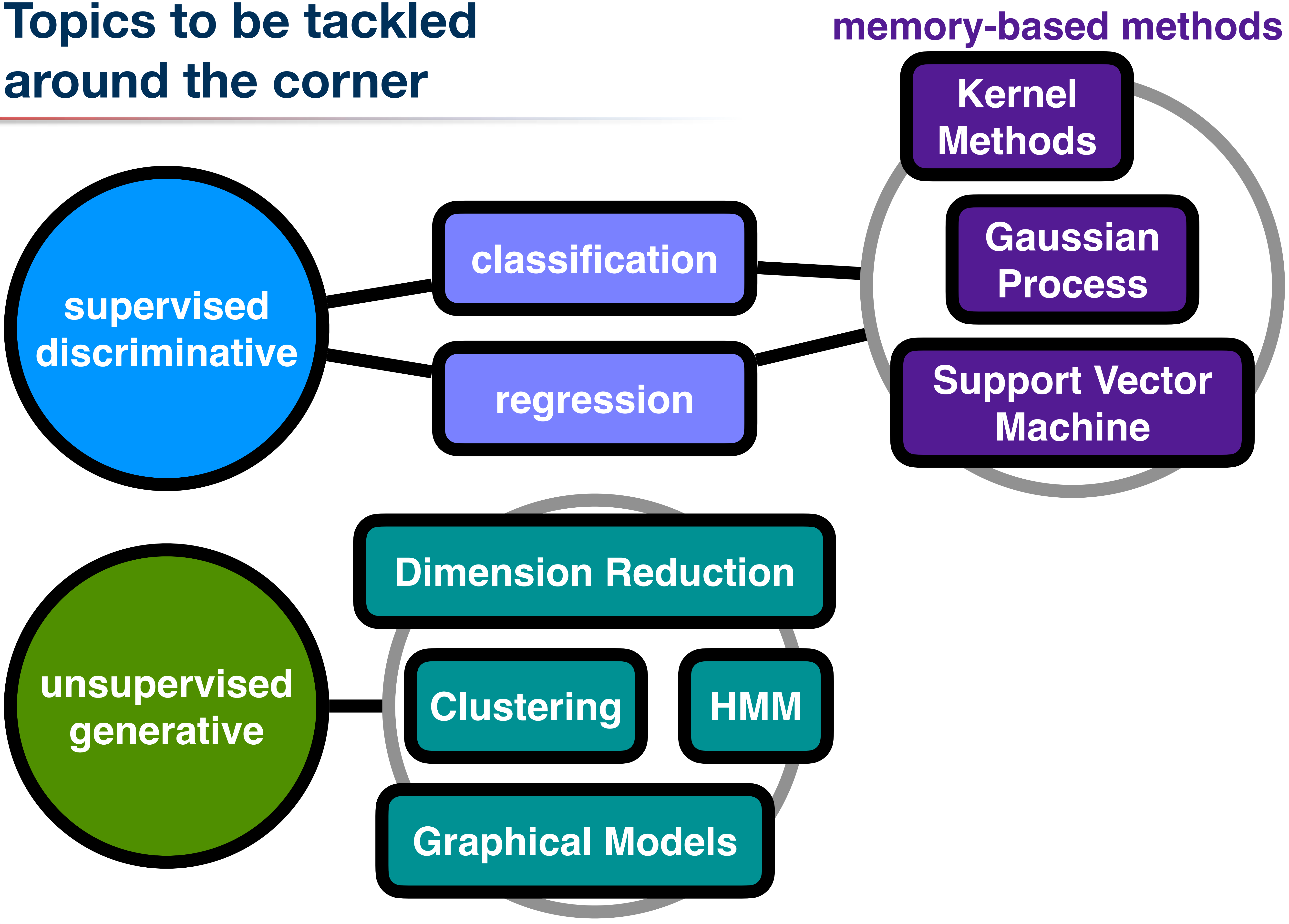
國立交通大學
National Chiao Tung University

Kernel, Gaussian Process, SVM
ML 2021 Fall

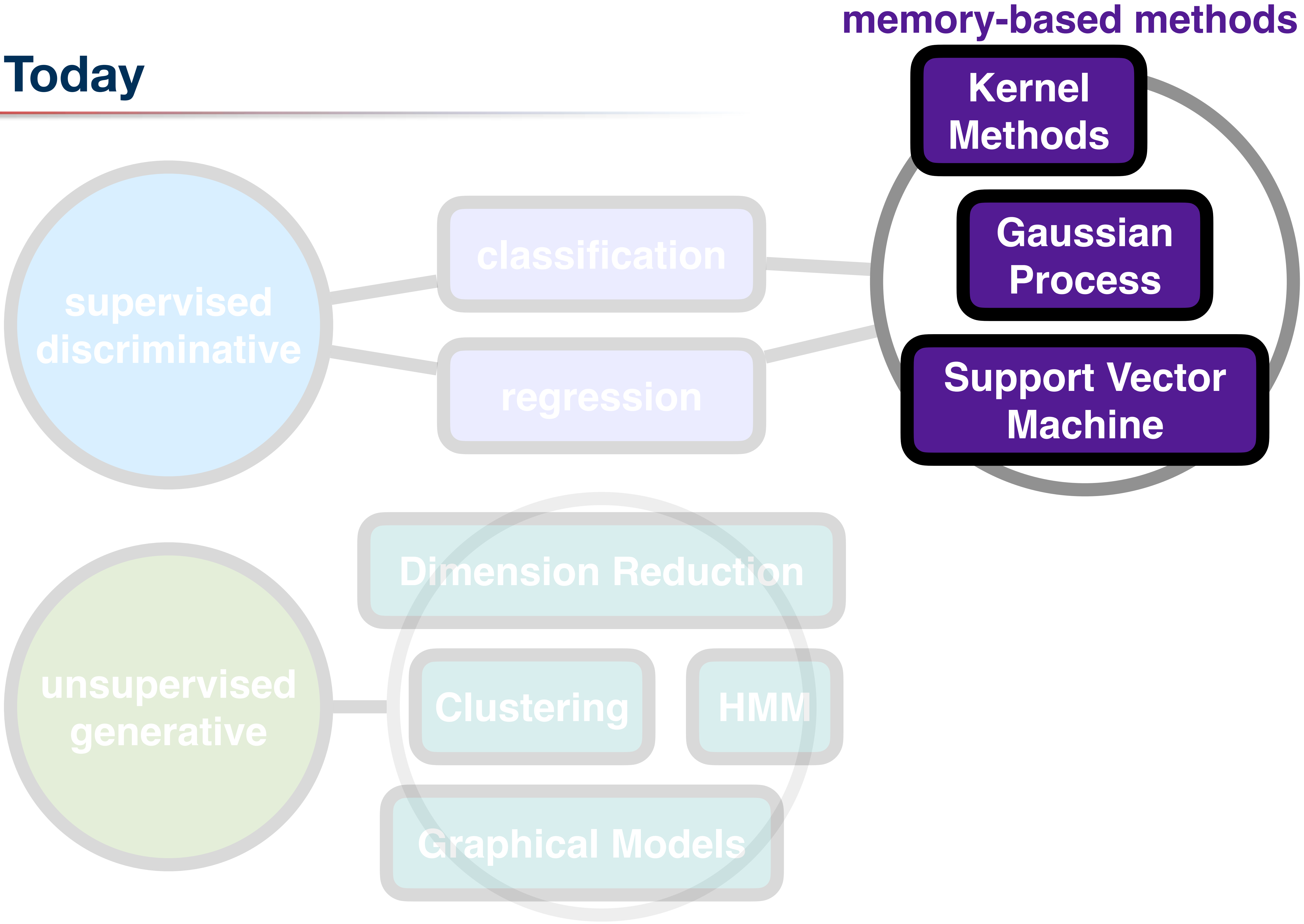
Wei-Chen (Walon) Chiu
邱維辰

Enriched Vision Applications
Laboratory

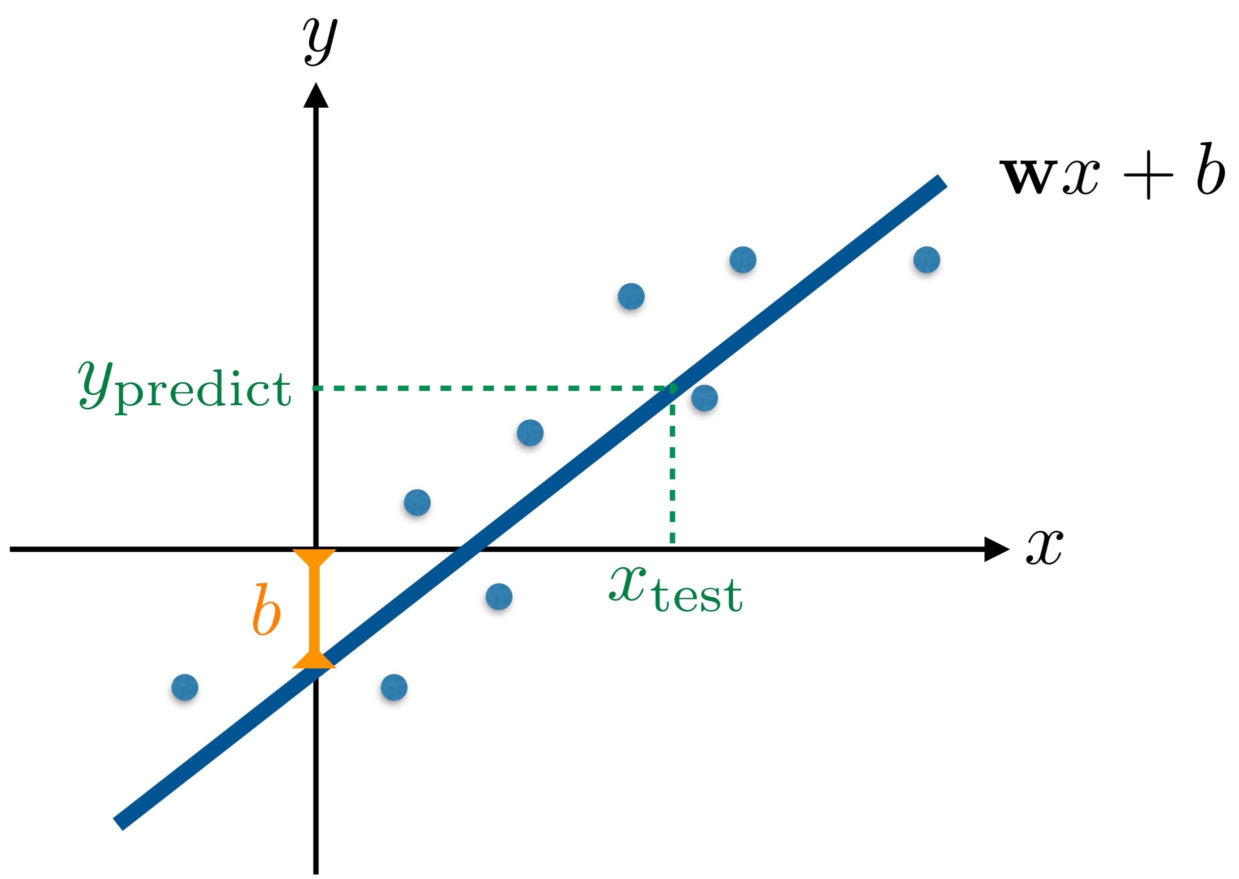
Topics to be tackled around the corner



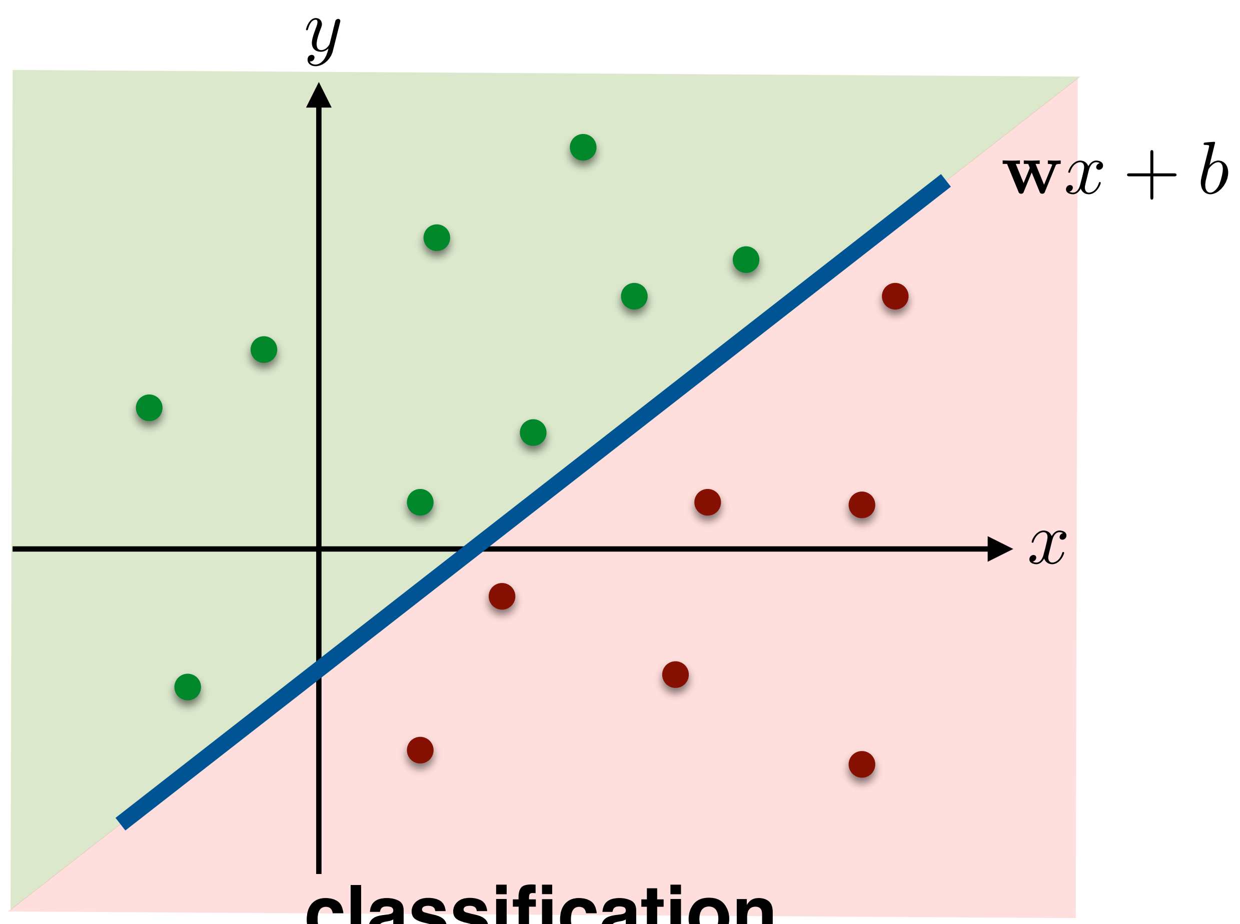
Today


memory-based methods
Kernel Methods
Gaussian Process
Support Vector Machine
supervised discriminative
classification
regression
unsupervised generative
Dimension Reduction
Clustering
HMM
Graphical Models

What you guys have learnt



regression



classification

- No matter linear parametric models for regression or classification, through away the training data after learning the parameters of the model.

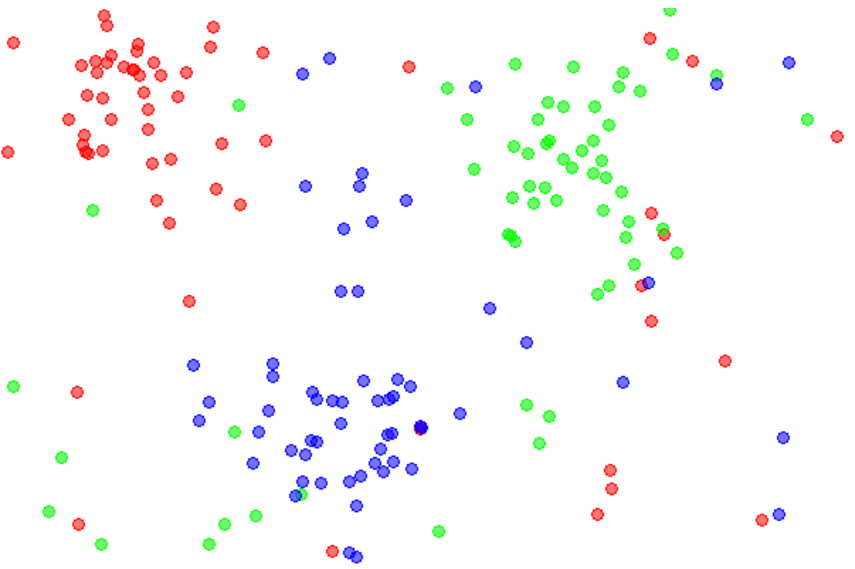
- Make prediction based on those parameters only.

Another way of thinking

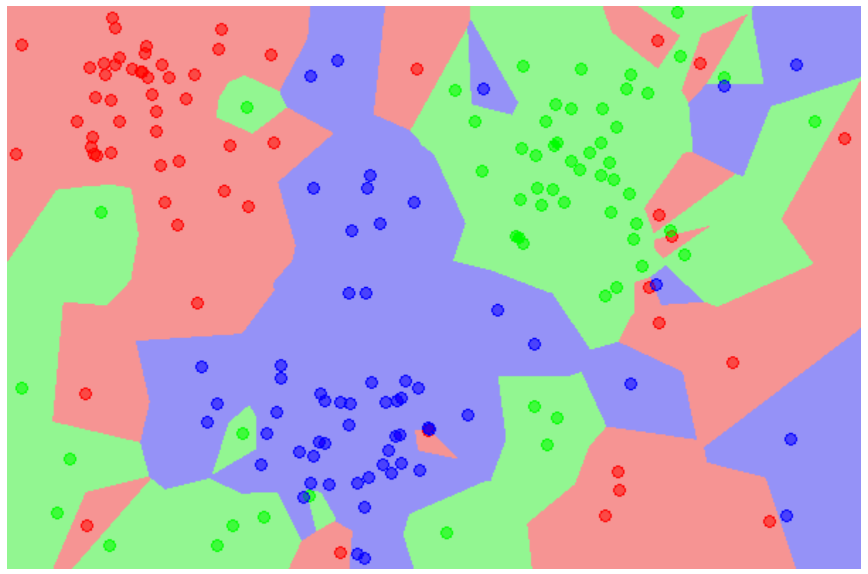
- Training data is valuable, try to keep it even till prediction
  - **Memory-based methods**
  - **We would like to utilise the training data for prediction!**
- For instance, KNN, Parzen

Another way of thinking

- Training data is valuable, try to keep it even till prediction
  - **Memory-based methods**
  - **We would like to utilise the training data for prediction!**
- For instance, **KNN**, Parzen probability estimation



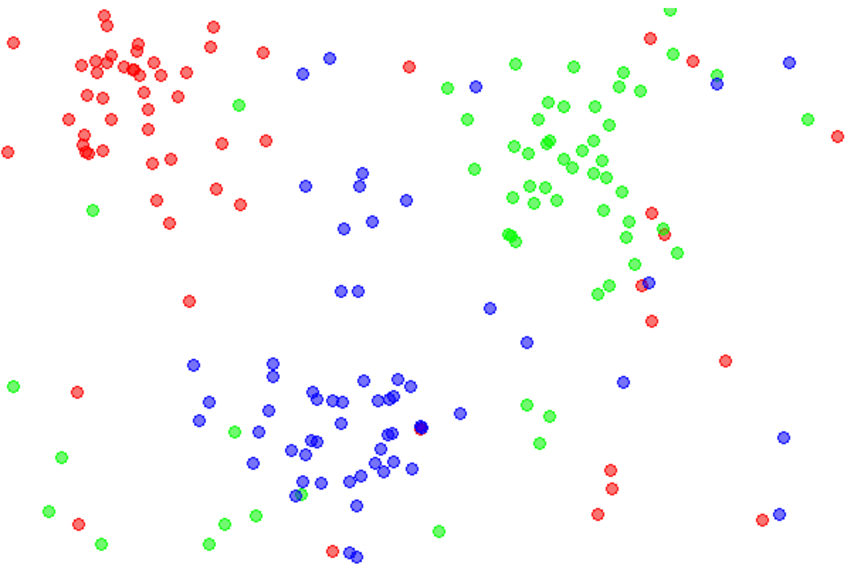
dataset

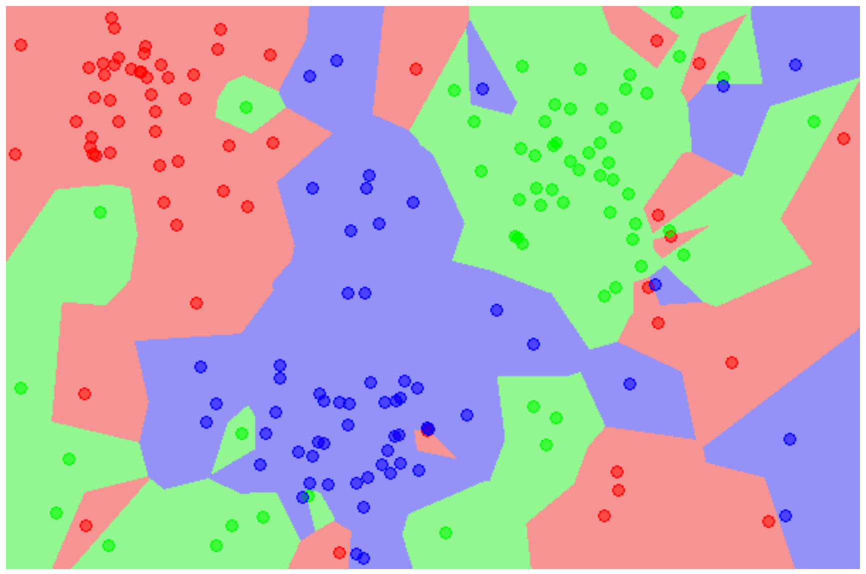


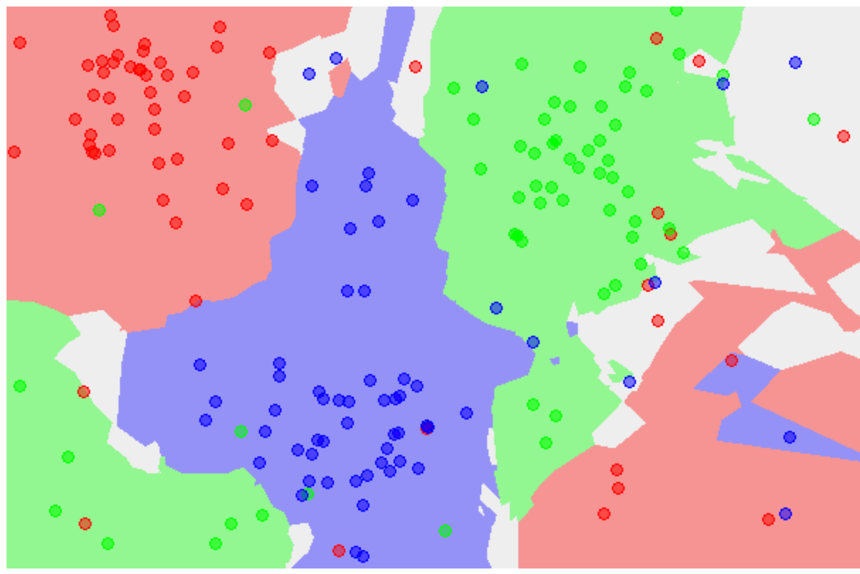
NN (nearest neighbour): find the closest training data point, take its class as prediction

Another way of thinking

- Training data is valuable, try to keep it even till prediction
  - **Memory-based methods**
  - **We would like to utilise the training data for prediction!**
- For instance, **KNN**, Parzen probability estimation

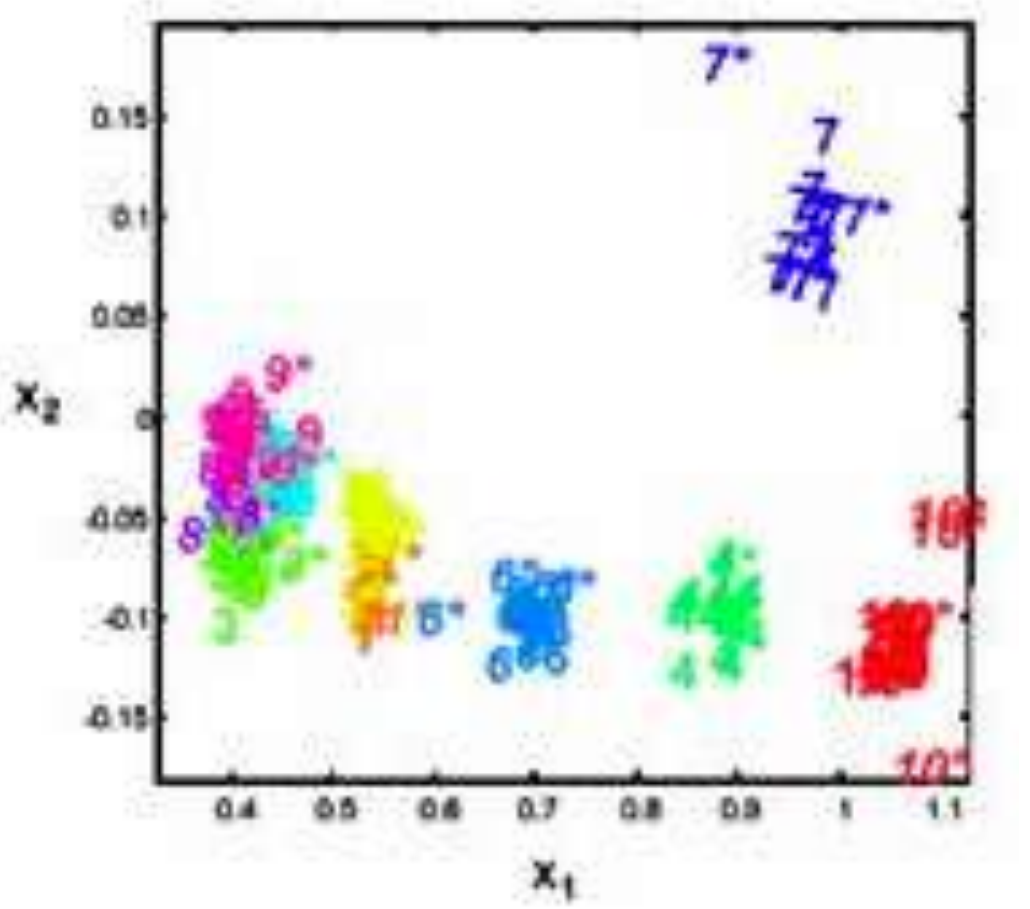

dataset


NN (nearest neighbour):

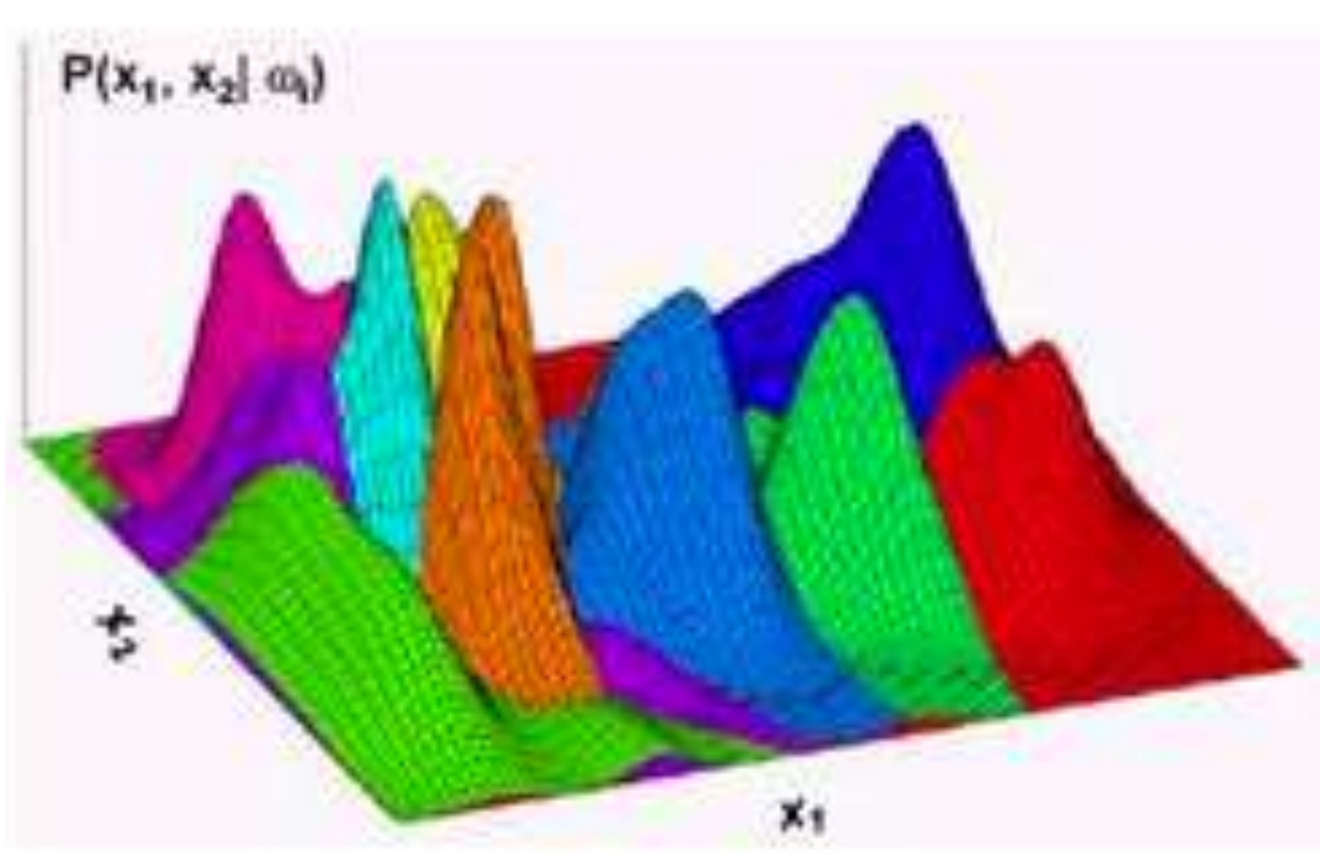

**KNN:
prediction based on
K closest training data
points, do majority vote**

Another way of thinking

- Training data is valuable, try to keep it even till prediction
  - **Memory-based methods**
  - **We would like to utilise the training data for prediction!**
- For instance, KNN, **Parzen probability estimation**



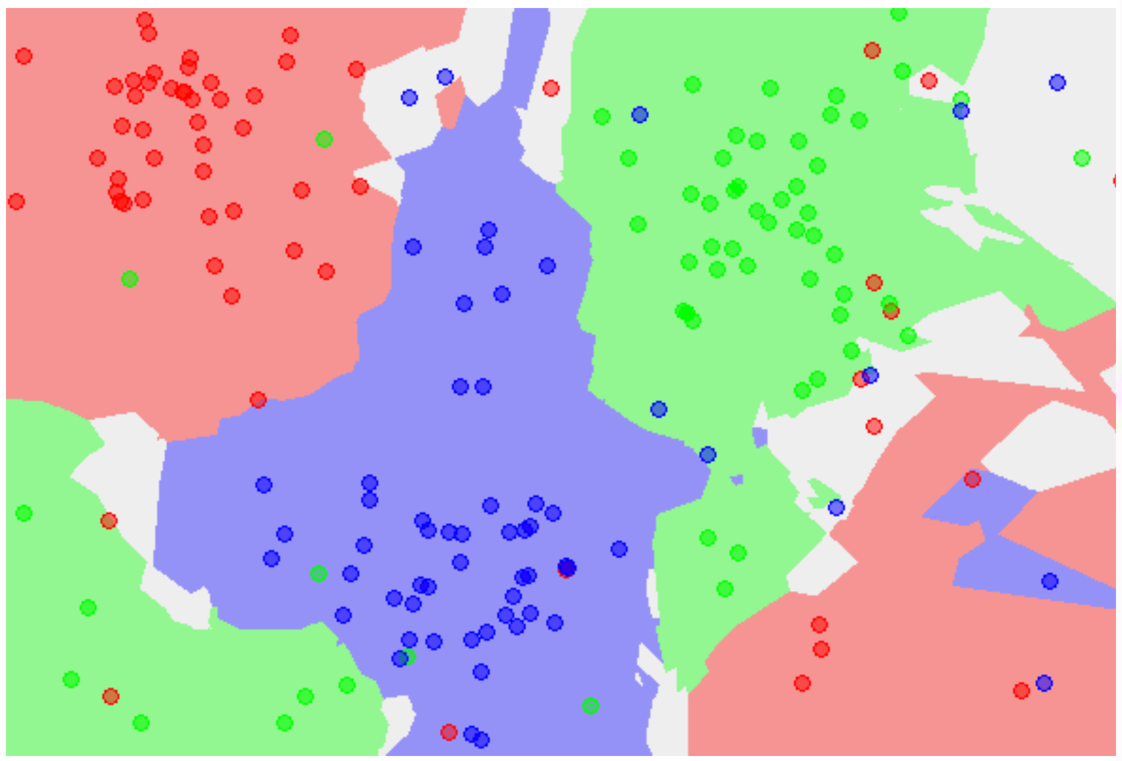
dataset



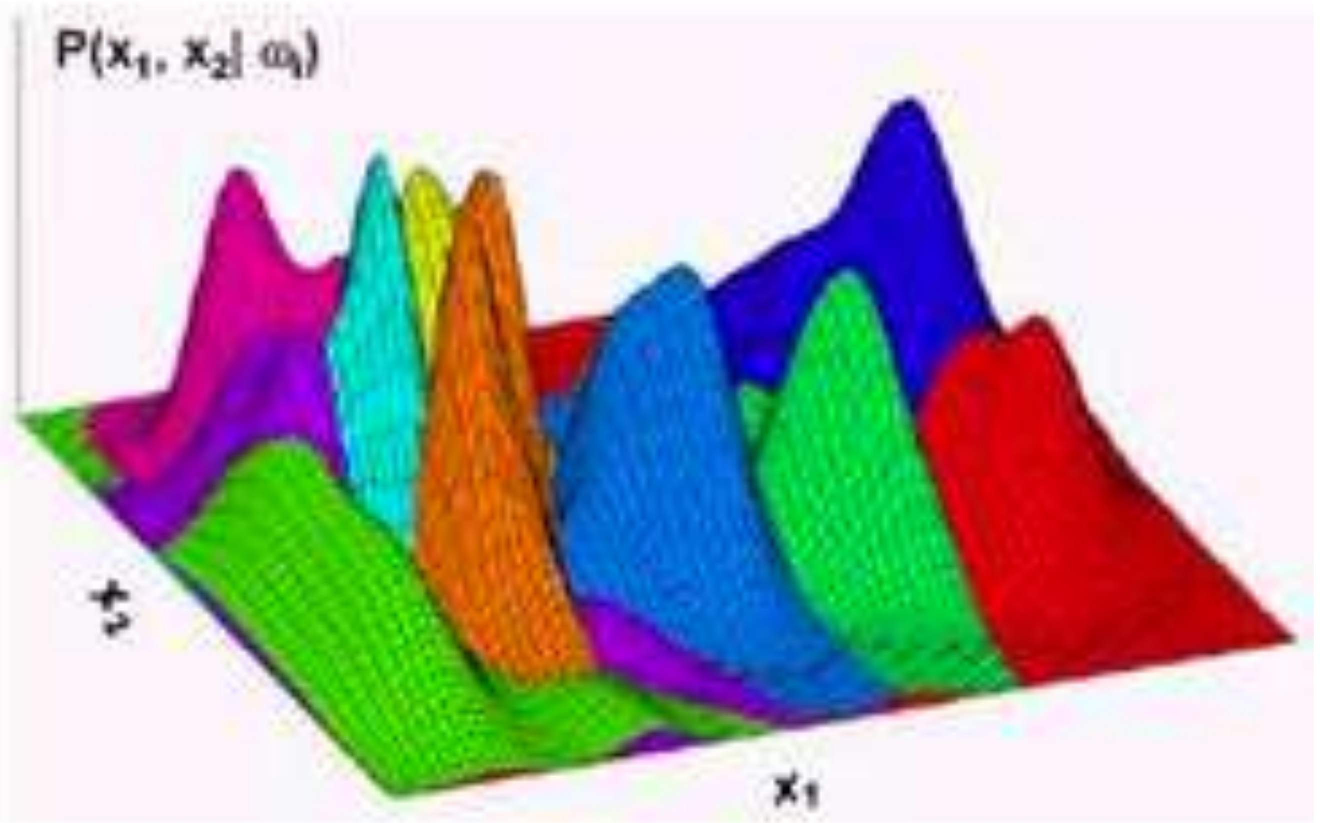
put **on each training data points**

Another way of thinking

- Training data is valuable, try to keep it even till prediction
  - **Memory-based methods**
  - **We would like to utilise the training data for prediction!**
  - **Need to evaluate the distance from "testing" to "training" data**



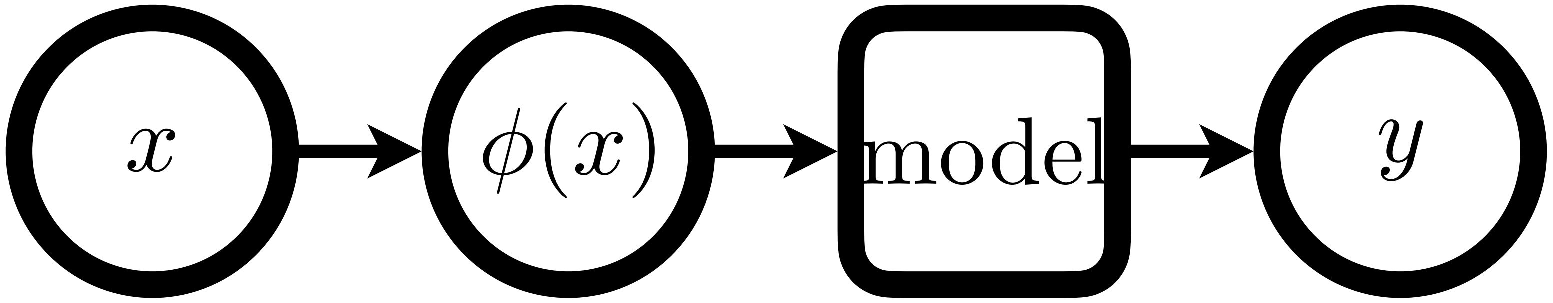
KNN



Parzen probability estimation

Data in the feature space

- General machine learning scheme



  - project x to a feature space by feature mapping ϕ
  - we need to evaluate the distance/similarity between data $\phi(x)$ in the feature space ☞ **dot product** is somehow related to similarity

$$\phi(\mathbf{x})^\top \phi(\mathbf{x}')$$

  - **kernel** function!

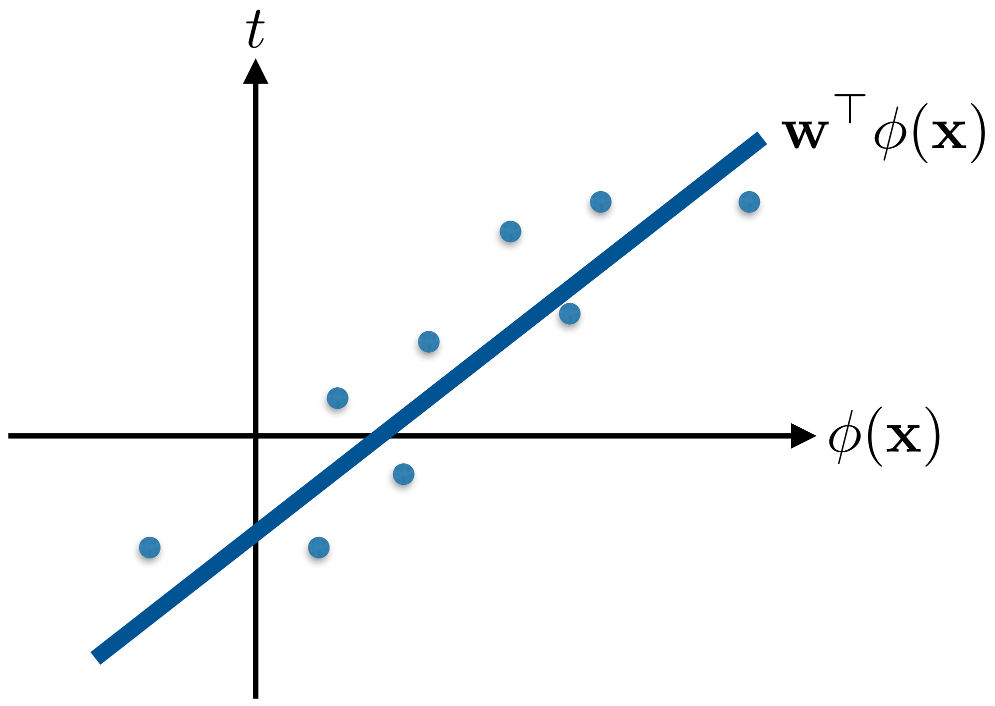
$$\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$$

  - … so what? why do we need this?

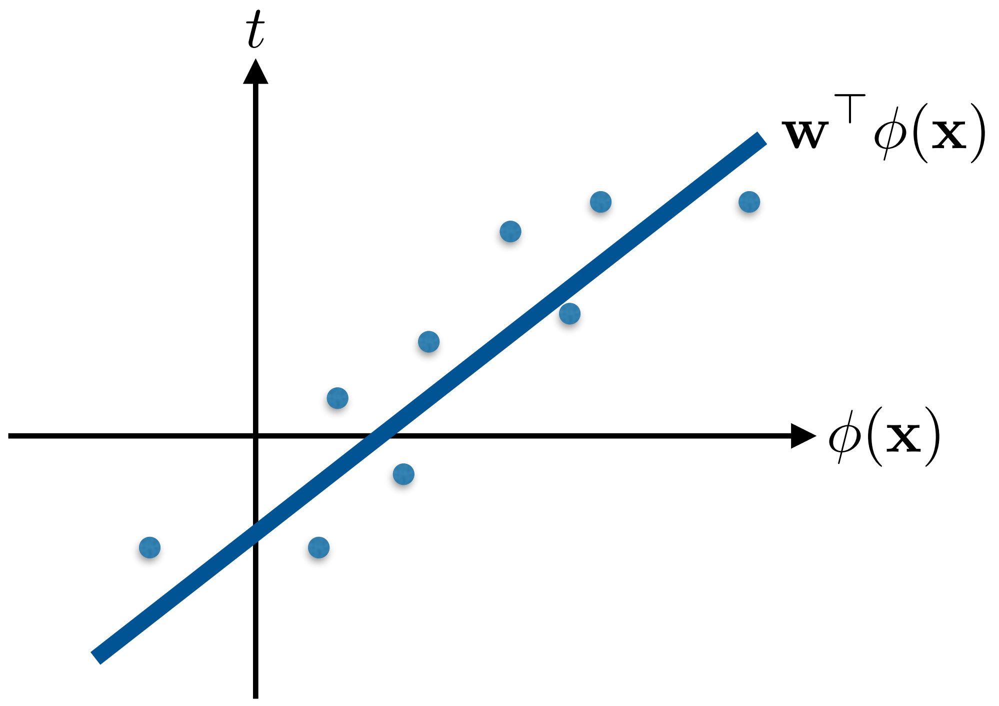
Example for kernel to exist

- Linear regression with L_2 regularisation

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$



Example for kernel to exist



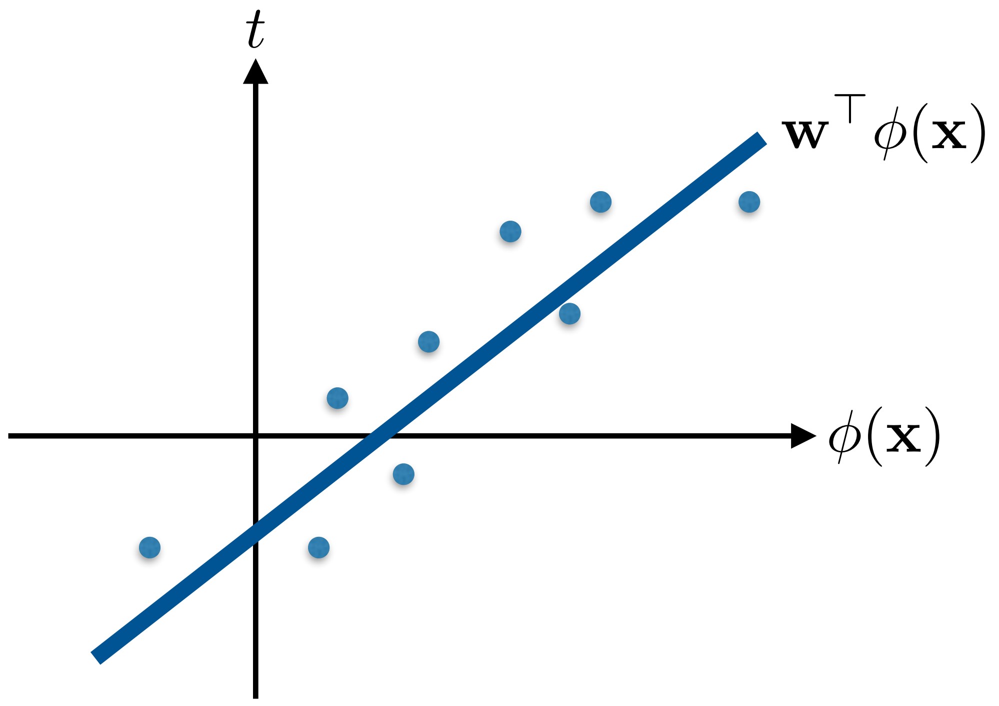
- Linear regression with L_2 regularisation

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

  - with $\frac{\partial J}{\partial \mathbf{w}} = 0$, we can see that $\underline{\mathbf{w} \text{ is the linear combination of } \phi(\mathbf{x})}$

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\underbrace{\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}}_{\text{coefficients}}\boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N}\underline{a_n}\boldsymbol{\phi}(\mathbf{x}_n) = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a}$$

Example for kernel to exist



- Linear regression with *L*₂ regularisation

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

  - with $\frac{\partial J}{\partial \mathbf{w}} = 0$, we can see that $\underline{\mathbf{w} \text{ is the linear combination of } \phi(\mathbf{x})}$

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\underbrace{\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}}_{\text{coefficients}}\boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N}\underline{a_n}\boldsymbol{\phi}(\mathbf{x}_n) = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a}$$

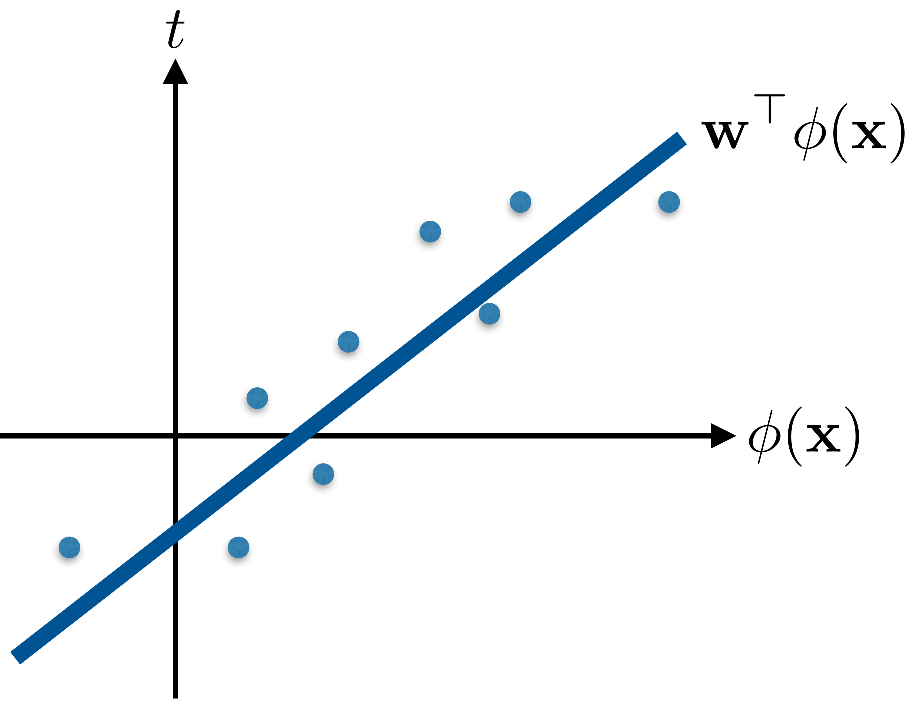
  - put $\mathbf{w} = \Phi^\top\mathbf{a}$ back to J

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{a}$$

helloworld! kernel $\phi(\mathbf{x})^\top\phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$

Example for kernel to exist

- Linear regression with L_2 regularisation



$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

here comes dual form!

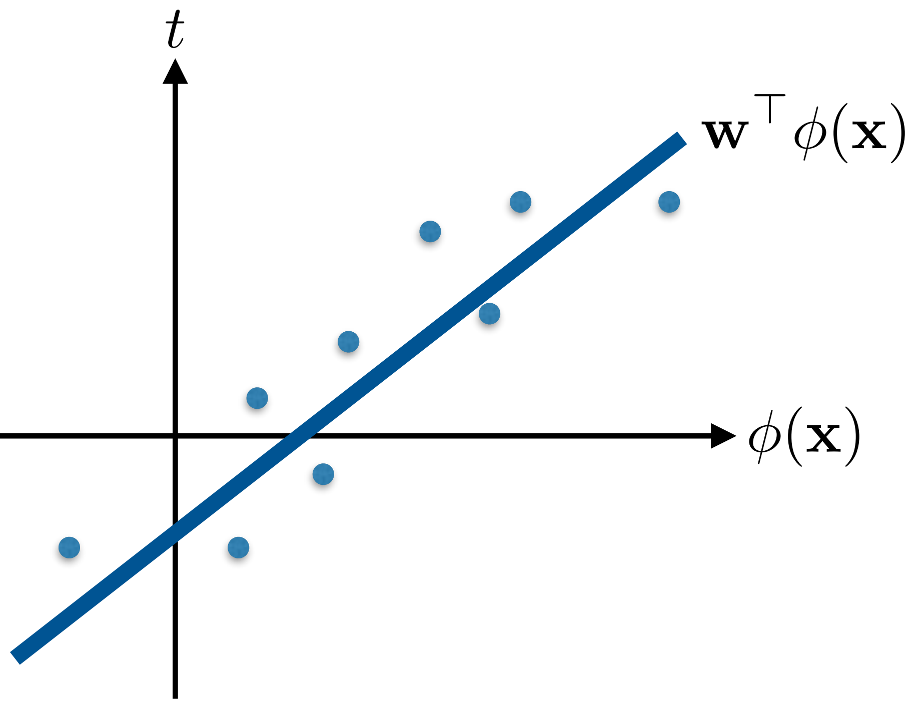
- put $\mathbf{w} = \Phi^\top \mathbf{a}$ back to J

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\underline{\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{a}$$

- $K_{nm} = \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$ Gram matrix: $K = \Phi\Phi^\top$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

Example for kernel to exist



- Linear regression with *L*2 regularisation

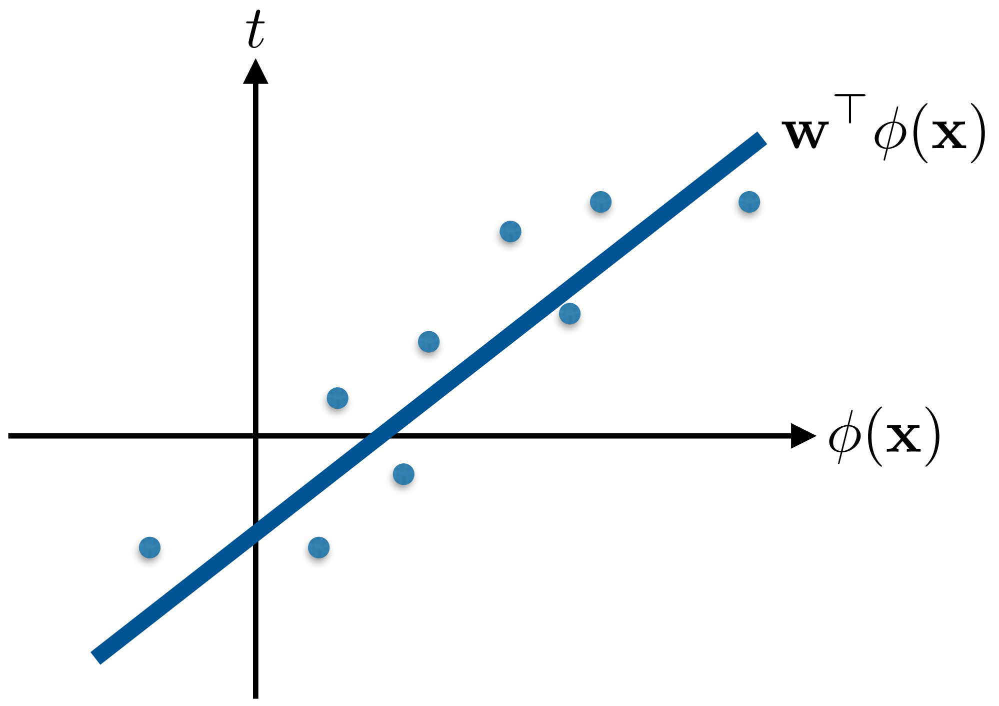
$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

$\updownarrow$ **dual form**

  - $$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$
  - with $\frac{\partial J}{\partial \mathbf{a}} = 0$, get $\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$.
  - when performing prediction:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

where vector $\mathbf{k}(\mathbf{x})$ with elements $k_n(\mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x})$



Example for kernel to exist

- Linear regression with L_2 regularisation

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

dual form

- $$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

- with $\frac{\partial J}{\partial \mathbf{a}} = 0$, get $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{t}.$

- when performing prediction:

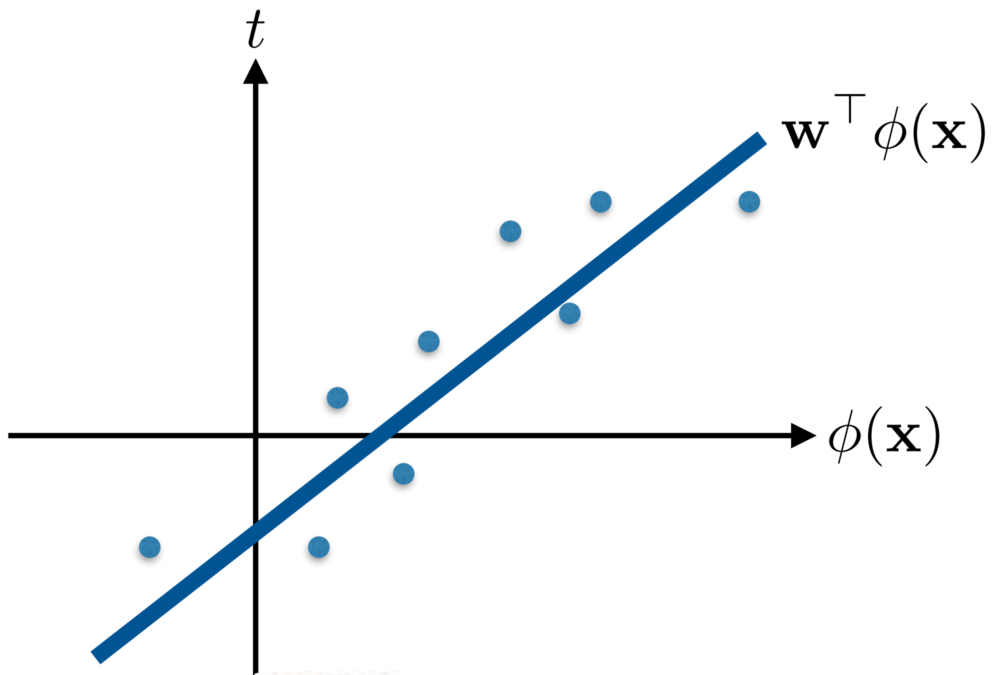
prediction based on linear combination of kernel functions evaluated at training data points

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

where vector $\mathbf{k}(\mathbf{x})$ with elements $k_n(\mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x})$

Example for kernel to exist

- Linear regression with L_2 regularisation



$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

$\updownarrow$ dual form

$$\boxed{J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.}$$

- with $\dfrac{\partial J}{\partial \mathbf{a}} = 0$, get $\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$.

ah-ha!

if x always appears as scalar product,
it might be good to try kernel!
known as "kernel trick"
a.k.a. "kernel substitution"

Feature, feature

- All about features:
  - **projection from original data space to feature space**
  - How to choose a proper feature space?
    - for instance: easier separation between classes (classification)
    - prior knowledge about the class of functions to be learned
    - subsets of a basis of the function space (e.g. Fourier, Wavelet, etc.)

$$\begin{tikzcd} x \arrow[r] & \phi(x) \arrow[r] & \boxed{\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')} \end{tikzcd}$$

Feature, feature

- All about features:
  - **projection from original data space to feature space**
  - How to choose a proper feature space?
    - for instance: easier separation between classes (classification)
    - prior knowledge about the class of functions to be learned
    - subsets of a basis of the function space (e.g. Fourier, Wavelet, etc.)

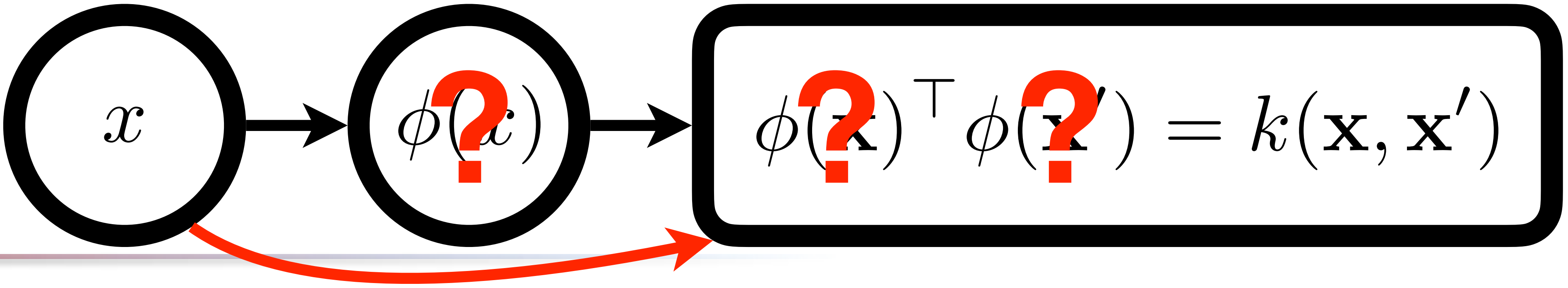
$$\begin{tikzcd} x \arrow[r] & \phi(x) \arrow[r] & \boxed{\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')} \end{tikzcd}$$

- Sometimes we don't know what is the proper feature mapping…
  - still want to use memory-based methods (training data in prediction)
  - perhaps just some kernel? don't care what exactly is the feature map?

$$\begin{tikzcd} x \arrow[r] \arrow[rr, bend right] & \phi(?) \arrow[r] & \boxed{\phi(?)^\top \phi(?') = k(\mathbf{x}, \mathbf{x}')} \end{tikzcd}$$

Some examples of kernel functions

- a very simple form $k(x, x') = e^{xx'}$


x
$\phi(x)$
?
$\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$
?
?

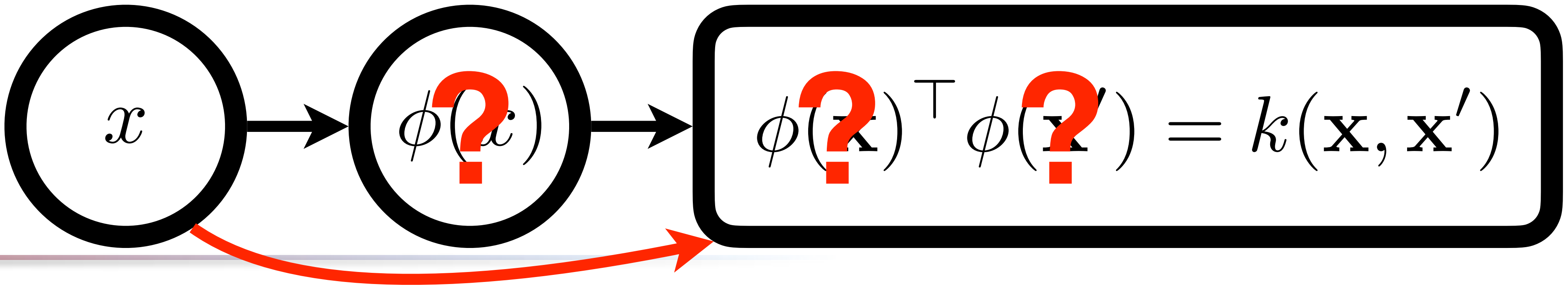
Some examples of kernel functions

- a very simple form $k(x, x') = e^{xx'}$
  - the corresponding feature map is

$$\forall x \in \mathbb{R}, r \in \mathbb{N} : \phi_r(x) = \frac{1}{\sqrt{r!}} x^r$$

$$k(x, x') = \underline{\sum_{r=0}^{\infty} \phi_r(x) \phi_r(x')} = \sum_{r=0}^{\infty} \frac{x^r}{\sqrt{r!}} \frac{(x')^r}{\sqrt{r!}} = \sum_{r=0}^{\infty} \frac{(xx')^r}{r!} = e^{xx'}$$

simple kernel but it has even countably infinitely many feature maps!



x
$\phi(x)$
?
$\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$
?
?

Some examples of kernel functions

- a very simple form $k(x, x') = e^{xx'}$
  - the corresponding feature map is

$$\forall x \in \mathbb{R}, r \in \mathbb{N} : \phi_r(x) = \frac{1}{\sqrt{r!}} x^r$$

$$k(x, x') = \underline{\sum_{r=0}^{\infty} \phi_r(x)\phi_r(x')} = \sum_{r=0}^{\infty} \frac{x^r}{\sqrt{r!}} \frac{(x')^r}{\sqrt{r!}} = \sum_{r=0}^{\infty} \frac{(xx')^r}{r!} = e^{xx'}$$

simple kernel but it has even countably infinitely many feature maps!

PS: simplest kernel is identity mapping

$$\phi(x) = x, \quad k(x, x') = x^\top x'$$

it is called linear kernel.

$$\begin{tikzcd} x \arrow[r] \arrow[rr, bend right] & \phi(\textbf{?}) \arrow[r] & \boxed{\phi(\textbf{?})^\top \phi(\textbf{?}') = k(\mathbf{x}, \mathbf{x}')} \end{tikzcd}$$

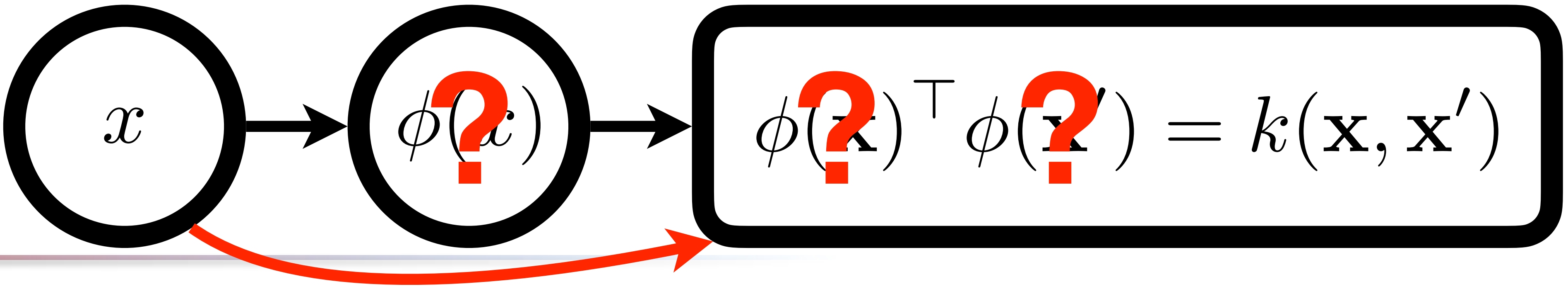
Some examples of kernel functions

- Given $x_i \in \mathbb{R}^3, \phi(x_i) \in \mathbb{R}^{10}$

$$\phi(x_i) = \Big[1, \sqrt{2}(x_i)_1, \sqrt{2}(x_i)_2, \sqrt{2}(x_i)_3, (x_i)_1^2, (x_i)_2^2, (x_i)_3^2, \sqrt{2}(x_i)_1(x_i)_2, \sqrt{2}(x_i)_1(x_i)_3, \sqrt{2}(x_i)_2(x_i)_3\Big]^\top$$

  - kernel function: $\phi(x_i)^\top \phi(x_j) = \underline{(1 + x_i^\top x_j)^2}$

feature map projects 3-D to 10-D, but it results as a very simple form to compute kernel!


x
$\phi(x)$
?
$\phi(\mathbf{x})^\top \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$
?
?

Some examples of kernel functions

- Radial Basis Kernel $e^{-\gamma\|x_i - x_j\|}$
  - assume $x \in \mathbb{R}^1$ and $\gamma > 0$

$$e^{-\gamma\|x_i - x_j\|} = e^{-\gamma(x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots\right)$$

power series definition

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}}x_i \times \sqrt{\frac{2\gamma}{1!}}x_j + \sqrt{\frac{(2\gamma)^2}{2!}}x_i^2 \times \sqrt{\frac{(2\gamma)^2}{2!}}x_j^2 + \cdots\right)$$

$$= \phi(x_i)^\top \phi(x_j)$$

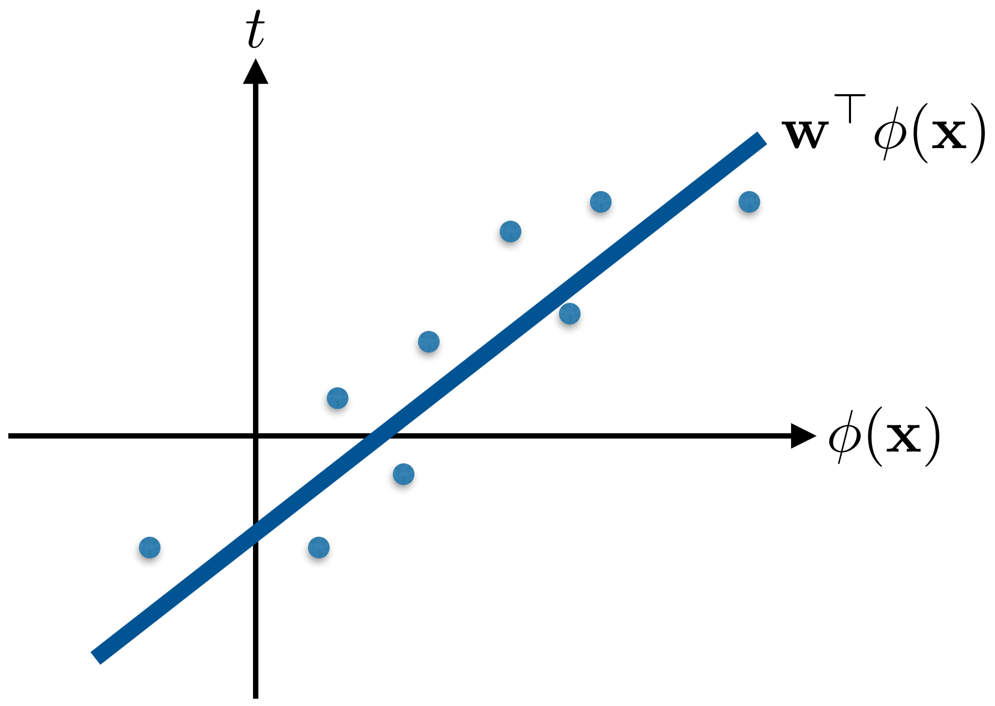
$$\text{where } \phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \cdots\right]^\top$$

simple kernel but infinite-D feature map

$$\begin{tikzcd} x \arrow[r] \arrow[rr, bend right] & \phi(\overset{?}{x}) \arrow[r] & \boxed{\phi(\overset{?}{\mathbf{x}})^\top \phi(\overset{?}{\mathbf{x}'}) = k(\mathbf{x}, \mathbf{x}')} \end{tikzcd}$$

Example for kernel to exist

- Linear regression with L_2 regularisation



$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

$\updownarrow$ **dual form**

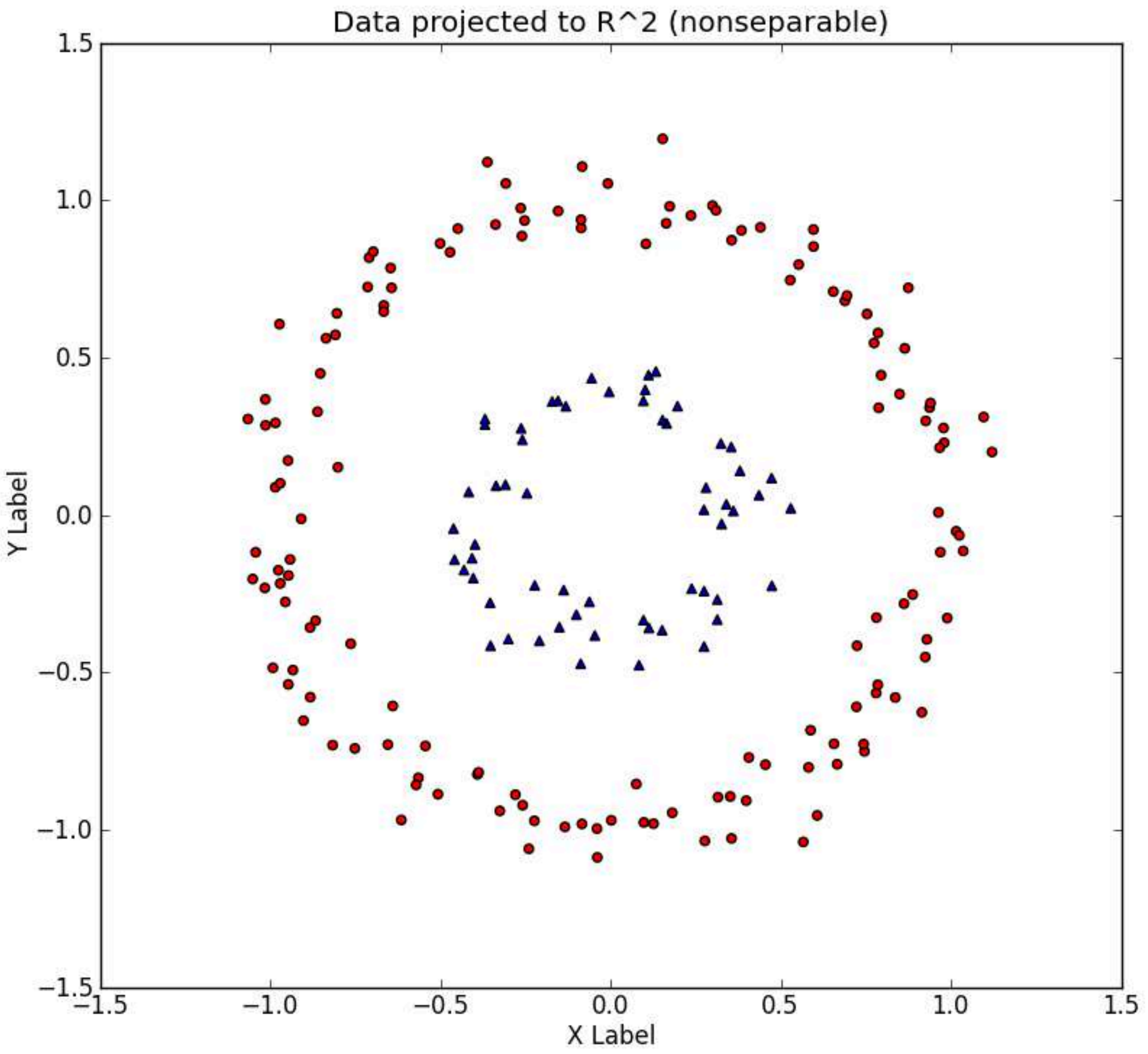
- $$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

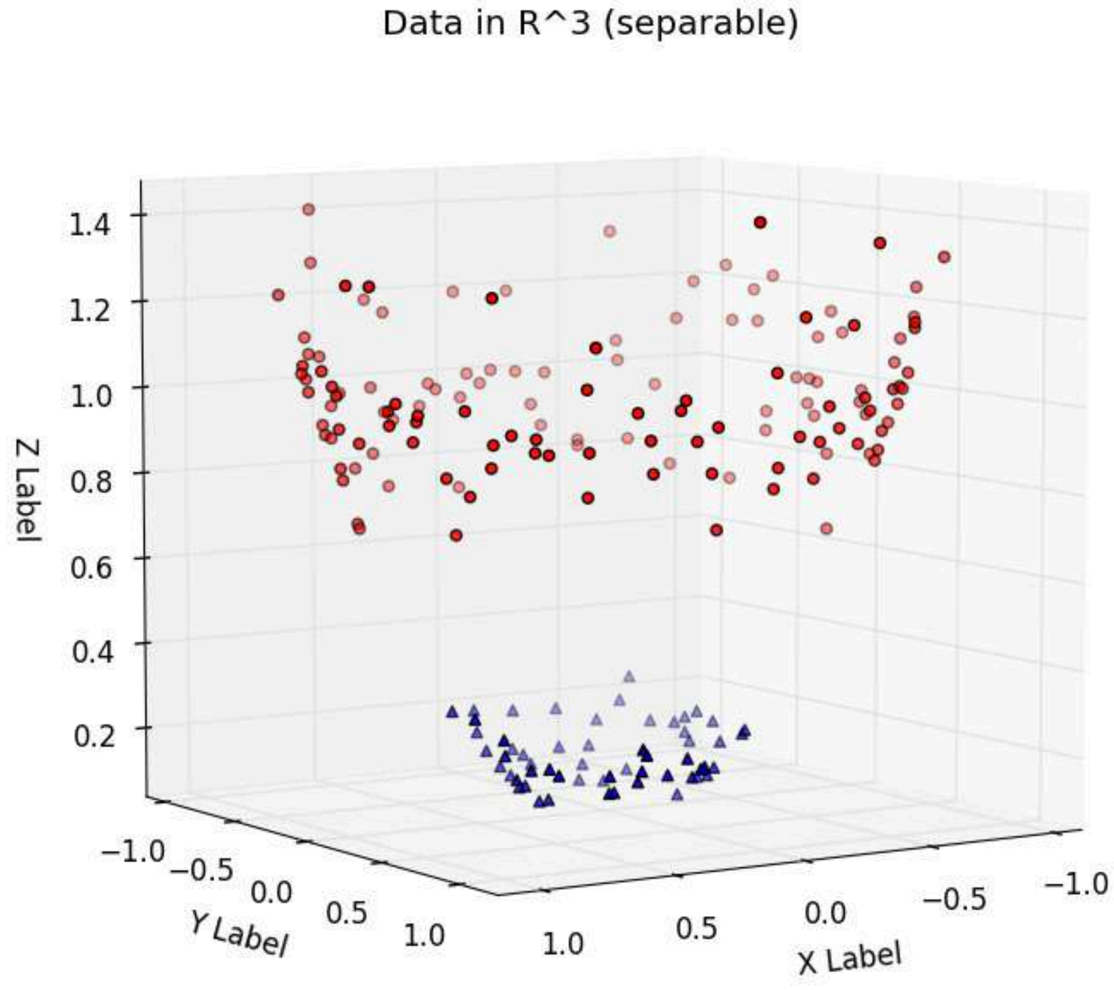
- with $\frac{\partial J}{\partial \mathbf{a}} = 0$, get $\boxed{\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}.}$

ah-ha!

even it seems more complicated to do matrix inversion in higher-D space, but we can directly work on kernels (usually simple) and avoid explicit introduction of feature vector φ(x), which allows us implicitly to use feature spaces of high, even infinite, dimensionality!

Why High Dimensional Feature Space





How to construct valid kernel?

- a necessary and sufficient condition:
  the $\text{Gram matrix: } K = \Phi\Phi^\top$ whose element are given by $k(x,x') = \phi(x)^\top\phi(x')$ is positive semidefinite

A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **positive** $\overbrace{\quad}^{\textbf{semi}}$ **definite kernel** if, for any finite J, any $x \in \mathcal{X}^J$ and any $c \in \mathbb{R}^J$:

$$0 \leq \sum_{i \in J} \sum_{j \in J} c_i c_j k(x_i, x_j)$$

The set of all real-valued positive definite kernels on $\mathcal{X}$ is denoted $\mathbb{R}_+^{\mathcal{X} \times \mathcal{X}}$

credit: Bjoern Andres@MPI-INF 27

How to construct valid kernel?

- a necessary and sufficient condition:
  the Gram matrix: $K = \Phi\Phi^\top$ whose element are given by $k(x, x') = \phi(x)^\top \phi(x')$ is positive semidefinite

- You can build kernel from kernel

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

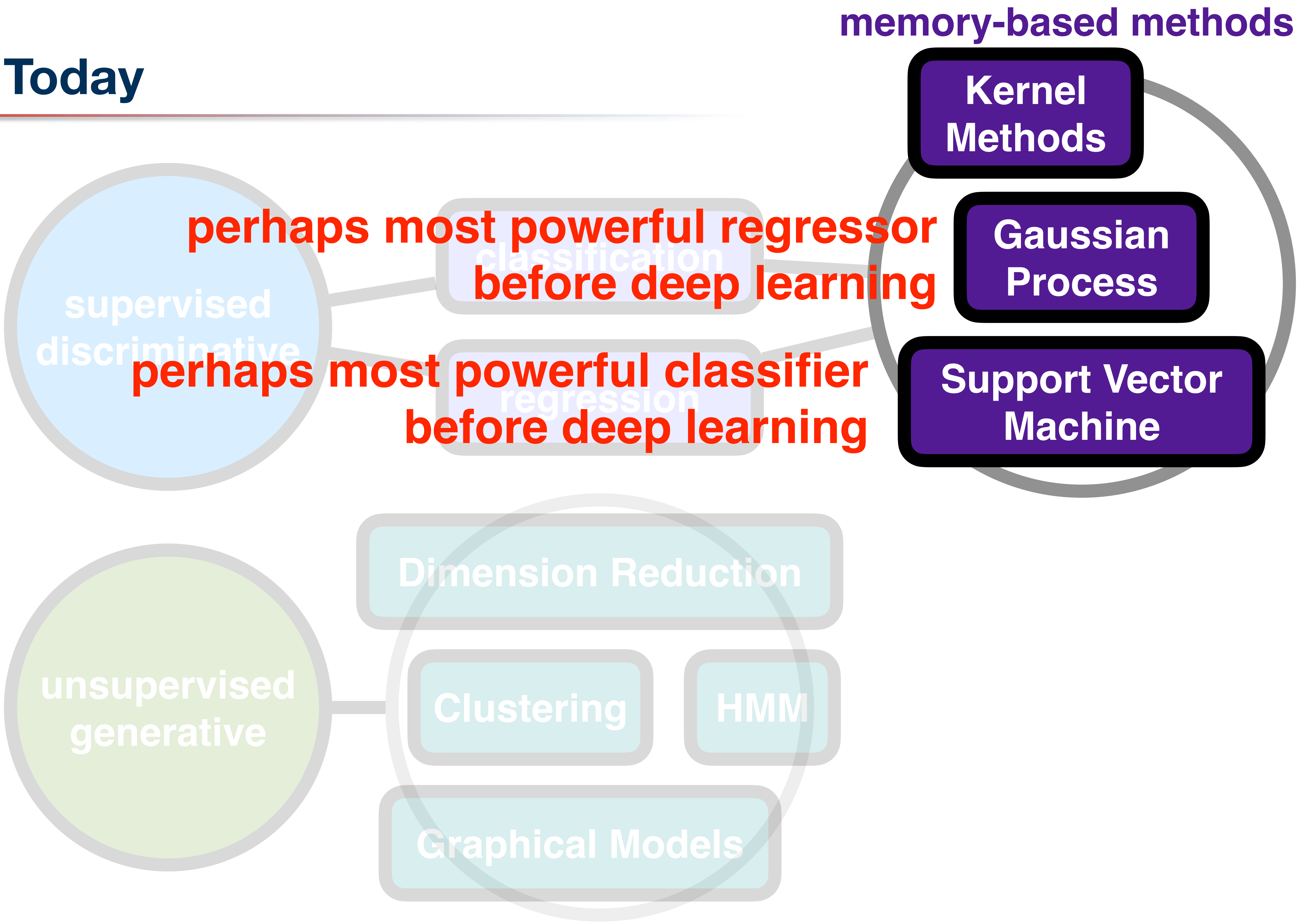
Recap!

- memory-based methods: we want to keep training data points for further use in the prediction

- need to have a metric to evaluate the distance/similarity between data points in the feature space: inner product ☞ kernel!
  - scalar product in $\boldsymbol{x}$ brings possibility to have kernel trick!

- kernel provide a way to compute inner product in feature space
  - simple kernel computation can be related to complicated feature map!
  - the kernel function itself is more important than the feature map (for memory-based methods), we don't even need to know how to compute feature map!
  - usually we imagine kernel is providing us a way to project the data into much higher dimensional space
  - we can easily build kernel from kernel!

Today


memory-based methods
Kernel Methods
Gaussian Process
Support Vector Machine
perhaps most powerful regressor before deep learning
perhaps most powerful classifier before deep learning
supervised discriminative
classification
regression
unsupervised generative
Dimension Reduction
Clustering
HMM
Graphical Models

Nonparametric Regression

- Previously on linear parametric regression

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{x})$$

  - prior distribution over $\mathbf{w}$, e.g., isotropic Gaussian $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$, induces a corresponding distribution over functions $y(\mathbf{x}, \mathbf{w})$
  - while given training samples with specific values $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, we can get random variables $y(\mathbf{x}_1), y(\mathbf{x}_2), \cdots, y(\mathbf{x}_N)$
  - the joint distribution $\mathbf{y} = \{y(\mathbf{x}_1), y(\mathbf{x}_2), \cdots, y(\mathbf{x}_N)\}$ is also Gaussian!

$$\mathbb{E}[\mathbf{y}] = \Phi\mathbb{E}[\mathbf{w}] = 0$$

$$\mathrm{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^\top] = \Phi\mathbb{E}[\mathbf{w}\mathbf{w}^\top]\Phi^\top = \frac{1}{\alpha}\Phi\Phi^\top = \mathbf{K}$$

Nonparametric Regression

- Previously on linear parametric regression

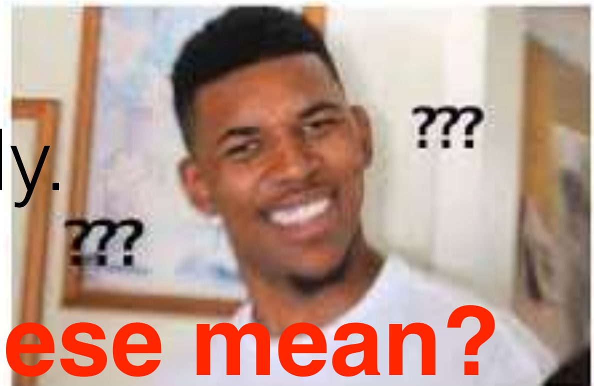
$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{x})$$

  - prior distribution over $\mathbf{w}$, e.g., isotropic Gaussian $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$, induces a corresponding distribution over functions $y(\mathbf{x}, \mathbf{w})$
  - while given training samples with specific values $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, we can get random variables $y(\mathbf{x}_1), y(\mathbf{x}_2), \cdots, y(\mathbf{x}_N)$
  - the joint distribution $\mathbf{y} = \{y(\mathbf{x}_1), y(\mathbf{x}_2), \cdots, y(\mathbf{x}_N)\}$ is also Gaussian!

$$\mathbb{E}[\mathbf{y}] = \Phi\mathbb{E}[\mathbf{w}] = 0$$

$$\mathrm{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^\top] = \Phi\mathbb{E}[\mathbf{w}\mathbf{w}^\top]\Phi^\top = \frac{1}{\alpha}\Phi\Phi^\top = \mathbf{K}$$

- Now we are going to introduce "**Gaussian Process**", which doesn't use parametric model but instead define a prior probability **distribution over functions** directly.

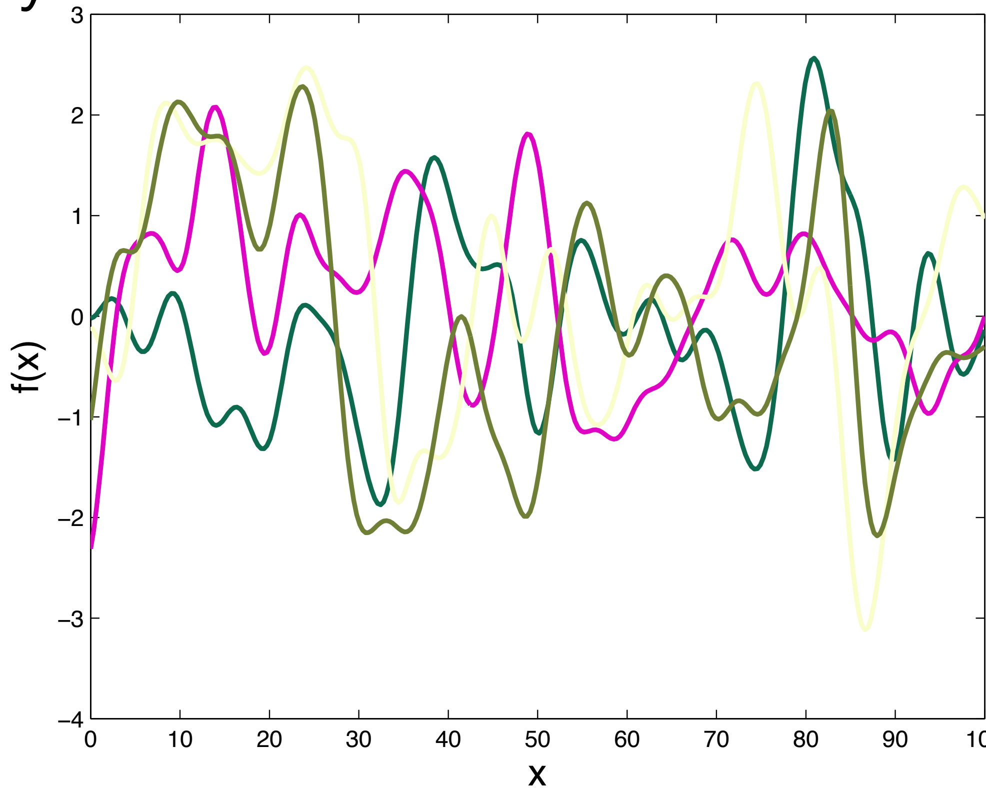


what do these mean?

Gaussian Process

For any set $\mathbf{S}$, a Gaussian Process on $\mathbf{S}$ is a set of random variables $(f(x), x \in \mathbf{S})$, so for any $n \in \mathbb{N}$ and $x_1, \cdots, x_n \in \mathbf{S}$, $\{f(x_1), \cdots, f(x_n)\}$ is (multivariate) Gaussian

- the number of elements in set $\mathbf{S}$ can be infinite many.
- the number of random variables $f(x)$ can be any
  - $\{f(x_9), f(x_5), f(x_2), f(x_7)\}$ can build a mean and a covariance function, Gaussian!
  - $\{f(x_7), f(x_8)\}$ can build another mean and another covariance function, Gaussian!



Gaussian Process

For any set $\mathbf{S}$, a Gaussian Process on $\mathbf{S}$ is a set of random variables $(f(x), x \in \mathbf{S})$, so for any $n \in \mathbb{N}$ and $x_1, \cdots, x_n \in \mathbf{S}$, $\{f(x_1), \cdots, f(x_n)\}$ is (multivariate) Gaussian

- the number of elements in set $\mathbf{S}$ can be infinite many.
- the number of random variables $f(x)$ can be any
  - $\{f(x_9), f(x_5), f(x_2), f(x_7)\}$ can build a mean and a covariance function, Gaussian!
  - $\{f(x_7), f(x_8)\}$ can build another mean and another covariance function, Gaussian!
- Gaussian Processes (GPs) are parameterised by a mean function $\mu(x)$, and a covariance function, or ***kernel***, $K(x, x')$

$$\mathrm{cov}(f_n, f_m) = \langle f_n - \mu(f_n), f_m - \mu(f_m) \rangle = \langle f_n, f_m \rangle - \mu(f_n)\mu(f_m)$$

Gaussian Process

For any set $\mathbf{S}$, a Gaussian Process on $\mathbf{S}$ is a set of random variables $(f(x), x \in \mathbf{S})$, so for any $n \in \mathbb{N}$ and $x_1, \cdots, x_n \in \mathbf{S}, \{f(x_1), \cdots, f(x_n)\}$ is (multivariate) Gaussian

- the number of elements in set $\mathbf{S}$ can be infinite many.
- the number of random variables $f(x)$ can be any
  - $\{f(x_9), f(x_5), f(x_2), f(x_7)\}$ can build a mean and a covariance function, Gaussian!
  - $\{f(x_7), f(x_8)\}$ can build another mean and another covariance function, Gaussian!
- Gaussian Processes (GPs) are parameterised by a mean function $\mu(x)$, and a covariance function, or ***kernel***, $K(x, x')$

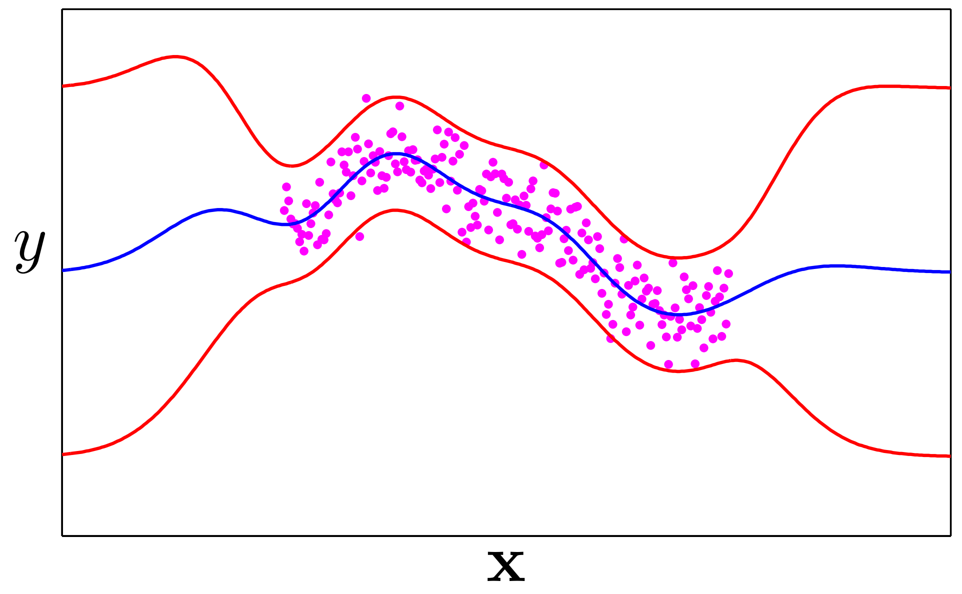
[any possible collection of random variables (from training data) creates a function]

☞ **Gaussian process is a distribution over functions**

Gaussian Process Regression

- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{N}\} = (\mathbf{X}, \mathbf{y})$
  - regression in not professional statement:
    given a bunch of training data, and predict new y for a test input x

- We want to learn a function f with error bars from data $\mathcal{D}$



Gaussian process

$$y_n = \boxed{f}(\mathbf{x}_n) + \epsilon_n$$

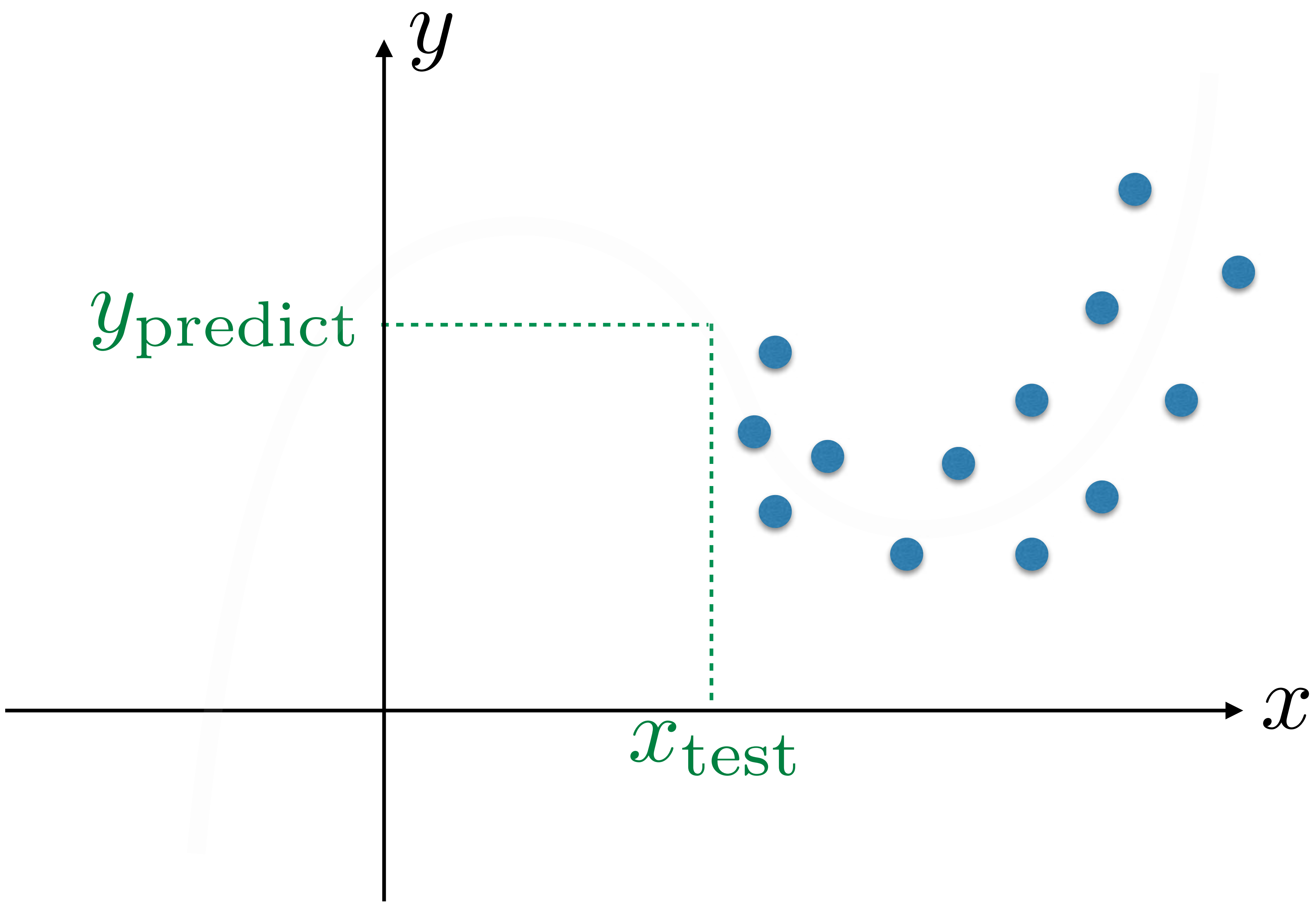
$$\underline{\epsilon_n \sim \mathcal{N}(\cdot|0, \beta^{-1})}$$

Gaussian Process Regression

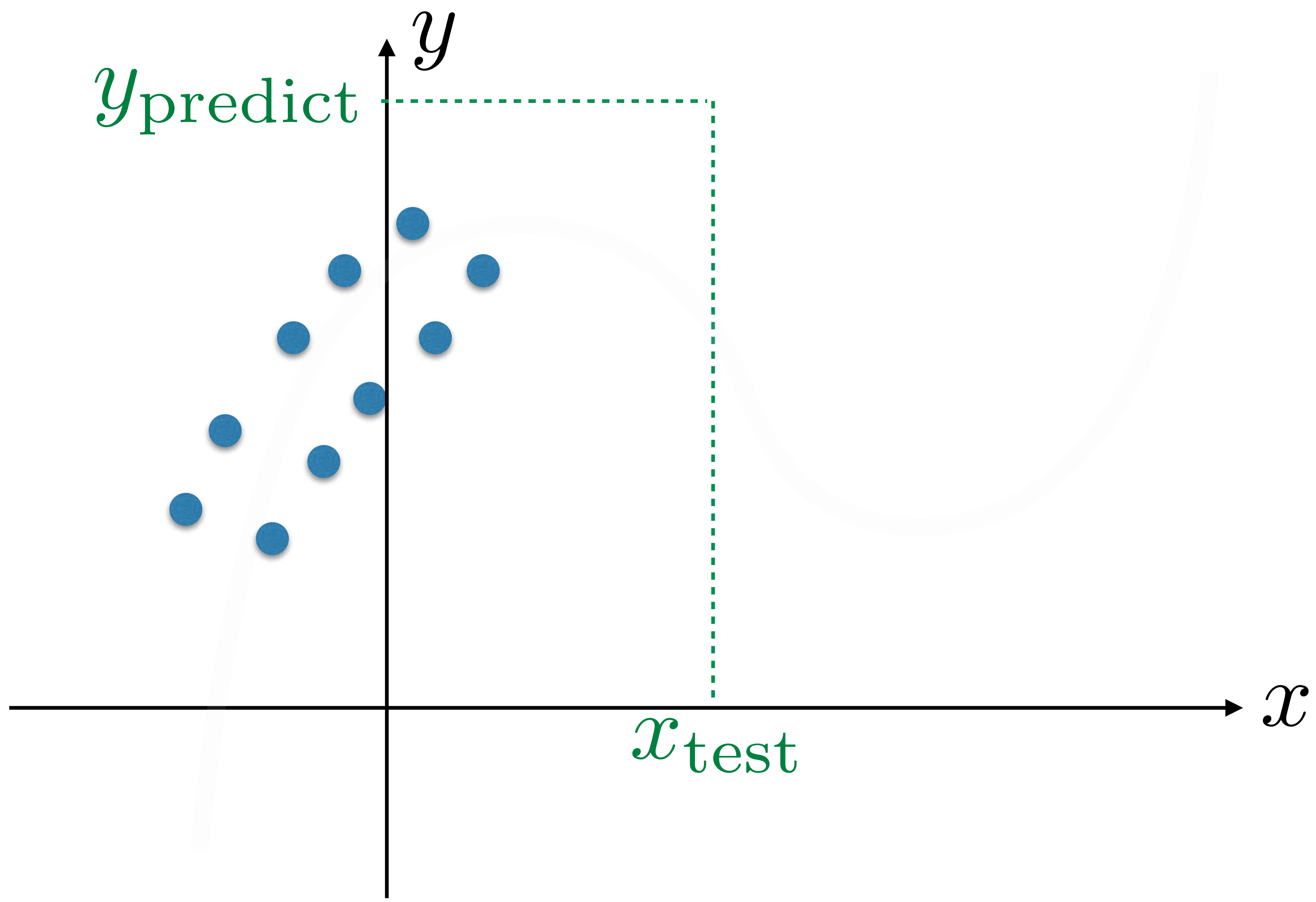
- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{n}\} = (X, \mathbf{y})$
  - regression in not professional statement:
    given a bunch of training data, and predict new y for a test input $\mathbf{x}$

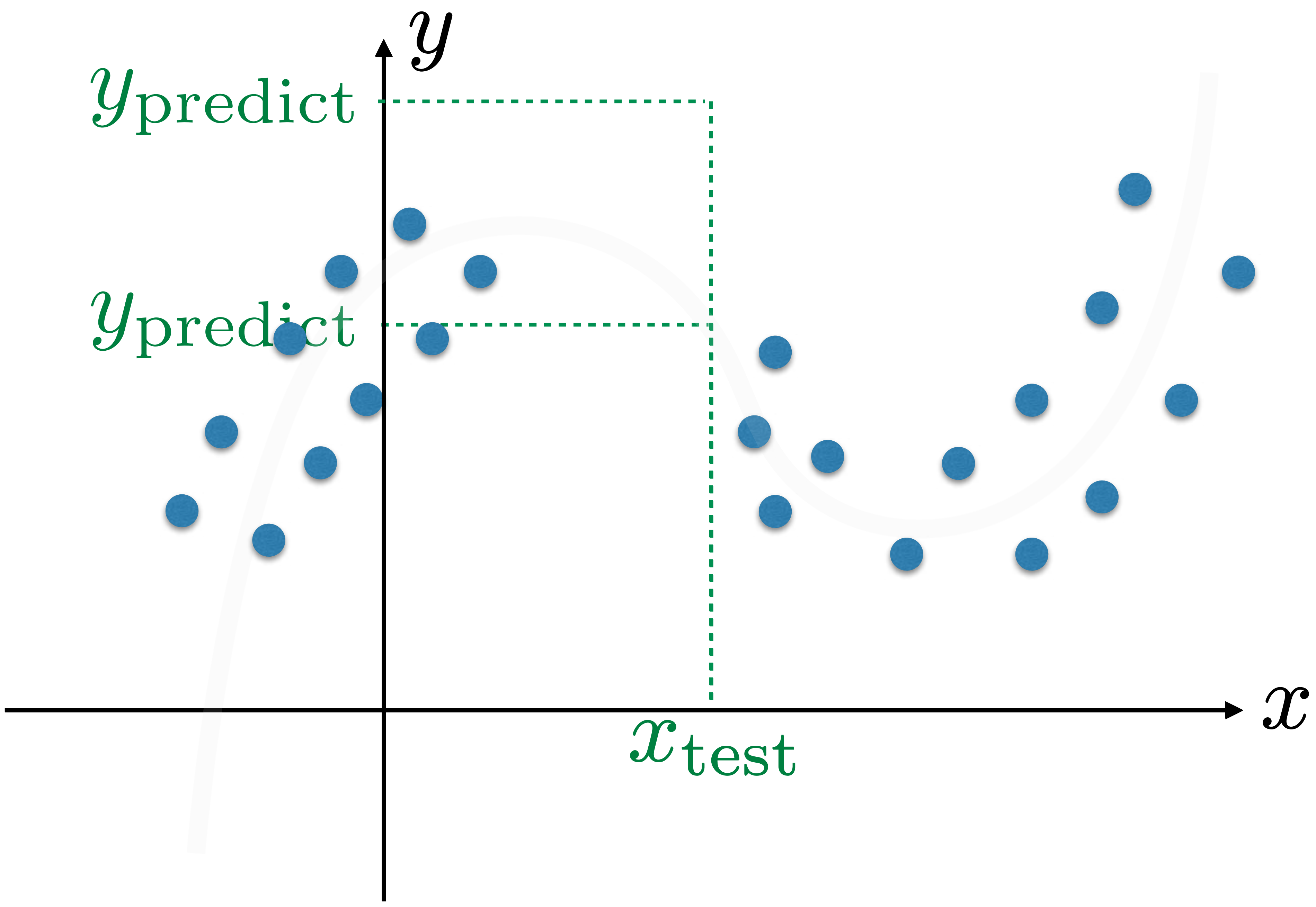
- Very basic ideas help you to understand GP regression:
  - It is a memory-based method!!!
  - Prediction is based on the relation between test input and training data!
  - Intuition: if $\mathbf{x}$ and $\mathbf{x}'$ are close to each other (in feature space), then their y will be also close

Gaussian Process Regression



Gaussian Process Regression





y
y_{predict}
y_{predict}
x
x_{test}

Gaussian Process Regression

- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{n}\} = (X, \mathbf{y})$
  - regression in not professional statement:
    given a bunch of training data, and predict new y for a test input $\mathbf{x}$

- Very basic ideas help you to understand GP regression:
  - It is a memory-based method!!!
  - Prediction is based on the relation between test input and training data!
  - Intuition: if $\mathbf{x}$ and $\mathbf{x}'$ are close to each other (in feature space), then their y will be also close
  - metric to evaluate the similarity/distance between $\mathbf{x}$ ☞ kernel!
  - GP is parameterised by a mean function $\mu(\mathbf{x})$, and a covariance function, or kernel, $K(\mathbf{x}, \mathbf{x}')$.
  - Since random variable $y_n = f(\mathbf{x}_n)$, we can compute the similarity between random variables by covariance

Gaussian Process Regression

- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{n}\} = (X, \mathbf{y})$
  - regression in not professional statement:
  - given a bunch of training data, and predict new y for a test input $\mathbf{x}$

The kernel function that determines K should be chosen to express the property that:
for points x_n and x_m that are similar,
the corresponding values $y(x_n)$ and $y(x_m)$ will be more strongly correlated than for dissimilar points.

- Very basic ideas help you to understand GP regression:
  - [illegible] based [illegible]!
  - [illegible] is based on the relation between test input and training data!
  - $\boxed{\text{Intuition: if } \mathbf{x} \text{ and } \mathbf{x}' \text{ are close to each other (\textbf{in feature space}), then their } y \text{ will be also close}}$
  - metric to evaluate the similarity/distance between $\mathbf{x}$ ☞ kernel!
  - GP is parameterised by a mean function $\mu(\mathbf{x})$, and a <u>covariance</u> function, or <u>kernel, $K(\mathbf{x}, \mathbf{x}')$</u>.
  - Since random variable $y_n = f(\mathbf{x}_n)$, we can compute the similarity between random variables by <u>covariance</u>

ah-ha!

Gaussian Process Regression

- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{n}\} = (X, \mathbf{y})$
  - regression in not professional statement:
    given a bunch of training data, and predict new y for a test input $\mathbf{x}$

- Very basic ideas help you to understand GP regression:

$$\boxed{y_n = f(\mathbf{x}_n) + \epsilon_n}$$
$$\epsilon_n \sim \mathcal{N}(\cdot|0, \beta^{-1})$$

We don't have any parametric model like $y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$
Gaussian process regression is nonparametric!

Gaussian Process Regression

- Imagine observing a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^{n}\} = (X, \mathbf{y})$
  - regression in not professional statement:
    given a bunch of training data, and predict new y for a test input $\mathbf{x}$

- Very basic ideas help you to understand GP regression:

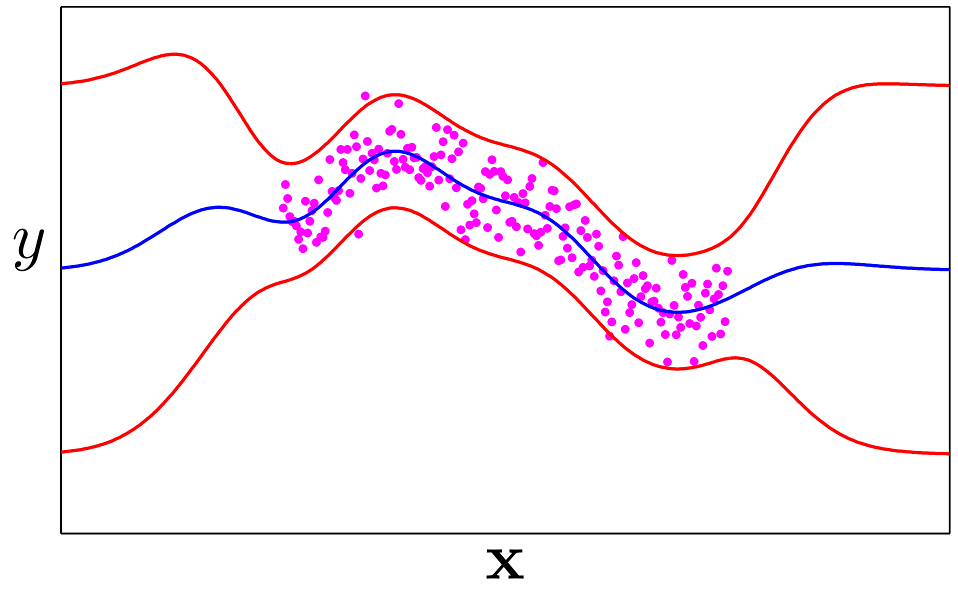
$$\boxed{y_n = f(\mathbf{x}_n) + \epsilon_n}$$
$$\epsilon_n \sim \mathcal{N}(\cdot|0, \beta^{-1})$$

We don't have any parametric model like $y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$
Gaussian process regression is nonparametric!

ah-ha!

"nonparametric" doesn't mean there is no parameters, instead it means: the number of parameters will grows with number of data!

Gaussian Process Regression



$$y_n = f(\mathbf{x}_n) + \epsilon_n$$
$$\epsilon_n \sim \mathcal{N}(\cdot|0, \beta^{-1})$$

for $\mathbf{f} = [f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N)]^\top$ and $\mathbf{y} = [y_1, \cdots, y_N]^\top$

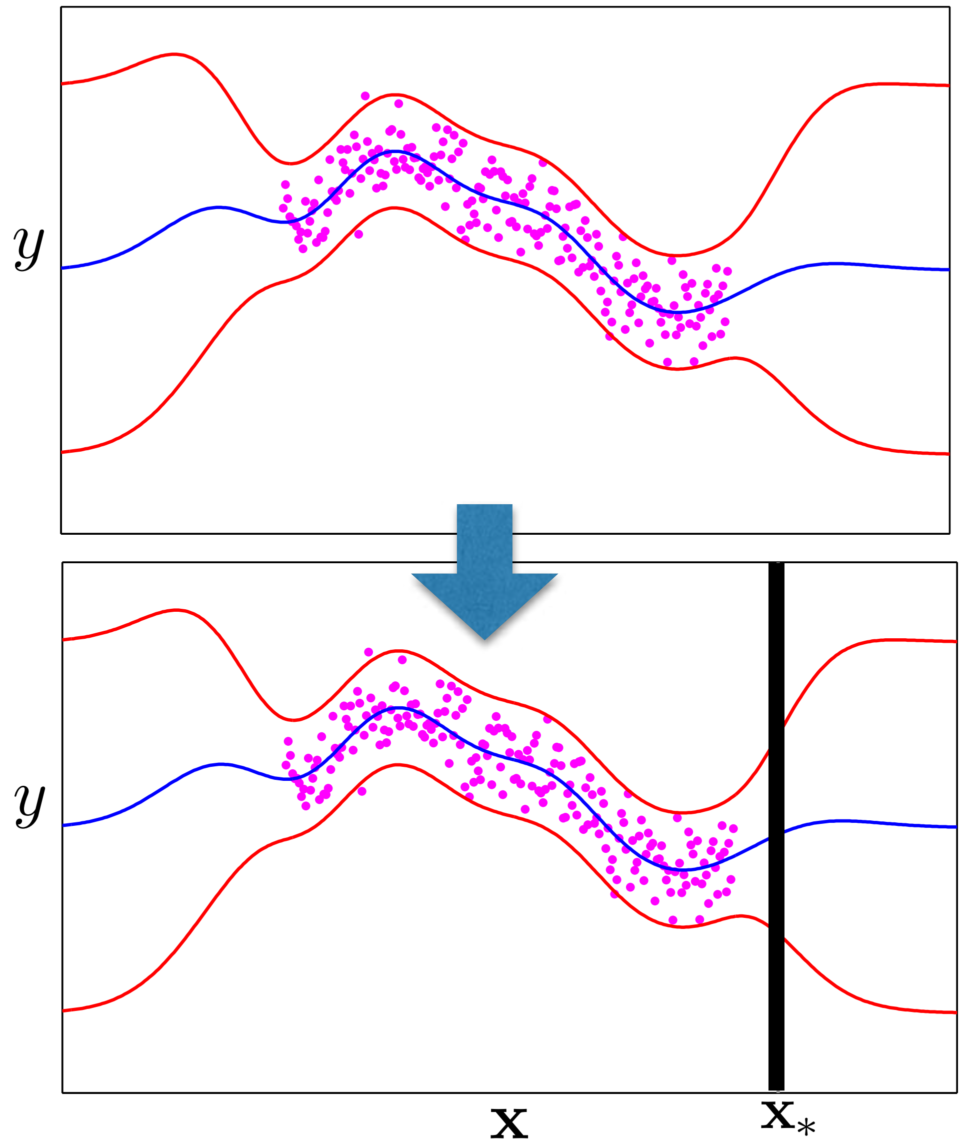
$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \beta^{-1}\mathbf{I}_N)$$
$$p(\mathbf{f}) = \mathcal{N}(\mathbf{0}, \mathbf{K})$$

marginal likelihood $p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\mathrm{d}\mathbf{f} = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C})$

where the covariance matrix $\mathbf{C}$ has elements

$$\mathbf{C}(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

Gaussian Process Regression



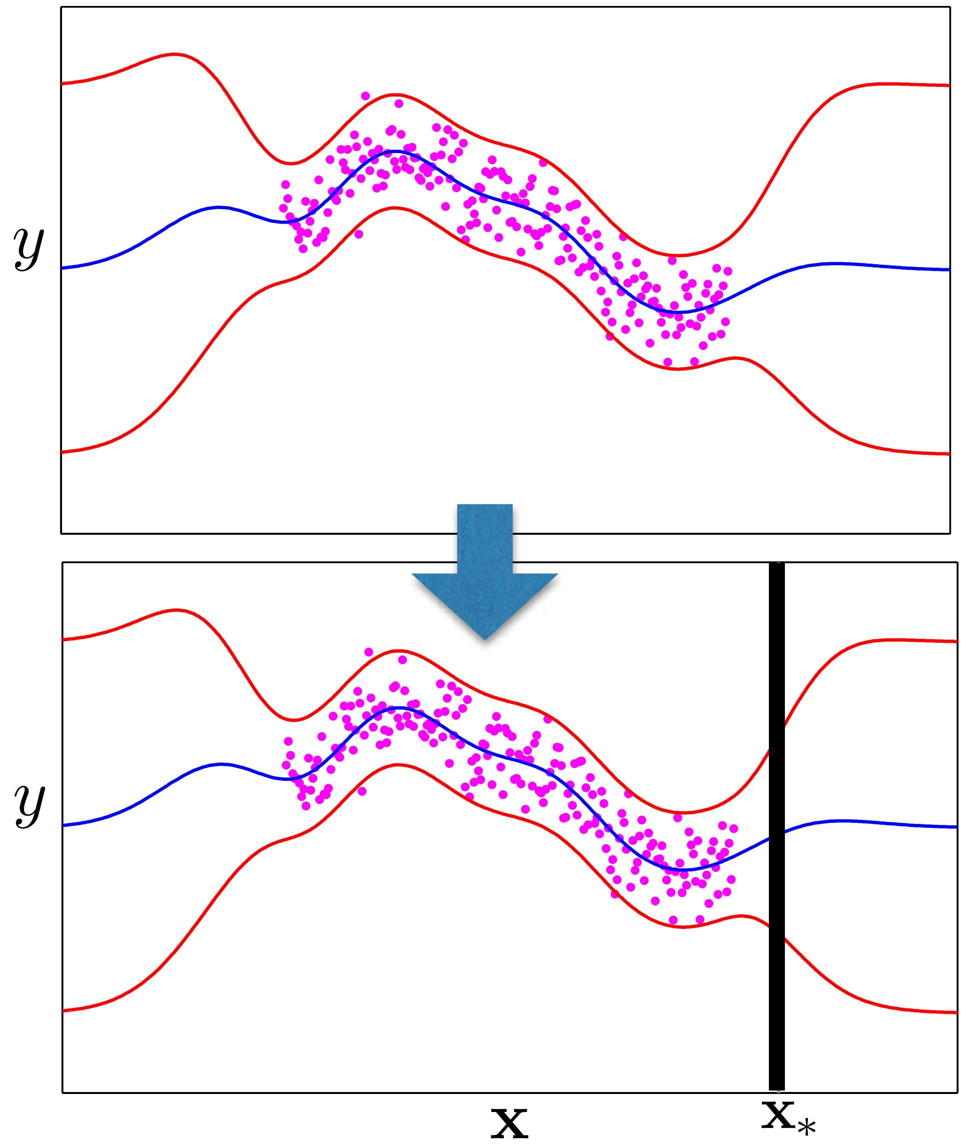
marginal likelihood

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\mathrm{d}\mathbf{f} = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C})$$

$$\mathbf{C}(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

prediction?

Gaussian Process Regression



marginal likelihood

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\mathrm{d}\mathbf{f} = \mathcal{N}(\mathbf{y}|\mathbf{0},\mathbf{C})$$

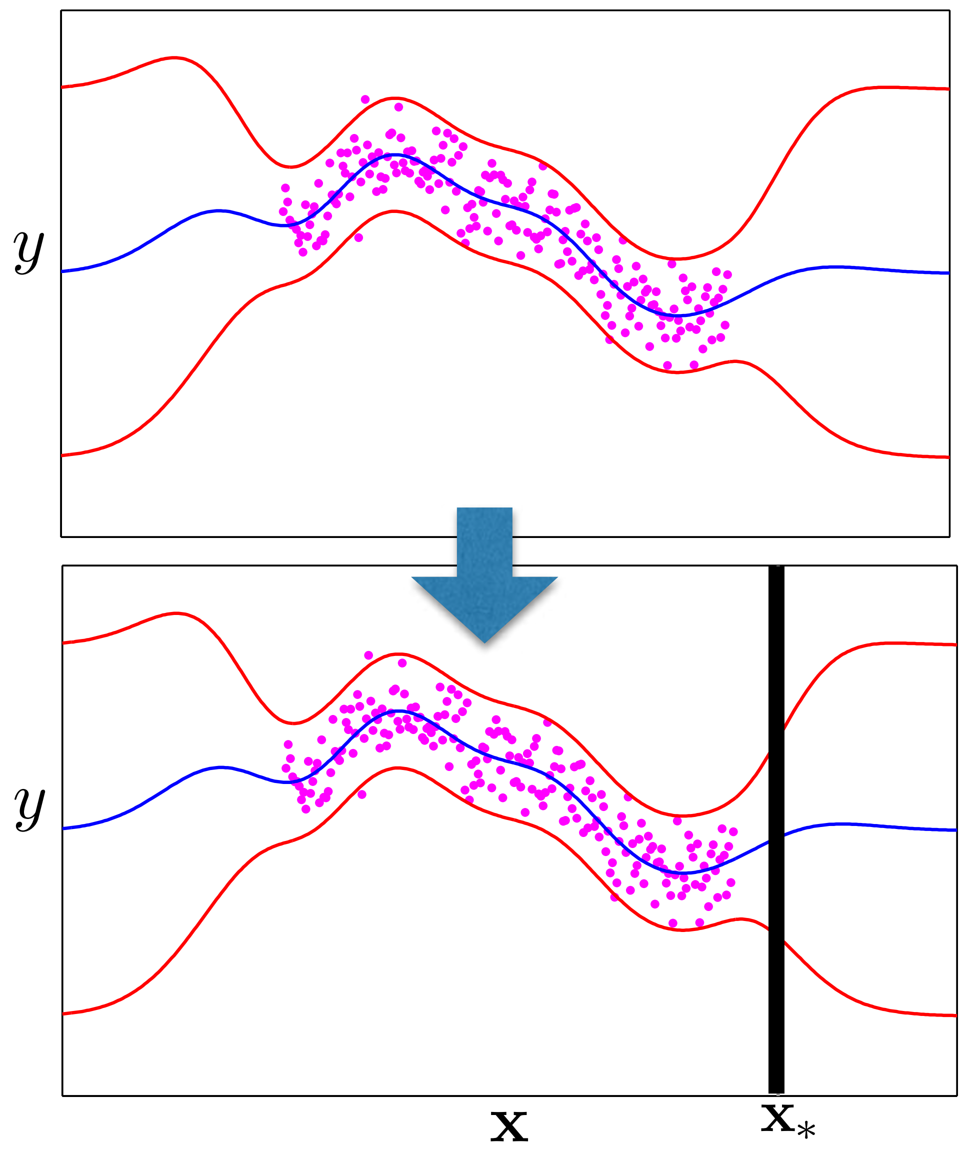
$$\mathbf{C}(\mathbf{x}_n,\mathbf{x}_m) = k(\mathbf{x}_n,\mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

prediction?

ah-ha!

Because we know that all ***f*** (training+testing) together become a multivariate Gaussian distribution ***G***, therefore if we want to predict the distribution of new ***f****, we just need to compute the covariance matrix of ***G***, then cut ***G*** on ***f**** to see the conditional distribution thus achieve prediction :D

Gaussian Process Regression



marginal likelihood

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\mathrm{d}\mathbf{f} = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C})$$

$$\mathbf{C}(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

prediction

denote $\mathbf{y}_{N+1} = [\mathbf{y}, y^*]^\top$ and $y^* = f(\mathbf{x}^*)$

$$p(\mathbf{y}_{N+1}) = \mathcal{N}(\mathbf{y}_{N+1}, |\mathbf{0}, \mathbf{C}_{N+1})$$

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C} & k(\mathbf{x}, \mathbf{x}^*) \\ k(\mathbf{x}, \mathbf{x}^*)^\top & k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1} \end{bmatrix}$$

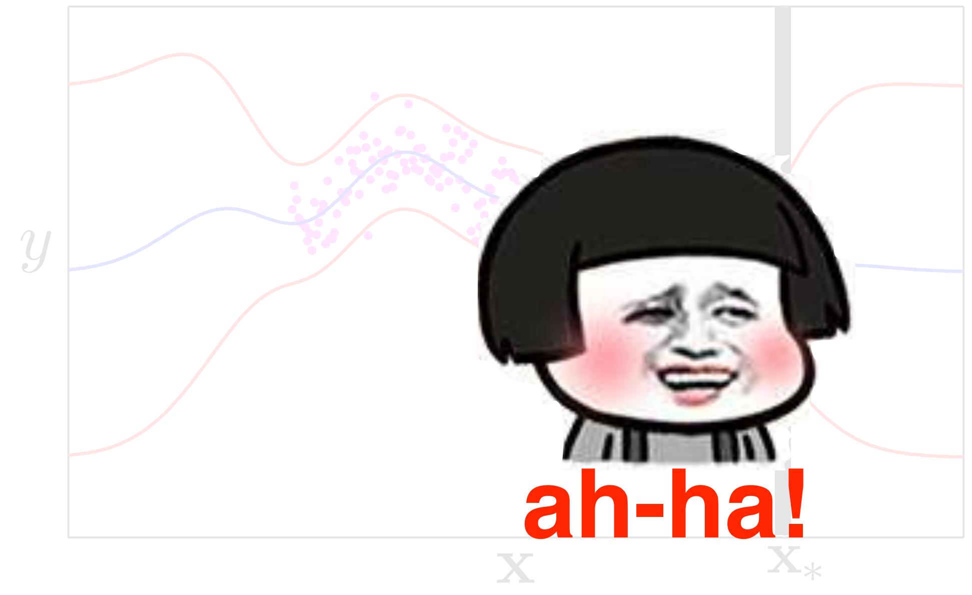
conditional distribution $p(y^*|\mathbf{y})$ is a Gaussian distribution with:

$$\mu(\mathbf{x}^*) = k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1}\mathbf{y}$$

$$\sigma^2(\mathbf{x}^*) = k^* - k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1} k(\mathbf{x}, \mathbf{x}^*)$$

$$k^* = k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1}$$

Gaussian Process Regression



prediction

denote $\mathbf{y}_{N+1} = [\mathbf{y}, y^*]^\top$ and $y^* = f(\mathbf{x}^*)$

$p(\mathbf{y}_{N+1}) = \mathcal{N}(\mathbf{y}_{N+1}, |\mathbf{0}, \mathbf{C}_{N+1})$

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C} & k(\mathbf{x}, \mathbf{x}^*) \\ k(\mathbf{x}, \mathbf{x}^*)^\top & k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1} \end{bmatrix}$$

1º kernel

conditional distribution $p(y^*|\mathbf{y})$ is a Gaussian distribution with:

2º conditional

$$\mu(\mathbf{x}^*) = k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1}\mathbf{y}$$
$$\sigma^2(\mathbf{x}^*) = k^* - k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1} k^*$$
$$k^* = k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1}$$

3º done!

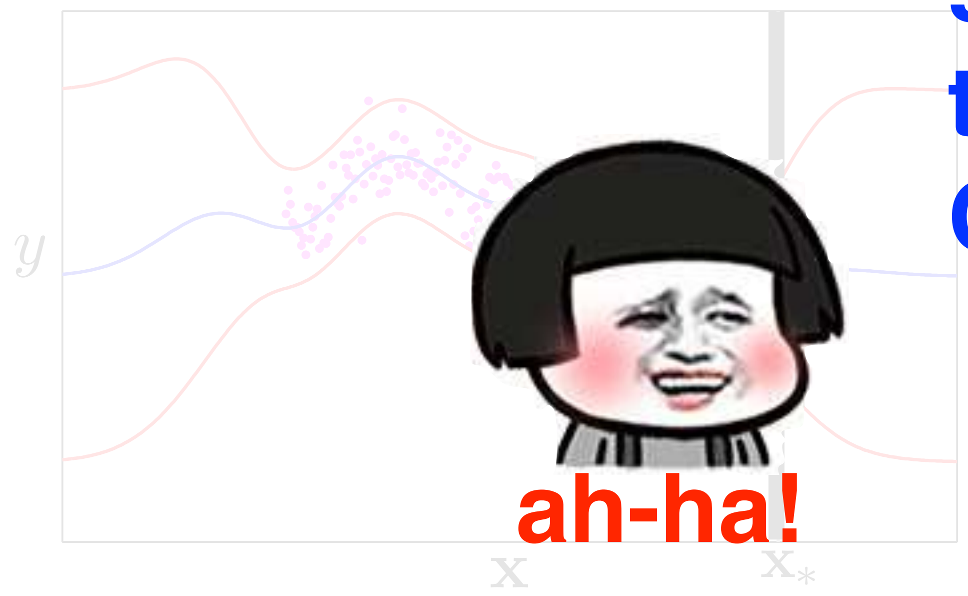
Gaussian Process Regression

similarity from testing x*
to all the training data!
$C^{-1}y$ is just a vector of scalars

prediction

denote $y_{N+1} = [\mathbf{y}, y^*]^\top$ and $y^* = f(\mathbf{x}^*)$

$$p(\mathbf{y}_{N+1}) = \mathcal{N}(\mathbf{y}_{N+1}, |\mathbf{0}, \mathbf{C}_{N+1})$$



ah-ha!

1º kernel

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C} & k(\mathbf{x}, \mathbf{x}^*) \\ k(\mathbf{x}, \mathbf{x}^*)^\top & k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1} \end{bmatrix}$$

conditional distribution $p(y^*|\mathbf{y})$ is a Gaussian distribution with:

2º conditional

$$\mu(\mathbf{x}^*) = \underline{k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1}\mathbf{y}}$$

$$\sigma^2(\mathbf{x}^*) = k^* - k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1} k^*$$

$$k^* = k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1}$$

3º done!

Gaussian Process: learning the kernel

Still remember this?

The kernel function that determines K should be chosen to express the property that:
for points x_n and x_m that are similar,
the corresponding values $y(x_n)$ and $y(x_m)$ will be more strongly correlated than for dissimilar points.

- Consider covariance function $\mathbf{C}$ with hyper-parameters ***θ***

$$k_\theta(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\{-\theta_1 \frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2}\} + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m$$

Gaussian Process: learning the kernel

Still remember this?
The kernel function that determines K should be chosen to express the property that:
for points x_n and x_m that are similar,
the corresponding values $y(x_n)$ and $y(x_m)$ will be more strongly correlated than for dissimilar points.

- Consider covariance function $\mathbf{C}$ with hyper-parameters $\boldsymbol{\theta}$

$$k_\theta(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\{-\theta_1 \frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2}\} + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m$$

  - Given $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^N\} = (\mathbf{X}, \mathbf{y})$, the marginal likelihood is function of $\boldsymbol{\theta}$

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

$$\ln\, p(\mathbf{y}|\theta) = -\frac{1}{2}\ln\, |\mathbf{C}_\theta| - \frac{1}{2}\mathbf{y}^\top \mathbf{C}_\theta^{-1}\mathbf{y} - \frac{N}{2}\ln\,(2\pi) ☞ \frac{\partial\, \ln\, p(\mathbf{y}|\theta)}{\partial\theta}$$

Gaussian Process: learning the kernel

Still remember this?

The kernel function that determines K should be chosen to express the property that:
for points x_n and x_m that are similar,
the corresponding values $y(x_n)$ and $y(x_m)$ will be more strongly correlated than for dissimilar points.

- Consider covariance function $\mathbf{C}$ with hyper-parameters $\boldsymbol{\theta}$

$$k_\theta(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left\{-\theta_1 \frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2}\right\} + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m$$

The keys in Gaussian Process Regression are:

1. **Choose kernel function**
2. **Estimate the proper hyper-parameters!**

- Given $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N = (\mathbf{X}, \mathbf{y})$, the marginal likelihood is function of $\boldsymbol{\theta}$

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

$$\ln p(\mathbf{y}|\theta) = -\frac{1}{2}\ln |\mathbf{C}_\theta| - \frac{1}{2}\mathbf{y}^\top \mathbf{C}_\theta^{-1}\mathbf{y} - \frac{N}{2}\ln (2\pi) ☞ \frac{\partial \ln p(\mathbf{y}|\theta)}{\partial \theta}$$

Feature selection can be achieved by ARD in GPs

Problem: Often there are *many* possible inputs that might be relevant to predicting a particular output. We need algorithms that automatically decide which inputs are relevant.

Automatic Relevance Determination:

Consider this covariance function:

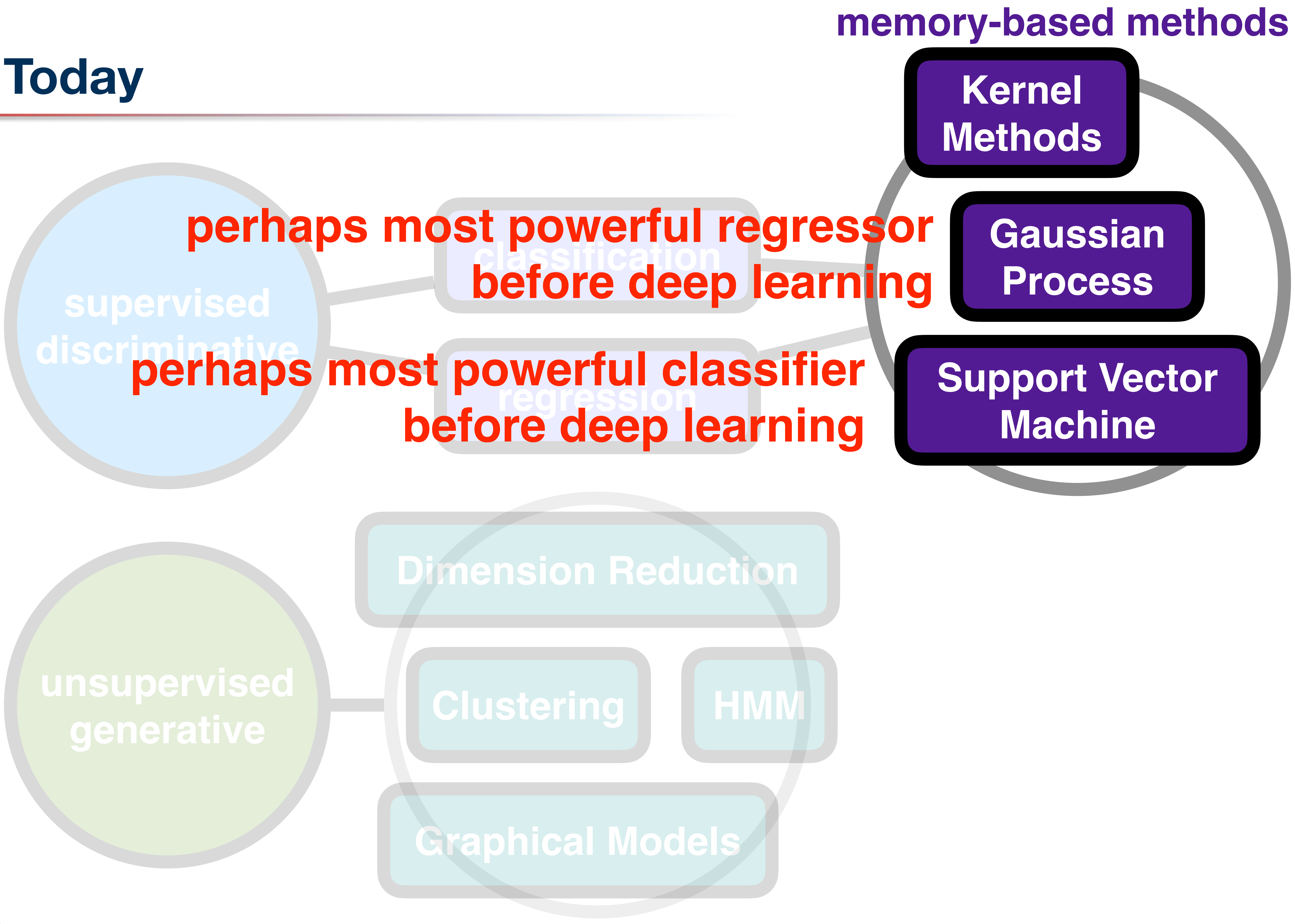
$$\mathbf{K}_{nn'} = v \exp\left[-\frac{1}{2} \sum_{d=1}^{D} \left(\frac{x_n^{(d)} - x_{n'}^{(d)}}{r_d} \right)^2 \right]$$

The parameter r_d is the length scale of the function along input dimension d.

As $r_d \to \infty$ the function f varies less and less as a function of $x^{(d)}$, that is, the dth dimension becomes *irrelevant*.

Given data, by learning the lengthscales $(r_1, \ldots, r_D)$ it is possible to do automatic feature selection.

Today


memory-based methods
Kernel Methods
Gaussian Process
Support Vector Machine
perhaps most powerful regressor before deep learning
perhaps most powerful classifier before deep learning
supervised discriminative
classification
regression
unsupervised generative
Dimension Reduction
Clustering
HMM
Graphical Models

Sparse kernel machines

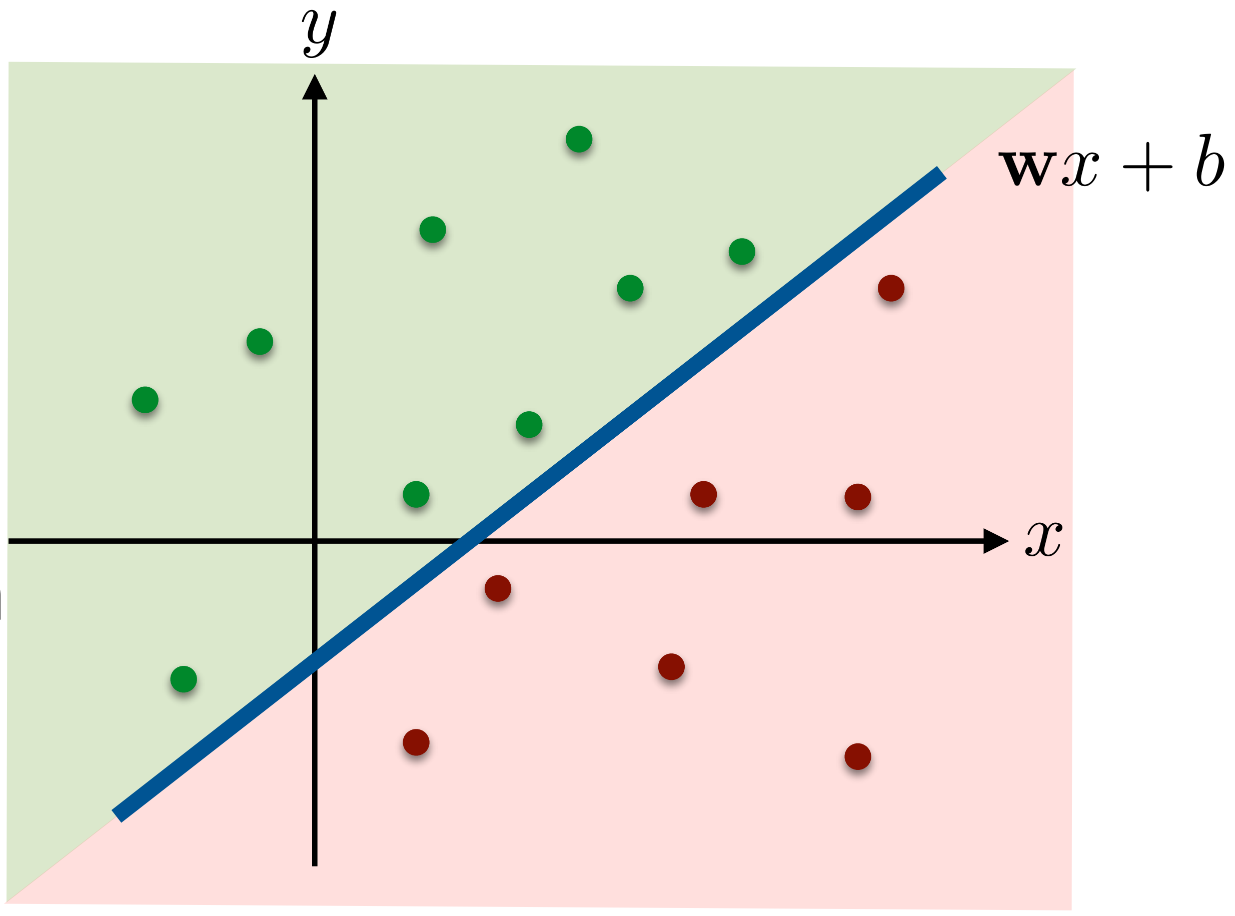
- Memory based methods: we would like to utilise the training data in the prediction!
  - What we have introduced (KNN, Parzen probability estimation, GPs) need to keep "ALL" training data … heavy memory demands!
  - Support Vector Machine (SVM), what we are going to introduce, is a kernel-based algorithm that have **sparse** solutions!
  - Predictions by kernel functions evaluated on **subset** of training data!

Sparse kernel machines

- Letz start from simple **binary classification** problem:

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

  - classification based on sign of $y(\mathbf{x})$
  - please note that explicit computing on $\phi(\mathbf{x})$ will be avoided by dual form ☞ kernel!

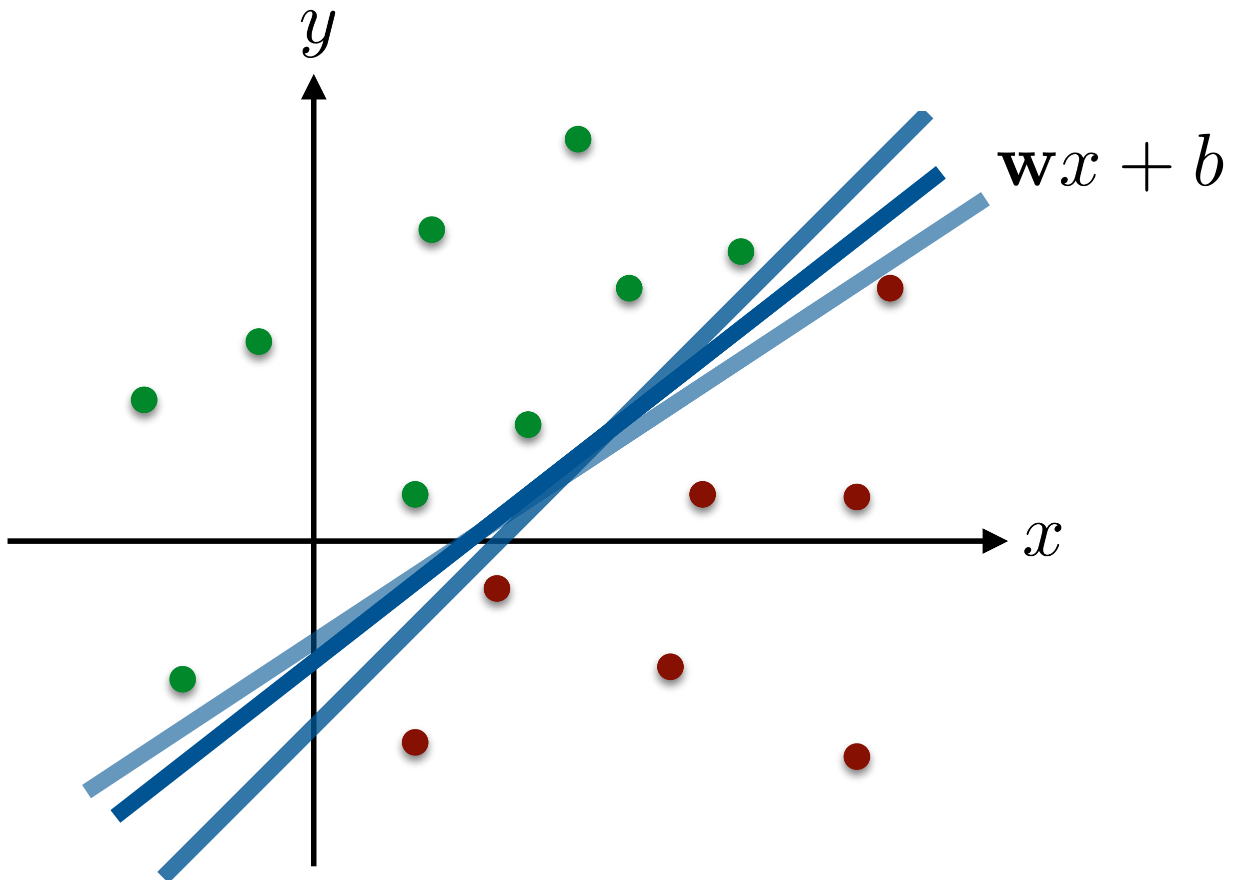


Sparse kernel machines

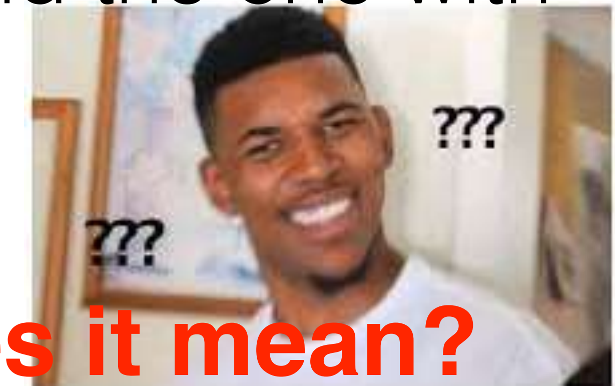
- Letz start from simple **binary classification** problem:

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

  - classification based on sign of $y(\mathbf{x})$
  - please note that explicit computing on $\phi(\mathbf{x})$ will be avoided by dual form ☞ kernel!



- When assuming the data is linearly separable, it means there will be infinite solutions for $\mathbf{w}$ and b, we should try to find the one with minimum generalisation error.



what does it mean?

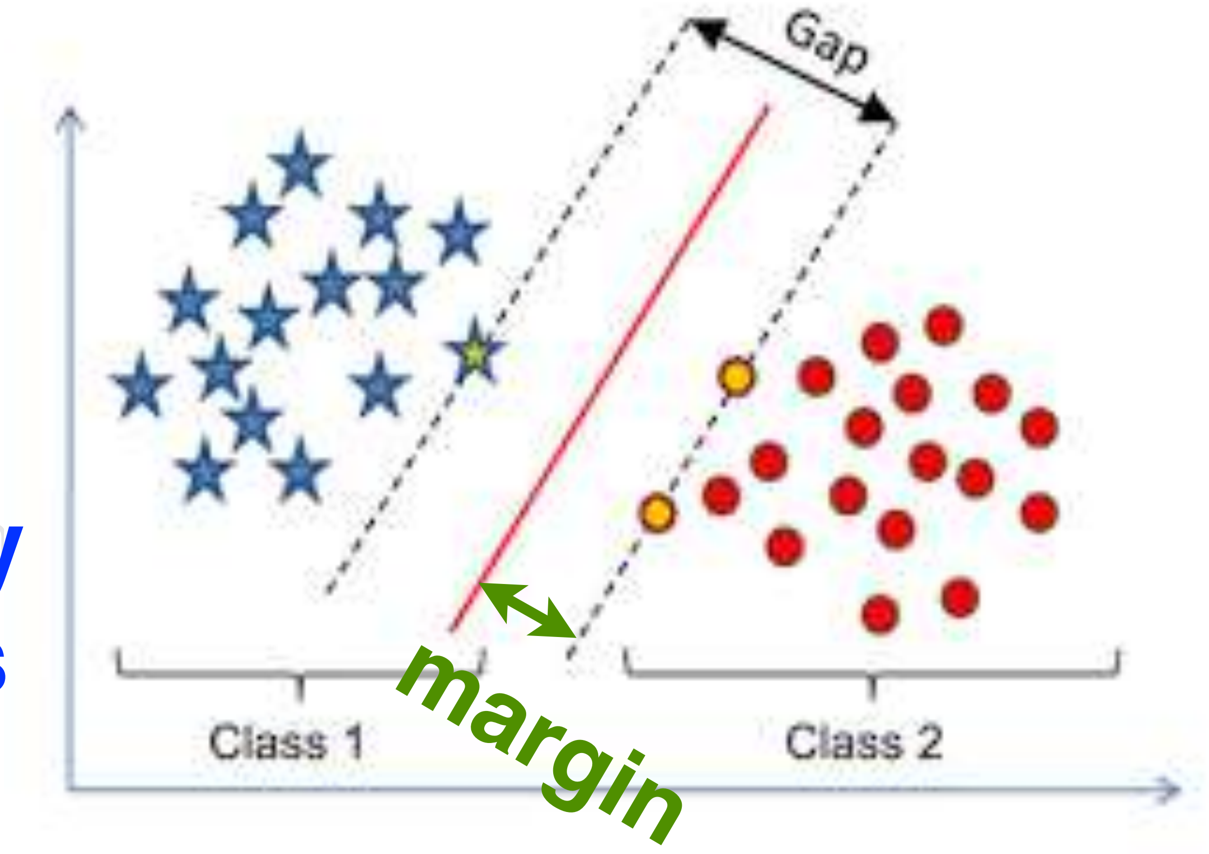
Sparse kernel machines

- Letz start from simple **binary classification** problem:

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

- [illegible] based on sign of $y(\mathbf{x})$
- please note that explicit computing [illegible]
  - ☞ kernel!

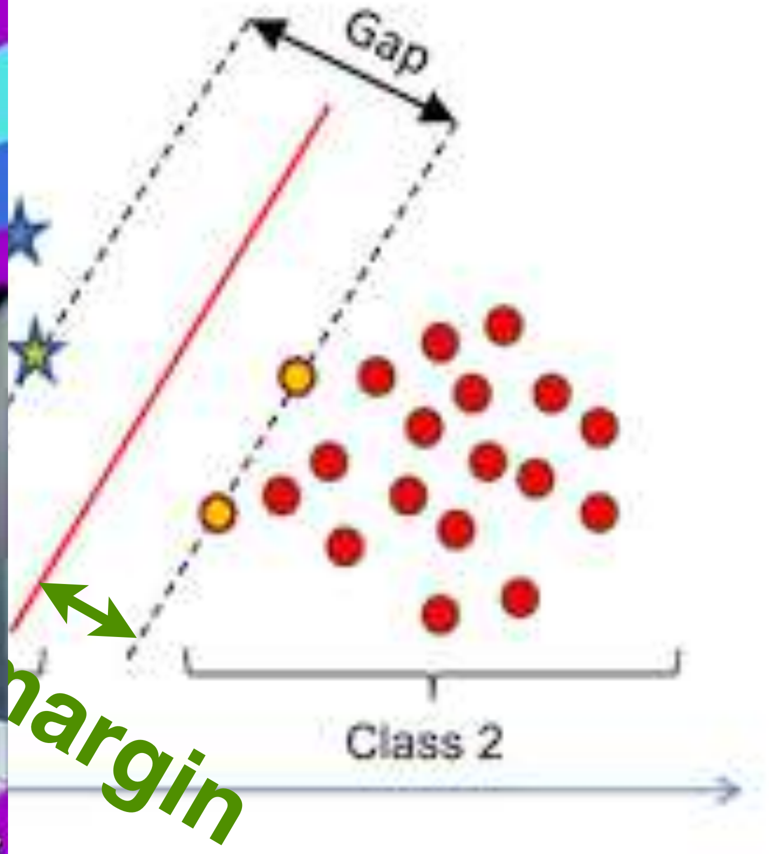
Maximise margin!
margin is the smallest distance b/w decision boundary and any of the training samples



- When assuming the data is linearly separable, it means there will be infinite solutions for $\mathbf{w}$ and b, we should try to find the one with minimum generalisation error.

Sparse kernel machines

Maximise margin!
margin is the sma
distance b/w decis
and any of the trai



$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

perpendicular distance from a data point x to a hyperplane $y(\mathbf{x}) = 0$ is given by

$$\frac{|y(\mathbf{x})|}{||\mathbf{w}||}$$

Sparse kernel machines

we are only interested in solutions for which all data points are correctly classified so that $\boxed{t_n y(\mathbf{x}_n) > 0}$ $\forall n$

t_n and $y(\mathrm{x}_n)$ with the same sign

$$\frac{t_n y(\mathbf{x})}{\|\mathbf{x}\|} = \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{x}\|}$$

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

perpendicular distance from a data point x to a hyperplane $y(\mathbf{x}) = 0$ is given by

$$\frac{\boxed{|y(\mathbf{x})|}}{\|\mathbf{w}\|}$$ absolute value

Sparse kernel machines

we are only interested in solutions for which all data points are correctly classified so that $t_n y(\mathbf{x}_n) > 0 \quad \forall n$

$$\frac{t_n y(\mathbf{x})}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$$

we want to maximise the margin!

$$\underset{\mathbf{w},b}{\operatorname{argmax}}\{\min_n \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}\}$$

margin is the smallest distance b/w decision boundary and any of the training samples

Sparse kernel machines

we are only interested in solutions for which all data points are correctly classified so that $t_n y(\mathbf{x}_n) > 0 \quad \forall n$

$$\frac{t_n y(\mathbf{x})}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$$

we want to maximise the margin!

$$\underset{\mathbf{w},b}{\operatorname{argmax}}\{\min_n \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}\}$$

note here if we scale $\mathbf{w}, b$ together $\frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$ won't change

so we set a condition to get the unique solution:

for the points closet to the hyperplane, $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) = 1$

which means, for all data points, $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1$

Sparse kernel machines

- Add these two criteria together!

$$\underset{\mathbf{w},b}{\operatorname{argmax}}\{\min_n \frac{\boxed{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n)+b)}}{\|\mathbf{w}\|}\} \qquad \boxed{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n)+b)=1}$$

- The overall optimisation problem becomes:

$$\text{maximise } \frac{1}{\|\mathbf{w}\|} \Rightarrow \text{minimise } \|\mathbf{w}\|$$

$$\text{subject to } t_n(\mathbf{w}^\top\phi(\mathbf{x}_n)+b) \geqslant 1$$

- Here comes Lagrange multipliers $a_n \geqslant 0 \quad \mathbf{a}=(a_1,\cdots,a_N)^\top$

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \sum_{n=1}^{N} a_n\{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n)+b)-1\}$$

remember regularisation story?

  - minimise w.r.t. $\mathbf{w}$ and b while maximise w.r.t. $\mathbf{a}$

Sparse kernel machines

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{n=1}^{N} a_n\{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) - 1\}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial \mathbf{w}} = 0 \quad ☞ \quad \underline{\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial b} = 0 \quad ☞ \quad 0 = \sum_{n=1}^{N} a_n t_n$$

w is the linear combination based on the feature representations of training data. If you put this w back to L(w, b ,a), it is easier to see that L has all x appearing as scalar product! ☞ kernel trick!

Sparse kernel machines

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \boxed{\sum_{n=1}^{N} a_n}\{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n) + \boxed{b}) \boxed{- 1}\}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial \mathbf{w}} = 0 \quad ☞ \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

cancel out

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial b} = 0 \quad ☞ \quad \boxed{0 = \sum_{n=1}^{N} a_n t_n}$$

- put $\mathbf{w}$ back to $L(\mathbf{w}, b, \mathbf{a})$ to get the **dual representation of the max-margin problem**, in which we maximise:

$$\tilde{L}(\mathbf{a}) = \boxed{\sum_{n=1}^{N} a_n} - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to} \quad a_n \geqslant 0, \quad n = 1, \cdots, N$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

Sparse kernel machines

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to} \quad a_n \geqslant 0, \quad n = 1, \cdots, N$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

use SMO (Sequential Minimal optimisation)
for QP (Quadratic Programming) to solution of $\mathbf{a}$
objective is quadratic and so any local optimum
will also be a global optimum provided that the
constraints define a convex region (since being linear)

Sparse kernel machines

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to} \quad a_n \geqslant 0, \quad n = 1, \cdots, N$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

after getting the solution for a, the prediction can be done by:

$$\boxed{\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^\top \phi(\mathbf{x}) + b \\ \mathbf{w} &= \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \end{aligned}}$$

☞ $$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \underline{k(\mathbf{x}, \mathbf{x}_n)} + b$$

kernel!

memory-based method!

Support Vector Machine

- See from **KKT condition**

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \sum_{n=1}^{N} a_n \underline{\{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n) + b) - 1\}}$$

$$a_n \geqslant 0$$

$$t_n y(\mathbf{x}_n) - 1 \geqslant 0$$

$$a_n\{t_n y(\mathbf{x}_n) - 1\} = 0$$

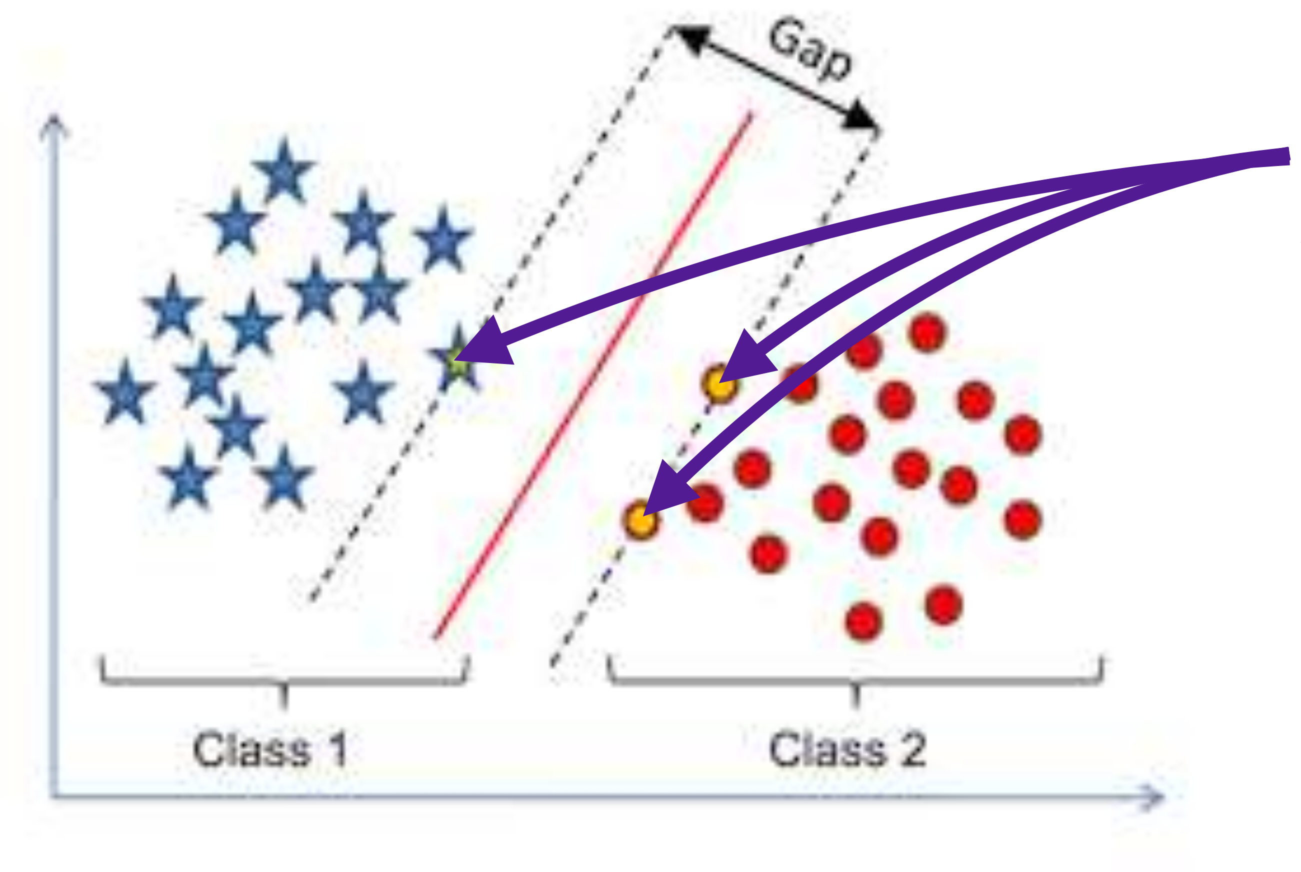
for every data point, either $\underline{a_n = 0}$ or $\underline{t_n y(\mathbf{x}_n) = 1}$

data points having this condition won't be used for prediction

data points having this condition are called: support vector

prediction: $y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$

Support Vector Machine



These data points (vectors) support the hyperplane! we can call them support vectors!

$t_n y(\mathbf{x}_n) = 1$

data points having this condition are called: support vector

remember few slides ago?

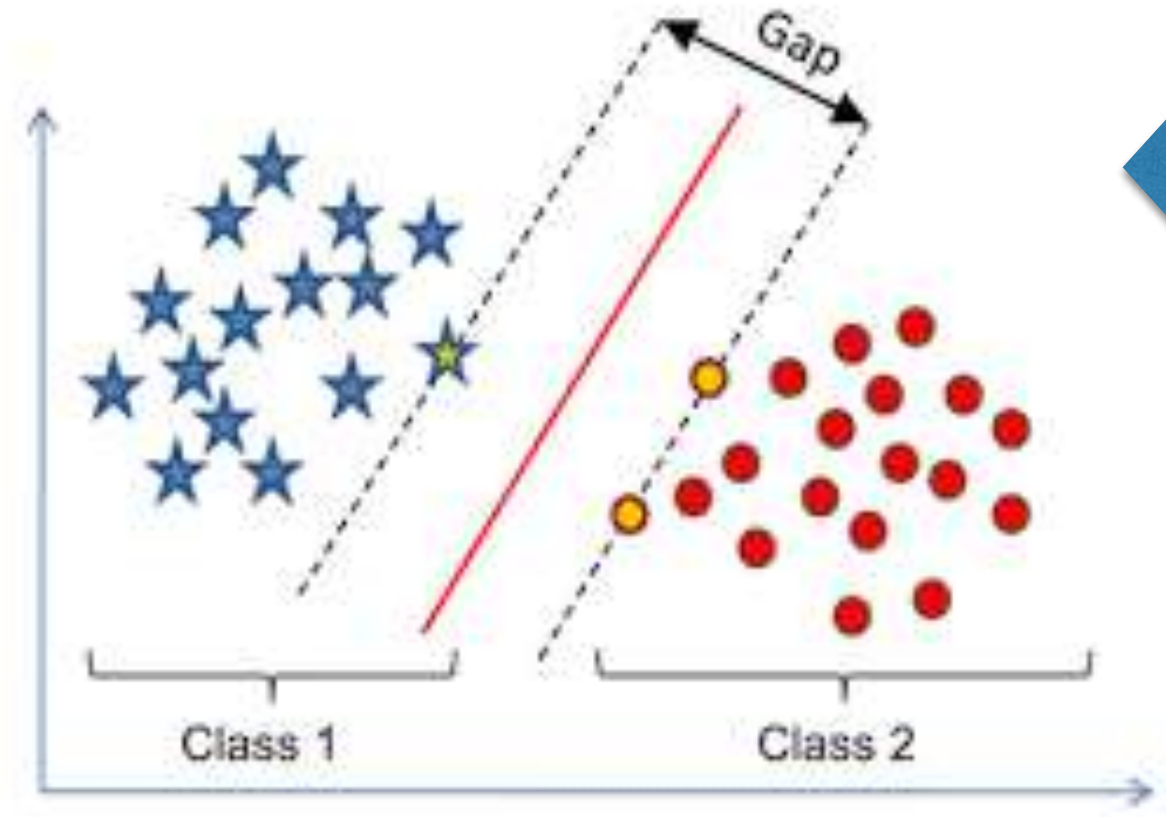
note here if we scale $\mathbf{w}, b$ together $\frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$ won't change

so we set a condition to get the unique solution:

for the points closet to the hyperplane, $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) = 1$

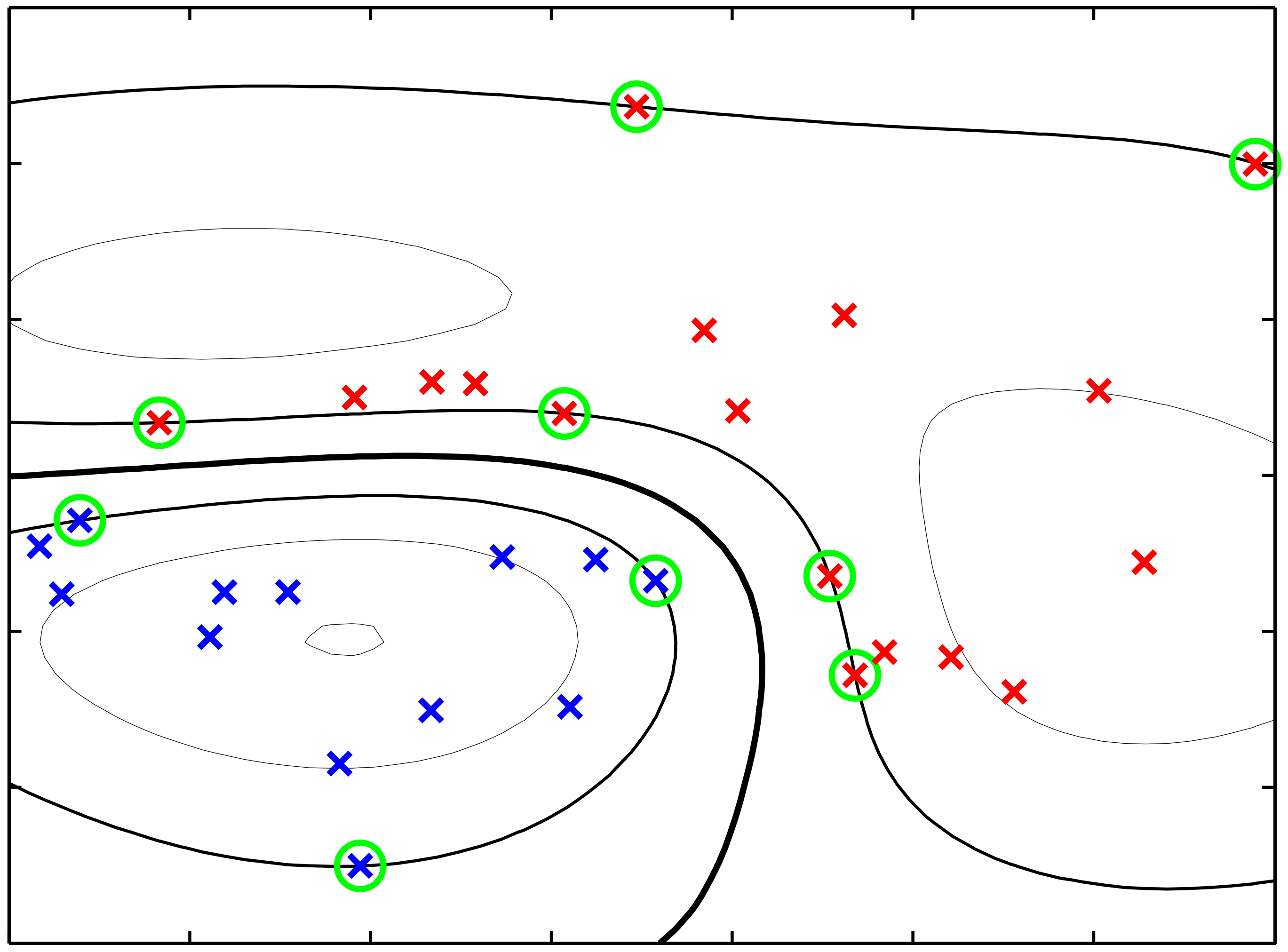
which means, for all data points, $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1$

Support Vector Machine



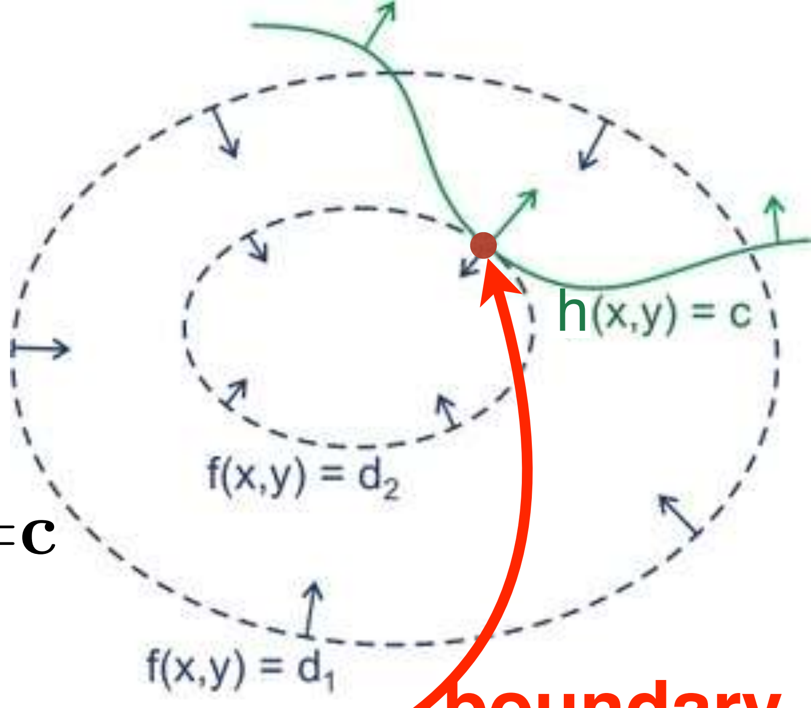
linearly separable in feature space (whose dimensionality could be much higher due to kernel function)

how decision boundary could be in original data space



KKT condition

- simple optimisation problem
  - Green line represents equality constraint $\mathbf{h(x,y)=c}$
    Blue line shows contour lines of function $\mathbf{f}$
  - Intuitively you can see in the optimum, the gradient of $\mathbf{f}$ is parallel to the gradient of $\mathbf{h}$



boundary solution

$\nabla f \in \text{span}\{\nabla h\}$

i.e. $\nabla f = -\eta \nabla h$

$$let \quad L(X,\lambda,\mu)=f(X)+\sum_{j=1}^{p}\lambda_j \underline{h_j(X)}+\sum_{k=1}^{q}\mu_k \underline{g_k(X)}$$

equality constraints — inequality constraints

KKT condition

$$\frac{\partial L}{\partial X}\bigg|_{\underline{X=X^*}}=0$$

$$\boxed{\lambda_j \neq 0,}$$

$$\boxed{u_k \geq 0, \quad \boxed{u_k g_k(X^*)=0}}$$

$$\boxed{h_j(X^*)=0 \qquad j=1,2,\ldots,p}$$

$$\boxed{g_k(X^*)\leq 0 \qquad k=1,2,\ldots q}$$

equality constraints should be satisfied

This holds for both interior or boundary solution, called as complementary slackness. (for interior solution, inequality constraints inactive, $u=0$)

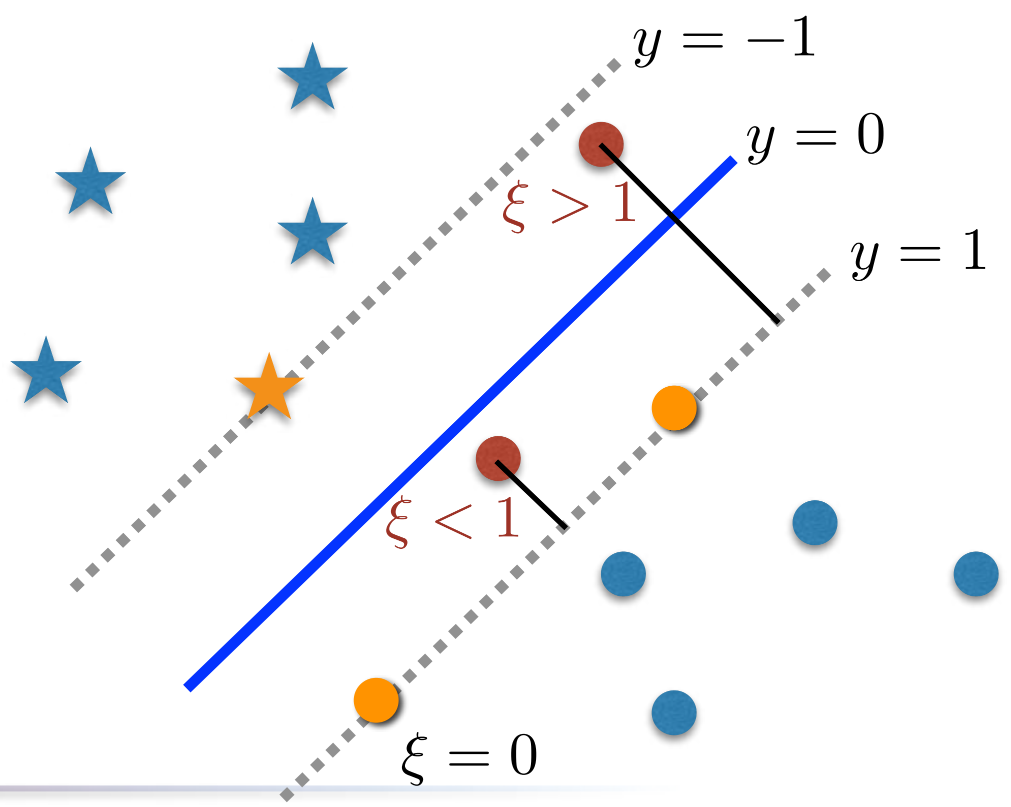
inequality constraints to be active

Support Vector Machine - with Soft Margin!

- Data points are allowed to be on the "wrong side" of boundary
  - But with a penalty ~~inversely~~ proportional to the distance from boundary

slack: $\xi_n = |t_n - y(\mathbf{x}_n)|$

  - inequality $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1$ replaced by $\underline{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1 - \xi_n}$ also $\underline{\xi_n \geqslant 0}$
  - optimisation problem now as subject to two constraints above $\underbrace{C \sum_{n=1}^{N} \xi_n}_{} + \frac{1}{2} \|\mathbf{w}\|^2$ **also minimise slack**



Support Vector Machine - with Soft Margin!

- Data points are allowed to be on the "wrong side" of boundary
  - But with a penalty inversely proportional to the distance from boundary

slack: $\xi_n = |t_n - y(\mathbf{x}_n)|$

  - inequality $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1$ replaced by $\underline{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geqslant 1 - \xi_n}$ also $\underline{\xi_n \geqslant 0}$
  - optimisation problem now as subject to <u>two constraints</u> above $\underbrace{C \sum_{n=1}^{N} \xi_n}_{\text{also minimise slack}} + \frac{1}{2} \|\mathbf{w}\|^2$

also minimise slack

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

- KKT condition

$$\begin{aligned} a_n &\geqslant 0 \\ t_n y(\mathbf{x}_n) - 1 + \xi_n &\geqslant 0 \\ a_n\left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) &= 0 \\ \mu_n &\geqslant 0 \\ \xi_n &\geqslant 0 \\ \mu_n \xi_n &= 0 \end{aligned}$$

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

- KKT condition

$$\begin{aligned} a_n &\geqslant 0 \\ t_n y(\mathbf{x}_n) - 1 + \xi_n &\geqslant 0 \\ a_n\left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) &= 0 \\ \mu_n &\geqslant 0 \\ \xi_n &\geqslant 0 \\ \mu_n\xi_n &= 0 \end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n.$$

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

- KKT condition

$$\begin{aligned}
\boxed{a_n \geqslant 0} \\
t_n y(\mathbf{x}_n) - 1 + \xi_n &\geqslant 0 \\
a_n\left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) &= 0 \\
\boxed{\mu_n \geqslant 0} \\
\xi_n &\geqslant 0 \\
\mu_n\xi_n &= 0
\end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow \boxed{a_n = C - \mu_n}.$$

☞ $a_n \leqslant C$

☞ $0 \leqslant a_n \leqslant C$

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

put back to $L(\mathrm{w, b, a})$

☞ $$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to $\quad 0 \leqslant a_n \leqslant C$

$$\sum_{n=1}^{N} a_n t_n = 0$$

use SMO for QP to get solution of a

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

likewise, after getting the solution for a, the prediction can be done by:

$$\boxed{\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^\top \phi(\mathbf{x}) + b \\ \mathbf{w} &= \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \end{aligned}}$$

☞ $$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \underline{k(\mathbf{x}, \mathbf{x}_n)} + b$$

kernel!

memory-based method!

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

- KKT condition

$$\boxed{\begin{aligned} a_n &\geqslant 0 \\ t_n y(\mathbf{x}_n) - 1 + \xi_n &\geqslant 0 \\ a_n\left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) &= 0 \end{aligned}}$$
$$\begin{aligned} \mu_n &\geqslant 0 \\ \xi_n &\geqslant 0 \\ \mu_n\xi_n &= 0 \end{aligned}$$

☞ for every data point,
either $\underline{a_n = 0}$ or $\underline{t_n y(\mathbf{x}_n) = 1 - \xi_n}$

not used in prediction

data points related to support vector

prediction: $y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$

Support Vector Machine - with Soft Margin!

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

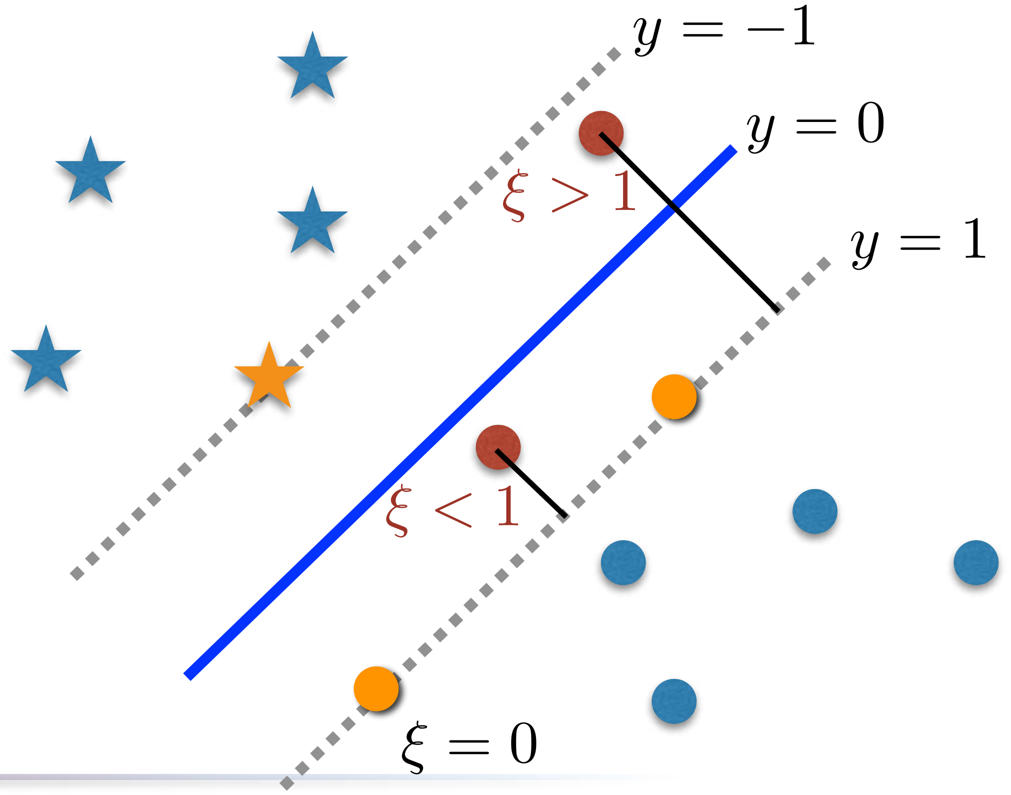
where $a_n \geqslant 0, \mu_n \geqslant 0$ are Lagrange multipliers

$0 \leqslant a_n \leqslant C$

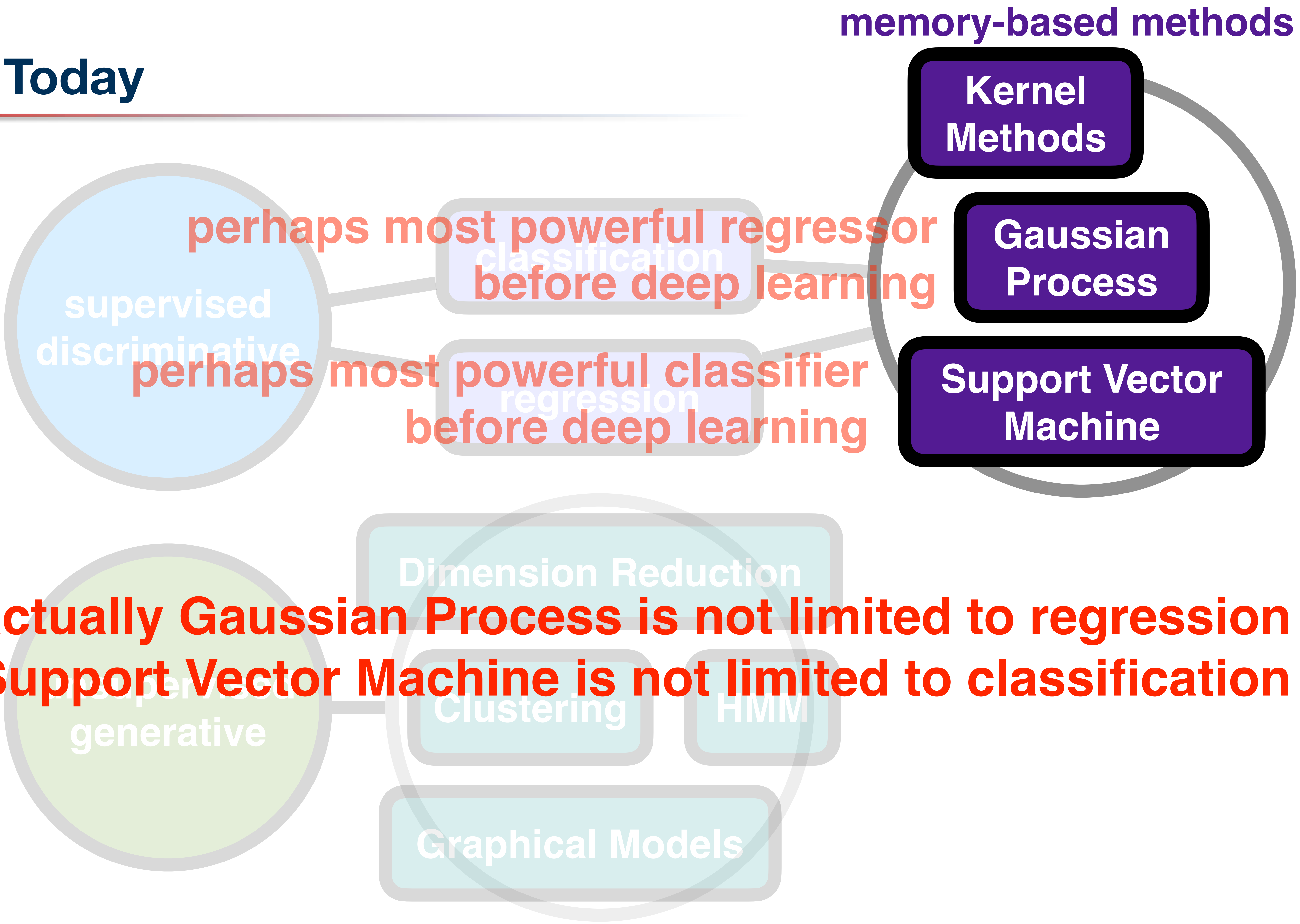
if $a_n = 0$, then such points not used in prediction

if $\boxed{a_n < C}$ because $a_n = C - \mu_n$ implies $\mu_n > 0$, from $\mu_n\xi_n = 0$ we get $\boxed{\xi_n = 0}$

if $a_n = C$, then such points lie inside the margin



Today



actually Gaussian Process is not limited to regression
Support Vector Machine is not limited to classification

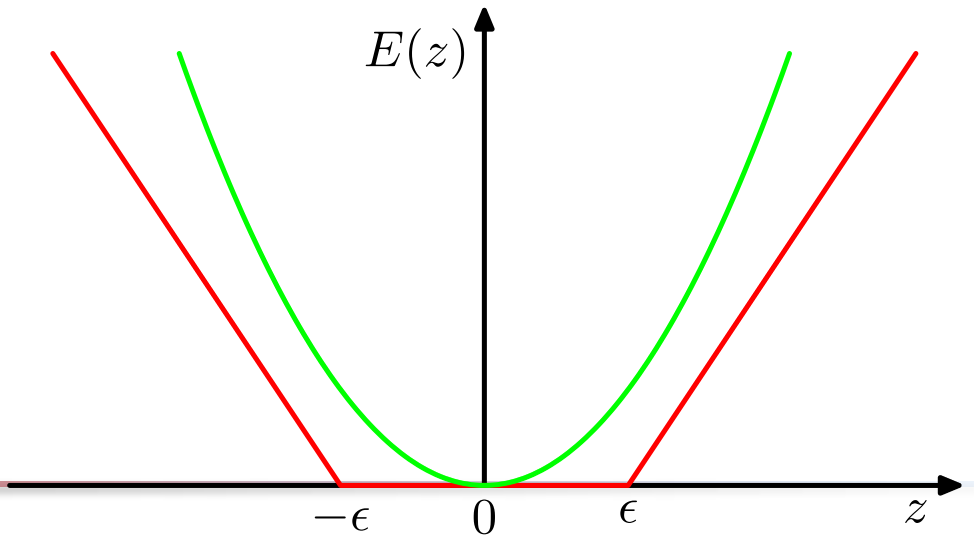
Support Vector Machine for Regression

- In simple linear regression, we minimise a regularised error function given by

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- To obtain sparse solution, we define an *ϵ-insensitive error function* which gives zero error if the absolute difference between the prediction $y(\mathbf{x})$ and the target t is less then ϵ where $\epsilon > 0$
  - Example:

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & \text{if } |y(\mathbf{x}) - t| < \epsilon; \\ |y(\mathbf{x}) - t| - \epsilon, & \text{otherwise} \end{cases}$$

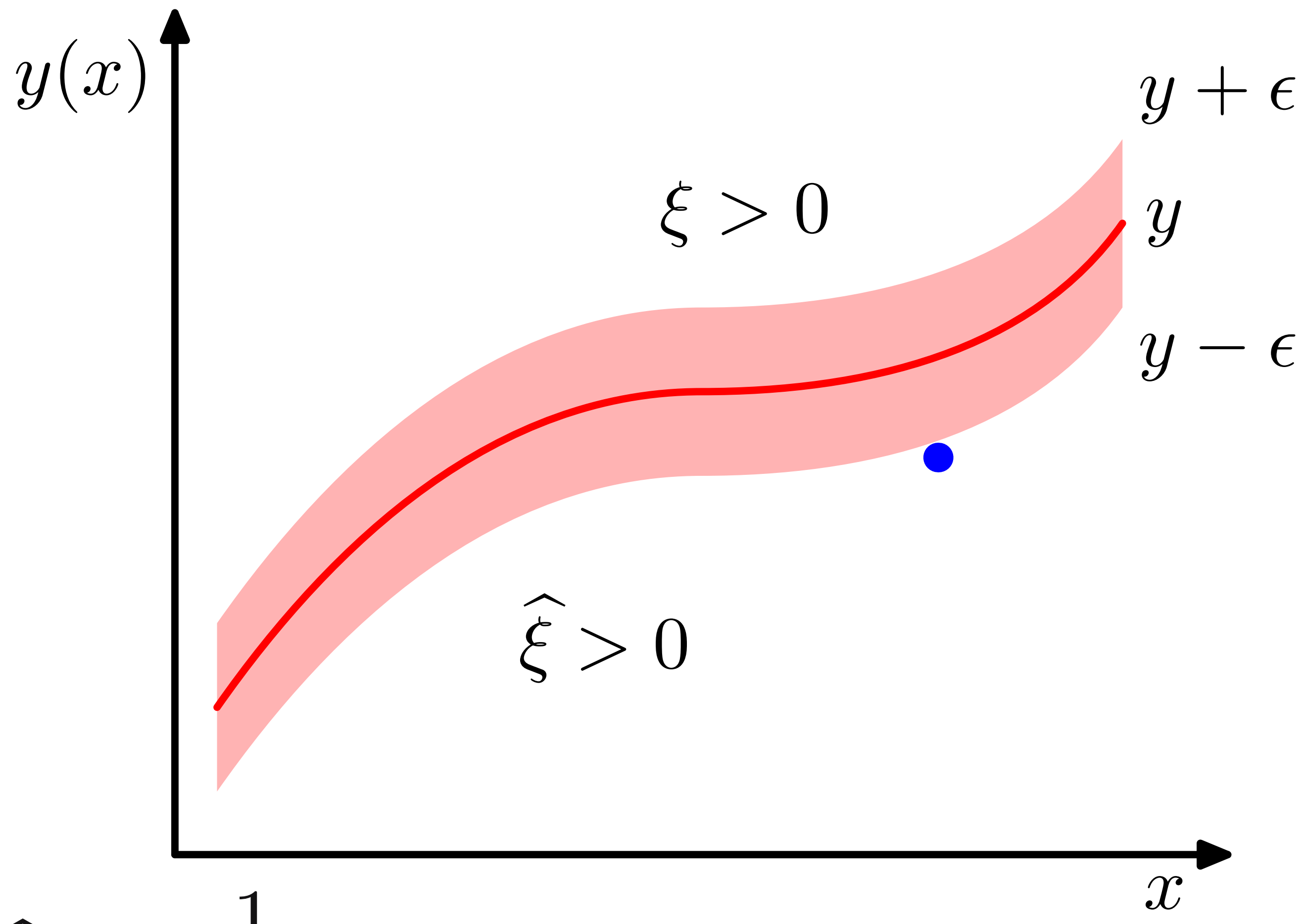


☞ $$C\sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2$$

Support Vector Machine for Regression

- With also slack :)

Illustration of SVM regression, showing the regression curve together with the ϵ-insensitive 'tube'. Also shown are examples of the slack variables ξ and $\widehat{\xi}$. Points above the ϵ-tube have $\xi > 0$ and $\widehat{\xi} = 0$, points below the ϵ-tube have $\xi = 0$ and $\widehat{\xi} > 0$, and points inside the ϵ-tube have $\xi = \widehat{\xi} = 0$.



$$C \sum_{n=1}^{N} (\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{subject to} \quad \xi_n \geqslant 0 \quad \hat{\xi}_n \geqslant 0$$

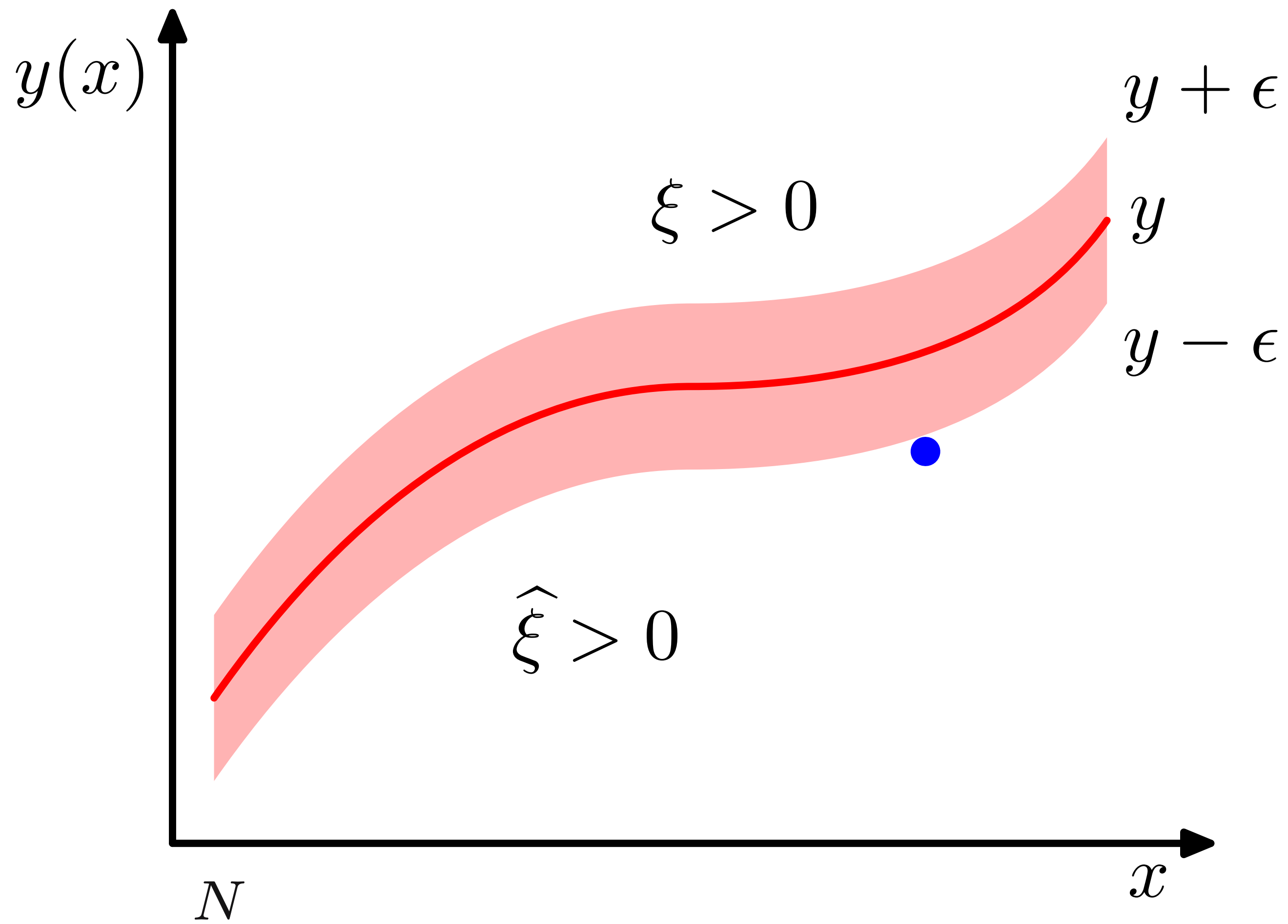
$$t_n \leqslant y(\mathbf{x}_n) + \epsilon + \xi_n$$

$$t_n \geqslant y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n$$

Support Vector Machine for Regression

- With also slack :)

Illustration of SVM regression, showing the regression curve together with the ϵ-insensitive 'tube'. Also shown are examples of the slack variables ξ and $\widehat{\xi}$. Points above the ϵ-tube have $\xi > 0$ and $\widehat{\xi} = 0$, points below the ϵ-tube have $\xi = 0$ and $\widehat{\xi} > 0$, and points inside the ϵ-tube have $\xi = \widehat{\xi} = 0$.



$$L = C\sum_{n=1}^{N}(\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}(\mu_n\xi_n + \widehat{\mu}_n\widehat{\xi}_n)$$
$$- \sum_{n=1}^{N} a_n(\epsilon + \xi_n + y_n - t_n) - \sum_{n=1}^{N}\widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n)$$

Support Vector Machine for Regression

- Dual representation
  - Maximising

$$\widetilde{L}(\mathbf{a},\widehat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n-\widehat{a}_n)(a_m-\widehat{a}_m)k(\mathbf{x}_n,\mathbf{x}_m) - \epsilon\sum_{n=1}^{N}(a_n+\widehat{a}_n)+\sum_{n=1}^{N}(a_n-\widehat{a}_n)t_n$$

$$\text{subject to} \quad 0 \leq a_n \leq C$$

$$0 \leq \hat{a}_n \leq C$$

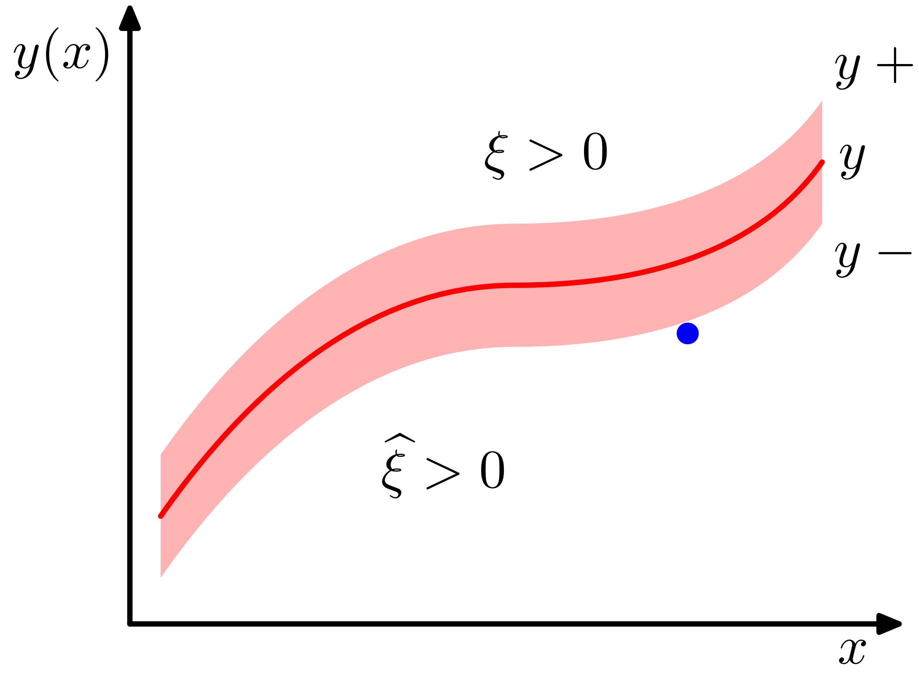
$$\sum_{n=1}^{N}(a_n-\widehat{a_n})=0$$

prediction: $y(\mathbf{x})=\sum_{n=1}^{N}(a_n-\widehat{a_n})k(\mathbf{x},\mathbf{x}_n)+b$

Support Vector Machine for Regression

- KKT condition

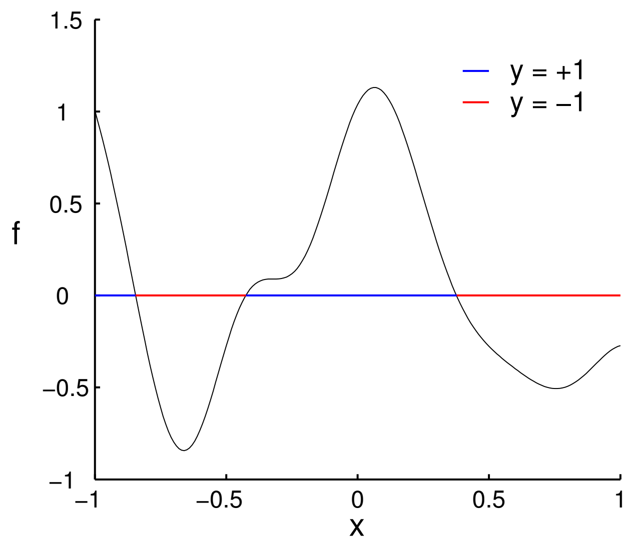
$$\begin{aligned} a_n(\epsilon + \xi_n + y_n - t_n) &= 0 \\ \widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n) &= 0 \\ (C - a_n)\xi_n &= 0 \\ (C - \widehat{a}_n)\widehat{\xi}_n &= 0. \end{aligned}$$

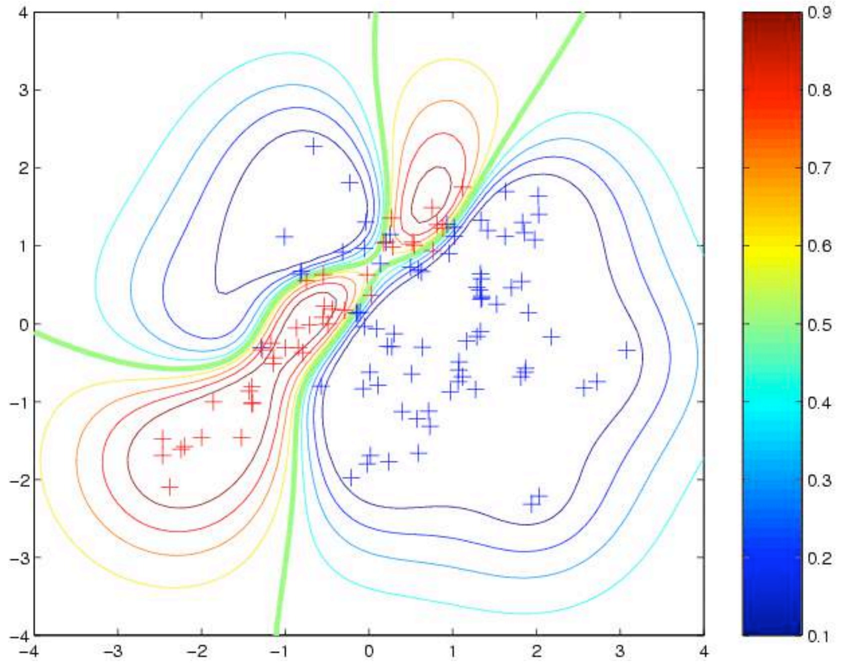


1. a_n can only be nonzero if $\epsilon + \xi_n + y_n - t_n$ =0
   ☞ data points either lies on the upper boundary of the tube: $\xi_n = 0$, or above $\xi_n > 0$
2. $\widehat{a}_n$ can only be nonzero if $\epsilon + \widehat{\xi}_n - y_n + t_n$ =0
   ☞ data points either lies on the lower boundary of the tube: $\widehat{\xi}_n = 0$, or below $\widehat{\xi}_n > 0$
3. $\epsilon + \xi_n + y_n - t_n$ and $\epsilon + \widehat{\xi}_n - y_n + t_n$ are incompatible
   ☞ for every data point, either a_n or $\widehat{a}_n$ (or both) must be nonzero
4. Support vectors are those data points either a_n nonzero or $\widehat{a}_n$ nonzero
   ☞ these are data points that lie on the boundary of the tube or outside the tube
5. All points within the tube have $a_n = \widehat{a}_n = 0$

Gaussian Process for Classification

- Binary classification problem
  - Given dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{n=1}^n\} = (X, \mathbf{y})$ where $\mathbf{y}$ are binary labels $\{-1, +1\}$, infer class label probabilities at new points.





  - There are many ways to relate function values to class probabilities

$$
p(y_i|f_i) = \begin{cases} \frac{1}{1+\exp(-y_i f_i)} & \text{sigmoid (logistic)} \\ \Phi(y_i f_i) & \text{cumulative normal (probit)} \\ \boldsymbol{H}(y_i f_i) & \text{threshold} \\ \epsilon + (1-2\epsilon)\boldsymbol{H}(y_i f_i) & \text{robust threshold} \end{cases}
$$

  - Inference not that easy: approximation.

Limitations of SVM w.r.t. GPs

- The outputs of an SVM represent decisions rather than posterior probabilities.
- The SVM was originally formulated for two classes, and the extension to K > 2 classes is problematic.
- Gaussian Process can learn the kernel parameters automatically from data, no cross-validation is needed.
- Gaussian Process can be used for automatic feature selection.
- Gaussian Process can incorporate interpretable noise models and priors over functions, and can sample from prior to get intuitions about the model assumptions.