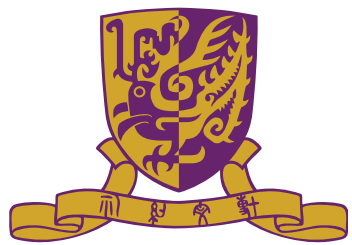
DDA5001 Machine Learning

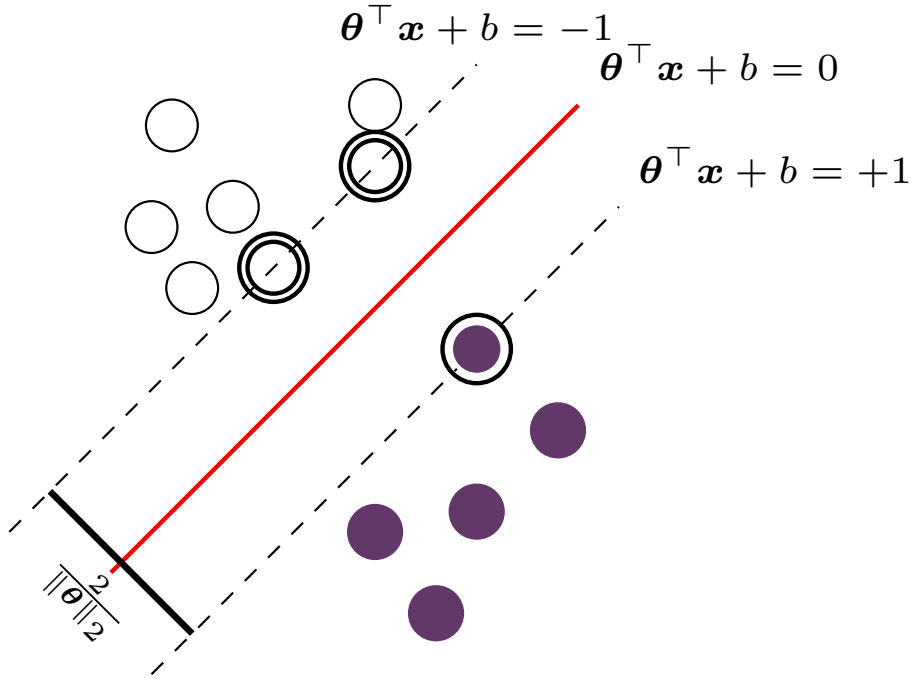
Kernel Method

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen



Recap: Hard-margin SVM



Hard-margin SVM:

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^d, b\in\mathbb{R}}{\text{minimize}} \ \|\boldsymbol{\theta}\|_2^2$$

$$\text{subject to } \ y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i + b) \geq 1, \quad i = 1, \ldots, n$$

Hard-margin SVM can be regarded as: Choose a classifier from all possible perceptron classifiers that has the largest margin.

Recap: Regularization View and Modified VC Analysis

- SVM is a linear classifier robust to noise. Such a robustness can be interpreted as: SVM is better 'regularized', which corresponds to the weight decay term in the objective function.
- SVM maximizes margin, which also helps reduce the generalization error by using margin ρ rather than number of parameters $d+1$.

Theorem: VC dimension of margin-ρ hyperplanes

Suppose the input space is the ball of radius R in $\mathbb{R}^d$, that is $\|\boldsymbol{x}\| \leq R$. Then,

$$d_{\text{VC}}(\rho) \leq \left\lceil R^2/\rho^2 \right\rceil + 1.$$

- Since its VC dimension is at most $d+1$. Thus, we can have

$$d_{\text{VC}}(\rho) \leq \min\{\left\lceil R^2/\rho^2 \right\rceil, d\} + 1.$$

- The bound suggests that the margin ρ can be used to control the model complexity. Hence, seeking for a max margin (max ρ) can lead to better Er_{out}.

Recap: Soft-margin SVM

Hinge loss:

$$h(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = \max(0, 1 - y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i + b))$$

- The hinge loss is a relaxation of the perceptron hard constraints.

The soft-margin SVM is to solve

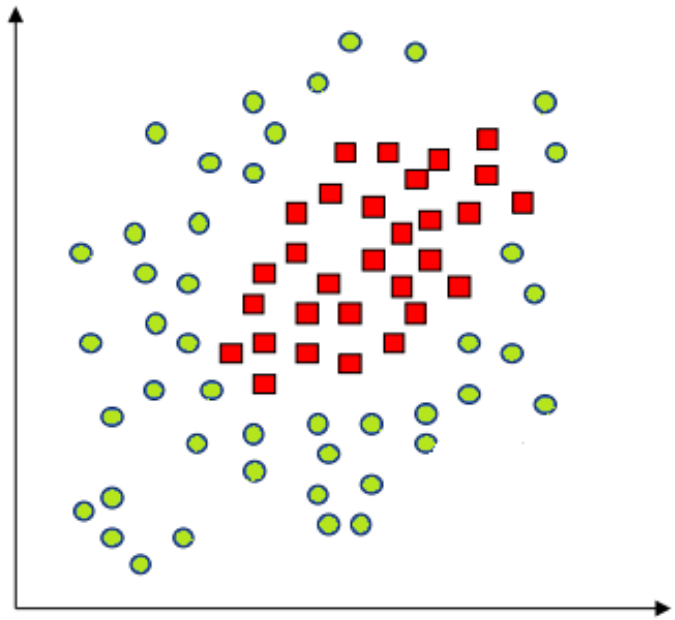
$$\underset{\boldsymbol{\theta} \in \mathbb{R}^d, b \in \mathbb{R}}{\text{minimize}} \ \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i + b)) + \lambda \|\boldsymbol{\theta}\|_2^2$$

- $\lambda > 0$ determines the trade-off between increasing the margin and ensuring that $\boldsymbol{x}_i$ lies on the correct side of the margin.
- If the data is linearly separable, then we can choose a sufficiently small λ to let the soft-margin SVM works the same as hard-margin SVM.
- If data is not linearly separable, soft-margin SVM can still provide a meaningful classifier.

Kernel Method

The Limitation of Linear Model

- We have studied Perceptron, LS, LR, SVM. They are all linear models.
- Sometimes linear model is restrictive.

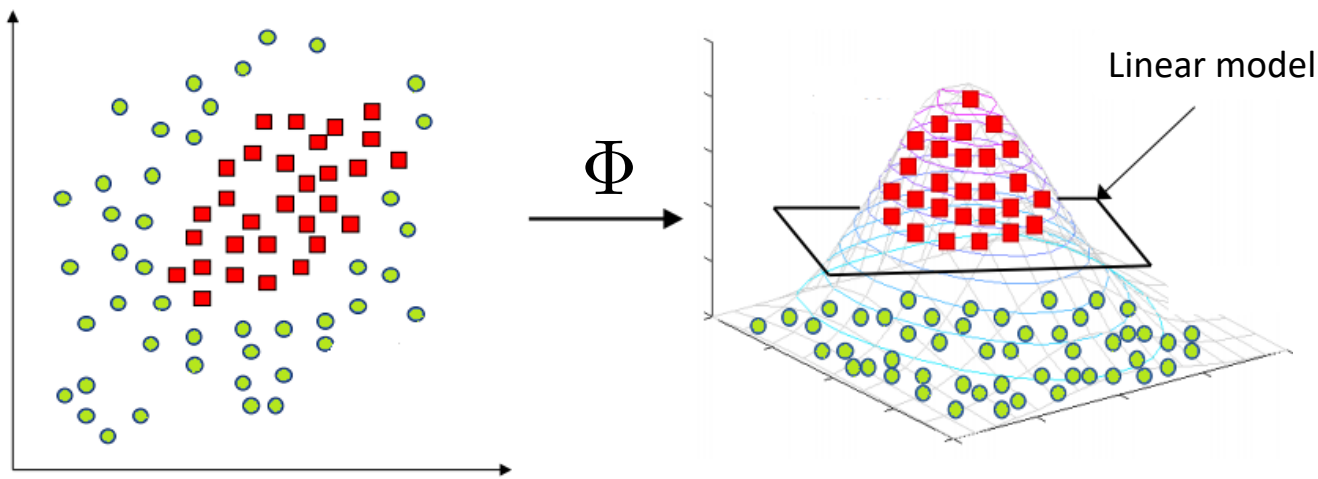


- Can you find a linear model that classify well these two classes?

Nonlinear Transform

The Underlying Idea

- No. We cannot find a linear model in dimension d.
- But, how about in higher dimension p?



Nonlinear Transformation to Data

Transformation function:

$$\Phi(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^p, \quad d < p.$$

Example in 1d:

$$\Phi(x) = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}.$$

Then, try a linear model on the transformed data $\{\Phi(\boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}_n)\}$.

Generalized Linear Model (Classification Case)

Let us take binary classification as an example. The same idea applies to regression.

- Original training data: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$.
- Transformed training data:

$$\{(\Phi(\boldsymbol{x}_i), y_i)\}_{i=1}^n.$$

- It becomes linearly separable even with linear model in $\boldsymbol{\theta} \in \mathbb{R}^p$

$$y_i = \operatorname{sign}\left(\Phi(\boldsymbol{x}_i)^\top \boldsymbol{\theta}\right), \quad \forall i = 1, \ldots, n.$$

$\rightsquigarrow$ Then applying Perceptron, LR, or SVM to learn $\widehat{\boldsymbol{\theta}}$ based on $\{(\Phi(\boldsymbol{x}_i), y_i)\}_{i=1}^n$.

Issues of Nonlinear Transformation

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \overset{\Phi}{\Longrightarrow} \quad \Phi(\boldsymbol{x}) = \begin{bmatrix} \Phi^{(1)}(\boldsymbol{x}) \\ \Phi^{(2)}(\boldsymbol{x}) \\ \vdots \\ \Phi^{(p)}(\boldsymbol{x}) \end{bmatrix}$$

where $p > d$.

- Drawback I: We can always find a separating hyperplane, by letting $p > n$.
  - However, this leads to overfitting.
  - Fortunately, this issue can be tackled by using regularization.
- Drawback II: Increasing computational load with increasing p (especially when $p \to \infty$).
  $\rightsquigarrow$ This issue can be solved by kernel method / trick.

Kernel Method / Trick

Assumption of Kernel Method

- We will study a trick used to avoid the previous computational issue for lifting data to higher dimension, i.e., the so-called kernel method.

Assumption on The Solution

We assume there exists $\boldsymbol{\alpha} \in \mathbb{R}^n$ such that the solution $\widehat{\boldsymbol{\theta}} = \sum_{j=1}^n \alpha_j \boldsymbol{x}_j$, where $\{\boldsymbol{x}_j\}_{j=1}^n$ are training samples.

- With this assumption, instead of formulating learning problem over $\boldsymbol{\theta} \in \mathbb{R}^d$, we can formulate learning problem over $\boldsymbol{\alpha} \in \mathbb{R}^n$ by replacing $\boldsymbol{\theta} = \sum_{j=1}^n \alpha_j \boldsymbol{x}_j$.
- This assumption is mild as long as $n \geq d$, as likely training samples $\{\boldsymbol{x}_j\}_{j=1}^n$ has d independent samples. In this case, the training data is easy to represent $\boldsymbol{\theta}$ using linear representation.

Kernel Method / Trick

- In linear model, we always have $\boldsymbol{\theta}^\top \boldsymbol{x}$. Plugging $\boldsymbol{\theta} = \sum_{j=1}^n \alpha_j \boldsymbol{x}_j$ into the linear model gives that many machine learning methods only involve the data through inner products

$$\boldsymbol{x}^\top \boldsymbol{x}',$$

  where $\boldsymbol{x}, \boldsymbol{x}'$ are two training samples.

- After applying the nonlinear transform Φ, the inner product becomes

$$\Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{x}') \in \mathbb{R}.$$

- Kernel method is a trick to evaluate the above inner product without explicitly calculating $\Phi(\boldsymbol{x})$ and $\Phi(\boldsymbol{x}')$.

Kernel

A function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel for a nonlinear transform Φ if

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{x}'), \quad \forall \boldsymbol{x}, \boldsymbol{x}'$$

Example of Kernel

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \stackrel{\Phi}{\Longrightarrow} \quad \Phi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

- The direct way of performing nonlinear transform:
  - Compute $\Phi(\boldsymbol{x})$ and $\Phi(\boldsymbol{x}')$.
  - Compute $\Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{x}')$.
- The kernel trick:

  $$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^2,$$

  which equals to $\Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{x}')$.
  $\rightsquigarrow$ All the computations remain in dimension d. We never need to go to dimension p, i.e., compute $\Phi(\boldsymbol{x})$, explicitly.

Valid Kernels

- In practice, we first define the kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.
- How can we verify if there exists a Φ such that

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{x}') \ ?$$

Mercer's theorem

κ is a valid kernel if and only if the kernel matrix

$$\boldsymbol{K}(i, j) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

is positive semidefinite for any set of data $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$.

- In practice, we do not always care if there is a corresponding Φ to the designed κ.
- Using κ with the best testing performance is often the guideline.

Example: The Polynomial Kernel

- The polynomial kernel up to degree m is defined by

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^m.$$

- We can always find a corresponding $\Phi : \mathbb{R}^d \to \mathbb{R}^p$ consists of polynomials of degrees at most m.
- This can be verified by using binomial theorem.

- More generally, we have the polynomial kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (a + b\boldsymbol{x}^\top \boldsymbol{x}')^m$$

to adjust scale.

Example: Gaussian / RBF kernel

Gaussian / Radial basis function (RBF) kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

- The corresponding $\Phi : \mathbb{R}^d \to \mathbb{R}^p$ has dimension $p = \infty$.
- This can be seen from Taylor's theorem. One dimensional example:

$$\kappa(x, x') = \exp\left(-(x - x')^2\right) = \exp(-x^2)\exp(x^2)\underbrace{\sum_{k=0}^{\infty} \frac{2^k x^k (x')^k}{k!}}_{\exp(2xx')}$$

- Hence, the RBF kernel can intuitively make any data linearly separable and regressible.

How to Use Kernel Trick: The Process

- Pick/design a kernel κ.
- Change the trainable parameters $\boldsymbol{\theta} = \sum_{j=1}^{n} \alpha_j \boldsymbol{x}_j$:

$$\boldsymbol{\theta}^\top \boldsymbol{x}_i = \sum_{i=1}^{n} \alpha_j \boldsymbol{x}_j^\top \boldsymbol{x}_i$$

  in loss function/decision rule.

- Applying the kernel trick: Replacing $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ with $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Learning Problems We Have Studied

General form:

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}).$$

- Least squares: $\ell_i(\boldsymbol{\theta}) = (\boldsymbol{\theta}^\top \boldsymbol{x}_i - y_i)^2$.
- Logistic regression: $\ell_i(\boldsymbol{\theta}) = \log(1 + \exp(-y_i \boldsymbol{\theta}^\top \boldsymbol{x}_i))$.
- SVM: $\ell_i(\boldsymbol{\theta}) = \max(0, 1 - y_i \boldsymbol{\theta}^\top \boldsymbol{x}_i) + \lambda \boldsymbol{\theta}^\top \boldsymbol{\theta}$.

If we invoke $\boldsymbol{\theta} = \sum_{j=1}^{n} \alpha_j \boldsymbol{x}_j$, we can 'kernalize' the above learning problems.

Let us take ℓ_2-regularized LS as an example to invoke kernel trick.

The Kernel Trick for Least Squares with Weight Decay

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \sum_{i=1}^{n} (\boldsymbol{\theta}^\top \boldsymbol{x}_i - y_i)^2 + \lambda \|\boldsymbol{\theta}\|_2^2$$

- We replace $\boldsymbol{\theta}$ by $\boldsymbol{\theta} = \sum_{j=1}^{n} \alpha_j \boldsymbol{x}_j$.

Kernelization:

$$\widehat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha} \in \mathbb{R}^n}{\text{argmin}} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \boldsymbol{x}_j^\top \boldsymbol{x}_i - y_i \right)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

$$\overset{\text{kernelize}}{=} \underset{\boldsymbol{\alpha} \in \mathbb{R}^n}{\text{argmin}} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - y_i \right)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$= \underset{\boldsymbol{\alpha} \in \mathbb{R}^n}{\text{argmin}} \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$$

Apply Kernel method for test data: With learned $\widehat{\boldsymbol{\alpha}}$ and a new test point $\boldsymbol{x}$, we have $y = f(\boldsymbol{x}) = \sum_{j=1}^{n} \widehat{\alpha}_j \kappa(\boldsymbol{x}_j, \boldsymbol{x})$, which is non-linear in $\boldsymbol{x}$.

Regularization Is Crucial for Kernel Methods

Kernelized problem:

$$\widehat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ and the kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$.

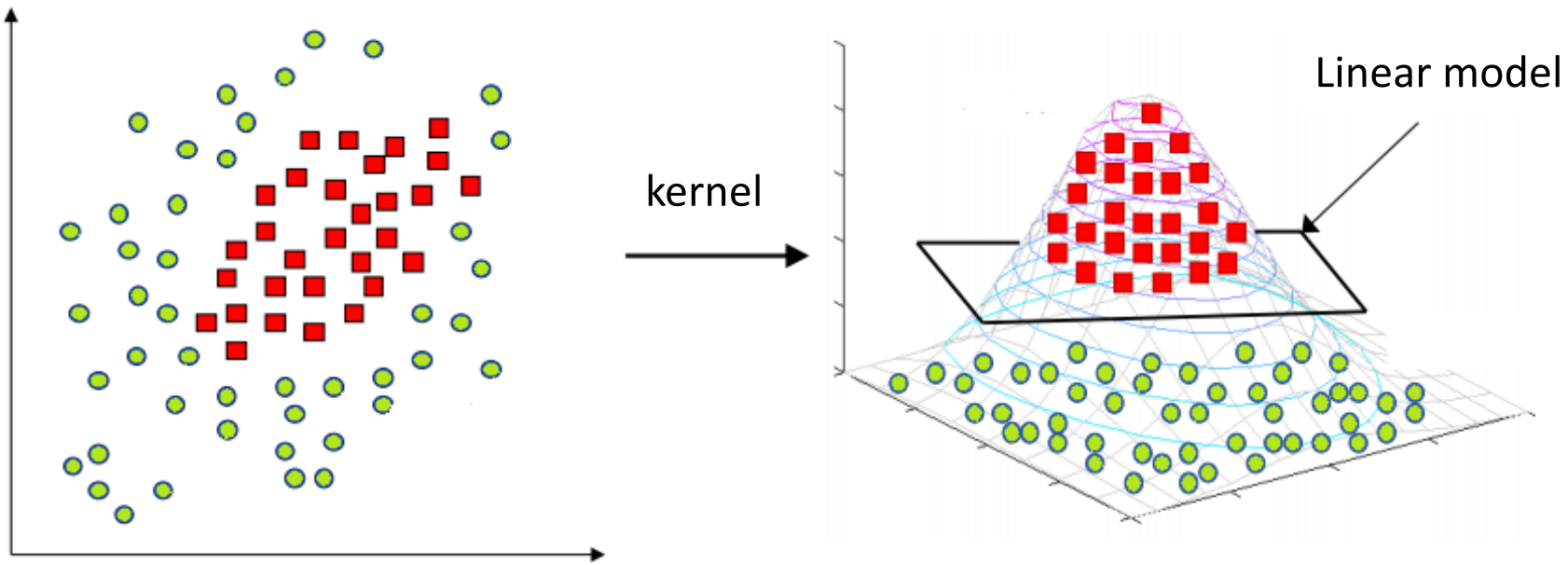
- What if $\lambda = 0$?
- Do you remember p? $\Phi : \mathbb{R}^d \to \mathbb{R}^p$. We often need $p > n$.
- We are indeed in dimension p.
- Thus, the solution

  $$\widehat{\boldsymbol{\alpha}} = \boldsymbol{K}^\dagger \boldsymbol{y}$$

  can overfit any data once $p > n$. $\rightsquigarrow$ This is always the case where Guassian / RBF kernel is used, as $p = \infty$.
- Summary: Kernel method is to use a as complex model as possible (with number of parameters p), then we use regularization to penalize it to the right model that does not have severe overfitting.

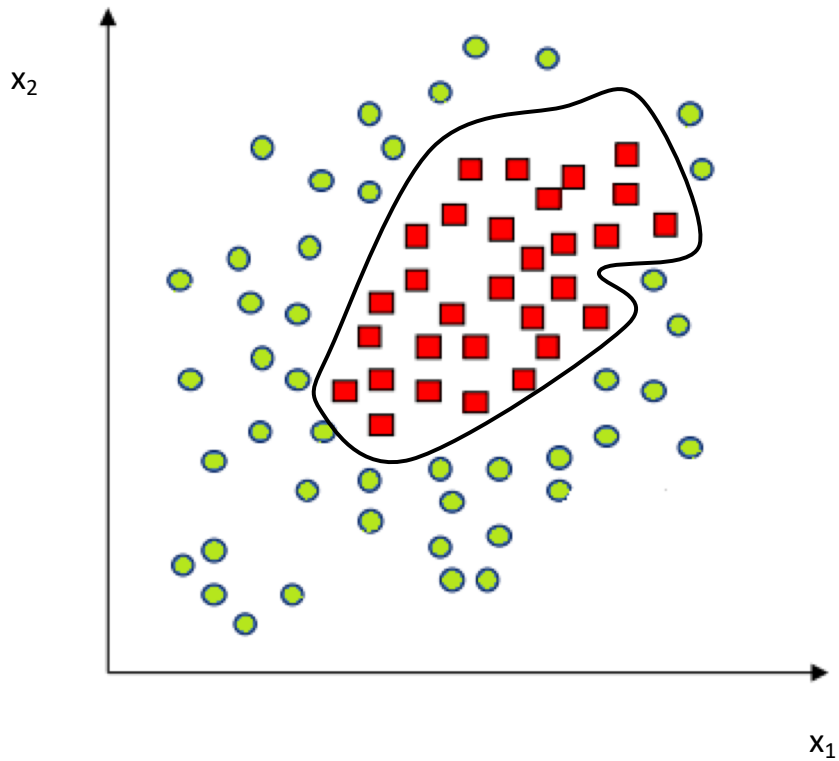
Applying Kernel Methods to Calcification

In Φ space: Linear in $\boldsymbol{\theta}$ and $\Phi(\boldsymbol{x})$.



Applying Kernel Methods to Calcification

Apply kernel method for test data: With learned $\widehat{\boldsymbol{\alpha}}$ and a new test point $\boldsymbol{x}$, we have $y \leftarrow f(\boldsymbol{x}) = \sum_{j=1}^{n} \widehat{\alpha}_j \kappa(\boldsymbol{x}_j, \boldsymbol{x})$, which is nonlinear in $\boldsymbol{x}$.



Kernel method can be viewed as using linear models to learn nonlinear classifier / regresser by non-linearly transforming the data — An intermediate method between linear and nonlinear models.

$\rightsquigarrow$ Next lecture: Unsupervised learning.

End of Linear Supervised Learning