IFI 9000 Analytics Methods
Support Vector Machines

by **Houping Xiao**

Spring 2021

Introduction

What is a separating hyperplane?

- It is a flat affine subspace of dimension $p-1$ in the p dimensional space
- It takes the following equation

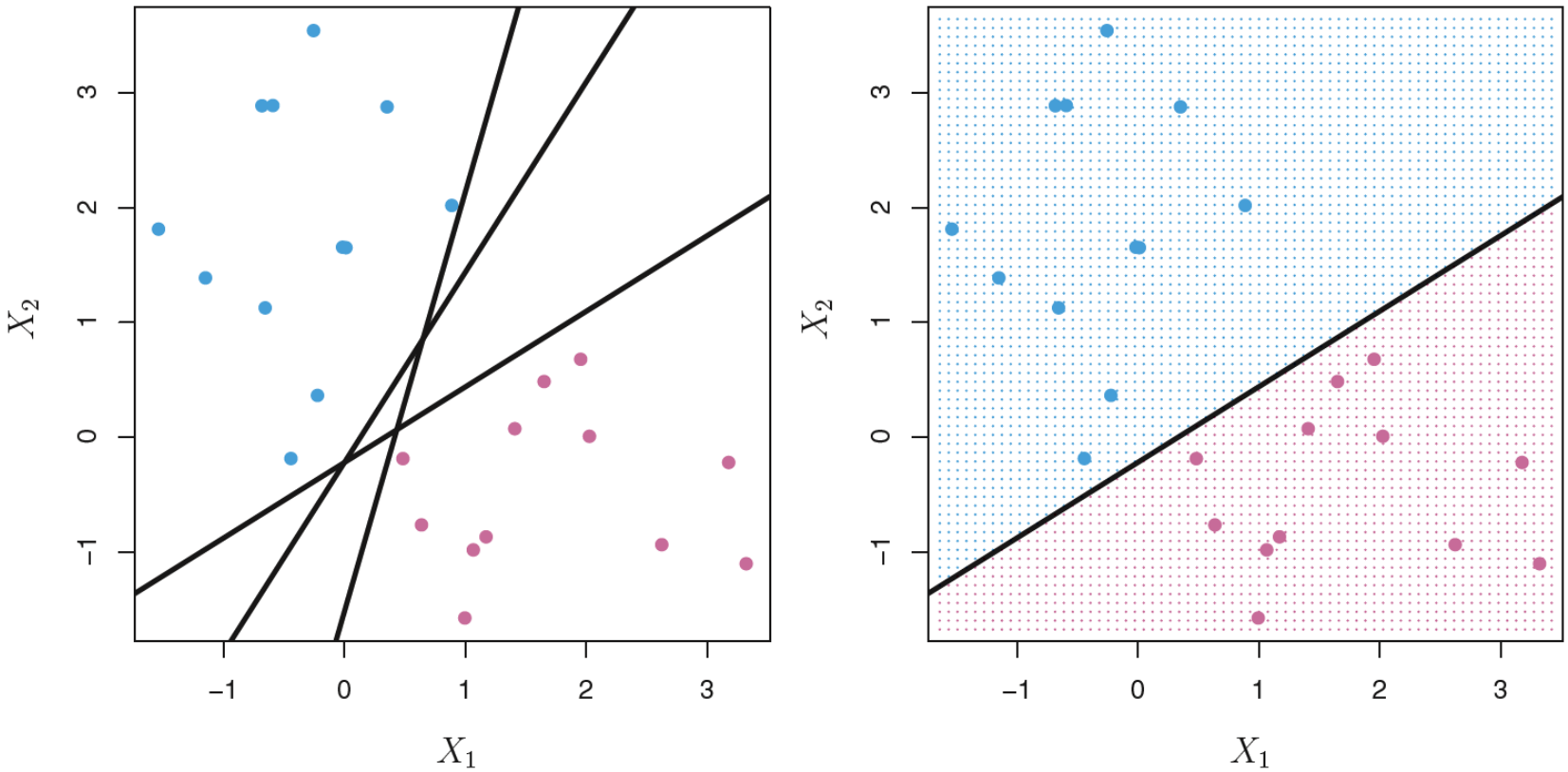
$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0.$$

- The vector $(\beta_1, \cdots, \beta_p)$ is the normal to the plane
- The sign of

$$f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

can be used as a class indicator for classification problems

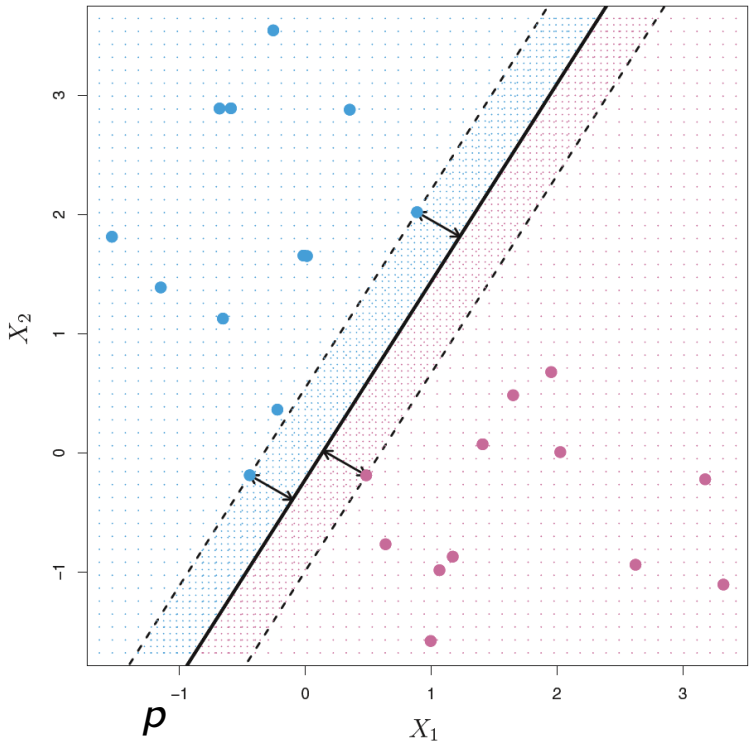
What is a separating hyperplane?



- Coding the two classes by color, we have $f(\boldsymbol{x}) > 0$ for one color and $f(\boldsymbol{x}) \leq 0$ for the other class. If we assign $y_i = 1$ or $y_i = -1$ to the classes, then a separating hyperplane will satisfy $y_i f(\boldsymbol{x}_i) > 0$, and the decision bounday is identified by $f(\boldsymbol{x}) = 0$.

Maximum Margin Classifier

- In selection of all possible separating hyperplanes, picking the ones that makes the **maximum symmetric gap** between the two classes **(maximum margin)**



$$\max_{\beta_0,\cdots,\beta_p} M \quad s.t. \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i,1} + \cdots + \beta_p x_{i,p}) \geq M, i = 1, \cdots, N$$

Convex Formulation of Maximum Margin Classifier

- The original problem

$$\max_{\beta_0,\cdots,\beta_p} M \quad s.t. \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i,1} + \cdots + \beta_p x_{i,p}) \geq M, i = 1, \cdots, N$$

can be written in vector notation as

$$\max_{\beta_0,\boldsymbol{\beta}} M \quad s.t. \quad ||\boldsymbol{\beta}||^2 = 1,$$

$$y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M, i = 1, \cdots, N$$

- To write the program in convex form, we can replace it with

$$\max_{\beta_0,\boldsymbol{\beta}} M \quad s.t. \quad \frac{y_i}{||\boldsymbol{\beta}||}(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M, \quad i = 1, \cdots, N$$

Convex Formulation of Maximum Margin Classifier

$$\max_{\beta_0,\boldsymbol{\beta}} M \quad s.t. \quad \frac{y_i}{||\boldsymbol{\beta}||}(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M, \quad i = 1, \cdots, N$$

in turn yields

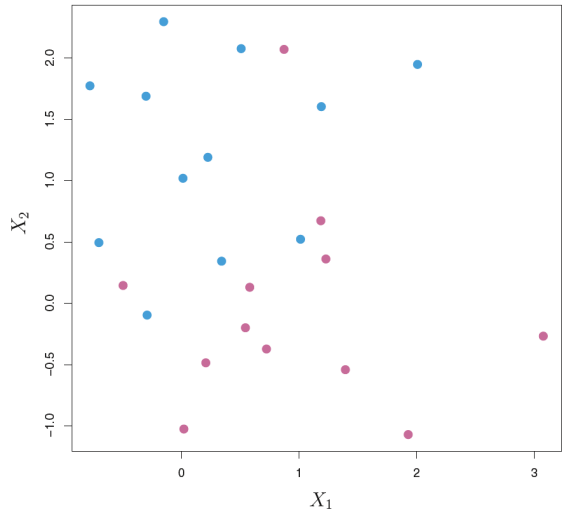
$$\max_{\beta_0,\boldsymbol{\beta}} M \quad s.t. \quad y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M||\boldsymbol{\beta}||, \quad i = 1, \cdots, N$$

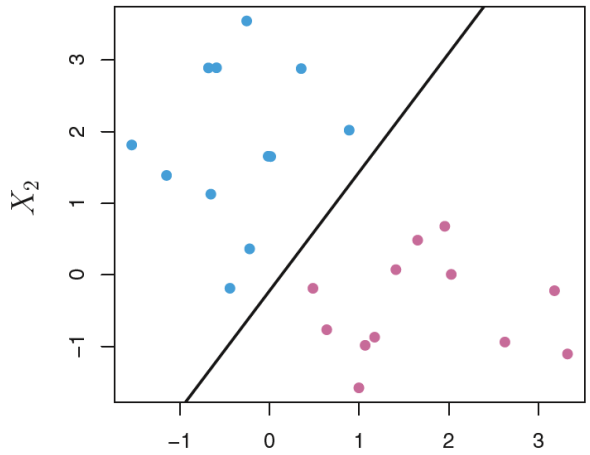
- We have a scaling ambiguity (if β_0, $\boldsymbol{\beta}$ are solutions, so are their scaled forms) we can pick $||\boldsymbol{\beta}|| = \frac{1}{M}$, which reduces the optimization to the convex form

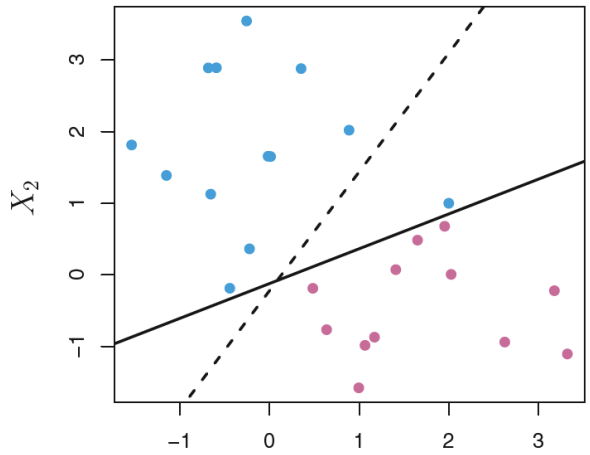
$$\max_{\beta_0,\boldsymbol{\beta}} ||\boldsymbol{\beta}|| \quad s.t. \quad y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq 1, \quad i = 1, \cdots, N$$

Non-ideal Cases

- In the majority of cases, we have non-separable or noisy data issues (here the model drastically changes due to the addition of a single point)

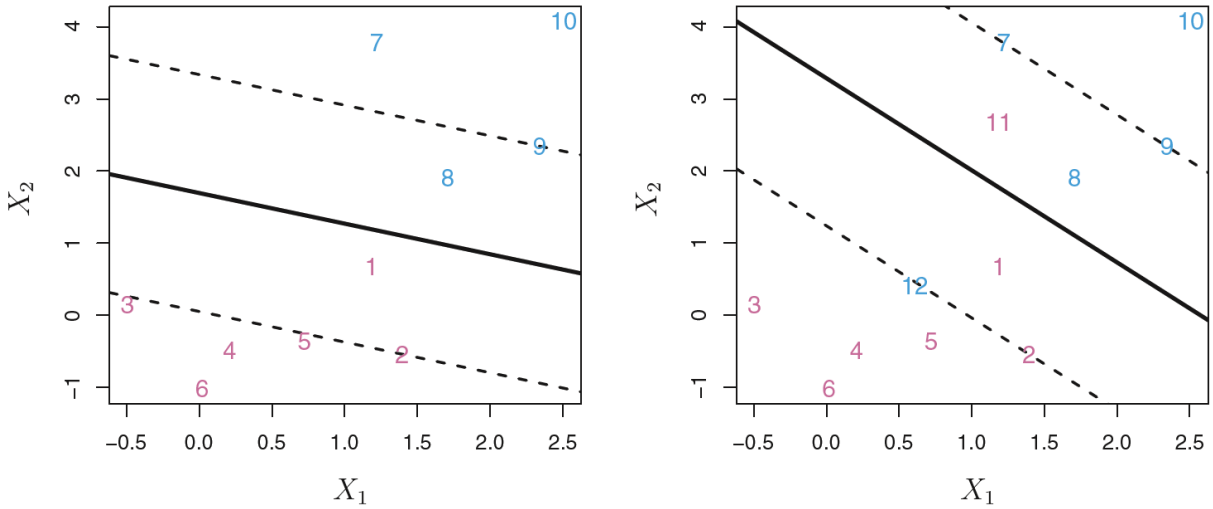






Support Vector Classifier

- Support vector classifier deals with the non-ideal issues by allowing some misclassification



- We find a **soft margin** that would include some misclassified points

$$\max_{\beta_0,\boldsymbol{\beta}} \quad M \quad s.t. ||\boldsymbol{\beta}||^2 = 1$$

$$y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M(1-\epsilon_i), i = 1, \cdots, N$$

$$\epsilon \geq 0, \sum_{i=1}^{N} \epsilon_i \leq C$$

Support Vector Classifier

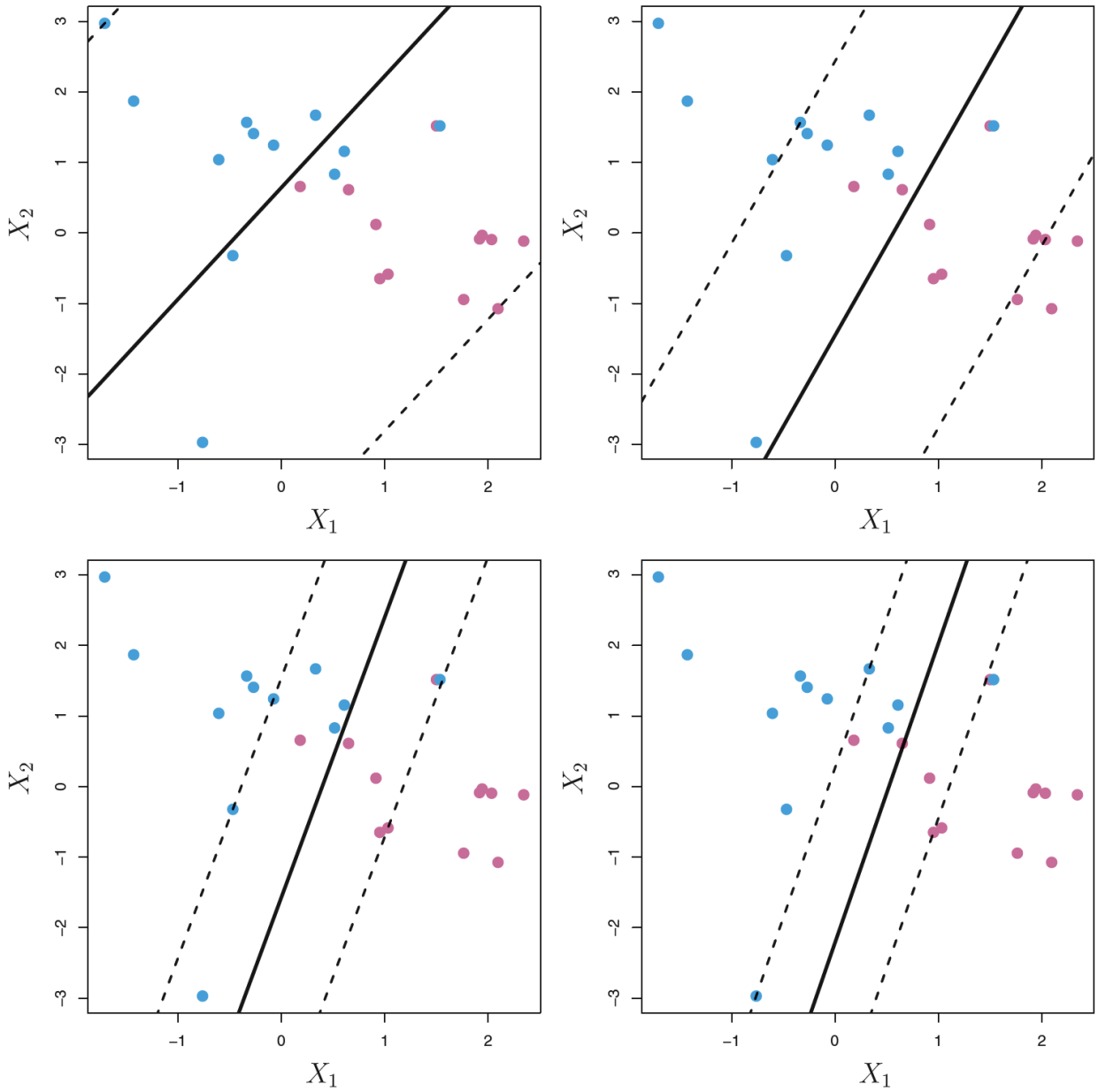
$$\max_{\beta_0,\boldsymbol{\beta},\epsilon_1,\cdots,\epsilon_N,M} \quad M \quad s.t. ||\boldsymbol{\beta}||^2 = 1$$

$$y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq M(1-\epsilon_i), i = 1, \cdots, N$$

$$\epsilon_i \geq 0, \sum_{i=1}^{N} \epsilon_i \leq C$$

The slack variables $\epsilon_1, \cdots, \epsilon_N$ would allow individual observations to be on the wrong side of the marginThe parameter C controls the width of the soft margin, larger C increases the tolerance for more misclassified points

Support Vector Classifier



Convex Formulation

- If we follow a similar procedure as before we end up with the following convex formulation

$$\max_{\beta_0,\boldsymbol{\beta},\epsilon_1,\cdots,\epsilon_N} ||\beta|| \quad s.t. \qquad y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq 1 - \epsilon_i, i = 1, \cdots, N$$

$$\epsilon_i \geq 0, \sum_{i=1}^{N} \epsilon_i \leq C$$

or in a regularized form as

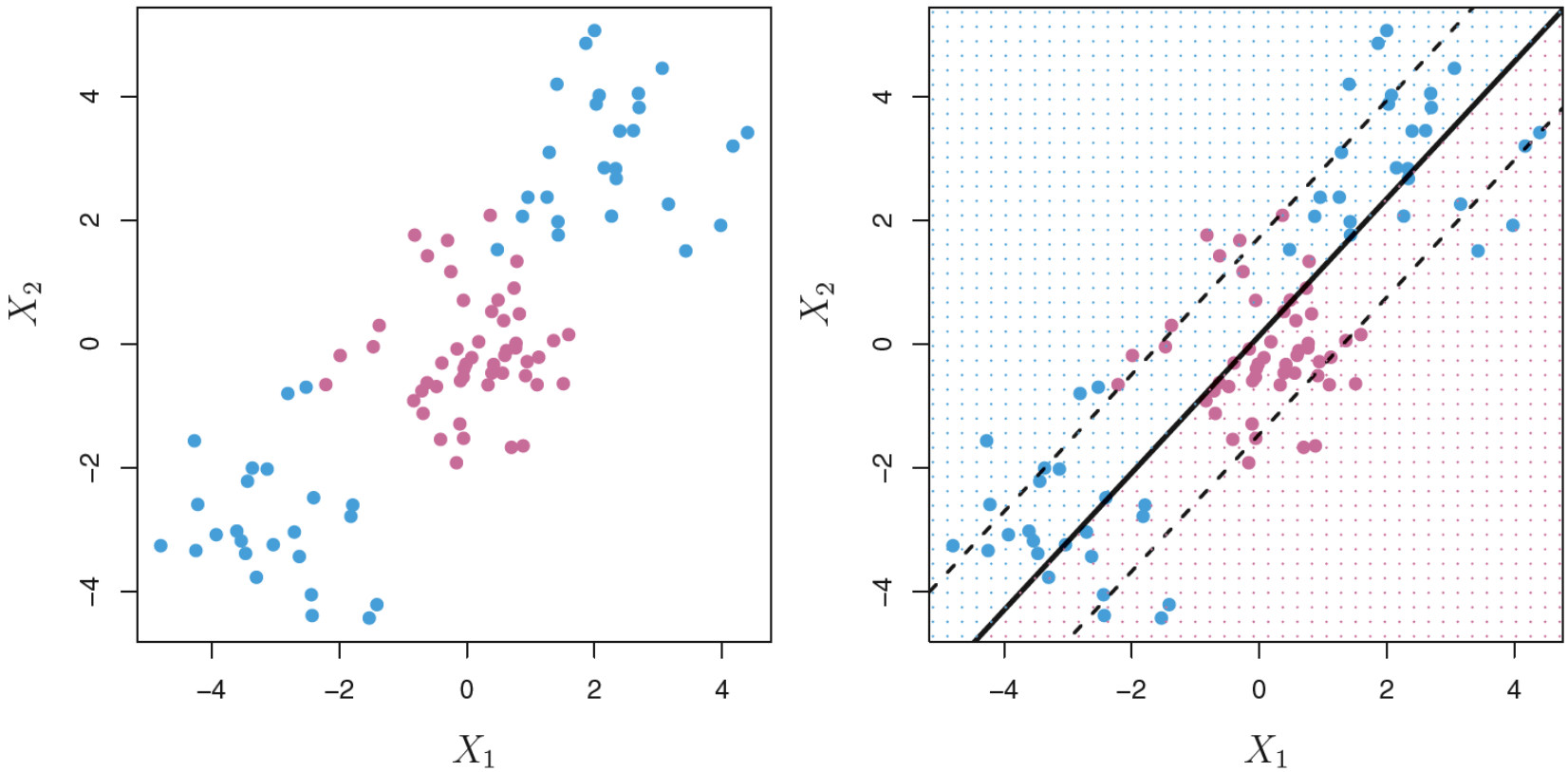
$$\max_{\beta_0,\boldsymbol{\beta},\epsilon_1,\cdots,\epsilon_N} ||\beta|| + \lambda \sum_{i=1}^{N} \epsilon_i \quad s.t. \quad y_i(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i) \geq 1 - \epsilon_i, i = 1, \cdots, N$$

$$\epsilon_i \geq 0, i = 1, \cdots, N$$

Try some coding demos!

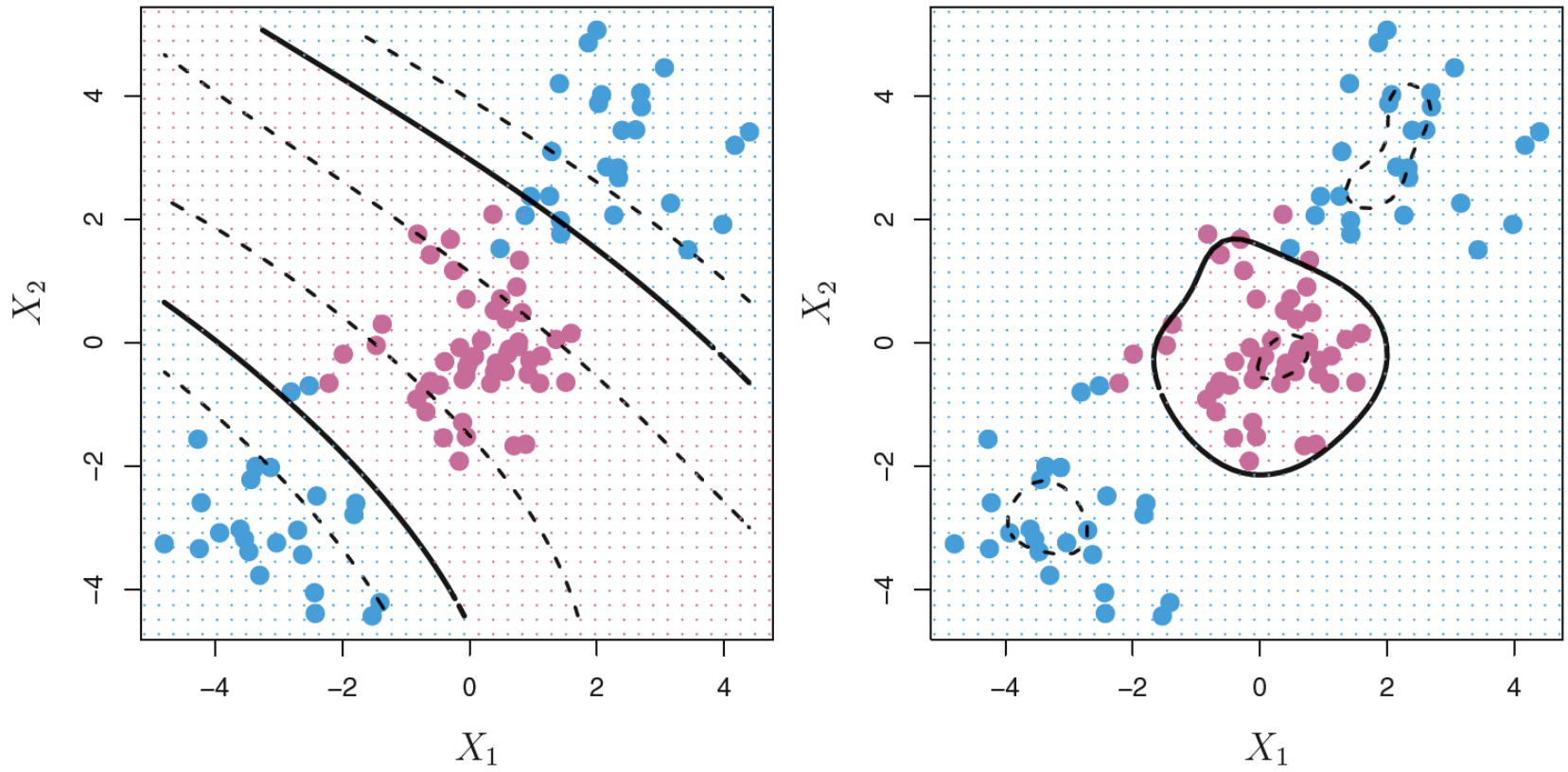
Generalization to Nonlinear Boundary

- Different classes cannot always be separated via a separating hyperplane



Feature Extension

- One remedy to handle nonlinear decision boundaries is adding nonlinear features to the variable list, e.g., $x_1, x_2, x_1^2, x_2^2, x_1x_2$, etc



- Features: $x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1^2x_2, x_1x_2^2$

Kernels

- Use of kernel is a more systematic way of introducing nonlinearities in SVM
- Generalization to kernels is easier to apply when working with the dual of the SVM convex program, however we skip such detail here
- Instead we look into an alternative way of representing the hyperplanes

$$f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}$$

or to incorporate all data points we can rewrite it as

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i^\top \boldsymbol{x}$$

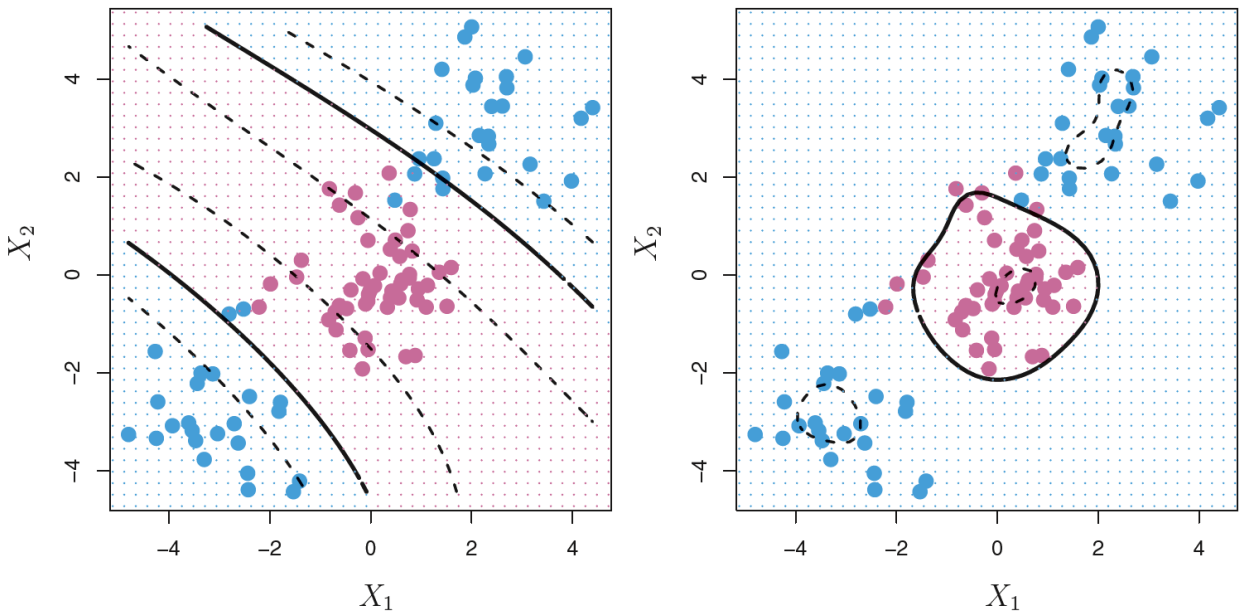
- In kernel methods, we replace $\boldsymbol{x}_i^\top \boldsymbol{x}$ with some (symmetric) function $k(\boldsymbol{x}_i, \boldsymbol{x})$:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{N} \alpha_i \cdot k(\boldsymbol{x}_i, \boldsymbol{x})$$

Radial Kernels

- Radial kernels are among the most robust and promising kernels used in SVM applications

$$k(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\gamma \sum_{j=1}^{p} (u_j - v_j)^2\right)$$



SVM: More Than Two Classes

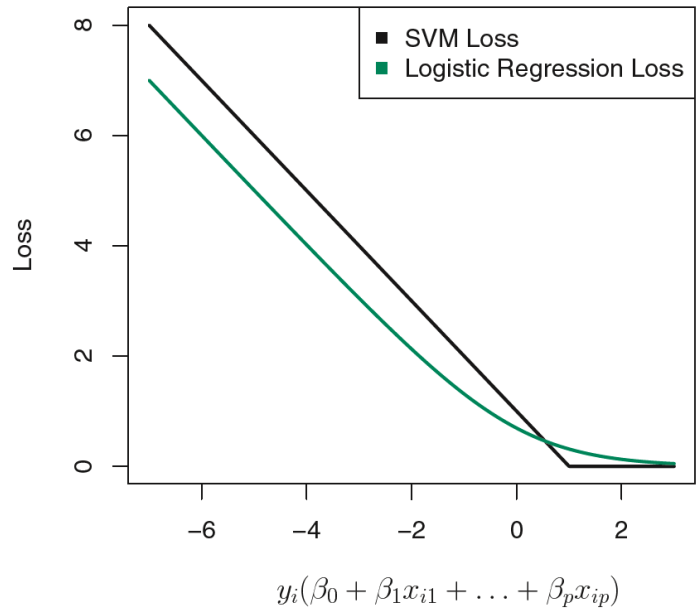
Here are two approaches we can use to classify $K > 2$ classes

- **One vs All (OVA):** Take one class, take remaining as the alternative class, repeat this for all k classes to get the classifiers $f_k(\boldsymbol{x})$. As the prediction for a sample $\boldsymbol{x}_t$ pick the class with largest $f_k(\boldsymbol{x}_t)$
- **One vs One (OVO):** Apply SVM to all combinations of two from the K classes, for each test point $\boldsymbol{x}_t$ count the number of times it lands on each class, then pick the max as the predicted class

OVO is usually more reliable, however for large number of classes OVA is computationally more desirable

SVM vs Logistic Regression

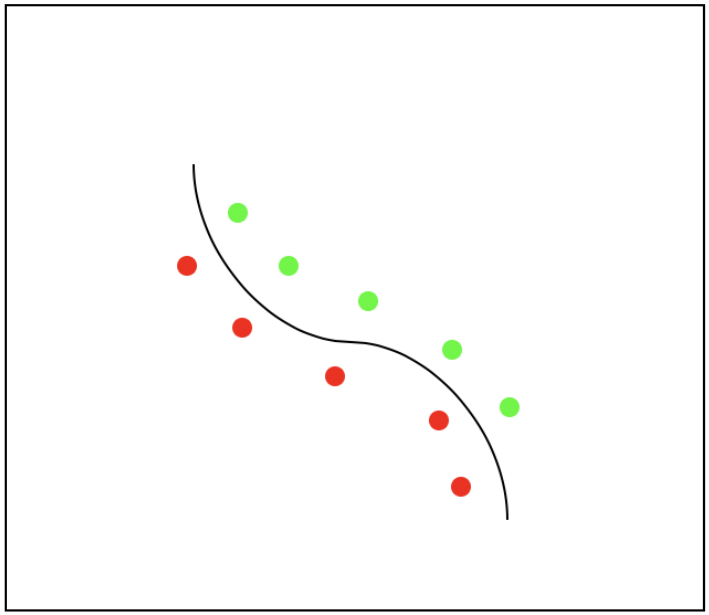
- The loss functions have some similarities when plotted as a function of $y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i$



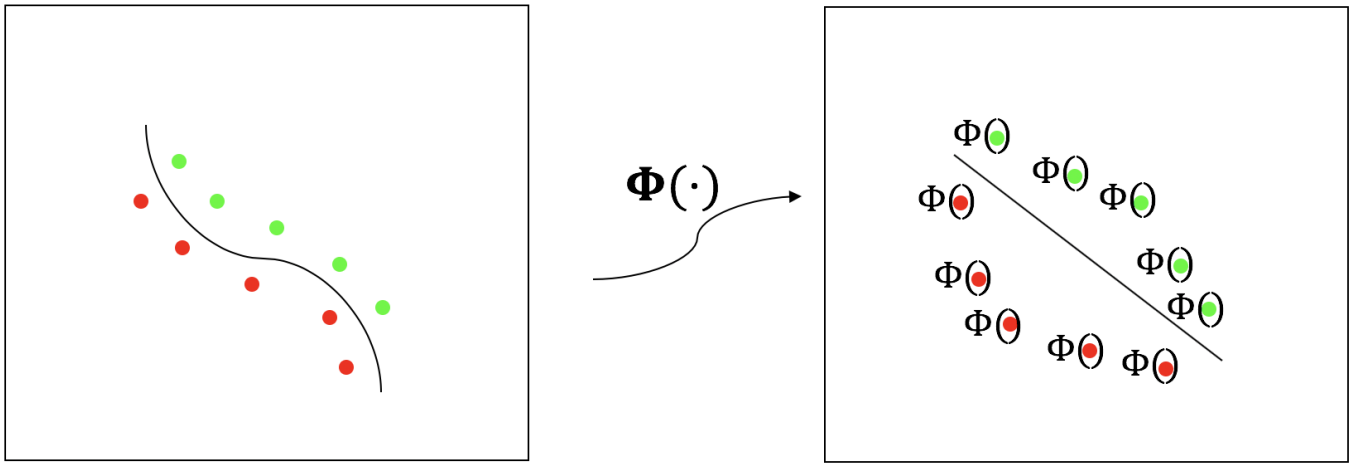
- LR becomes ill-posed when the classes are (almost) separable, LDA and SVM are fine
- LR would provide class probabilities
- SVMs easily generalize to nonlinear boundaries, LR and LDA are computationally harder

Why Kernel Methods are Needed?

- When the relations between dependent and independent variables are **nonlinear** in the original space, i.e., none hyperplane can be found to separate the data for classification problem in SVM



Why Kernel Methods are Needed?



- Kernel methods allow us to map the data (no separable in the original space) into a new space (possibly high-dimensional space) where data are linear separable
- Commonly used mapping:
  - $x \longrightarrow \{x, x^2\}$
  - $\{x_1, x_2\} \longrightarrow \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$

Kernel Functions

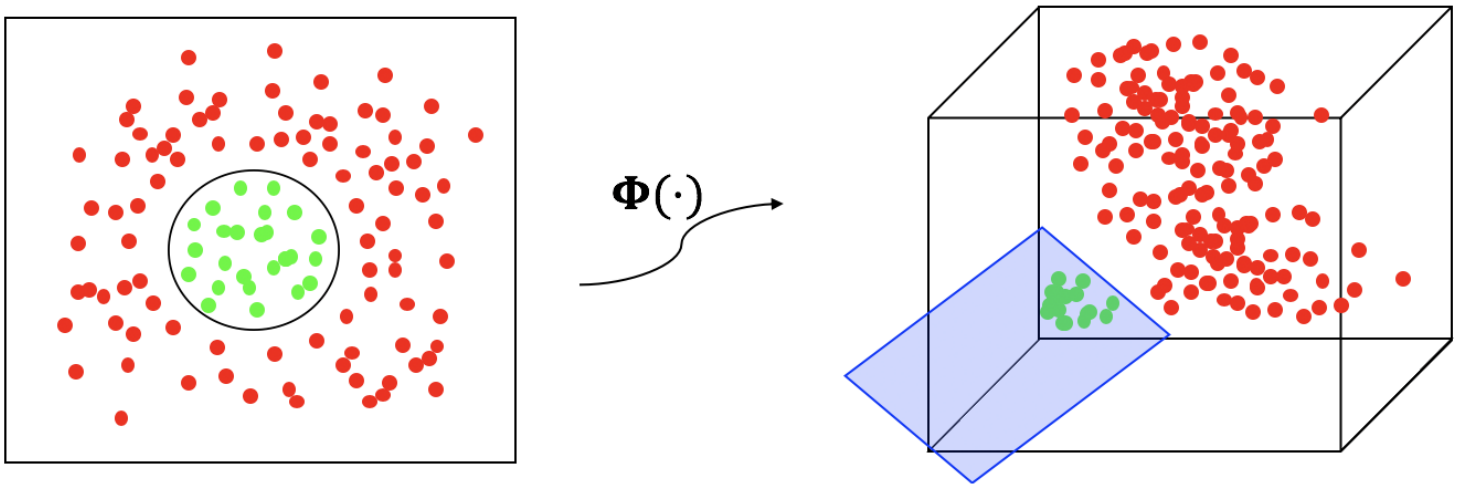
- **Computationally difficulty**: More features are in demand to capture the nonlinearity (i.e., higher dimensional representation)
- **Kernel functions**: Given $\boldsymbol{x}_i, \boldsymbol{x}_j$ in the original space $\mathbb{R}$, ϕ is a kernel function if

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$$

- $\phi(\boldsymbol{x}_i)$ is the coordinate representation in new space.
- The kernel function is a similarity measure; not need to calculate the $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ individually. We are more interested of their inner product

An Example of Kernel Methods

- $\phi : \boldsymbol{x} \in \mathbb{R}^2 \to \mathbb{R}^3$, $\Phi(x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2)$
- Easy to check $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^2 = f(\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$



- $\phi(x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. Left: input space, Right: feature space
- Different feature transformation can share the same kernel function
- $\psi(x_1, x_2) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$, then

$$\langle \psi(\boldsymbol{x}_i), \psi(\boldsymbol{x}_j) \rangle = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^2 = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Kernel Methods in Ridge Regression

- Ridge regression

$$\min_{\boldsymbol{\beta}} \quad \sum_{i=1}^{N} ||y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i||^2 + \lambda \sum_{i=1}^{N} \beta_i^2$$

- Prediction at $\boldsymbol{x}'$:

$$\begin{aligned} y^* &= \boldsymbol{\beta}^* \boldsymbol{x}' \\ &= \left((\lambda I_D + \boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \right)^\top \boldsymbol{x}' \\ &= \boldsymbol{y}^\top \left(\lambda I_N + \boldsymbol{X}\boldsymbol{X}^\top \right)^{-1} \boldsymbol{X}\boldsymbol{x}' \end{aligned}$$

Kernel Methods in Ridge Regression

- $y^* = \boldsymbol{y}^\top (\lambda I_N + \boldsymbol{X}\boldsymbol{X}^\top)^{-1} \boldsymbol{X}\boldsymbol{x}'$

$\boldsymbol{X}\boldsymbol{X}^\top$

$$\boldsymbol{X}\boldsymbol{X}^\top = \begin{pmatrix} \langle \boldsymbol{x}_1, \boldsymbol{x}_1 \rangle & \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle & \cdots & \langle \boldsymbol{x}_1, \boldsymbol{x}_N \rangle \\ \langle \boldsymbol{x}_2, \boldsymbol{x}_1 \rangle & \langle \boldsymbol{x}_2, \boldsymbol{x}_2 \rangle & \cdots & \langle \boldsymbol{x}_2, \boldsymbol{x}_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{x}_N, \boldsymbol{x}_1 \rangle & \langle \boldsymbol{x}_N, \boldsymbol{x}_2 \rangle & \cdots & \langle \boldsymbol{x}_N, \boldsymbol{x}_N \rangle \end{pmatrix}$$

$\boldsymbol{X}\boldsymbol{x}'$

$$\boldsymbol{X}\boldsymbol{x}' = \begin{pmatrix} \langle \boldsymbol{x}_1, \boldsymbol{x}' \rangle \\ \langle \boldsymbol{x}_2, \boldsymbol{x}' \rangle \\ \vdots \\ \langle \boldsymbol{x}_N, \boldsymbol{x}' \rangle \end{pmatrix}$$

Generalizing to Non-Linear Regression

- **Ridge regression**: consider the feature transformation $\phi = (\phi_1, \cdots, \phi_P)$ and

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_1) & \cdots & \phi_P(\boldsymbol{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \phi_2(\boldsymbol{x}_2) & \cdots & \phi_P(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_N) & \phi_2(\boldsymbol{x}_N) & \cdots & \phi_P(\boldsymbol{x}_N) \end{pmatrix}$$

- Making a prediction at

$$y^* = \boldsymbol{y}^\top (\lambda I_N + \mathbf{\Phi}\mathbf{\Phi}^\top)^{-1} \mathbf{\Phi} \phi(\boldsymbol{x}')$$

where $(\mathbf{\Phi}\mathbf{\Phi}^\top)_{ij} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$

What Is A Kernel Trick?

- Ridge Regression

$$y^* = \boldsymbol{y}^\top \left(\lambda I_N + \boldsymbol{X}\boldsymbol{X}^\top\right)^{-1} \boldsymbol{X}\boldsymbol{x}'$$

- Replace inner product with a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- Replace $\boldsymbol{X}\boldsymbol{X}^\top$ with the Gram matrix $\boldsymbol{K}$ where $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- Kernel Regression

$$y^* = \boldsymbol{y}^\top \left(\lambda I_N + \boldsymbol{K}\right)^{-1} k(\boldsymbol{X}, \boldsymbol{x}')$$

Kernel Trick

- We want to map the patterns into a high-dimensional feature space F and compare them using a dot product
- To avoid working in the F, choose a map function mapping to feature space where the dot product can be directly evaluated using a nonlinear function in the input space
- this is called the kernel trick

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$$

Kernel Methods in SVM

- Recall the solution of dual problem:

$$\hat{\boldsymbol{\beta}} = \sum_{i=1}^{N} \hat{\alpha}_i y_i \boldsymbol{x}_i$$

- Decision rule:

$$\hat{y} = sign\left(\sum_{i \in SV} \hat{\alpha}_i y_i (\boldsymbol{x}_i^\top \boldsymbol{x}) + \beta_0 \right)$$

- Using kernel would give us:

$$\hat{y} = sign\left(\sum_{i \in SV} \hat{\alpha}_i y_i \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle + \beta_0 \right)$$

How to Construct a Kernel?

- The simplest kernel function we have is that

$$k(\boldsymbol{x}_i, \boldsymbol{x}_i) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

- We start with basis functions

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$$

- Direct design
  - Measure similarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
  - Gram matrix must be positive semi-definite
  - k should be symmetric
- Creating more complicated kernels based on the known kernels k_1 and k_2
  - $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = ck_1(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = f(\boldsymbol{x}_i)k_1(\boldsymbol{x}_i, \boldsymbol{x}_j)f(\boldsymbol{x}_j)$
  - $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = q(k_1(\boldsymbol{x}_i, \boldsymbol{x}_j))$, q is polynomial
  - $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k_1(\boldsymbol{x}_i, \boldsymbol{x}_j) + k_2(\boldsymbol{x}_i, \boldsymbol{x}_j), k_1(\boldsymbol{x}_i, \boldsymbol{x}_j)k_2(\boldsymbol{x}_i, \boldsymbol{x}_j)$

Popular Kernels

- Mercer Kernel: $\boldsymbol{K}$ is positive definite
- Gaussian Kernel or Radial Basis Function (RBF)

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2\right)$$

  Mapping inputs to an infinite dimensional space
- Cosine Similarity

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_i}{||\boldsymbol{x}_i||||\boldsymbol{x}_i||}$$

- Probabilistic Kernel Functions

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = Pr(\boldsymbol{x}_i|\boldsymbol{\theta})Pr(\boldsymbol{x}_j|\boldsymbol{\theta})$$

  Two inputs are more similar if both have high probabilities
- Bayesian Kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \int Pr(\boldsymbol{x}_i|\boldsymbol{\theta})Pr(\boldsymbol{x}_j|\boldsymbol{\theta})Pr(\boldsymbol{\theta})d\boldsymbol{\theta}$$

Kernel Machines

- We can use kernel function to generate new features. Apply the kernel function for each input and a set of P centroids

$$\Phi(\boldsymbol{x}) = [k(\boldsymbol{x}, \boldsymbol{\mu}_1), k(\boldsymbol{x}, \boldsymbol{\mu}_1), \cdots, k(\boldsymbol{x}, \boldsymbol{\mu}_2)]$$

- How to choose centroids $\boldsymbol{\mu}_i$
  - Randomly selection
  - Clustering
  - Using all available data points

The End