Kernel Models

MLI
Module 1, Lecture 5
18th May 2021

[Adriano Koshiyama]

Overview

- **Back to Feature Maps**
  - Overview
  - Back to Ridge Regression
  - Explicit Feature Maps
  - Kernel Ridge Regression
  - Kernel Trick
  - Pros and cons of the dual representation
- **Kernels**
  - Definition
  - Kernelization
  - Some key facts and results
  - A bestiary of kernels
- **Support Vector Machines**
  - Main Idea and Details
  - Linear Support Vector Machines
  - Kernel/Nonlinear Support Vector Machines
  - Kernel selection and fine-tuning

© 2021 MLI

Back to Feature Maps

- Overview
- Back to Ridge Regression
- Explicit Feature Maps
- Kernel Ridge Regression
- Kernel Trick
- Pros and cons of the dual representation

Overview

- We show how a linear method such as least squares may be lifted to a (potentially) higher dimensional space to provide a nonlinear regression.

Map // Selection	Explicit	Implicit
Explicit	Basis Functions with Stepwise	Regularization
Implicit	Kernelization with Stepwise	**Kernel Methods with Regularization**

- **Implicit** Selection are widely adopted in Machine Learning
- **Explicit** Mapping is not very systematic, being more problem-dependent, but some recipes are universal (interactions, z-scores, etc.)
- **Implicit** Mapping with **Implicit** Feature selection is **Our Focus Today!**
- **Implicit** mapping with **Explicit** feature selection is not generally used

Back to Ridge Regression

- If we consider the Ridge Regression case, we have

$$\min_{\boldsymbol{\beta}(\lambda)} \boldsymbol{MSE}[\boldsymbol{\beta}(\lambda)] = \left|\left|\boldsymbol{y} - \boldsymbol{X\beta}\right|\right|_2^2 + \lambda\left|\left|\boldsymbol{\beta}\right|\right|_2^2, \text{ for a given } \lambda \geq 0$$

- we can rewrite this expression as

$$\min_{\boldsymbol{\beta}(\lambda)} \boldsymbol{MSE}[\boldsymbol{\beta}(\lambda)] = \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{y} + \boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{X\beta} + \lambda\boldsymbol{\beta}^T\boldsymbol{I}_{JxJ}\boldsymbol{\beta}$$

- First order condition: $\nabla_{\boldsymbol{\beta}(\lambda)} \boldsymbol{MSE}[\boldsymbol{\beta}(\lambda)] = 0$, then

$$-2\boldsymbol{X}^T\boldsymbol{y}^T + \boldsymbol{X}^T\boldsymbol{X\beta} + \lambda\boldsymbol{I}_{JxJ}\boldsymbol{\beta} = \boldsymbol{0} \rightarrow -2\boldsymbol{X}^T\boldsymbol{y} + (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_{JxJ})\boldsymbol{\beta} = \boldsymbol{0}$$

- Yielding the solution, akin to Linear Regression, but with a new param

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_{JxJ}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- Second order condition: $\boldsymbol{H}[\boldsymbol{MSE}[\boldsymbol{\beta}(\lambda)]]$ is positive semi-definite

$$\boldsymbol{H}\left[\boldsymbol{MSE}[\boldsymbol{\beta}(\lambda)]\right] = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_{JxJ}\right)^{\boldsymbol{T}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_{JxJ}\right)$$

since $\boldsymbol{X}^T\boldsymbol{X}$ is positive semi-definite, $\boldsymbol{I}_{JxJ}$ is positive definite, the hessian matrix will be positive semi-definite for a suitable choice of λ (usually $\lambda \geq 0$)

PS: remember to scale $\boldsymbol{X}$ columns (like having zero mean and unit std)

Explicit Feature Maps

- Overall, the main idea is to extend the model, including new features engineered using some **nonlinear mapping** (a.k.a. basis function)

$$y_i = \boldsymbol{\alpha}^T \phi(\boldsymbol{x}_i) + \varepsilon_i$$

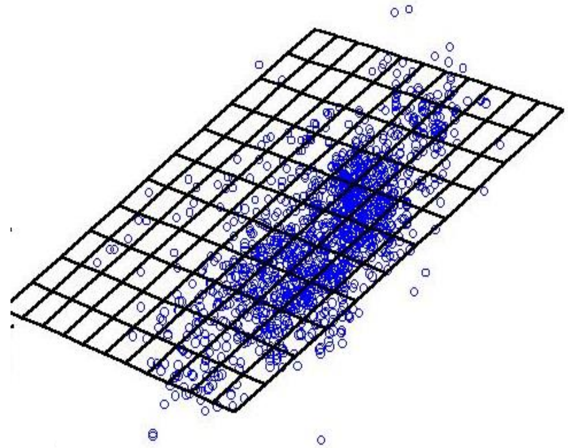
with $\phi: S^J \to \mathcal{R}^M$, usually $M > J$; hence more coefficients have to be fitted

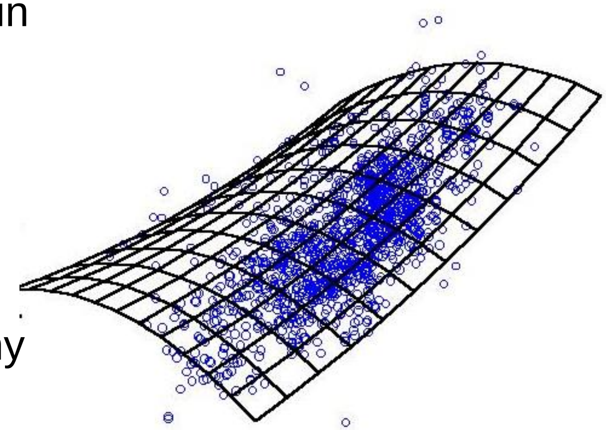
- Since the model is linear in the parameters $\boldsymbol{\alpha}^T$, we can call $\phi(\boldsymbol{x}_i) = \boldsymbol{z_i}$ and use the usual solution $\boldsymbol{w} = \left(\boldsymbol{Z^T Z}\right)^{-1} \boldsymbol{Z^T y}$, with $\boldsymbol{Z} = [\boldsymbol{z_1}, \dots, \boldsymbol{z_n}]^{\boldsymbol{T}}$

- Some commonly used feature maps, adopted in many situations
  - Polynomial: ϕ: $x_i \to (1, x_i, x_i^2, x_i^3, \dots)$
  - Veronese/ANOVA/Interaction mapping

ϕ: $x_{i1}, x_{i2} \to (1, x_{i1}, x_{i2}\ x_{i1}^2, x_{i2}^2, x_{i1} x_{i2}, \dots)$

- In fact, log, sqrt, one-hot encoding, etc. and any other transformation





Kernel Ridge Regression

- Is it possible to create polynomial features, without incurring in all the additional calculations, memory storage, etc.? **A: Yes**
- **First step**: rewrite Ridge Regression solution by

$$\boldsymbol{X^T X \beta} + \lambda \boldsymbol{I}_{JxJ}\boldsymbol{\beta} = \boldsymbol{X^T y} \Rightarrow \lambda \boldsymbol{I}_{JxJ}\boldsymbol{\beta} = \boldsymbol{X^T y} - \boldsymbol{X^T X \beta} \Rightarrow \boldsymbol{\beta} = \left(\frac{1}{\lambda}\right)\boldsymbol{X^T}[\boldsymbol{y} - \boldsymbol{X\beta}]$$

hence, $\boldsymbol{\beta}$ can be expressed as a linear combination of the columns of $\boldsymbol{X^T}$

$$\boldsymbol{\beta} = \boldsymbol{X^T \alpha}$$

with $\boldsymbol{\alpha} \in \mathbb{R}^n$ being the dual coefficient vector; further manipulation show

$$\boldsymbol{\alpha} = \left(\frac{1}{\lambda}\right)[\boldsymbol{y} - \boldsymbol{X\beta}] \Rightarrow \boldsymbol{\alpha}\lambda = [\boldsymbol{y} - \boldsymbol{XX^T \alpha}] \Rightarrow (\boldsymbol{XX^T} + \lambda \boldsymbol{I}_{nxn})\boldsymbol{\alpha} = \boldsymbol{y}$$

- therefore, $\boldsymbol{\alpha} = \left(\boldsymbol{XX^T} + \lambda \boldsymbol{I}_{nxn}\right)^{-\mathbf{1}}\boldsymbol{y}$, implying that $\boldsymbol{\alpha}$
  - minimizes $\boldsymbol{MSE}[\boldsymbol{\alpha}(\lambda)] = \left|\left|\boldsymbol{y} - \boldsymbol{XX^T\alpha}\right|\right|_{\mathbf{2}}^{\mathbf{2}} + \lambda\, \boldsymbol{\alpha^T XX^T \alpha}$
  - written in terms of inner products of rows ($\boldsymbol{XX^T}$) not columns ($\boldsymbol{X^T X}$)

Kernel Trick

- **Second step**: since $\boldsymbol{\alpha}$ is written in terms of inner products of rows

$$\boldsymbol{X}\boldsymbol{X}^T = \begin{bmatrix} \langle \boldsymbol{x_1}, \boldsymbol{x_1} \rangle & \cdots & \langle \boldsymbol{x_1}, \boldsymbol{x_n} \rangle \\ \vdots & \ddots & \vdots \\ \langle \boldsymbol{x_n}, \boldsymbol{x_1} \rangle & \cdots & \langle \boldsymbol{x_n}, \boldsymbol{x_n} \rangle \end{bmatrix}$$

- if, we consider a feature map of $\boldsymbol{x_i}$, $\boldsymbol{\phi}: S^J \rightarrow \mathcal{R}^M$, we can rewrite the Gram-Matrix $\boldsymbol{X}\boldsymbol{X}^T$ after feature mapping by

$$. = \begin{bmatrix} \langle \boldsymbol{\phi}(\boldsymbol{x_1}), \phi(\boldsymbol{x_1}) \rangle & \cdots & \langle \boldsymbol{\phi}(\boldsymbol{x_1}), \boldsymbol{\phi}(\boldsymbol{x_n}) \rangle \\ \vdots & \ddots & \vdots \\ \langle \boldsymbol{\phi}(\boldsymbol{x_n}), \boldsymbol{\phi}(\boldsymbol{x_1}) \rangle & \cdots & \langle \boldsymbol{\phi}(\boldsymbol{x_n}), \boldsymbol{\phi}(\boldsymbol{x_n}) \rangle \end{bmatrix} = \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \cdots & k(\boldsymbol{x_1}, \boldsymbol{x_n}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x_n}, \boldsymbol{x_1}) & \cdots & k(\boldsymbol{x_n}, \boldsymbol{x_n}) \end{bmatrix} = \boldsymbol{K}$$

- where $k(\boldsymbol{x_k}, \boldsymbol{x_i}) \coloneqq \langle \boldsymbol{\phi}(\boldsymbol{x_k}), \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle$ is a (**kernel**) function that **replicates** a certain **feature mapping plus an inner product**!
- Example: consider that $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^{\mathbf{2}}$ (think as two rows of a 2-d dataset) and $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2$, then $k(\boldsymbol{x}, \boldsymbol{z}) = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 = \cdots$

$$\ldots = \left[x_1^2\,, x_2^2, \sqrt{2} x_1 x_2\right]^T \left[z_1^2\,, z_2^2, \sqrt{2} z_1 z_2\right] = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z}) \rangle$$

for a map $\boldsymbol{\phi}(\boldsymbol{x}) = [x_1^2, x_2^2, \sqrt{2} x_1 x_2]$, close to a second degree polynomial

- hence,$\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z}) \rangle$ takes roughly $O(n^2)$ whilst $k(\boldsymbol{x}, \boldsymbol{z})$ takes $O(n)$

Dual Representation – Pros and Cons

- **Primal Form: Computation with Basis Functions**
  - **Training**: $O(J^p)$ for feature mapping and $O(nJ^2 + J^3)$ for estimation
    - Dataset $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$
    - Basis functions: ϕ_1, ..., ϕ_k where ϕ: $\mathbb{R}^J \rightarrow \mathbb{R}$
    - Feature map: $\boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \dots, \phi_k(\boldsymbol{x})]$
    - Solution: $\boldsymbol{\beta} = \left(\boldsymbol{Z^T Z}\right)^{-1} \boldsymbol{Z^T y}$, with $\boldsymbol{Z} = [\boldsymbol{\phi}(\boldsymbol{x_1}), \dots, \boldsymbol{\phi}(\boldsymbol{x_n})]^{\boldsymbol{T}}$
  - **Prediction**: $\hat{y}(\boldsymbol{x_i}) = \boldsymbol{\beta^T \phi}(\boldsymbol{x_i})$, taking $O(k)$ operations
- **Dual Form: Implicit Feature Map**
  - **Training**: $O(Jn)$ to create $\boldsymbol{K}$ and $O(Jn^2 + n^3)$ for estimation
    - Dataset $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$
    - Kernel Function: k: $\mathbb{R}^J \times \mathbb{R}^J \rightarrow \mathbb{R}$
    - Kernel Matrix: $\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \cdots & k(\boldsymbol{x_1}, \boldsymbol{x_n}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x_n}, \boldsymbol{x_1}) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix}$
    - Solution: $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I}_{nxn})^{-\boldsymbol{1}} \boldsymbol{y}$
  - **Prediction**: $\hat{y}(\boldsymbol{v}) = \sum_{\boldsymbol{i}} \alpha_i \, k(\boldsymbol{x_i}, \boldsymbol{v})$, taking $O(nJ)$ operations

Kernels

- Definition
- Kernelization
- Some key facts and results
- A bestiary of kernels

© 2021 MLI

Definition

- **Kernel**
  - A function k: $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a kernel on $\mathcal{X}$ if there exists a Hilbert space $\mathcal{F}$ and a map $\boldsymbol{\phi}$: $\mathcal{X} \rightarrow \mathcal{F}$, such that $k(x, z) = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z}) \rangle_{\mathcal{F}}$

- In our more usual terminology
  - $\boldsymbol{\phi}$: $\mathcal{X} \rightarrow \mathcal{F}$ is called a **feature map**
  - $\mathcal{F}$ is called a **feature space**

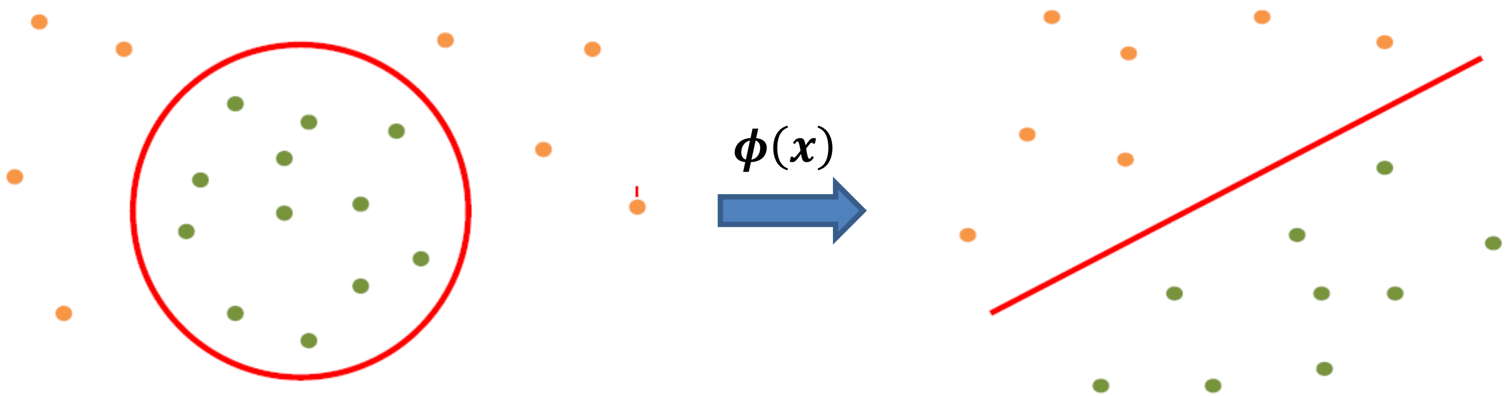
- Hence, a kernel function is a composition of "feature map" with scalar product. Our previous example, $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^2$, we have:
  - $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^{\mathbf{2}}$ (homogenous 2-degree polynomial kernel)
  - $\boldsymbol{\phi}(\boldsymbol{x}) = [x_1^2, x_2^2, \sqrt{2} x_1 x_2]$ or $\boldsymbol{\phi}(\boldsymbol{x}) = [x_1^2, x_2^2, x_1 x_2, x_1 x_2]$

is one of the most common instantiations of a kernel function

- **Note that a kernel function is not always is associated with an unique feature map!**

Kernelization

- Kernels allow to make linear algorithms work on non-linear features, by replicating a very complicated feature mapping by a kernel function over the raw features
- Example: Support Vector Machine: $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}, y_i \in \{-1,1\}$



$$f(\boldsymbol{x}) = \operatorname{sgn}(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x})$$

$\boldsymbol{\beta} = \sum_i \alpha_i \boldsymbol{x}_i y_i$ solves a QP involving $\boldsymbol{X^T X}$, with α_i representing “dual variables

$$f(\boldsymbol{z}) = \operatorname{sgn}\left(\beta_0 + \sum_i \alpha_i\, k(\boldsymbol{x}_i, \boldsymbol{z})\right)$$

$$= \boldsymbol{\alpha^T k(X, z)}$$

$\boldsymbol{\alpha}$ solves a QP involving kernel matrix $\boldsymbol{K}$

Key facts and results

- **What functions are kernels?**

- Given a function k: $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, which properties of k guarantee that there exists a Hilbert space $\mathcal{F}$ and a map $\boldsymbol{\phi}$: $\mathcal{X} \rightarrow \mathcal{F}$, such that $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z}) \rangle_{\mathcal{F}}$?
- **Subtle note**: we have generalized the definition of **finite-dimensional** feature maps $\boldsymbol{\phi}$: $\mathbb{R}^J \rightarrow \mathbb{R}^m$ to **infinite-dimensional** feature maps $\boldsymbol{\phi}$: $\mathbb{R}^J \rightarrow \mathcal{F}$ (Hilbert space)

- **First**: a function h: $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ **is positive semidefinite** if is symmetric and the matrix $\boldsymbol{H}$ made of $\boldsymbol{H}_{ixj} = h(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for every *i* and *j* is positive semidefinite ($\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v} \geq 0$)
- **Theorem (Positive Semidefinite Kernels)**
  - A function k is positive semidefinite if and only if

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z}) \rangle,\ \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^J$$

for some feature map $\boldsymbol{\phi}$: $\mathbb{R}^J \rightarrow \mathcal{F}$ and Hilbert space $\mathcal{F}$

Key facts and results

- **Theorem (Moore-Aronszajn, 1950)**
- For every positive semidefinite kernel $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there is an unique feature map $\boldsymbol{\phi}\colon \mathcal{X} \to \mathcal{F}$ (up to a isomorphism) into an unique Hilbert space $\mathcal{F}$ (the so-called Reproducing Kernel Hilbert Space – RKHS)
  - **RKHS**: $\boldsymbol{\phi}\colon \boldsymbol{x} \to \boldsymbol{k}(\boldsymbol{x},.)$ and $\langle \boldsymbol{k}(\boldsymbol{x},.), \boldsymbol{f}(.)\rangle_{\mathcal{F}} = f(\boldsymbol{x})$, $\forall f \in \mathcal{F}$
- **Theorem (Kimeldorf-Wahba, 1971) "Representer Theorem"**
  - Let $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$ be data points
  - Let k be a positive semidefinite kernel with RHKS $\mathcal{F}$
  - Let $\mathcal{L}\colon \mathbb{R}^J \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be any loss function
  - Let $\Omega\colon [0, \infty) \to \mathbb{R}$ strictly increasing penalty function
- Then, there is a minimizer of the type $f(x) = \alpha^T \boldsymbol{k}(x,.)$ to $R(f) = \sum_i \mathcal{L}\left(\boldsymbol{x}_i, y_i, f(\boldsymbol{x}_i)\right) + \Omega\left(||f||_{\mathcal{F}}\right)$

A bestiary of kernels

- **Linear**: $k(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{z}$ (usually $\boldsymbol{A}$ is a positive definite matrix)
- **Dot-product**: $k(\boldsymbol{x},\boldsymbol{z}) = f(\langle \boldsymbol{x},\boldsymbol{z}\rangle)$ (usually f is a positive definite)
- **Polynomial**: $k(\boldsymbol{x},\boldsymbol{z}) = (\theta\langle \boldsymbol{x},\boldsymbol{z}\rangle + c)^d$ (general formulation)
- **Sigmoid**: $k(\boldsymbol{x},\boldsymbol{z}) = tanh(\sigma\langle \boldsymbol{x},\boldsymbol{z}\rangle + c))$ (not positive definite…)
- **Radial basis function kernels**

Gaussian/RBF Kernel

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

Laplacian Kernel

$$k(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|}{\sigma}\right)$$

- **Proposition**
- For positive definite kernels $k_1, k_2, \ldots$ and $c_i > 0$, any of the composition below is also a positive definite kernel

$$k_1(\boldsymbol{x},\boldsymbol{z}) + c_1 \qquad c_1 k_1(\boldsymbol{x},\boldsymbol{z}) + c_2 k_2(\boldsymbol{x},\boldsymbol{z}) \qquad k_1(\boldsymbol{x},\boldsymbol{z}) k_2(\boldsymbol{x},\boldsymbol{z})$$

$$\frac{k_1(\boldsymbol{x},\boldsymbol{z})}{\sqrt{k_1(\boldsymbol{x},\boldsymbol{x}) k_1(\boldsymbol{z},\boldsymbol{z})}}$$

$\sum_{i=1}^{\infty} c_i \langle \boldsymbol{x},\boldsymbol{z}\rangle^i$
(in case of convergence)

© 2021 MLI

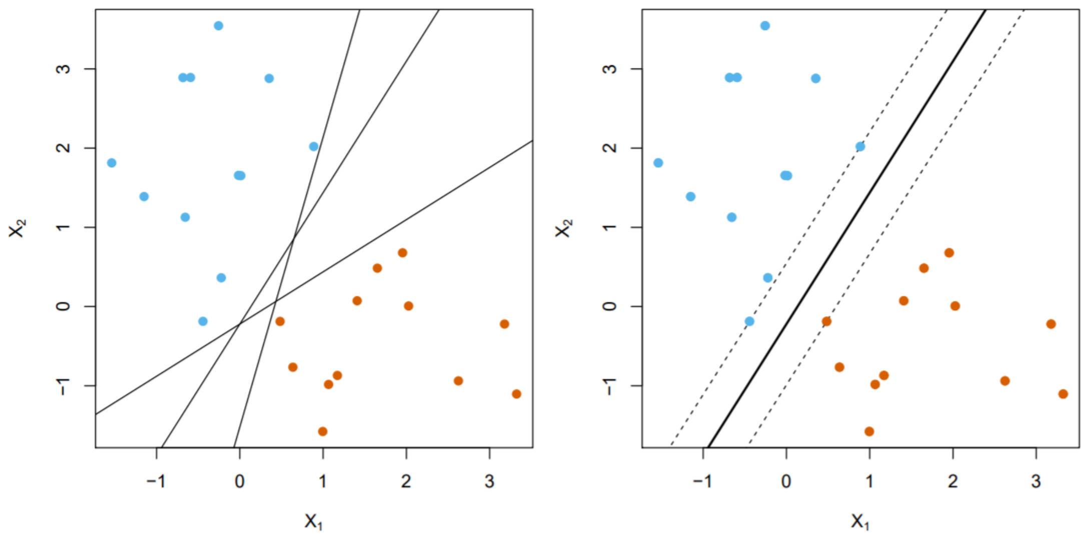
Support Vector Machines

- Main Idea and Details
- Linear Support Vector Machines
- Kernel/Nonlinear Support Vector Machines
- Kernel selection and fine-tuning

© 2021 MLI

Main Idea and Details

- **Main idea**: given a $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$, $y_i \in \{-1, 1\}$, $H = \{\boldsymbol{x} \in \mathbb{R}^J : < \boldsymbol{w}, \boldsymbol{x} > = c\}$, with normal vector $\boldsymbol{w}$ and offset *c*, separating points from class -1 and class 1 with the largest symmetric **margin of $\pm\boldsymbol{\eta}$**



- **A contrast**: whilst Logistic Regression first build a conditional probability model, and then classify based in a threshold, SVMs bypass the first step, and build a classifier directly!

Linear Support Vector Machines

- **Hard margin Linear SVM**: Find a hyperplane $H = \{\boldsymbol{x} \in \mathbb{R}^J : < \boldsymbol{w}, \boldsymbol{x} > = c\}$, with normal vector $\boldsymbol{w}$ and offset c, separating points from class -1 and class 1 with the largest symmetric **margin of** $\pm\boldsymbol{\eta} = \frac{\mathbf{1}}{||\boldsymbol{w}||}$

$$\max_{\boldsymbol{w},c} \eta\,, \qquad s.t. \, y_i \cdot (< \boldsymbol{w}, \boldsymbol{x_i} > -c) \geq 1, \qquad y_i \in \{-1, 1\}$$

Too strict in practice!

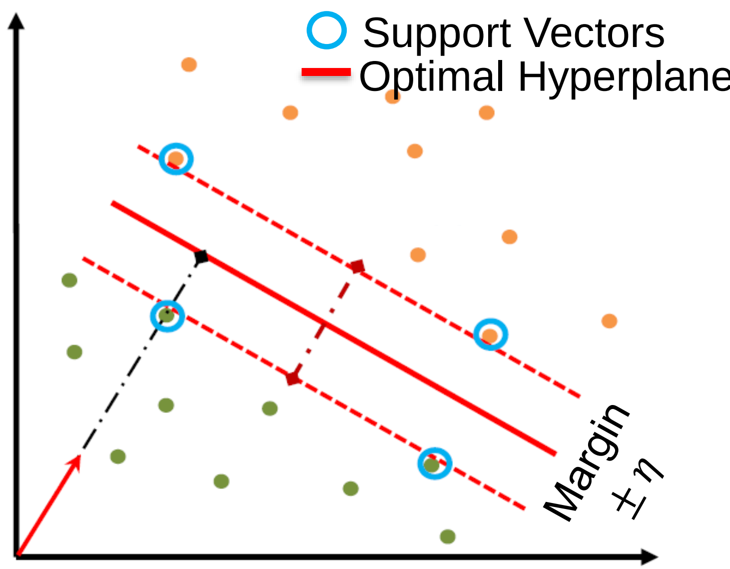
- **Soft margin Linear SVM**

$$\min_{\boldsymbol{w},c} \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_i \xi_i\,,$$

$$s.t.: \; y_i \cdot (< \boldsymbol{w}, \boldsymbol{x_i} > -c) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

where ξ_i is a slack variable for each data point; it measures how much the i-th sample is allowed to violate the margin. Two conflicting objectives, maximize $\boldsymbol{\eta}$ but minimizing ξ_i as possible; C **is the hyperparameter that control this trade-off**

Both problems can be solved via Quadratic Programming solvers



© 2021 MLI

Kernel Support Vector Machines

Primal form

$$\min_{\boldsymbol{w},c} \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_i \xi_i\,,$$

$$s.t.\, y_i \cdot (<\boldsymbol{w}, \boldsymbol{x_i}> - c) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Dual form

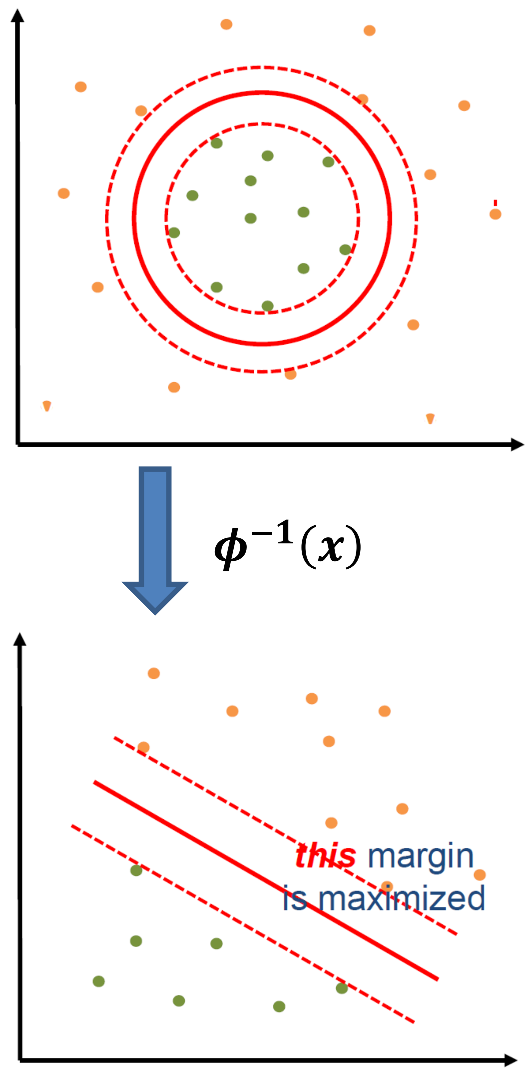
$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha} - \sum_i \alpha_i$$

$s.t.\, \boldsymbol{y}^T\boldsymbol{\alpha} = 0$, with $0 \leq \alpha_i \leq C$, *i=1,…,n*

with $\boldsymbol{Q}_{ik} \coloneqq y_i y_k k(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is an *n x n positive definite matrix* and C an upper bound for the dual weights

Decision function

$$f(\boldsymbol{z}) = \operatorname{sgn}\left(\sum_i y_i \alpha_i\, K(\boldsymbol{x}_i, \boldsymbol{z}) \; + c\right)$$



Also solvable via QP, via Slater Theorem

Kernel selection and fine-tuning

- **Kernel Ridge Regression** (remember to scale input variables)
  - Start with a **Grid-search strategy, making it fine-grained** after a certain range looks promising
  - Try a range of regularization values: $\lambda = [0.0001, 1.0]$
  - RBF Kernel is preferred; values ranging: $\sigma = [0.01, 10.0]$
  - Use RMSE, **but also MAD and Pseudo-R2** (Square of the Correlation) for hyperparameter selection
- **Support Vector Machines** (remember to scale input variables)
  - **Grid-search strategy**: similar to KRR
  - Try a range of penalty values: $C = [0.001, 10.0]$
  - RBF Kernel is preferred; values ranging: $\sigma = [0.01, 10.0]$
  - **F1-Score tends to be a good generic metric**, but also Area Under the ROC curve (AUC), and precision/recall are important metrics. Accuracy is fine when the classes are well-balanced

Main Reading

- Chapters 5.8 and 12, of Elements of Statistical Learning, Hastie T., Tibshirani R. and Friedman J.
- Chapelle, O. (2007). Training a support vector machine in the primal. Neural computation, 19(5), 1155-1178.

email: faculty@mlinstitute.com

Further Reading

- Chapters 2 and 3 of Kernel Methods for Pattern Analysis,Shawe-Taylor J., and Cristianini N., Cambridge University Press
- Chapter 19, Support Vector Machines and Kernel Methods, Computer Age Statistical Inference, Efron B., Hastie T.
- Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. The annals of statistics, 1171-1220.
- General Reference: Scholkopf, B., & Smola, A. J. (2001). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

email: faculty@mlinstitute.com

DISCLAIMER

The material in this presentation is based on information that we consider reliable, but we do not warrant that it is accurate or complete, and it should not be relied on as such. Opinions expressed are current opinions only. We are not soliciting any action based upon this material. Neither the author, his employers, any operating arm of his employers nor any affiliated body can be held liable or responsible for any outcomes resulting from actions arising as a result of delivering this presentation. This presentation does not constitute investment advice nor should it be considered as such.

The views expressed in this presentation represent those of the lecturer in his or her individual private capacity and should not be taken to be the views of any employer or any affiliated body, including the MLI, or of the lecturer as an employee of any institution or affiliated body. Either he/she or his/her employers may or may not hold, or have recently held, a position in any security identified in this document.

This presentation is © MLI 2021. No part of this presentation may be copied, reproduced, distributed or stored in any form including electronically without express written permission in advance from the author.