Lecture 16: Kernel Ridge Regression

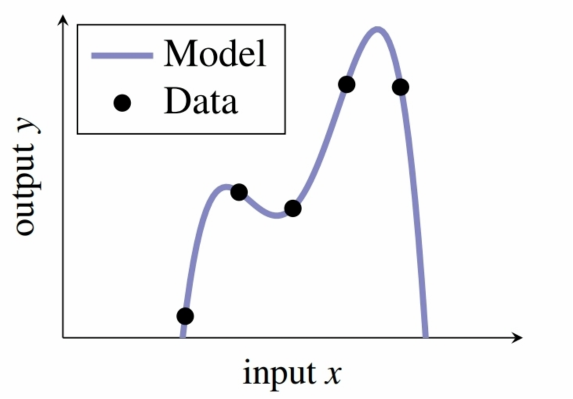
- Previously, in our discussion on polynomial regression, we had shown how could map the input variables to a new feature space of polynomials

$$\underbrace{y = a_0 + a_1 x + \epsilon}_{\text{Original input space}} \xrightarrow{\text{Mapping}} \underbrace{y = b_0 + b_1 x + b_2 x^2 + \cdots + b_p x^p + \epsilon}_{\text{New input space}}$$

- While we created these non-linear transformations of the original input, we were still using linear regression, since the parameters $b_0, b_1, \dots, b_p$ appear linearly with $\underline{\phi}(x) = [1 \ \ x \ \ x^2 \ \dots \ x^p]^T$ as the new input

$$\boxed{y = \underline{\theta}^T \underline{\phi}(x) + \epsilon}$$

Linear regression with a 4th order polynomial



- For vector-valued input $\underline{x}$, the non-linear transformation could be expressed as

$$\underset{1\times 1}{y} = \underbrace{\underline{\phi}^T(\underline{x})}_{1\times d}\ \underset{d\times 1}{\underline{\theta}} + \underset{1\times 1}{\epsilon}$$

$$\underline{x} \in \mathbb{R}^p$$

$$\underline{\phi}(\underline{x}) \in \mathbb{R}^d$$

$$\underline{\theta} \in \mathbb{R}^d$$

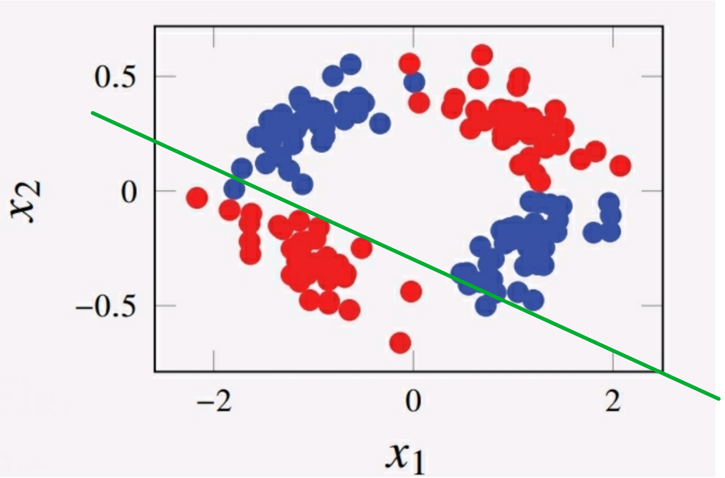
- Any choice of nonlinear transformation $\underline{\phi}(\underline{x})$ can be used!

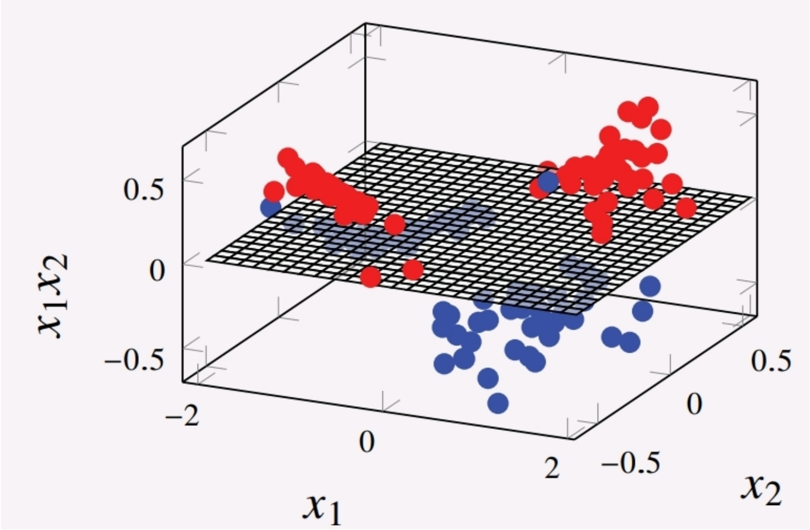
- Writing the vectorized linear regression for training data $\{\underline{x}_i, y_i\}_{i=1}^N$

$$\underline{\underline{X}} = \underset{N\times p}{\begin{bmatrix} — & \underline{x}_1^T & — \\ — & \underline{x}_2^T & — \\ & \vdots & \\ — & \underline{x}_N^T & — \end{bmatrix}}, \quad \underline{\underline{\Phi}}(\underline{\underline{X}}) = \underset{N\times d}{\begin{bmatrix} — & \underline{\phi}(\underline{x}_1)^T & — \\ — & \underline{\phi}(\underline{x}_2)^T & — \\ & \vdots & \\ — & \underline{\phi}(\underline{x}_N)^T & — \end{bmatrix}}, \quad \underline{y} = \underset{N\times 1}{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}$$

$$\boxed{\underline{y} = \underline{\underline{\Phi}}(\underline{\underline{X}})\ \underline{\theta} + \underline{\epsilon}}$$

Example of non-linear feature transformation for classification





A linear classifier would not work on the original input space

(there is no line that can separate the two classes)

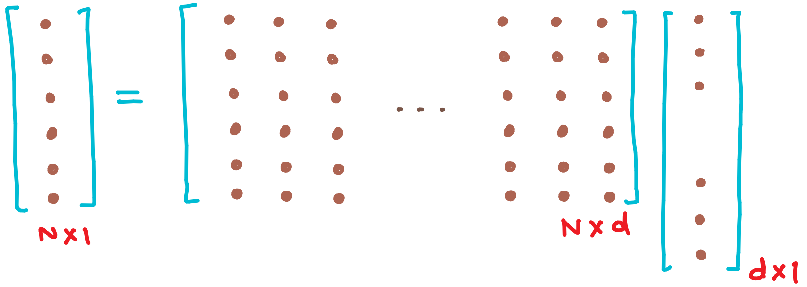
With an introduction of an extra feature $x_1 x_2$ the problem becomes linearly separable

A carefully engineered transformation $\underline{\phi}(x)$ in linear regression or linear classification may perform very well for a specific ML problem

- We would like a $\underline{\phi}(\underline{x})$ that would work for most problems

- Thus, $\underline{\phi}(\underline{x})$ should contain a lot of transformations that could possibly be of interest to most problems

- Therefore, we should choose d, the dimension of $\underline{\phi}(\underline{x})$, really large
  - $\underset{\text{\# of input features}}{d} > \underset{\text{\# of training data-points}}{N}$ and eventually let $d \to \infty$



- However, increasing the flexibility of a model also means it can overfit the training data

- We will have to use some kind of regularization to prevent overfitting

- Let us use an L_2-regularization

- Reformulating the linear regression with transformed features $\underline{\phi}(\underline{x})$, we get the estimate of parameters as

$$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\arg\min}\ J(\underline{\theta})$$

$$= \underset{\underline{\theta}}{\arg\min} \left[\frac{1}{N} \sum_{i=1}^{N} \left(y_i - \underline{\phi}(\underline{x}_i)^T \underline{\theta} \right)^2 + \lambda \| \underline{\theta} \|_2^2 \right]$$

- Linear regression with L_2-regularization has closed-form solution

$$\hat{\underline{\theta}} = \left(\underline{\underline{X}}^T \underline{\underline{X}} + N\lambda \underline{\underline{I}} \right)^{-1} \underline{\underline{X}}^T \underline{y} \qquad (\text{recall !})$$

$$\underbrace{\hat{\underline{\theta}}}_{d \times 1} = \left(\underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d \times N} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})}_{N \times d} + \underbrace{N\lambda \underline{\underline{I}}}_{d \times d} \right)^{-1} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d \times N} \underbrace{\underline{y}}_{N \times 1}$$

- Linear regression with L_2-regularization has closed-form solution

$$\underbrace{\hat{\underline{\theta}}}_{d\times 1} = \left(\underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d\times N} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})}_{N\times d} + \underbrace{N\lambda \underline{\underline{I}}}_{d\times d} \right)^{-1} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d\times N} \underbrace{\underline{y}}_{N\times 1}$$

- The downside of choosing a very large number of features, d, is that size$(\hat{\underline{\theta}}) = d \to \infty$ and storing $\hat{\underline{\theta}}$ digitally becomes a problem!

- During prediction, we need to use the d-dimensional parameter vector $\hat{\underline{\theta}}$

$$\hat{y}(\underline{x}_*) = \underset{1\times d}{\underline{\phi}(\underline{x}_*)^T} \; \underset{d\times 1}{\hat{\underline{\theta}}}$$

- But if $d \to \infty$, how to scale computations or meet storage demands ??

- Let's try to reformulate the prediction

$$\hat{y}(\underline{x}_*) = \underbrace{\underline{\phi}(\underline{x}_*)^T}_{1\times d}\, \underbrace{\hat{\underline{\theta}}}_{d\times 1} = \hat{\underline{\theta}}^T \underline{\phi}(\underline{x}_*)$$

$$= \Big[\underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{\underline{\Phi}}(\underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)}_{d\times d}^{-1} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d\times N} \underbrace{\underline{y}}_{N\times 1} \Big]^T \underbrace{\underline{\phi}(\underline{x}^*)}_{d\times 1}$$

$$= \underbrace{\underline{y}^T}_{1\times N} \underbrace{\underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})}_{N\times d} \underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{\underline{\Phi}}(\underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)}_{d\times d}^{-1} \underbrace{\underline{\phi}(\underline{x}^*)}_{d\times 1}}_{N\times 1}$$

This entire expression is independent of 'd' and if we could compute this N-dimensional vector directly, then it would be great

- However, $\underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{\underline{\Phi}}(\underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)}_{d\times d}^{-1}$ still requires inverting a d×d matrix!!

$$\hat{y}(\underline{x}_*) = \underline{y}^T \underline{\underline{\Phi}}(\underline{\underline{X}}) \underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{\underline{\Phi}}(\underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)}_{d \times d}^{-1} \underline{\phi}(\underline{x}_*)$$

- To prevent inverting a $d \times d$ matrix, where d is very large, lets use a matrix identity: $\underline{\underline{A}}\left(\underline{\underline{A}}^T \underline{\underline{A}} + \underline{\underline{I}}\right)^{-1} = \left(\underline{\underline{A}}\,\underline{\underline{A}}^T + \underline{\underline{I}}\right)^{-1} \underline{\underline{A}}$

$$\hat{y}(\underline{x}_*) = \underbrace{\underline{y}^T}_{1 \times N} \underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})\, \underline{\underline{\Phi}}(\underline{\underline{X}})^T + N\lambda \underline{\underline{I}}\right)^{-1}}_{N \times N} \underbrace{\underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})}_{N \times d}\, \underbrace{\underline{\phi}(\underline{x}_*)}_{d \times 1}}_{N \times 1}$$

- We can now compute $\hat{y}(\underline{x}_*)$ without having to deal with any d-dimensional quantities, if we can compute $\underline{\underline{\Phi}}(\underline{\underline{X}})\, \underline{\underline{\Phi}}(\underline{\underline{X}})^T$ & $\underline{\underline{\Phi}}(\underline{\underline{X}})\, \underline{\phi}(\underline{x}^*)$ directly

$$\underset{N \times d}{\underline{\underline{\Phi}}(\underline{\underline{X}})} = \begin{bmatrix} \text{——} & \underline{\phi}(\underline{x}_1)^T & \text{——} \\ \text{——} & \underline{\phi}(\underline{x}_2)^T & \text{——} \\ & \vdots & \\ \text{——} & \underline{\phi}(\underline{x}_N)^T & \text{——} \end{bmatrix} \qquad \underset{d \times N}{\underline{\underline{\Phi}}(\underline{\underline{X}})^T} = \begin{bmatrix} | & & | \\ \underline{\phi}(\underline{x}_1) & \cdots & \underline{\phi}(\underline{x}_N) \\ | & & | \end{bmatrix}$$

$$\hat{y}(\underline{x}_*) = \underbrace{\underline{y}^T}_{1\times N} \underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{x}})\,\underline{\underline{\Phi}}(\underline{\underline{x}})^T + N\lambda \underline{\underline{I}}\right)}_{N\times N}{}^{-1} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{x}})\;\underline{\phi}(\underline{x}_*)}_{N\times 1}$$

- Lets look at the two matrix multiplications

$$\underbrace{\underbrace{\underline{\underline{\Phi}}(\underline{\underline{x}})}_{N\times d}\;\underbrace{\underline{\underline{\Phi}}(\underline{\underline{x}})^T}_{d\times N}}_{N\times N} = \underbrace{\begin{bmatrix} \underline{\phi}(\underline{x}_1)^T\underline{\phi}(\underline{x}_1) & \underline{\phi}(\underline{x}_1)^T\underline{\phi}(\underline{x}_2) & \cdots & \underline{\phi}(\underline{x}_1)^T\underline{\phi}(\underline{x}_N) \\ \underbrace{\underline{\phi}(\underline{x}_2)^T}_{1\times d}\underbrace{\underline{\phi}(\underline{x}_1)}_{d\times 1} & \underline{\phi}(\underline{x}_2)^T\underline{\phi}(\underline{x}_2) & \cdots & \underline{\phi}(\underline{x}_2)^T\underline{\phi}(x_N) \\ \vdots & \vdots & \vdots & \vdots \\ \underline{\phi}(\underline{x}_N)^T\underline{\phi}(\underline{x}_1) & \underline{\phi}(\underline{x}_N)^T\underline{\phi}(\underline{x}_2) & \cdots & \underline{\phi}(\underline{x}_N)^T\underline{\phi}(x_N) \end{bmatrix}}_{N\times N}$$

$$\underbrace{\underbrace{\underline{\underline{\Phi}}(\underline{\underline{x}})}_{N\times d}\;\underbrace{\underline{\phi}(\underline{x}_*)}_{d\times 1}}_{N\times 1} = \begin{bmatrix} \underline{\phi}(\underline{x}_1)^T\,\underline{\phi}(\underline{x}_*) \\ \underline{\phi}(\underline{x}_2)^T\,\underline{\phi}(\underline{x}_*) \\ \vdots \\ \underline{\phi}(\underline{x}_N)^T\,\underline{\phi}(\underline{x}_*) \end{bmatrix}$$

- $\underline{\phi}(\underline{x})^T\,\underline{\phi}(\underline{x}')$ is an inner product between two d-dimensional vectors $\underline{\phi}(\underline{x})$ & $\underline{\phi}(\underline{x}')$

- $\underline{\phi}(\underline{x})$ enters the prediction $y(\underline{x}_*)$ only as these inner products

Lets take an example of polynomial transformation

$\underset{1\times 1}{x} \leftarrow$ scalar $(p=1)$

$\underline{\phi}(x)$ is say a third-order scaled polynomial of the form : $\underset{4\times 1}{\begin{bmatrix} 1 \\ \sqrt{3}\,x \\ \sqrt{3}\,x^2 \\ x^3 \end{bmatrix}}$

$$\underline{\phi}(x)^T\,\underline{\phi}(x') = \begin{bmatrix} 1 & \sqrt{3}x & \sqrt{3}x^2 & x^3 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{3}\,x' \\ \sqrt{3}\,x'^2 \\ x'^3 \end{bmatrix}$$

$$= 1 + 3xx' + 3x^2x'^2 + x^3x'^3 = (1+xx')^3$$

In general, if $\underline{\phi}(x)$ is a suitably scaled polynomial of order 'd', then

$$\underbrace{\underline{\phi}(x)^T\,\underline{\phi}(x')}_{\text{inner product}} = (1+xx')^d$$

- Usually to compute $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$
  - One has to first compute d-dimensional vectors $\underline{\phi}(\underline{x})$ and $\underline{\phi}(\underline{x}')$, and
  - then compute their inner product $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$

- However, for the previous example, we found that we could have just evaluated the expression $(1+ x x')^d$ directly

- Important point: If we make the choice of $\underline{\phi}(\underline{x})$ s.t. the inner product $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$ can be computed without first calculating $\underline{\phi}(\underline{x})$, we can let $d \rightarrow$ very large

- Infact, if you don't really care about $\underline{\phi}(\underline{x})$ explicitly sometimes, then the need of deriving $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$ can be bypassed by using the concept of kernels

Introducing the idea of kernels

kappa

- In simple terms, a kernel $K(\underline{x}, \underline{x}')$ is any function that takes two arguments $\underline{x}$ and $\underline{x}'$ from the same space $\mathbb{R}^p$ and returns a scalar

- We will mostly limit ourselves to kernels that are real-valued and symmetric

  i.e. $K(\underline{x}, \underline{x}') = K(\underline{x}', \underline{x}) \in \mathbb{R}$ for all $\underline{x}$ and $\underline{x}'$

  For example, $K(x, x') = (1 + x x')^d$ is such a kernel

- In fact, the inner product of two non-linear input transformation is also an example of a kernel:

$$\boxed{K(\underline{x}, \underline{x}') = \underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')}$$

- So instead of choosing $\underline{\phi}(\underline{x})$ and deriving its inner product $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$ sometimes one can choose a kernel $K(\underline{x}, \underline{x}')$ directly ← KERNEL TRICK

If $\underline{x}$ enters the model as $\underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x}')$ only, we can choose a kernel $K(\underline{x}, \underline{x}')$ directly, instead of choosing $\underline{\phi}(\underline{x})$ KERNEL TRICK

- Mathematically, we can rewrite

$$\hat{y}(\underline{x}_*) = \underbrace{\underline{y}^T}_{1\times N} \underbrace{\left(\underline{\underline{\Phi}}(\underline{\underline{X}})\, \underline{\underline{\Phi}}(\underline{\underline{X}})^T + N\lambda \underline{\underline{I}}\right)^{-1}}_{N\times N} \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})\ \underline{\phi}(\underline{x}_*)}_{N\times 1}$$

as

$$\boxed{\hat{y}(\underline{x}_*) = \underline{y}^T \Big(\underset{N\times N}{\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})} + N\lambda \underline{\underline{I}} \Big)^{-1} \underset{N\times 1}{\underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)}}$$

where

$$\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) = \begin{bmatrix} K(\underline{x}_1, \underline{x}_1) & K(\underline{x}_1, \underline{x}_2) & \cdots & K(\underline{x}_1, \underline{x}_N) \\ K(\underline{x}_2, \underline{x}_1) & K(\underline{x}_2, \underline{x}_2) & \cdots & K(\underline{x}_2, \underline{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(\underline{x}_N, \underline{x}_1) & K(\underline{x}_N, \underline{x}_2) & \cdots & K(\underline{x}_N, \underline{x}_N) \end{bmatrix}, \quad \underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*) = \begin{bmatrix} K(\underline{x}_1, \underline{x}_*) \\ K(\underline{x}_2, \underline{x}_*) \\ \vdots \\ K(\underline{x}_N, \underline{x}_*) \end{bmatrix}$$

- Recall, linear regression with L_2-regularization was called as ridge regression

- $$\hat{y}(\underline{x}_*) = \underline{y}^T \left(\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) + N\lambda \underline{\underline{I}} \right)^{-1} \underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)$$

  This equation describes linear regression with L_2-regularization using a kernel, hence is called kernel ridge regression

  $\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) \longleftarrow$ Gram matrix (or Gramian matrix)

- The design choice is now to select a kernel $K(\underline{x}, \underline{x}')$ instead of $\underline{\phi}(\underline{x})$

- In practice, choosing $K(\underline{x}, \underline{x}')$ is much easier than choosing an appropriate $\underline{\phi}(\underline{x})$ especially when the number of transformed features (i.e. d) is very large

- From computation point of view, we can choose $K(\underline{x}, x')$ arbitrarily, as long as we can compute

$$\hat{y}(\underline{x}_*) = \underline{y}^T \Big(\underbrace{\overset{\text{Gram matrix}}{\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})} + N\lambda \underline{\underline{I}}}_{\text{this must be invertible}} \Big)^{-1} \underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)$$

- For the inverse $\left(\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)^{-1}$ to exist, we will restrict ourselves to kernels for which the Gram matrix $\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})$ is always PSD

positive semi-definite

A matrix $\underline{\underline{M}}$ is said to be PSD if

- $\underline{v}^T \underline{\underline{M}}\, \underline{v} \geqslant 0$ for all $\underline{v}$
- equivalently, all eigenvalues of $\underline{\underline{M}} \geqslant 0$

- Kernels $K(\underline{x}, \underline{x}')$ that leads to a PSD $\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})$ are called PSD kernels

Examples of positive semi-definite kernels

- Squared exponential kernel (also known as radial basis function, RBF exponentiated quadratic, Gaussian kernel)

$$K(\underline{x}, \underline{x}') = \exp\left(-\frac{\|\underline{x}-\underline{x}'\|_2^2}{2\ell^2}\right)$$

where $\ell > 0$ is a hyperparameter to be chosen by the user (by cross-validation)

- Polynomial kernel

$$K(\underline{x}, \underline{x}') = (c + x^T x')^{d-1}$$

order of polynomial

- You will see more examples of symmetric PSD kernels later

$$\hat{y}(\underline{x}_*) = \underline{y}^T \underbrace{\left(\underline{\underline{K}}(\underline{\underline{X}},\underline{\underline{X}}) + N\lambda\underline{\underline{I}}\right)}_{N\times N \text{ matrix}}^{-1} \underline{\underline{K}}(\underline{\underline{X}},\underline{x}_*)$$

- Inversion of a high-dimensional matrix is a very heavy operation

- Do we need to invert the matrix $\left(\underline{\underline{K}}(\underline{\underline{X}},\underline{\underline{X}}) + N\lambda\underline{\underline{I}}\right)$ every time we predict for a new test input $\underline{x}_*$?

  — Not necessary

- We can introduce an N-dimensional vector $\underline{\hat{\alpha}}$ ← (Dual parameter)

Test prediction

$$\underset{1\times N}{\underline{\hat{\alpha}}^T} = \underbrace{\begin{bmatrix}\hat{\alpha}_1\\ \hat{\alpha}_2\\ \vdots\\ \hat{\alpha}_N\end{bmatrix}^{T}}_{1\times N} = \underbrace{\underline{y}^T}_{1\times N}\underbrace{\left(\underline{\underline{K}}(\underline{\underline{X}},\underline{\underline{X}}) + N\lambda\underline{\underline{I}}\right)}_{N\times N}^{-1} \Rightarrow \boxed{\hat{y}(\underline{x}_*) = \underline{\hat{\alpha}}^T\,\underline{\underline{K}}(\underline{\underline{X}},\underline{x}_*)}$$

So now, we only need to compute and store $\underline{\hat{\alpha}}$ and $\underline{\underline{X}}$

Summary of Kernel Ridge Regression (KRR)

Training

Input: Training data $\mathcal{T} = \{ \underline{x}_i , y_i \}_{i=1}^{N}$, a kernel K, regularization parameter λ

Output: Dual parameter $\hat{\underline{\alpha}}$

- Assemble $\underline{\underline{X}}$ and compute $\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})$
- Compute $\hat{\underline{\alpha}}$ as

$$\hat{\underline{\alpha}}^T = \underline{y}^T \left(\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) + N\lambda \underline{\underline{I}} \right)^{-1}$$

Prediction with kernel ridge regression

Input: Learned dual parameter $\hat{\underline{\alpha}}$ and test input $\underline{x}_*$

Output: Prediction $\hat{y}(\underline{x}_*) = \hat{\underline{\alpha}}^T \underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)$

Primal vs Dual formulation

$$\hat{\underline{\theta}} = \left(\underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{\underline{\Phi}}(\underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)^{-1} \underline{\underline{\Phi}}(\underline{\underline{X}})^T \underline{y}$$

$$\hat{y}(\underline{x}_*) = \underline{\phi}(\underline{x}_*)^T \hat{\underline{\theta}}$$

Primal formulation

of linear regression

- $\hat{\underline{\theta}} \in \mathbb{R}^d$, $d \to \infty$

$$\hat{\underline{\alpha}} = \underline{y}^T \left(\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) + N\lambda \underline{\underline{I}}\right)^{-1}$$

$$\hat{y}(\underline{x}_*) = \hat{\underline{\alpha}}^T \underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)$$

Dual formulation

of linear regression

- $\hat{\underline{\alpha}} \in \mathbb{R}^N$, $N \to$ # of data pts (finite)

- By comparing the two formulation, we can find a relation between $\hat{\underline{\theta}}$ and $\hat{\underline{\alpha}}$

$$\hat{y}(\underline{x}_*) = \hat{\underline{\theta}}^T \underline{\phi}(\underline{x}_*) = \hat{\underline{\alpha}}^T \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})\, \underline{\phi}(\underline{x}_*)}_{\underline{\underline{K}}(\underline{\underline{X}}, \underline{x}_*)}$$

$$\Rightarrow \boxed{\underset{d\times 1}{\hat{\underline{\theta}}} = \underbrace{\underline{\underline{\Phi}}(\underline{\underline{X}})^T}_{d\times N} \underset{N\times 1}{\hat{\underline{\alpha}}}}$$

← this is a general result of Representer theorem

Simplied version of Representer's Theorem

Theorem: Let $\hat{y}(\underline{x}) = \underline{\theta}^T \underline{\phi}(\underline{x})$ with a fixed nonlinear transform $\underline{\phi}(\underline{x})$, with $\underline{\theta}$ learned from training data $\{\underline{x}_i, y_i\}_{i=1}^N$

(The dimensionality of $\underline{\theta}$ and $\underline{\phi}(\underline{x})$ need not be finite)

Furthermore, let $L(y, \hat{y})$ be any arbitrary loss function & $h: [0, \infty] \longrightarrow \mathbb{R}$ be a strictly monotonically increasing function

Then, the estimate $\hat{\underline{\theta}}$ which is the argmin of the cost function $J(\theta)$, i.e.

$$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\text{argmin}} \; \frac{1}{N} \sum_{i=1}^{N} L\Big(y_i, \underbrace{\underline{\theta}^T \underline{\phi}(\underline{x}_i)}_{\hat{y}(\underline{x}_i)}\Big) + h\left(\|\underline{\theta}\|_2^2\right)$$

can be written as

$$\hat{\underline{\theta}} = \underline{\underline{\Phi}}(\underline{X})^T \underline{\alpha}, \quad \text{with some } N\text{-dimensional vector } \underline{\alpha}$$

- What does the representor theorem mean?
  - It suggests that if $\hat{y}(\underline{x}) = \underline{\theta}^T \underline{\phi}(\underline{x})$, and $\underline{\theta}$ is to be learned using any loss function and L_2-regularization, then $\hat{\underline{\theta}}$ can be learned also from its dual parameter $\hat{\underline{\alpha}}$, using: $\hat{\underline{\theta}} = \underline{\underline{\Phi}}(\underline{x})\,\hat{\underline{\alpha}}$

  - An important implication of the representer theorem is that L_2-regularization is crucial in order to obtain the dual formalism, and we could not have obtained KRR with say L_1-regularization

  - Representer theorem is very important for most kernel methods. It tell us that we can express some models in terms of dual parameters $\underline{\alpha}$ which are of finite length N, and a kernel $K(\underline{x}, \underline{x}')$, instead of the primal parameters $\underline{\theta}$ (maybe of infinite length d) and a $\underline{\phi}(\underline{x})$