Kernel Methods

Expanding Dimensions

- Linear classifiers are great, but what if there exists no linear decision boundary
- First, we want to make linear classifiers non-linear by applying basis function (feature transformations) on the input feature vectors.
- For a data vector $\boldsymbol{x} \in R^d$,we apply the transformation $\boldsymbol{x} \rightarrow \emptyset(\boldsymbol{x})$ where $\emptyset(\boldsymbol{x}) \in R^D$. $(D \gg d)$
- Usually, we add dimensions that capture non-linear interactions among the original features.

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \text{ and } \emptyset(\boldsymbol{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$$

- This new representation $\emptyset(\boldsymbol{x})$ is very expressive and allows for non-linear decision boundaries
- But the dimensionality is extremely high. This makes our algorithm prohibitively slow.

Expanding Dimensions

- Suppose we have the following non-linear regression function and solve it using linear regression model

$$y = 3x + \sin(x)$$

- Transform the original variable x into two variabels x1 and x2 such

$$x1 = x \quad and \quad x2 = \sin(x)$$

- One can erstimate the parameters using this linear regression formula

$$y = w_1 * x1 + w_2 * x2$$

- In this approach, we write the response variable y as a linear combination of mapping functions $\emptyset_i(x)$

$$\mathrm{y} = w_1 * \emptyset_1(x) + w_2 * \emptyset_2(x)$$

where $\emptyset_1 : x \rightarrow x$ and $\emptyset_2 : x \rightarrow sin(x)$

- Does it work?

Expanding Dimensions

- If we use gradient descent with any one of standard loss functions, it is known that the gradient is a linear combination of the input samples.

- In gradient update rule, we can express w as a linear combination of all input vectors

$$w = \sum_{i=1}^{n} \alpha_i x_i$$

- For example in SVM, we have seen that $w = \sum_{i=1}^{n} \alpha_i y_i x_i$

- We will show that, throughout the entire gradient descent optimization, such coefficients $\alpha_1, \ldots, \alpha_n$ must always exist, as we can re-write the gradient updates entirely in terms of updating the α_i coefficients:

Proof by Induction

- We will prove $w = \sum_i \alpha_i x_i$ in the case of linear regression (SSE) with gradient descent
- Proof by induction:

$$l(w) = \frac{1}{2}\sum_i \left(w^\top x_i - y_i\right)^2$$

$$\frac{\partial l}{\partial w} = \sum_i \left(w^\top x_i - y_i\right) x_i$$

- Base case: $w^0 = \sum_i 0 x_i$
  - $\forall i,\ \alpha_i = 0$ satisfies the base case
- Induction assumption: $w^j = \sum_i \alpha_i x_i$ is true for some α assignment
- Proof: $w^{j+1} = w^j - s\frac{\partial l(w)}{\partial w} = \sum_i \alpha_i x_i - s\sum_i \left(w^\top x_i - y_i\right) x_i$

$$= \sum_i \underbrace{\left(\alpha_i - s\left(w^\top x_i - y_i\right)\right)}_{\text{Scalar : new } \boldsymbol{\alpha}} x_i$$

Kernel Linear Regression

- Now that w can be written as a linear combination of the training set, we can also express the inner-product of w with any input x_i in terms of inner-products between training inputs:

$$w^T x_j = \sum_{i=1}^{n} \alpha_i x_i x_j$$

- Consequently, we can also re-write the original squared-loss from

$$l(w) = \frac{1}{2} \sum_{i=1}^{n} \left(w^T x_i - y_i \right)^2$$

entirely in terms of inner-product between training inputs:

$$l(\alpha) = \frac{1}{2} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j x_j x_i - y_i \right)^2$$

- In other words, we can perform the entire gradient descent update rule without ever expressing w explicitly.
- Instead of w, we just keep track of the n coefficients $\alpha_1, \ldots, \alpha_n$. ($\#$ of $\alpha \ll \#$ of w)

The Kernel Trick

- In kernelized linear classifier (x is mapped to the higher dimension space: $x \mapsto \phi(x)$).
- We wanted to do linear regression in the new features $\emptyset(x)$. But, $\emptyset(x)$ can be very high-dimension or even infinite-dimension.
- Solution: linear regression can be done by just using inner product of two features!

1. Write the learning algorithm in terms of $\langle x_i, x_j \rangle$
2. Define a kernel $K(x, z)$ (e.g., Gaussian kernel, poly kernel)
3. Replace all $\langle \mathrm{x}, \mathrm{z} \rangle$ operation by $K(x, z)$

Recap: Linear Regression

- Linear regression minimizes the following squared loss regression loss function,

$$J(w) = \text{RSS}(w) = \sum_{i=1}^{N} \left(x_i^T w - y_i\right)^2$$

- The solution of linear regression can be written in closed form

$$w = \left(\mathrm{X}^T\mathrm{X}\right)^{-1}\mathrm{X}^T\mathrm{y} \text{ or } w = \left(X\mathrm{X}^T\right)^{-1}X\mathrm{y}$$

- Moore-Penrose Matrix Inverse: When A is $m \times n$ matrix $(m \neq n)$,

$A^{-1} = \left(\mathrm{X}^T\mathrm{X}\right)^{-1}\mathrm{X}^T\mathrm{y}$ if $m < n$. Solving $XA = Y$

$A^{-1} = \left(X\mathrm{X}^T\right)^{-1}X\mathrm{y}$ if $m > n$. Solving $AX = Y$

Kernel Linear Regression

$$w = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y} \qquad (\mathrm{X} = [x_i, x_2, \ldots x_n])$$

- We also know that w is a linear combination of all input vectors

$$w = \sum_{i=1}^{n} \alpha_i \, x_i = X\vec{\alpha}$$

- Therefore,

$$X\vec{\alpha} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y}$$

$$X^T X X^T X\vec{\alpha} = X^T X X^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y}$$

$$X^T X X^T X\vec{\alpha} = X^T\mathbf{X}\mathbf{y}$$

$$\emptyset(X)^T\emptyset(X)\emptyset(X)^T\emptyset(X)\vec{\alpha} = \emptyset(X)^T\emptyset\mathbf{(X)y}$$

$$K^2\vec{\alpha} = K\mathbf{y}$$

$$\vec{\alpha} = K^{-1}\mathbf{y}$$

where K is a kernel function

Kernel Function

- With a finite training set of n samples, inner products are often pre-computed and stored in a Kernel Matrix:

$$k_{ij} = \emptyset(x_i)^T \emptyset(x_j)$$

- If we store the matrix, we only need to do simple inner-product look-ups and low-dimensional computations throughout the gradient descent algorithm. The final classifier becomes:

$$h(x) = \sum_{j=1}^{n} \alpha_j \emptyset(x_j)^T \emptyset(x) = \sum_{j=1}^{n} \alpha_j k(x_j, x)$$

Kernel Linear Regression

- During test-time we also only need these coefficients to make a prediction on a test-input x_t, and can write the entire classifier in terms of inner-products between the test point and training points:

- The prediction of a test point then is as follows ($w = \sum_i \alpha_i x_i = X\vec{\alpha}$)

$$h(z) = z^T w \Rightarrow \emptyset(z)^T \emptyset(X)\vec{\alpha} = k(x,z)\vec{\alpha} = k(x,z)K(x,x)^{-1}y$$

where $k(x,z)$ is the kernel (vector) of the test point with the training points.

$$k(x,z) = [k(x_0,z), k(x_1,z), \dots, k(x_n,z)]$$

$$K(x,x) = \begin{bmatrix} k(x_0,x_0) & \dots & k(x_0,x_n) \\ \dots & \dots & \dots \\ k(x_n,x_0) & \dots & k(x_n,x_n) \end{bmatrix}$$

- This formula is same as Gaussian Regression! (Refer to p. 27 in Hyperparameter Tuning slides)

Kernel Function

- Let's go back to the previous example, $\emptyset(\boldsymbol{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$
- The inner product $\emptyset(\boldsymbol{x})^{\boldsymbol{T}}\emptyset(\boldsymbol{z})$ can be formulated as

$$\emptyset(x)^T\emptyset(z) = 1 \cdot 1 + x_1 z_1 + x_2 z_2 + \cdots + x_1 x_2 z_1 z_2 + \cdots + x_1 \ldots x_d z_1 \ldots z_d = \prod_{k=1}^{d}(1 + x_k z_k)$$

- The sum of 2^d terms becomes the product of d terms. We can compute the inner product from the above formula in time O(d) instead of O(2^d).
- We define the function

$$k(x_i, x_j) = \emptyset(x_i)^T\emptyset(x_j)$$

Updating α in Kernel Linear Regression (Gradient Approach)

- We previously established that $J(w) = \sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 = \sum_{i=1}^{n}\left(\sum_{j=1}^{n}\alpha_j \emptyset\left(x_j\right)^T \emptyset(x_i) - y_i\right)^2$ and $w = \sum_{j=1}^{n}\alpha_j \emptyset\left(x_j\right)$
- It follows that (stochastic)

$$\frac{\partial}{\partial \alpha_i} J(\alpha) = \gamma_i = 2\left(\sum_{j=1}^{n}\alpha_j k_{ij} - y_i\right)$$

- The gradient update in iteration t+1 becomes (we update α instead of w)

$$\alpha_i^{t+1} \leftarrow \alpha_i^t - 2s\left(\sum_{j=1}^{n}\alpha_j^t k_{ij} - y_i\right)$$

- As we have n such updates to do, the amount of work per gradient update in the transformed space is $O(n^2)$ --- far better than $O(2^d)$
- Kernel technique can be applied in many machine learning algorithms
  - kernel linear regression
  - kernel logistic regression
  - kernel PCA
  - kernel SVM (in SVM chapter)
  - kernel knn (hw)

Kernelized Logistic Regression

- Error function of ordinary logistic regression(OLR) is

$$\ell(w) = \sum_{i=1}^{n} -y_i \log p(y=1|x,w) - (1-y_i)\log[1-p(y=1|x,w)] \qquad (\text{target value} = \{1,0\})$$

$$= \sum_{i=1}^{n} \log\left(1+\exp\left(-y_i w^T x_i\right)\right) \qquad (\text{Assume target value} = \{1.-1\})$$

- As we have seen in linear regression, we know that $w = \sum_{j=1}^{n} \alpha_j x_j$. Suppose we map the x to the higher dimension space: $x \mapsto \phi(x)$

$$\ell(\alpha) = \sum_{i=1}^{n} \log\left(1+\exp\left(-y_i \sum_{j=1}^{n} \alpha_j x_j^T x_i\right)\right)$$

$$= \sum_{i=1}^{n} \log\left(1+\exp\left(-y_i \sum_{j=1}^{n} \alpha_j k(x_i,x_j)\right)\right)$$

Kernelized Logistic Regression

- Therefore,

$$\ell(\alpha) = \min_{\alpha} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \sum_{j=1}^{n} \alpha_j \, k(x_i, x_j)\right)\right)$$

- Gradient update rule of Kernel logistic regression is

$$\alpha_i \coloneqq \alpha_i - \gamma \frac{\partial}{\partial \alpha_i} \ell(\alpha)$$

Kernel PCA

Kernel PCA

- Are there ways to find non-linear, low-dimensional manifolds?
- In the higher dimensional space, we can then do PCA
- The result will be non-linear in the original data space!
- Similar idea to support vector machines

Kernel PCA

- Assumption: the projected new features have zero mean

$$m^{\Phi} = \frac{1}{N}\sum_{i=1}^{N} \Phi(x_i) = 0$$

- The covariance matrix of the projected features is M by M, calculated by

$$C^{\Phi} = \frac{1}{N}\sum_{i=1}^{N} \left(\Phi(x_i) - m^{\Phi}\right)\left(\Phi(x_i) - m^{\Phi}\right)^T = \frac{1}{N}\sum_{i=1}^{N} \Phi(x_i)\Phi(x_i)^T$$

- Its eigenvalues λ_j and eigenvectors v_j are given by

$$C^{\Phi} v_j = \lambda_j v_j$$

where $j = 1,2,\ldots,M$

Kernel PCA

- From the previous two equations, we have

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_i)\Phi(x_i)^T v_j = \lambda_j v_j$$

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_i)\underbrace{(\Phi(x_i)^T v_j)/\lambda_j}_{\alpha_{ji}} = v_j$$

which can be written as

$$v_j = \frac{1}{N}\sum_{i=1}^{N}\alpha_{ji}\Phi(x_i)$$

- Finding the eigenvectors is equivalent to finding the coefficients α_{ji}, $i = 1, \dots, N,\ j = 1, \dots, M$
- By substituting v_j above, we have

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_i)\Phi(x_i)^T \sum_{l=1}^{N}\alpha_{jl}\Phi(x_l) = \lambda_j \sum_{l=1}^{N}\alpha_{jl}\Phi(x_l)$$

Kernel PCA

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_i)\Phi(x_i)^T\sum_{l=1}^{N}\alpha_{jl}\Phi(x_l)=\lambda_j\sum_{l=1}^{N}\alpha_{jl}\Phi(x_l)$$

- We can re-write this as

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_i)\sum_{l=1}^{N}\alpha_{jl}K(x_i,x_l)=\lambda_j\sum_{i=1}^{N}\alpha_{jl}\Phi(x_l)$$

- Multiply this by $\Phi(x_k)^T$ to the left:

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_k)^T\Phi(x_i)\sum_{l=1}^{N}\alpha_{jl}K(x_i,x_l)=\lambda_j\sum_{i=1}^{N}\alpha_{jl}\Phi(x_k)^T\Phi(x_l)$$

Kernel PCA

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(x_k)^T\Phi(x_i)\sum_{l=1}^{N}\alpha_{jl}K(x_i,x_l) = \lambda_j\sum_{i=1}^{N}\alpha_{jl}\Phi(x_k)^T\Phi(x_l)$$

- We plug in the kernel again

$$\frac{1}{N}\sum_{i=1}^{N}K(x_k,x_i)\sum_{l=1}^{N}\alpha_{jl}K(x_i,x_l) = \lambda_j\sum_{i=1}^{N}\alpha_{jl}K(x_k,x_i) \qquad \forall j,k$$

- Then we can use the matrix notation

$$K^2\alpha_j = N\lambda_j K\alpha_j$$
$$K\alpha_j = N\lambda_j\alpha_j$$

where α_j is the N dimensional column vector $\alpha_j = \left(\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jN}\right)^T$

Kernel PCA and New Data

- The eigenvector problem becomes

$$K\alpha_j = N\lambda_j\alpha_j$$

- Recall the relationship of a_k to v_j in kernel PCA is

$$v_j = \frac{1}{N}\sum_{i=1}^{N}\alpha_{ji}\Phi(x_i)$$

- For a new point x, its projection onto the principal components is:

$$\Phi(x)^T v_j = \sum_{i=1}^{N}\alpha_{ji}\Phi(x)^T\Phi(x_i) = \sum_{i=1}^{N}\alpha_{ji}K(x, x_i)$$

Kernel Functions

Kernel functions

- Can any function be used as a kernel?
  - $K_\phi: X \times X \mapsto \mathbb{R}$
- No, because there must be an underlying mapping ϕ such that
  - $K_{i,j} = \phi(x_i)^\top \phi(x_j) \mapsto \mathbb{R}$
- As a result, $K = \Phi(X)^\top \Phi(X)$ where $\Phi(X) = [\phi(x_1), \phi(x_2), \ldots, \phi(x_n)]$
  - For any vector q we get $q^\top \underbrace{\left(\Phi(X)^\top \Phi(X)\right)}_{K} q = \left(\Phi(X)^\top q\right)^2 \geq 0$
- That is, a kernel function **must** result in a positive semidefinite matrix

$$M \text{ positive semi-definite} \iff \mathbf{x}^\top M \mathbf{x} \geq 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n$$

Relationship between Kernel and Transform

- Example: 2-dimensional vectors $x=[x_1 \ x_2]$; let $K(x_i,x_j)=(1 + x_i^T x_j)^2$,
- Need to show that $K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$:

$$K(x_i,x_j)=(1 + x_i^T x_j)^2 = 1+ x_{i1}^2 x_{j1}^2 + 2 x_{i1}x_{j1} x_{i2}x_{j2}+ x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$
$$= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1}x_{i2} \ x_{i2}^2 \ \sqrt{2}x_{i1} \ \sqrt{2}x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1}x_{j2} \ x_{j2}^2 \ \sqrt{2}x_{j1} \ \sqrt{2}x_{j2}]$$
$$= \varphi(x_i)^T \varphi(x_j), \quad \text{where } \varphi(x) = [1 \ x_1^2 \ \sqrt{2} x_1x_2 \ x_2^2 \ \sqrt{2}x_1 \ \sqrt{2}x_2]$$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\varphi(x)$ explicitly).

Common Kernel functions

- Linear: $K_\phi(x_i, x_j) = x_i^\top x_j$
- Polynomial: $K_\phi(x_i, x_j) = (1 + x_i^\top x_j)^k$
- RBF: $K_\phi(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$
- Sigmoid: $K_\phi(x_i, x_j) = \tanh(kx_i^\top + c)$, $\tanh(x) = \frac{\sin(x)}{\cos(x)}$

Polynomial kernel

- $K_\phi(x_i, x_j) = (1 + x_i^\top x_j)^2$
- Assuming: $x_1 = A = [a_1, a_2], x_2 = B = [b_1, b_2]$
- $K_\phi(A, B) = (1 + a_1 b_1 + a_2 b_2)^2 = 1 + 2a_1 b_1 + 2a_2 b_2 + a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$
- This is a dot product of two vectors
  - $\phi(A) = [1, a_1\sqrt{2}, a_2\sqrt{2}, a_1^2, a_2^2, a_1 a_2\sqrt{2}]$
  - $\phi(B) = [1, b_1\sqrt{2}, b_2\sqrt{2}, b_1^2, b_2^2, b_1 b_2\sqrt{2}]$
- Mind blowen
  - This is all (2nd order) features combination
  - We computed the dot product without computing the actual vectors
- Can be generalized! $K_\phi(x_i, x_j) = (1 + x_i^\top x_j)^k$

Radial basis function (RBF) kernel

- The most widely used

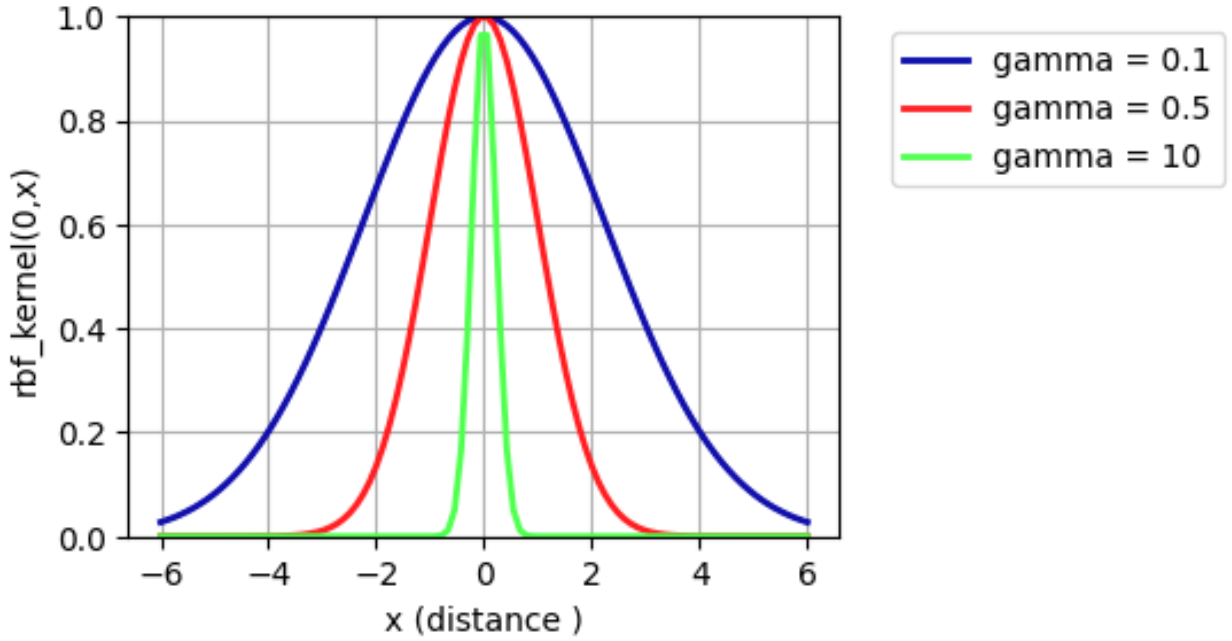
$$K_{\phi}(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

  - σ is the variance and our hyperparameter
  - $\|x_i - x_j\|$ is the Euclidean (L2-norm) Distance
- $\phi: X \mapsto \mathbb{R}^{\infty}$
- The maximum value that the RBF kernel can be is 1 and occurs when $\|x_i - x_j\| = 0$ which is when $x_i = x_j$
- RBF enforces local structure; Only a few samples are used.
- When the points are the same, there is no distance between them and therefore they are extremely similar
- When the points are separated by a large distance, then the kernel value tends to 0 which would mean that the points are dissimilar

Radial basis function (RBF) kernel

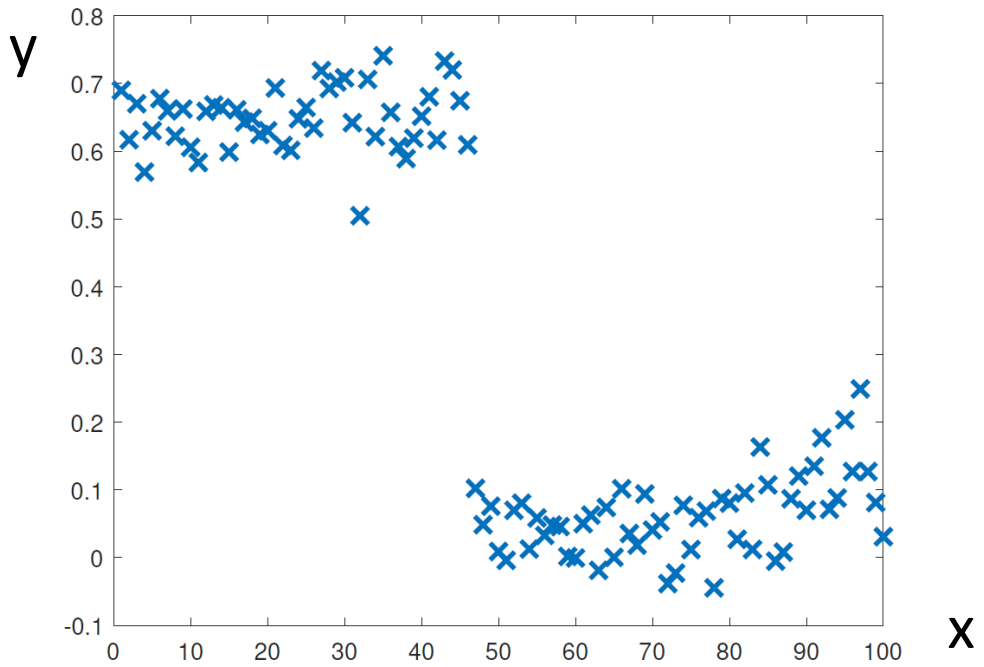
- It is important to find the right value of σ that fits the required similarity conditions (domain dependent)
- $K_\phi(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$
  - $\gamma = \frac{1}{2\sigma}$ (sklearn)

- Similar assumption to KNN:
  - Closer observations are similar



Example

- Goal: Use the kernel linear regression to fit the data points shown as follows.
- Which kernel to choose? Let us consider the RBF.

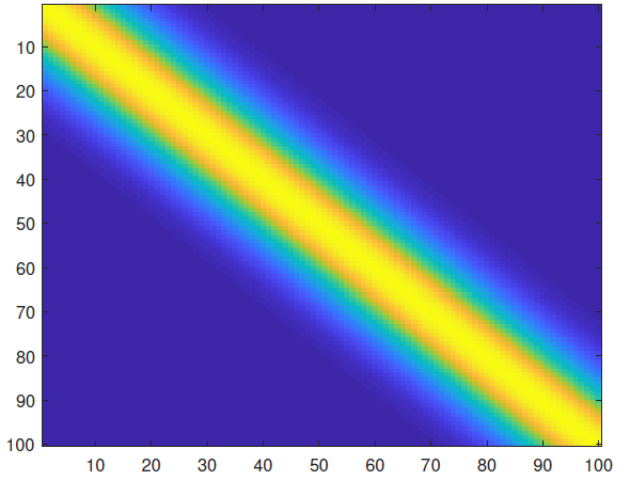


Example

- The RBF is defined as

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$$ for some σ

- The matrix K looks something below



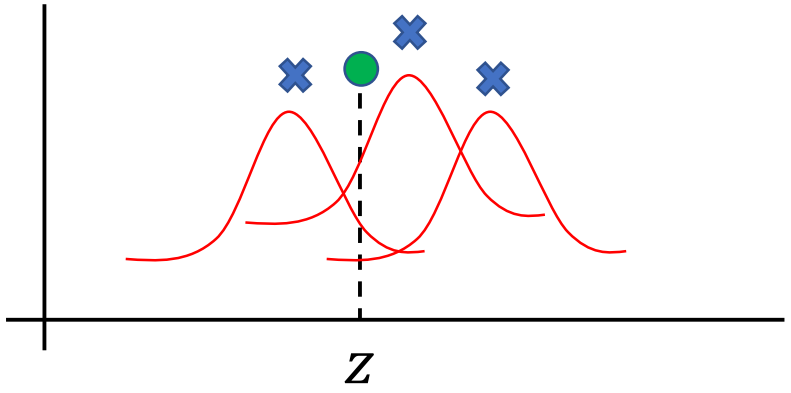
- K is a banded diagonal matrix.

Example

- The predicted value for z is

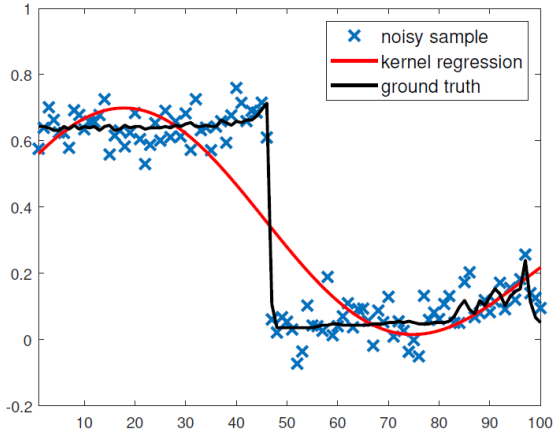
$$h(z) = \sum_i \alpha_i k(x_i, z) = \sum_i \alpha_i k(x_i, z) = \sum_i \alpha_i \exp\left(-\frac{\|z - x_i\|^2}{2\sigma^2}\right)$$

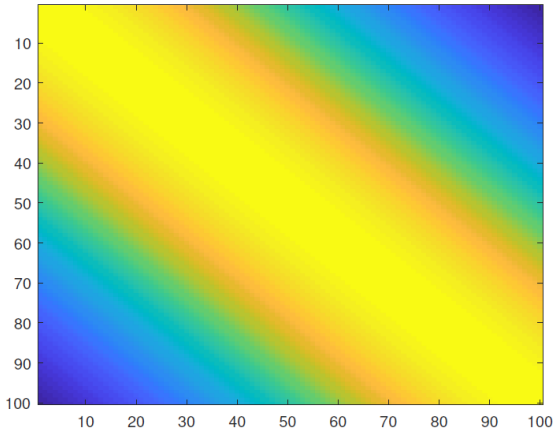
- Pictorially, the predicted function $h(z)$ can be viewed as the linear combination of the Gaussian kernels.



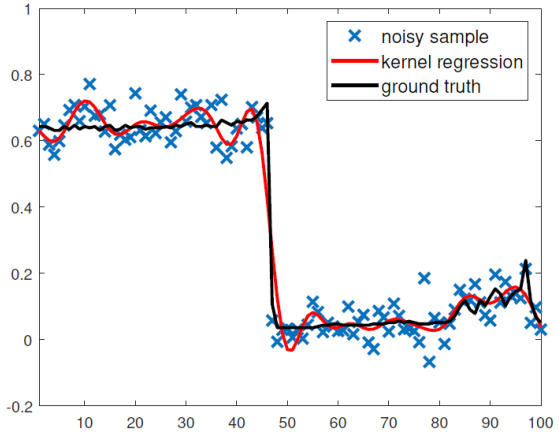
Effect of σ

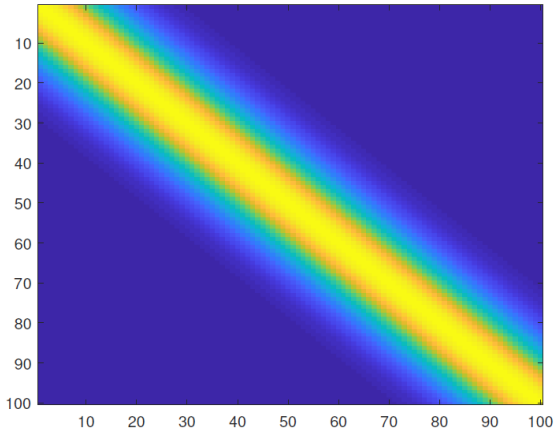
- Large σ : Flat kernel. Over-smoothing.
- Small σ : Narrow kernel. Under-smoothing.
- Below shows an example of the fitting and the kernel matrix K.



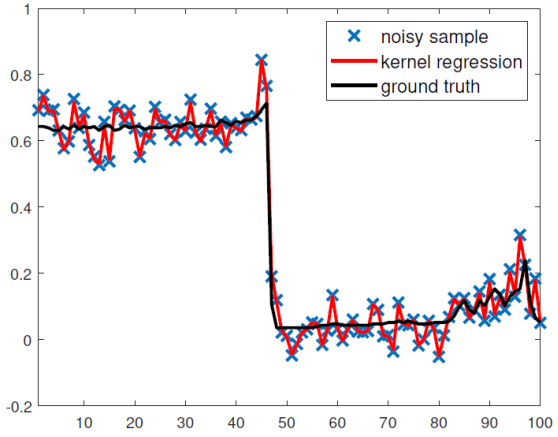


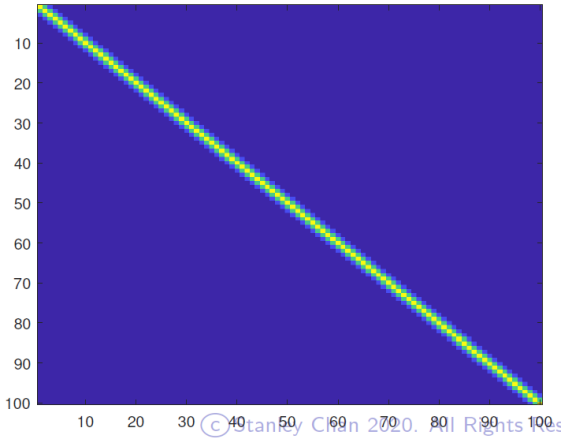
σ is large





σ just right





σ is small

Kernels in Support Vector Machines

- Example. RBF for SVM
- Radial Basis Function is often used in support vector machine.
- Poor choice of parameter can lead to low training loss, but with the risk of over-fit.
- Under-fitted data can sometimes give better generalization.