Feature maps and Kernels

CE-477: Machine Learning - CS-828: Theory of Machine Learning
Sharif University of Technology
Fall 2024

Fatemeh Seyyedsalehi

References of the lecture are mentioned in the last slide

Feature maps

- What if the output variable y can be more accurately represented as a non-linear function of x?
  - We need a more expressive family of models than linear models
  - A stronger hypothesis space

- Example:
  - We start by considering setting cubic functions:

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

  - We can view the cubic function as a linear function over the a different set of feature variables

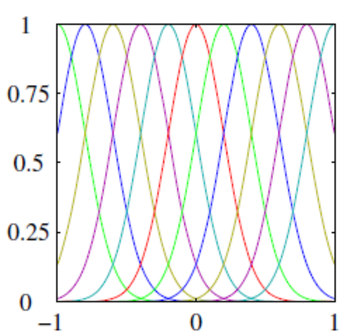
Feature maps

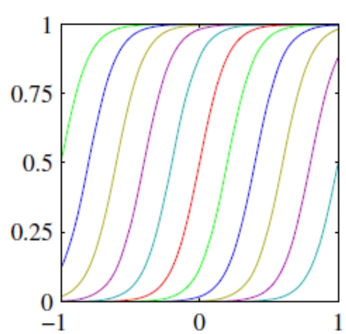
- Concretely, let the following function

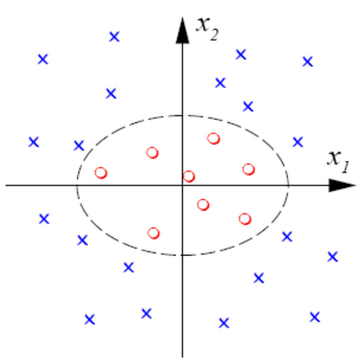
$$\phi : \mathbb{R} \rightarrow \mathbb{R}^4 \qquad \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} = \in \ \mathbb{R}^4$$

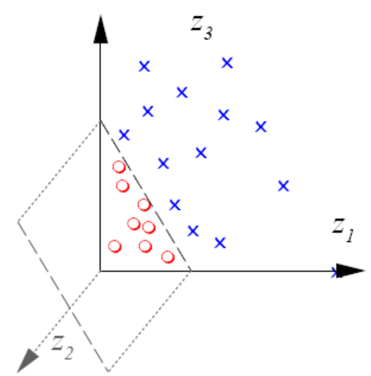
- Thus, a cubic function of the variable x can be viewed as a linear function over the variables (x).

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 = \boldsymbol{w}^T \phi(x)$$









Revisiting SSE update rule

- The gradient descent algorithm for fitting the following model

$$\boldsymbol{w}^T \phi(x)$$

- The update rule

$$\boldsymbol{w} = \boldsymbol{w} + \eta \sum_{i=1}^{N} \left(y^i - \boldsymbol{w}^T x^i\right) x^i$$

- Consider a feature map as: $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$

Revisiting SSE update rule

- New update rule:

$$\boldsymbol{w} = \boldsymbol{w} + \eta \sum_{i=1}^{N} \left(y^i - \boldsymbol{w}^T \phi(x^i)\right) \phi(x^i)$$

- The corresponding stochastic gradient descent update rule

$$\boldsymbol{w} = \boldsymbol{w} + \eta \left(y^i - \boldsymbol{w}^T \phi(x^i)\right) \phi(x^i)$$

- Gradient update above becomes computationally expensive when the features map is high-dimensional.

Revisiting SSE update rule

- d= 1000 -> each update needs computing and storing a 1000000000 dimensional vector

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} = \in \ \mathbb{R}^4 \longrightarrow \phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_1^2 \\ x_1 x_2 \\ \dots \\ x_1^3 \\ x^2 x_2 \\ \dots \end{bmatrix}$$

- Solution: kernel method

Revisiting SSE update rule

- The main observation is that at any time, $\boldsymbol{w}$ can be represented as a linear combination of the following vectors

$$\phi(x^1), \dots, \phi(x^n)$$

- Showing this inductively:
  - Initialization:

$$\boldsymbol{w} = 0 = \sum_{i=1}^{n} 0.\phi(x^i)$$

  - Assume at some point we have:

$$\boldsymbol{w} = \sum_{i=1}^{n} \beta_i.\phi(x^i)$$

  - Then: …

Revisiting SSE update rule

- Then:

$$\boldsymbol{w} = \boldsymbol{w} + \eta \sum_{i=1}^{N} \left(y^i - \boldsymbol{w}^T \phi(x^i)\right) \phi(x^i)$$

$$= \sum_{i=1}^{n} \beta_i . \phi(x^i) + \eta \sum_{i=1}^{N} \left(y^i - \boldsymbol{w}^T \phi(x^i)\right) \phi(x^i)$$

$$= \sum_{i=1}^{N} \underbrace{\left(\beta_i + \eta\left(y^i - \boldsymbol{w}^T \phi(x^i)\right)\right)}_{new\ \beta_i} \phi(x^i)$$

Revisiting SSE update rule

- Replacing $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \sum_{i=1}^{n} \beta_i . \phi(x^i)$$

- The update rule is:

$$\forall i \in \{1, \dots, n\} \;\; \beta_i = \beta_i + \eta \left(y^i - \left(\sum_{j=1}^{n} \beta_i . \phi(x^j) \right)^T \phi(x^i) \right)$$

- The inner product of two feature map:

$$\phi(x^j)^T \phi(x^i): \;\; \langle \phi(x^j), \phi(x^i) \rangle$$

- Two benefits of this new update rule:
  - We can pre-compute these pairs
  - They does not necessarily require computing ϕs explicitly.

Revisiting SSE update rule

- Prediction time:

$$\boldsymbol{w}^{\boldsymbol{T}}\phi(x) = \sum_{i=1}^{n} \beta_i . \phi\left(x^i\right)^T \phi(x) = \sum_{i=1}^{n} \beta_i \, K(x^i, x)$$

- All we need to know about the feature map is encapsulated in the corresponding kernel function.

- The kernel function corresponding to a feature map

$$\chi \times \chi \rightarrow \mathbb{R}$$
$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

Mercer Theorem

- Kernel trick → Extension of many well-known algorithms to kernel-based ones
  - By substituting the dot product with the kernel function
    - $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$
    - $k(\boldsymbol{x}, \boldsymbol{x}')$ shows the dot product of $\boldsymbol{x}$ and $\boldsymbol{x}'$ in the transformed space.
    - kernel functions K can indeed be efficiently computed, with a cost proportional to d (the dimensionality of the input) instead of p.

- Idea: when the input vectors appears only in the form of dot products, we can use kernel trick
  - Solving the problem without explicitly mapping the data
    - Explicit mapping is expensive if $\boldsymbol{\phi}(\boldsymbol{x})$ is very high dimensional

Mercer Theorem

- Construct kernel functions directly
  - Ensure that it is a valid kernel
    - Corresponds to an inner product in some feature space.

- Kernels as similarity metrics:
  - We can think of $k(\boldsymbol{x}, \boldsymbol{z})$ as some measurement of how similar are $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{z})$, or of how similar are $\boldsymbol{x}$ and $\boldsymbol{z}$.

- We need a way to test whether a kernel is valid without having to construct $\boldsymbol{\phi}(\boldsymbol{x})$

Mercer Theorem

- Let K: $\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ be given. Then for K to be a valid kernel, it is **necessary and sufficient** that for any $\{x^1, \dots, x^N\}$ the corresponding kernel matrix is **symmetric positive semi-definite**.

- The Gram matrix $\boldsymbol{K}_{N\times N}$: $K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$
  - For a set of training points $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\}$

$$K = \begin{bmatrix} k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}) & \cdots & k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(N)}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}) & \cdots & k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N)}) \end{bmatrix}$$

Kernel examples

- Polynomial kernel

$$
\begin{aligned}
K(x,z) &= (x^T z + c)^2 \\
&= \sum_{i,j=1}^{d} (x_i x_j)(z_i z_j) + \sum_{i=1}^{d} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2.
\end{aligned}
$$

- Gaussian kernels
  - Corresponds to an infinite dimensional feature map

$$
K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right).
$$

Kernel examples

- Kernels also can be defined on general types of data
  - Kernel functions do not need to be defined over vectors
    - Just we need a symmetric positive definite matrix

- Thus, many algorithms can work with general (non-vectorial) data
  - Kernels exist to embed strings, trees, graphs, …
  - E.g. for sets:

$$k(A, B) = 2^{|A \cap B|}$$

  - E.g. for graphs: random walk kernel, counts the number of paths in random walks of two graphs

References

- [1]: Andrew Ng, Machine learning, Stanford (main_notes.pdf)