Introduction to Machine Learning

Kernel Methods

Varun Chandola

March 8, 2021

Outline

Contents

1 Kernel Methods

1.1 Kernel Regression

- Ridge regression estimate:

$$\mathbf{w} = (\lambda I_D + \mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

- Prediction at $\mathbf{x}^*$:

$$y^* = \mathbf{w}^\top\mathbf{x}^* = ((\lambda I_D + \mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y})^\top\mathbf{x}^*$$

- Still needs training and test examples as D length vectors
- Rearranging above (Sherman-Morrison-Woodbury formula or *Matrix Inversion Lemma* [See Murphy p120, Matrix Cookbook])

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

The above mentioned "rearrangement" can be obtained using the *Matrix Inversion Lemma*, which in general term states for matrices **E,F,G,H**:

$$(\mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G})^{-1}\mathbf{F}\mathbf{H}^{-1} = \mathbf{E}^{-1}\mathbf{F}(\mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F})^{-1}$$

Setting $\mathbf{H} = \mathbf{I}$ and $\mathbf{E} = -a\mathbf{I}$, where a is a scalar value, we get:

$$(a\mathbf{I} + \mathbf{F}\mathbf{G})^{-1}\mathbf{F} = \mathbf{F}(a\mathbf{I} + \mathbf{G}\mathbf{F})^{-1} \tag{1}$$

Consider the prediction equation for ridge regression (we use the fact that $(\lambda\mathbf{I}_D + \mathbf{X}^\top\mathbf{X})$ is a square and symmetric matrix):

$$\begin{aligned} y^* &= ((\lambda\mathbf{I}_D + \mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y})^\top\mathbf{x}^* \\ &= \mathbf{y}^\top\mathbf{X}(\lambda\mathbf{I}_D + \mathbf{X}^\top\mathbf{X})^{-1}\mathbf{x}^* \end{aligned}$$

Using the result in (1) with $a = \lambda$, $\mathbf{F} = \mathbf{X}$, and $\mathbf{X}^\top = \mathbf{G}$:

$$y^* = \mathbf{y}^\top(\lambda\mathbf{I}_N + \mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{x}^*$$

$\mathbf{X}\mathbf{X}^\top$?

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

$\mathbf{X}\mathbf{x}^*$?

$$\mathbf{X}\mathbf{x}^* = \begin{pmatrix} \langle\mathbf{x}_1,\mathbf{x}^*\rangle \\ \langle\mathbf{x}_2,\mathbf{x}^*\rangle \\ \vdots \\ \langle\mathbf{x}_N,\mathbf{x}^*\rangle \end{pmatrix}$$

- Consider a set of P functions that can be applied on input example $\mathbf{x}$

$$\boldsymbol{\phi} = \{\phi_1, \phi_2, \ldots, \phi_P\}$$

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

- Prediction:

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

- Each entry in $\boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ is $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle$

We have already seen one such non-linear transformation in which one attribute is expanded to $\{1, x, x^2, x^3, \ldots, x^d\}$.

2 Kernel Trick

- Replace dot product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with a function $k(\mathbf{x}_i, \mathbf{x}_j)$
- Replace $\mathbf{X}\mathbf{X}^\top$ with $\mathbf{K}$

$$K[i][j] = k(\mathbf{x}_i, \mathbf{x}_j)$$

- $\mathbf{K}$ - *Gram Matrix*
- k - kernel function
  - Similarity between two data objects

Kernel Regression

$$y^* = \mathbf{y}^\top (\lambda \mathbf{I}_N + \mathbf{K})^{-1} k(\mathbf{X}, \mathbf{x}^*)$$

2.1 Choosing Kernel Functions

- Already know the simplest kernel function:

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

Approach 1: Start with basis functions

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

Approach 2: Direct design (good for non-vector inputs)

- Measure **similarity** between $\mathbf{x}_i$ and $\mathbf{x}_j$
- Should follow *Mercer's Condition*
  - **Kernel/Gram matrix must be positive semi-definite**
- k should be *symmetric*

For instance, consider the following kernel function for two-dimensional inputs, ($\mathbf{x} = (x_1, x_2)$):

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^\top \mathbf{z})^2 \\ &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^\top (x_z^2, \sqrt{2}z_1 z_2, z_2^2) \\ &= \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{z}) \end{aligned}$$

where the feature mapping $\boldsymbol{\phi}(\mathbf{x})$ is defined as:

$$\boldsymbol{\phi}(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^\top$$

2.2 Constructing New Kernels Using Building Blocks

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= c k_1(\mathbf{x}_i, \mathbf{x}_j) \\ k(\mathbf{x}_i, \mathbf{x}_j) &= f(\mathbf{x}) k_1(\mathbf{x}_i, \mathbf{x}_j) f(\mathbf{x}_j) \\ k(\mathbf{x}_i, \mathbf{x}_j) &= q(k_1(\mathbf{x}_i, \mathbf{x}_j)) \ q \text{ is a polynomial} \\ k(\mathbf{x}_i, \mathbf{x}_j) &= exp(k_1(\mathbf{x}_i, \mathbf{x}_j)) \\ k(\mathbf{x}_i, \mathbf{x}_j) &= k_1(\mathbf{x}_i, \mathbf{x}_j) + k_2(\mathbf{x}_i, \mathbf{x}_j) \\ k(\mathbf{x}_i, \mathbf{x}_j) &= k_1(\mathbf{x}_i, \mathbf{x}_j) k_2(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

3 Kernels

- **Radial Basis Function** or **Gaussian Kernel**

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

- **Cosine Similarity**

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

One can start with a Mercer kernel and show through the **Mercer's theorem** how it can be expressed as an inner product. Since $\mathbf{K}$ is positive definite we can compute an eigenvector decomposition:

$$\mathbf{K} = \mathbf{U}^\top \mathbf{\Lambda} \mathbf{U}$$

Each element of $\mathbf{K}$ can be rewritten as:

$$\mathbf{K}_{ij} = (\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:,i})^\top(\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:,j})$$

Let $\boldsymbol{\phi}(\mathbf{x}_i) = \mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}_{:,i}$. Then we can write:

$$\mathbf{K}_{ij} = \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j)$$

- The *squared dot product* kernel ($\mathbf{x_i}, \mathbf{x_j} \in \Re^2$):

$$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^\top \mathbf{x_j} \triangleq \boldsymbol{\phi}(\mathbf{x_i})^\top \boldsymbol{\phi}(\mathbf{x_j})$$

$$\boldsymbol{\phi}(\mathbf{x_i}) = \{x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2\}$$

- What about the Gaussian kernel (radial basis function)?

$$k(\mathbf{x_i}, \mathbf{x_j}) = exp\left(-\frac{1}{2\gamma^2}||\mathbf{x}_i - \mathbf{x}_j||^2\right)$$

- Assume $\gamma = 1$ and $\mathbf{x} \in \Re$ (denoted as x)

$$\begin{aligned} k(x_i, x_j) &= exp(-x_i^2)exp(-x_j^2)exp(2x_ix_j) \\ &= exp(-x_i^2)exp(-x_j^2)\sum_{k=0}^{\infty}\frac{2^k x_i^k x_j^k}{k!} \\ &= \sum_{k=0}^{\infty}\left(\frac{2^{k/2}}{\sqrt{k!}}x_i^k exp(-x_i^2)\right)\left(\frac{2^{k/2}}{\sqrt{k!}}x_j^k exp(-x_j^2)\right) \end{aligned}$$

- *Using Maclaurin Series Expansion*

$$k(x_i, x_j) = \begin{pmatrix} 1 \\ 2^{1/2}x_i^1 exp(-x_i^2) \\ \frac{2^{2/2}}{2}x_i^2 exp(-x_i^2) \\ \vdots \end{pmatrix} \times \begin{pmatrix} 1 \\ 2^{1/2}x_j^1 exp(-x_j^2) \\ \frac{2^{2/2}}{2}x_j^2 exp(-x_j^2) \\ \vdots \end{pmatrix}^\top$$

One can note above that since computing the RBF/Gaussian kernel is same as taking a dot product of two vectors of infinite length, it is equivalent to mapping the input features into an infinite dimensional space.

4 Kernel Machines

- We can use kernel function to *generate* new features
- Evaluate kernel function for each input and a set of K centroids

$$\boldsymbol{\phi}(\mathbf{x}) = [k(\mathbf{x}, \boldsymbol{\mu}_1), k(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, k(\mathbf{x}, \boldsymbol{\mu}_K)]$$

$$y = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \quad y \sim Ber(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}))$$

- If k is a Gaussian kernel $\Rightarrow$ Radial Basis Function Network (RBF)
- How to choose $\boldsymbol{\mu}_i$?
  - Clustering
  - Random selection

4.1 Generalizing RBF

- Another option: Use every input example as a "centroid"

$$\boldsymbol{\phi}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), k(\mathbf{x}, \mathbf{x}_2), \ldots, k(\mathbf{x}, \mathbf{x}_N)]$$

4.2 Extensions to Non-Vector Data Examples

- What if $\mathbf{x} \notin \Re^D$?
- Does $\mathbf{w}^\top \mathbf{x}$ make sense?
- How to adapt?
  1. Extract features from $\mathbf{x}$
  2. Is not always possible
- Sometimes it is easier/natural to compare two objects.
  - A similarity function or **kernel**
- Domain-defined measure of similarity

Example 1. **Strings:** Length of longest common subsequence, inverse of edit distance

Example 2. **Multi-attribute Categorical Vectors:** Number of matching values

References