CS 189 Spring 2019 Introduction to Machine Learning

Exam Prep Dis 2 Solutio

Q1. Multiple Choice

(a) Which of the following are properties that a kernel matrix always has?

- ○ Invertible
- ○ All the entries are positive
- ○ At least one negative eigenvalue
- ● Symmetric

(b) Let $X_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $i = 1, ..., N$ be a dataset that, as-is, is not linearly separable. We define a feature mapping Φ to deal with the non-linearity. To model the data, using Kernalized Ridge Regression instead of Ridge Regression is a good idea if:

- ○ The mapping Φ generates fewer features than d.
- ● The mapping Φ corresponds to the kernel function $k(X_i, X_j) = e^{-\frac{1}{2}\|X_i - X_j\|_2^2/\sigma^2}$ for some value of σ.
- ○ The feature map is $\Phi(X_i) = e^{-\|X_i\|_2^2/\sigma^2}$ for some value of σ.
- ○ $\Phi(X_i) \in \mathbb{R}^D$ and $D < N$.

(c) Which of the following are true of kernels?

- ● The inner product space that a kernel maps to may be infinite dimensional.
- ○ A kernel is valid only if its Gram matrix is positive definite.
- ● Kernels are useful if our data is non-linear in the current feature space, but may be linear in higher dimensions.
- ● $e^{-\alpha\|\vec{x_i} - \vec{x_j}\|_2^2}$ is a kernel which lies in infinite dimensional space.

All except 'A kernel is valid only if its Gram matrix is positive definite'.

(d) Which of the following is a valid kernel function for vectors of the same length, $\vec{x}$ and $\vec{y}$?

- ● $k(\vec{x}, \vec{y}) = \vec{x}^\top \vec{y}$
- ● $k(\vec{x}, \vec{y}) = e^{-\frac{1}{2}\|\vec{x} - \vec{y}\|_2^2}$
- ● $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^p$ for some degree p
- ○ $k(\vec{x}, \vec{y}) = k_1(\vec{x}, \vec{y}) - k_2(\vec{x}, \vec{y})$ for valid kernels k_1 and k_2.

All except $k(\vec{x}, \vec{y}) = k_1(x, y) - k_2(x, y)$ for valid kernels k_1 and k_2. This is not a valid kernel because valid kernel functions are not in general closed under subtraction, rather they are closed only under positive, linear combinations.

Q2. Kernels

- What is the primary motivation for using the kernel trick in machine learning algorithms?

  If we want to map sample points to a very high-dimensional feature space, the kernel trick can save us from having to compute those features explicitly, thereby saving a lot of time.

  (Alternative solution: the kernel trick enables the use of infinite-dimensional feature spaces.)

- Prove that for every design matrix $X \in \mathbb{R}^{n \times d}$, the corresponding kernel matrix is positive semidefinite.

  For every vector $\mathbf{z} \in \mathbb{R}^n$,

  $$\mathbf{z}^\top K \mathbf{z} = \mathbf{z}^\top X X^\top \mathbf{z} = |X^\top \mathbf{z}|^2,$$

  which is clearly nonnegative.

- Now suppose that ξ is a random variable. Suppose we define a function $k_\phi : X \times X \to \mathbb{R}$ where X is a set as

  $$k_\phi(x, y) := \mathbb{E}_\xi[\phi(x, \xi)\phi(y, \xi)] \,,$$

  where $\phi(x, \xi)$ is some arbitrary function. Prove that this k_ϕ defines a valid kernel.

  Let $x_1, ..., x_n$ be a set of data points, and let K denote the kernel matrix formed from these data points. Let $\alpha \in \mathbb{R}^n$ be a fixed vector. Observe that:

  $$\begin{aligned}
  \alpha^\top K \alpha &= \sum_{i,j} \alpha_i \alpha_j k_\phi(x_i, x_j) \\
  &= \sum_{i,j} \alpha_i \alpha_j \mathbb{E}_\xi[\phi(x_i, \xi)\phi(x_j, \xi)] \\
  &= \mathbb{E}_\xi \left[\sum_{i,j} \alpha_i \alpha_j \phi(x_i, \xi)\phi(x_j, \xi) \right] \\
  &= \mathbb{E}_\xi \left[\left(\sum_{i,j} \alpha_i \phi(x_i, \xi) \right)^2 \right] \\
  &\geq 0 \,.
  \end{aligned}$$

  Since α is arbitrary, this shows that K is PSD.

- Now let ξ denote a random integer. Use part (c) to show that the following k is a kernel on the space of subsets of integers: for $A, B \subseteq \mathbb{N}$, define

  $$k(A, B) = \Pr(\xi \in A \cap B) \,.$$

  Letting $p(i) = \Pr(\xi = i)$, we write:

  $$\Pr(\xi \in A \cap B) = \sum_{i \in \mathbb{N}} \mathbf{1}\{i \in A\}\mathbf{1}\{i \in B\}p(i) = \mathbb{E}_\xi[\mathbf{1}\{\xi \in A\}\mathbf{1}\{\xi \in B\}] \,.$$

  We can now apply part (c) with $\phi(X, \xi) = \mathbf{1}\{\xi \in X\}$.

Q3. Initialization of weights for back propagation

Recall that back propagation is simply a clever method to solve for the gradient of the loss function so we can use it in a numerical optimization method such as gradient descent. Methods like these require the parameters to be initialized to some value. In the case of logistic regression, we were able to set the weights to be 0. Assume a fully connected 1-hidden layer network with K output nodes. For notation, let us assume that $x_j^{(l)} = g(S_j^{(l)})$, and $S_j^{(l)} = \sum_i^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$, where g is the nonlinearity.

(a) Imagine that we initialize the values of our weights to be some constant w. After performing the forward pass, what is the relation between the members of the set $\{x_j^{(1)} : j = 1, \ldots, d^{(1)}\}$?

Since all of the weights are equal, we have

$$x_j^{(1)} = g\left(\sum_i^{d^{(0)}} w x_i^{(0)}\right) = g\left(w \sum_i^{d^{(0)}} x_i^{(0)}\right)$$

Note that this equation does not depend on j, thus, all the members of the set are equal. Also note that the $x_i^{(0)}$ are just the inputs.

(b) After the backwards pass of back propagation, what is the relation between the members of the set $\{\delta_i^{(1)} : i = 1, \ldots, d^{(1)}\}$?

Again, since all of the weights are equal,

$$\delta_i^{(1)} = g'\left(S_i^{(1)}\right) \sum_j^{d^{(2)}} \delta_j^{(2)} w$$

The sum term does not depend on i and is the same for all values of i in layer l. The $S_i^{(1)}$ are equal for the same reason as part (a). The members of the set are equal.

(c) After the weights are updated and one iteration of gradient descent has been completed, what can we say about the weights? Our gradient descent update looks like this:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \delta_j^{(l)} x_i^{(l-1)}$$

One can see that all $w_{ij}^{(1)}$ will be the same for all values of j. Also, $w_{ij}^{(2)}$ will be the same for all values of i. This pattern will continue and will not break for as many iterations you do. This is because the $x_j^{(1)}$ will always be the same for all j, due to the "outgoing" symmetry in the weights in layer 1, and the $d_j^{(1)}$ will always be the same due to the "incoming" symmetry in the weights in layer 2.

(d) To solve this problem, we randomly initialize our weights. This is called symmetry breaking. Why are we able to set our weights to 0 for logistic regression?

Logistic regression can be viewed as a neural network with one output node and zero hidden layers. In logistic regression, the loss function is convex. Any starting point should lead us to the global optimum.