CS 189 Introduction to Machine Learning
Spring 2021 Jonathan Shewchuk

DIS10

1 Kernelizing Nearest Neighbors

In this question, we will be looking at how we can kernelize k-nearest neighbors (k-NN). k-NN is a simple classifier that relies on nearby training points to decide what a test point's class should be.

Given a test point q, there are two steps to decide what class to predict.

1. Find the k training points nearest q.
2. Return the class with the most votes from the k training points.

For the following parts, assuming that our training points $x \in \mathbb{R}^d$, and that we have n training points.

(a) What is the runtime to classify a newly specified test point q, using the Euclidean distance? (**Hint:** use heaps to keep track of the k smallest distances.)

(b) Let us now "lift" the points into a higher dimensional space by adding all possible mononomials of the original features with degree at most p. What dimension would this space be (asymptotically)? What would the new runtime for classification of a test point q be, in terms of n, d, k, and p?

(c) Instead, we can use the polynomial kernel to compute the distance between 2 points in the lifted $O(d^p)$-dimensional space without having to move all of the points into the higher dimensional space. Using the polynomial kernel, $k(x, y) = (x^T y + \alpha)^p$ instead of Euclidean distance, what is the runtime for k-NN to classify a test point q? (Note: α is a hyperparameter, which can be tuned by validation.)

2 When is $k(x, y)$ a Kernel?

Let $\mathbb{R}^d$ be the vector space that contains our training and test points. For a function $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ to be a valid kernel, it suffices to show that either of the following conditions is true:

1. k has an inner product representation: $\exists\ \Phi : \mathbb{R}^d \to \mathcal{H}$, where $\mathcal{H}$ is some (possibly infinite-dimensional) inner product space such that $\forall x_i, x_j \in \mathbb{R}^d,\ k(x_i, x_j) = \Phi^T(x_i)\Phi(x_j)$.
2. For every sample $x_1, x_2, \ldots, x_n \in R^d$, the kernel matrix

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & k(x_i, x_j) & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix}$$

is positive semidefinite.

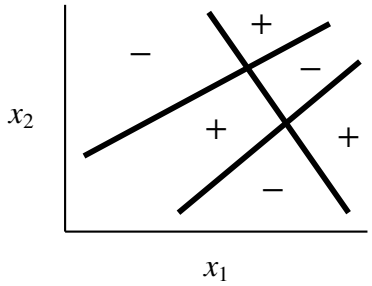
(a) Show that the first condition implies the second one, i.e. if $\forall x_i, x_j \in \mathbb{R}^d,\ k(x_i, x_j) = \Phi^T(x_i)\Phi(x_j)$ then the kernel matrix K is PSD.

(b) Given a positive semidefinite matrix A, show that $k(x_i, x_j) = x_i^\top A x_j$ is a valid kernel.

(c) Show why $k(x_i, x_j) = x_i^\top(\text{rev}(x_j))$ (where rev(x) reverses the order of the components in x) is *not* a valid kernel.

3 The Multi-layer Perceptron (MLP)

(a) Consider a target function whose + and − regions are illustrated below. Draw the perceptron components, and express f as a Boolean function of its perceptron components.



Take-away: a complicated target function, such as the one above, that is composed of perceptrons can be expressed as an `OR` of `AND`s of the component perceptrons.

Let's develop some intuition for why this might be useful.

(b) Over two inputs x_1 and x_2, how can `OR`, `AND`, and `NOT` each be implemented by a single perceptron? What about `NAND` and `NOR`? Assume each unit uses a *hard threshold* activation function: for a constant α,

$$h(x) = \begin{cases} 1 & \text{if } w^\top x \geq \alpha, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

(c) We have seen that `XOR` is not linearly separable, and hence it cannot be implemented by a single perceptron. Draw a fully connected three-unit neural network that has binary inputs $X_1, X_2, \mathbf{1}$ and output Y, where Y implements the `XOR` function $Y = \texttt{XOR}(X_1, X_2)$. Again, assume each unit uses a *hard threshold* activation function.

(d) (self-study) Create a neural network with a single hidden layer (of any number of units) that implements the function (A or NOT B) XOR (NOT C or NOT D).

Take-away: If f can be decomposed into perceptrons using an `OR` (or `NOR`) of `AND`s, then it can be implemented by a 3-layer perceptron. A one-hidden-layer MLP is a universal Boolean function. How cool is that?!

4 Mercer's Theorem (Optional)

A vast variety of algorithms in Machine Learning involve calculating dot products of d-dimensional feature vectors. These algorithms can be strengthened by adding nonlinear combinations of the original features as new features. However, calculating the dot products of the new, larger, feature vectors can take much longer. The 'kernel trick' solves this very elegantly by expressing the dot products of the longer vectors as functions of vectors in the original d-dimensional space.

We saw in class that the quadratic kernel could be written as the dot product of two higher dimensional vectors (considering only two features for simplicity):

$$\begin{aligned}(x^T z + 1)^2 &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\ &= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1 x_2 & \sqrt{2}x_1 & \sqrt{2}x_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 & z_2^2 & \sqrt{2}z_1 z_2 & \sqrt{2}z_1 & \sqrt{2}z_2 & 1 \end{bmatrix}^T \\ &= \Phi^T(x)\Phi(z)\end{aligned}$$

In this problem, we shall learn how to decompose a general kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ into a dot product of two higher-dimensional vectors. For technical purposes, we shall assume that $k(x, y)$ satisfies the Hilbert-Schmidt condition, $\int k^2(x, y)\, dxdy < \infty$. We shall first define a vector space F consisting of (so-called L^2) functions $f : \mathbb{R}^d \mapsto \mathbb{R}$.

Recall that a linear transformation from a vector r to a vector s can be written using a matrix A as $s(i) = \sum_j A(i, j)r(j)$.

(a) Write down the linear transformation L_k from function space to function space, $L_k : F \mapsto F$, that the kernel function $k(x, y)$ induces.

(b) How would one define eigenfunctions $\phi(x)$ of L_k?

Since the kernel function is symmetric positive semi definite, it is possible to prove that the above eigenvalues are all non-negative and that the eigenfunctions are orthonormal. This statement is called Mercer's Theorem. Recall that the eigendecomposition for symmetric A can be written (component-wise) as $A(i, j) = \sum_k \lambda_k v_k(i) v_k(j)$.

(c) In analogy with the eigendecomposition for matrices, write down the eigendecomposition for the kernel function $k(x, y)$ in terms of its eigenvalues and eigenvectors. Then, write it as a dot product of two higher-dimensional (possibly infinite dimensional) vectors $\Phi(x)$ and $\Phi(y)$.