

## 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 monomials 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:  $\alpha$  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:  $\exists \Phi : \mathbb{R}^d \rightarrow \mathcal{H}$ , 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  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$ , 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  $k(x_i, x_j) = x_i^T A x_j$  is a valid kernel.
- (c) Show why  $k(x_i, x_j) = x_i^T (\text{rev}(x_j))$  (where  $\text{rev}(x)$  reverses the order of the components in  $x$ ) is *not* a valid kernel.

### 3 Decision Trees and Random Forests

Random forests are a specific ensemble method where the individual models are decision trees trained in a randomized way so as to reduce correlation among them. Because the basic decision tree building algorithm is deterministic, it produces the same tree every time if we give it the same dataset and use the same hyperparameters (stopping conditions, etc.).

Consider constructing a multi-class binary decision tree on  $n$  training points with  $d$  real-valued features. The splits are chosen to maximize the information gain. We only consider splits that form a linear boundary orthogonal to one of the coordinate axes.

- (a) Give an example or disprove: For every  $n \geq 3$ , there exists some discrete probability distribution on  $n$  objects whose entropy is negative.

One may be concerned that the randomness introduced in random forests may cause trouble. For example, some features or sample points may never be considered at all. We investigate this phenomenon in parts (b)–(d).

- (b) Consider  $n$  training points in a feature space of  $d$  dimensions. Consider building a random forest with  $T$  binary trees, each having exactly  $h$  internal nodes. Let  $m$  be the number of features randomly selected (from among  $d$  input features) at each treenode. For this setting, compute the probability that a certain feature (say, the first feature) is never considered for splitting in any treenode in the forest.
- (c) Now let us investigate the possibility that some sample point might never be selected. Suppose each tree employs  $n' = n$  bootstrapped (sampled with replacement) training sample points. Compute the probability that a particular sample point (say, the first sample point) is never considered in any of the trees.

*Hint: recall that  $e^x = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n$*

- (d) Compute the values of the two probabilities you obtained in parts (b) and (c) for the case where there are  $n = 2$  training points with  $d = 2$  features each,  $T = 10$  trees with  $h = 4$  internal nodes each, and we randomly select  $m = 1$  potential splitting features in each treenode. You may leave your answer in a fraction and exponentiated form, e.g.,  $\left(\frac{51}{100}\right)^2$ . What conclusions can you draw about the concern mentioned in part (b)?

## 4 Decision Boundary Visualization for Decision Trees and Random Forests

In this problem, we will visualize the decision boundaries of decision trees and random forests using an interactive Jupyter notebook. You don't have to write any code yourself! You can find the notebook [here](#) — make sure you're signed into your Berkeley email address to access it.

*(Here is another useful tool for visualizing the decision boundaries of various machine learning models: <https://ml-visualizer.herokuapp.com>.)*