CS 189 Introduction to Machine Learning
Spring 2023 Jonathan Shewchuk

DIS9

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.)

Solution: To classify a point q, we first need to calculate the distances to every other point. The Euclidean distance is $\sqrt{(x_1 - q_1)^2 + (x_2 - q_2)^2 + \ldots + (x_d - q_d)^2}$, and so for one point, takes $O(d)$ time to compute. For n points, we get $O(nd)$ time.

As we compute the distances, we can use a max-heap to keep track of the k shortest distances seen so far. In the worst case, we have n insertions, taking $O(\log k)$ time each.

This gives us a total run time of $O(nd + n \log k)$.

(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?

Solution: Our training points now have d^p features. Therefore, it takes $O(d^p)$ time to compute the Euclidean distance between two points. For n points, the total time is $O(nd^p)$.

We again use a max-heap to keep track of the k shortest distances seen so far. In the worst case, we have n insertions, taking $O(\log k)$ time each.

This gives us a total runtime of $O(nd^p + n \log k)$.

(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.)

Solution: Let the $O(d^p)$-dimensional 'lifted' point corresponding to a point x in d-dimensional space be represented by $\Phi(x)$. The Euclidean distance squared between the test point q and a training point x in the 'lifted' space is $\|\Phi(q) - \Phi(x)\|^2$. We then have:

$$\begin{aligned}\|\Phi(q) - \Phi(x)\|^2 &= \Phi^T(q)\Phi(q) - 2\Phi^T(q)\Phi(x) + \Phi^T(x)\Phi(x) \\ &= k(q,q) - 2k(q,x) + k(x,x)\end{aligned}$$

We note that the kernel calculations take $O(d)$ time. Therefore, for n points, we get $O(nd)$ time. Following the same logic as before, we end up with a total runtime of $0(nd + n \log k)$.

Note: We assume that exponentiation (a^b) of two floating-point numbers a, b takes $O(1)$ time.

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.

Solution: $\forall a \in \mathbb{R}^n, a^T K a = \sum_{i,j} a_i a_j k(x_i, x_j) = \sum_{i,j} a_i a_j \Phi^T(x_i)\Phi(x_j) = \| \sum_i a_i \Phi(x_i)\|_2^2 \geq 0$

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

Solution: We can show k admits a valid inner product representation:

$$k(x_i, x_j) = x_i^\top A x_j = x_i^\top P D^{1/2} D^{1/2} P^\top x_j = \langle D^{1/2} P^\top x_i, D^{1/2} P^\top x_j \rangle = \langle \Phi(x_i), \Phi(x_j) \rangle$$

where $\Phi(x) = D^{1/2} P^\top x$

(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.

Solution: A counterexample: We have that $k((-1, 1), (-1, 1)) = -2$, but this is invalid since if k is a valid kernel then $\forall x,\ k(x, x) = \langle \Phi(x), \Phi(x) \rangle \geq 0$.

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.

Solution: False. The entropy is always nonnegative since $-p \log_2 p$ is nonnegative when $p \in (0, 1]$. (For an object with probability $p = 0$, we can either ignore the object or adopt the convention that $0 \log 0 = 0$, as $\lim_{n \to 0^+} = 0$.)

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.

Solution: The probability that it is not considered for splitting in a particular node of a particular tree is $1 - \frac{m}{d}$. The subsampling of m features at each treenode is independent of all others. There is a total of hT treenodes and hence the final answer is $(1 - \frac{m}{d})^{hT}$.

(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 $e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$

Solution: The probability that it is not considered in one of the trees is $\left(1 - \frac{1}{n}\right)^n$, which approaches $1/e$ as $n \to \infty$. Since the choice for every tree is independent, the probability that it is not considered in any of the trees is $(1 - \frac{1}{n})^{nT}$, which approaches e^{-T} as $n \to \infty$.

(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)?

Solution: $\frac{1}{2^{40}}$ and $\frac{1}{2^{20}}$. It is quite unlikely that a feature or a sample will be missed.

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`.)