

The Nearest Neighbor Learning Model

We now come to the discussion of learning models. We have already encountered a pretty powerful one, namely the multilayer perceptron. There are two properties of this learning model that make it appealing.

1. **Universal Approximation:** In principle we can come arbitrarily close to implementing any function by tuning the available parameters (weights, number of hidden layers, number of hidden units). Thus we “can” use a multilayer perceptron to do anything. However we must take care to ensure falsifiability, and if in addition we require low test error, then we must incorporate prior information.
2. **Efficient Learning Algorithm:** There needs to be an efficient algorithm for outputting a function in your learning model, given the data set. In other words, an efficient learning algorithm should exist. We have available first order optimization techniques and an efficient algorithm that gives us the necessary gradients (backpropagation).

Multilayer networks, therefore, satisfy our two criteria, and we will look for these two criteria in the learning models that we will study. How are we to choose among learning models that satisfy these two criteria? It appears that if we have a learning model that satisfies criterion (1) then any other learning model can be approximated within this learning model. Further if this learning model has an efficient learning algorithm, then why even consider other learning models? The reason is that (1) and (2) are only necessary conditions, and they do not tell the whole story. Different learning models present different tradeoffs. Some are harder to train but once training is done, function evaluation is easy. Some learning models are more amenable to the incorporation of a given kind prior information than others. It is easier to perform online learning with some learning models than others (online learning addresses the problem of sequentially generated data, and each time a new data point is generated, one does not want to re-learn on the whole data set, but rather perform an incremental learning on the newly generated point.

Multilayer networks tend to take a reasonably long time to train, but once training is done, one has a compact representation of the data set, and function evaluation is fast. The data set no longer needs to be stored, so for large data sets, (for example image recognition data sets), this can be a considerable memory saving. Neural networks also have a pleasing interpretation as a set of “neurons” that sequentially get activated, so they might represent the kinds of prior information relevant to biological / “neuron-like” driven processes.

So far in our study of learning, we have been mostly concerned with the ability to learn and generalize, and the general theory of controlling the test error using regularization techniques. As such, we have been living in a world of infinite computation and infinite memory and discussed only the necessary properties that learning models should have. These issues still play a central role, but we are now also interested in the computational efficiency, as techniques must be applicable to the real world. Since computational resources are rapidly increasing, some of the issues discussed may become mute as time goes on, but the fundamental principles should be clear.

1 The Nearest Neighbor Algorithm

The nearest neighbor learning algorithm is probably the what a first grader would try to do if faced with a learning algorithm. As such, it has certain advantages and disadvantages.

Advantages:

1. Very easy to implement.
2. Simple and intuitive motivation.
3. *NO* training involved.
4. Tends to have good performance, by virtue of simplicity.

Disadvantages:

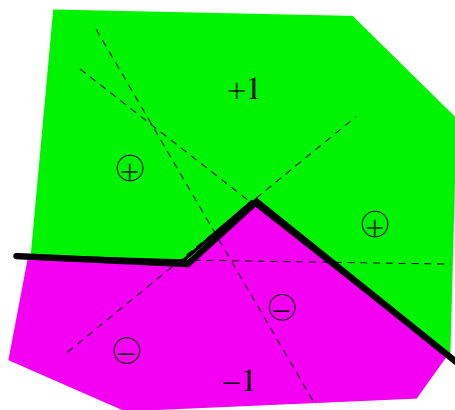
1. Memory and computation requirements for function evaluation are high.
2. “Embarrassingly Simple”. If asked by one’s supervisor to learn from a data set, and in 1 second you return to him and say you are done (by virtue of no training being involved), and then you describe your simple algorithm, the supervisor might wonder what she is paying you so much for 😊.

That having been said, the nearest neighbor algorithm is a very powerful and useful learning model. The idea behind the algorithm is extremely simple. There is no training, and given the data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, the function $g(\mathbf{x})$ is computed by finding the $\mathbf{x}_i \in D$ that is closest (where for the moment we define closest using the Euclidean metric), and outputting $g(\mathbf{x}) = y_i$, the value of the output for that closest \mathbf{x}_j . The formal specification of the algorithm is as follows.

Computation of the Nearest Neighbor Output Function $g(\mathbf{x})$

1. Compute the N “distances” $d_i = d(\mathbf{x}, \mathbf{x}_i)$, where $d(\mathbf{x}, \mathbf{z}) = \sum_j (x_j - z_j)^2$ if one is using the Euclidean metric.
2. Let j^* be such that $d_{j^*} \leq d_j$ for all j . If this j^* is unique, then output y_{j^*} . If the j^* is not unique then one chooses a tie breaking policy (that is usually arbitrary). A reasonable one is to output the value of +1 if the number of +1’s among these nearest neighbors is greater than or equal to the number of -1’s, and output -1 otherwise.

One of the virtues of this method is that there is no training required, and new data points are accommodated instantaneously by simply throwing them into the data set. The output function is fully determined by the data points and does not depend on any parameters that need to be set. As such, this learning model falls into a class of models known as non-parametric. It is clear that the underlying assumption behind the success of this model is that points that are close together tend to have the same function value. In other words, if we were to write the true target function as $f(\mathbf{x}) = \text{sign}(h(\mathbf{x}))$, then $h(\mathbf{x})$ is assumed to be a smooth function in some sense. It is instructive to look at the decision boundaries. An example is shown in the figure to the right. The decision boundary is easily constructed by



considering the perpendicular bisectors of the various pairs of points. In general it should be clear that the decision boundary will be piecewise linear. In multi-dimensions, the boundary becomes piecewise hyper-planar, and becomes hard to visualize. Before we discuss the issues of computation, we first consider how we expect the nearest neighbor learning model to fare. The first observation is that this learning model is not falsifiable - the training error is guaranteed to be zero. This is not a good sign, from the generalization point of view. How bad can the generalization be? We see that the training error is always zero, so we can ask what the test error is (the true probability of error). It turns out that this cannot be too high, for smooth functions. First we need to define what it means to be smooth a little more carefully.

Imagine that there is some true process that generates outputs from inputs. So, given the input \mathbf{x} , there is some probability that the true process outputs $+1$. Call this probability $p(\mathbf{x})$. Then, the optimal decision (target function) is to predict $+1$ if $p(y|x) \geq \frac{1}{2}$ and -1 otherwise. This is the Bayes optimal decision function if the risk is the probability of error. In other words, $f(\mathbf{x}) = \text{sign}\left(p(\mathbf{x}) - \frac{1}{2}\right)$. We will say that $f(\mathbf{x})$ is smooth if $p(\mathbf{x})$ is a continuous function of \mathbf{x} . What is the probability of error, given \mathbf{x} . Call this $\pi(\mathbf{x})$. The average probability of error is given by $\int d\mathbf{x} \pi(\mathbf{x})$. If we use the Bayes optimal rule, the probability of error is

$$\pi_B(\mathbf{x}) = \min\{p(\mathbf{x}), 1 - p(\mathbf{x})\} \quad (1)$$

If we use the nearest neighbor classifier, suppose that the nearest neighbor to \mathbf{x} is the \mathbf{x}_{nn} , then an error occurs in one of two situations:

1. $y(\mathbf{x}) = +1$ and $y_{nn} = -1$. The probability that this happens is $p(\mathbf{x})(1 - p(\mathbf{x}_{nn}))$.
2. $y(\mathbf{x}) = -1$ and $y_{nn} = +1$. The probability that this happens is $(1 - p(\mathbf{x}))p(\mathbf{x}_{nn})$.

The probability of error will thus be the sum of these two terms,

$$\pi_{nn}(\mathbf{x}) = p(\mathbf{x})(1 - p(\mathbf{x}_{nn})) + (1 - p(\mathbf{x}))p(\mathbf{x}_{nn}) \quad (2)$$

We now consider the case that $N \rightarrow \infty$. In this case, the nearest neighbor to x will approach \mathbf{x} , i.e., $\mathbf{x}_{nn} \xrightarrow{N \rightarrow \infty} \mathbf{x}$, and because $p(\mathbf{x})$ is assumed continuous, $p(\mathbf{x}_{nn}) \xrightarrow{N \rightarrow \infty} p(\mathbf{x})$ we see that

$$\pi_{nn}(\mathbf{x}) \xrightarrow{N \rightarrow \infty} 2p(\mathbf{x})(1 - p(\mathbf{x})) = 2\pi_B(\mathbf{x})(1 - \pi_B(\mathbf{x})) \leq 2\pi_B(\mathbf{x}) \quad (3)$$

Integrating (3), we find that in the limit $N \rightarrow \infty$, the test probability of error is also bounded by the Bayes test probability of error as follows

$$\pi_{nn} = \int d\mathbf{x} \pi_{nn}(\mathbf{x}) = 2 \int d\mathbf{x} \pi_B(\mathbf{x})(1 - \pi_B(\mathbf{x})) \leq 2 \int d\mathbf{x} \pi_B(\mathbf{x}) - 2 \int d\mathbf{x} \pi_B^2(\mathbf{x}) \leq \pi_B(1 - \pi_B) \leq 2\pi_B \quad (4)$$

Thus, we see that despite being unfalsifiable, the Nearest Neighbor method cannot have too bad a test error. However, it is unsettling that the training error has absolutely nothing to do with the test error. We thus consider the following generalization.

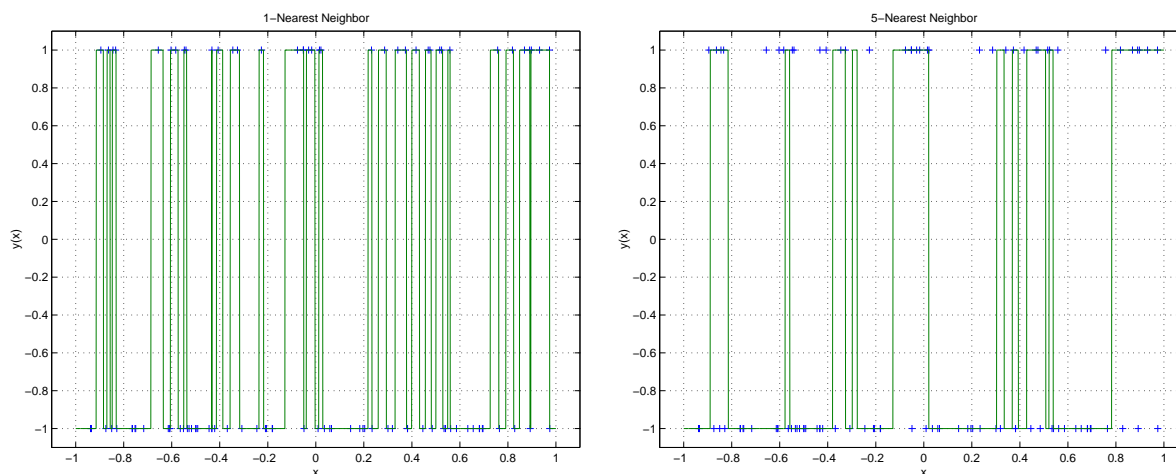
2 The K -Nearest Neighbor Learning Algorithm

Instead of only looking at the nearest neighbor, one looks at the K - nearest neighbors, where K is some fixed number, usually chosen to be an odd number.

Computation of the K -Nearest Neighbor Output Function $g(\mathbf{x})$

1. Compute the N “distances” $d_i = d(\mathbf{x}, \mathbf{x}_i)$, where $d(\mathbf{x}, \mathbf{z}) = \sum_j (x_j - z_j)^2$ if one is using the Euclidean metric.
2. Let d_{j^*} be such that $d_j \geq d_{j^*}$ for at least $N - K + 1$ of the points and $d_j \leq d_{j^*}$ for at least K of the points. Usually this j^* is unique. The output is $+1$ if the number of $+1$'s among the $\geq K$ points with $d_j \leq d_{j^*}$ is greater or equal to the number of -1 's. Otherwise output -1 .

This learning model is clearly falsifiable for $K > 1$. Take for example $K = 2$. With 2 data points, one of which is -1 and the other $+1$, the data point that is -1 is misclassified. The effect of the picking $K > 1$ is to “average” over the K nearby points and has the effect of smoothening. This is mostly useful when there is noise in the data or when the prior information is that the target function (the Bayes optimal decision function) is known to be smooth. One way to pick the parameter K is to use cross validation.



The figure above illustrates this smoothening effect in 1 dimension, comparing the output function of 1-Nearest Neighbor and 5-Nearest Neighbor.

How about the accuracy of the K nearest neighbor. Two very powerful theorems can be shown which we give the intuition for. In the limit $N \rightarrow \infty$, if $K \rightarrow \infty$ and $K/N \rightarrow 0$, so for example if we pick $K = \sqrt{N}$, then

1. The training probability of error will approach the test probability of error.
2. The test probability of error will approach the Bayes optimal risk, i.e., $\pi_{nn} \rightarrow \pi_B$.

All this is providing that $p(\mathbf{x})$ is continuous. Thus, it appears that not only are we guaranteed good generalization, but we are guaranteed good test error, under a certain kind of prior information. This is generally a characteristic of non-parametric methods. It should be pointed out that a similar kind of result holds for neural networks for a similar kind of prior information. For example, if there is one hidden layer, and the activation function is non-polynomial, then if $N_H \rightarrow \infty$, where N_H is the number of hidden units, such that $N_H/N \rightarrow 0$ then a similar result holds.

We present the intuition for the result. Consider a point \mathbf{x} and a region in around the point of size ϵ . A point will land in this region with probability proportional to ϵ . Thus about $N\epsilon$ points

will land in this region, and since $K/N \rightarrow 0$, the K nearest neighbors will all land in this region ϵ . If we can choose this region to be arbitrarily small so that $p(\mathbf{x})$ is effectively constant, hence, the fraction of points in this region that are $+1$ will be $\sim p(\mathbf{x})$. Thus we will be classifying according to the true $p(\mathbf{x})$ as $N \rightarrow \infty$, and thus will achieve the Bayes optimal risk. Further, training error (the number of misclassified points) will be approaching this true risk. We do not present the details for a rigorous proof, but the intuition should be clear.

3 Complexity of K -Nearest Neighbor

The reader may have already noticed two of the major pitfalls of the K -Nearest Neighbor algorithm.

1. In order to perform a classification, we need to compute all the N distances. This means that all the data points need to be stored. In d dimensions, this is an $O(Nd)$ memory requirement, which can be quite large.
2. In order to compute the K nearest neighbors, suppose that one uses a priority queue. Then the time to perform this calculation would be $O((N + K) \log K)$ in the worst case.

Thus, the memory requirement is order N and the compute time for function evaluation is order N for fixed K . So, for example, to compute the training error for a given K , one would need to evaluate N^2 $d(\mathbf{x}_i, \mathbf{x}_j)$'s, an order dN^2 operation, and for each data point, one would have to perform $O((N + K) \log K)$ comparisons to get the K nearest neighbors, yielding a total complexity of $O(dN^2 + N(N + K) \log K)$ just to compute the training error. This is largely unacceptable and is the main reason why this algorithm is not used more often in practice. In order to make the algorithm more practical, we need to address these two issues. We address the storage issue using a technique called *data editing* and we address the computation issues using a *branch and bound* technique.

3.1 Data Editing

There are two forms of data editing that we will present. The basic idea is to decrease the size of the data set that one has to store, and will usually result in a change in performance.

3.1.1 Conditional Means in a Partition of the Space

Assuming that the space in which the data lie can be considered compact, this space can be partitioned into regions. Call these regions R_1, \dots, R_M . Within each region i , one collects up the positive data points (supposing there are n_i^+ of them) and computes their mean, μ_i^+ , and similarly, one computes the mean of the n_i^- negative data points in this region, μ_i^- . If no positive or negative data points exist in the region, then there will be no associated μ_i^+ or μ_i^- . The entire data set is now replaced by the *edited* data set

$$D_E = \{(\mu_1^+, 1, n_1^+), (\mu_1^-, 1, n_1^-), \dots, (\mu_M^+, 1, n_M^+), (\mu_M^-, 1, n_M^-)\} \quad (5)$$

One uses this data set as the basis of the new classifier. The $(\mu_i^\pm, \pm 1)$ can be viewed as the new data points, and the n_i^\pm can be viewed as weightings. In the simplest case, one ignores the weightings and classifies according to the 1-Nearest Neighbor rule using this data set. One could use the

weightings to classify according to a K -Nearest Neighbor rule, where for example if the nearest point to \mathbf{x} were $(\mu_7^+, +1, 4)$, that would represent 4 positive points at μ_7^+ . Thus, if one were (say) using the 5-Nearest Neighbor rule, this point would account for 4 of the nearest neighbors and one could classify the point as +1, without even searching for a fifth neighbor.

Under suitable regularity conditions, such approaches can be shown to give comparable results (in terms of test error) to the original rule, however, we do not discuss these further.

3.1.2 Data Condensing

The purpose of this form of editing is to construct a new, smaller data set that performs approximately the same classification as the original set. Suppose one is using the K -Nearest Neighbor rule. Then, the idea is to throw away as many points as is possible, such that if one were to classify *all* the points using the K -Nearest Neighbor rule based *only* the retained set, the resulting classifications would be the same as if one classified all the data based upon the the entire data set.

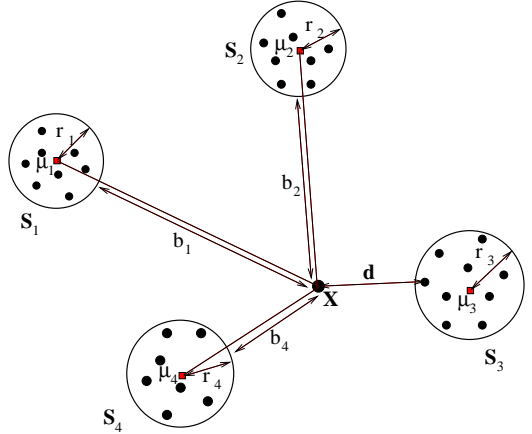
An iterative algorithm to approximately achieve this goal is as follows. It will be useful to imagine two bags, a *store* bag, S , and a *grab* bag, G . The store bag will eventually be retained, and the grab bag will contain the points that have been discarded. The main idea is to iteratively discard the data points from the store bag that are not needed to classify themselves “correctly”. Now examine the discarded points to check which ones continue are “incorrectly” classified and restore them to the store bag. Here we use “correct” defined with respect to the original classification.

1. Initialize the store bag to all the data points, and the grab bag to empty. Classify all the data points according to the entire data set (using the K -Nearest Neighbor rule) to obtain the initial classifications y_j
2. *Grab Step*: Cycle through the data points in the store bag ($\mathbf{x}_i \in S$), sequentially classifying each according to the remaining data points in the store bag (excluding itself), to obtain the classification y_i^S . If $y_i^S = y_i$, the initial classification, then delete \mathbf{x}_i from the store bag and place it in the grab bag. Continue this process until all the points have been cycled through once.
3. *Store Step*: Cycle through the data points in the grab bag ($\mathbf{x}_j \in G$), and sequentially classify the data point using the points that in the store bag, to obtain the classification y_j^G . If $y_j^S \neq y_j$, then remove the point from the grab bag and place it in the store bag. Continue this process until all the points have been cycled through once.
4. Iterate back to step 2 until the sets stop changing or some other stopping criterion is reached.

3.2 The Branch and Bound Algorithm

If the data set is stored in an unprincipled way, then it will be necessary to look at all the data points before we come up with (say) the nearest neighbor. However, if the data were stored in a more ordered fashion, it is possible that on average, we may not have to look at all the points. We will describe the algorithm in the context of searching for the nearest neighbor, but it should be clear that the algorithm can be extended to compute the K nearest neighbors. The main idea is to store the the data in a tree like data structure so that the search for the nearest neighbor can be performed efficiently by not having to search down branches where one can obtain a lower bound for the distance that is higher than the current best found.

The basic idea is to cluster the data into representative groups, S_j , where each group has the smallest possible radius. This is illustrated in the figure to the right, where the data is clustered into 4 clusters. Suppose that this has been achieved (we will discuss a technique for achieving this later). The idea is that associated to each cluster is the center of the cluster, μ_j , and the the radius of the cluster, r_j . Suppose that one has searched one of the clusters to emerge with a candidate point as the nearest neighbor, call this point \mathbf{x}_{j^*} . Let $d(\mathbf{x}, \mathbf{x}_{j^*}) = d$ be the distance from the candidate nearest neighbor to the point of interest, \mathbf{x} . We can now compute lower bounds for the distances in the other clusters as follows. Suppose that $\mathbf{x}_k \in S_j$, then, by the triangle inequality, $d(\mathbf{x}, \mu_j) \leq d(\mathbf{x}, \mathbf{x}_k) + d(\mathbf{x}_k, \mu_j) \leq d(\mathbf{x}, \mathbf{x}_j) + r_j$, hence we can compute a lower bound b_j for the distance between any $\mathbf{x}_k \in S_j$ and \mathbf{x} as follows,



$$\min_{x_k \in S_j} d(\mathbf{x}, \mathbf{x}_k) \geq b_j = d(\mathbf{x}, \mu_j) - r_j \quad (6)$$

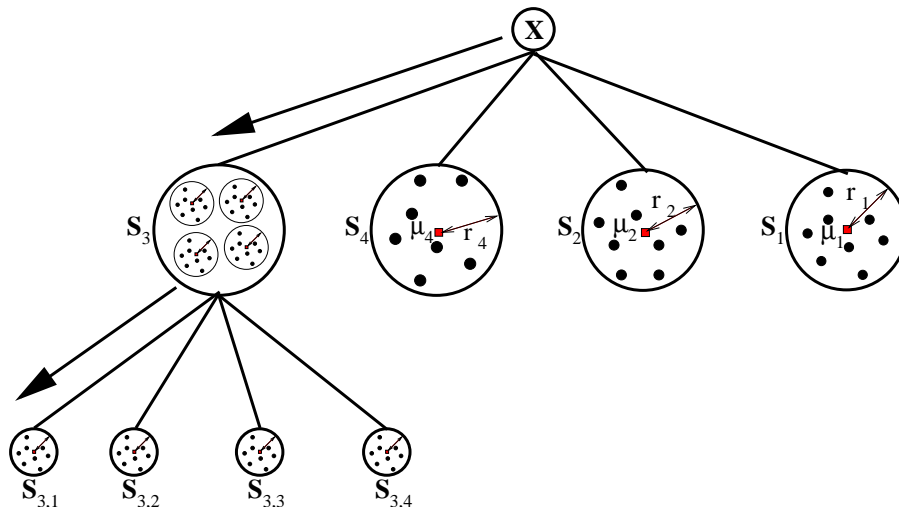
. Thus we do not need to search through clusters that satisfy the inequality $b_j \geq d$, the current closest distance. Further, there is no reason not to have an exactly analogous cluster structure within the clusters themselves, and so on, hence the following recursive algorithm appears.

Branch and Bound Algorithm for Computation of the Nearest Neighbor

1. Create the data structure for storing the points as follows. Cluster the points into L sets, S_i , (say) $L = 4$ clusters. These are the first level clusters. For each cluster, maintain the centers, μ_i , and the radii, r_i . Within each cluster, S_i , cluster the data into L sub clusters $S_{i,j}$ for $j = 1, \dots, L$. Maintain the centers, $\mu_{i,j}$ and the radii, $r_{i,j}$. These are the second level clusters. Continue this recursive clustering until one gets down to clusters of size one point.
2. To find the nearest neighbor to a new point, \mathbf{x} , there are two steps
 - (a) *Branch Step*: First compute the b_j . Then find the nearest neighbor in the cluster with the smallest b_j . This is clear if the cluster only has points, and if it has clusters itself, then this is step is done recursively.
 - (b) *Bound Step*: Look into the cluster with the next highest b_j only if that cluster does not satisfy the bound (6), otherwise stop and output the result.

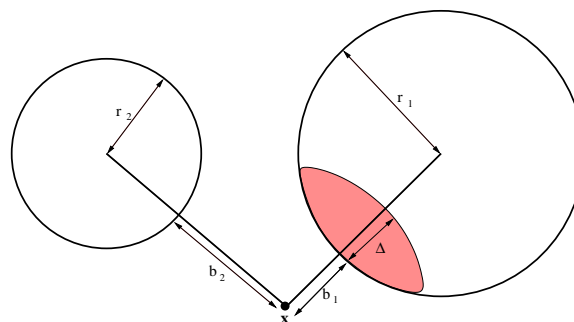
It is clear that the search can be posed on a tree as follows. Each of the L first level clusters form the first L possible branches of the tree. The second level clusters for a given first level cluster represents the possible branches after having taken the the first branch leading to this first level cluster. We take the first branch that minimizes the b_j . If one sorts the branches so that the minimum b_i is always on the left, then the first path traversed in this tree is the left most path. A path from a root to a leaf identifies a point that is a candidate for the nearest neighbor. One

then backtracks up one branch and checks if the other branches can be bounded. If so, then one can back track further. If not, one searches the other branch from the leftmost path recursively. Eventually, one backtracks all the way up to the root (x) and has either bounded or searched each possible branch at which point one has found the nearest neighbor. This is illustrated in the figure below.



Advanced Topic: Computational Complexity of the Branch and Bound Algorithm.

It is clear that in the worst case, one might have to search the entire tree, hence taking $O(N)$ time, which is no better than could have been done using other algorithms. We thus consider the average case analysis and present an informal argument. We consider the case that L , the number of clusters at each level is 2. If the clusters are “well formed”, then the number of points in each of the first level clusters is about $N/2$ and in the second level it is expected to be $N/4$ and so on. Thus, the depth of the tree on average will be $\log_2 N$. We now consider the probability that the search down the first branch will result in the the second branch being bounded and not having to be searched. Without loss of generality suppose that the first search went into S_1 , then $b_2 > b_1$. Thus write $b_2 = b_1 + \delta$. Since we assume that x is random and we like to get the average case behavior, averaged over x , we see that Δ is a random variable, depending on x . The situation is illustrated in the following figure.



The probability that the second branch be bounded is the probability that the closest point in the first cluster lands in the shaded region. If $\Delta > 2r_1$ then the probability is 1. If not then

the probability is given by the ratio of the volume of the shaded region to the volume of the r_1 d-dimensional sphere, where we assume that the distribution of the minimum is uniform within the sphere. The \mathbf{x} 's for which $\Delta > 2r_1$ roughly defines a half space, and assuming a uniform or near uniform probability distribution for \mathbf{x} , the probability that this occurs is roughly 1/2. Thus the probability that the second cluster be bounded is given roughly by

$$P[\text{bound}] \approx \frac{1}{2}(1 + P[\text{bound}|\Delta < 2r_1]) \quad (7)$$

which is greater than 1/2. We will, for ease of notation, call this probability P . Since \mathbf{x} is random, this picture remains the same, no matter what cluster level we are at, and, thus, the probability that the second cluster gets bounded is the same independent of the cluster level. In order to decide which branch to go down first only takes 2 computations of b_i 's and one comparison, so is $O(1)$. We develop a recursion for the expected computation time C_N for N data points as follows. With probability $P > 1/2$, the second branch is bounded and we have to do $C_{N/2} + O(1)$ computations, since the search in the first cluster involves roughly $N/2$ points. With probability $1 - P < 1/2$, the second branch is not bounded and we have to do $2C_{N/2} + O(1)$ computations. Thus we find that the expected number of computations to satisfy the following recursion

$$C_N = (2 - P)C_{N/2} + O(1) \quad (8)$$

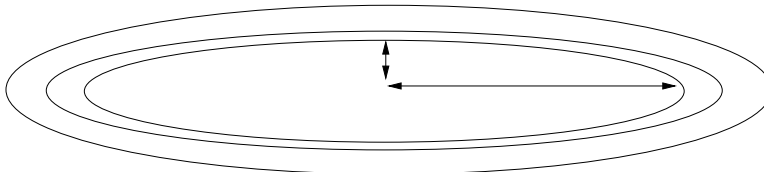
Solving this recursion, the details of which we do not present, one finds that

$$C_N = O(N^{\log_2(2-P)}) \quad (9)$$

and since $1 > P > 1/2$, $\log_2(2 - P)$ can be significantly less than 1, giving considerable improvement over the standard $O(N)$ algorithm.

3.3 Practical Issues

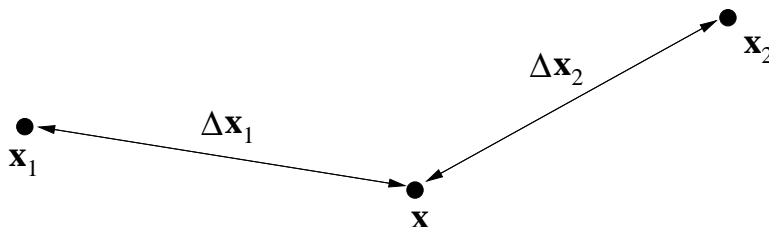
All along, we have been making use of the metric, without discussing it in any great detail. The choice of metric can have a considerable effect on the performance. The intuition is shown in the following figure.



If the input distribution is heavily asymmetric as shown above, then the vertical displacement is “equivalent” to the horizontal displacement shown above. But the standard euclidean distance that we have been using up to now treats them totally differently. For this reason, it is often better to use a metric that takes the input distribution into account. Such a metric is the Mahanalobis distance, which we do not discuss here. The reason is that this metric issue disappears if one has performed input normalization, which should do!

However, on a different level, one could try to determine an optimal metric in the following sense. We would like to classify according to a “nearest” neighbor. In otherwords we want to find

a neighbor such that we expect the function value to have changed least and classify according to that neighbor.



Consider the neighbors of \mathbf{x} shown above. The change in f in coming from \mathbf{x}_1 is given to first order by $\Delta f_1 \approx \mathbf{g}_1^T \Delta \mathbf{x}_1$, where \mathbf{g}_1 is the gradient of f at x_1 . The change in f in coming from \mathbf{x}_2 is given to first order by $\Delta f_2 \approx \mathbf{g}_2^T \Delta \mathbf{x}_2$. Thus we should compare $|\Delta f_1|$ to $|\Delta f_2|$ and if $|\Delta f_1| < |\Delta f_2|$, this means that $f(\mathbf{x})$ is closer to $f(\mathbf{x}_1)$ than $f(\mathbf{x}_2)$ and hence we should classify according to \mathbf{x}_1 . Thus we should compare distance based upon $\Delta \mathbf{x}_i \mathbf{g}_i \mathbf{g}_i^T \Delta \mathbf{x}_i$. We do not know \mathbf{g}_i , but we can try to estimate $E[\mathbf{g}\mathbf{g}^T]$ as follows. We would like to obtain an “average” \mathbf{g} that relates changes in y to changes in \mathbf{x} . Thus over all distinct pairs, we would like $y_i - y_j = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{g}$. The least squares solution to this set of equations is given by

$$\mathbf{g} = \Sigma^{-1} \mathbf{q} \quad (10)$$

where

$$\Sigma = E[\Delta \mathbf{x} \Delta \mathbf{x}^T] \quad (11)$$

is the covariance matrix and can be estimated from the data and

$$\mathbf{q} = E[\Delta y \Delta \mathbf{x}] \quad (12)$$

is the correlation between $\Delta \mathbf{x}$ and Δy and can also be estimated from the data. Thus the “optimal” metric has the form

$$\Delta \mathbf{x}^T \mathbf{D} \Delta \mathbf{x} \quad (13)$$

which reduces to the Euclidean metric if $\mathbf{D} = \mathbf{I}$ but is in general some positive definite symmetric matrix.

4 The Nearest Neighbor Revisited: r -Near Neighbors

One complaint that may still remain is that the nearest neighbor may be very far away. In this event, one expects the nearest neighbor to have very little to do with what the function value should be at the point of interest. It might be better to assign some default value, such as the majority over the entire data set. For this reason, it is worth considering an alternate though very similar approach.

Instead of looking at nearest neighbors (say K of them), rather, look at *all* the neighbors within some distance of the point of interest. If these points are within a distance r of the point of interest, then we say that they are r -near neighbors. The algorithm is as follows.

r -Near Neighbor Computation of Output

1. Given the point \mathbf{x} , determine the subset of the data that lie in the ball of radius r at \mathbf{x} ,

$$B_r(x) = \{\mathbf{x}_i \in D \text{ s.t. } \|\mathbf{x} - \mathbf{x}_i\| \leq r\} \quad (14)$$

2. If $B_r(x)$ is empty, then output $+1$ if the number of $+1$ data points in the entire data set is greater than or equal to the number of -1 data points.
3. If $B_r(x)$ is not empty, then output $+1$ if the number of $+1$ data points in $B_r(\mathbf{x})$ is greater than or equal to the number of -1 data points in $B_r(\mathbf{x})$.

As with the K -Nearest Neighbor method, a theorem regarding the accuracy of the r -Nearest Neighbor method also holds. Namely, if $r \rightarrow 0$ and $Nr^d \rightarrow \infty$, so for example $r = N^{-1/2d}$, then

1. The training probability of error will approach the test probability of error.
2. The test probability of error will approach the Bayes optimal risk, i.e., $\pi_{nn} \rightarrow \pi_B$.

The intuition is that r is going to zero so that points in $B_r(\mathbf{x})$ will reflect $p(\mathbf{x})$. Further, r is not decaying to zero too fast, guaranteeing that there are many (infinitely many as $N \rightarrow \infty$) points in $B_r(\mathbf{x})$ hence the estimate of $p(\mathbf{x})$ becomes increasingly more accurate, which is essentially the intuition behind the proof of the two claims.

5 The Nearest Neighbor Method for Regression

We have only discussed the nearest neighbor method in the context of two class classification. It is clear that the extension to multi-class is not hard and regression (which is a “limit” of multi-class) is also not hard. For regression, instead of taking a majority vote (which would not really make sense for regression), one could take an average of the K -nearest neighbors or the r -near neighbors. Theorems regarding the accuracy for regression under given error measures can also be proved. However we do not go into the technicalities here.