CS 760: Machine Learning Spring 2022

Topic 9: Nearest Neighbors

Instructor: Daniel L. Pimentel-Alarcón

DO NOT POLLUTE! AVOID PRINTING, OR PRINT 2-SIDED MULTIPAGE.

9.1 Introduction

One of the most fundamental problems in machine learning is classification, which can be summarized as assigning a label (class) $\mathrm{y} \in \{1, 2, \ldots, \mathrm{C}\} =: [\mathrm{C}]$ to a data point $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$. For example:

- Given a data vector $\mathbf{x}$ containing an individual's information (e.g., genome or demographics), determine whether such individual has Alzheimer's ($\mathrm{y} = 1$) or not ($\mathrm{y} = 0$).
- Given a vectorized image $\mathbf{x}$ of a human face, determine which person (y) amongst a database of C individuals is depicted in $\mathbf{x}$.
- Given a vectorized image $\mathbf{x}$ of a character, determine which symbol (y) amongst an alphabet of size C corresponds to $\mathbf{x}$.

Nearest neighbors is one of the simplest classification methods. The main idea is to assign each data point to the class of its most *similar* pre-classified points. Since there are several ways to measure similarity, there are also several flavors of nearest neighbors. In all of them, we assume we already have a collection of training data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\mathrm{N} \in \mathbb{R}^{\mathrm{D}}$ whose corresponding classes $\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{N} \in [\mathrm{C}]$ are already known. Given a new data point $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$ whose class is unknown, the goal is to determine its corresponding class $\mathrm{y} \in [\mathrm{C}]$.

9.2 Nearest Neighbor

The simplest form of nearest neighbors simply assigns the new point to the class of its *closest* point in our data, that is, $\mathrm{y} = \mathrm{y}_\mathrm{i}$, where

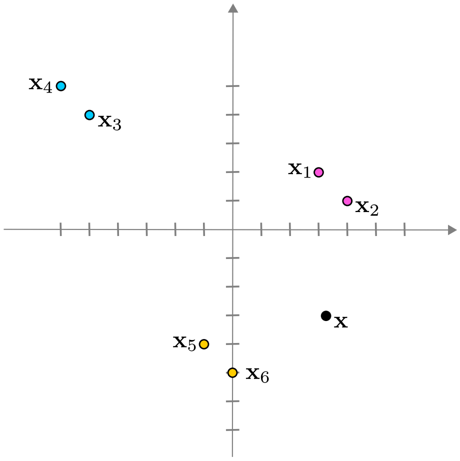
$$\mathrm{i} \;=\; \underset{\mathrm{i} \in [\mathrm{N}]}{\arg\min}\; \delta(\mathbf{x}, \mathbf{x}_\mathrm{i}). \tag{9.1}$$

Notice that there are several ways to measure distance, and depending on the application one may be better than other. Arguably the most widely used distances for continuous features are the euclidean distance $\|\mathbf{x} - \mathbf{x}_\mathrm{i}\|_2$ and the Manhattan distance $\|\mathbf{x} - \mathbf{x}_\mathrm{i}\|_1$. Similarly, for discrete features one of the most popular choices is the Hamming distance $\|\mathbf{x} - \mathbf{x}_\mathrm{i}\|_0$. We point out that $\| \cdot \|_0$ is only *called* a norm, but really isn't; do you know which property of a norm it does not satisfy?

Example 9.1. Consider the following training data:

$$\mathbf{x}_1 = \begin{bmatrix}3\\2\end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix}4\\1\end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix}-5\\4\end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix}-6\\5\end{bmatrix} \quad \mathbf{x}_5 = \begin{bmatrix}-1\\-4\end{bmatrix} \quad \mathbf{x}_6 = \begin{bmatrix}0\\-5\end{bmatrix},$$

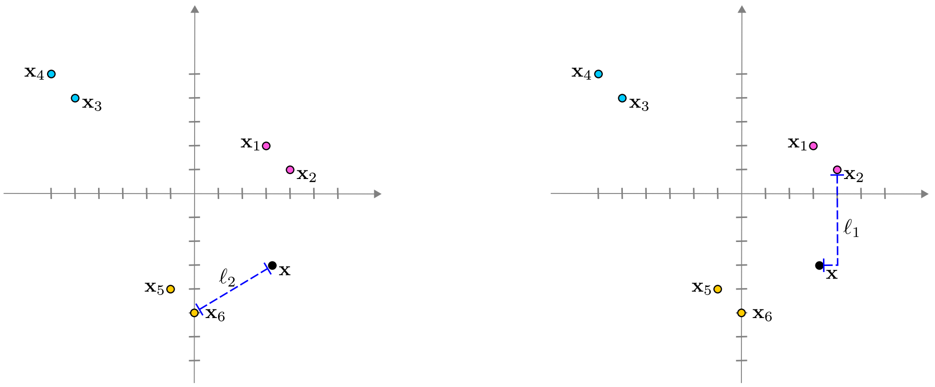
with classes $\mathrm{y}_1 = 1$, $\mathrm{y}_2 = 1$, $\mathrm{y}_3 = 2$, $\mathrm{y}_4 = 2$, $\mathrm{y}_5 = 3$, $\mathrm{y}_6 = 3$ and suppose you observe an additional sample $\mathbf{x} = [3.25 \quad -3]^\mathsf{T}$:



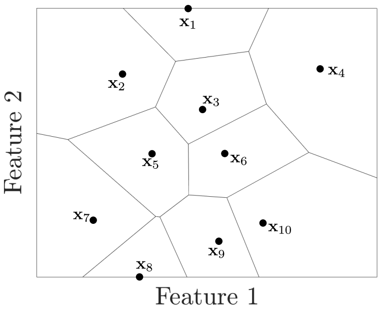
Then $\|\mathbf{x} - \mathbf{x}_1\| = \sqrt{(3.25-3)^2 + (-3-2)^2} = 5.0062$. Similarly

$$\|\mathbf{x} - \mathbf{x}_2\| = 4.07, \quad \|\mathbf{x} - \mathbf{x}_3\| = 10.82, \quad \|\mathbf{x} - \mathbf{x}_4\| = 12.23, \quad \|\mathbf{x} - \mathbf{x}_5\| = 4.37, \quad \|\mathbf{x} - \mathbf{x}_6\| = 3.82.$$

Since $\mathbf{x}$ is closest to $\mathbf{x}_6$, we assign $\mathbf{x}$ to the same class as $\mathbf{x}_6$, i.e., $\mathrm{y} = 3$. Notice, however, that if we use the ℓ_1-norm, then $\|\mathbf{x}-\mathbf{x}_1\| = 5.25$, $\|\mathbf{x}-\mathbf{x}_2\| = 4.75$, $\|\mathbf{x}-\mathbf{x}_3\| = 15.25$, $\|\mathbf{x}-\mathbf{x}_4\| = 17.25$, $\|\mathbf{x}-\mathbf{x}_5\| = 5.25$, and $\|\mathbf{x} - \mathbf{x}_6\| = 5.25$, whence $\mathbf{x}$ is closest to $\mathbf{x}_2$, in which case we conclude $\mathrm{y} = 1$:



The decision regions are often depicted in a *Voronoi diagram*, where each polyhedron indicates the region of the feature space corresponding to each training sample:



9.3 K-Nearest Neighbors

Using a single neighbor as reference can be quite sensitive to noise. Hence, to make our nearest neighbor algorithm more robust, we can simply use a *consensus* approach: rather than using a single neighbor, we an use K, and assign $\mathbf{x}$ to the class with the most votes. In this case, the K-*nearest neighbors* algorithm can be summarized as follows:

Algorithm 1: K-Nearest Neighbors

Input: Training pairs $\{\mathbf{x}_\mathrm{i}, \mathrm{y}_\mathrm{i}\}_{\mathrm{i}=1}^{\mathrm{N}}$, and new datapoint $\mathbf{x}$.
Compute distances: For each i, compute $\delta(\mathbf{x}, \mathbf{x}_\mathrm{i})$.
Find K **nearest neighbors:** Let $\mathbf{x}_{(\mathrm{k})}$ denote the k^{th} closest neighbor to $\mathbf{x}$, and let $\mathrm{y}_{(\mathrm{k})}$ denote its corresponding class, such that

$$\delta(\mathbf{x}, \mathbf{x}_{(1)}) \;\leq\; \delta(\mathbf{x}, \mathbf{x}_{(2)}) \;\leq\; \cdots \;\leq\; \delta(\mathbf{x}, \mathbf{x}_{(\mathrm{N})}).$$

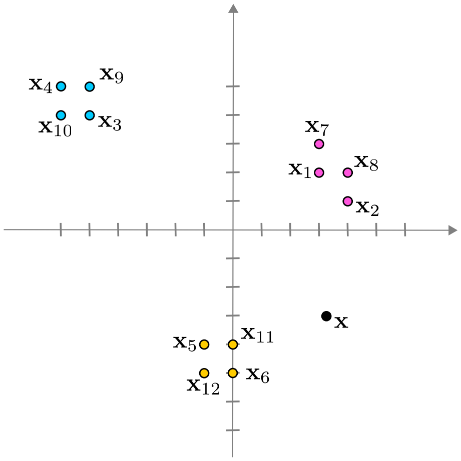
Output: Assign new point to the class of the majority amongst nearest neighbors, i.e.,

$$\hat{\mathrm{y}} \;=\; \arg\max_{c \in [\mathrm{C}]} \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathbb{1}_{\{c = \mathrm{y}_{(\mathrm{k})}\}} \tag{9.2}$$

Example 9.2. In addition to the data in Example 9.1, suppose we also have:

$$\mathbf{x}_7 \;=\; \begin{bmatrix} 3 \\ 3 \end{bmatrix} \qquad \mathbf{x}_8 \;=\; \begin{bmatrix} 4 \\ 2 \end{bmatrix} \qquad \mathbf{x}_9 \;=\; \begin{bmatrix} -5 \\ 5 \end{bmatrix} \qquad \mathbf{x}_{10} \;=\; \begin{bmatrix} -6 \\ 4 \end{bmatrix} \qquad \mathbf{x}_{11} \;=\; \begin{bmatrix} 0 \\ -4 \end{bmatrix} \qquad \mathbf{x}_{12} \;=\; \begin{bmatrix} -1 \\ -5 \end{bmatrix},$$

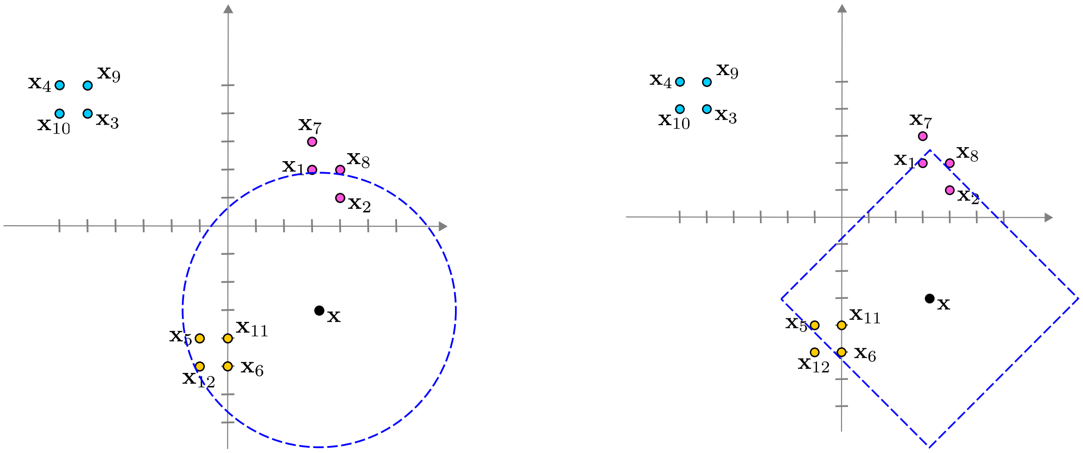
with classes $\{1, 1, 2, 2, 3, 3\}$:



The distances to $\mathbf{x}$ are:

$$\begin{aligned} &\|\mathbf{x}-\mathbf{x}_7\| = 6.01, && \|\mathbf{x}-\mathbf{x}_8\| = 5.06, && \|\mathbf{x}-\mathbf{x}_9\| = 11.49, \\ &\|\mathbf{x}-\mathbf{x}_{10}\| = 11.60, && \|\mathbf{x}-\mathbf{x}_{11}\| = 3.40, && \|\mathbf{x}-\mathbf{x}_{12}\| = 4.6971. \end{aligned}$$

We thus conclude that the closest K = 5 points (with respect to the ℓ_2-norm) are $\{11, 6, 2, 5, 12\}$, which belong to classes $\{3, 3, 1, 3, 3\}$. By consensus, we conclude that $\mathbf{x}$ belongs to class 3. You can verify that if we use the ℓ_1 norm, the closest K = 5 points are $\{11, 2, 1, 5, 6\}$, which belong to classes $\{3, 1, 1, 3, 3\}$, and hence this time we also conclude by consensus that $\mathbf{x}$ belongs to class 3:



Notice that the shape of the ℓ_2-ball (boundary with all equidistant points) is a circle, while the shape of the ℓ_1-ball is a diamond!

Also notice that I chose to use K = 5 neighbors. Why 5? Why not 3, or 6, or any other number? Can you think of a strategy to determine the best value for K?

9.4 Nearest Neighbors Regression

The main intuition behind nearest neighbors classification can also be used for *regression.* The only difference is that instead of returning the consensus class in (9.2), we would return the average response of the nearest neighbors:

$$\hat{\mathrm{y}} \;=\; \frac{1}{\mathrm{K}}\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{y}_{(\mathrm{k})}. \tag{9.3}$$

Nearest neighbors is such a simple algorithm that one can easily mix it with others. For example, one can do *local linear regression*, which identifies the parameter $\boldsymbol{\theta}$ that best approximate the response of nearest neighbors as a linear function of their features:

$$\hat{\boldsymbol{\theta}} \;=\; \arg\min_{\boldsymbol{\theta}} \sum_{\mathrm{k}=1}^{\mathrm{K}} \|\mathrm{y}_{(\mathrm{k})} - \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}_{(\mathrm{k})}\|_2 \;=\; (\mathbf{X}_{\mathrm{K}}^{\mathsf{T}}\mathbf{X}_{\mathrm{K}})^{-1}\mathbf{X}_{\mathrm{K}}^{\mathsf{T}}\mathbf{y}_{\mathrm{K}},$$

where $\mathbf{y}_{\mathrm{K}}$ and $\mathbf{X}_{\mathrm{K}}$ are formed as in linear regression, but restricted to the K nearest neighbors:

$$\mathbf{y}_{\mathrm{K}} \;:=\; \begin{bmatrix}\mathrm{y}_{(1)} & \mathrm{y}_{(2)} & \cdots & \mathrm{y}_{(\mathrm{K})}\end{bmatrix}^{\mathsf{T}}, \qquad \mathbf{X}_{\mathrm{K}} \;:=\; \begin{bmatrix}\mathbf{x}_{(1)} & \mathbf{x}_{(2)} & \cdots & \mathbf{x}_{(\mathrm{K})}\end{bmatrix}^{\mathsf{T}}.$$

Once $\hat{\boldsymbol{\theta}}$ is identified, one can predict the response the same as in linear regression:

$$\hat{\mathrm{y}} \;=\; \mathbf{x}^{\mathsf{T}}\hat{\boldsymbol{\theta}}.$$

One can also get creative and find parameters that best approximate the weighted response:

$$\hat{\boldsymbol{\theta}} \;=\; \arg\min_{\boldsymbol{\theta}} \sum_{\mathrm{k}=1}^{\mathrm{K}} w_{(\mathrm{k})}\|\mathrm{y}_{(\mathrm{k})} - \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}_{(\mathrm{k})}\|_2,$$

where $w_{(\mathrm{k})}$ is the weight assigned to the k^{th} nearest neighbor. Popular choices involve inverse functions of distance, for example

$$w_{(\mathrm{k})} \;=\; \frac{1}{\delta(\mathbf{x}, \mathbf{x}_{(k)})}, \qquad \text{or} \qquad w_{(\mathrm{k})} \;=\; e^{-\delta(\mathbf{x},\mathbf{x}_{(k)})^2}.$$

This is often called *locally weighted regression.*

Another option is to weight the contribution of each nearest neighbor according to its distance from the new sample, so that instead of (9.3), the prediction is given by the weighted average of the response of the nearest neighbors:

$$\hat{\mathrm{y}} \;=\; \frac{\sum_{\mathrm{k}=1}^{\mathrm{K}} w_{(\mathrm{k})}\mathrm{y}_{(\mathrm{k})}}{\sum_{\mathrm{k}=1}^{\mathrm{K}} w_{(\mathrm{k})}}.$$

Similarly, for classification, instead of (9.2) we could choose the class with the largest weighted presence:

$$\hat{\mathrm{y}} \;=\; \arg\max_{c\in[\mathrm{C}]} \sum_{\mathrm{k}=1}^{\mathrm{K}} w_{(\mathrm{k})}\mathbb{1}_{\{c=\mathrm{y}_{(\mathrm{k})}\}}.$$

This is often called *distance-weighted nearest neighbors.*

9.5 Speeding up K-Nearest Neighbors

There are two general strategies to speed up this algorithm: subsampling and efficient search.

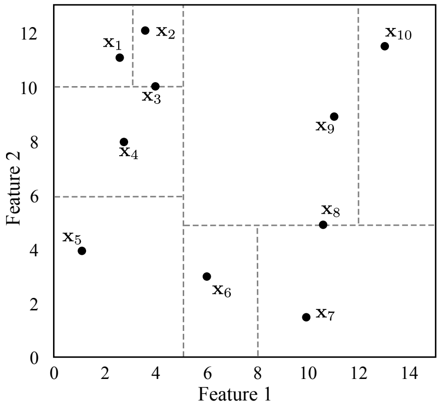
9.5.1 Subsampling

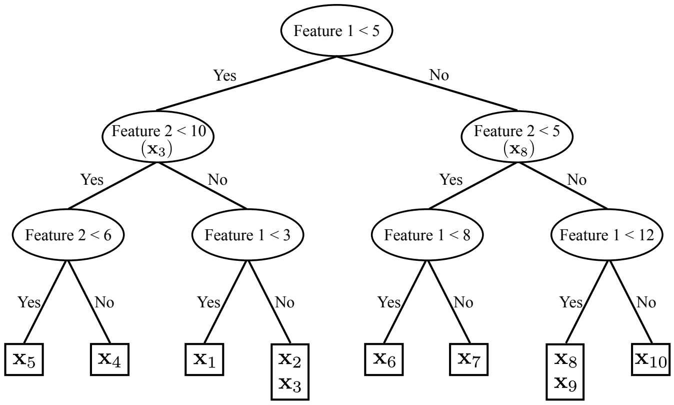
The main idea is to cleverly select a subset of data that still achieves an accurate classification, to reduce the required computation of distances. Representative strategies include:

- **Incremental Deletion.** Start with the entire dataset $\{(\mathbf{x}_\mathrm{i}, \mathrm{y}_\mathrm{i})\}_{\mathrm{i}=1}^{\mathrm{N}}$, and iteratively remove $(\mathbf{x}_\mathrm{i}, \mathrm{y}_\mathrm{i})$ if it can be correctly classified using only the remaining data points.
- **Incremental Growth.** Start with nothing, and iteratively include $(\mathbf{x}_\mathrm{i}, \mathrm{y}_\mathrm{i})$ if it cannot be correctly classified with previously included data points.

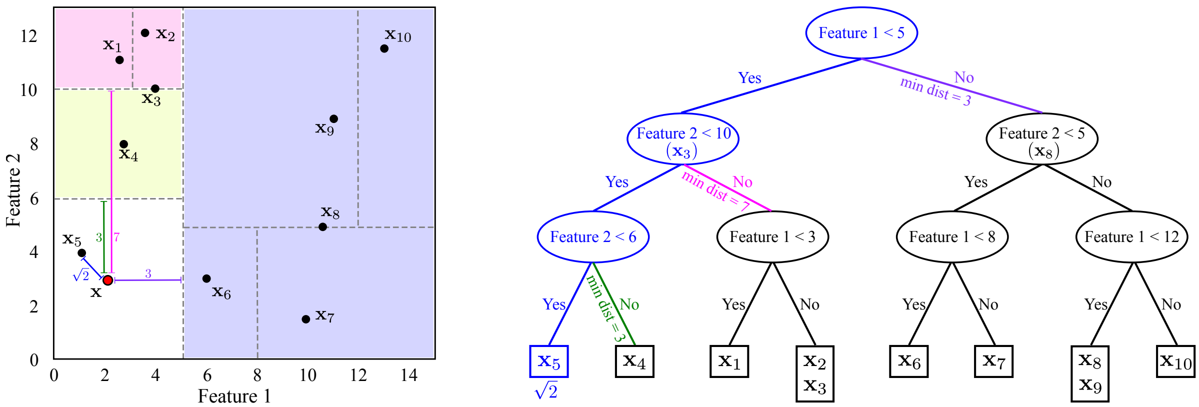
9.5.2 Efficient Search

Another speed-up alternative is to use smart methods to identify the closest neighbors without computing all distances. One example of this approach are *k-dimensional trees*, or *k-d* trees. The main idea is to partition the feature space into several regions, and search for the nearest neighbors only in the promising regions – or rather, skip the unpromising ones. More precisely, *k-d* trees iteratively partition the feature space according to the median value of the feature with highest variance in the training dataset. Then we construct a search tree according to this partition, where each split in the feature space corresponds to a node in the tree. For example, the partition and tree of the following dataset $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10} \in \mathbb{R}^2$ would look like:





Given a new observation $\mathbf{x}$, we use *branch-and-bound* search. That is, we traverse the different regions of the feature space according to the tree, keeping track of the minimum possible distance with *alternative regions* defined by the unselected nodes of each split. Once a leaf is reached, we compute the distances between $\mathbf{x}$ and all the *candidate* data points in the region/leaf. If the minimum possible distance from an alternative region is larger than the distance to a candidate nearest neighbor, then that region/node and its descendants do not need to be further searched/visited, because any point in such region will be farther from $\mathbf{x}$ than the candidate. In our example, we traverse the tree to reach the leaf of candidate $\mathbf{x}_5$, at which point we compute the distance $\|\mathbf{x} - \mathbf{x}_5\| = \sqrt{2}$. Since $\sqrt{2}$ is smaller than 3 and 7 (the minimum possible distances to the alternative regions, depicted in purple, pink and green), we do not need to traverse those alternative nodes, and we know that we have found the nearest neighbor.



In the worst-case scenario, this approach will compute $\mathcal{O}(N)$ distances, but in expectation it will only require $\mathcal{O}(\log_2 N)$. In our example we only required computing the distance to one point and three alternative regions, instead of $N = 10$ points that the naive brute force approach would require.

9.6 Advantages and Limitations

Some advantages of K-nearest neighbors include:

- It is simple to implement.
- It adapts nicely to on-line learning.
- It is robust to noise when $K > 1$.
- It tends to work well in practice.

Some limitations of K-nearest neighbors include:

- It is sensitive to varying ranges of feature values.
- It is sensitive to irrelevant and correlated features (locally weighted regression aims to address this issue).
- Classification/prediction can be inefficient (methods like *k-d* trees aim to address this issue).
- It doesn't provide much insight into problem domain because there is no explicit model.