Nearest Neighbor Predictors

September 2, 2018

Perhaps the simplest machine learning prediction method, from a conceptual point of view, and perhaps also the most unusual, is the *nearest-neighbor* method, which can be used for either classification or regression.

In this method, we just remember the entire training set T. At test time, when given a new data point $\mathbf{x}$ in the data space X, the nearest-neighbor predictor returns the label (for classification) or value (for regression) y_n corresponding to the training data point $\mathbf{x}_n$ that is closest to $\mathbf{x}$ in a pre-specified metric.

Thus, there is no training: Just store all of

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \ ,$$

regardless of the nature of data points $\mathbf{x}_n$ or labels or values y_n, and define some distance in the data space X. For example,

$$\Delta(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 \ ,$$

the squared Euclidean distance. The classifier is then

$$h(\mathbf{x}) = y_{\nu(\mathbf{x})} \quad \text{where} \quad \nu(\mathbf{x}) = \arg \min_{n=1,\ldots,N} \Delta(\mathbf{x}, \mathbf{x}_n) \ .$$

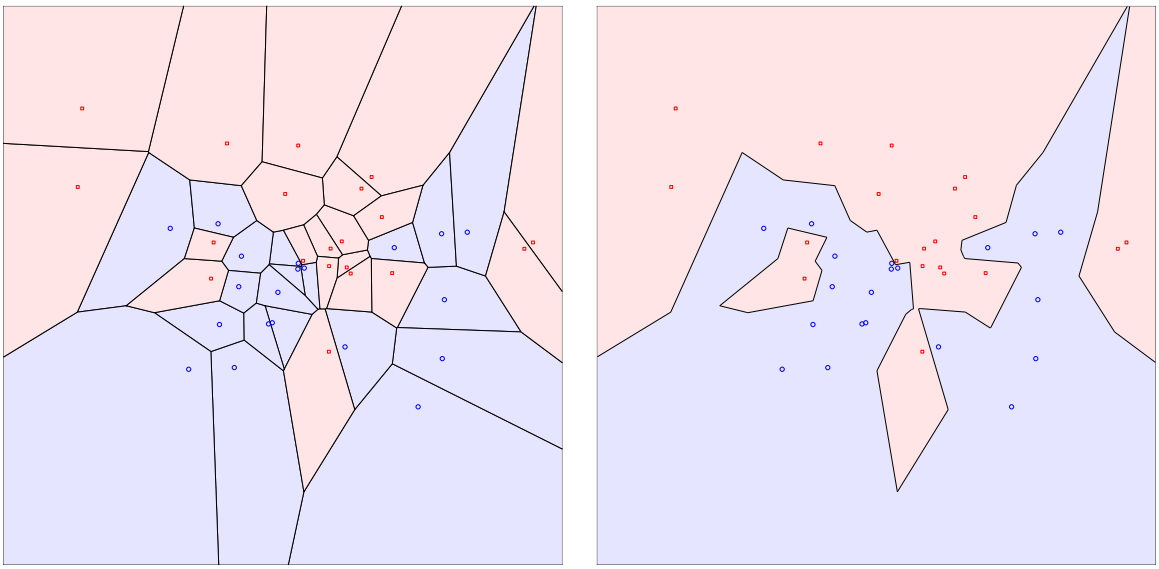
In words: To find the value $y = h(\mathbf{x})$ for a new data point $\mathbf{x}$, find the index $n = \nu(\mathbf{x})$ of the training data point $\mathbf{x}_n$ that is closest to $\mathbf{x}$ and return the value y_n for that training sample.

There is no distinction between classification and regression here: If Y is categorical, h returns a label, and is therefore a classifier. Otherwise, h returns a value (anything indeed: a single real number, a vector in any space,...), and is therefore a regressor. In addition, the number of classes is irrelevant for nearest-neighbor classification: Y can have any number of elements.

A first unusual property of the nearest-neighbor predictor is that training takes no time and inference (computing the value of $h(\mathbf{x})$ for a given $\mathbf{x}$) is slow: Computing the $\arg\min_n$, at least in a naive implementation, requires comparing $\mathbf{x}$ with all the training data points $\mathbf{x}_n$, so the time required for prediction is linear in N, the size of the training set. This is the opposite of what happens with most machine learning methods, which take a possibly long training time and are usually fast at inference time.

One can do better than linear-time, in practice, by building a data structure (typically based on R-trees or k-d trees or locality-sensitive hashing schemes) to store T. This construction requires time initially, but then finding the nearest neighbor to $\mathbf{x}$ is faster in practice. Theoretically, building an exact nearest-neighbor search structure is an NP-hard problem. However, practically efficient implementations exist at the cost of some approximation. For instance, rather than obtaining the closest sample, some of these implementations guarantee that a sample within ρ of closest is returned, for a given $\rho > 1$. This means that if the truly nearest sample to $\mathbf{x}$ is $\mathbf{x}_{\nu^*(\mathbf{x})}$ but h returns the value associated to $\mathbf{x}_{\nu(\mathbf{x})}$, then

$$\Delta(\mathbf{x}, \mathbf{x}_{\nu(\mathbf{x})}) < \rho \Delta(\mathbf{x}, \mathbf{x}_{\nu^*(\mathbf{x})}) \ .$$



(a) (b)

Figure 1: (a) The black lines are the edges of the Voronoi diagram of the sample points (blue circles and red squares), considered with no regard to their label values. Each cell (polygon) of the Voronoi diagram is subsequently colored in the color corresponding to the label of the training sample in it. The red and blue regions are the two subsets of the partition of data space X into sets corresponding to the two classes. Any new data point $\mathbf{x}$ in the red region is classified as a red square. Any other data point is classified as a blue circle. (b) The black lines trace the boundary between the two sets of the partition.

In words, the distance between $\mathbf{x}$ and $\mathbf{x}_{\nu(\mathbf{x})}$ is never more than ρ times the distance between $\mathbf{x}$ and the nearest training data point. One can pick any ρ strictly greater than one: the closer ρ is to one, the more it takes to construct the data structure, and the more storage space it takes. With these data structures, if one is willing to pay the ρ approximation, the typical situation is restored: Building the data structure can be considered as part of "training time," which is now long, and testing the predictor is comparatively fast.

Incidentally, small approximations are often entirely acceptable in this type of problems. This is because the particular notion of distance chosen is often a crude way to capture "dissimilarity" (what is so special about the Euclidean distance anyway?) and a small discrepancy when looking for the nearest point to $\mathbf{x}$ is often negligible when compared to the looseness in the choice of distance.

In few dimensions, perhaps $d = 2, 3$, one can build an exact data structure called the Voronoi diagram: Partition space into convex regions that are either polygons (in 2D) or polyhedra (in 3D), one region V_n for each training sample $(\mathbf{x}_n, y_n)$. The defining property of V_n is that any point $\mathbf{x} \in V_n$ is closer to $\mathbf{x}_n$ than it is to any other training data point. Figure 1(a) illustrates this concept for a subset of the data in a set introduced in a previous note.[1] If we only draw the edges between Voronoi regions that belong to data points with different labels, we obtain the partition that the nearest-neighbor classifier induces in the data space X, as shown by the black lines in Figure 1(b).

The Voronoi diagram is a purely conceptual construction for nearest-neighbor predictors: While this

[1] A subset of the data is used to make the figure more legible. There would be no problem in computing the Voronoi diagram for the whole set.

construction would lead to an efficient classifier for new data points $\mathbf{x}$, it is rare that data space is two- or three-dimensional in typical classification or regression problems. While the Voronoi diagram is a well-defined concept in more dimensions, there is no known efficient algorithm to construct it. Instead, the approximate algorithms mentioned earlier are used for classification when both d (the dimensionality of X) and N (the number of training samples) are very large. When d is large but N is not, the nearest neighbor of $\mathbf{x}$ can be found efficiently by a linear scan of all the elements in the training set.

For regression, each Voronoi region has in general a different value y_n, so the hypothesis space $\mathcal{H}$ of all functions that the nearest-neighbor regressor can represent is the set of all real-valued functions that are piecewise constant over Voronoi-like regions.

Overfitting and k-Nearest Neighbors

It should be clear from both concept and Figure 1 that nearest-neighbor predictors can represent functions (for regression) or class boundaries (for classification) of arbitrary complexity, as long as the training set T is large enough. In light of our previous discussions, it should also be clear that this high degree of expressiveness of $\mathcal{H}$ is not necessarily a good thing, as it can lead to overfitting: For instance, for binary classification, the exact boundary between the two classes will follow the vagaries of the exact placement of data points that are near other data points of the opposite class. Even the simple example in Figure 1(b) should convince you that a slightly different choice of training data would lead to quite different boundaries.

More explicitly, Figure 2 shows the boundaries defined by the nearest-neighbor classifiers for (a) a training set T with 200 points and (b-f) five random samples of 40 out of the 200 points in T. There is clearly quite a bit of *variance* in the results, and rather than drawing a boundary that somehow captures the bulk of the point distributions, each nearest-neighbor classifier follows every twist and turn of the complicated space between blue circles and red squares. This is an example of overfitting.

The notion of *k-nearest-neighbors* is introduced to control overfitting to some extent, where k is a positive integer. Rather that returning the value y_n of the training sample whose data point $\mathbf{x}_n$ is nearest to the new data point $\mathbf{x}$, the k-nearest-neighbors *regressor* returns the *average* (or sometimes the median) of the values for the k data points that are nearest to $\mathbf{x}$. The k-nearest-neighbors *classifier*, on the other hand, returns the *majority* of the labels for the k data points that are nearest to $\mathbf{x}$. For classification, it is customary to set k to an odd value, so that the concept of 'majority' requires no tie-breaking rule.

As an example, Figure 3 draws approximate decision regions for the same data sets as in Figure 2, but using a 9-nearest-neighbors classifier rather than a 1-nearest-neighbor classifier. The variance in the boundaries between regions is now smaller. In some sense, the 9-nearest-neighbors classifier captures more of the "gist" of the two point distributions, and less of the fine detail along the boundary between them. The 9-nearest-neighbors classifier overfits less than the 1-nearest-neighbors classifier.

Figure 4 (b) illustrates k-nearest-neighbors regression in one dimension, $d = 1$ for the dataset in panel (a) of the Figure. The line for $k = 1$ (orange) overfits dramatically, as it interpolates each data point. The line for $k = 100$ does quite a bit of smoothing, and reduces overfitting.

Points for Further Exploration

Some of the following questions will be explored later in the course. Others will be covered in some of the homework assignments. The rest are left for you to muse about.

- How to choose the metric Δ for a particular regression or classification task? For instance, if some features (entries of $\mathbf{x}$) are irrelevant but vary widely, they may limit the usefulness of a metric that includes them.

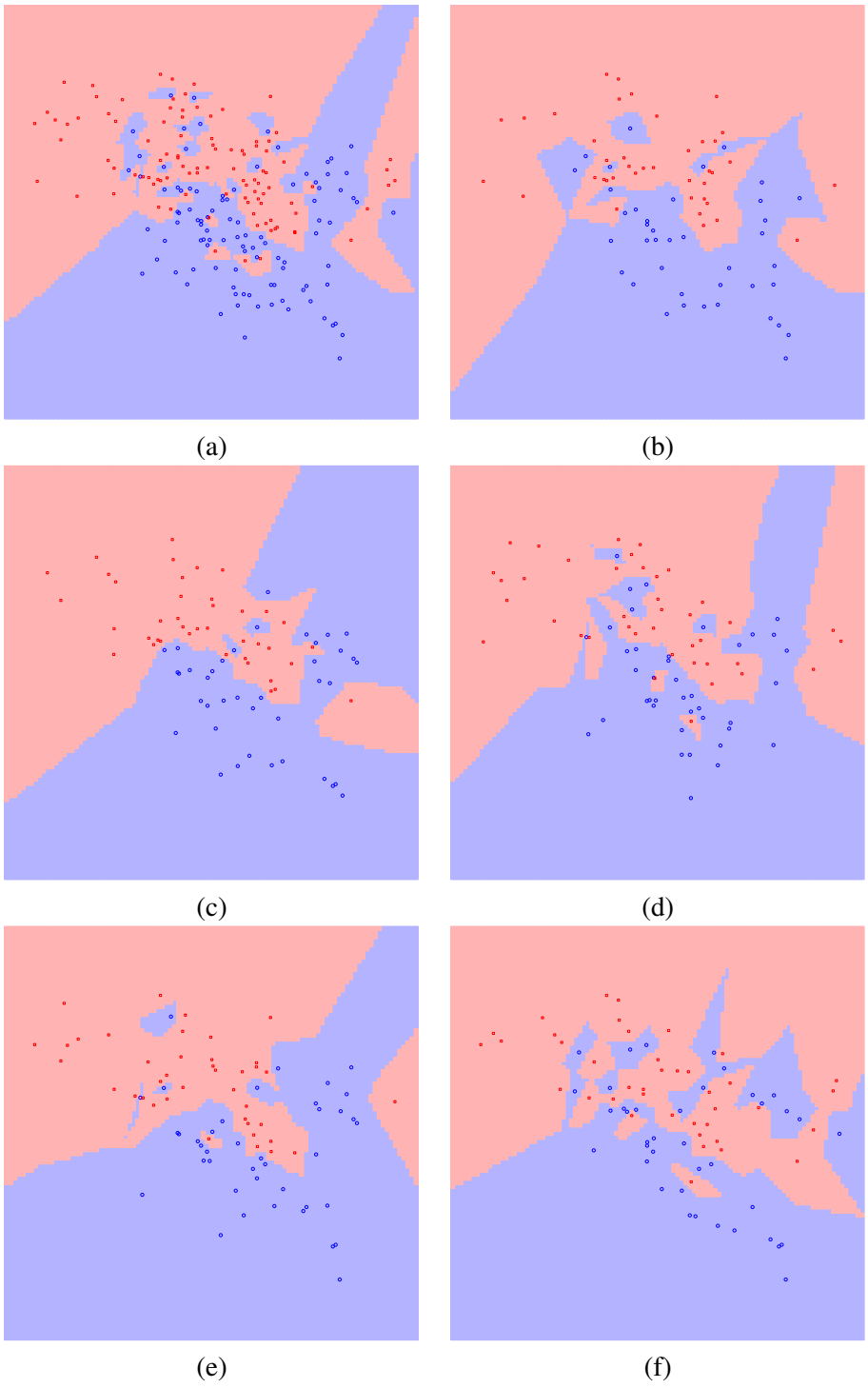


Figure 2: (a) The decision boundary of the nearest-neighbor classifier for a training set T with 200 points, 100 points per class. (b-f) The decision boundary of the nearest-neighbor classifier for five sets of 40 points (20 per class) each, drawn randomly out of T. As a programming alternative (which works for any classifier when $d = 2$), these regions were approximated by creating a dense grid of points on the plane, classifying each point on the grid, and placing a small square of the proper color at that grid point.

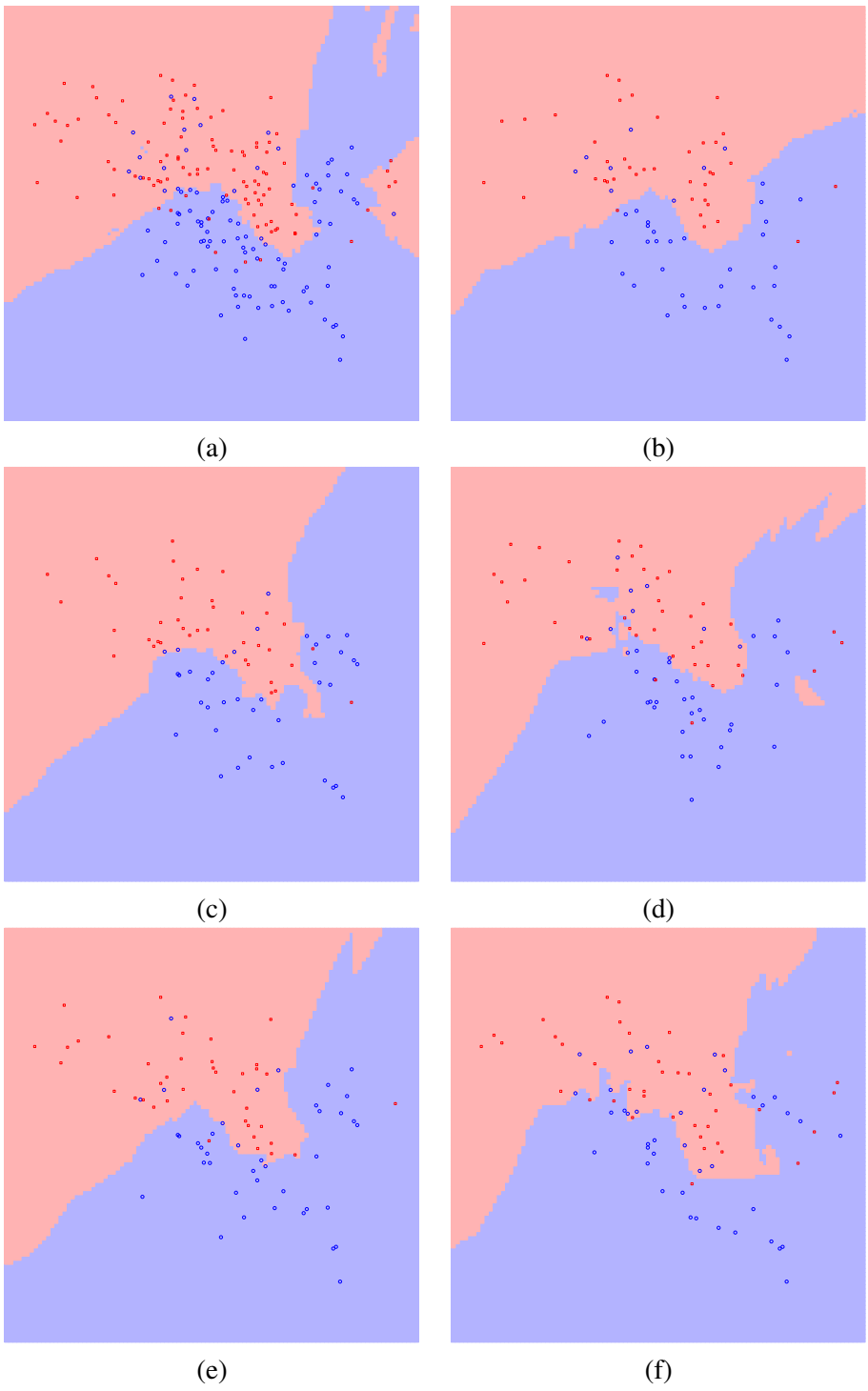
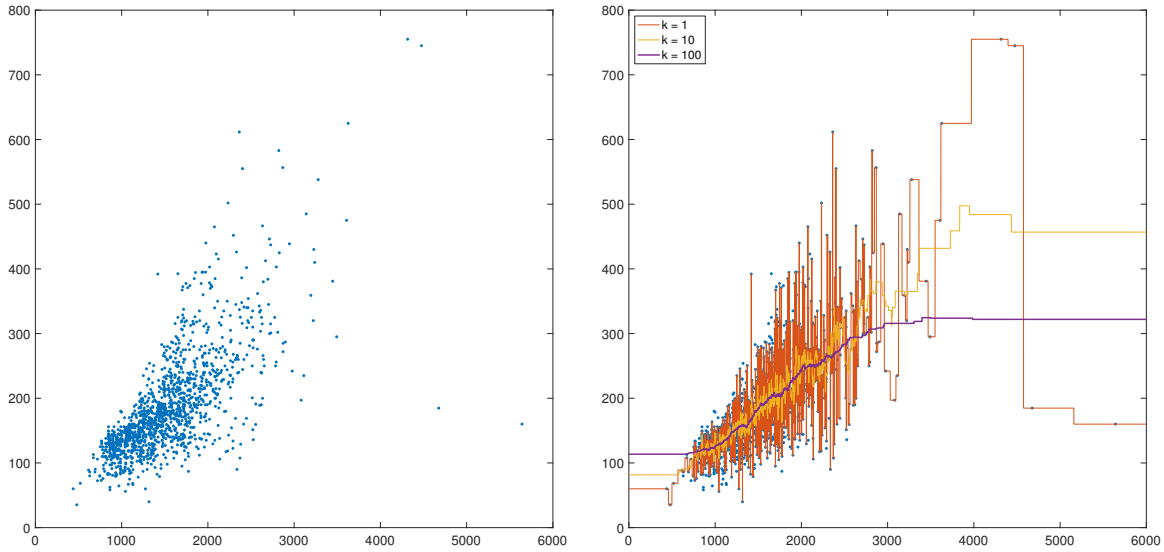


Figure 3: (a-f) The approximate decision regions of the 9-nearest-neighbor classifier for the same training sets as in Figure 2.



(a) (b)

Figure 4: (a) The horizontal axis denotes the gross living area in square feet of a set of 1379 homes in Ames, Iowa. The vertical axis is the sale price in thousands of dollars [1]. (b) An example of k-nearest-neighbor regression on the data in (a). The three lines are the k-nearest-neighbor regression plots for $k = 1, 10, 100$.

- Will Δ be informative in many dimensions, given our discussion on distances when d is high? This point is related to the previous one.
- How to choose k?
- Is it fruitful to think of *weighted* averaging in a k-nearest-neighbors regressor? Weights might help modulate the trade-off between accuracy (small k) and generalization (large k).
- Can a k-nearest-neighbors predictor be viewed as the solution of a loss minimization problem? What is the loss in that case?
- What applications does k-nearest-neighbors prediction suit best? Training is easy, and can be done incrementally (just add new data to T), but inference is potentially expensive.

References

[1] D. De Cock. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.