CHAPTER

# 3 Nearest Neighbor Methods

A very simple class of learners are nearest neighbor models. In essence, nearest neighbor methods reflect the most basic concept of prediction: if we wish to predict the target value associated with a feature measurement $x$, we should mainly use the data located near $x$ to estimate it. "Neighbor-based" methods store a set of training examples, and make their predictions by finding the closest examples to $x$. This is an example of "exemplar-based" modeling, in which specific data examples are stored and used to define the model.

## 3.1 Nearest neighbor prediction

Recall that we are given a set of training examples, which we denote as $D = \{x^{(i)}, y^{(i)}\}$. Then, the *nearest neighbor* (or 1-nearest neighbor) predictor $f(x)$ takes the form

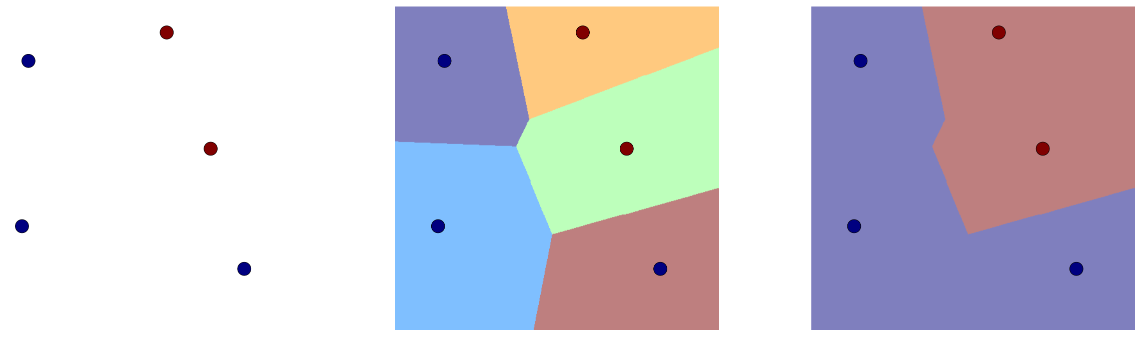
$$f(x\,;\,D) = y^{(i^*)} \qquad \text{where} \qquad i^* = \arg\min_i d(x, x^{(i)}),$$

where $d(x, x')$ is a *distance* or *dissimilarity* function that measures how far the vectors $x$ is from $x'$. The $\arg\min$ operator then finds the data point $i^*$ whose feature vector $x^{(i^*)}$ is closest to the test point $x$, and our predictor returns that data point's value $y^{(i^*)}$. This procedure trivially works for both classification and regression, since we are returning some data point's target value.

In practice, most implementations of nearest neighbor methods use the standard (squared) Euclidean distance,

$$d(x, x') = \|x - x'\|^2 = \sum_j (x_j - x'_j)^2.$$

Note that, since we are only interested in *which* data point is closest, we do not need to bother taking the square root; the data point $i^*$ that has the smallest distance will also have the smallest squared distance. However, a major strength of nearest-neighbor methods is that they can easily use arbitrary, potentially complex distance functions as well. For example, if our data points correspond to text documents, we may be able to define a distance function (or, conversely, a similarity function) that measures how close two documents are, and use this to select our nearest neighbor. For sequence data, such as in genetics, we could define similarity by aligning two sequences, or computing their "edit distance", the number of changes that need to be made to transform one into the other. For simplicity and for easy visualization, here we confine our attention to using Euclidean distance.



(a) Training data (b) Voronoi regions (c) Decision boundary

Figure 3.1: (a) A small set of 5 training data for classification. Using a nearest-neighbor classifier, the learner always predicts the value of the closest data point. (b) We can view the data points as partitioning the space into polygons, each point "claiming" all locations $x$ that are closer to it than any other point. (c) For a nearest-neighbor classifier, the decision boundary is then piecewise linear, composed of the subset of boundaries between data of different classes.

"Learning" such a predictor is essentially trivial – we simply store the training data in a database, and at test time, we find the data point $i^*$ which is closest to the test point $x$. However, this means that the memory requirements of the learner are $O(mn)$, i.e., increase linearly with both number of data ($m$) and their size (the number of features, $n$). Similarly, each prediction at test time must find the nearest data point, which is nominally also $O(mn)$, to search through $m$ points and calculate a sum of $n$ squared values for each. (Both storage and prediction time can sometimes be improved using computational tricks; see Section **??**.)

## Characterizing the learner

It is easy to see that the resulting function $f(x\,;\,D)$ is piecewise constant, with abrupt (discontinuous) changes in its predictions only when the identity of the nearest data point changes. Consider the classification example in Figure **??**, with two features $x_1$, $x_2$, and the target value $y$ indicated by color. We can think of each training example, pictured in Figure **??**(a), as "claiming" a region of the feature space that is closer to that example than any other. For Euclidean distance, the set of points that are equidistant between $x^{(i)}$ and $x^{(j)}$ is simply the line (or in higher dimensions, plane or hyperplane) that bisects the line segment connecting $x^{(i)}$ and $x^{(j)}$. Thus, the data points partition the feature space into a set of polygons, whose faces are located midway between two neighboring data; this partitioning is called the "Voronoi regions" of the data points (see Figure **??**(b)). Within each polygon, our prediction is the class of its associated data point. Recall that the decision boundary is the set of points at which our classifier (or decision function) $f$ changes value; here, it is easy to see that the decision boundary is piecewise linear, made up of the segments of the partitioning whose data have different values (Figure **??**(c)).

The same reasoning applies to regression in more than one dimension; each data point $x^{(i)}$ "claims" a region of feature space around it, that is closer to $x^{(i)}$ than any other point $x^{(j)}$, $j \neq i$ in the data set, and our prediction within this region is the constant value, $y^{(i)}$.

One point about nearest neighbor methods, and exemplar-based models more generally, is that their potential to represent complex functions increases with the number of exemplars they store. For example, a nearest neighbor model with only four data points must correspond to a relatively simple decision function; the function can only partition the feature space into four parts; but the more data are stored by the model, the more complex the predictor $f$ may be.

In general, the more data we are able to use, the better our model will fit. However, increasing the number of data will also dramatically increase the complexity of the predictor.

Claim: as $m \to \infty$, error rate is at most twice the Bayes error rate. Proof: TODO

Illustrate concept of overfitting using overlapping classes; clearly we do not believe that the small region should be predicted this way.

Notion of averaging over several nearby data, to "smooth out" such noise. K-nearest neighbor methods.

## 3.2 K-Nearest neighbor methods

Simple generalization of nearest neighbor, in which we select the $k$ nearest data points and combine them to make our prediction.

Notation?

Breaking ties

Increasing $k$ makes the effective decision boundary more smooth,

This statement is imprecise, of course – the actual decision boundary remains piecewise linear, since it corresponds to locations at which the $k$ nearest points change identity (which must correspond to locations where the $k$th nearest point becomes equidistant to the $k + 1$st). Since this involves more potential pairs of points, the decision boundary may actually have more segments. But, speaking imprecisely, since the function depends on the average of several data points, there is less emphasis on individual points (which may be noisy), and we can see that the function looks less complex and "noisy" as $k$ increases; see Figure 3.2.

Training and test error as a function of $k$: $k = 1$ means the training error will be zero (the nearest data point to $x^{(i)}$ is itself, which correctly outputs $y^{(i)}$). However, we are likely to get some test data wrong TODO As $k$ increases, we begin to get some of these data wrong (since their class does not match that of the other data in their neighborhood), and our training error rate increases. At the other extreme, when $k = m$, the number of data, our prediction function is simply a constant – we output the majority class value in our training dataset. Our training error is then quite large – in a balanced, binary classifier it will be nearly $50\%$ – but it is likely to be quite similar to our test error rate, assuming we have enough training data to accurately estimate the relative fraction of each class.

Match the basic pattern of overfitting –

Selecting a value of $k$? Can't be done using training data alone (it would always pick $k = 0$!).

Validation data

Cross-validation?

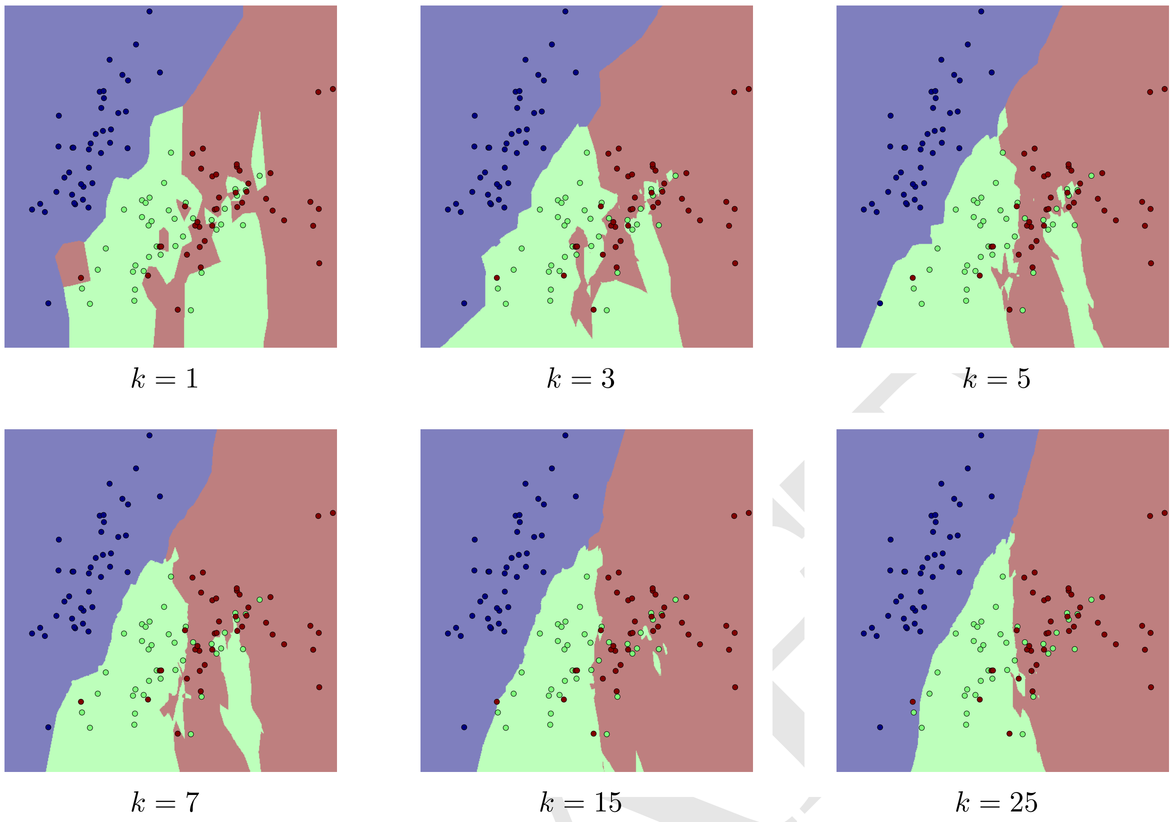


Figure 3.2: A $k$-nearest neighbor classifier on the Iris data, for varying settings of $k$. At $k = 1$, each training point claims a region and predicts its target value, making many small "islands" of different predictions. As $k$ increases, each location looks at a larger area to decide its prediction, so that a single red data point in a region dominated by blue, for example, will start to predict blue instead. By $k = 25$, our predictions are holistically simpler and more regional, predicting blue in the upper left, green at the bottom, and red to the right.

## 3.3 Weighted neighborhood methods

Another variation is to involve several nearby data points, but not to treat these data points equally – instead, we pay more attention to data that are close, compared to data further away.

Weighted average (or weighted vote, for classification)

Define weights using relative similarity, such as the Gaussian (or "Radial basis") similarity

$$w \propto \exp(-\|x - x'\|^2/C) \qquad\qquad \sum w^{(j)} = 1$$

Similarity kernel; cite Nadarya-Watson, etc.

Because the weights vary smoothly, the resulting regression function will now be smooth, rather than locally constant

For classification, it can result in smooth decision boundaries

In fact, since the weights decay toward zero at increasing distance, we can simply use the weights to define the neighborhood. However, it may still be computationally useful to restrict our predictor to only use the $k$ nearest data points, if they can be found more quickly than in linear time (in $m$).

## 3.4 The “Curse of Dimensionality”

Unfortunately, nearest-neighbor based methods tend to perform poorly in high dimension. One reason for this is termed the *curse of dimensionality*: that as the dimension $n$ increases, a sample of fixed size $m$ tend to become equidistant from one another.

**fill in**

## 3.5 Computational considerations

Computation is $O(mn)$ for each prediction, so $O(mm'n)$ to predict at $m'$ test points.

When $m$ is sufficiently large, spatial data structures can be used to speed up performance

Example for $n = 1$ – an efficient way to find the $k$ nearest neighbors is to first sort the data by their feature values, then find the insertion points of the test data.

A multidimensional version of this data structure is the $k$-d tree (or $k$-dimensional tree), which organizes the data to try to ensure that nearby data (in the feature space) are grouped and stored together.

**add more detail**

Approximate methods can also be applied; for example, locality sensitive hashing **?**.

Also mention data set reduction techniques and coresets.

## 3.6 Connections to density estimation