Lecture 1

k-Nearest Neighbor Algorithms for Classification and Prediction

1 k-Nearest Neighbor Classification

The idea behind the k-Nearest Neighbor algorithm is to build a classification method using no assumptions about the form of the function, $y = f(x_1, x_2, ...x_p)$ that relates the dependent (or response) variable, y, to the independent (or predictor) variables $x_1, x_2, ...x_p$. The only assumption we make is that it is a "smooth" function. This is a non-parametric method because it does not involve estimation of parameters in an assumed function form such as the linear form that we encountered in linear regression.

We have training data in which each observation has a y value which is just the class to which the observation belongs. For example, if we have two classes y is a binary variable. The idea in k-Nearest Neighbor methods is to dynamically identify k observations in the training data set that are similar to a new observation , say $(u_1, u_2, ...u_p)$, that we wish to classify and to use these observations to classify the observation into a class, $\widehat{v}$.If we knew the function f, we would simply compute $\widehat{v} = f(u_1, u_2, ...u_p)$. If all we are prepared to assume is that f is a smooth function, a reasonable idea is to look for observations in our training data that are near it (in terms of the independent variables) and then to compute $\widehat{v}$ from the values of y for these observations. This is similar in spirit to the interpolation in a table of values that we are accustomed to doing in using a table of the Normal distribution. When we talk about neighbors we are implying that there is a distance or dissimilarity measure that we can compute between observations based on the independent variables. For the moment we will confine ourselves to the most popular measure of distance: Euclidean distance. The Euclidean distance between the points $(x_1, x_2, ...x_p)$ and $(u_1, u_2, ...u_p)$ is $\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}$. We will examine other ways to define distance between points in the space of predictor variables when we discuss clustering methods.

The simplest case is $k = 1$ where we find the observation that is closest (the nearest neighbor) and set $\widehat{v} = y$ where y is the class of the nearest neighbor. It is a remarkable fact that this simple, intuitive idea of using a single nearest neighbor to classify observations can be very powerful when we have a large number of observations in our training set. It is possible to prove that the misclassification error of the 1-NN scheme has a misclassification probability that is no worse than twice that of the situation where we know the precise probability density functions for each class. In other words if we have a large amount of data and used an arbitrarily sophisticated classification rule, we would be able to reduce the misclassification error at best to half that of the simple 1-NN rule.

For k-NN we extend the idea of 1-NN as follows. Find the nearest k neighbors and then use a majority decision rule to classify a new observation.The advantage is that higher values of k provide smoothing that reduces the risk of overfitting due to noise in the training data. In typical applications k is in units or tens rather than in hundreds or thousands. Notice that if $k = n$, the number of observations in the training data set, we are merely predicting the class that has the majority in the training data for all observations irrespective of the values of $(u_1, u_2, ...u_p)$. This is clearly a case of oversmoothing unless there is no information at all in the independent variables about the dependent variable.

Example 1

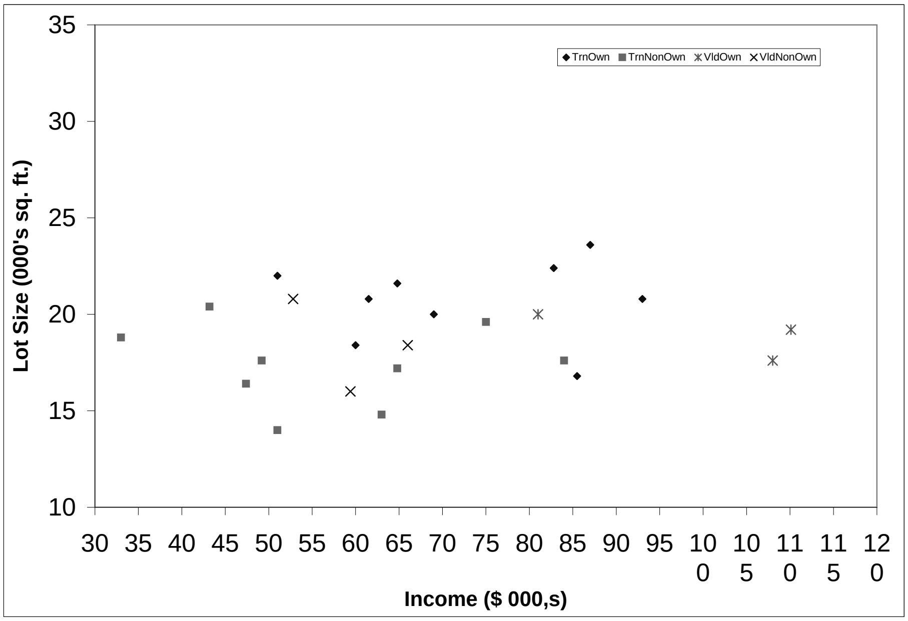
A riding-mower manufacturer would like to find a way of classifying families in a city into those that are likely to purchase a riding mower and those who are not likely to buy one. A pilot random sample of 12 owners and 12 non-owners in the city is undertaken. The data are shown in Table I and

Figure 1 below:

Table 1

Observation	Income ($000's)	Lot Size (000's sq. ft.)	Owners=1, Non-owners=2
1	60	18.4	1
2	85.5	16.8	1
3	64.8	21.6	1
4	61.5	20.8	1
5	87	23.6	1
6	110.1	19.2	1
7	108	17.6	1
8	82.8	22.4	1
9	69	20	1
10	93	20.8	1
11	51	22	1
12	81	20	1
13	75	19.6	2
14	52.8	20.8	2
15	64.8	17.2	2
16	43.2	20.4	2
17	84	17.6	2
18	49.2	17.6	2
19	59.4	16	2
20	66	18.4	2
21	47.4	16.4	2
22	33	18.8	2
23	51	14	2
24	63	14.8	2

How do we choose k? In data mining we use the training data to classify the cases in the validation data to compute error rates for various choices of k. For our example we have randomly divided the data into a training set with 18 cases and a validation set of 6 cases. Of course, in a real data mining situation we would have sets of much larger sizes. The validation set consists of observations 6, 7, 12, 14, 19, 20 of Table 1. The remaining 18 observations constitute the training data. Figure 1 displays the observations in both training and validation



data sets. Notice that if we choose k=1 we will classify in a way that is very sensitive to the local characteristics of our data. On the other hand if we choose a large value of k we average over a large number of data points and average out the variability due to the noise associated with individual data points. If we choose k=18 we would simply predict the most frequent class in the data set in all cases. This is a very stable prediction but it completely ignores the information in the independent variables.

Table 2 shows the misclassification error rate for observations in the validation data for different choices of k.

Table 2

k	1	3	5	7	9	11	13	18
Misclassification Error %	33	33	33	33	33	17	17	50

We would choose k=11 (or possibly 13) in this case. This choice optimally trades off the variability associated with a low value of k against the oversmoothing associated with a high value of k. It is worth remarking that a useful way to think of k is through the concept of "effective number of parameters". The effective number of parameters corresponding to k is n/k where n is the number of observations in the training data set. Thus a choice of k=11 has an effective number of parameters of about 2 and is roughly similar in the extent of smoothing to a linear regression fit with two coefficients.

2 k-Nearest Neighbor Prediction

The idea of k-NN can be readily extended to predicting a continuous value (as is our aim with multiple linear regression models), by simply predicting the average value of the dependent variable for the k nearest neighbors. Often this average is a weighted average with the weight decreasing with increasing distance from the point at which the prediction is required.

3 Shortcomings of k-NN algorithms

There are two difficulties with the practical exploitation of the power of the k-NN approach. First, while there is no time required to estimate parameters from the training data (as would be the case for parametric models such as regression) the time to find the nearest neighbors in a large training set can be prohibitive. A number of ideas have been implemented to overcome this difficulty. The main ideas are:

1. Reduce the time taken to compute distances by working in a reduced dimension using dimension reduction techniques such as principal components;

2. Use sophisticated data structures such as search trees to speed up identification of the nearest neighbor. This approach often settles for an "almost nearest" neighbor to improve speed.

3. Edit the training data to remove redundant or "almost redundant" points in the training set to speed up the search for the nearest neighbor. an example is to remove observations in the training data set that have no effect on the classification because they are surrounded by observations that all belong to the same class.

Second, the number of observations required in the training data set to qualify as large increases exponentially with the number of dimensions p. This is because the expected distance to the nearest neighbor goes up dramatically with p unless the size of the training data set increases exponentially with p. An illustration of this phenomenon, known as "the curse of dimensionality", is the fact that if the independent variables in the training data are distributed uniformly in a hypercube of dimension p, the probability that a point is within a distance of 0.5 units from the center is

$$\frac{\pi^{p/2}}{2^{p-1}p\Gamma(p/2)}$$

The table below is designed to show how rapidly this drops to near zero for different combinations of p and n, the size of the training data set.

				p				
n	2	3	4	5	10	20	30	40
10,000	7854	5236	3084	1645	25	0.0002	2×10^{-10}	3×10^{-17}
100,000	78540	52360	30843	16449	249	0.0025	2×10^{-9}	3×10^{-16}
1,000,000	785398	523600	308425	164493	2490	0.0246	2×10^{-8}	3×10^{-15}
10,000,000	7853982	523600	3084251	1644934	24904	0.2461	2×10^{-7}	3×10^{-14}

The curse of dimensionality is a fundamental issue pertinent to all classification, prediction and clustering techniques. This is why we often seek to reduce the dimensionality of the space of predictor variables through methods such as selecting subsets of the predictor variables for our model or by combining them using methods such as principal components, singular value decomposition and factor analysis. In the artificial intelligence literature dimension reduction is often referred to as factor selection.