MLCC: Machine Learning Crash Course **Spring 2014**

Lecture 2- Local Methods and Bias Variance Trade-Off

Lecturer: F.Odone–L. Rosasco

First, we describe a simple yet efficient class of algorithms, the so called memory based learning algorithms, based on the principle that nearby input points should have the similar/same output. Then, we discuss the problem of tuning the learning parameters and present the concept of Bias-Variance trade-off.

2.1 Nearest Neighbor

Consider a training set

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}.$$

Given an input $\bar{x}$, let

$$i' = \arg \min_{i=1,\ldots,n} \|\bar{x} - x_i\|^2$$

and define the nearest neighbor (NN) estimator as

$$\hat{f}(\bar{x}) = y_{i'}.$$

Every new input point is assigned the same output as its nearest input in the training set. We add few comments. First, while in the above definition we simply considered the Euclidean norm, the method can be promptly generalized to consider other measure of similarity among inputs. For example if the input are binary strings, i.e. $X = \{0,1\}^D$, one could consider the Hamming distance

$$d_H(x, \bar{x}) = \frac{1}{D} \sum_{j=1}^{D} \mathbf{1}_{[x^j \neq \bar{x}^j]}$$

where x^j is the j-th component of a string $x \in X$.
Second, the complexity of the algorithm for predicting any new point is $O(nD)$– recall that the complexity of multiplying two D-dimensional vectors is $O(D)$. Finally, we note that NN can be fairly sensitive to noise.

2.2 K-Nearest Neighbor

Let $d_{\bar{x}} = (\|\bar{x} - x_i\|^2)_{i=1}^n$ be the array of distances of a new point $\bar{x}$ to the input points in the training set. Let $s_{\bar{x}}$ be the above array sorted in increasing order and $I_{\bar{x}}$ the corresponding vector of indices. Let $K_{\bar{x}} = \{I_{\bar{x}}^1, \ldots, I_{\bar{x}}^K\}$ be the array of the first K entries of $I_{\bar{x}}$. The K-nearest neighbor estimator (KNN) is defined as,

$$\hat{f}(\bar{x}) = \sum_{i' \in K_{\bar{x}}} y_{i'}.$$

In classification KNN can be seen as a *voting scheme* among the K nearest neighbors and K is taken to be odd to avoid ties. The parameter K controls the stability of the KNN estimate: when K is small the algorithm is sensitive to the data (and simply reduces to NN for $K = 1$). When K increases the estimator becomes more stable. In classification, it eventually simply becomes the ratio of the number of elements for each class. The question of how to best choose K will be the subject of a future discussion.

2.3 Parzen Windows

In KNN each of the K neighbors has equal weights in determining the output of a new point. A more general approach is to consider estimators of the form,

$$\hat{f}(\bar{x}) = \frac{\sum_{i=1}^{n} y_i k(\bar{x}, x_i)}{\sum_{i=1}^{n} k(\bar{x}, x_i)},$$

where $k : X \times X \to [0, 1]$ is a suitable function, which can be seen as a similarity measure on the input points. The function k defines a *window* around each point and is sometimes called a Parzen window. A classification rule is obtained considering the sign of $\hat{f}(\bar{x})$.

Many examples of k depend on the distance $\|x - x'\|$, $x, x' \in X$. For example we can consider

$$k(x', x) = \mathbf{1}_{\|x-x'\| \leq r}.$$

This choice induce a Parzen window analogous to KNN, here the parameter K is replaced by the *radius* r. More generally it is interesting to have a decaying weight for point which are further away. For example considering

$$k(x', x) = (1 - \|x - x'\|)_+ \mathbf{1}_{\|x-x'\| \leq r},$$

where $(a)_+ = a$, if $a > 0$ and $(a)_+ = 0$, otherwise. Another possibility is to consider fast decaying functions such as a Gaussian

$$k(x', x) = e^{-\|x-x'\|^2/2\sigma^2}.$$

or an exponential

$$k(x', x) = e^{-\|x-x'\|/\sqrt{2}\sigma}.$$

In all the above methods there is a parameter r or σ that controls the influence that each neighbor has on the prediction.

2.4 High Dimensions

The following simple reasoning highlights a phenomenon which is typical of dealing with high dimensional learning problems. Consider a unit cube in D dimensions, and a smaller cube of edge e. How shall we choose e to capture 1% of the volume of the larger cube? Clearly, we need $e = \sqrt[D]{.01}$. For example $e = .63$ for $D = 10$ and $e = .95$ for $D = 100$. The edge of the *small* cube is virtually the *same* length of that of the *large* cube. The above example illustrates how in high dimensions our intuition of neighbors and neighborhoods is challenged.

2.5 Tuning and Bias Variance Variance Decomposition

Here we study the question of how to choose the number of neighbors K in KNN: is there an optimal choice of K? Can it be computed in practice? Most of the ideas and results in the following generalize to a large class of learning algorithms beyond KNN. Indeed, many (most) learning algorithms depend on one or more parameter controlling their performances.

For the sake of simplicity we consider a regression model

$$y = f_*(x) + \delta,$$

where $\mathbf{E}\delta = 0$ and $\mathbf{E}\delta^2 = \sigma^2$. Moreover, we consider the least square loss function to measure errors, so that the performance of the KNN algorithm is given by the expected loss

$$\mathbf{E}_{x,y}(y - \hat{f}_K(x))^2 = \mathbf{E}_x \underbrace{\mathbf{E}_{y|x}(y - \hat{f}_K(x))^2}_{\varepsilon(K)}.$$

To get an insight on how to choose K, we analyze theoretically how this choice influences the expected loss. In fact, in the following we simplify the analysis considering the performance of KNN $\varepsilon(K)$ at a given point x.

First, note that

$$\varepsilon(K) = \sigma^2 + \mathbf{E}_{y|x}(f_*(x) - \hat{f}_K(x))^2,$$

where σ^2 can be seen as an irreducible error term. Second, to study the latter term we introduce the *expected* KNN algorithm,

$$\mathbf{E}_{y|x}\hat{f}_K(x) = \frac{1}{K}\sum_{\ell \in K_x} f_*(x_\ell).$$

We have

$$\mathbf{E}_{y|x}(f_*(x) - \hat{f}_K(x))^2 = \underbrace{(f_*(x) - \mathbf{E}_{y|x}\hat{f}_K(x))^2}_{Bias} + \underbrace{\mathbf{E}_{y|x}(\mathbf{E}_{y|x}\hat{f}_K(x) - \hat{f}_K(x))^2}_{Variance}$$

Finally, we have

$$\varepsilon(K) = \sigma^2 + (f_*(x) - \frac{1}{K}\sum_{\ell \in K_x} f_*(x_\ell))^2 + \frac{\sigma^2}{K}$$

2.6 The Bias Variance Trade-Off

We are ready to discuss the behavior of the (point-wise) expected loss of the KNN algorithm as a function of K. As it is clear from the above equation, the variance decreases with K. The bias is likely to increase with K, if the function f_* is suitably smooth. Indeed, for small K the few closest neighbors to x will have values close to $f_*(x)$, so their average will be close to $f_*(x)$. Whereas, as K increases neighbors will be further away and their average might move away from $f_*(x)$. A larger bias variance is preferred when data are few/noisy to achieve a better control of the variance, whereas the bias can be decreased as more data become available, hence reducing the variance. For any given training set, the best choice for K would be the one striking the optimal trade-off between bias and variance (that is the value minimizing their sum).

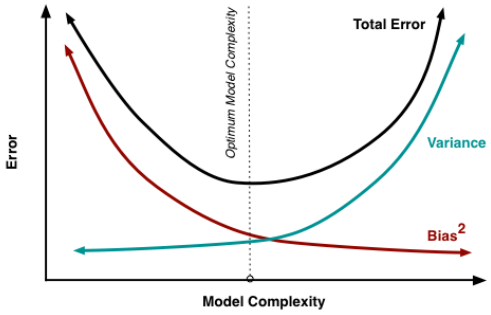

Figure 2.1. The Bias-Variance Tradeoff. In the KNN algorithm the parameter K controls the achieved (model) complexity.

2.7 Cross Validation

While instructive, the above analysis is not directly useful in practice since the data distribution, hence the expected loss, is not accessible. In practice, data driven procedures are used to find a proxy for the expected loss. The simplest such procedure is called hold-out cross validation. Part of the training S set is hold-out, to compute a (holdout) error to be used as a proxy of the expected error. An empirical bias variance trade-off is achieved choosing the value of K that achieves minimum hold-out error. When data are scarce the hold-out procedure, based on a simple "two ways split" of the training set, might be unstable. In this case, so called V-fold cross validation is preferred, which is based on multiple data splitting. More precisely, the data are divided in V (non overlapping) sets. Each set is hold-out and used to compute an hold-out error which is eventually averaged to obtained the final V-fold cross validation error. The extreme case where $V = n$ is called leave-one-out cross validation.