ISML-II: Machine Learning **Spring 2014**

Lecture 4- Local Methods

Lecturer: Lorenzo Rosasco *Scribe: Lorenzo Rosasco*

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.

4.1 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. To see this it is useful to visualize the decision boundary of the nearest neighbor algorithm.

4.2 K-Nearest Neighbor

Consider

$$d_{\bar{x}} = (\|\bar{x} - x_i\|^2)_{i=1}^n$$

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, and

$$K_{\bar{x}} = \{I_{\bar{x}}^1, \ldots, I_{\bar{x}}^K\}$$

be the array of the first K entries of $I_{\bar{x}}$. The K-nearest neighbor estimator (KNN) is defined as,

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

or $\hat{f}(\bar{x}) = \frac{1}{K} \sum_{i' \in K_{\bar{x}}} y_{i'}$. 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.

4.3 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 $\hat{f}(\bar{x})$.
In many examples the function k depends on the distance $\|x - x'\|$, $x, x' \in X$. For example,

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

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:

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

or

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

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

4.4 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.