Introduction to Machine Learning **Fall Semester, 2013**

Lecture 1: October 13

Lecturer: Mariano Schain *Scribe: ym*

1.1 K-Means

The k-means has an objective function F, where

$$F((\mu_1, \ldots, \mu_k), (S_1, \ldots, S_k)) = \sum_{i=1}^{k} \sum_{j \in S_i} \|x_j - \mu_i\|_2^2$$

In each iteration we have two actions:

Assign sets each point to its closest center, i.e.$C_j^t = \arg\min_i \|x_j - \mu_i\|^2$ and $S_i^t = \{x_j | C_j^t = i\}$.

Update minimizes F by re-computing the centers. I.e., $\mu_i = (1/|S_i|) \sum_{j \in S_i} x_j$.

The value of F (as a function of the centers and cluster assignments) decreases with the iteration (until it stops). Theoretically we are not guaranteed convergence because we might loop between configurations of identical F value. (See Figure 1)

More importantly, we might have a bad solution. (See the example of 3-means in Figures 2 and 3.)

How can we overcome the convergence problem? We can select a few random starting point and select the best (the one that has the lowest observed F.)

The dependency on the number of clusters is illustrated in Figure 4.

An example for using the k-means. We have a picture with 512×512 pixels, each 24 bits (i.e., each has 8 bits for each color). We like to do a compression to 4 bits per pixel. We can view the input as 2^{18} 3-dimensional vectors (the colors of each pixel). We run a 16-means algorithms on this input. When the algorithm ends we have 16 clusters, and each pixel belongs to a cluster. Now we give each pixel the name of the cluster, and for each cluster we keep its center. The total size in only $4 \cdot 2^{18} + 16 \cdot 24$ versus $24 \cdot 2^{18}$ before.

1.2 Nearest Neighbor

The input we have are points and their classification, i.e., (x, y). The goal is to compute a function $f(x)$ and hopefully $f(x) \approx y$. In order to make the problem sound, we need to select a loss function. For binary prediction an intuitive loss is the $0-1$-loss, where $L(a, b) = 0$ iff

$a = b$ and otherwise $L(a, b) = 1$. We can also have a quadratic loss $L_2(a, b) = (a - b)^2$. We set $L_f(x, y) = L(f(x), y)$, i.e., $a = f(x)$ and $b = y$.

To illustrate the influence of k, assume that the samples (x, y) are drawn from a joint distribution. Our goal is to select the hypothesis f that minimizes

$$E_{(x,y)}L_f(x, y) = E_x E_{y|x}(f(x) - y)^2$$

The optimal hypothesis is $\hat{f}$ such that $\hat{f}(x) = E[y|x]$.

we can not use this in practice, since we do not know the distribution of (x, y). We can view the k-NN as an approximation of $\hat{f}$.For binary classification (as in the lecture) we can set $f(x) = majority(x_{[1]}, \ldots, x_{[k]})$. For categorical classification we can use plurality (instead of majority), i.e., $f(x) = \arg\max_c\{x_{[j]}|y_{[j]} = c\}$. For continuous values we can set $f(x) = (1/k)\sum_{j=1}^{k} y_{[j]}$.

The difference from $\hat{f}$ is in two places: (1) we use points near x rather than x itself. (2) we use the empirical average based only on a few points (k) rather than the underlying average of the conditional distribution $(y|x)$.

The influence of the number of clusters k on the error is in Figure 5 (note that k decreases to the right):

For very large k (say $k = n$) the error is large, since we a grouping together very different examples, hurting the approximation (1) above.

For small k (say $k = 1$) we approximate the conditional average based a very small sample set, hurting approximation (2) above therby fitting also noise.

We can try to estimate the error of each k. We can do this using cross validation. We partition the training example (for example) to 90% train and 10% validation. We build the k-NN using the 90% and use the 10% validation to check its error. We can do many random splits and take their average. This method is called *cross validation.*