Introduction to Machine Learning

Recitation 1

Lecturer: Regev Schweiger *Scribe: Yishay Mansour*

1.1 K-Means

In the k-means algorithm, in each iteration we have two actions:

Assign: Sets each point to its closest center:

$$C_i^t = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j^{t-1}\|^2, S_j^t = \{i | C_i^t = j\}$$

Update: Minimizes F by re-computing the centers:

$$\boldsymbol{\mu}_j^t = (1/|S_j^t|) \sum_{i \in S_j^t} \mathbf{x}_i$$

The algorithm has an objective function F, where

$$F((\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k), (S_1, \ldots, S_k)) = \sum_{j=1}^{k} \sum_{i \in S_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|_2^2$$

The value of F (as a function of the centers and cluster assignments) decreases with each iteration, untilit stops (Figure 1.1). Theoretically we are not guaranteed convergence because we might loop between configurations of identical F value. More importantly, we might have a bad solution (see the example of 3-means in Figure 1.2), in the sense that it might be a *local* solution but not a *global* solution.

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

Another issue is *overfitting* - while increasing k decreases F on the data we see, future data may not behave well according to the k clusters we see. The dependency on the number of clusters is illustrated in Figure 1.3.

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 would 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.

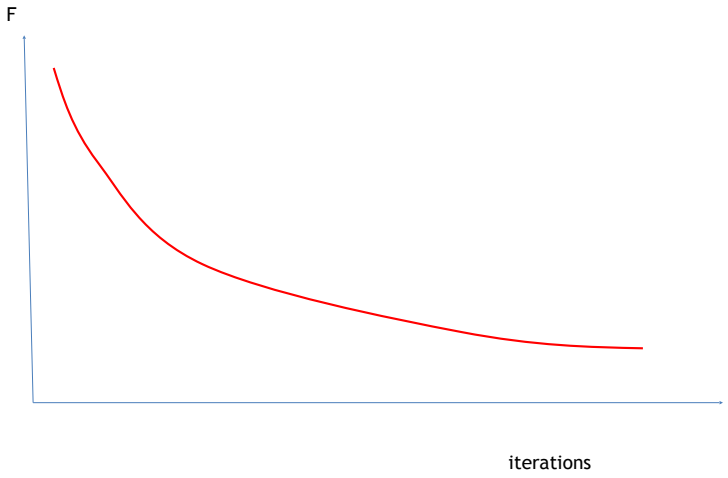

Figure 1.1: Objective function F as a function of the number of iterations

1.2 k-Nearest Neighbors

The input we have are points and their classification, i.e., $(\mathbf{x}, y)$. The goal is to compute a function $f(\mathbf{x})$ and hopefully $f(\mathbf{x}) \approx y$. For binary classification (as in the lecture) we can set $f(\mathbf{x}) = majority(\mathbf{x}_{[1]}, \ldots, \mathbf{x}_{[k]})$. For categorical classification we can use plurality (instead of majority), i.e., $f(x) = \arg\max_c \{\mathbf{x}_{[j]} | y_{[j]} = c\}$. For continuous values we can set $f(x) = (1/k) \sum_{j=1}^{k} y_{[j]}$.

1.3 Validation

Overfitting is a big problem. When we are learning the parameters of some classifier or a regression function, there is a danger that this classifier will not generalize from the data on which we've learned it, to new data we haven't yet seen.

For example, suppose we are given a set of points x_i, t_i, and we wish to fit a function $f(x) = t$ (e.g., Figure 1.4). One way to deal with overfitting is to divide our available data into *training set* and *test set*. We use the training data to learn our classifier/regression function, but evaluate its performance on the test data. We have as an input a set of examples S. We have an algorithm that given a sample T generates a hypothesis h_T (a suggested classifier or a regression function). We also have a loss function L, from which we can calculate the error of h_T's predictions on the testing data. For example, the algorithm fits the best polynomial curve of a given degree M, and the loss function is L^2. It is evident

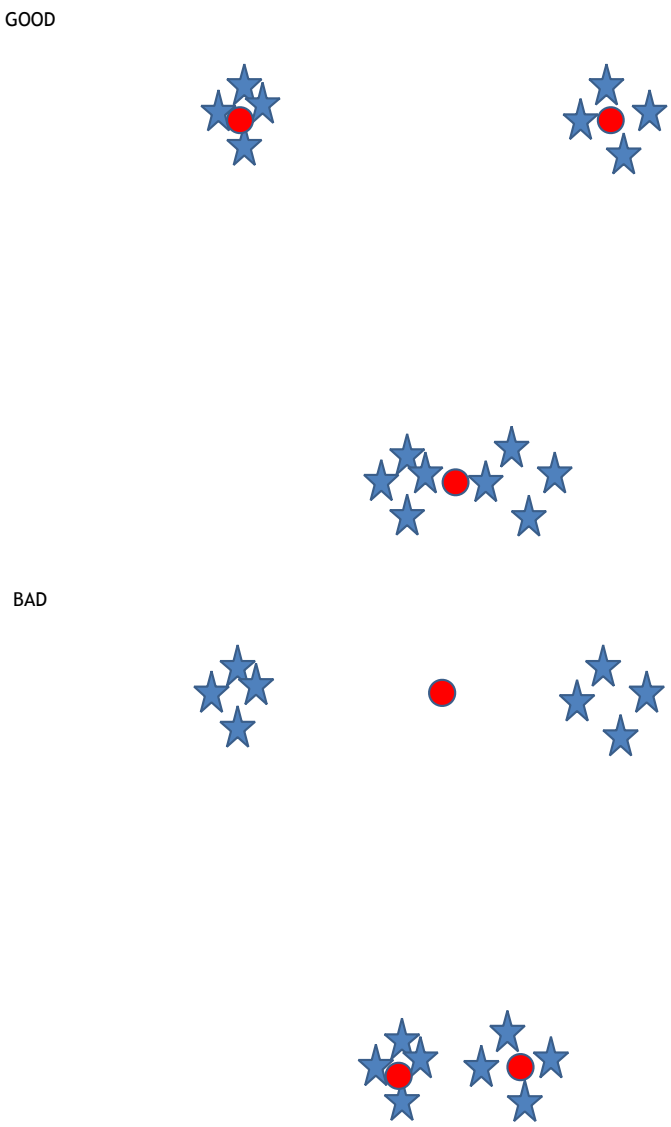

Figure 1.2: Bad and good solutions for 3 clusters

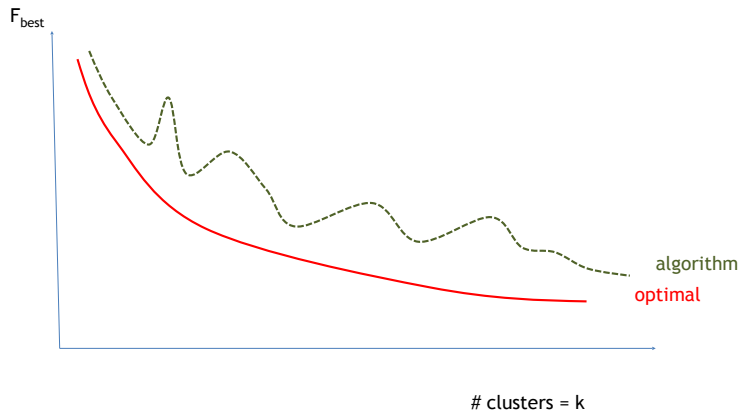

Figure 1.3: Dependency of F on the number of clusters, optimal vs. algorithm

that while the training error (i.e. the error calculated on the training set) decreases with M, the test error decreases and then increases, when overfitting becomes dominant (Figure 1.5).

Cross validation is a method to test the performance of your classifier when there is a limited amount of data available.

In the cross validation method, we will partition the sample *randomly* to k equal-size parts. Let $S_1, \ldots, S_k$ be the partition. We will run k iterations of our learning algorithm, where in iteration i we have as input $S - S_i$, and compute a hypothesis h_i. We test the hypothesis h_i on S_i and compute its observed error $error_i$. Our prediction of the error of our hypothesis would be the average of the observed errors, i.e., $\frac{1}{k}\sum_{i=1}^{k} error_i$.

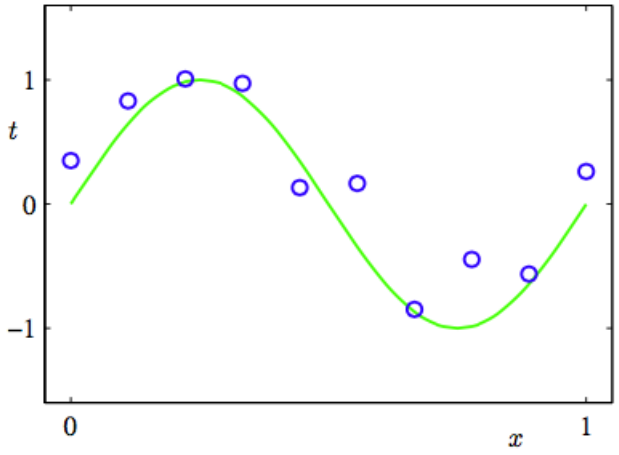

Figure 1.4: 10 pairs of (x_i, t_i), drawn from $t = sin(2\pi x)$ with random Gaussian noise. Adopted from *Christopher M. Bishop - Pattern Recognition and Machine Learning.*

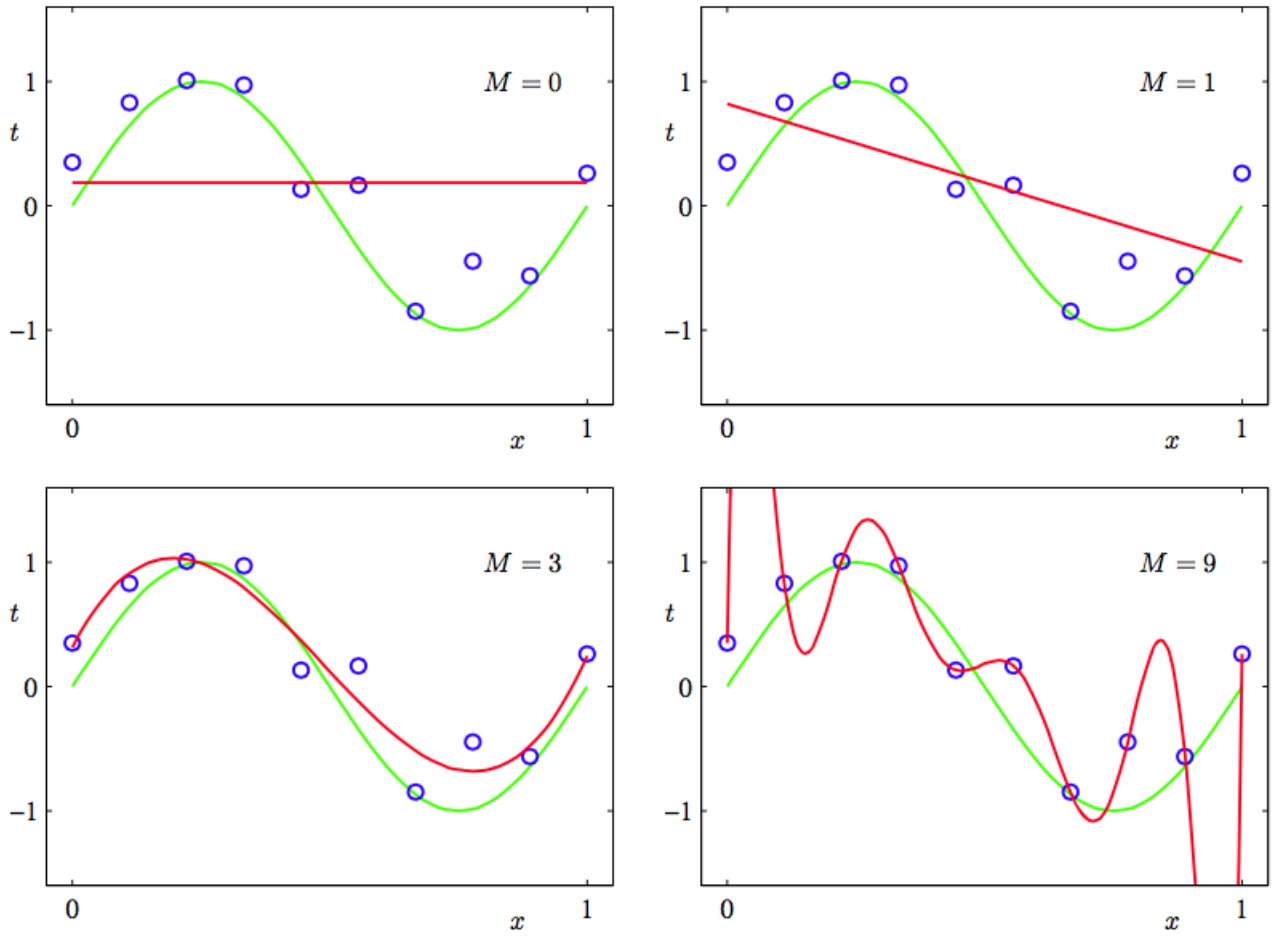

Figure 1.5: Different fitted polynomial curves for various degrees M.