6.036 Recitation Notes
April 6th, 2017
Nathan Landman, Elizabeth Shen

For more detailed derivations, see pages 74-89 of the lecture notes.

1 Generalization Error

Suppose our data follows some unknown distribution P^*. Then, training and test data are essentially independently drawn samples.

The test error is

$$\mathcal{E}(h) = E_{(x,y)\sim P^*}[\![yh(x) \leq 0]\!] \tag{1}$$

Let the set of all classifiers be $\mathcal{H}$; ideally, we would like to find the classifier $h^* \in \mathcal{H}$ that minimizes the test error. However, we can only find $\hat{h}$, the classifier that minimizes **training error**.

Thus, our problem is to find a classifier h that performs well on the training data, and also generalizes well. For this reason, the expression in Eq. (1) is also called generalization error.

Since training data is randomly sampled, we could get many points that don't conform with the overall model. Thus, we can only bound our generalization error with *high probability*.

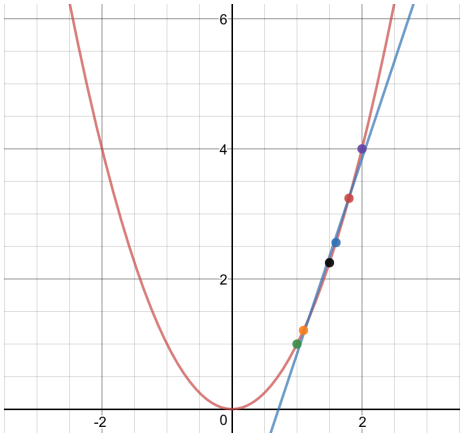

Figure 1: Points sampled from a quadratic model, but that seem to conform to a linear distribution

2 PAC

The "technical term" for this is "Probably Approximately Correct", or PAC guarantees, which states:

Definition 2.1. With probability at least $1 - \delta$ over the choice of the training set S_n from P^*, any classifier $\hat{h}$ that minimizes the training error $\mathcal{E}_n(\hat{h})$ has generalization error $\mathcal{E}(\hat{h}) \leq \epsilon$.

In other words, we want to minimize our generalization error ϵ, given a confidence level δ, training set size n, and classifier set $\mathcal{H}$. How many bad classifiers could there be? The probability that a classifier generalizes poorly but makes no mistakes on the training set is at most $(1-\epsilon)^n$. Then, the probability that all perfect classifiers are bad is $|\mathcal{H}|(1-\epsilon)^n$ and we have $\delta \leq |\mathcal{H}|(1-\epsilon)^n$.

Performing some algebra gives our final result

$$\epsilon \leq \frac{\log|\mathcal{H}| + \log(1/\delta)}{n} \tag{2}$$

3 Growth Function, VC Dimension

Note that the above only holds when $\mathcal{H}$ is finite- however, many sets of classifiers are infinite.

To account for this, instead of working with the number of classifiers, we can do something clever- use the number of possible labelings that a set of classifiers induces. For n points, there are 2^n distinct labelings.

A set of classifiers may not be able to generate all possible labelings, so we introduce the *growth function* $N_{\mathcal{H}}(n)$.

Definition 3.1. Growth function: The maximum number of labelings of n points that a set of classifiers $\mathcal{H}$ can correctly classify.

Doing some more complex math (not needed for this course), we derive:

$$\mathcal{E}(\hat{h}) \leq \mathcal{E}_n(\hat{h}) + \sqrt{\frac{\log N_{\mathcal{H}}(2n) + \log(4/\delta)}{n}}, \text{ for all } \hat{h} \in \mathcal{H} \tag{3}$$

This bound is only meaningful when $\log N_{\mathcal{H}}(n) \ll n$, so we introduce an additional complexity measure called the *Vapnik-Chervonenkis* dimension, denoted $d_{\mathcal{H}}$.

Definition 3.2. VC dimension: The largest number of points that arranged in some optimal configuration, can perfectly classified with classifiers from $\mathcal{H}$, regardless of their labelling.

It turns out for good PAC bounds, we need $n > d_{\mathcal{H}}$.

Example 3.1 In lecture, we showed that the VC-dimension of linear classifiers in $\mathbb{R}^d$ is $d + 1$.

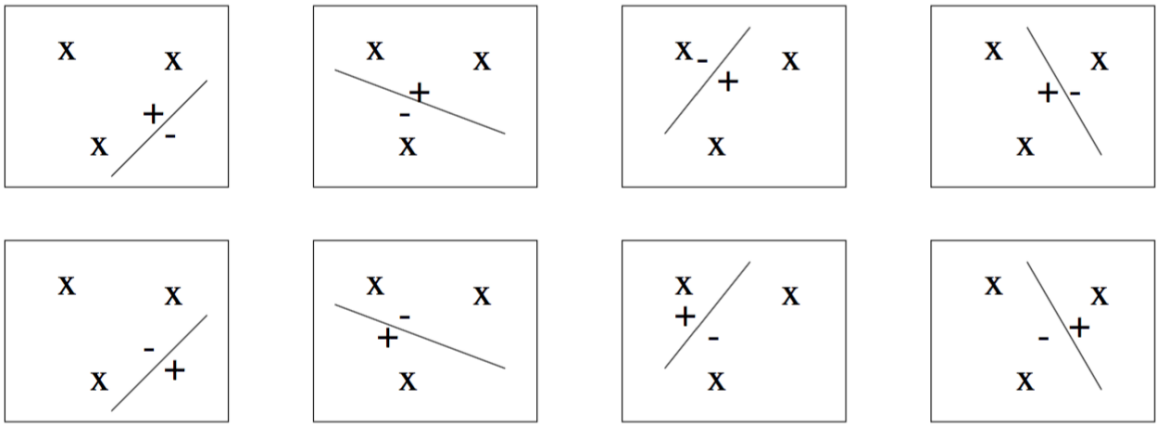

Figure 2: Linear classifiers can shatter 3 points.

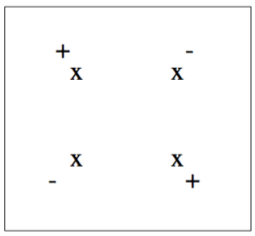

Figure 3: Linear classifiers **cannot** classify 4 points.

Exercise 3.1. *What is the VC-dimension of axis-aligned rectangles in* $\mathbb{R}^2$*?*

4 Unsupervised Learning: Clustering

4.1 Introduction

In unsupervised learning problems, we are given unlabeled data and want to infer hidden structure in the data. One such task is clustering, which groups points together based on perceived similarities.

Generally, clustering problems have two **inputs**.

- A dataset consisting of n samples: $S_n = \{x^{(i)} | i = 1...n\}$
- The number of clusters, k, used to separate the data.

Given this information, we want to **output** a set of k clusters, $\{C_1, C_2, ..., C_k\}$, which satisfy the following inequalities:

- $C_i \cap C_j = \varnothing, \forall i \neq j$, i.e. no two sets contain the same sample.
- $\bigcup_{j=1}^{n} C_j = S_n$, i.e. the union of all sets contains all n points.

We sometimes also assign a *representative* to each cluster, $z(i)$, which can be used as an example of the cluster.

4.2 Examples

Clustering can be used to group movies based on genres, news articles based on topic (like Google News), languages by similar origin, or even the MNIST dataset used in Project 2 (if the digits were unlabeled).

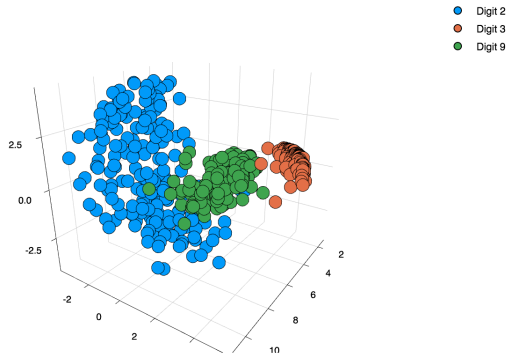

Figure 4: Clustering of MNIST digits 2, 3, and 9. Labels not used during clustering, but used for visualization.

4.3 Clustering as Cost Minimization

In order to find clusters with points "as close to each other as possible", we first need to define a distance metric. For our purposes, we will use the squared Euclidean distance:

$$dist(x^{(i)}, x^{(j)}) = ||x^{(i)} - x^{(j)}||^2 = \sum_{l=1}^{d} (x_l^{(i)} - x_l^{(j)})^2 \tag{4}$$

Next, we need to establish the *distortion*, or cost associated with each cluster. This could be measured by: the summed distance between every pair of points within the cluster, the diameter of the cluster, the summed distance between each point and the mean of the cluster, etc.

We will be using the last mentioned configuration.

Finally, the overall cost of the current cluster configuration is just the sum of each cluster's distortion.

$$Cost(C_1, ..., C_k) = \sum_{j=1}^{k} Cost(C_j) \tag{5}$$

The following unsupervised learning algorithms work to minimize this cost function in order to generate the best possible clustering.

4.4 K-Means Algorithm

The K-means algorithm defines the distortion of a cluster C_j as the distance of each sample in the cluster to the centroid of the cluster, $z^{(j)}$. It iteratively generates new centroids and clusters until the total cost converges[1].

$$Cost(C_1, ..., C_k, z^{(1)}, ..., z^{(k)}) = \sum_{j=1}^{k} \sum_{i \in C_j} ||x^{(i)} - z(j)||^2 \tag{6}$$

The pseudocode is as follows:

1. Initialize random centroids $z^{(1)}, \cdots, z^{(k)}$.
2. Repeat until there is no further change in cost:
   (a) Assign points to each centroid. For each $j = 1, ..., k$: $C_j = \{i \text{ s.t. } x^{(i)} \text{ is closest to } z^{(j)}\}$
   (b) Given a cluster, pick the centroid. For each $j = 1, ..., k$: $z^{(j)} = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}$

The time complexity of each iteration is $O(kn)$, which is acceptable.

So why is the centroid the best cluster representative for our cost function? The proof is left as an exercise.

Exercise 4.1. *Show that given a cluster C_j, the centroid $z^{(j)} = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}$ minimizes the sum-of-squares loss. Do this by setting the gradient of the cost function w.r.t $z^{(j)}$ to 0 and solving.*

4.5 K-Medoids Algorithm

One problem with K-means clustering is that the centroid is often not a data point, which doesn't make sense in some contexts, such as returning a top news story in each category.

The K-medoids algorithm fixes this issue by selecting one of the original points as the cluster representative, now called an *exemplar*. The algorithm is otherwise very similar to K-means.

4.6 K++ Initialization

Another issue with K-means and K-medoids is that since the algorithms converge to a local minimum, they may perform poorly when initial representatives are close together. To combat this, we can use the K++ initialization mechanism, which picks the k centers one at a time, weighted towards choosing points further away from already selected centers.

Initialization: Select the first exemplar $z^{(1)}$ uniformly at random from all points.
For $j = 1, ..., k-1$:

1. For each point $x^{(i)}$ and newest exemplar $z^{(j)}$, compute the distance $dist(x^{(i)}, z^{(j)})$.
2. Select a new exemplar $z^{(j+1)}$ at random, where point $x^{(i)}$ is chosen with probability proportional to $dist(x^{(i)}, z^{(j)})$.

[1]The proof that k-means converges is in the text.