[Summary]Unsupervised Learning

Clustering

Unsupervised Learning: Introduction

Unsupervised learning is contrasted from supervised learning because it uses an **unlabeled** training set rather than a labeled one.

Clustering is good for:

- Market segmentation
- Social network analysis
- Organizing computer clusters
- Astronomical data analysis

K-Means Algorithm

The K-Means Algorithm is the most popular and widely used algorithm for automatically grouping data into coherent subsets.

1. Randomly initialize two points in the dataset called the **cluster centroids**.
2. Cluster assignment: assign all examples into one of two groups based on which cluster centroid the example is closest to.

3. Move centroid: compute the averages for all the points inside each of the two cluster centroid groups, then move the cluster centroid points to those averages.
4. Re-run (2) and (3) until we have found our clusters.

Our main variables are:

- K (number of clusters)
- Training set $x^1, x^2, \ldots, x^m$
- Where $x^i \in \mathbb{R}^n$

Note that we **will not use** the $x_0 = 1$ convention.

Cluster Assignment

The **first for-loop** is the '**Cluster Assignment**' step. We make a vector c where c(i) represents the centroid assigned to example x(i).

We can write the operation of the Cluster Assignment step more mathematically as follows:

$$c^i = \arg\min_{k} \left|\left|x^i - \mu_k\right|\right|^2$$

That is, each c^i contains the index of the centroid that has minimal distance to x^i.

The square convention serves two purposes, **minimize more sharply** and **less computation**.

Move Centroid

The **second for–loop** is the '**Move Centroid**' step where we move each centroid to the average of its group.

More formally, the equation for this loop is as follows:

$$\mu_k = \frac{1}{n}[x^{k_1} + x^{k_2} + \cdots + x^{k_n}] \in \mathbb{R}^n$$

Where each of $x^{k_1}, x^{k_2}, \cdots, x^{k_n}$ are the training examples assigned to group $m\mu_k$

If you have a cluster centroid with **0 points** assigned to it, you can randomly **re–initialize** that centroid to a new point. You can also simply **eliminate** that cluster group.

After a number of iterations the algorithm will **converge**, where new iterations do not affect the clusters.

Note on non–separated clusters: some datasets have no real inner separation or natural structure. K–means can still evenly segment your data into K subsets, so can still be useful in this case.

Optimization Objective

Recall some of the parameters we used in our algorithm:

- c^i = index of cluster (1,2,...,K) to which example x(i) is currently assigned
- μ_k = cluster centroid k ($\mu_k \in \mathbb{R}n$)

- μ_{c^i} = cluster centroid of cluster to which example x(i) has been assigned

Using these variables we can define our **cost function**:

$$J(c^i, \dots, c^m,\ \mu_1, \dots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m} \left|\left|x^i - \mu_{c^i}\right|\right|^2$$

Our **optimization objective** is to minimize all our parameters using the above cost function:

$$\min_{c,\mu} J(c, \mu)$$

That is, we are finding all the values in sets c, representing all our clusters, and μ, representing all our centroids, that will minimize **the average of the distances** of every training example to its corresponding cluster centroid.

The above cost function is often called the **distortion** of the training examples.

Cluster assignment

In the **cluster assignment step**, our goal is to:

Minimize J(…) with $c^1,\ \dots,\ c^m$ (holding $\mu_1,\ \dots,\ \mu_K$ fixed)

In the **move centroid** step, our goal is to:

Minimize J(…) with $\mu_1,\ \dots,\ \mu_K$

With k–means, it is **not possible for the cost function to sometimes increase**. It should always descend.

Random Initialization

There's one particular recommended method for randomly initializing your cluster centroids.

1. Have K<m. That is, make sure the number of your clusters is less than the number of your training examples.
2. Randomly pick K training examples. (the selected examples are unique).
3. Set $\mu_1, \ldots, \mu_K$ equal to these K examples.

K–means **can get stuck in local optima**. To decrease the chance of this happening, you can run the algorithm on many different random initializations. In cases where K<10 it is strongly recommended to run a loop of random initializations.

Choosing the Number of Clusters

Choosing K can be quite arbitrary and ambiguous.

The elbow method: plot the cost J and the number of clusters K. The cost function should reduce as we increase the number of clusters, and then flatten out. Choose K at the point where the cost function starts to flatten out.

However, fairly often, the curve is **very gradual**, so there's no clear elbow.

Note: J will **always** decrease as K is increased. The one exception is if k–means gets stuck at a bad local optimum.

Another way to choose K is to observe how well k–means performs on a **downstream purpose**. In other words, you choose K that proves to be most useful for some goal you're trying to achieve from using these clusters.