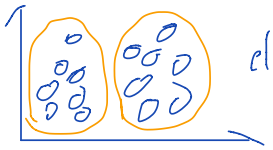
Clustering — Unsupervised Learning Intro

Unsupervised Learning



clustering algorithm

Training set $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots\cdots, x^{(m)}\}$

Application of clustering.

- Market segmentation
- Social Network analysis
- Organize Computing clusters
- Astronomical data analysis
- ……

clustering is an example of unsupervised learning

Clustering — K-means algorithm

- Input:
  - K (number of clusters)
  - Training set $\{x^{(1)}, x^{(2)}, \cdots\cdots, x^{(m)}\}$

    $x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)
- Procedure
  - Randomly initialize K cluster centroids $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^n$
  - Repeat {

    for $i = 1$ to m (cluster assignment)

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

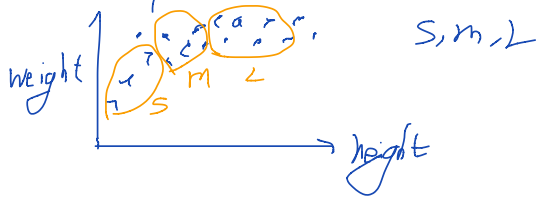
for k = 1 to K: (update centroid)

μ_k := average (mean) of points assigned to cluster k

}

If not points assigned to a certain centroid during a certain step, then elimate the corresponding centroid or randomly re-initialize that centroid

K-means for non-separated clusters



S, M, L

- clustering — optimization objective

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

Optimization objective

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

$\min_{c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K} J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)$

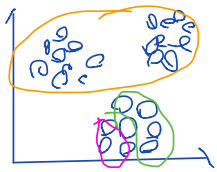
Cost function / distortion of K-mean algorithm

- clustering – Random initialization

  - should have $K < m$
  - Randomly pick K training examples
  - set $\mu_1, \dots \mu_K$ equal to these K examples

Problem of local optima

Assume $K=3$, could end up with:

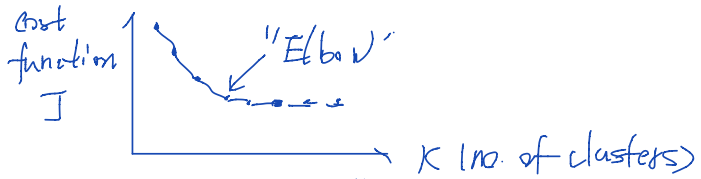


Solution: Run K-means clustering several times

```
for i=1 to 100 {        → 50 – 1000
    Randomly initialize K-means
    Run k-means, Get c^(1), ... c^(m), μ1, ... μK
    Compute cost function (distortion)
        J(c^(1), ... c^(m), μ1, ... μK)
}
Pick clustering that give lowest cost J(...)
```

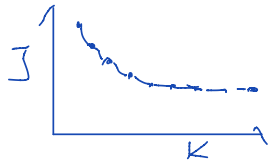
- Clustering – choosing the number of clusters

— Elbow method



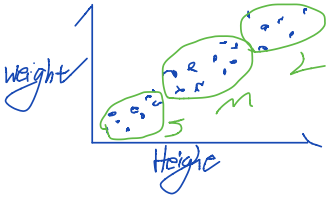
choose K as where corresponding to the "Elbow"

* but "Elbow" is not always obvious, e.g.

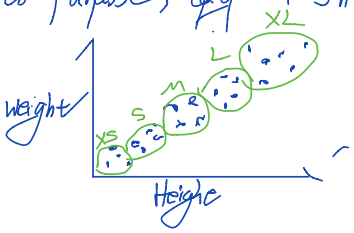


— Base on later purpose

Sometimes you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose, e.g. T-shirt size:



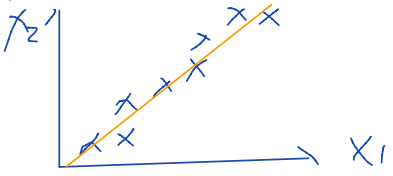
size: S, M, L



size: XS, S, M, L, XL

Dimensionality Reduction

- Motivation I: Data Compression

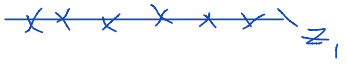


$x^{(1)} \in \mathbb{R}^{(2)} \rightarrow z^{(1)} \in \mathbb{R}$

$x^{(2)} \in \mathbb{R}^{(2)} \rightarrow z^{(2)} \in \mathbb{R}$

$x^{(3)} \in \mathbb{R}^{(3)} \rightarrow z^{(3)} \in \mathbb{R}$

2D $\rightarrow$ 1D



* this is purely a data projection

3D → 2D: Project 3D data to 2D plane

- Motivation II: Data visualization

reduce dimentionality to view high dimention data on 2D or 3D or 1D

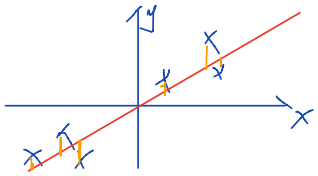
Principal Component Analysis (PCA)

- Problem formulation

Reduce 2D → 1D: find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) on to which to project the data so as to minimize the projection error.

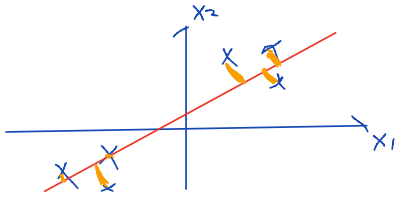
Reduce from n-D → k-D: find k vectors $u^{(1)}, \dots, u^{(k)}$ on to which to project the data, so as to minimize the projection error.

* PCA is not Linear Regration



Linear regression

{ x Vs y



PCA

{ x_1 Vs x_2, no y

vertical distance projection distance

- PCA Algorithm

1) Data preprocessing (feature scaling / mean normalization)

- $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$
- Replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$
- if different features on different scales, scale features to have comparable ranges

Problem: how to compute $\mu^{(i)}$ and z

2) Algorithm: reducing data from $n \to k$:

- Compute "Covariance matrix"

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} \underbrace{(x^{(i)})}_{n \times 1} \underbrace{(x^{(i)})^T}_{1 \times n} \Rightarrow n \times n$$

- Compute "eigenvectors of matrix Σ":

[U, S, V] = svd(sigma) (In mathematica)

svd: Singular Value decomposition

$$U (n \times n) = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{K} \text{ eigenvectors}$$

Take the first k eigenvectors

- Take first k eigenvectors ... - $\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \end{bmatrix}$

U_{reduce} — $n \times k$

$$z = U_{reduce}^T x$$

PCA actually minimizes squared projection error

- Reconstruction from compressed representation

  How do you go back from $z^{(i)}$ to $x^{(i)}$?

  $z \in \mathbb{R}^k \rightarrow x \in \mathbb{R}^n$

$$z = U_{reduce}^T x \Rightarrow x_{approx}^{(i)} = \underset{n \times 1}{U_{reduce}} \cdot \underset{k \times 1}{z^{(i)}}$$

- choosing the number of principal Component k

Averaged squared projection error: $\frac{1}{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

Total variation in the data: $\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2$

Typically, choose k to be smallest value so that:

$$\frac{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \quad (1\%)$$

("99% of variance is retained")

Algorithm:

try $k = 1, 2, \ldots$

Compute $U_{reduce}, z^{(1)}, \ldots z^{(m)}, x_{approx}^{(1)} \ldots x_{approx}^{(m)}$

check if $\frac{\frac{1}{m} \| \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01$

if true, break. find good k value

if False, continue.

$[U, S, V] = \text{Svd}(\text{Sigma})$

$S = \begin{bmatrix} S_{11} & & \\ & S_{22} & \\ & & \ddots \\ & & & S_{nn} \end{bmatrix}$

for a given k:

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.1$$

Only need to update numerator for different k

- Advice for applying PCA

Use PCA to speed up learning algorithm

Supervised learning speed up

$(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$

Extract inputs:

unlabled dataset: $x^{(1)}, x^{(2)}, \ldots x^{(m)} \in \mathbb{R}^n$

$\Downarrow$ PCA

$z^{(1)}, z^{(2)}, \ldots z^{(m)} \in \mathbb{R}^{(m \times k)}$

New training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots (z^{(m)}, y^{(m)})$

New test example

$x \xrightarrow{\text{PCA}} z$

$y = h_\theta(z) \longrightarrow$ predictions

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets

Application of PCA:

- Reduce memory/disk needed to store data
- Speed up learn algorithm
  (choose K by % of variance retain)
- Visualization
  K=2 or K=3 for visualize high dimentional data

Bad use of PCA I:

Bad thinking: PCA → reduce num of features
→ prevent overfitting

Actually: This might be ok, but isn't a good way to address overfitting. Use regularization instead

Because: PCA doesn't information of y involved, it might through away some valuable information.
While regularization has all information retained.

Bad use of PCA II.

Design of ML system (Bad)

1) Get training set $(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})$

2) Run PCA to reduce $x^{(i)}$ to $z^{(i)}$

3) Train

4) Test on test set: map $x_{test}^{(i)}$ to $z_{test}^{(i)}$, run $h(z^{(i)})$

Suggestion:

Before implementing PCA, first ~~try~~ running ML on ~~the~~ raw data, Only if ~~that~~ doesn't do what you want, ~~then~~ implement PCA