ML

Andrew Ng

Coursera

Week 8

Date:

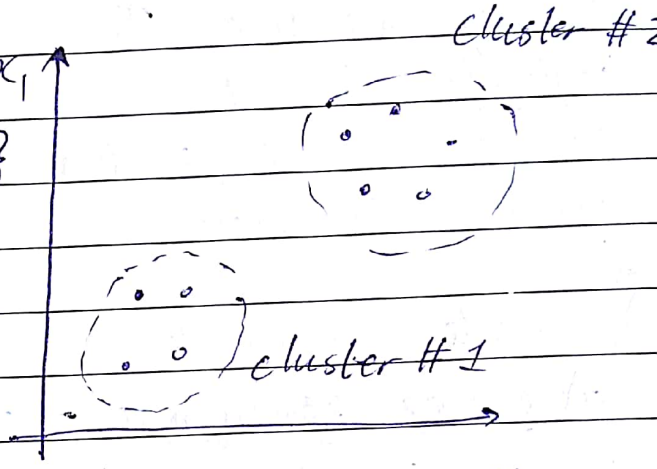
Unsupervised Learning Intro

Clustering - unsupervised learning algo

↳ learn from unlabeled data

Training set:

$\{x^{(1)}, x^{(2)}, x^{(3)} \ldots x^{(m)}\}$



In unsupervised learning we give untabelled training set to an algorithm & ask the algo to find some structure within the data. The structure found in the dataset ~~are~~ is shown as clusters.

Clustering algo - an algo that finds clusters

Applications of clustering

- Market segmentation
- social network analysis
- ~~&etc~~
- organise computing clusters and etc

↗ which computers communicate between each other the most & hence change network according to that

Victory

Date:

K-Means Algorithm

↳ most ~~popular~~ popular, widely used clustering algo (an iterative algo)

Steps of k-means :

1) randomly initialize ~~2~~ points called cluster centroids - the # of these points depend on the # of clusters you wish to group your data into

2) Iterate over a, b

a) cluster assignment - each data point is assigned to a cluster centroid depending on how close the data point is from the cluster centroid (data point assigned to closest cluster centroid)

b) move centroid - the cluster centroid location is changed to the average of all of the points assigned to that cluster centroid

c) Iterate until no data points switch from one cluster to the other in assignment & hence average position of cluster centroid does not change.

↳ algorithm converges

Victory

$k \in \{1, 2, \ldots, K\}$

Date:

Inputs to algorithm:
- k (number of clusters)
- training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop "$x_0 = 1$ convention")

This convention is only present in supervised learning, not in unsupervised learning

capital "K" - total # of clusters
lowercase "k" - indexing into # of clusters, count variable

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

$c^{(i)}$ holds closest cluster index of $x^{(i)}$

Repeat {

cluster assignment step:

for $i = 1$ to m

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

$$c^{(i)} = \min_k \|x^{(i)} - \mu_k\|^2$$

adding square is simply convention

index into all possible clusters

code continued...

Date:

Move Centroid {

for $k = 1$ to K

$\mu_k :=$ average (mean) of points assigned to cluster k

}

suppose $x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$ assigned to cluster 2 then:

$c^{(1)} = 2, c^{(5)} = 2, c^{(6)} = 2, c^{(10)} = 2$

& the move centroid step is then:

$$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$$

If a centroid has 0 points assigned to it then how is the move centroid step done?

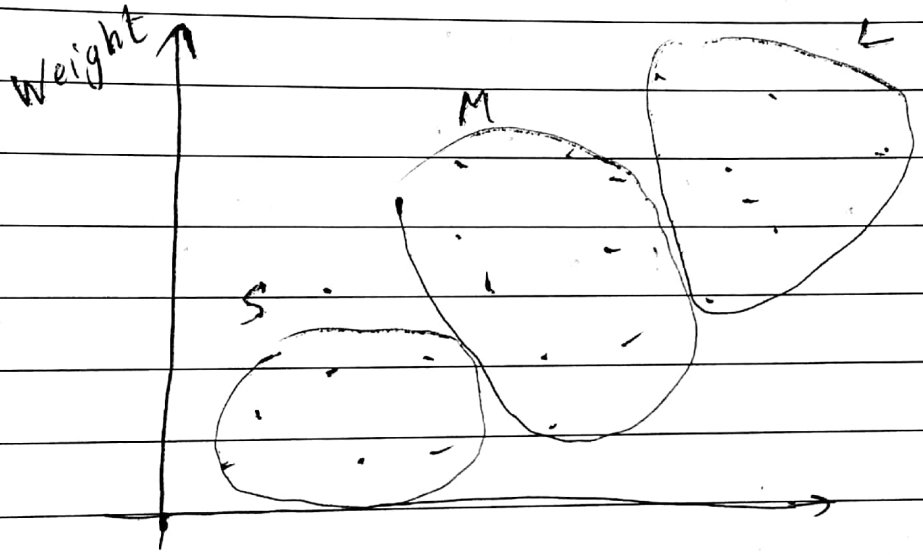
1) oftenly the ~~centr~~ cluster centroid is dropped

2) seldomly (if K clusters are really needed) then the cluster centroid is randomly initialised again

Date:

K means for non-separated clusters

Application scenario : a T-shirt manufacturer has gathered the weight & height of people within a certain population where he/she plans on selling T-shirts. Find the size of S, M, L that he can use to better fit the population.



Apply the clustering algo on this data set with 3 centroids

↳ sort of an example of market segmentation

Date:

Optimization Objective

K means also has an optimization objective — a cost function it is trying to minimise

$c^{(i)}$ = index of cluster (1, 2, ... K) to which $x^{(i)}$ is currently assigned

μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Keep in mind that: $k \in \{1, 2, \ldots, K\}$

so if $x^{(i)}$ → assigned to cluster 5

then $c^{(i)} = 5 \rightarrow \mu_{c^{(i)}} = \mu_5$

Optimisation Objective :

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

Sometimes called distortion cost function or distortion of k-means algo

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

Victory

Page No.

wrt = with respect to

Date:

Cluster assignment step → it carries out this:

min $J(\ldots)$ wrt $c^{(1)}, c^{(2)}, \ldots, c^{(m)}$
(holding $\mu_1, \ldots, \mu_K$ fixed)

Move centroid step carries this out:

min $J(\ldots)$ wrt $\mu_1, \ldots \mu_K$

It can be proved mathematically that this is happening

This can be seen as a 2-step process where initially $J(\ldots)$ is minimised wrt c & then it minimises it wrt μ

$J(\ldots)$ used to

$J(\ldots)$ can be used to find better clusters & help k-means avoid local optima.

Date:

Random Initialization

↳ how to initialise k-means & how to avoid local optima

How do you randomly initialise the cluster centroids?

- you should have $K < m$.
- randomly pick K training examples in your ~~data set~~ training data set
- set $\mu_1, \ldots, \mu_K$ equal to these K examples then:

$\mu_1 = x^{(i)}$

$\mu_2 = x^{(j)}$ (for some random $i, j, \ldots, l$)

$\vdots$

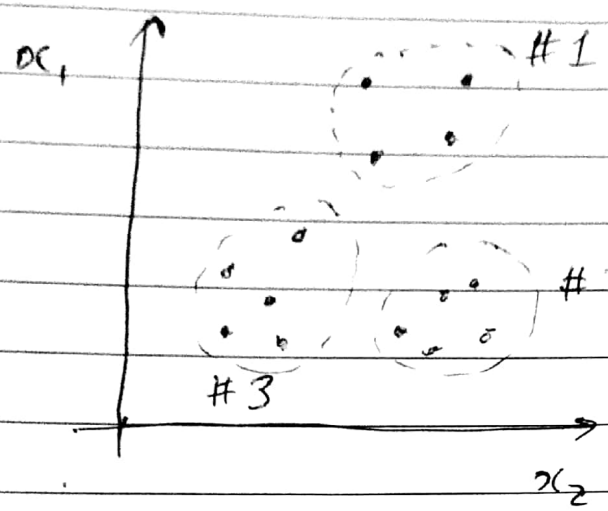
$\mu_K = x^{(l)}$

K-means may end up converging to different solutions depending on how the clusters were initialised

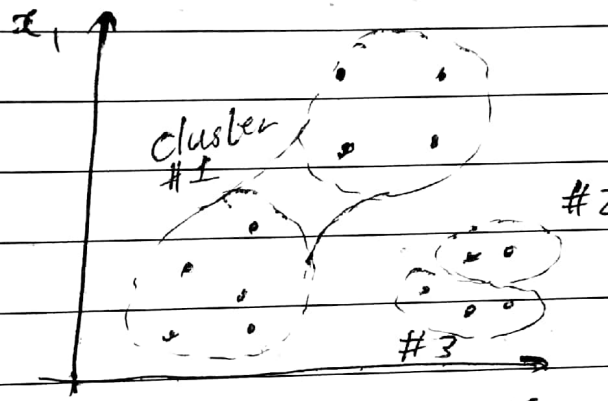
↳ K-means can end up with different solutions & can end up at a local optima

Date:

Suppose the data set:



Global optima of this data set



Local optima of this data set

least possible value of cost

not least possible value of cost

optima

The global ~~function~~ minimises function $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$ ~~optimally~~ while the ~~local~~ local optima does not

To avoid the local optima run ~~the~~ k-means multiple times with random initialization & find the best solution possible this way

Page No.

Victory

→ 50 – 100

Date:

for i = 1 to 100 {

Randomly initialize K-means
Run K-means. Get $c^{(1)}, \ldots, c^{(2)}, \mu_1, \ldots, \mu_K$
Compute cost function (distortion)
$J(c^{(1)}, \ldots, c^{(2)}, \mu_1, \ldots \mu_K)$
}

Run K-means multiple of times & then pick clustering that gives the lowest cost.

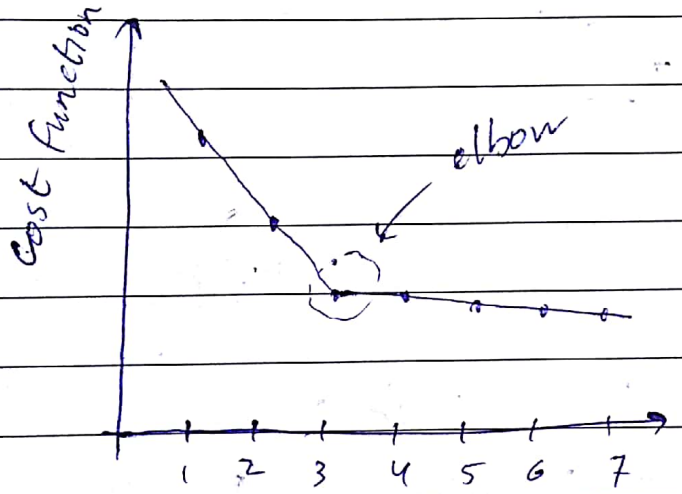
If there are few K clusters (till up till 10) then doing random initializations can help in choosing a better optima. But if K is large (greater than 10) then doing multiple random initialisation is less likely to make a great difference.

Choosing the # of clusters.

↳ done mostly by hand

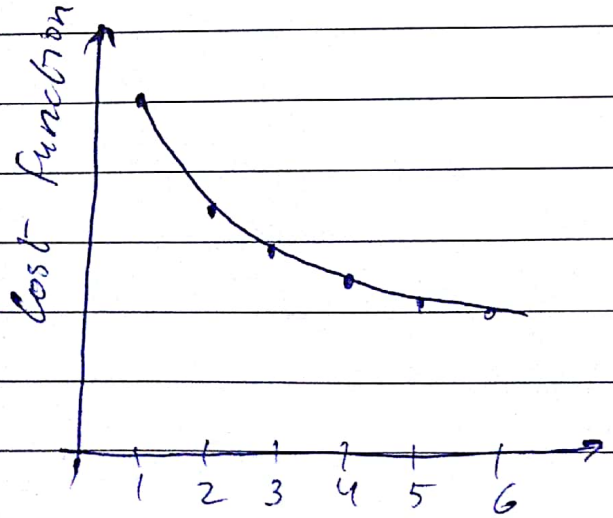
The reason why at times choosing the # of clusters is difficult is because it's ambiguous how many clusters there even are in the dataset

Elbow method



It's clear that 3 is a good candidate for the # of clusters here as it is the elbow

However the elbow method can be ambigous at times as well:

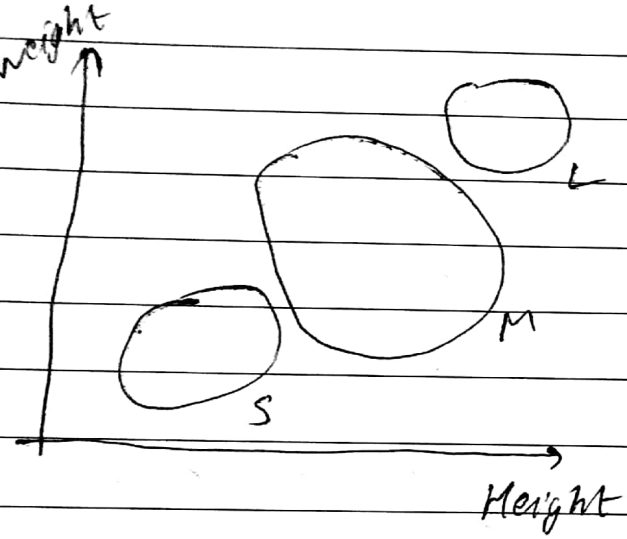


No elbow hence ambiguous.
The elbow method can be used but chances are it may not work & won't be able to give an answer

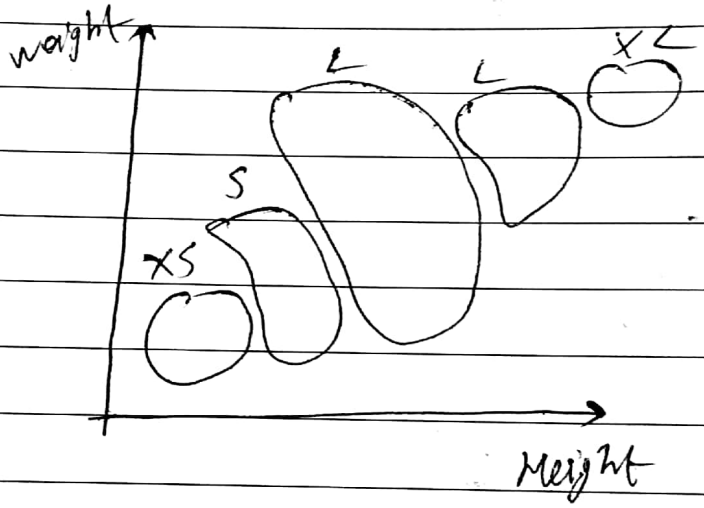
Date:

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. Looking back at the t-shirt manufacturer example:

K=3 : S, M, L



K=5 : XS, S, M, L, XL



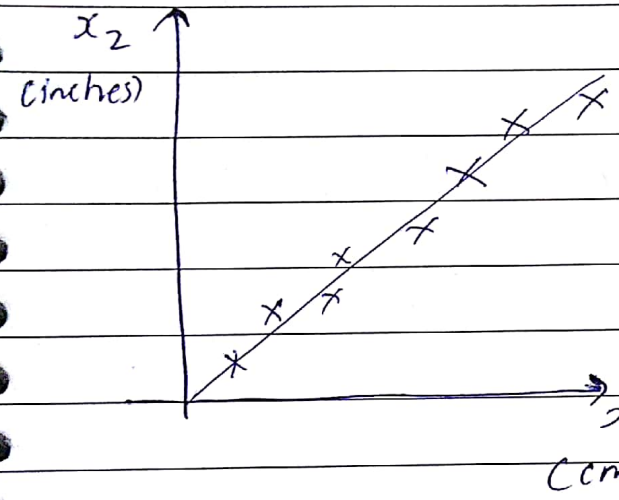
The T-shirt business may have their own performance metrics as to whether 3 or 5 sizes will fit them better.

Data compression

Dimentionality reduction - type of unsupervised algo

↳ can help achieve data compression

↳ allows to speed up learning algorithms



Suppose that x_1 & x_2 both measure the length of some object in cm & inches respectively. The data points don't ~~like~~ lie perfectly on the line because the features could have been rounded off to the nearest-inch or cm

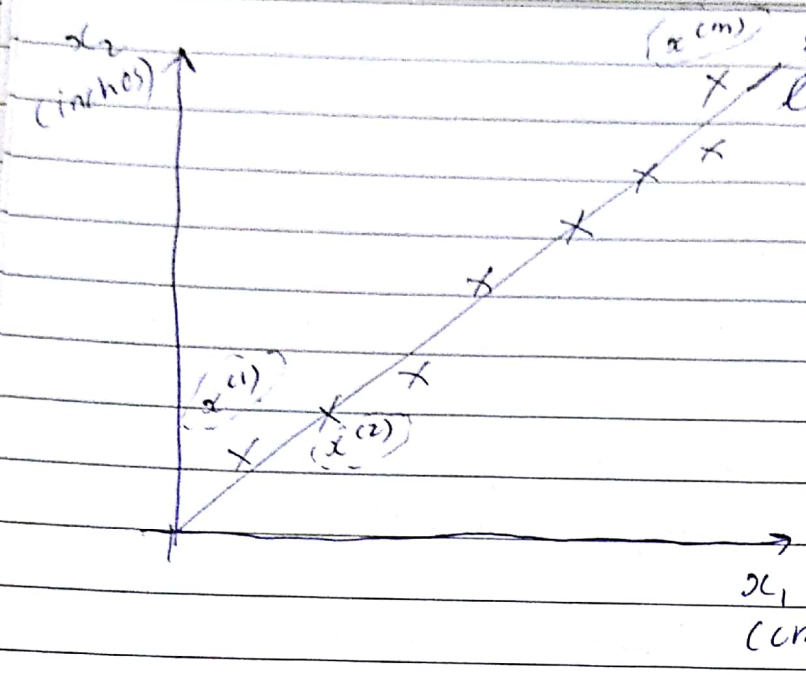
Reduce this data from 2D to 1D

↳ reduces redundancy

If you have highly correlated feature then you might want to loose a dimension (example: pilot enjoyment & pillot skill will be highly correlated but not the same thing)

Previously shown graph:

Date:



Reducing 2D to ~~2D~~ 1D

Project all the data points on the ℓ line & then measure the position of all of these examples on ℓ

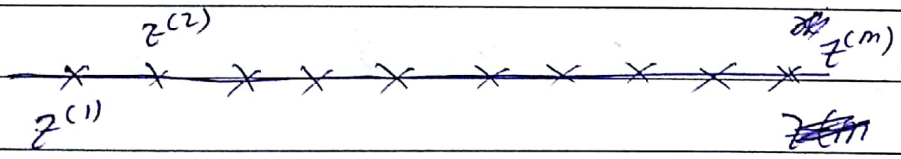
Create new feature z which is the position of data point on line ℓ

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

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

$\vdots$

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



Basically this method effectively replaces the need to have 2 dimensions. Now you can have just 1 (~~this is an approximation~~ to ~~the original~~

Date:

the new data set is an approximation to the original data set

Consequence of doing this:
- halves memory requirement
- makes learning algorithm run quickly

It is possible to reduce 10000 to 1000 or 3D to 2D if needed

↳ project all points in 3D space to a plane

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \end{bmatrix} \in \mathbb{R}^3 \rightarrow z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix} \in \mathbb{R}^2$$

Data Visualisation

↳ second application of dimentionality reduction

You can take a very lary dataset suppose that has 50 columns & reduce each data point into a 3D or 2D vector which you can plot to understand the data better.

Note: After dimentionality reduction, the new features are not ascribed any meaning, the scientist must find that himself

Victory

Page No.

Principal Component Analysis – algo that allows us to do dimentionality reduction

Date:

~~Principle compo~~

Principal Component Analysis Problem Formulation

PCA (Principal Component Analysis) – most widely used algo of dimentionality reduction

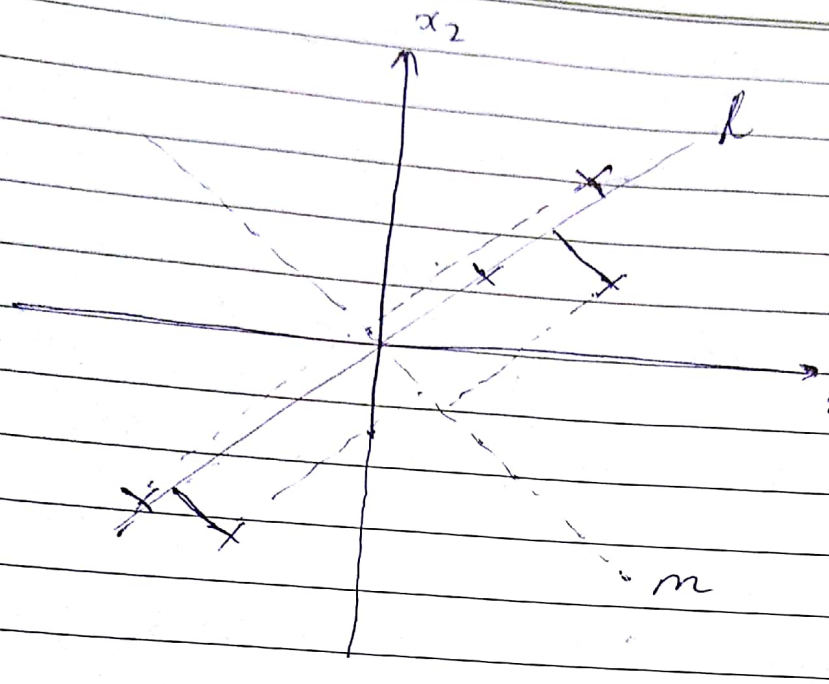
~~Problem~~: PCA finds a lower dimensional surface onto to which to project the data so that the sum of squares of the distances (from data points) to the ~~line~~ (surface) is minimized

Projection error – ~~sum~~ distance of ~~line~~ (surface) from data point

Before applying PCA, it's standard practice to first perform mean normalisation so that all features have 0 mean & perform feature scaling (features have comparable ranges)

2D → 1D

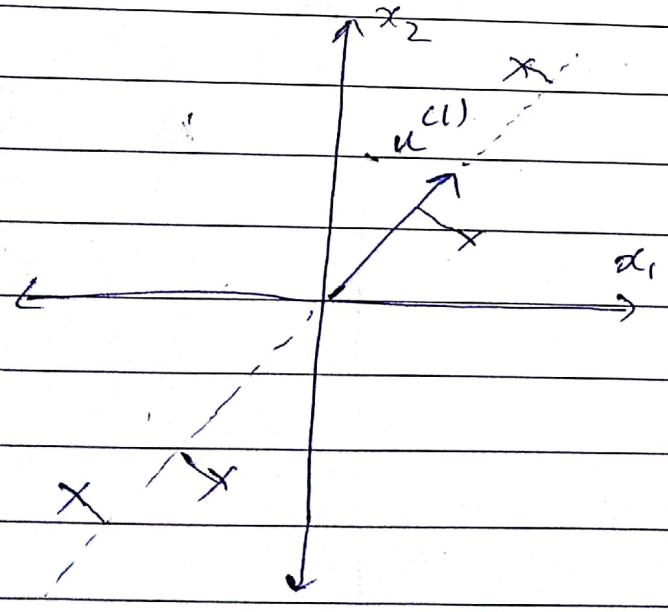
Date:



lower dimensional surface here is a line. PCA chooses such a line so that the distance is minimized between data point & line

l will be chosen as opposed to m because distance is lower with l.

PCA problem formulation



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

Does not matter if PCA outputs $-u^{(1)}$ as the direction is still the same.

The projection ~~of~~ is usually at 90° / orthogonal

~~or thus~~ Orthogonal projection distance is considered the error

Date:

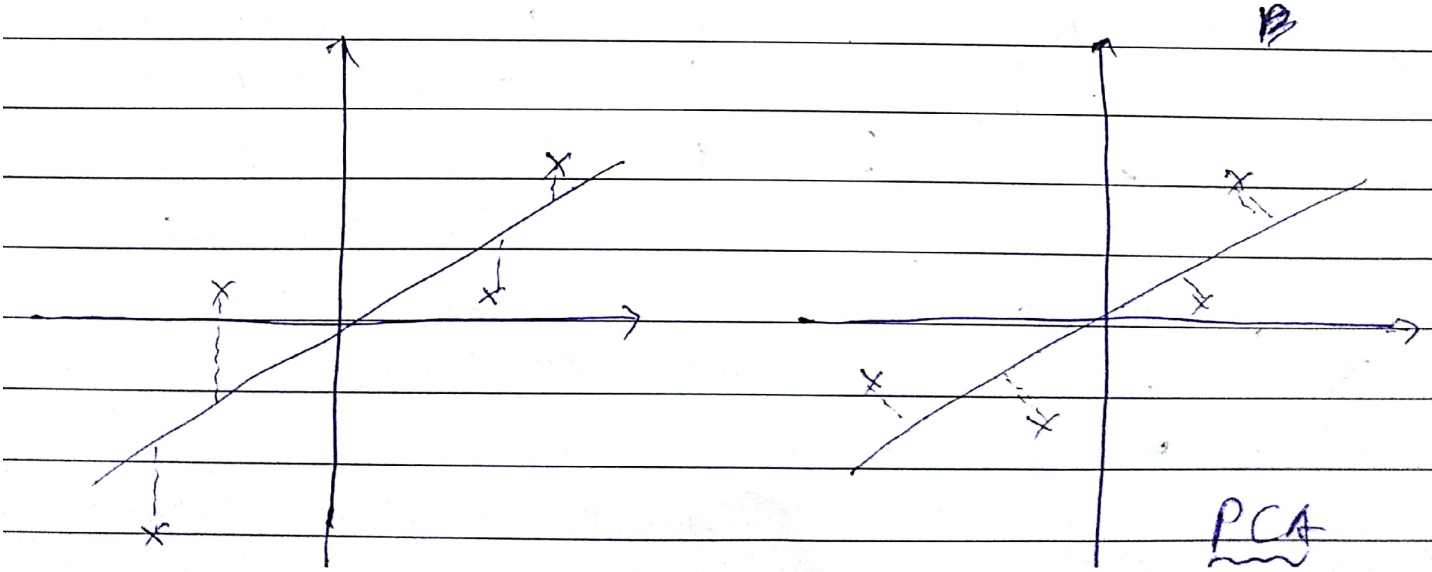
Reduce from n-dimension to k-dimension:
find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error

In the case of 3D → 2D : k = 2
Hence find $u^{(1)}$ & $u^{(2)}$ that creates a plane onto which data points can be projected

Linear algebra jargon: find set of vectors $u^{(1)}, u^{(2)} \ldots u^{(k)}$ & project the data onto the linear subspace spanned by this set of k vectors

PCA is not linear regression

↳ there are some similarities but ~~not exactly the same~~ different algos



~~Linear regression~~:
non-orthogonal distances minimized

Date:

In PCA the orthogonal distances are minimised

linear regression has a special variable y that we are trying to predict, PCA has no such variable

By convention, we choose $u^{(1)}$ so that $\|u^{(1)}\| = \sqrt{[u^{(1)}]^2 + [u_2^{(2)}]^2} = 1$

PCA Algorithm

Data preprocessing

Before ~~any~~ PCA is ~~done~~ executed the following must be done:

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

Preprocessing (feature scaling, mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

μ_j: mean of feature j

mean normalization of individual feature

Replace each $x_j^{(i)}$ with $x_j - \mu_j$

If different features on different scales, ~~scl~~ scale features to have comparable range of values

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

s_j ← standard deviation of feature j

Victory

Page No.

Date:

The PCA calculates 2 things:

- $u^{(1)}, u^{(2)}, \ldots u^{(k)}$
- the values that higher dimensional points are mapped on to in the lower dimension

example: 3D → 2D

$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$

↳ find this value

$$z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

The mathematical proof of how to find these values is extremely complicated

Procedure of how to compute the wanted values:

PCA algorithm

Reduce data from n-dimensions to k-dimensions:

Compute "covariance matrix":

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

Σ: greek letter sigma

$(x^{(i)})$: $n \times 1$, $(x^{(i)})^T$: $1 \times n$

Σ: $n \times n$

Date:

Compute "eigenvectors" of matrix Σ :

$[U, S, V] = svd(sigma)$

↳ singular value decomposition

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix} \quad U \in \mathbb{R}^{n \times n}$$

take the first k dimensions required to create lower dimensional surface

reduced version of U

↳ $u^{(1)}, \ldots, u^{(k)}$

Define $U_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}$

To find out z in : $x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$ do this :

$$z^{(i)} = U_{reduce}^{T} x^{(i)} = \begin{bmatrix} — & u^{(1)} & — \\ — & u^{(2)} & — \\ & \vdots & \\ — & u^{(k)} & — \end{bmatrix} x^{(i)}$$

$z^{(i)} \in \mathbb{R}^k$

$k \times n$ $\quad$ $n \times 1$

$k \times 1$

done when features take on very different ranges of values

Date:

~~Principal~~ PCA algo summary

After mean normalization (ensure every feature has 0 mean) & optionally feature scalling :

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T = \underbrace{\frac{1}{m}(X^T)(X)}$$

Vectorised implementation where

$$X = \begin{bmatrix} — x^{(1)} — \\ \vdots \\ — x^{(m)} — \end{bmatrix}$$

```
[U, S, V] = svd(Sigma);
Ureduce = U(:, 1:k);
z = Ureduce' * x;
```

And as it is with unsupervised learning $x \in \mathbb{R}^n$

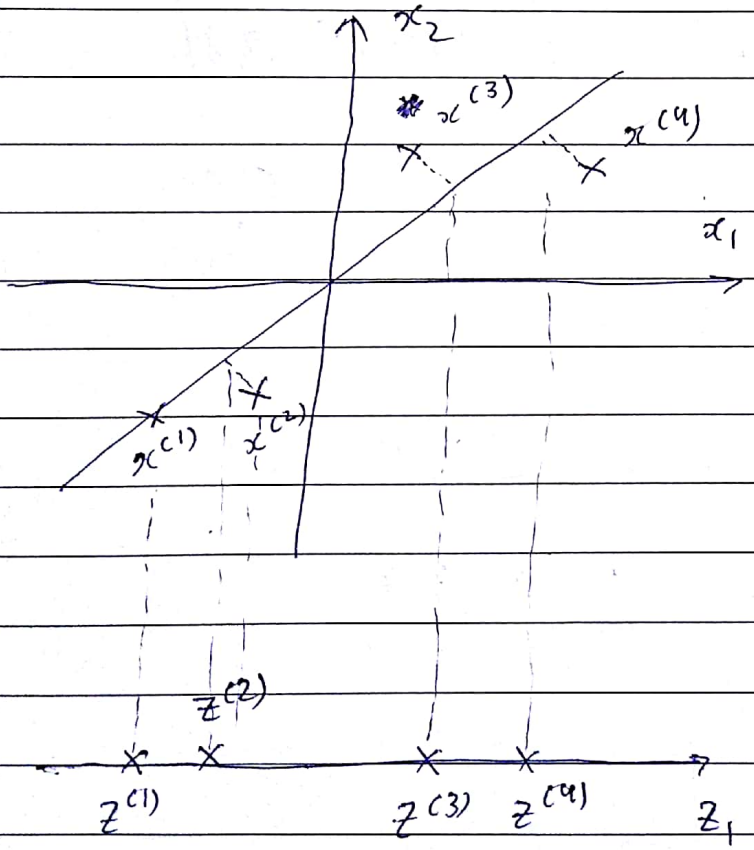
↳ $x_0 = 1$ is not used or does not exist

Date:

Reconstruction from compressed representation

PCA is a compression algorithm where a 1000D data is compressed into a 100D feature vector

How do you go back from the compressed representation to the original representation (or an approximation of it)



$$z^{(i)} = U_{reduce}^{T} x$$

Go back:

$z \in \mathbb{R} \rightarrow x \in \mathbb{R}$

$$x_{approx}^{(i)} = U_{reduce} \; z^{(i)}$$

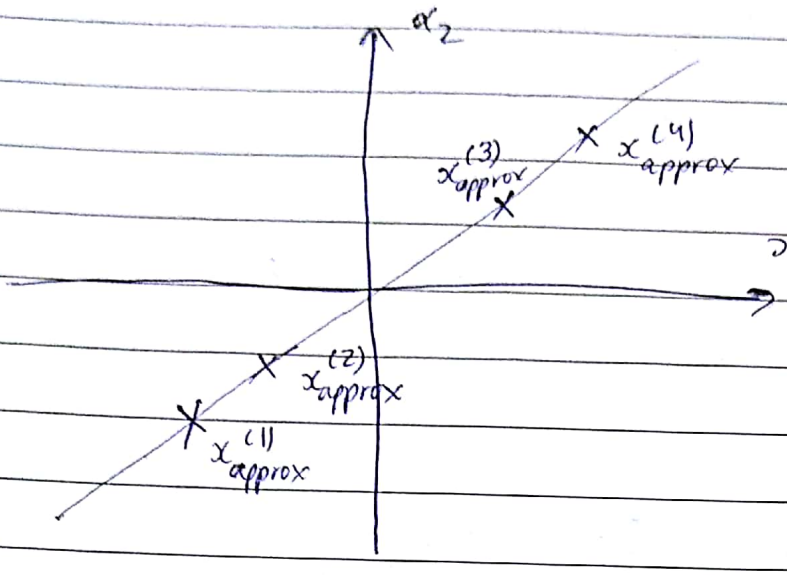
$(n \times 1) \quad (n \times k) \quad (k \times 1)$

$n \times 1$

if the square projection error is small then $x_{approx}^{(i)} \approx x^{(i)}$

square projection error when PCA was applied

Reconstructed values of x:



The values reconstructed will be on the line

Choosing the Number of Principal Components

k = # of principal components

PCA minimizes this

Avg squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x^{(i)}_{approx} \|^2$

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

↳ how much the data varries on average

Date:

Typically, choose k to be the smallest value so that:

$$\frac{\text{Avg squared projection error}}{\text{Variation in data}} \leq 0.01$$

↳ 1%

"99% variance is retained"

If 0.01 value changed then variance retained changes as well:

$$\frac{\text{Avg squared projection error}}{\text{Variation in data}} \leq 0.05$$

"95% variance retained"

$$\frac{\text{Avg squared projection error}}{\text{Variation in data}} = \frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2}$$

Variance retained tends to be chosen to be 95% - 99% usually

This eqn involves k, where by reducing k the variance retained may drop so we must find such a value of k where variance retained remains in the 95% - 99% range

↳ Even for this value, k is significantly reduced as most real life features tend to be highly correlated

Date:

Implementation of algo:

Inefficient version:

~~Try PCA~~

Try PCA with $k=1$ & onwards until (1) becomes true

Compute U_{reduce}, $z^{(1)}, z^{(2)}, \ldots z^{(m)}$, $x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

Check if:

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

chooses the smallest value of K where condition (1) becomes true

Efficient version:

$[U, S, V] = svd(Sigma)$

find the smallest k that satisfies (2)

For given k:

$$\text{(2)} \quad 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

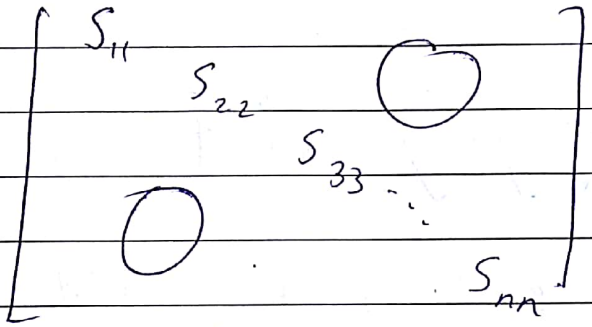
mathematically eqv. to (1)

Victory

Date:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

$\rightarrow S =$ 

Only diagonal in this matrix has non-zero values. All other values are zeros

values used from here

In the inefficient implementation:

1) U_{reduce} needs to be defined iteratively
2) $z^{(1)}, \ldots, z^{(m)}$ must be calculated iteratively
3) $x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$ must be calculated iteratively
4) Formula (1) evaluated

↳ all this creates a large overhead

Date:

If you simply run:

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$$

this will give the % of variance that will be retained

↳ tells how well the new data set will approximate the original data set

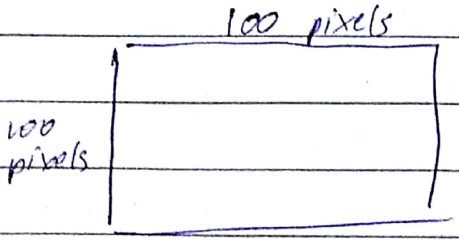
Advice for Applying PCA

Supervised learning speed up with PCA

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

Suppose $x^{(i)}$ is in high dimension

↳ such as in computer vision



image

= 10'000 pixels / features of input

Having so many features can cause algo to run slowly

Date:

Extract the inputs:

Unlabeled dataset: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

PCA

$z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^{1000}$

New training set:

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

Now feed this new training set into a learning algorithm such as logistic regression:

$$h_\theta(z) = \frac{1}{1 + e^{-\theta^T z}}$$

PCA

$z \leftarrow x$

Mapping $x^{(i)} \rightarrow z^{(i)}$ on training set creates some parameters that apply only to the training set, not any other data set:

- feature normalization (making mean = 0)
- feature scaling
- U_{reduce}

none of these values obtained from training set can be applied to CV or test set

"not sure about these"

Date:

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 $x_{cv}^{(i)}$ & $x_{test}^{(i)}$ in the ~~test~~ CV & test sets

Learning algo can run much faster now due to reduced dimensionality.

Application of PCA

- Compression:
  - reduce memory / disk needed to store data
  - speed up the learning algo ← usually 99% of variance is retained by users for this purpose
  - ↳ choose k by % of variance retained
- Visualisation
  - ↳ k is chosen to be 2 or 3 as only 2D or 3D can be plotted

% of variance retained – measure of how much info is thrown away when finding lower dimensional representation.

Date:

Bad use of PCA: To prevent overfitting

Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the # of features to $k < n$

(suppose $k = 1000$, $n = 10'000$)

Thus, fewer features, less likely to overfit

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead:

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Reason for not choosing PCA for overfitting:

When finding lower-dimensional approximation, PCA throws away some info without knowing the values of y

↳ okay if 99% variance is retained but not okay if some valuable info is thrown away

Regularization will work better to prevent overfitting because minimization problem knows values of y hence less likely to throw away valuable info.

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x^{(i)}_{test}$ to $z^{(i)}_{test}$. Run $h_\theta(z)$ on:

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

- How about doing the whole thing without using PCA?
- Before implementing PCA, first try running whatever you want to do with the original / raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA & consider using $z^{(i)}$

Date:

Run PCA if your learning algo ends up running too slowly, or requirement for memory is too large