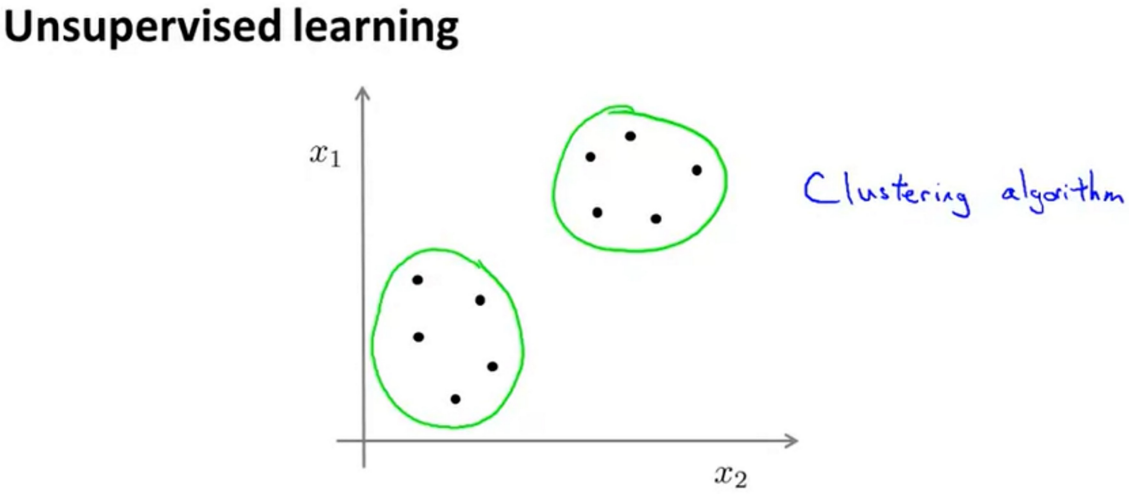
Week 8

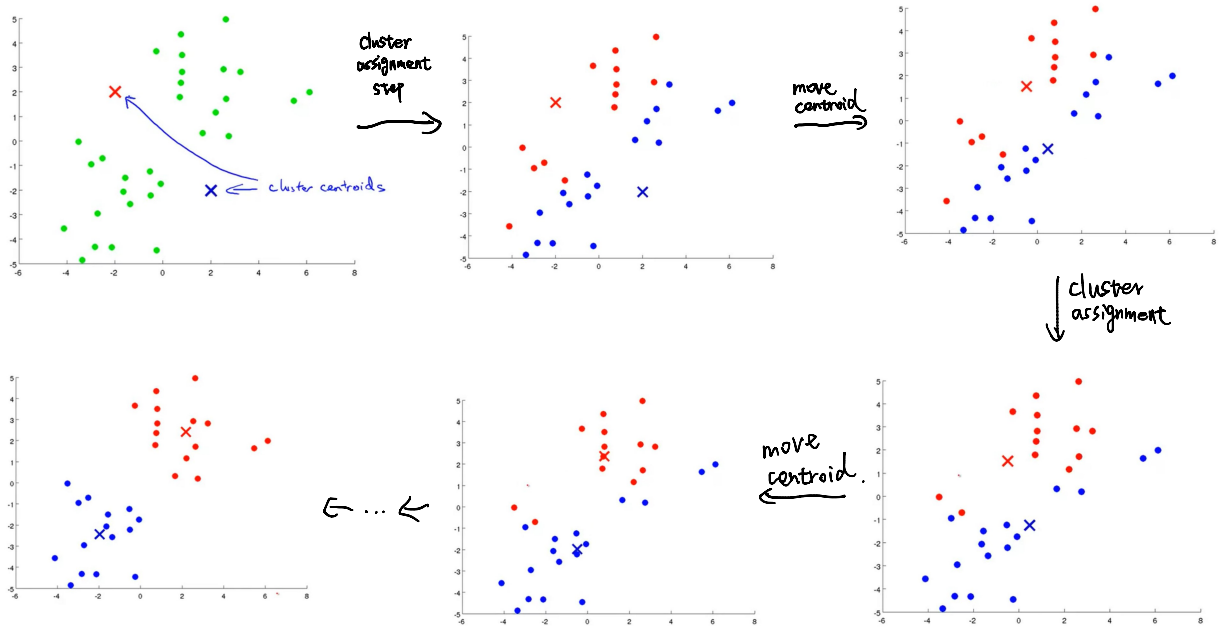
- Clustering:
  a. Unsupervised Learning:Introduction:
  No label!!!



  b. K-Means Algorithm:

The first step is to randomly initialize two points, called the cluster centroids.

Cluster assignment step: going through each of the examples and depending on whether it's closer to which cluster centroid, it is going to assign each data points to one of the two cluster centroids.

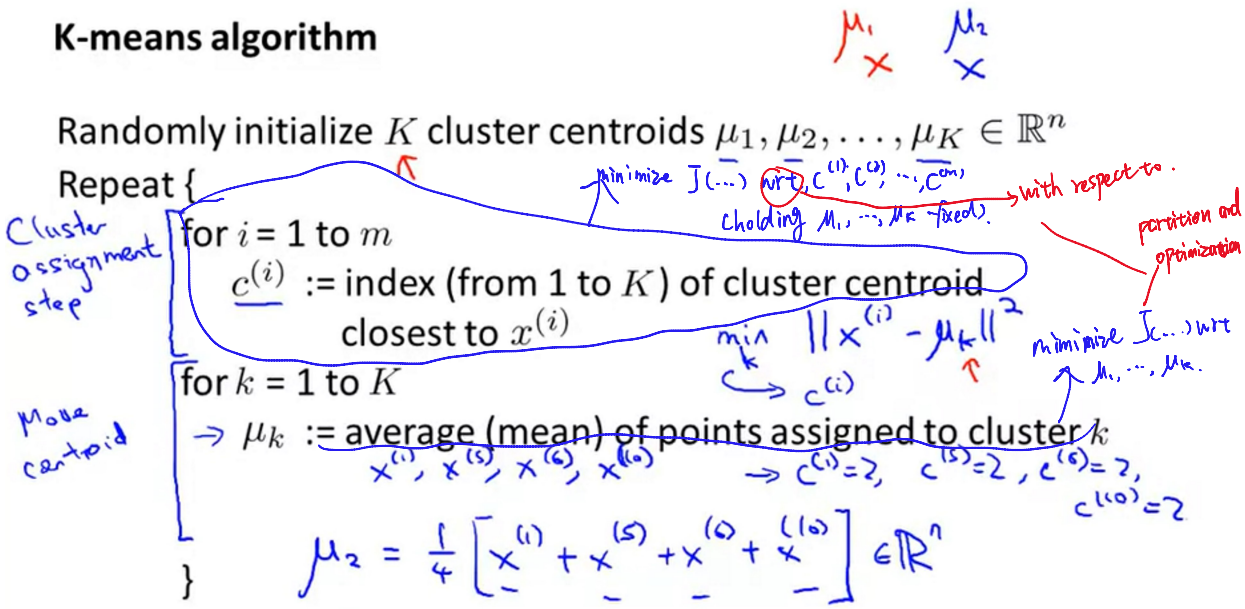


Move centroid step: moving each of the centroid to the

average of the points colored the same colour.

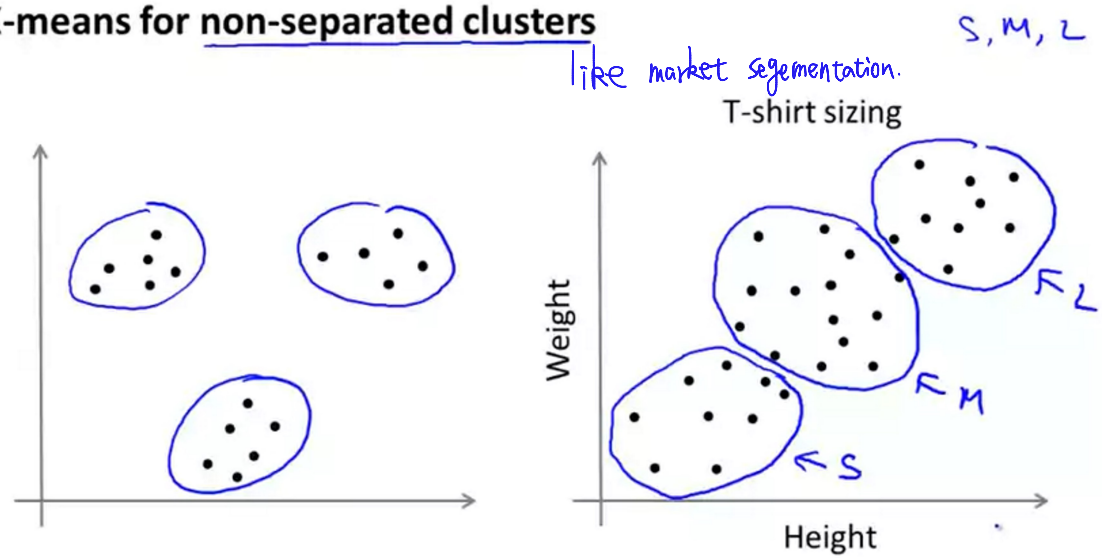
Input:K(number of clusters), training set{$x^{(1)}$,$x^{(2)}$,…,$x^{(m)}$}

K-means algorithm



if there is a cluster centroid no points with zero points assigned to it. In that case the more common thing to do is to just eliminate that cluster centroid.

K-means for non-separated clusters



c. Optimization Intialization:

Knowing what is the optimization objective of k-means will help us to debug the learning algorithm and just make sure that k-means is running correctly.

Help k-means find better clusters and avoid the local optima.

K-means optimization objective

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

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

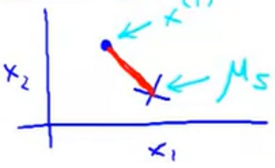
$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Optimization objective:

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

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

Distortion



d. Random Intialization:

Random initialization

Should have $K < m$

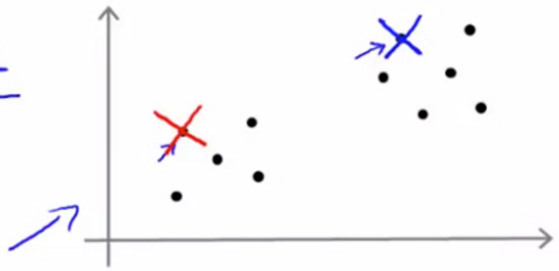
Randomly pick K training examples.

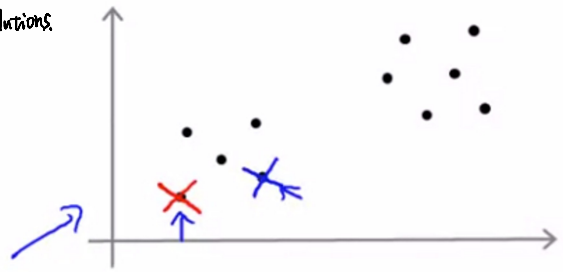
Set $\mu_1, \dots, \mu_K$ equal to these K examples.

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

$\mu_2 = x^{(j)}$

different initializations can result in different solutions





For i = 1 to 100 {

50 - 1000

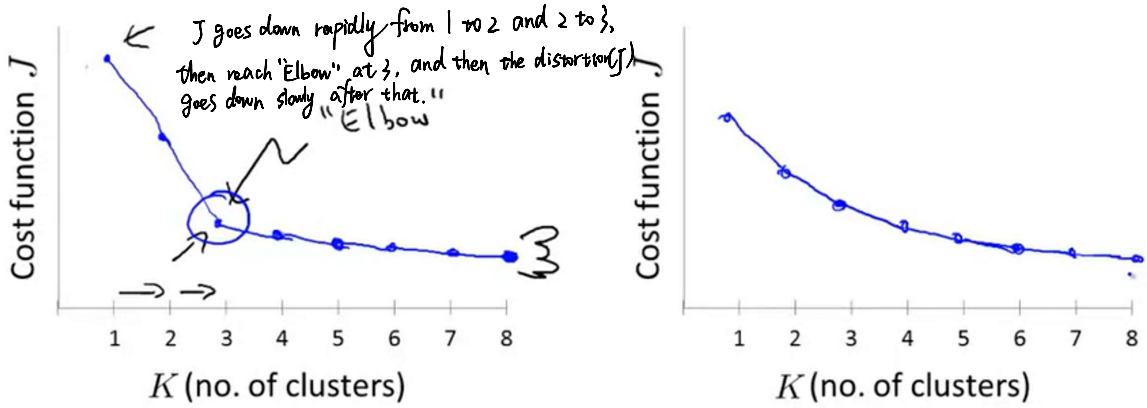
→ Randomly initialize K-means.
Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$.
Compute cost function (distortion)
→ $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$
}

Pick clustering that gave lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

K = 2 - 10

e. Choosing the Number of Clusters:

Elbow method:



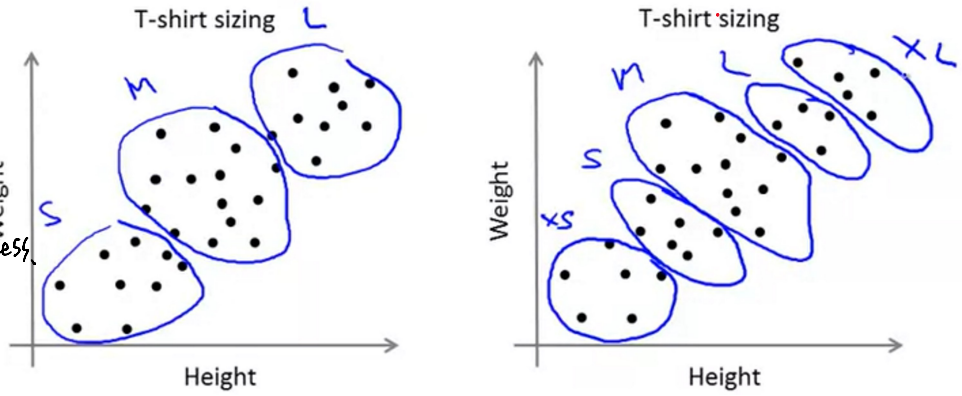
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

K=3 S, M, L

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

E.g.

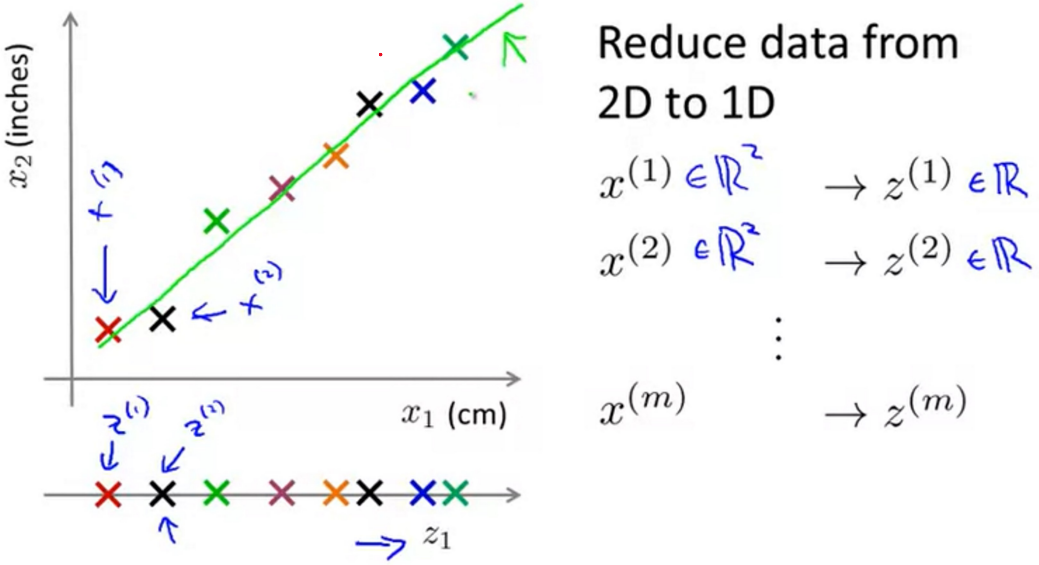
think about this from the perspective of the T-shirt business



Dimensionality Reduction

- Motivation:
  a. Motivation I:Data Compression:

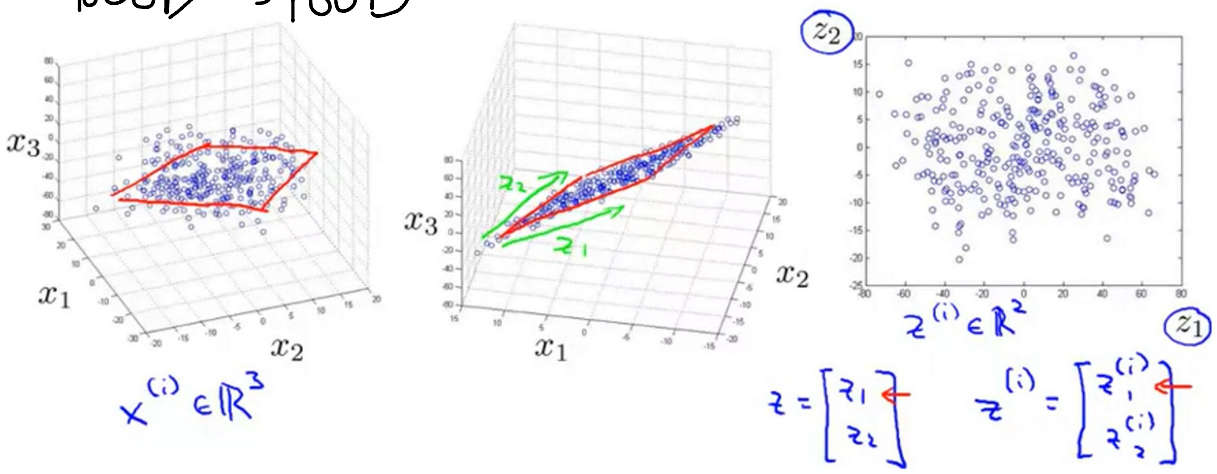
     it says z_1 is a new feature that specifies the location of each of those points on this green line



So this is an approximation to the original training self because I have projected all of my training examples onto a line.

Reduce data from 3D to 2D

1000D → 100D



b. Motivation II:Visualization:

$x \in \mathbb{R}^{50}$ $x^{(i)} \in \mathbb{R}^{50}$

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
⇒Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

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

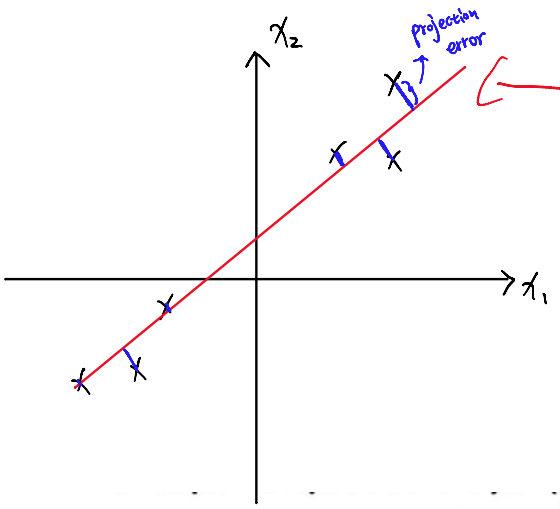
Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

It doesn't represent a physical meaning.

Reduce data from 50D to 2D

- Principal Component Analysis:

a. Principal Component Analysis Problem Formulation:



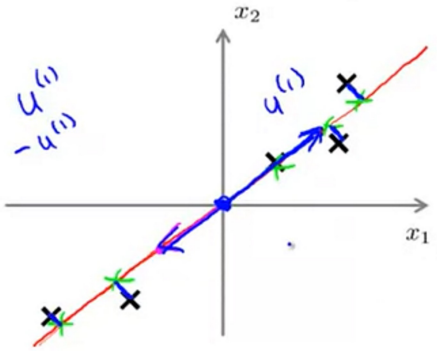
what PCA does?

It tries to find a lower dimensional surface, project the data onto the red line. So that the sum of squares of these little blue line segment is minimized.

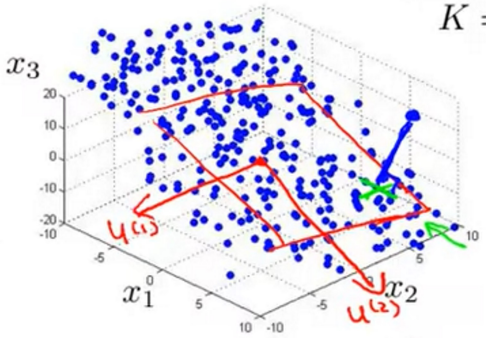
projection error.

before that mean normalization and feature scaling is important.

project the data onto the linear subspace spanned by the set of k vectors.



$3D \to 2D$
$K = 2$

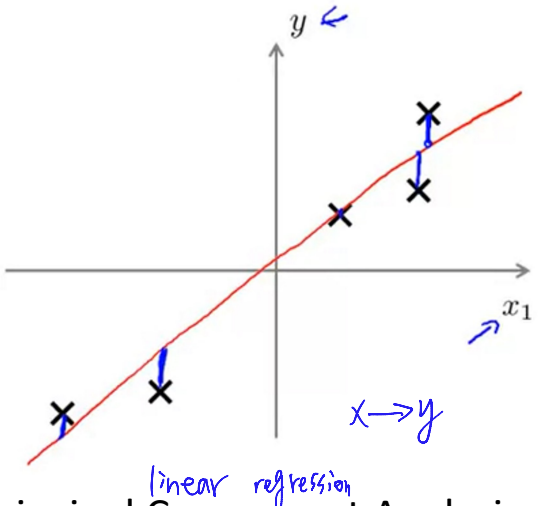


Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

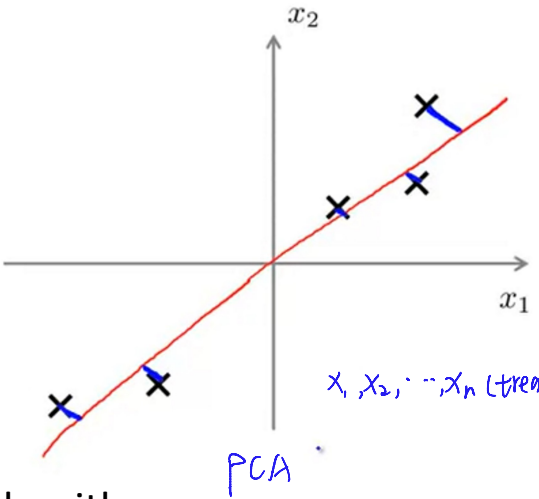
a set of vectors.

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.

PCA is not linear regression



linear regression



$x_1, x_2, \cdots, x_n$ (treat equally).

PCA

b. Principal Component Analysis Algorithm:

Data preprocessing

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

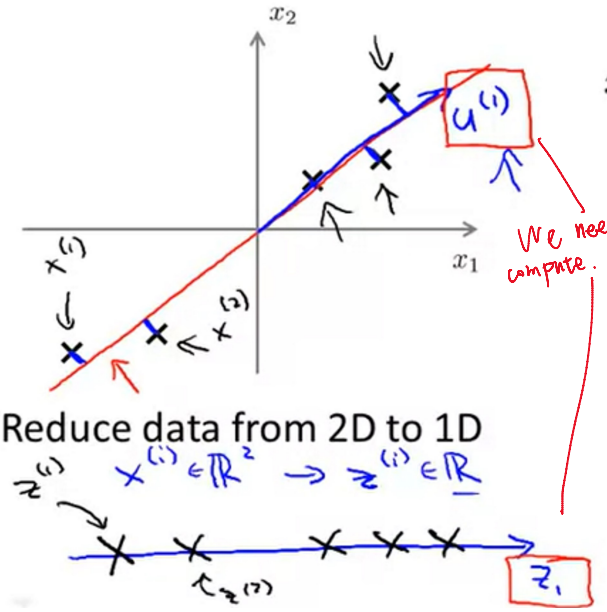
Preprocessing (feature scaling/mean normalization):

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

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

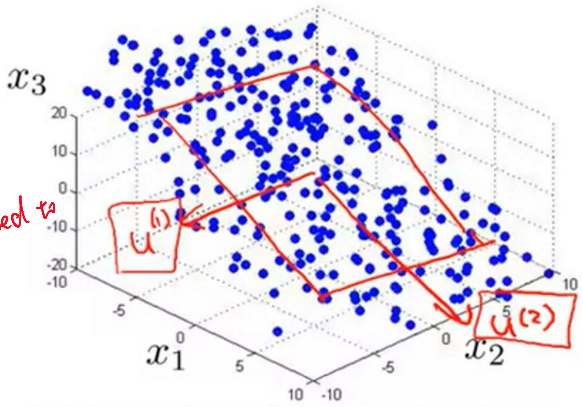
If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

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



Reduce data from 2D to 1D

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



Reduce data from 3D to 2D

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

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

Sigma

Compute "eigenvectors" of matrix Σ:

```
[U,S,V] = svd(Sigma);
```

n×n matrix.

$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix}$ (first k columns)

$U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \ldots, u^{(k)}$.

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

what we are going to take first k columns from U matrix.

$z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x = \begin{bmatrix} -(u^{(1)})^T- \\ \vdots \\ -(u^{(k)})^T- \end{bmatrix} x$

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

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

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

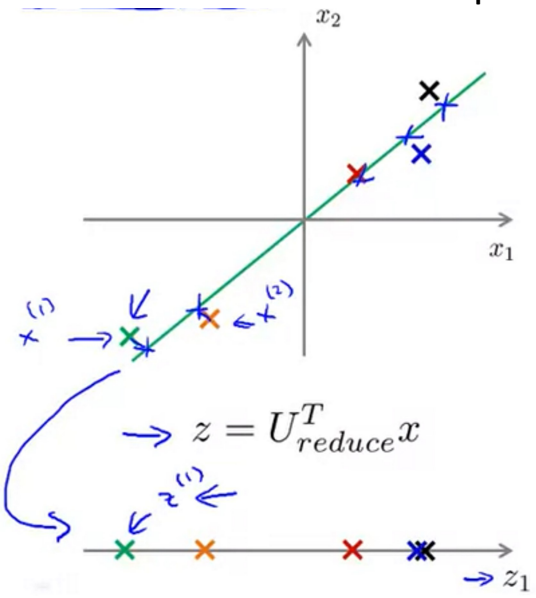
$X = \begin{bmatrix} -x^{(1)T}- \\ \vdots \\ -x^{(m)T}- \end{bmatrix}$

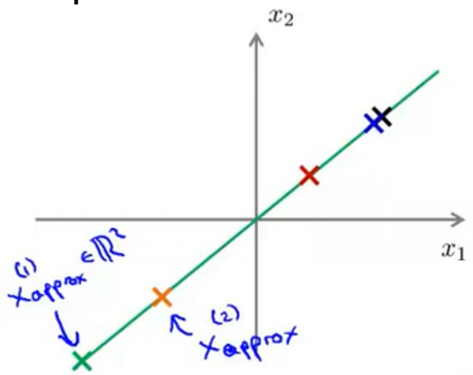
Sigma = (1/m) * X' * X;

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

$x \in \mathbb{R}^n$ $\quad x_0 = 1$

- Applying PCA:
  a. Reconstruction from Compressed Representation:





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

$x_{approx}^{(1)} = U_{reduce} \cdot z^{(1)}$

$\mathbb{R}^n$ $\quad n \times k$ $\quad k \times 1$ $\quad n \times 1$

  b. Choosing the Number of Principal Components:

Choosing k (number of principal components)

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

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:

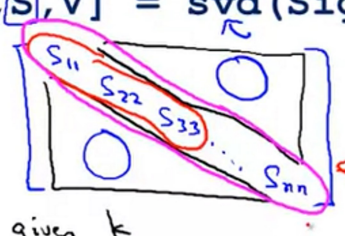
Try PCA with $k = 1$

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

Check if

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

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



For given k

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

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

Pick the smallest value of k for which :

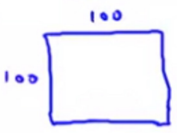
$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

c. Advice for Applying PCA:

Supervised learning speedup

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

$x^{(i)} \in \mathbb{R}^{10,000}$



Extract inputs:

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

$\downarrow PCA$

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

U_{reduce}

New training set:

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

to fit model

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

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.

$x \rightarrow z$

Applicant of PCA

- compression
  - Reduce memory/disk needed to store data.
  - Speed up learning algorithm.

choose k by 99% of variance retain

- Visualization.

→ k=2 or 3 for visualization.

PCA throws away or reduces the dimension of data without knowing what the values of y is.

Bad use of PCA: To prevent overfitting

→ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$. (1000 < 10000)
Thus, fewer features, less likely to overfit.

Bad!

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

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

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 $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \ldots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

→ 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 and consider using $z^{(i)}$.

learning algorithm ends up running too slowly or the memory or disk requirement is too large.