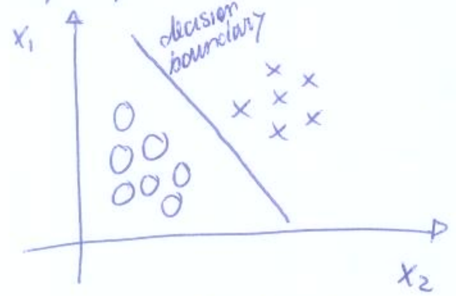
Introduction to unsupervised learning

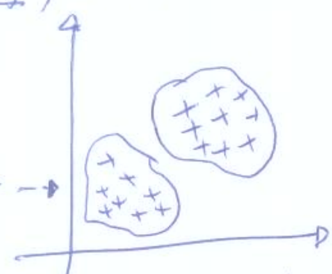
- Learning from unlabeled data.

Recall our typical examples for problems with supervised learning



we usually had a training set like this $\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\dots,(x^{(m)},y^{(m)})\}$.
Our objective was to find a decision boundary separating O and x.

In unsupervised learning, all we have is a training set of $\{x^{(1)},x^{(2)},\dots,x^{(m)}\}$ with no labels



Then we ~~devlop~~ develop algorithms to find structures in the data. The simplest structure is a group. Therefore we have many algorithms to find ~~structures~~ groups or clusters in our data.

Clustering has several applications such as

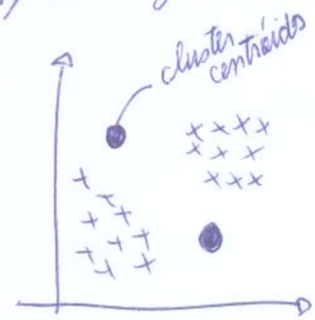
1. Image Retrieval / Info Retrieval.
2. Image classification using visual dictionaries.
3. market segmentation (find groups of similar customers).
4. social networks (find groups of people with similar interests, coherent groups of friends, etc)
5. Organization of datacenters (how to connect computers faster to attend demand, etc)
6. Astronomical data analysis (understanding galaxy formations).

K - means algorithm

→ most popular and widely used algorithm to date.

Suppose we want to group our data into two clusters (groups).

①

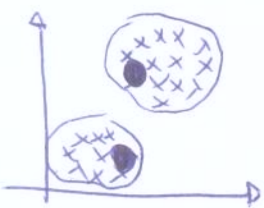


②
Randomly initialize clusters.

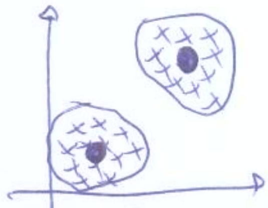
③ cluster assignment

* K-means has two main steps
Iterative algorithm.

① cluster assignment step
② ~~s~~ move centroids step



④
move centroids and new assignments



⑤
move centroids and new assignments

More formally: the k-means algorithm takes as input:

- K (number of clusters). → for now, just assume ~~you~~ we know this by construction.
- Training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$.
- $x^{(i)} \in \mathbb{R}^n$ ($x_0 = 1$ by convention) used before $\boxed{\text{is not}}$ used here).
-

K-means algorithm

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

2. Repeat {

cluster assignments:
for $i \leftarrow 1$ to m {
  $c^{(i)} \leftarrow$ index (from 1 to K) of cluster centroid closest to $x^{(i)}$. $\rightarrow \min_{k} \| x^{(i)} - \mu_k \|^2$ (norm)
}

the 2 (square) here is by convention only

centroid move:
for $k \leftarrow 1$ to K { (little k, big K)
  $\mu_k \leftarrow$ mean of points assigned to cluster k.
}

}

For instance, suppose μ_2 is the currently the center of cluster 2 with the points just assigned $x^{(1)}, x^{(2)}, x^{(3)}$. Then the new centroid is going to be $\mu_2 = \dfrac{\sum_{i=1}^{3} x^{(i)}}{3} = \dfrac{x^{(1)} + x^{(2)} + x^{(3)}}{3}$;

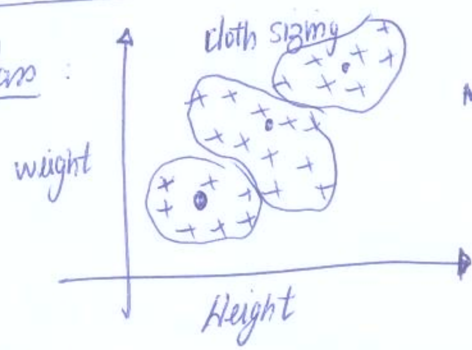
$\mu_2 \in \mathbb{R}^n$ just like before.

$\mu_2 = \dfrac{\text{~~sum~~ sum over members of 2}}{\text{size of 2}}$.

<u>Question</u>: what if a cluster doesn't have any element?
- ↳ Eliminate such cluster reducing the number of clusters from K to K-1.
- ↳ Another solution without eliminating the cluster is just reinitialize ~~such~~ this cluster's centroid.

What happens when we don't have very well separated clusters?

Example from Andrew Ng's class:



~~market~~ market segmentation Example.

suppose we want 3 sizes S, M, and L

Optimization ~~obj~~ objective

- for all algorithms we studied so far, ~~except genes~~ we had an optimization function or cost function even genetic algorithms have it in order to measure adaptation and fitness.

- K-means also has a cost function.

K-means optimization objective

$c^{(i)}$ = index of clusters $(1, 2, \ldots, k)$ to which example $x^{(i)}$ is currently assigned

μ_K = cluster centroid k $(\mu_K \in \mathbb{R}^n)$. } K big number of clusters an k little is an index.

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

For instance, suppose $x^{(i)}$ was assigned to cluster 5. It means then ~~them~~ that $c^{(i)} = 5$. Finally, $\boxed{\mu_{c^{(i)}} = \mu_5}$

Optimization objective:

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\|x^{(i)} - \mu_{c^{(i)}}\|^2}_{\text{square distance between } x^{(i)} \text{ and its cluster center/centroid.}}$$

therefore we want

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

- This cost function sometimes is called Distortion

If we recall our K-means algorithm, we have that:

1. step of cluster assignment minimizes $J(\ldots)$ keeping $\mu_1, \ldots, \mu_K$ fixed. Finds $c^{(1)}, \ldots, c^{(K)}$
2. step of move centroid minimizes $J(\ldots)$ keeping $c^{(i)}$ fixed. Finds $\mu_1, \ldots, \mu_K$.

We can $\boxed{\text{use}}$ this cost function to find better clusters and avoid some local minima.

Random Initialization

Recalling the algorithm:

1. **Randomly** initialize $\underline{K}$ cluster centroids $\mu_1, \ldots, \mu_K \in \mathbb{R}^n$
2. Repeat {

    for $i \leftarrow 1$ to m

    $\quad c^{(i)} \leftarrow$ index (from 1 to $\underline{K}$) of cluster centroid closest to $x^{(i)}$

    for $k \leftarrow 1$ to $\underline{K}$

    $\quad \mu_k \leftarrow$ mean of points assigned to cluster k.

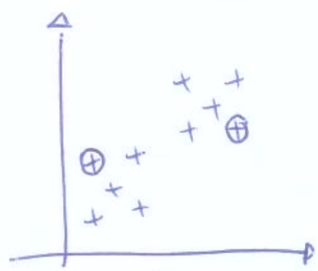
    }

How to do this Random ?

1. Should have $\underline{K} < m$.
2. Randomly pick $\underline{K}$ training examples
3. set $\mu_1, \ldots, \mu_{\underline{K}}$ equal to these $\underline{K}$ examples

Example $\underline{K} = 2$.

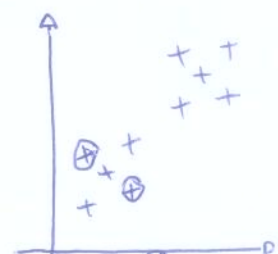
1



2a

Pick 2 examples for centroids (lucky choices!)

Another example

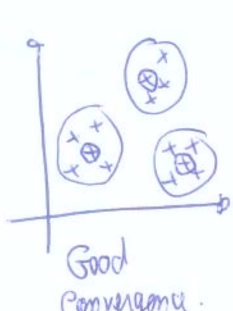


2b

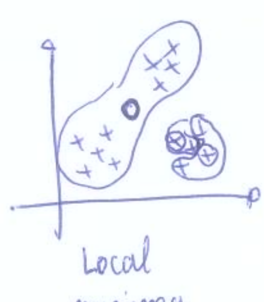
Pick 2 examples for centroids (less lucky in this case!)

$$\begin{cases} \mu_1 = x^{(i)} \\ \mu_2 = x^{(j)} \\ \vdots \\ \mu_{\underline{K}} = x^{(k)} \end{cases}$$

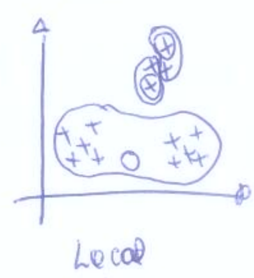
One thing is obvious : depending on the initialization K-means will converge differently. In ~~particle~~ particular, it can converge to local optima.



Good Convergence.



Local minima



Local minima.

To avoid such cases or to diminish the chances of that happening, we can run K-means different times with different initializations.

Random initialization{

for i=1 to 1000{

Randomly initialize K-means.

Run K-means.

Get $c^{(1)}, \dots, c^{(m)}; \mu_1, \dots, \mu_{\underline{K}}$

Compute cost function (distortion) $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_{\underline{K}})$.

}

Pick clustering that gave lowest cost $J(\dots)$.

}

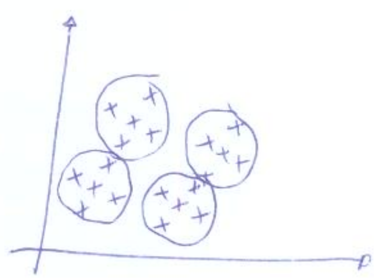
with $\overbrace{\text{many}}^{>10 \text{ for instance}}$ clusters this doesn't mean that much but with a $\overbrace{\text{few clusters}}^{<10}$ only, it may help substantially.

Alright, but how to choose K the number of clusters?

1. The most common answer is manually through experience, using visualization techniques, analyzing the outputs of clustering, etc.

The main problem <u>here</u> is <u>ambiguity</u>

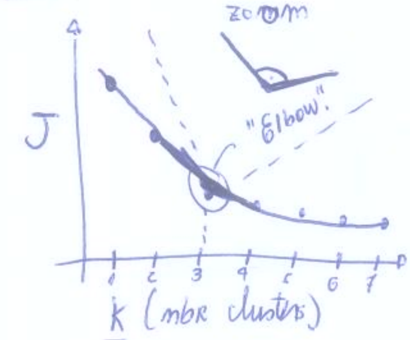
<u>vs</u>



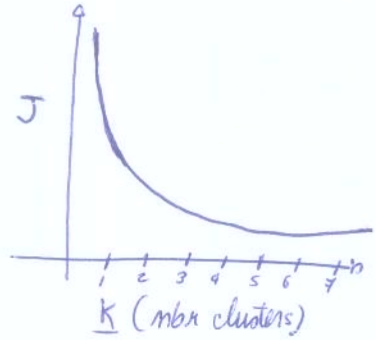
And this <u>spirit</u> of ambiguity is common in SL since we don't have <u>labels</u> to rely upon.

In spite of this natural ambiguity, we can use a method known as the

Elbow method



Distortion goes rapidly down until 3 and start stabilizing.



however, sometimes it is ambiguous.

Do not have high expectations with this method.

Another way of choosing K.

Sometimes we use K-means for a later purpose. Evaluate K-means based on a metric for how well it performs for that later purpose. E.g.: image classification using visual dictionaries (choose a dictionary size and check classif. accuracy on a set of examples).

Other examples:
- image segmentation: choose K, cluster, segment image, check results.
- image compression: choose K, compress image, measure quality.

Observations.

① There are many other clustering methods such as
- K-medoids clustering
- hierarchical clustering
- agglomerative clustering
- etc.

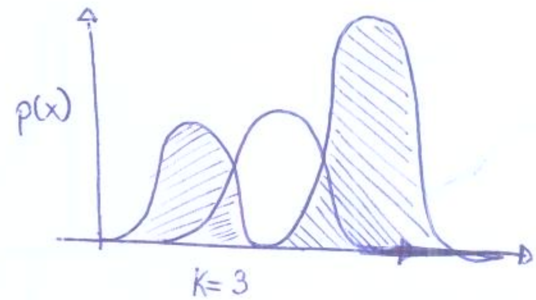
② In K-means we had what we call a hard assignment policy: each data point $x^{(i)}$ can belong to only one cluster. What about relaxing this constraint? Sometimes, for vectors near the boundary the hard assignment policy may be a poor choice.

③ Let's now consider a soft-assignment policy where the strength of the assignment depends on the distance.

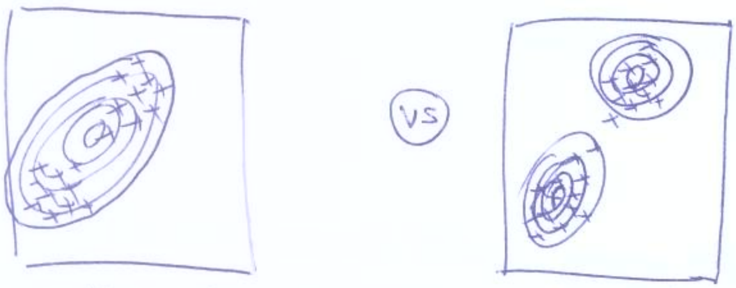
Gaussian mixture model (GMM)

Combine simple models into a complex model

$$p(x) = \sum_{k=1}^{K} \underbrace{\pi_k}_{\text{mixing coefficient}} \underbrace{\mathcal{N}(x \mid \mu_k, \Sigma_k)}_{\text{component}}$$



$$\forall k: \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1.$$



single Gaussian

Mixture of two Gaussians

Cost function for fitting a GMM

centroids, covariances → as we model as Gaussians we need the σs.

for a point $x^{(i)}$, $\quad p(x^{(i)}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(i)} \mid \vec{\mu}_k, \Sigma_k)$

The $\boxed{\text{likelihood}}$ of the GMM for M points (assuming independence) is

$$\prod_{i=1}^{N} p(x^{(i)}) = \prod_{i=1}^{M} \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(i)} \mid \vec{\mu}_k, \Sigma_k).$$

we can simplify this in order to optimize by taking the (negative) log likelihood

$$\mathcal{L}(\theta) = -\sum_{i=1}^{m} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(i)} / \vec{\mu}_k, \Sigma_k).$$

where θ are the parameters we wish to estimate ($\vec{\mu}_k$ and Σ_k).

$\mathcal{L}(\theta)$ is our $J(\theta)$ here written as $\mathcal{L}(\theta)$ to remember its association with the term log-likelihood.

To minimize $L(\theta)$ or $J(\theta)$, we differentiate w.r.t. μ_k

$$\frac{\partial \sigma(\theta)}{\partial \mu_k} = \sum_{i=1}^{m} \underbrace{\frac{\pi_k \cdot \mathcal{N}(x^{(i)} \mid \vec{\mu}_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(x^{(i)} \mid \vec{\mu}_j, \Sigma_j)}}_{\gamma_{ik}} \underbrace{\Sigma_k^{-1}}_{\text{covariance}} (x^{(i)} - \vec{\mu}_k)$$

Rearranging

$$\sum_{i=1}^{m} \gamma_{ik} \vec{\mu}_k = \sum_{i=1}^{m} \gamma_{ik} x^{(i)}.$$

therefore $\vec{\mu}_k = \frac{1}{\underset{\text{examples in group } k}{M_k}} \sum_{i=1}^{M} \gamma_{ik} \cdot X^{(i)}$ // weighted mean

where

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(\vec{X}^{(i)} \mid \vec{\mu}_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(x^{(i)} \mid \vec{\mu}_j, \Sigma_j)}, \quad M_k = \sum_{i=1}^{m} \gamma_{ik}.$$

- and γ_{ik} are called responsibilities of mixture component k for vector $\vec{x}^{(i)}$.
  M_k is the effective number of vectors assigned to group k.

- γ_{ik} play a similar role to the assignment variables 1 or 0 multiplying a group in hard K-means. There we could think of γ_{ik} as 1 if a datapoint is in group k and zero to the other groups.

Differentiating Σ_k gives

$$\Sigma_k = \frac{1}{M_k} \cdot \sum_{i=1}^{M} \gamma_{ik} (\vec{x}^{(i)} - \vec{\mu}_k)(\vec{x}^{(i)} - \vec{\mu}_k)^T \quad \text{weighted covariance}$$

and w.r.t. π_k enforcing the constraint $\underbrace{\sum_k}_{\text{sum}} \pi_k = 1$ gives $\pi_k = \frac{M_k}{M}$.

which is the average responsibility for the component.

Expectation maximization (EM) algorithm

Step 1 : Expectation

compute responsibilities using current parameters $\vec{\mu}_K, \Sigma_K$ (assignment)

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}\left(\vec{x}^{(i)} \mid \vec{\mu}_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}\left(x^{(i)} \mid \vec{\mu}_j, \Sigma_j\right)}$$

nbr groups

Step 2 : Maximization : Re-estimate parameters using computed responsibilities

$$\vec{\mu}_K = \frac{1}{M_K} \sum_{i=1}^{M} \gamma_{ik} \vec{x}^{(i)}$$

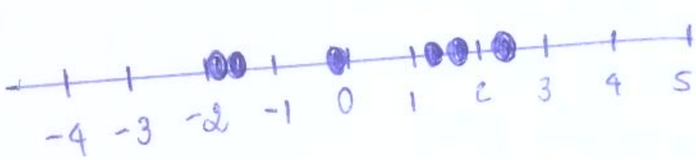
$$\Sigma_K = \frac{1}{M_K} \sum_{i=1}^{M} \gamma_{ik} \left(\vec{x}^{(i)} - \vec{\mu}_K\right) \cdot \left(\vec{x}^{(i)} - \vec{\mu}_K\right)^T$$

$$\pi_K = \frac{M_K}{M} \text{ where } M_K = \sum_{i=1}^{M} \gamma_{ik}.$$

Repeat until convergence.

Example in 1D :

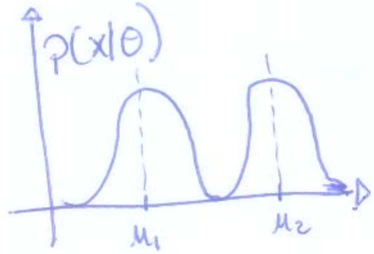
Data : $x = (x_1, x_2, \dots, x_M)$.



Objective : fit GMM with $k = 2$ components

Model : $p(x_i \mid \Theta) = \sum_K \pi_K \cdot \mathcal{N}(x_i \mid \vec{\mu}_K, \sigma_K)$ where $\sum_{k=1}^{K^{3a}} \pi_k = 1$

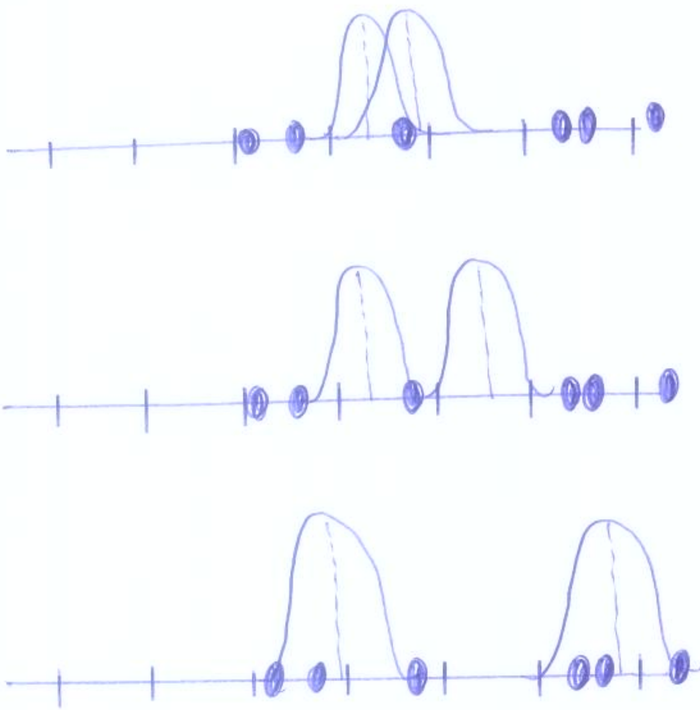
Parameters : $\Theta = \{\pi, \mu, \sigma\}$. Keep π, σ fixed and estimate μ.



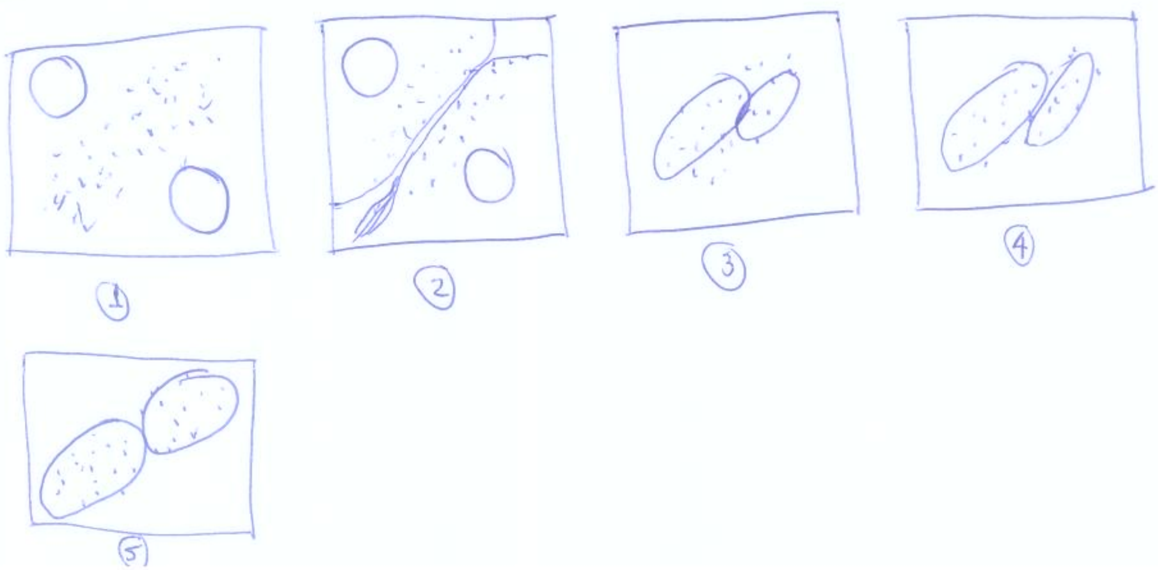
Intuition of EM:

E-step: compute soft assignment of the points using current θ.

M-step: update parameters θ using current responsibilities.



2-D example



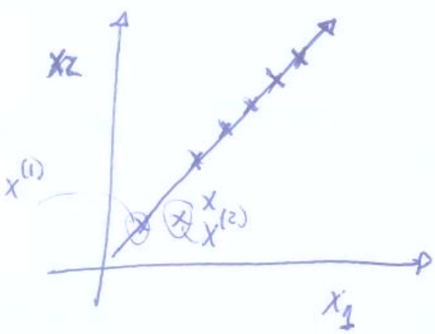
Practical Aspects :

1. Normally we ~~o~~ use K-means to initialize EM algorithms
2. Choice of K
3. Also can converge to local min.

Dimensionality Reduction

Two main reasons:
- Data compression
  - for speeding up learning algorithms
  - compress data for memory efficiency purposes.
- Visualization

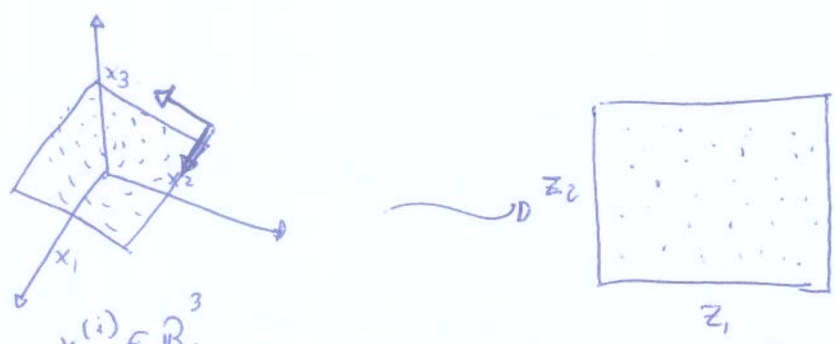
Suppose we have a dataset with <u>many</u> features (here are two)



- Reduce data from 2D to 1D.
- Data may be redundant (cm x meters x inches from different sources).

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

Data compression 3D → 2D



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

Data visualization

Suppose we have a dataset such that each $x^{(i)} \in \mathbb{R}^{100}$ and we want to quickly visualize some of its <u>properties</u>:

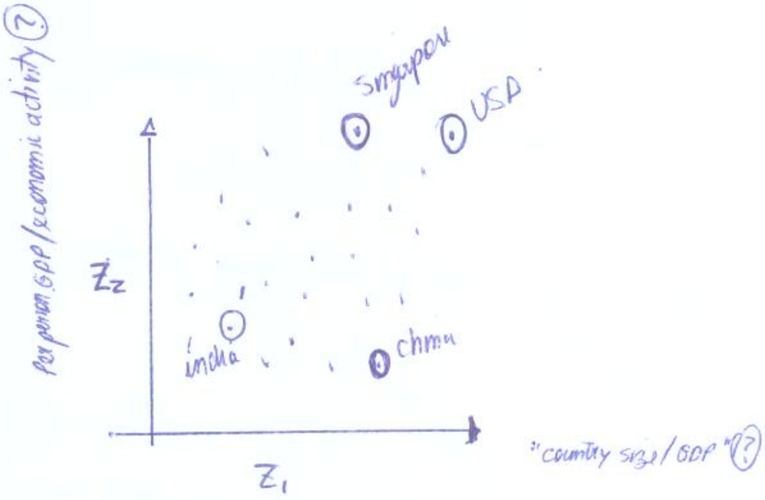
pais	GDP(tri) \$	GDP (\$) Per capita	...	GINI
canada	1.6	39.2		32
china	5.9	7.5		46
⋮			⋮	
singapore	0.22	56.7		42
USD	14.5	46.8		40

$$x^{(i)} \in \mathbb{R}^{100} \to z^{(i)} \in \mathbb{R}^2.$$

* In $\mathbb{R}^2$ it is up to use to understand what the data is telling us (what each $z^{(i)}$ means).

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

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
⋮		
Singapore	0.5	1.7
USA	2	1.5



Before going into details about Dimensionality Reduction, let's see why it is <u>important</u>.

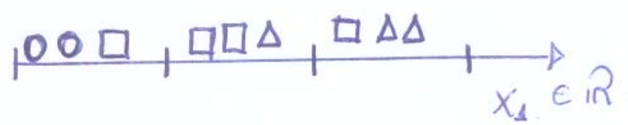
<u>The curse of dimensionality</u>

1. Term coined by Bellman in 1961.
2. Refers to the problem associated with multivariate data analysis as the dimensionality increases.

<u>Consider a simple example</u>: a 3-class pattern recognition problem.

1. A simple approach would be divide the feature space into uniform <u>bins</u>.
2. Compute the ratio of examples for each class at each bin and,
3. for a new input, find its bin and choose the predominant class in that bin

In our toy problem, we decide to start with one single feature and divide the <u>real line</u> into ③ segments



→ After doing this, we notice there exists <u>overlap</u> among the <u>classes</u>. The solution?
Increase complexity / add more features. Let's start in $\mathbb{R}^2$.

We decide to preserve the granularity of each axis. Hence the number of bins goes from 3 in $\mathbb{R}$ (1D) to $3^2 = 9$ in $\mathbb{R}^2$ (2D).

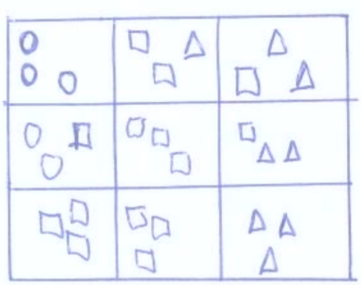
Now we need to decide → maintain the density of examples per bin (A)
or
→ Keep the number of examples we had for (1D). (B)

In case of (A)

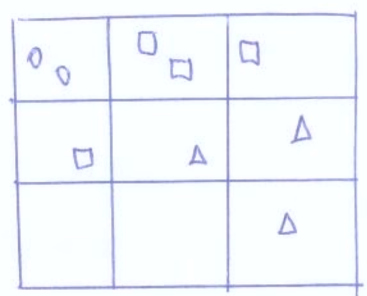
Increases the number of examples from $\underbrace{m = 9}_{1D}$ to $m = 27$ (2D)

In case of (B)

Results in a very sparse 2D slatter plot.



Cte density



Const. # examples

what about moving to $\mathbb{R}^3$?

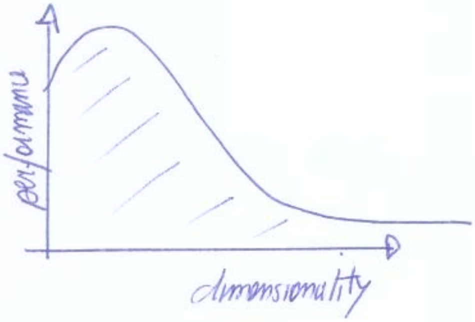
1. number of bins → $3^3 = 27$.
2. cte density → 81 needed examples
3. for the same number of examples (cte # examples), 3D feature space is almost empty.

* Our approach to divide the sample space is naive but even with more interesting solutions, the sparsity problem still exists.

* How to deal with the curse of dimensionality?

1. By incorporating prior knowledge
2. By providing increasing smoothness of the objective function (e.g. regularization)
3. By reducing dimensionality.

In practice, <u>Curse of dim</u> means there is, for a given sample size (dataset of a given size) a max # of features above which the performance of a classifier will degrade instead of improve.



<u>Implications of the curse of dimensionality</u>

1. Exponential growth in # of examples required to maintain a given sampling density. for M examples / bin and D dimensions, # of examples is M^D !

2. Exponential growth in the complexity of the <u>objective function</u> (a density estimate) with increasing dimensionality.
   - To learn well, a more complex target function requires denser sample points!
   - "A function in high-dimensional space is likely to be more complex than one in lower dimensional spaces, and those complications are harder to discern" <u>Friedman</u>.

3. What if it's not a <u>Gaussian</u>? For 1-D, we have several density functions. For $\mathbb{R}^n$ dim spaces, only the multivariate Gaussian is <u>available</u>.

4. Humans → Good to spot patterns in 1, 2 and 3 dimensions → ~~Bad~~ Pretty bad when $n > 3$.

How to reduce dimensionality then?

① feature extraction: create a subset of new features by combining the existing ones

② feature selection: choosing a subset of all the features (the ones more informative)

$$\begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}^T \xrightarrow{\text{Selection}} \begin{bmatrix} x_{i}^{(i)} \\ x_{i+1}^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}^T \qquad \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}^T \xrightarrow{\text{Extraction}} \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \\ \vdots \\ z_d^{(i)} \end{bmatrix}^T = f\left(\begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}^T\right)$$

Let's focus on feature extraction:

① Given a feature space $x^{(i)} \in \mathbb{R}^m$, find a mapping $z = f(x): \mathbb{R}^m \to \mathbb{R}^d$ with $d < m$. such that the transformed vector $z^{(i)} \in \mathbb{R}^d$ preserves (most of) the information and/or structure in $\mathbb{R}^m$.

② An optimal mapping $z = f(x)$ will be the one that results in low distortion/error.

③ In general, the optimal mapping will be a non-linear function. However, there is no systematic way to generate non-linear functions.

④ For this reason, we often are limited with linear transforms $\boxed{\vec{z} = W \cdot \vec{x}}$

↳ ⓩ is a linear projection of ⓧ.

⑤ when the mapping is a non-linear function, the reduced space is called a $\boxed{\text{manifold}}$

$$\begin{bmatrix} x_1^{(i)} \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & & \\ w_{d1} & w_{d2} & & \cancel{w_{dN}} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

wdm

⑥ MLPs multi layer perceptrons can help with non-linear mappings.

(1) The selection of the feature extraction mapping $z = f(x)$ is guided by an objective function to be <u>minimized</u> or <u>maximized</u>.

(2) Depending on the criteria, feature extraction techniques can be of one out of two categories:

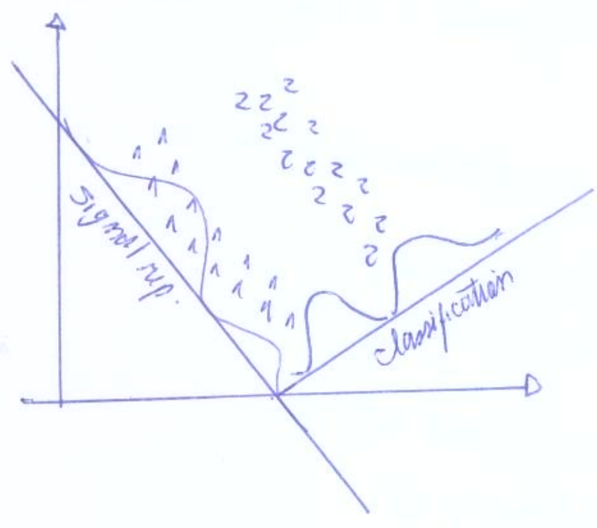
(a) <u>Signal representation</u>: objective is to represent samples accurately in a lower dimensional space

(b) <u>Classification</u>: ~~objecti~~ objective is to <u>enhance</u> class discrimination in the lower dimensional space.

<u>Two ~~ff techniques~~ techniques are highlighted</u>:

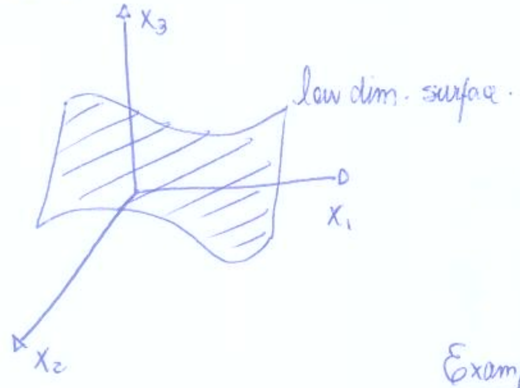
(1) Principal component analysis (example of (a)).

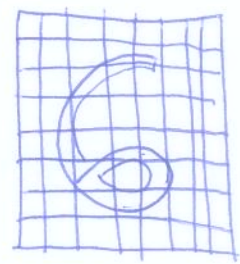
(2) Linear Discriminant Analysis (example of (b)).



Why reduce dimensionality - Example

① Intrinsic dimension of data: often data is measured in high dimensions but its actual variation lies on a low dimensional surface (plus noise).





Example: 64×64 sub-image $\to \{0, 1\}^{4096}$.

① there is irrelevant info/noise.

② feature extraction × selection: what would be better?

Projection to lower dimensions

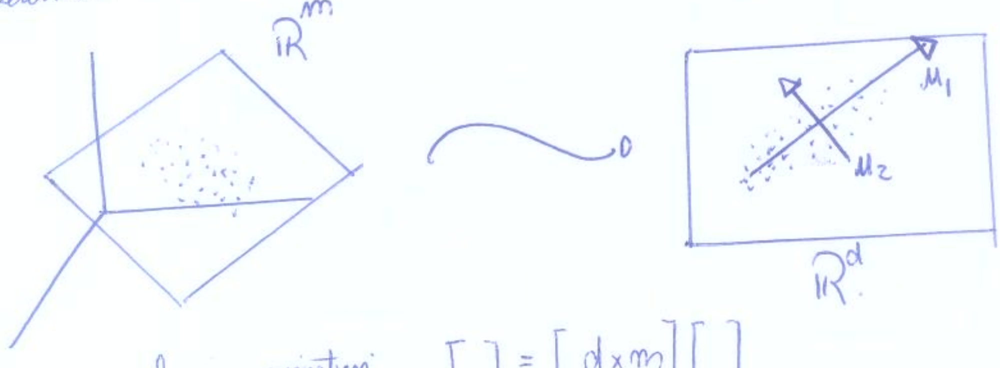
$$\mathbb{R}^m \rightsquigarrow \mathbb{R}^d \quad \text{where } m \gg d \text{ and often } d = 2 \text{ (e.g. when visualizing data).}$$

$$\Downarrow$$

$$\begin{bmatrix} \ \end{bmatrix} = \begin{bmatrix} d \times m \end{bmatrix} \begin{bmatrix} \ \end{bmatrix}.$$

Principal Component Analysis

① Determine a set of (orthogonal) axes which ~~best~~ best represent the data



a linear projection $\begin{bmatrix} \ \end{bmatrix} = \begin{bmatrix} d \times m \end{bmatrix} \begin{bmatrix} \ \end{bmatrix}$.

Steps determine a set of (orthogonal) axes which best represent the data

Step 1 : Compute the vector to the data centroid ($\vec{c}$)

Step 2: Compute the principal axes (u_i).

In more details

1. scale and normalise the data.

$$\vec{c} = \frac{1}{N} \sum_{i=1}^{M} x^{(i)}$$

and transform the data so that $\vec{c}$ becomes the new origin

$$\vec{x}^{(i)} \leftarrow \vec{x}^{(i)} - \vec{c}$$

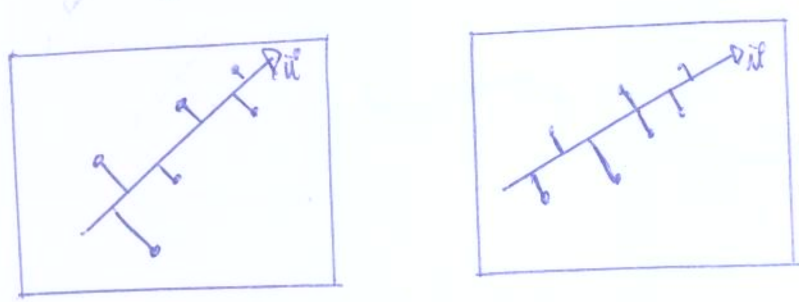
optionally we can scale (z-norm, max-min, etc)

$$x^{(i)} \leftarrow \frac{x^{(i)} - \vec{c}}{\sigma} \quad \text{or} \quad x^{(i)} \leftarrow \frac{x^{(i)} - \vec{c}}{\cancel{X}_{max} - X_{min}}$$

2. Compute the first principal axis: determine the direction that best explains the data. Find a direction (unit vector) $\vec{u}$ such that

$$\sum_i \left(x^{(i)} - (\vec{u}^T . x^{(i)})\vec{u}\right)^2 \text{ is minimized or}$$

$$\sum_i \left(u^T x^{(i)}\right)^2 \text{ is maximized. This is the direction of } \underline{\text{max}} \text{ variation.}$$



To enforce $\|\vec{u}\| = 1$, let's introduce a Lagrange multiplier and find the stationary point of

$$J = \sum_i \left(U^T x^{(i)}\right)^2 + \lambda\left(U^T U\right).$$

with respect to $\vec{u}$.

$$J = \sum_i U^T \left(x^{(i)} x^{(i)T}\right) u + \lambda\left(1 - U^T U\right).$$

$$= U^T \underbrace{\sum_i \left(x^{(i)} x^{(i)T}\right)} \cdot u + \lambda\left(1 - U^T U\right)$$

$$= U^T S \, U + \lambda\left(1 - U^T U\right)$$

Explanation

$$\left(A^T B\right)^2 = \left(A^T B\right) \cdot \left(A^T\right.$$
$$= A^T \left(B \cdot B^T\right)\left(A\right.$$
$$= A^T B B^T A$$

where S is the $m \times m$ simmetrix matrix $S = \sum_i x^{(i)} x^{(i)T}$

Then

$$\frac{\partial J}{\partial u} = 2S\vec{U} - 2\lambda\vec{U} \quad \text{and hence } S\vec{U} = \lambda\vec{U}.$$

In other words $\vec{U}$ is an eigen vector of S. Thus the variation

$$\sum_i \left(U^T x^{(i)}\right)^2 = \vec{U}^T S \vec{U} = \lambda \underbrace{U^T U}_{\text{Identity}} = \lambda$$

is maximized by the eigen vector u_1 corresponding to the largest eigen value λ_1 of S

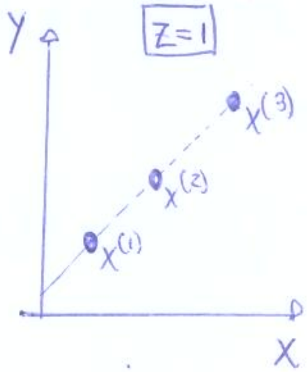
u_1 is then the first principal component.

3 Now compute the next axis, orthogonal to u_1.

This must be, again, an eigen vector of S, since $S\vec{u} = \lambda\vec{u}$ gives all the stationary points of variation, and hence is given by u_2, the eigenvector corresponding to the second largest eigenvalue λ_2 of S. U_2 is the 2nd principal component.

Example

Data: 3 points $x^{(1)}$, $x^{(2)}$, and $x^{(3)} \in \mathbb{R}^3$.



$$x^{(1)} = (1, 1, 1)^T$$
$$x^{(2)} = (2, 2, 1)^T$$
$$x^{(3)} = (3, 3, 1)^T.$$

centroid $\bar{X} = (2, 2, 1)^T$.

Centered data are now

$$x^{(1)} = (-1, -1, 0)$$
$$x^{(2)} = (0, 0, 0)$$
$$x^{(3)} = (1, 1, 0)$$

writing $X = \left[x^{(1)}, x^{(2)}, x^{(3)}\right]$, we have $X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

$$S = \sum_i x^{(i)} x^{(i)T} = XX^T = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Its eigen-vector decomposition is

$$S = \left[\vec{U}_1, \vec{U}_2, \vec{U}_3\right] \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot \left[\vec{U}_1, \vec{U}_2, \vec{U}_3\right]^T$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then $\cancel{Z}$ $z^{(i)} = \vec{U}_1^T x^{(i)}$ and $z^{(i)} = \{-\sqrt{2}, 0, \sqrt{2}\}$ for the points $x^{(i)}$.

Doing for one case: $\vec{U}_1^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix}$

$$x^{(1)} = \begin{bmatrix} -1 \\ -1 \\ 0 \end{bmatrix}$$

$$z^{(1)} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 0 \end{bmatrix} = -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} = -\frac{2}{\sqrt{2}} = -\frac{\cancel{2}\sqrt{2}}{\cancel{2}} = -\sqrt{2}$$

<u>Now we are ready for the algorithm</u>

Given data $\{x^{(1)}, \dots, x^{(m)}\} \in \mathbb{R}^n$

① Compute the centroide, center the data and optionally <u>scale</u> it

$$\vec{c} = \frac{1}{M} \sum_{i=1}^{M} \vec{x}^{(i)}.$$

and transform the data so that $\vec{c}$ becomes the new origin

$$\vec{x}^{(i)} \leftarrow x^{(i)} - \vec{c}.$$

② write the centered data in matrix form $X = \begin{bmatrix} | & | & | & & | \\ x^{(1)}, & x^{(2)}, & x^{(3)}, & \dots, & x^{(m)} \\ | & | & | & & | \end{bmatrix}$ and compute the covariance matrix ⓢ

$$S = \frac{1}{M} \cdot \sum_i x^{(i)} x^{(i)T} = \frac{1}{M} \cdot XX^T.$$

③ Compute the eigen decomposition of S

$$S = UDU^T$$

④ Principal components are the columns $\vec{u}_i$ of U ordered by the magnitude of the eigenvalues in D.

⑤ The dimensionality of the data is reduced by the projection $\vec{z} = U_d^T \cdot X$ where U_d are the first d columns of U and $\vec{z}$ is a d-vector.

Observations

① PCA is a linear transformation that rotates the data so that it is maximally decorrelated.

② A limitation of PCA is the linearity - it can't fit curved surfaces well.

How much is lost by PCA approximation

The eigenvectors (U) provide an orthogonal basis for any $\vec{x} \in \mathbb{R}^m$.

$$X = \sum_{j=1}^{m} (\vec{U}_j^T \vec{X}) \vec{U}_j \quad * \text{ matrices } U, X, U.$$

The PCA approx with (d) components is

$$\tilde{X} = \sum_{j=1}^{d} (U_j^T X) U_j$$

so, the error is

$$X - \tilde{X} = \sum_{j=d+1}^{m} (u_j^T x) u_j$$

Using $U_j^T U_K = \delta_{jK}$, the squared error is

$$\| X - \tilde{X} \|^2 = \left(\sum_{j=d+1}^{m} (u_j^T \cdot x) U_j \right)^2 = \sum_{d+1}^{m} U_j^T (xx^T) u_j$$

Hence the mean squared error (mse) is

$$\frac{1}{M} \sum_{i}^{M} \| x^{(i)} - \tilde{x}^{(i)} \|^2 = \frac{1}{M} \sum_{d+1}^{N} U_j^T \cdot (x^{(i)} x^{(i)T}) U_j = \sum_{d+1}^{m} U_j^T \, S \, U_j$$

and since $SU_j = \lambda U_j$,

$$\boxed{\frac{1}{M} \sum_{i}^{M} \| x^{(i)} - \tilde{x}^{(i)} \| = \sum_{d+1}^{m} U_j^T S U_j = \sum_{d+1}^{m} \lambda_j}$$

(1) what the last equation shows is that the squared reconstruction error is given by the sum of eigenvalues of the unused eigenvectors.

---x---