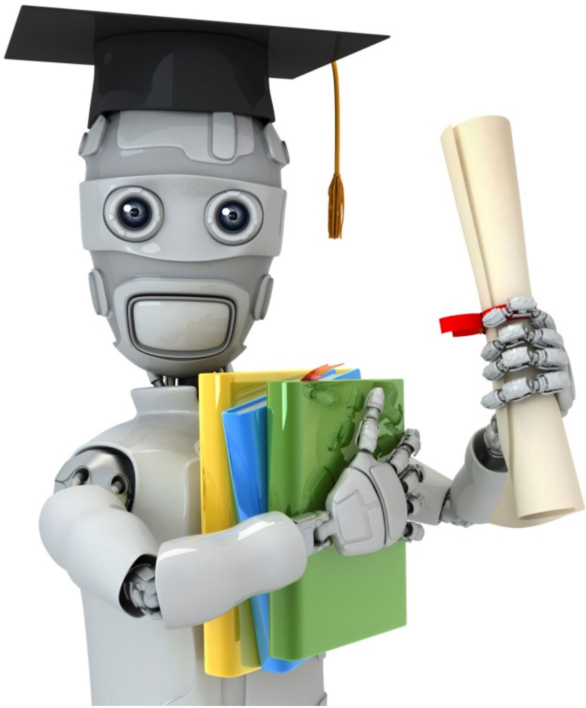
2nd type unsupervised learning algorithm

Dimensionality Reduction

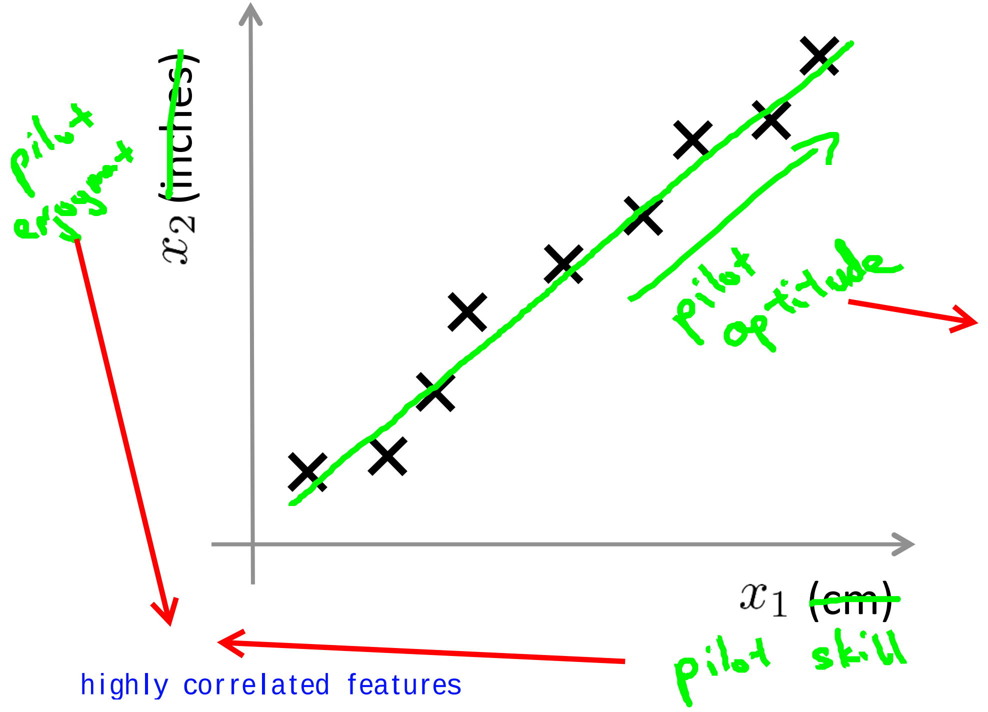
Motivation I: Data Compression

This can help our algorithm to run faster!



Machine Learning

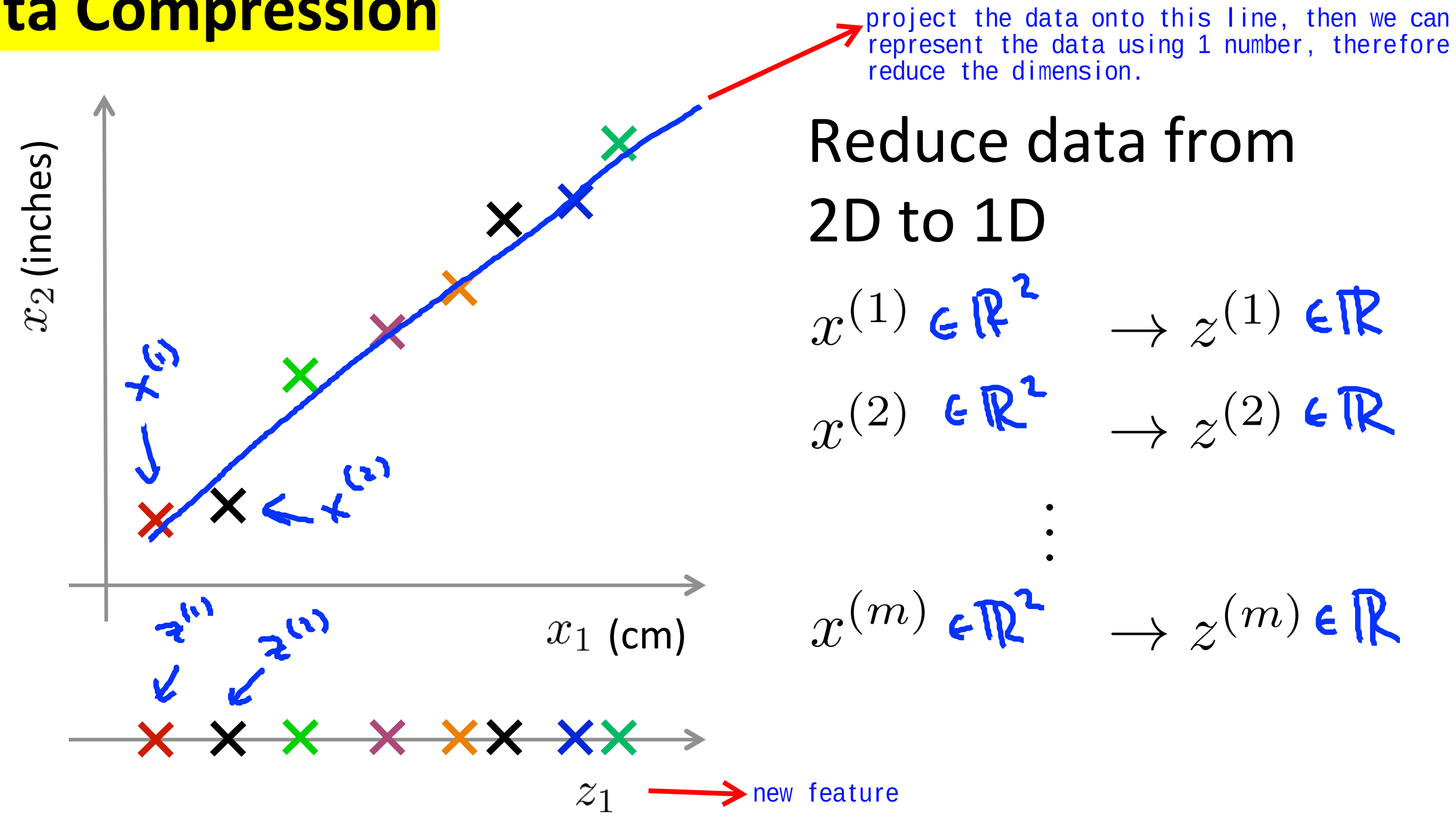
Data Compression



one-D data

Reduce data from 2D to 1D

Data Compression



Reduce data from 2D to 1D

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

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

$$\vdots$$

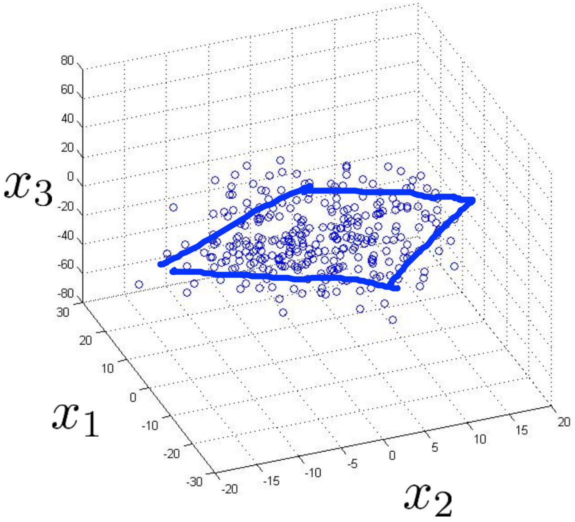
$$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$$

Data Compression

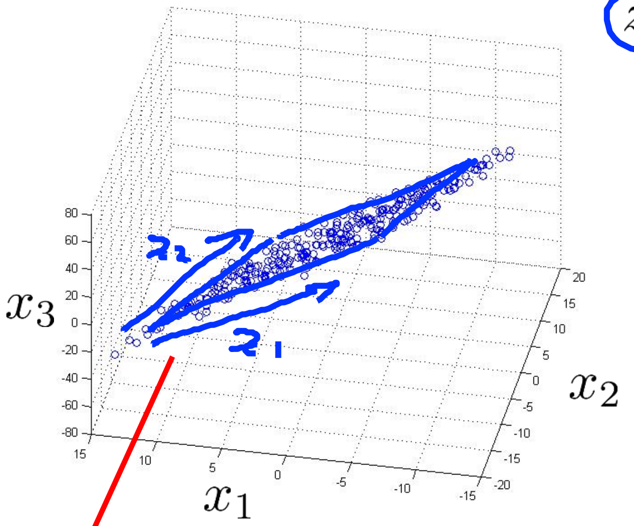
In a typical setting, we may have 10000 dimensions

1000D → 100D

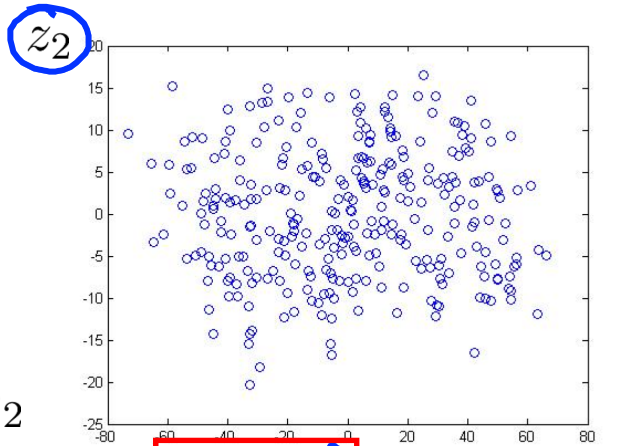
Reduce data from 3D to 2D



$\boxed{x^{(i)} \in \mathbb{R}^3}$



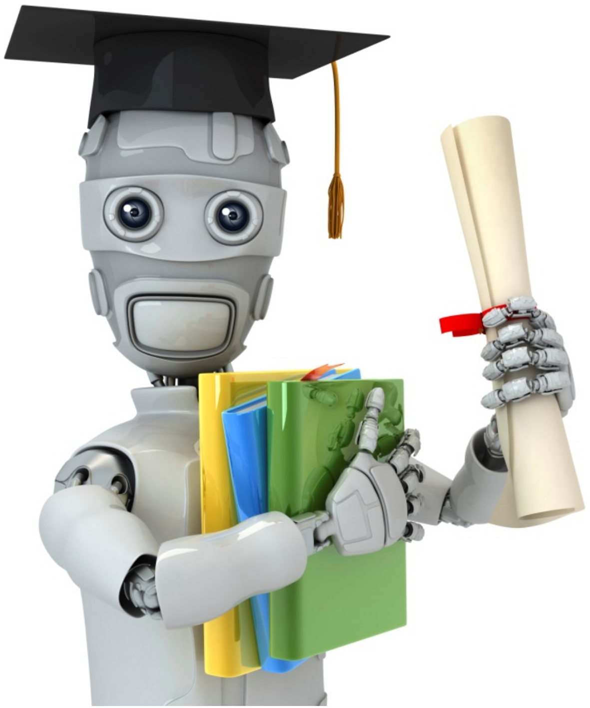
Project the data into this plane



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

z_1

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



Machine Learning

Dimensionality Reduction

Motivation II: Data Visualization

Fact about some countries in the world

Data Visualization

$\boxed{x \in \mathbb{R}^{50}}$

50 features

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

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	…
…	…	…	…	…	…	…	

[resources from en.wikipedia.org]

Andrew Ng

Data Visualization

with a new pair of features, here we only visualize data with 2 D

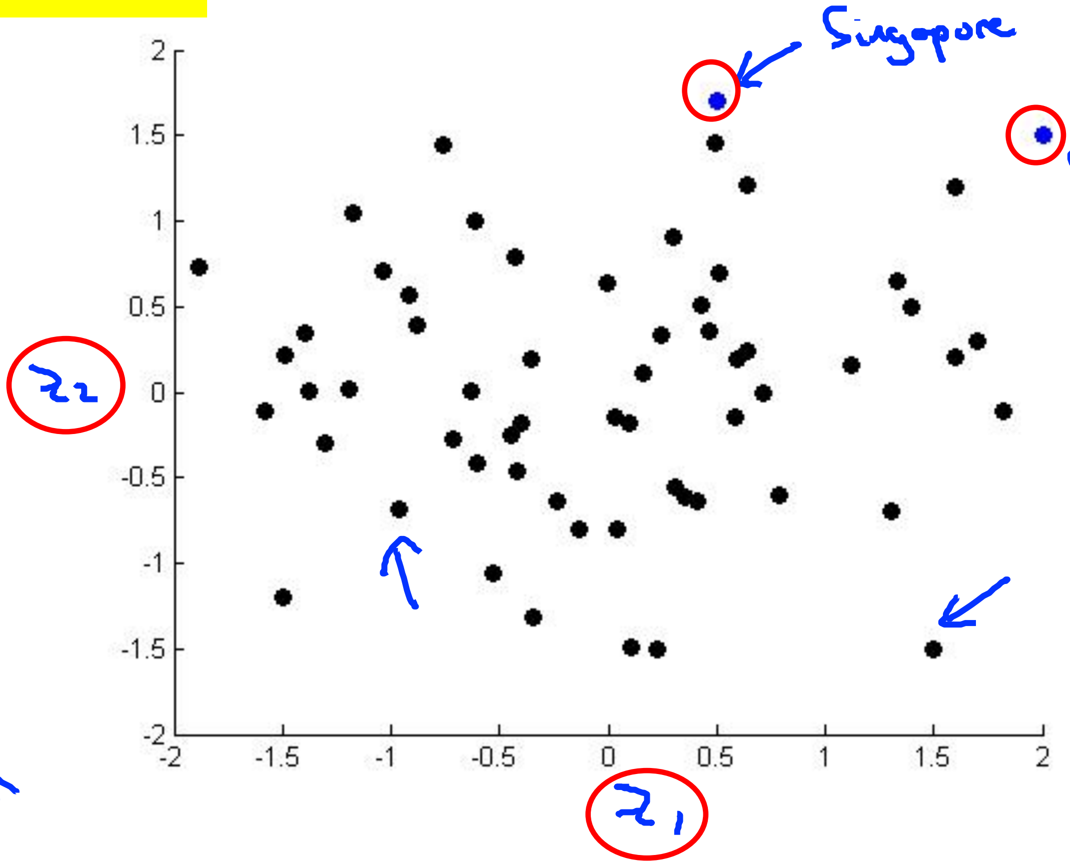
$z^{(i)} \in \mathbb{R}^2$

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
…	…	…

Reduce data from 50D to 2D

Data Visualization

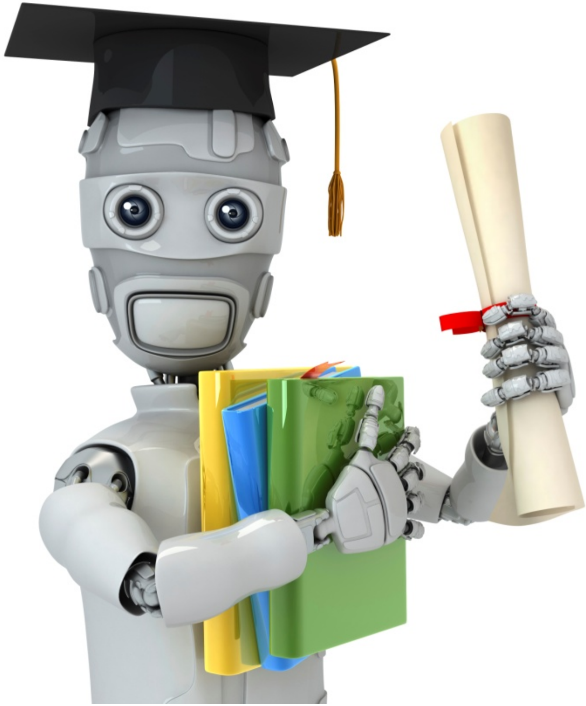
per. person GDP (economic activity)



Country size / GDP

1

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



Machine Learning

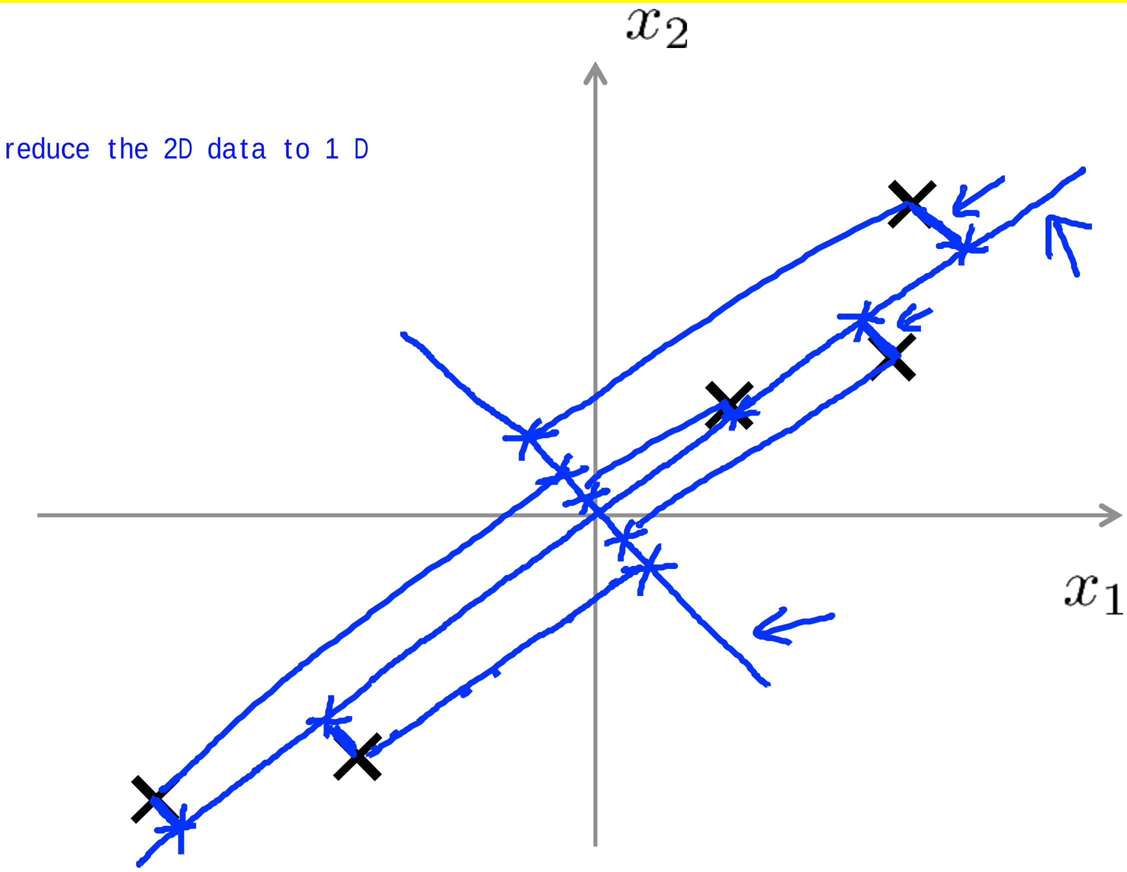
Dimensionality Reduction

Principal Component Analysis problem formulation

By far, the most popular and commonly used unsupervised learning algorithm

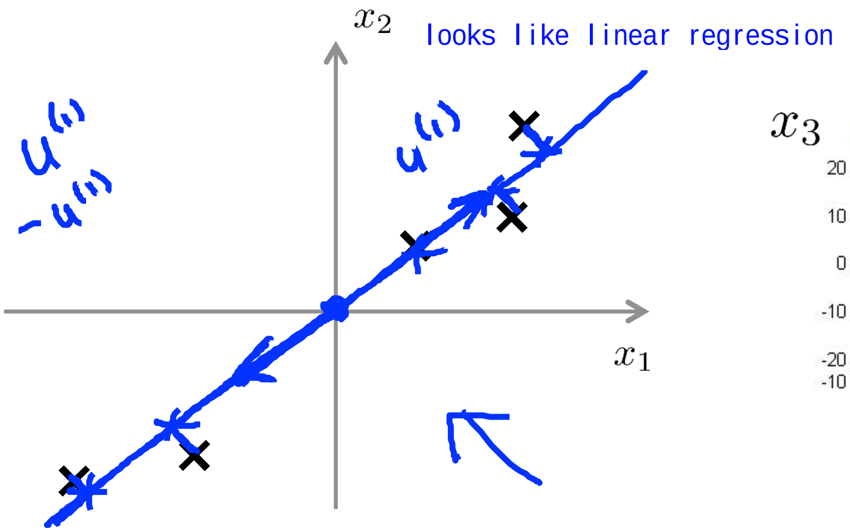
Principal Component Analysis (PCA) problem formulation

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



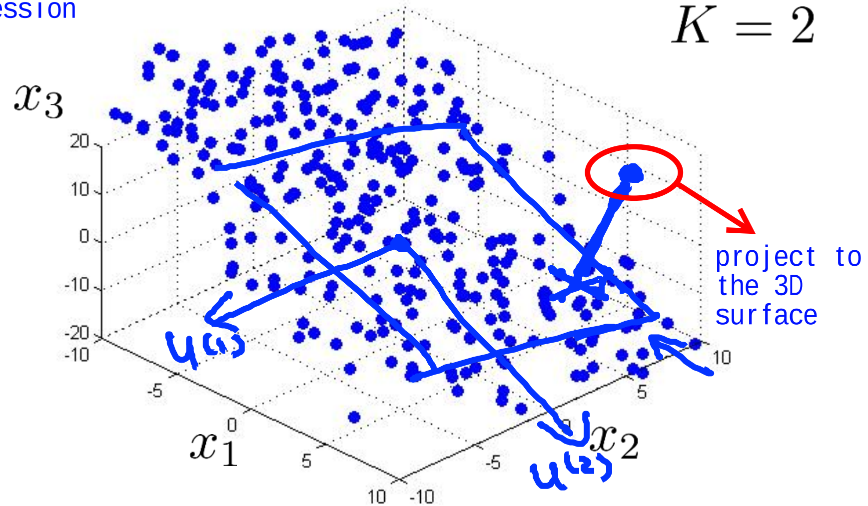
Formal statement

Principal Component Analysis (PCA) problem formulation



$3D \rightarrow 2D$

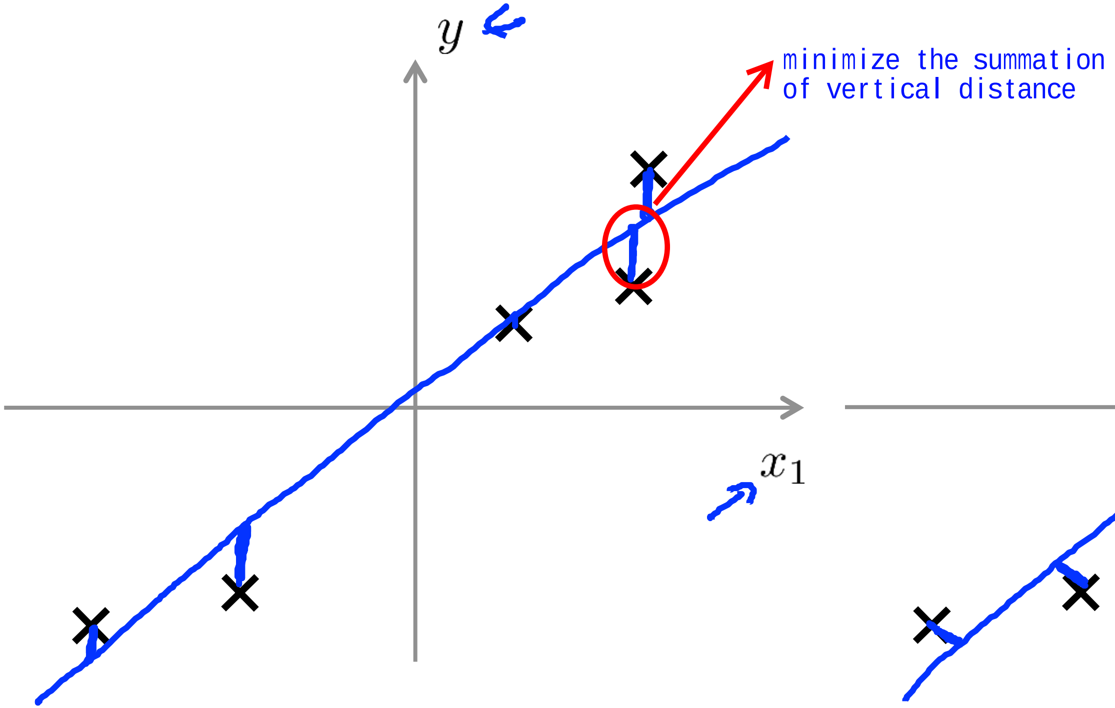
$K = 2$

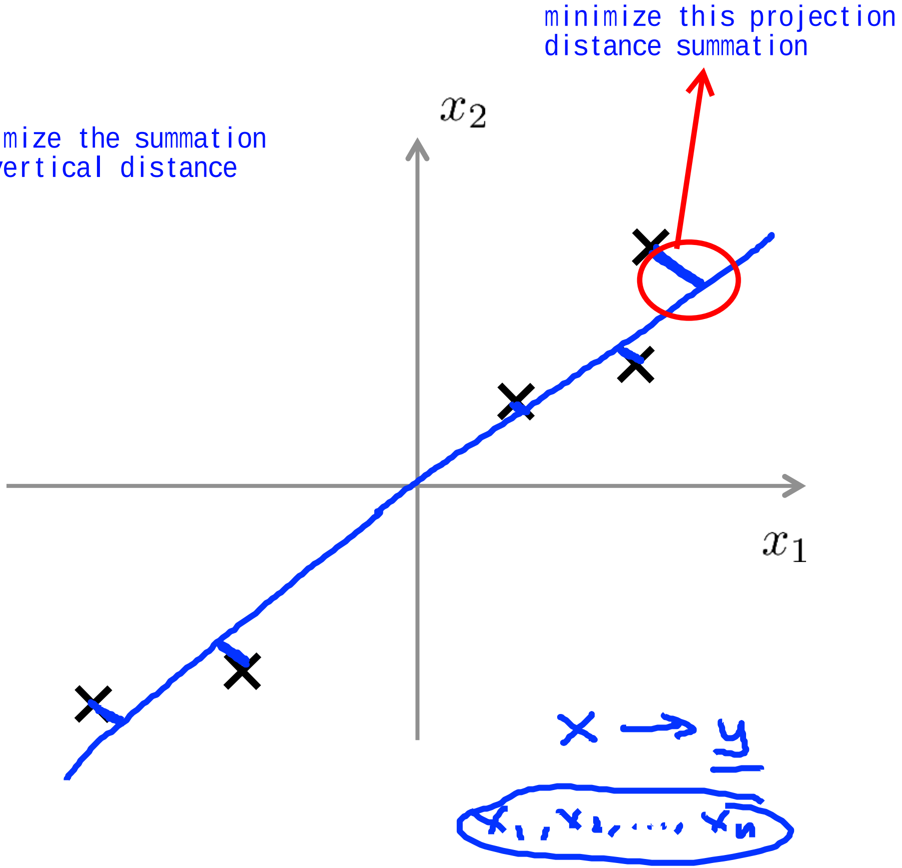


Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

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.

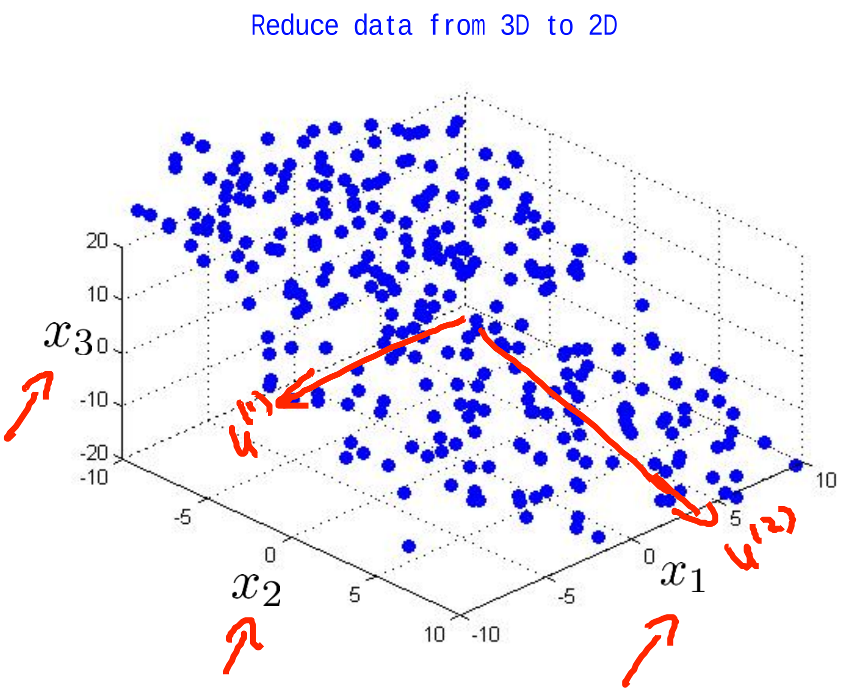
PCA is not linear regression

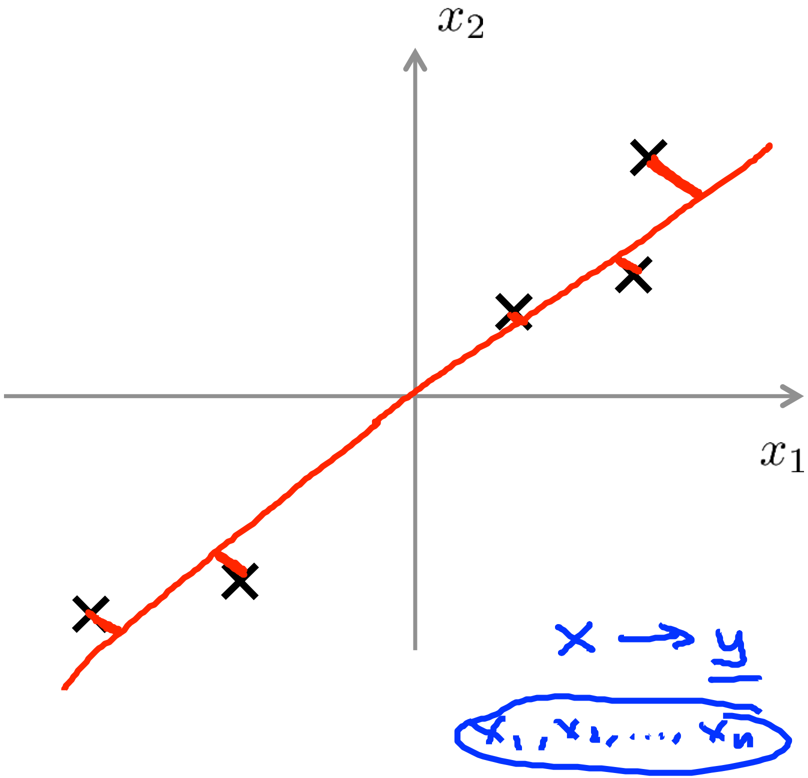


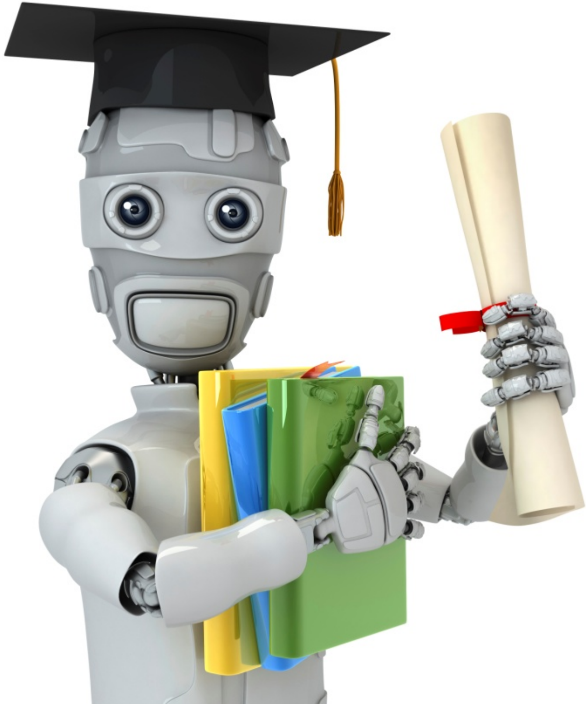


Andrew Ng

PCA is not linear regression







Dimensionality Reduction

Principal Component Analysis algorithm

Machine Learning

Data preprocessing

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

Preprocessing (feature scaling/mean normalization):

Before we actually apply PCA

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

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

zero mean feature

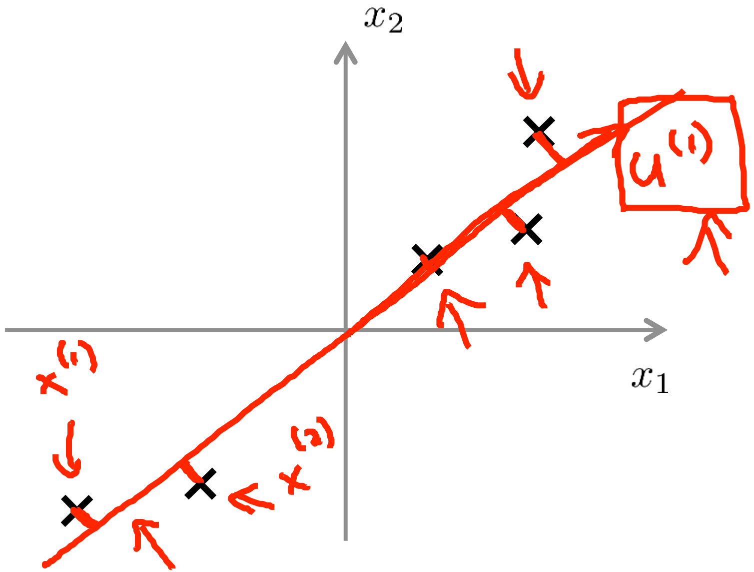
If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

feature scaling

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

some measures of the feature j
e.g. Max value

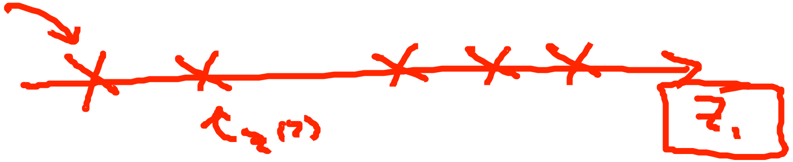
Principal Component Analysis (PCA) algorithm

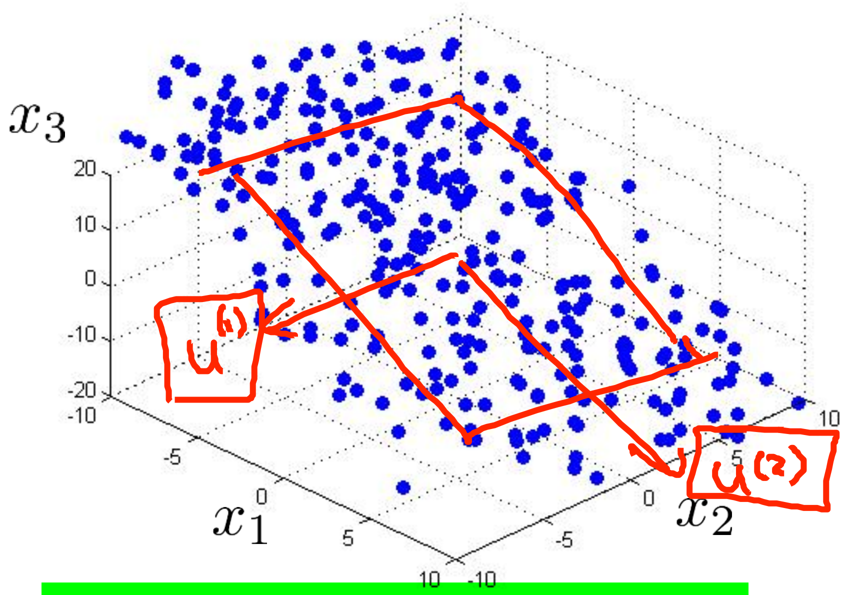


Reduce data from 2D to 1D

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

$z^{(1)}$





Reduce data from 3D to 2D

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

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

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

$n\times n$ Sigma

Compute "eigenvectors" of matrix Σ:

→ Singular value decomposition

eig(Sigma)

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

$n\times n$ matrix

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \ldots & u^{(n)} \\ | & | & | & & | \end{bmatrix}$$

k

we mainly require this one

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

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

Andrew Ng

Principal Component Analysis (PCA) algorithm

From `[U,S,V] = svd(Sigma)`, we get:

$$\rightarrow U = \left[\begin{array}{cccc} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(n)} \\ | & | & & | \end{array}\right] \in \mathbb{R}^{n \times n}$$

k

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

low dimensional representation of x^{i}

K dimensional vector

$$z^{(i)} = \underbrace{\left[\begin{array}{cccc} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(k)} \\ | & | & & | \end{array}\right]^T}_{n \times k} x^{(i)} = \underbrace{\underbrace{\left[\begin{array}{c} \text{---} (u^{(1)})^T \text{---} \\ \vdots \\ \text{---} (u^{(k)})^T \text{---} \end{array}\right]}_{k \times n} \underbrace{x^{(i)}}_{n \times 1}}_{k \times 1}$$

U_{reduce}

$z \in \mathbb{R}^k$

Principal Component Analysis (PCA) algorithm summary

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

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

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

$\boxed{\text{Sigma} = (1/m) * X' * X;}$

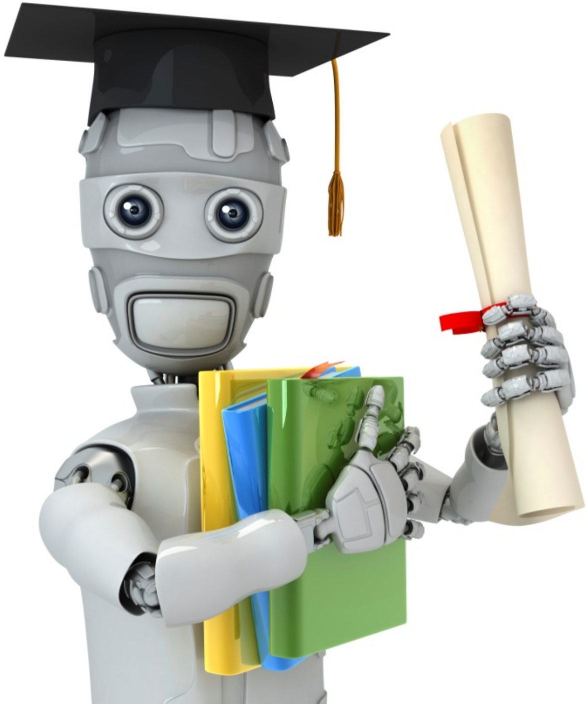
Vector representation of sigma!!

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

we grab the 1st k columns

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

find the z representation



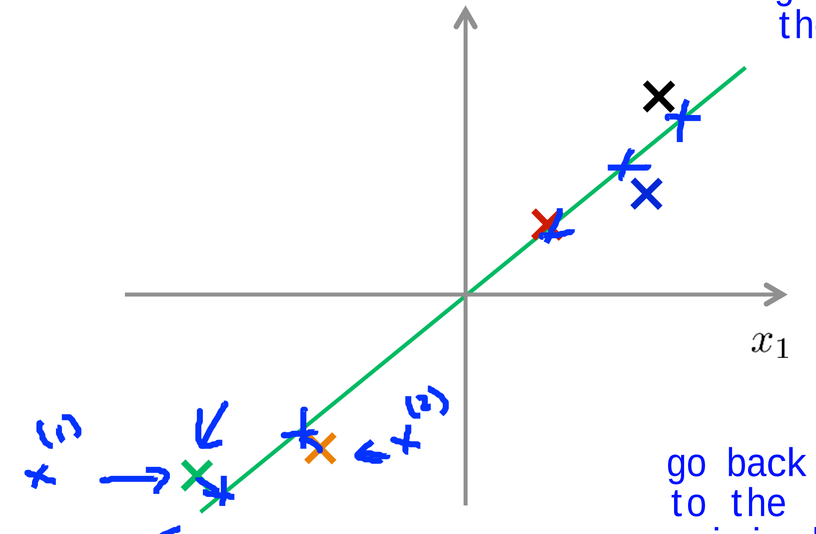
Machine Learning

Dimensionality Reduction

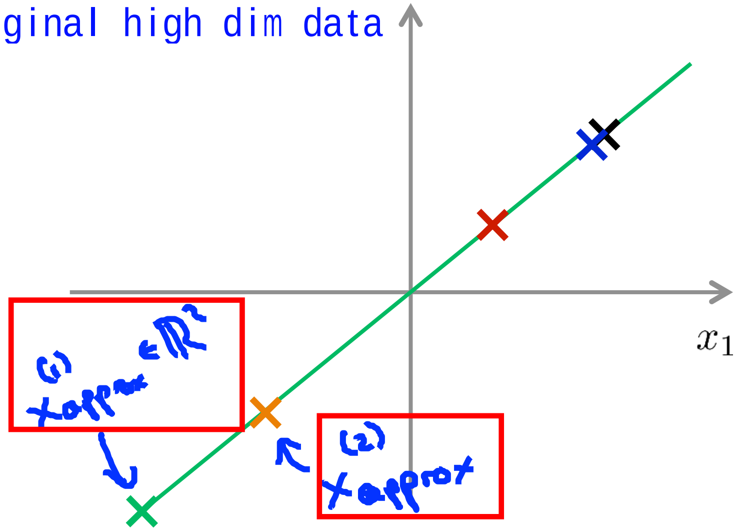
Reconstruction from compressed representation

This section shows how do we go back from 100 dimensional data to 1000 dimensional data!

Reconstruction from compressed representation

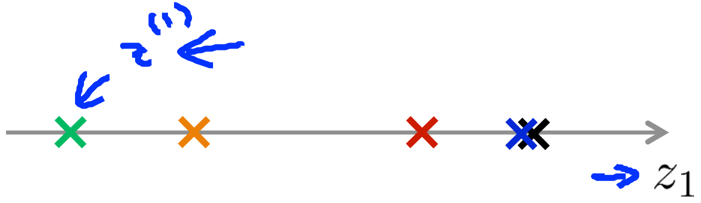


go back from low dim to the original high dim data



go back to the original data

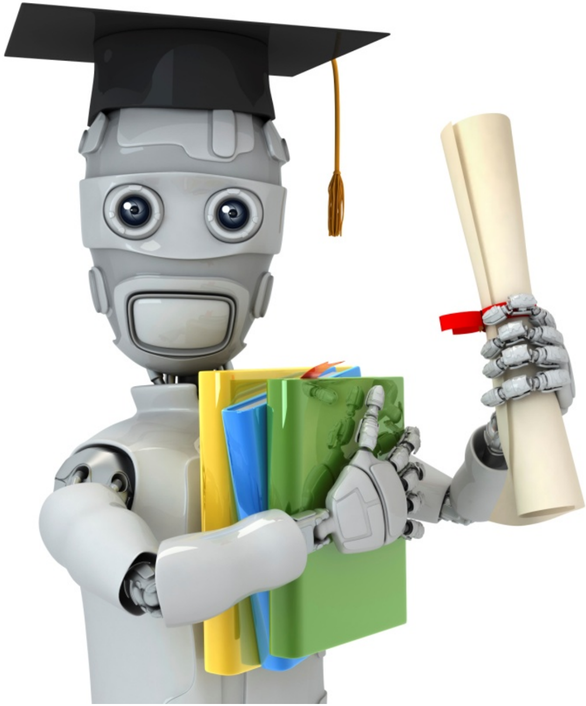
$$\boxed{z = U_{reduce}^T x}$$



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

$$\tilde{x}^{(1)} \approx x^{(1)}_{approx} = \underbrace{\underbrace{U_{reduce}}_{n\times k} \cdot \underbrace{z^{(1)}}_{k\times 1}}_{n\times 1}$$

$$x^{(1)}_{approx} \in \mathbb{R}^n$$



Machine Learning

Dimensionality Reduction

Choosing the number of principal components

number of principal components to retain

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

k is used to estimate x_approx

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

$\cancel{0.05}$ $\cancel{5\%}$

0.10 (10%)

"~~99%~~ of variance is retained"

~~95%~~ 90%

Choosing k (number of principal components)

much more efficient

Algorithm:

Try PCA with $\cancel{k=1}$ $\cancel{k=2}$ $\cancel{k=3}$ $k=4$ ...

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

Check if

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

$k=17$

Equivalent!!!

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

$$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & S_{33} & \\ & & & \ddots \\ & & & & S_{nn} \end{bmatrix}$$

$k=3$

For given k

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

Simpler Calculation!!!

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

Choosing k (number of principal components)

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

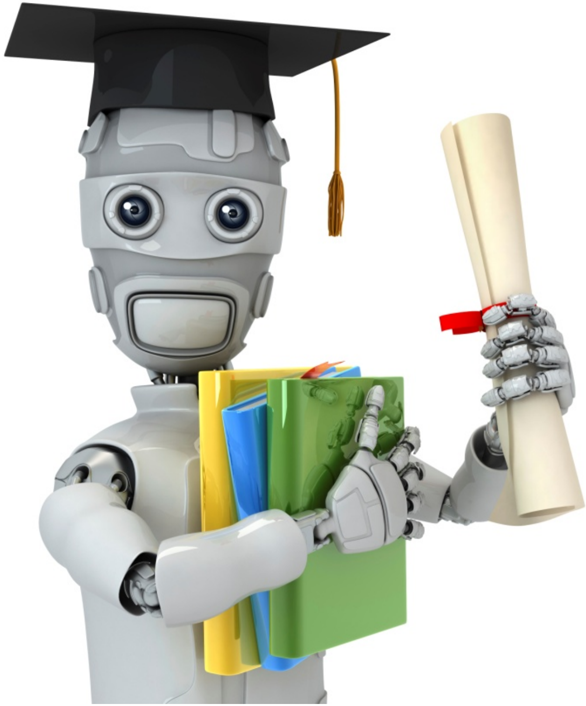
Pick smallest value of k for which

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

$k=100$

(99% of variance retained)

Summary!



Machine Learning

Dimensionality Reduction

Advice for applying PCA

One common use of PCA

Supervised learning speedup

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

very large feature

100

100

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

Extract inputs:

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

$\downarrow PCA$

x

$\downarrow$

z

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

U_{reduce}

New training set:

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

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

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.

$x \rightarrow z$

Then apply this mapping to cross validation and test sets

Application of PCA

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

Choose k by % of variance retain

- Visualization

$k = 2$ or $k = 3$

This is not a good way to use PCA

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.

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

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 $\{(\cancel{z^{(1)}}\, x^{(i)}, y^{(1)}), \ldots, (\cancel{z^{(m)}}\, x^{(m)}, y^{(m)})\}$
- 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 $\boxed{x^{(i)}}$. Only if that doesn't do what you want, then implement PCA and consider using $\underline{z^{(i)}}$.