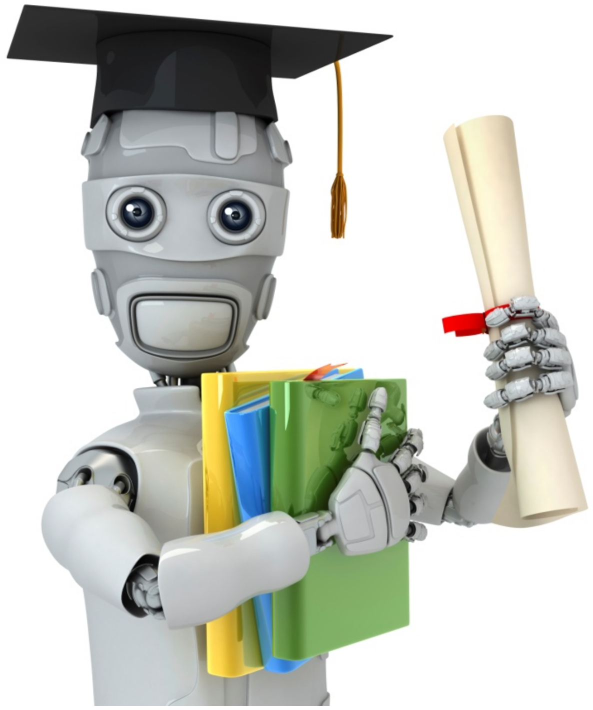
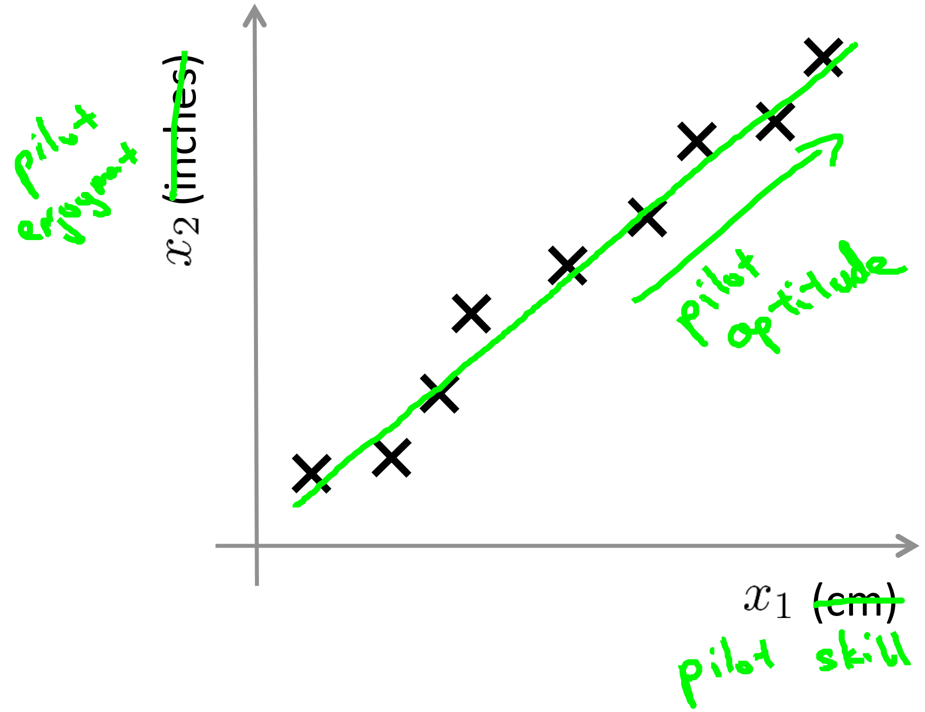


Machine Learning

Dimensionality Reduction

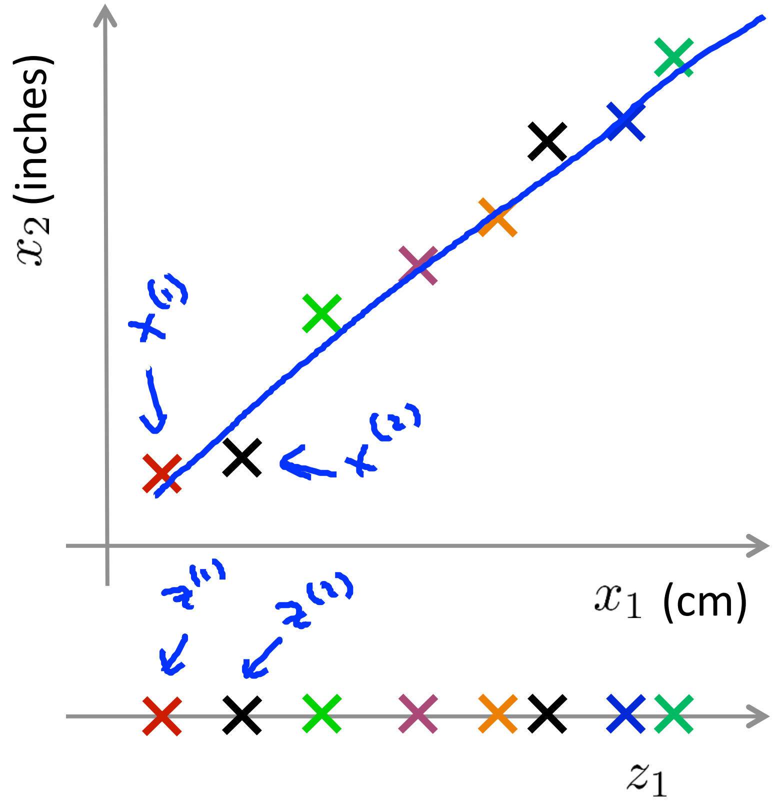
Motivation I: Data Compression

Data Compression



Reduce data from 2D to 1D

Data Compression



Reduce data from 2D to 1D

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

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

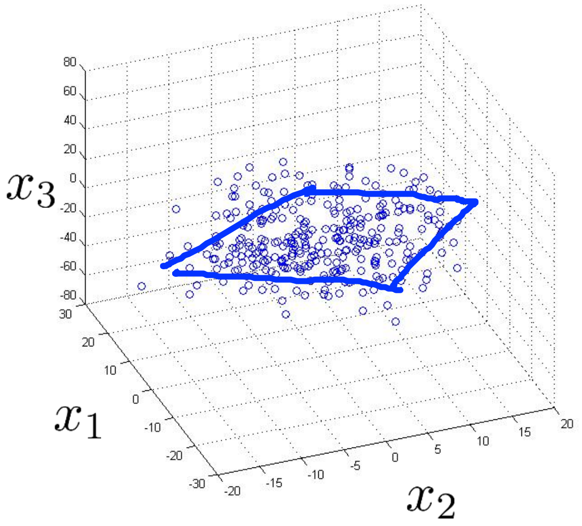
$\vdots$

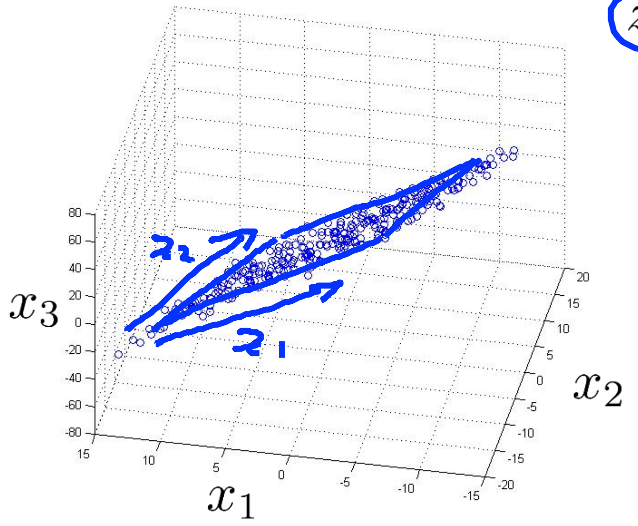
$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$

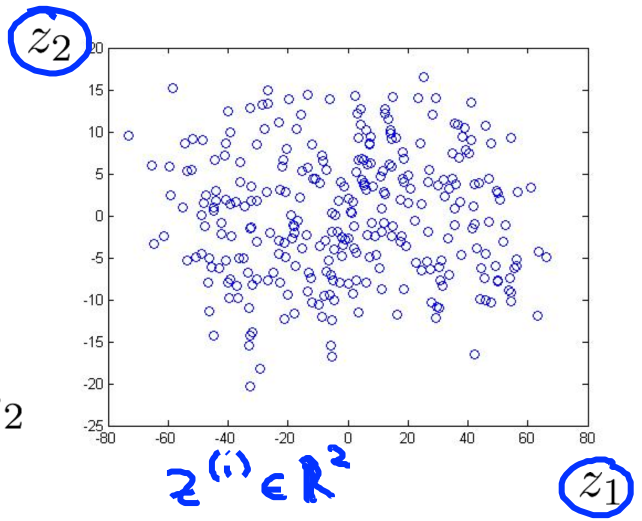
Data Compression

$1000D \rightarrow 100D$

Reduce data from $\underline{\text{3D to 2D}}$







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

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

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

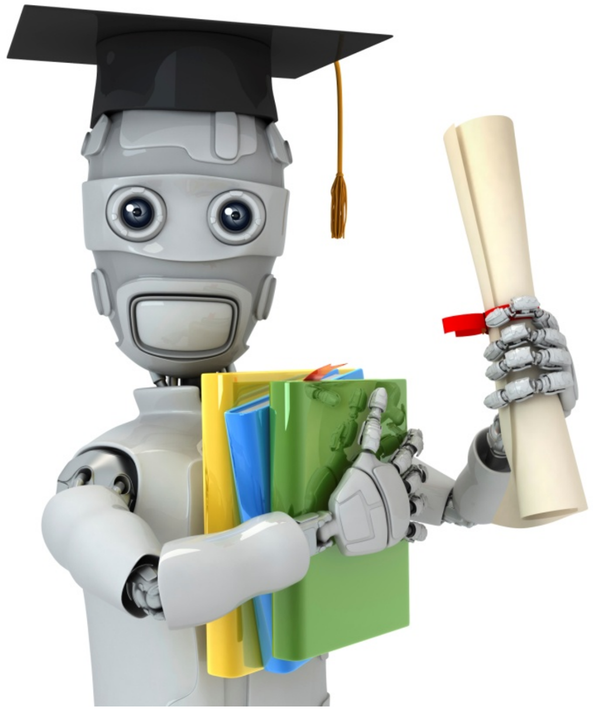
Andrew Ng

Suppose we apply dimensionality reduction to a dataset of m examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in \mathbb{R}^n$. As a result of this, we will get out:

- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k \leq n$.
- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k > n$.
- [✓ A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in \mathbb{R}^k$ for some value of k and $k \leq n$.]

Correct

- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in \mathbb{R}^k$ for some value of k and $k > n$.



Machine Learning

Dimensionality Reduction

Motivation II: Data Visualization

Data 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	…
…	…	…	…	…	…	…	

[resources from en.wikipedia.org]

So is there something we can do to try to understand our data better?

Andrew Ng

Data Visualization

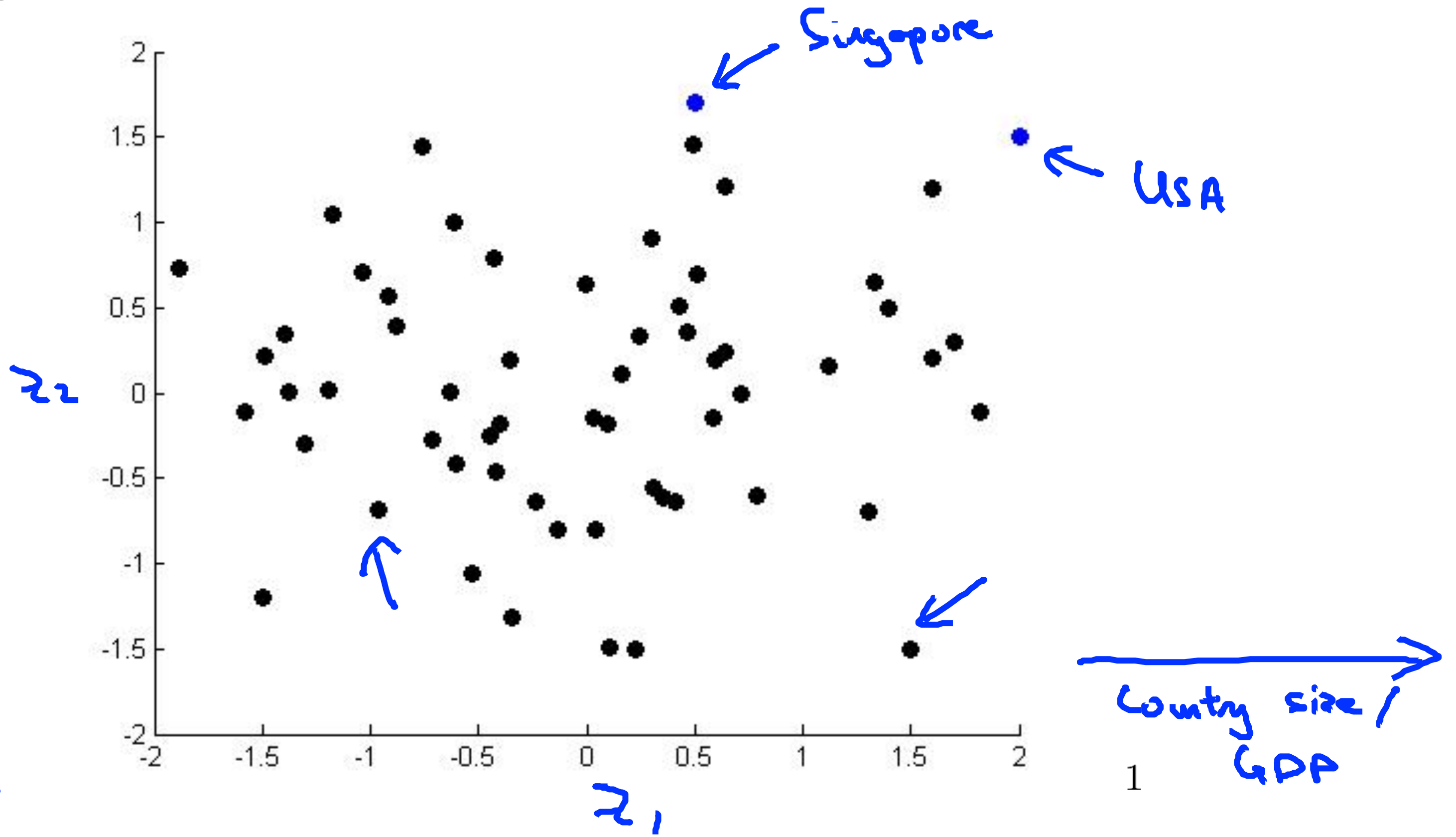
Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
…	…	…

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

Reduce data from 50D to 2D

Andrew Ng

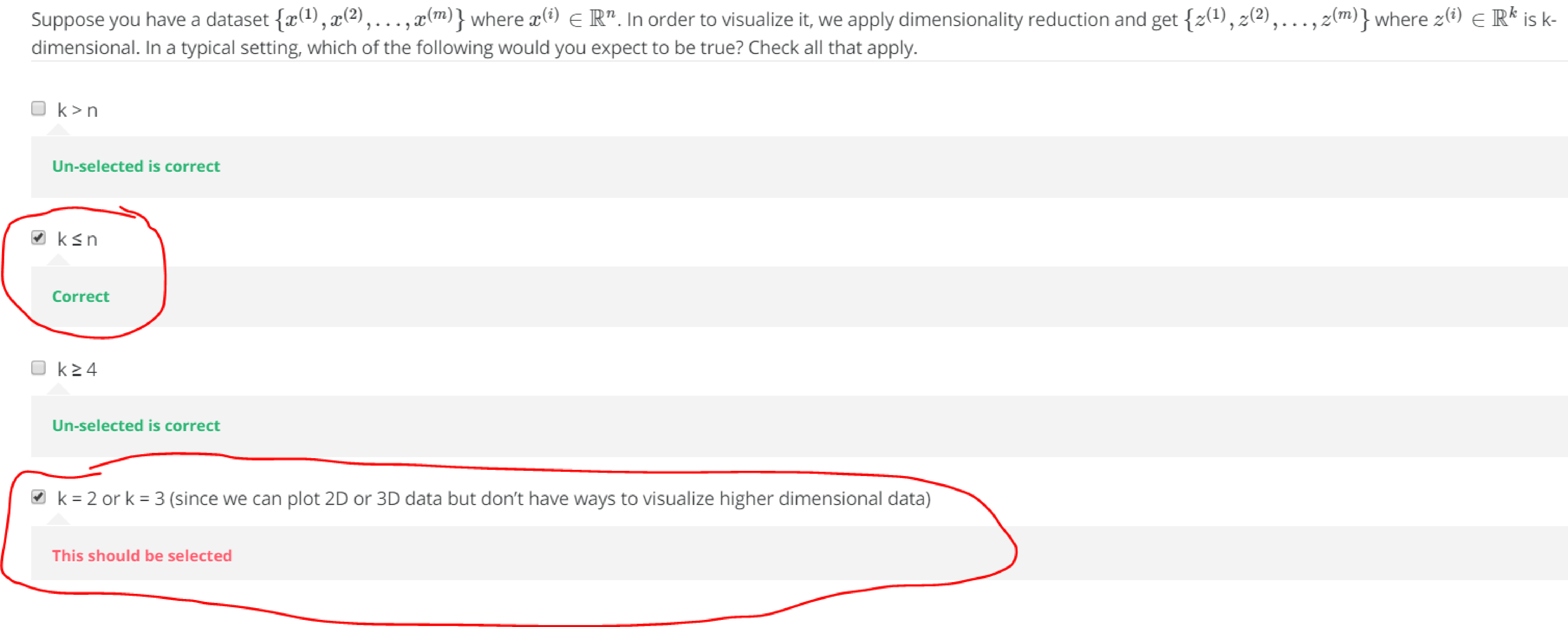
Data Visualization



per. person GDP (economic activity)

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

Andrew Ng



Suppose you have a dataset $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ where $x^{(i)} \in \mathbb{R}^n$. In order to visualize it, we apply dimensionality reduction and get $\{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$ where $z^{(i)} \in \mathbb{R}^k$ is k-dimensional. In a typical setting, which of the following would you expect to be true? Check all that apply.

- [ ] k > n

  Un-selected is correct

- [✓] k ≤ n

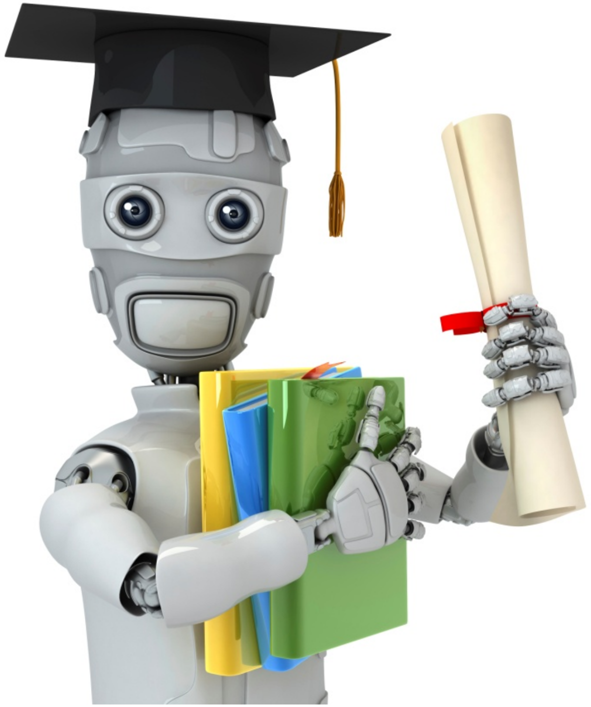
  Correct

- [ ] k ≥ 4

  Un-selected is correct

- [✓] k = 2 or k = 3 (since we can plot 2D or 3D data but don't have ways to visualize higher dimensional data)

  This should be selected



Dimensionality Reduction

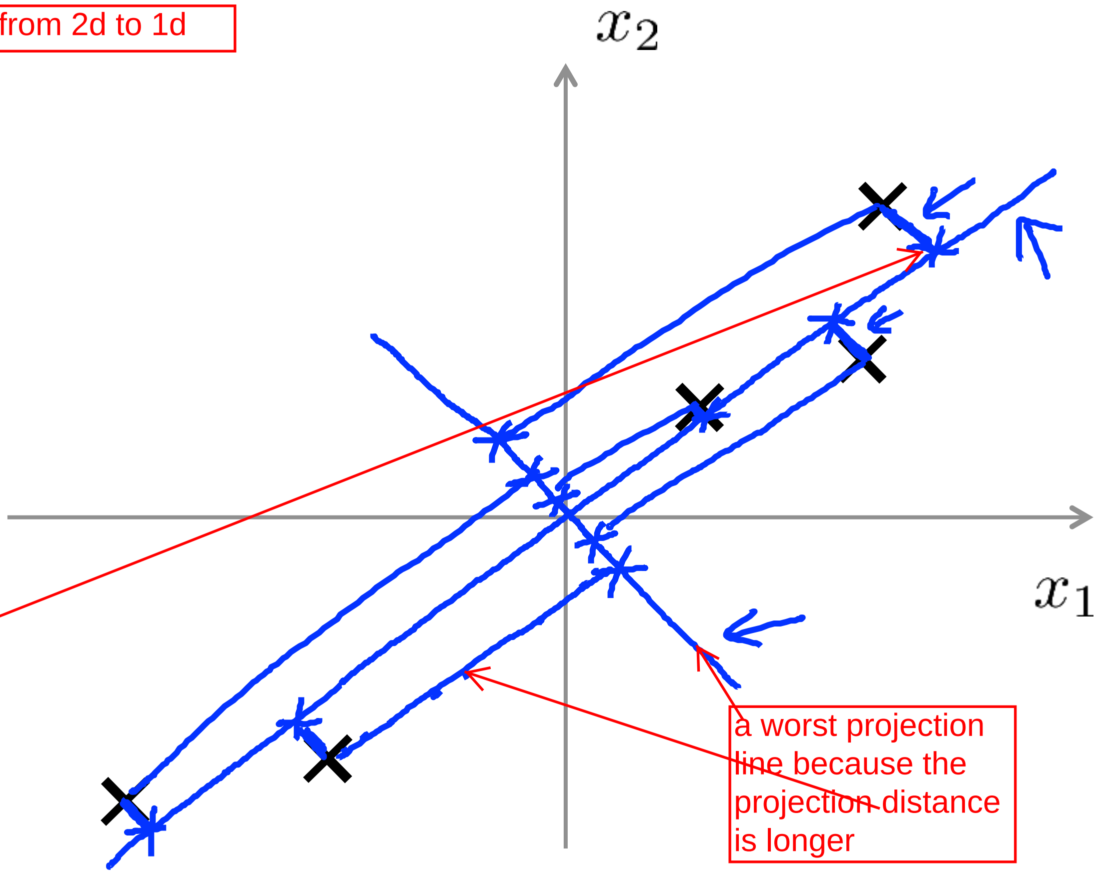
Principal Component Analysis problem formulation

For the problem of dimensionality reduction, by far the most popular, the most commonly used algorithm is something called principle components analysis, or PCA.

Principal Component Analysis (PCA) problem formulation

we wanna reduce it from 2d to 1d

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



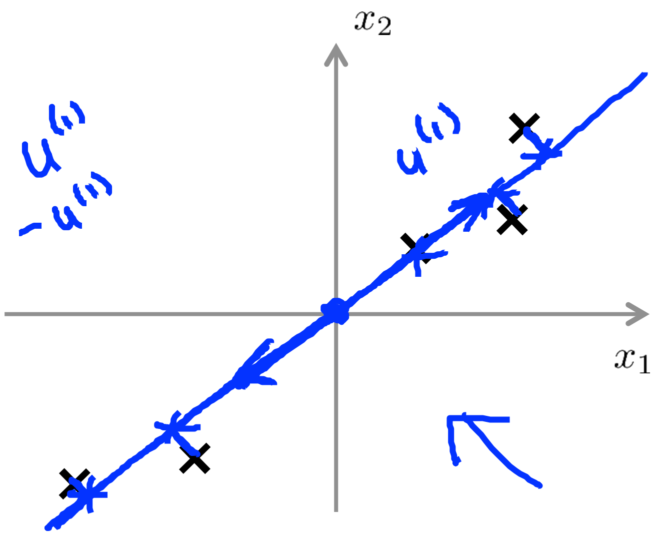
what PCA does formally is it tries to find a lower dimensional surface, really a line in this case, onto which to project the data so that the sum of squares of these little blue line segments is minimized.

a worst projection line because the projection distance is longer

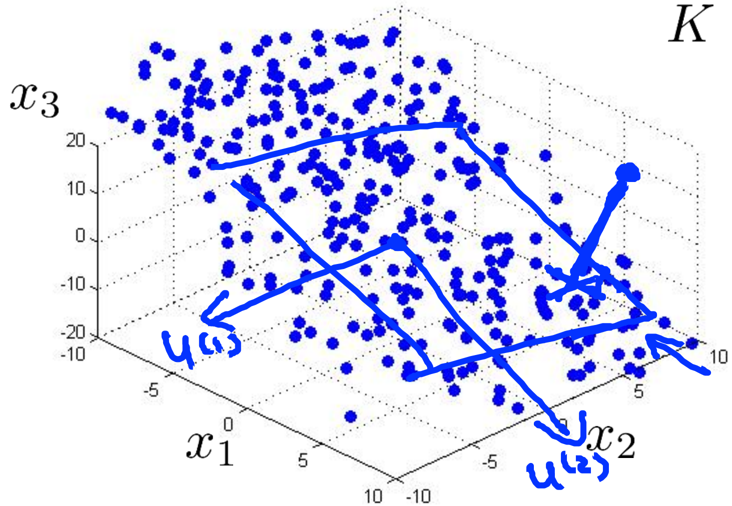
before applying PCA, it's standard practice to first perform mean normalization at feature scaling so that the features x1 and x2 should have zero mean, and should have comparable ranges of values.

Principal Component Analysis (PCA) problem formulation

pca gives u1 or -u1 doesnt matter



$3D \rightarrow 2D$
$K = 2$



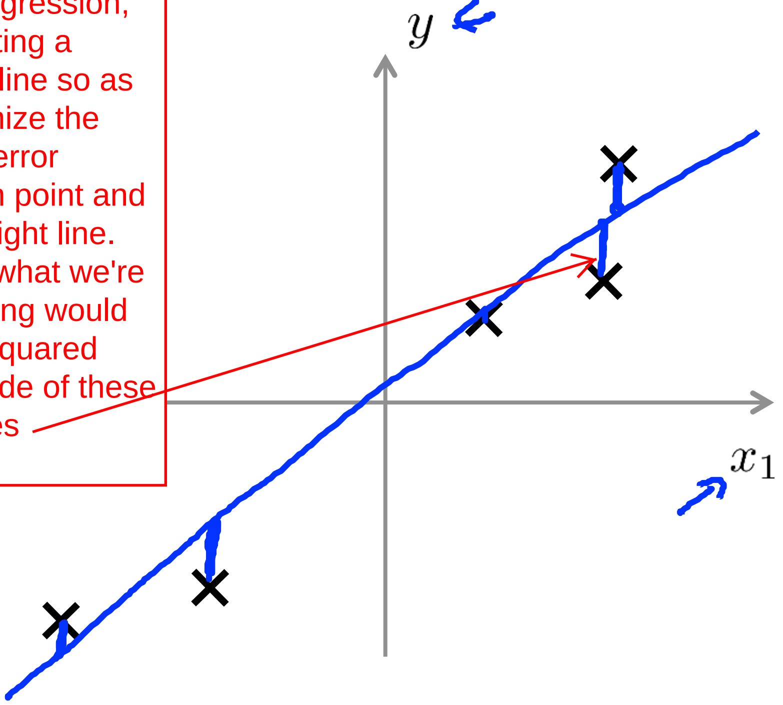
Reduce from 2-dimension to 1-dimension: Find a direction (a vector $\underline{u^{(1)} \in \mathbb{R}^n}$) onto which to project the data so as to minimize the projection error.

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

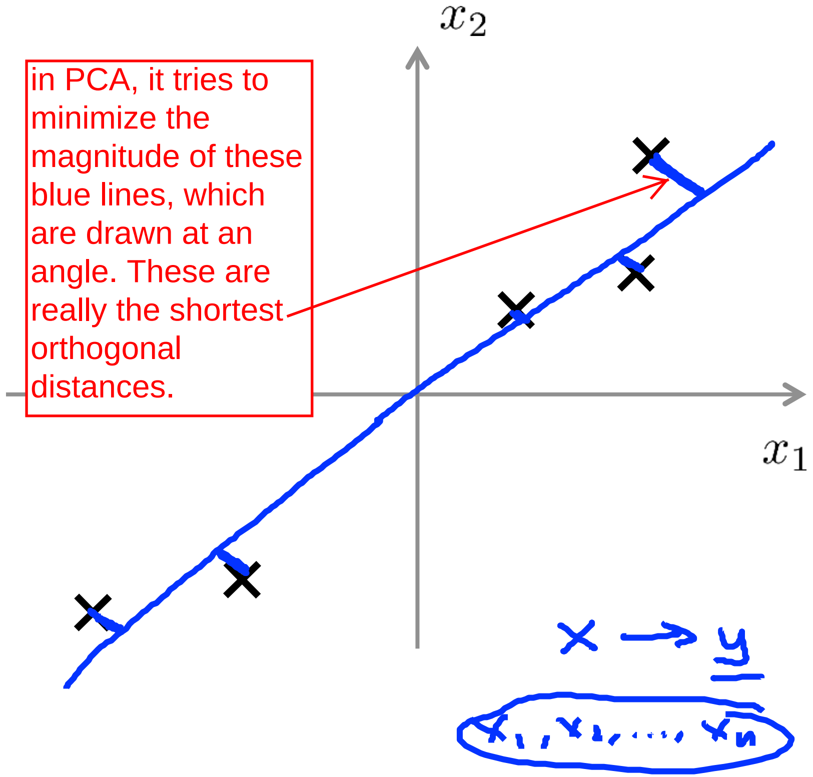
the definition of this is to find the set of vectors u(1), u(2), .. to u(k). And we're going to project the data onto the linear subspace spanned by this set of k vectors so as to minimize the sort of projection distance(projection error).

PCA is not linear regression

linear regression, we're fitting a straight line so as to minimize the square error between point and this straight line. And so what we're minimizing would be the squared magnitude of these blue lines

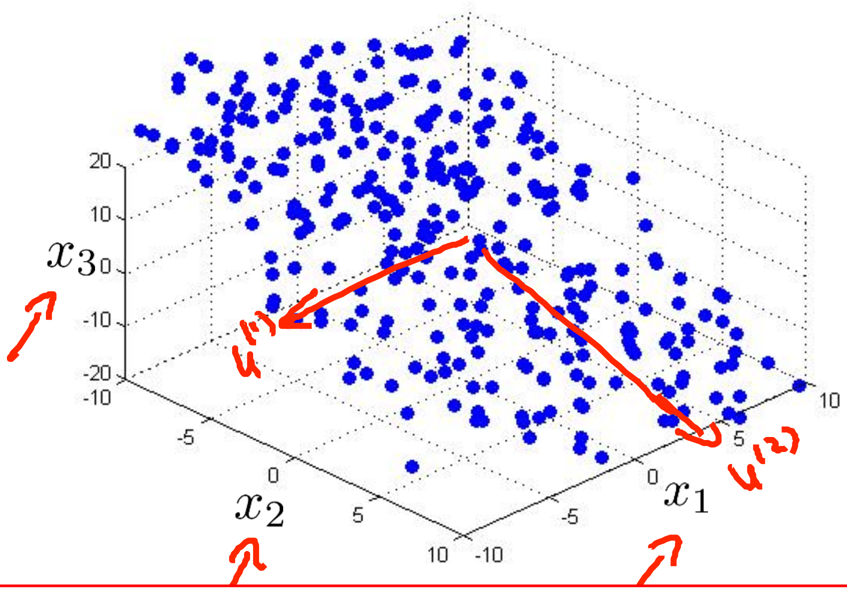


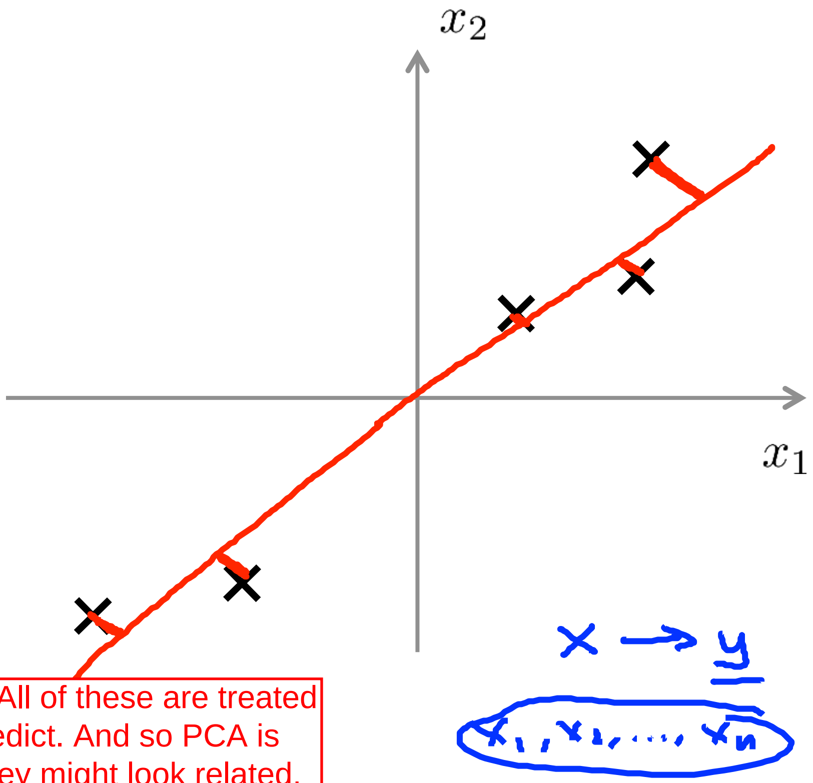
in PCA, it tries to minimize the magnitude of these blue lines, which are drawn at an angle. These are really the shortest orthogonal distances.



Andrew Ng

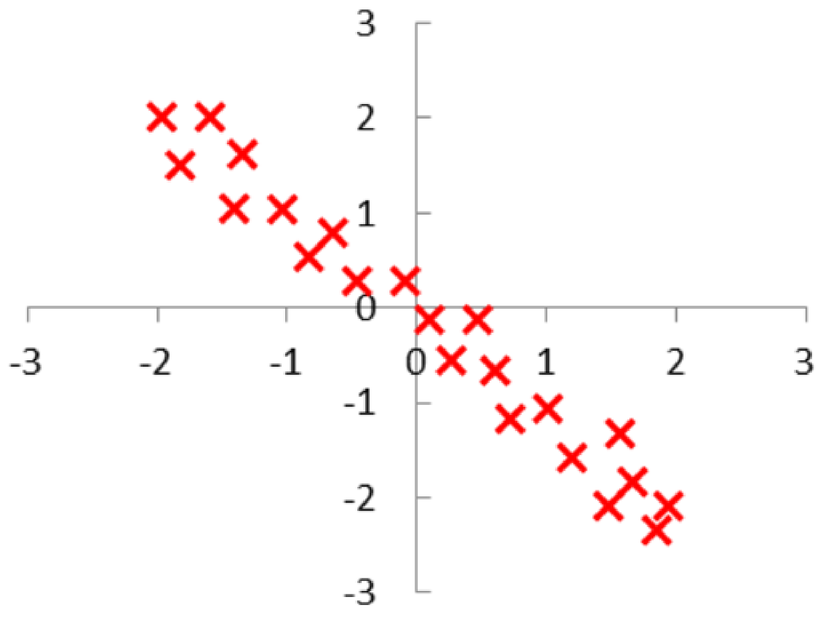
PCA is not linear regression





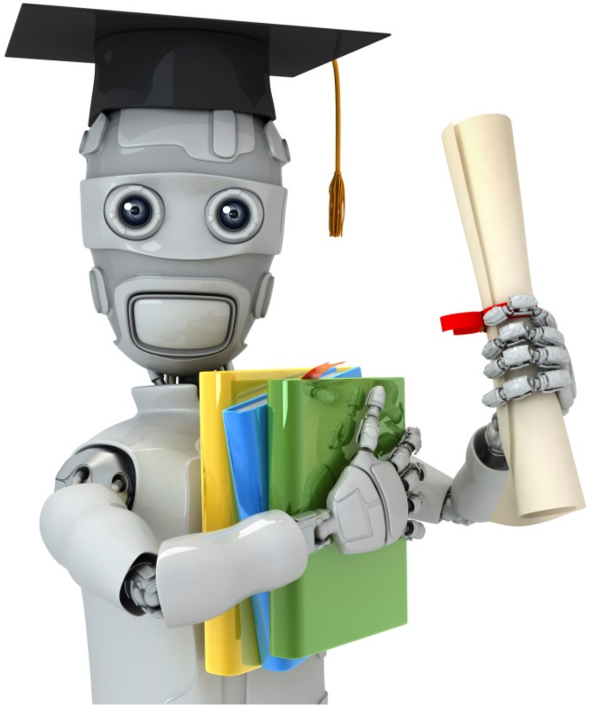
I have three features, x1, x2, x3, and all of these are treated alike. All of these are treated symmetrically and there's no special variable y that I'm trying to predict. And so PCA is not a linear regression, and even though at some cosmetic level they might look related, these are actually very different algorithms.

Suppose you run PCA on the dataset below. Which of the following would be a reasonable vector $u^{(1)}$ onto which to project the data? (By convention, we choose $u^{(1)}$ so that $\|u^{(1)}\| = \sqrt{(u_1^{(1)})^2 + (u_2^{(1)})^2}$, the length of the vector $u^{(1)}$, equals 1.)



- [ ] $u^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- [ ] $u^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- [ ] $u^{(1)} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
- [✓] $u^{(1)} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Correct



Machine Learning

Dimensionality Reduction

Principal Component Analysis algorithm

Data preprocessing

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

Preprocessing (feature scaling/$\underline{\text{mean normalization}}$):

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

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

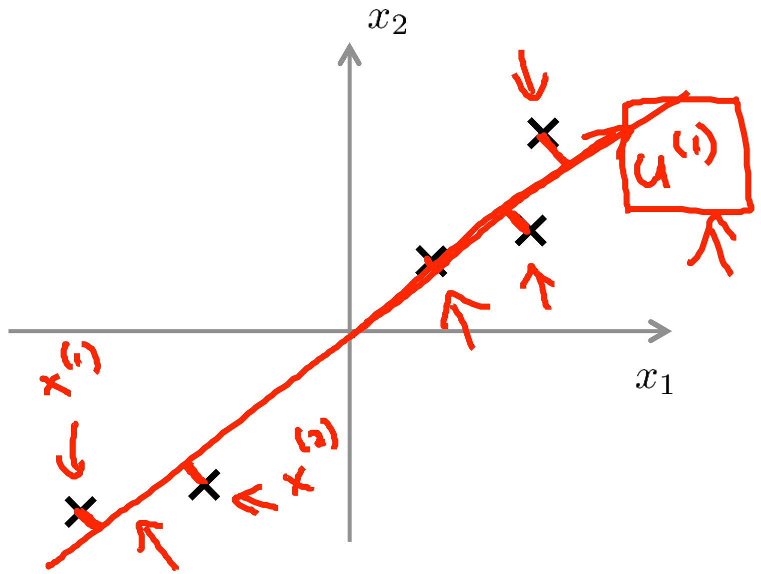
If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

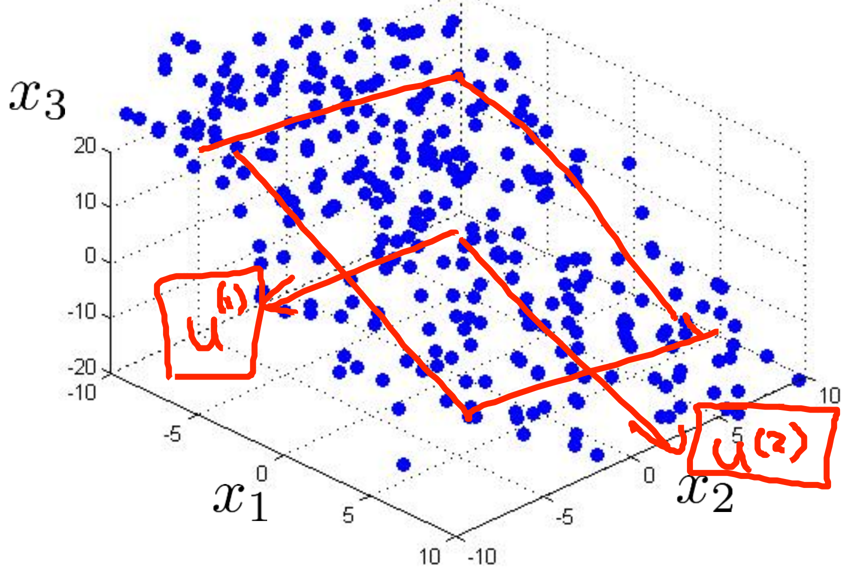
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

sj is some measure of the beta values of feature j. it could be the max minus min value, or more commonly, it is the standard deviation of feature j.

Principal Component Analysis (PCA) algorithm

we want to find out u1,u2 and z1,z2



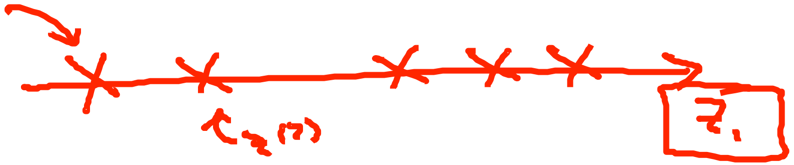


Reduce data from 2D to 1D

Reduce data from 3D to 2D

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

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



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

Andrew Ng

U and V are orthogonal and their inverse matrices are equal to their transposes

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

Sigma

Compute "eigenvectors" of matrix Σ :

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

n×n matrix

→ Singular value decomposition

eig(Sigma)

eig can also be used to compute the same thing. and It turns out that the SVD function give you the same vectors, although SVD is a little more numerically stable.

ui forms orthonormal basis

$$U_{n\times n} = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix}$$

$$\underbrace{u^{(1)}\; u^{(2)}\; u^{(3)}}_{k}$$

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

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

if we want to reduce the data from n-D to k-D: take the first k vectors.

SVD is a decomposition for arbitrary-size matrices, while EIG applies only to square matrices. They are very much related:

Principal Component Analysis (PCA) algorithm

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

$$\rightarrow U = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{k} \in \mathbb{R}^{n\times n}$$

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

$$z^{(i)} = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}}_{\substack{n\times k \\ U_{reduce}}}^{T} x^{(i)} = \underbrace{\underbrace{\begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix}}_{k\times n} \overset{\swarrow}{\underset{n\times 1}{x^{(i)}}}}_{k\times 1}$$

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

Andrew Ng

Principal Component Analysis (PCA) algorithm summary

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

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

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

mxn

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

nxm mxn

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

nxk

kx1

kxn

nx1

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

~~$x_0 = 1$~~

not using x0=1 convention

Andrew Ng

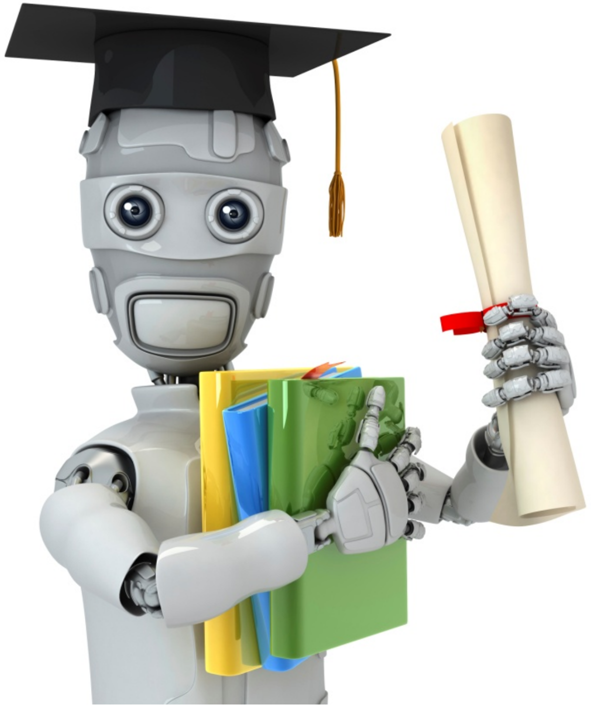
In PCA, we obtain $z \in \mathbb{R}^k$ from $x \in \mathbb{R}^n$ as follows:

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

Which of the following is a correct expression for z_j?

- ○ $z_j = (u^{(k)})^T x$
- ○ $z_j = (u^{(j)})^T x_j$
- ○ $z_j = (u^{(j)})^T x_k$
- [✓ $z_j = (u^{(j)})^T x$]

Correct



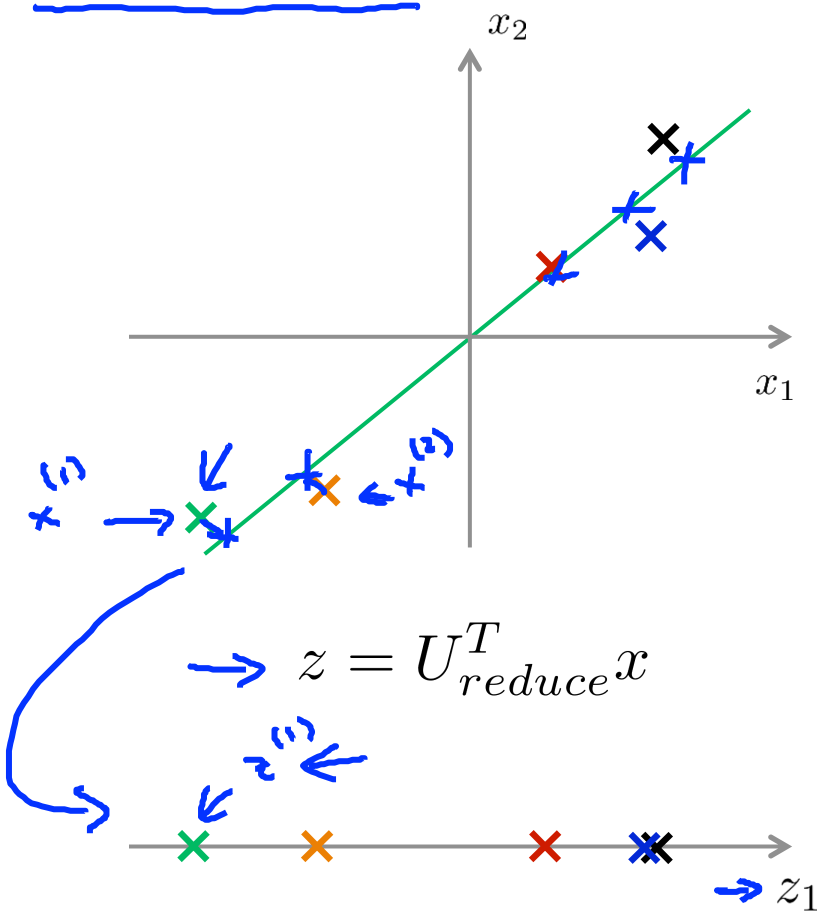
Machine Learning

Dimensionality Reduction

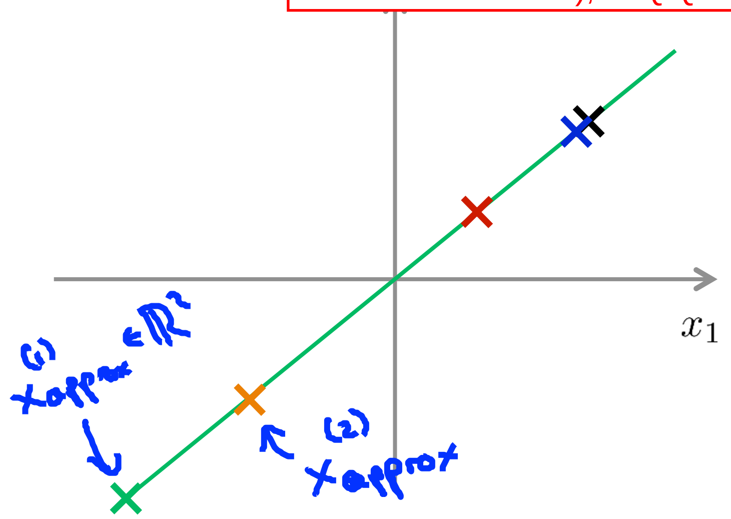
Reconstruction from compressed representation

Reconstruction from compressed representation

In linear algebra, an orthogonal matrix (or real orthogonal matrix) is a square matrix with real entries whose columns and rows are orthogonal unit vectors (i.e., orthonormal vectors),=> Q'Q = QQ' = I



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



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

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

Andrew Ng

Suppose we run PCA with k = n, so that the dimension of the data is not reduced at all. (This is not useful in practice but is a good thought exercise.) Recall that the percent / fraction of variance retained is given by: $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$ Which of the following will be true? Check all that apply.

- [✓] U_{reduce} will be an $n \times n$ matrix.

Correct

- [✓] $x_{\text{approx}} = x$ for every example x.

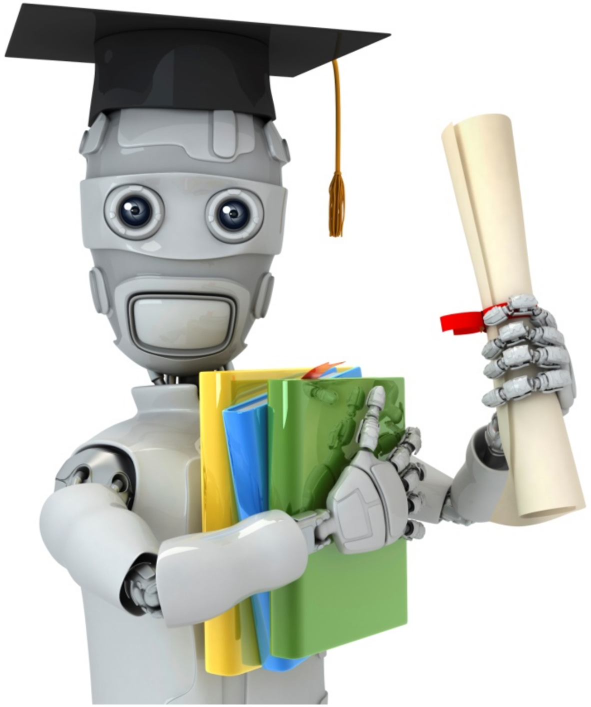
Correct

- [✓] The percentage of variance retained will be 100%.

Correct

- [ ] We have that $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} > 1$.

Un-selected is correct



Machine Learning

Dimensionality Reduction

Choosing the number of principal components

Choosing k (number of principal components)

PCA try to minimize

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

the average distance between x and it's projections

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

On average, how far are my training examples from the origin

Typically, choose k to be smallest value so that

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

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

0.10 (10%)

$\rightarrow$ "~~99%~~ of variance is retained"

~~95%~~ 90%

you say : I chose k so that 99% of the variance was retained.

For many data sets, in order to retain 99% of the variance, you can often reduce the dimension of the data significantly and still retain most of the variance. Because for most real life data many features are just highly correlated, and so it turns out to be possible to compress the data a lot and still retain 99% of the variance how to implement :

Choosing k (number of principal components)

Algorithm:

Try PCA with ~~$k=1$~~ $k=2$ $k=3$ $k=4$ …

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

Check if

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

$k = 17$

try k=1 to xx untill you reach this condition

its inefficient !!

The singular values are the diagonal entries of the S matrix and are arranged in descending order.

better way :

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

$$\rightarrow S = \begin{bmatrix} S_{11} & & & 0 \\ & S_{22} & & \\ & & S_{33} & \\ 0 & & & \ddots & \\ & & & & S_{nn} \end{bmatrix}$$

nxn

zeros

$k=3$

For given k

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

increase k so that

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

this way you only need to call SVD once

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)

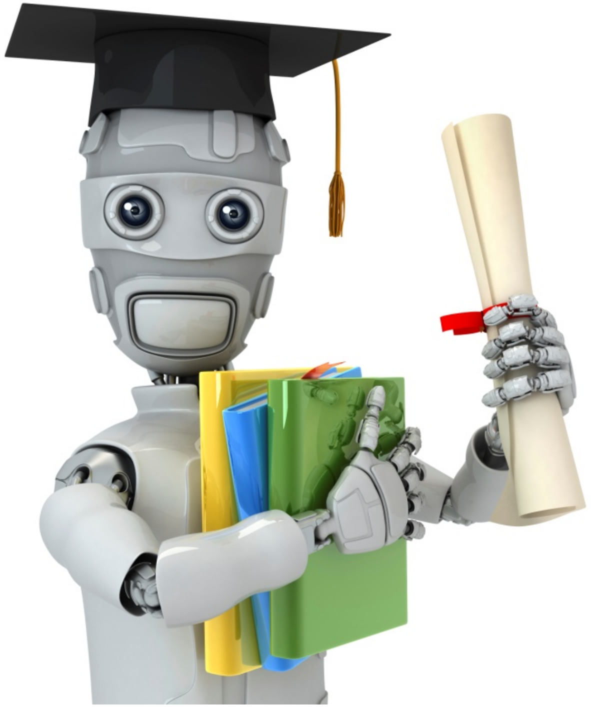
if you want to explain to others what you just did, a good way to explain the performance of your implementation of PCA to them, is actually to take this quantity and compute what this is, and that will tell you what was the percentage of variance retained. That's a measure of your square of construction error.

Previously, we said that PCA chooses a direction $u^{(1)}$ (or k directions $u^{(1)}, \ldots, u^{(k)}$) onto which to project the data so as to minimize the (squared) projection error. Another way to say the same is that PCA tries to minimize:

- ○ $\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)}||^2$
- ○ $\frac{1}{m}\sum_{i=1}^{m} ||x_{\text{approx}}^{(i)}||^2$
- [✓ $\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{\text{approx}}^{(i)}||^2$]

Correct

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



Machine Learning

Dimensionality Reduction

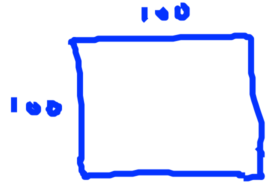
Advice for applying PCA

PCA can be sometimes used to speed up the running time of a learning algorithm, how to actually do that ? and some advice about how to apply PCA

Supervised learning speedup

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

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



Extract inputs:

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

$\downarrow PCA$

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

$$\begin{tikzcd} x \arrow[d] \\ z \end{tikzcd}$$

$\underline{U_{reduce}}$

New training set:

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

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

Note: $\underline{\text{Mapping}}$ $x^{(i)} \rightarrow z^{(i)}$ 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.

$\underline{x \rightarrow z}$

Application of PCA

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

Choose k by % of variance retain

- Visualization

$k=2$ or $k=3$

Andrew Ng

$\boxed{\text{Bad use of PCA}}$: To prevent overfitting

→ Use $\underline{z^{(i)}}$ instead of $\underline{x^{(i)}}$ to reduce the number of features to $\underline{\overset{1000}{k}} < \underline{n}$. ← 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.

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

Andrew Ng

PCA does not use the labels y, You are just looking at your inputs xi, and you're using PCA to find a lower-dimensional approximation to your data. So what PCA does, is it throws away some information or reduces the dimension of your data without knowing what the values of y is, so this is probably okay using PCA this way
if 99 percent of the variance is retained, but it might also throw away some valuable information. And using regularization will often give you at least as good a method for preventing over-fitting and regularization will often just work better, because when you are applying linear regression or logistic regression or some other method with regularization, this minimization problem actually knows what the values of y are, and so is less likely to throw away some valuable information, whereas PCA doesn't make use of the labels and is more likely to throw away valuable information.

PCA is sometimes used where it shouldn't be

we often start with this plan:

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 $\{(\cancel{z^{(1)}}\ x^{(1)}, y^{(1)}), \ldots, (\cancel{z^{(m)}}\ x^{(m)}, y^{(m)})\}$
- Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \ldots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

but

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 $\underline{z^{(i)}}$.

run too slowly, require too large memory or disk

Andrew Ng

Which of the following are good / recommended applications of PCA? Select all that apply.

- [✓] To compress the data so it takes up less computer memory / disk space

Correct

- [✓] To reduce the dimension of the input data so as to speed up a learning algorithm

Correct

- [ ] Instead of using regularization, use PCA to reduce the number of features to reduce overfitting

Un-selected is correct

- [✓] To visualize high-dimensional data (by choosing k = 2 or k = 3)

Correct