Machine Learning
Week 8
Section 2
Dimensionality Reduction

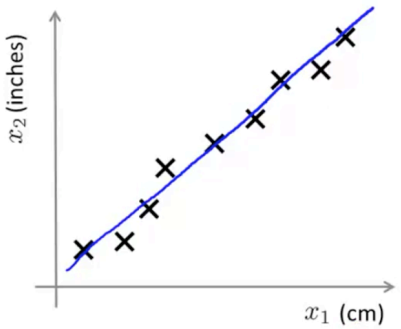
In this module, we look at an introduction to Principal Component Analysis and see how it can be used for data compression to speed up learning algorithms as well as for visualization of complex datasets.

Motivation 1: Data Compression

Lets look at another type of unsupervised learning problem called dimensionality reduction.

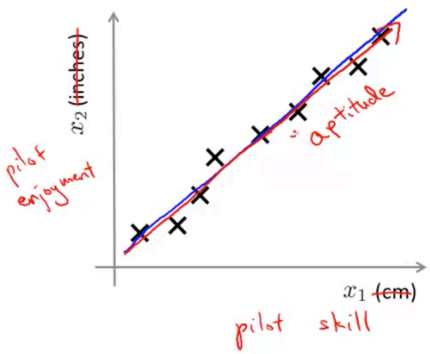
There are many good uses of dimensionality reduction, one is data compression.

Dimensionality Reduction / Data Compress example: Say for some data set we had a length feature that was in inches and one in centimeters that represented the same value, instead of having two separate features we can reduce it to one value to measure the length.



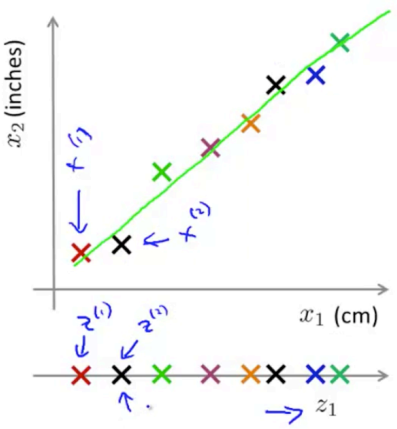
Reduce data from
2D to 1D

Another example in which two features don't necessary hold the same value but correlate an attribute could be pilot skill & enjoyment which could be indicating pilot aptitude:



Reduce data from
2D to 1D

Mathematically we do the following projection:



Reduce data from
2D to 1D

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

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

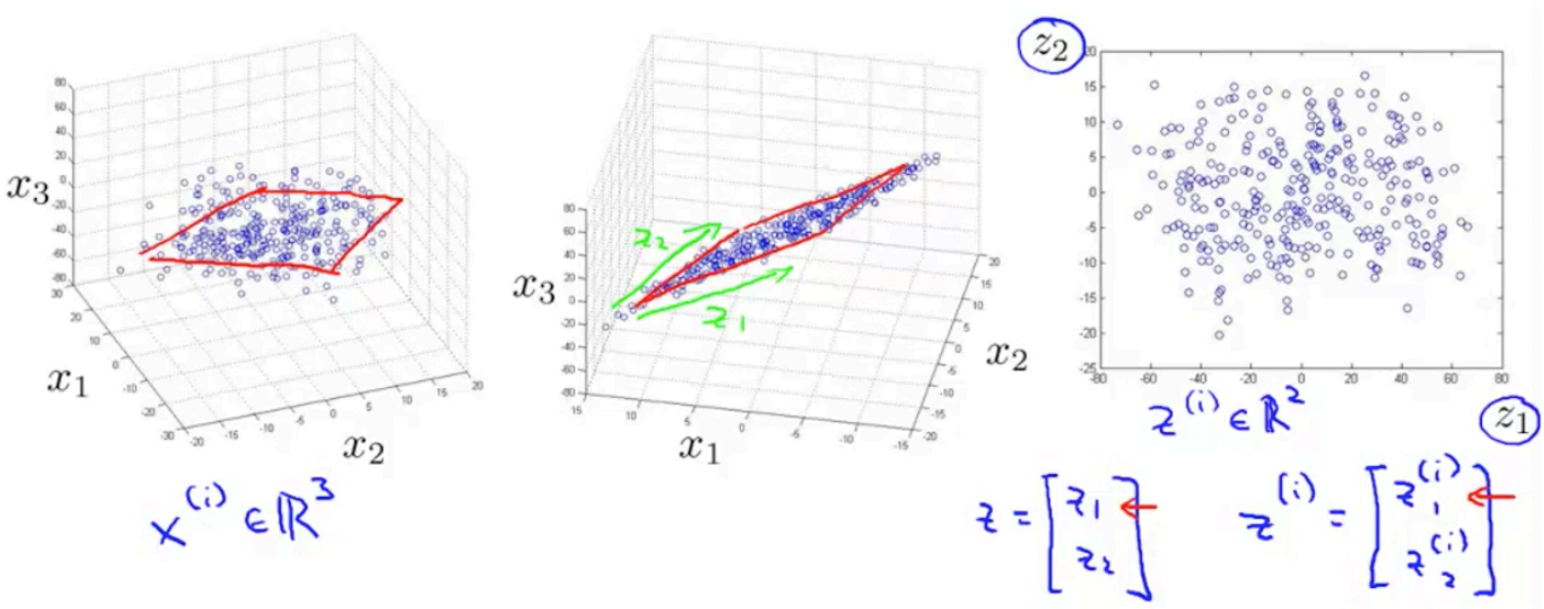
$\vdots$

$x^{(m)} \rightarrow z^{(m)}$

This halves the memory / space requirement for storing our data and allows for faster computation.

We can also reduce much higher dimensional data (1,000D -> 100D), but because of human limitations lets look at 3D -> 2D.

Maybe most of our data lies on a plane and can perform the following projection:



Motivation 2: Visualization

A second application of dimensionality reduction is data visualization.

Let us say we have a large dataset table of countries economic and population metrics, how can we visualize this data?

Data Visualization

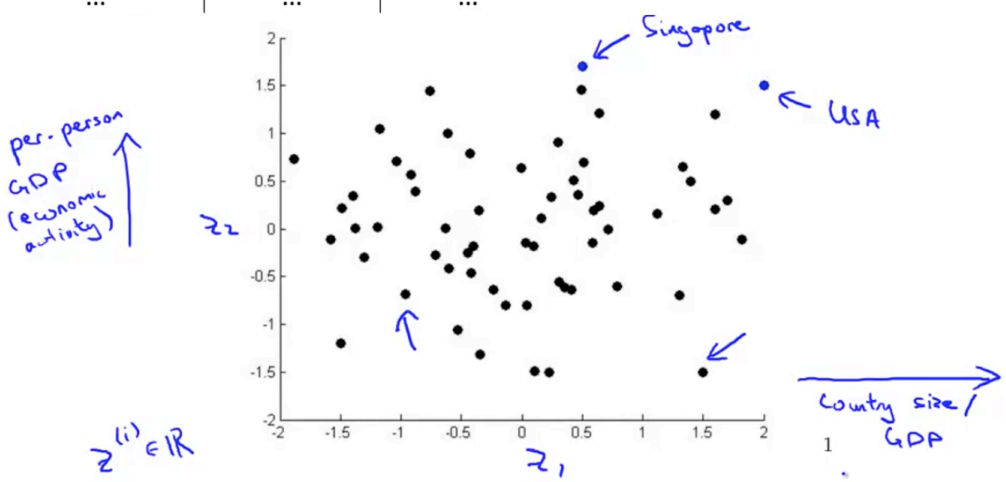
$x \in \mathbb{R}^{50}$ $x^{(i)} \in \mathbb{R}^{50}$

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	...
...	...	...	...	...	...	...	

Maybe we can come up with a different feature representation as follows in which we have a pair of numbers z1,z2 which summarize our 50 features, we can plot these countries in R2 (Reduce the data from 50D to 2D)

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

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 is up to us to interpret what z1,z2 are / represent and can be somewhat ambiguous.

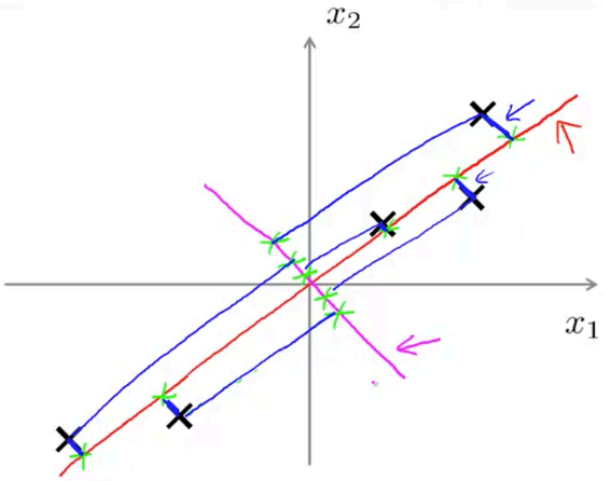
These types of visualizations may help us more succinctly capture what the main dimensions of variations are for our data.

In the next video we'll start to develop a specifically algorithm called Principal Component Analysis (PCA)

Principal Component Analysis Problem Formulation

The most commonly used algorithm for dimensionality reduction is PCA.

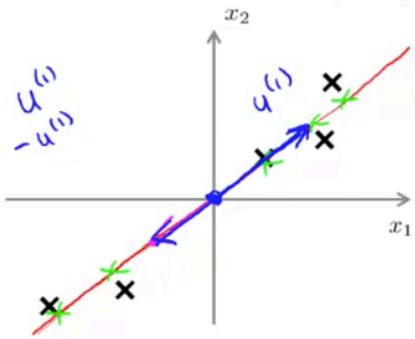
Let's say we have a data set in R2 which we want to project to R1. The red line would be the best line on which to project x1,x2 to minimize the projection error (blue lines). The magenta line would be terrible.

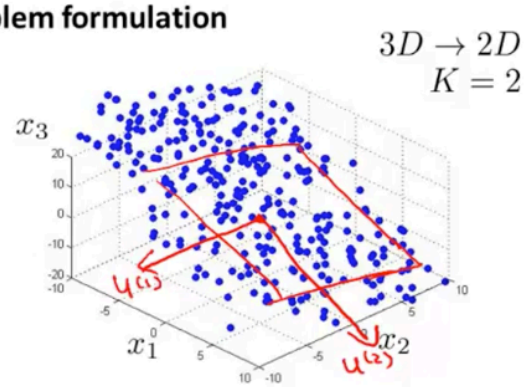


Also note, before applying PCA it is standard practice to first perform mean normalization and feature scaling so the features x1,x2 should have zero mean and have a comparable range of values.

More generally,

Principal Component Analysis (PCA) problem formulation





The sign of u doesn't matter.

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 trying to find a lower dimensional surface onto which to project the data so as to minimize the squared projection error.

In the next video we will discuss how to find this lower dimensional surface.

Principal Component Analysis Algorithm

Let us discuss the PCA algorithm.

First, before applying PCA, there is a data pre-processing step which we should always do.

We need to perform mean normalization and any necessary feature scaling.

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 $x_j - \mu_j$.

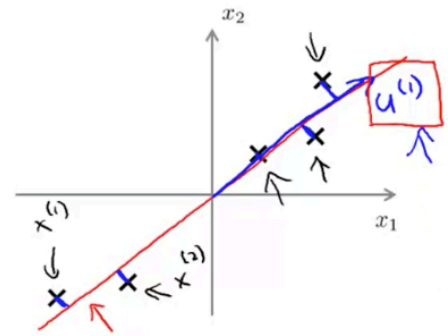
If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

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

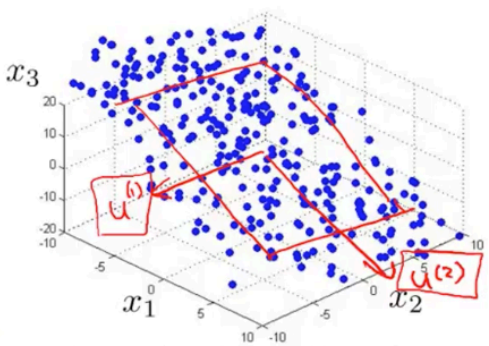
sj is the max minus min value or standard deviation of feature j.

Once this data preprocessing is done, the PCA algorithm is as follows:

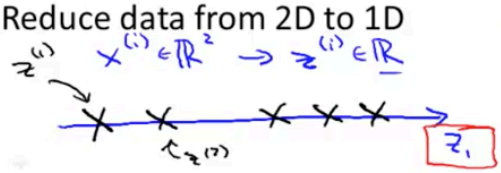
We are trying to find a lower dimensional subspace onto which we project the data so as to minimize the squared projection error.



Reduce data from 2D to 1D



Reduce data from 3D to 2D



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

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

PCA needs to find/compute the vector(s) which define the subspace / surface, and then lastly the new values / vectors z1,z2, etc...

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

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

Sigma

Compute "eigenvectors" of matrix Σ:

→ Singular value decomposition

eig (Sigma)

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

nxn matrix.

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

k

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

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

** Note, it is really obnoxious notation, but the sigma on the left is the Covariance Matrix (NxN), while the LHS is a sum.

svd == singular value decomposition

eig does the same thing and returns the Eigen vectors as well but svd is 'more' numerically stable.

U == (NxN matrix) of u vectors

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

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

k

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

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

$z \in \mathbb{R}^k$ $\quad$ $n \times k$ $\quad$ U_{reduce} $\quad$ $k \times n$ $\quad$ $n \times 1$ $\quad$ $k \times 1$

To summarize the algorithm:

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;
```

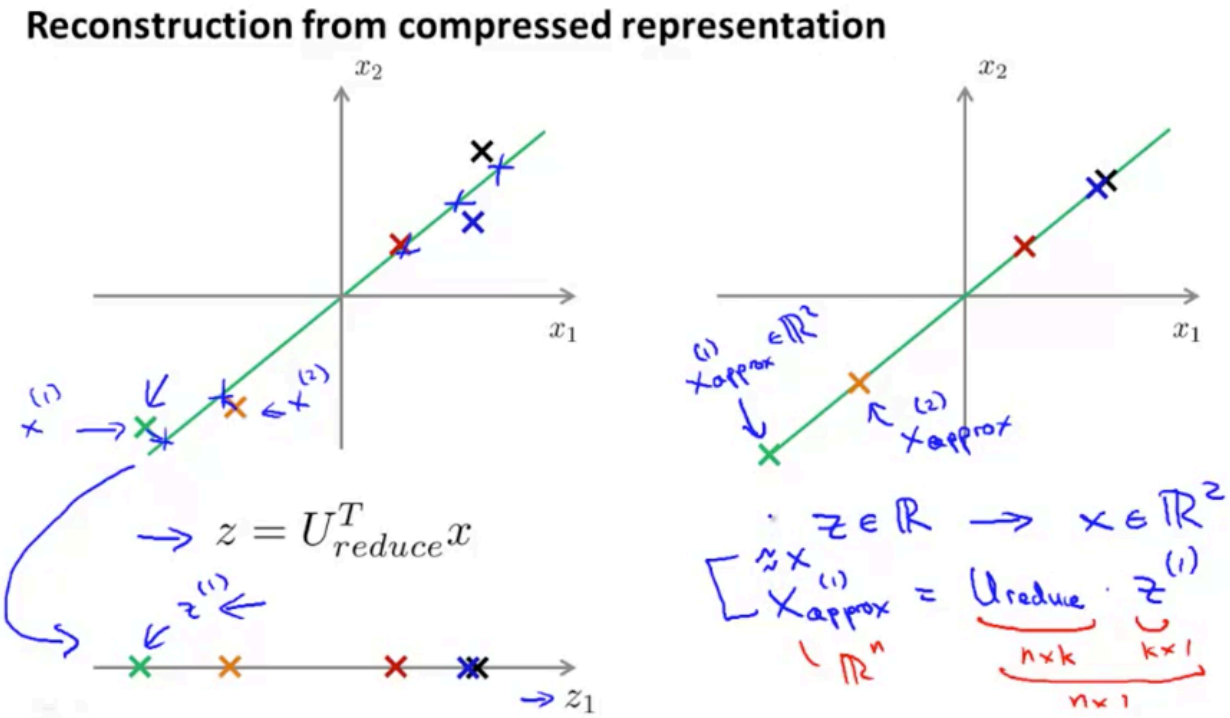
$x \in \mathbb{R}^n$ $\quad$ ~~$x_0 = 1$~~

Reconstruction from Compressed Representation

Previously we mentioned that PCA could be a compression algorithm. Ex. Going from 1000 dimensional data to 100 dimensional data.

If that's the case, what is the way to 'go back' from the compressed representation to an approximation of the original high-dimensional data?

Ex. Given a point z1, how do we go back to the x1,x2 space?



We get back an approximation to the original data.

Choosing the Number of Principal Components

In the PCA Algorithm we take N dimensional features and reduce them to some K dimensional feature representation.

K is a parameter of the PCA Algorithm and is called the Number of Principal Components.

How do we choose this parameter K?

Recall PCA tries to minimize the squared projection error.

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

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

0.05 0.10 5% (10%)

"99% of variance is retained"

95% 90%

Typically we want the variance retained to as high as possible while also accomplishing the task of reducing the dimensionality (so it's a bit of a trade of, you have to feel it out.)

We implement this as follows: We start with a small 'K' and work upwards.

Obviously this approach is not that efficient but luckily from the 'svd' function we get the vector 'S' which is a square matrix and can do the following:

Choosing k (number of principal components)

Algorithm:

Try PCA with $k = 1$ $k=2$ $k=3$ $k=4$ ...

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

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

$k = 17$

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

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

For given k

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

$k = 3$

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

Andrew

To summarize, to choose 'k'

Choosing k (number of principal components)

→ `[U,S,V] = svd(Sigma)`

Pick smallest value of k for which

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

k=100

(99% of variance retained)

Typically when reducing ML datasets, we see that we are able to retain a high variance as many features in these datasets are correlated. (95%-99%)

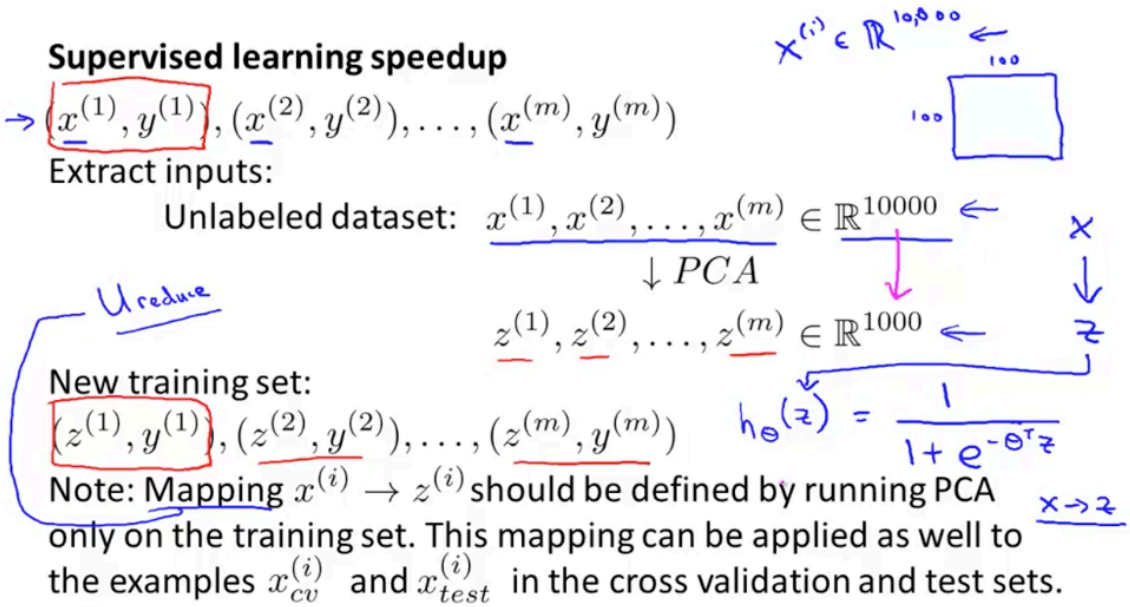
Advice for Applying PCA

PCA can sometimes also be used to speed up the running time of a learning algorithm.

How does this work and how can we do it?

For Dr. Ng, He often uses PCA to speed up supervised learning algorithms.

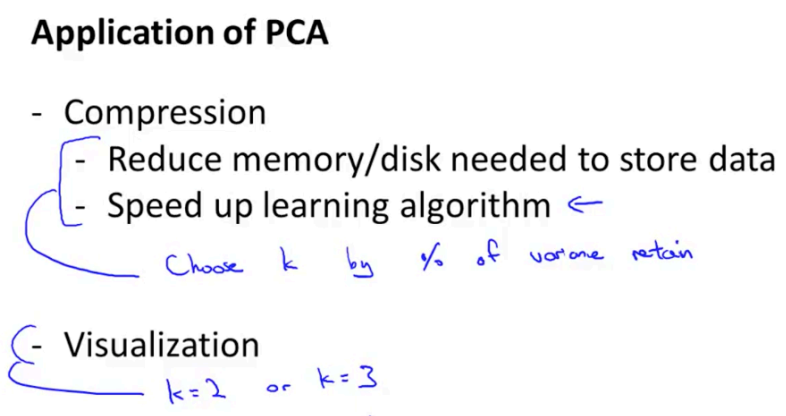
For some supervised learning algorithm & dataset (an image task w/ 100 x 100 pixel images)



Only run PCA on the training dataset.

** Note I am not clear on how to map the y labels back down to the reduced inputs z...

To summarize our applications on PCA



Bad use of PCA: One frequent misuse of PCA is when people try to reduce the number of features in a dataset using PCA in order to try to prevent overfitting.

This is because we run PCA on the inputs 'X' to reduce their dimension as if it were an unlabeled dataset without knowing what 'y' is. It could throw out some valuable information.

There are some situations in which you can use PCA in this way, (cases in which you retain very high variance) but otherwise is not recommended.

It is better to use regularization instead.

Other cases where PCA is used incorrectly:

Ex. A 'Flawed' Project Plan

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 and consider using $z^{(i)}$.