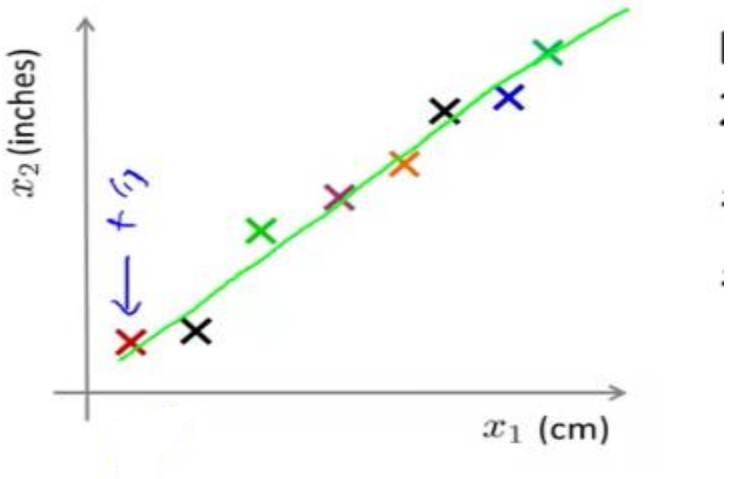
Data Compression:

Data compression is reducing data from a higher dimension to a lower dimension. Example: from 1000 dimensions to 100 dimensions or from 3D to 2D or from 2D to 1D.

Consider the data given below:

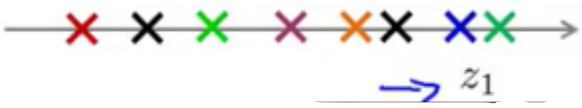


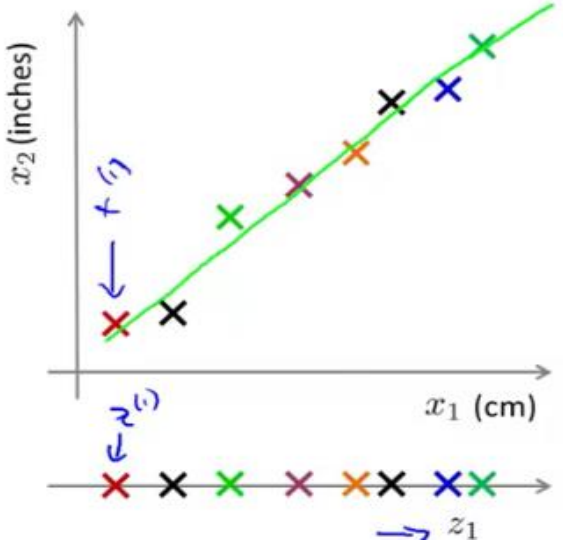
Example 1:

We have 2-dimensional data plotted on the graph. Though the data should line on a straight line it doesn't due to rounding off that has been done. Now we want to reduce this data to 1D or simply a line.

Note: x^1 represents first row of the data. There are 2 columns x_1 and x_2 in the data.

To convert this data to 1D we take projections of x^1, x^2….. x^m and then obtain the line on which all the projections fall. This line is called z_1 and has only 1 dimension.





Reduce data from 2D to 1D

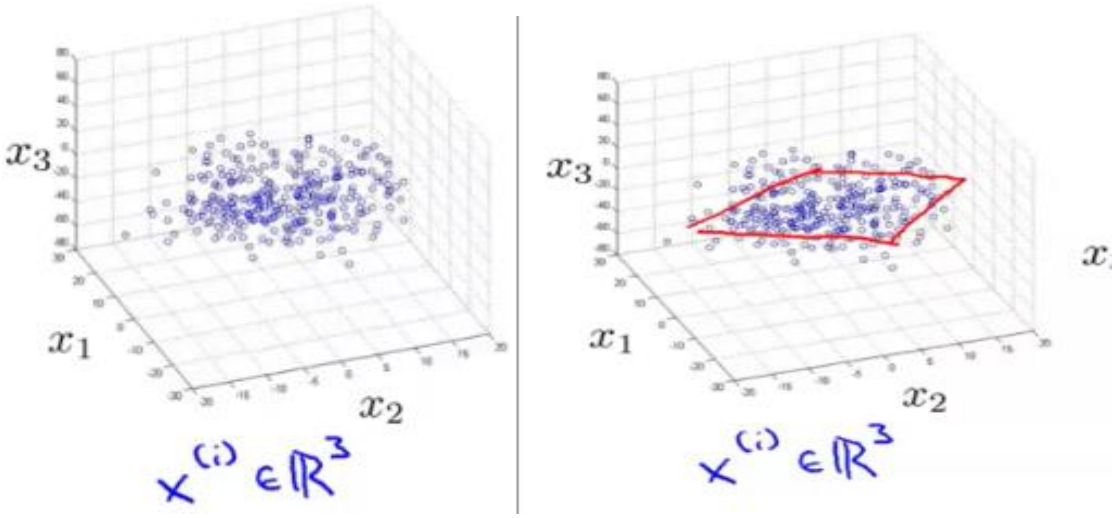
$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$

$x^{(2)}$

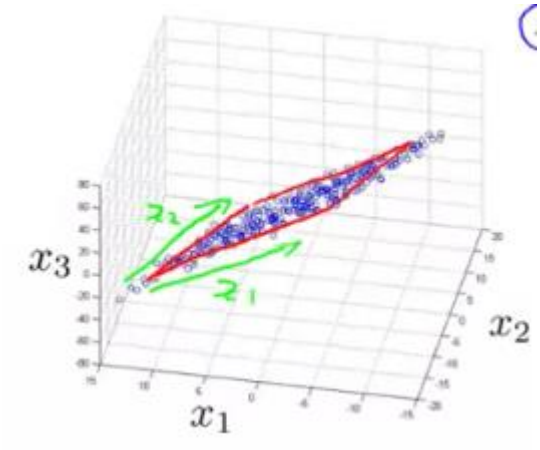
Example 2:

Reducing data from 3D to 2D

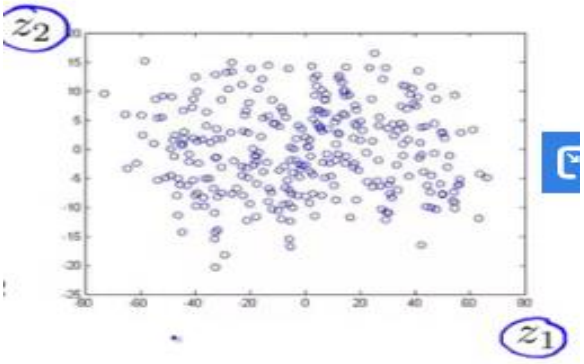
Consider the dataset given below:



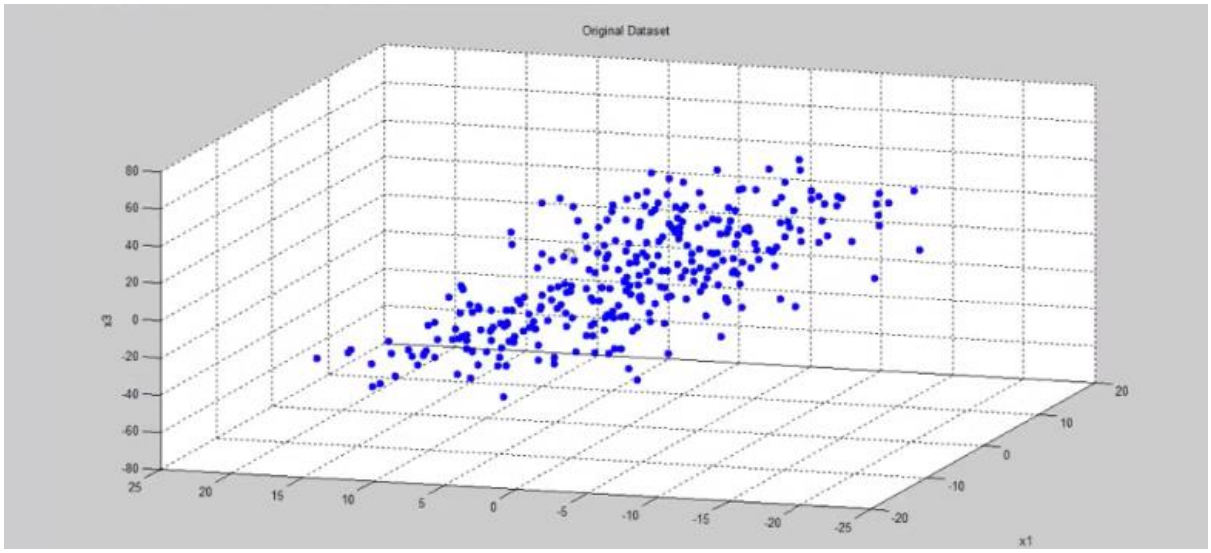
Project the dataset on a plane:



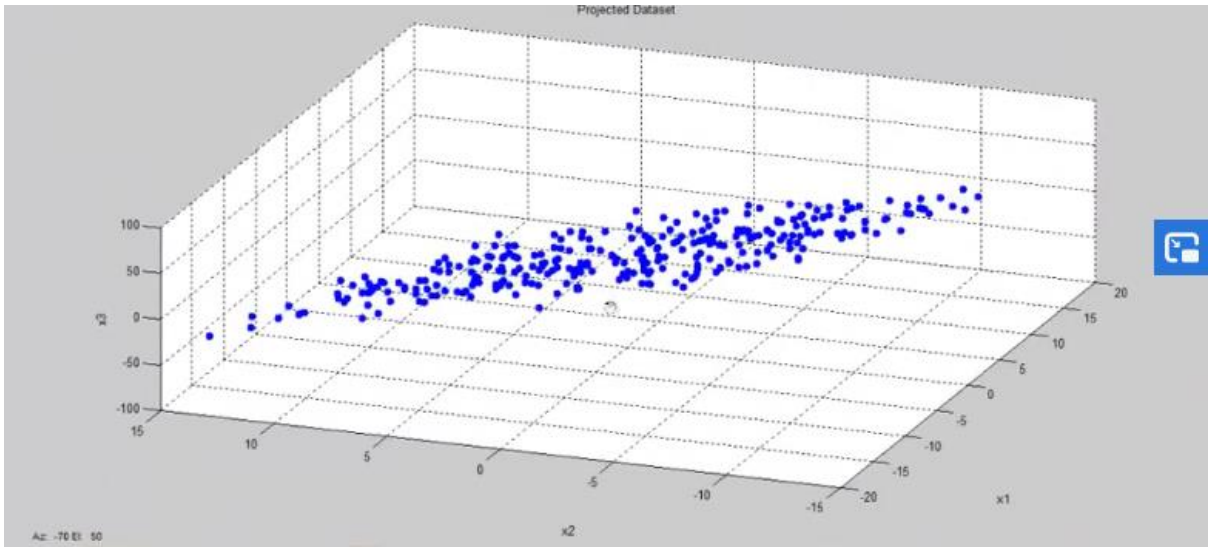
2D image of the plane:



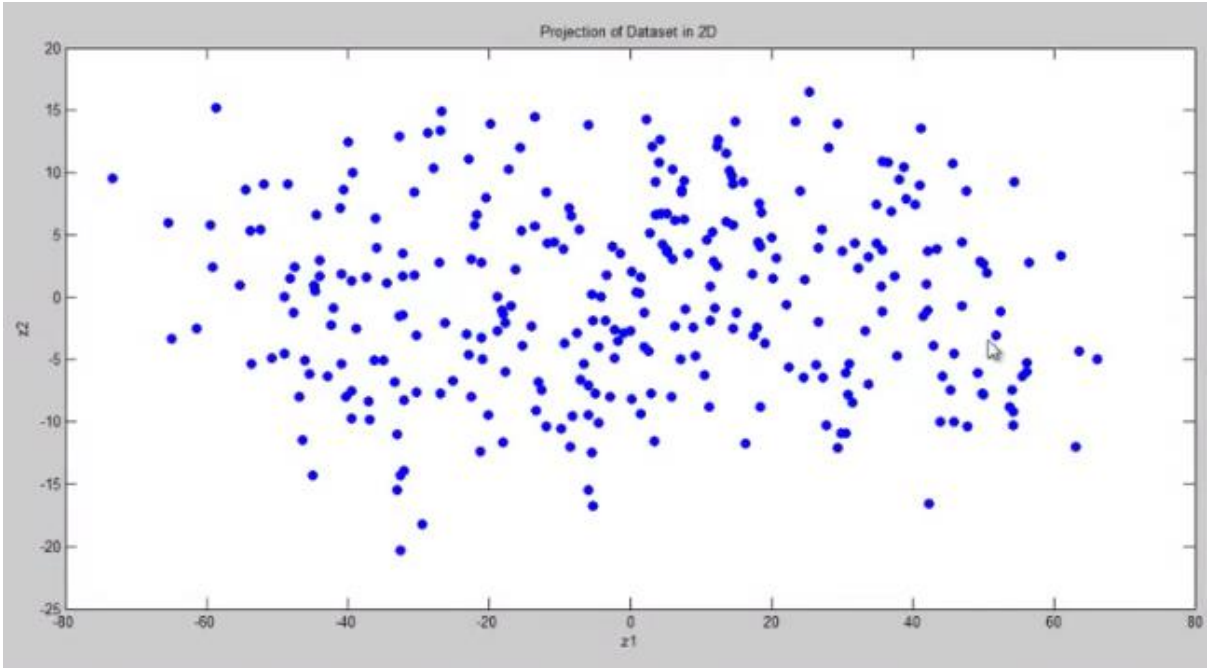
3D representation of the above example:



Projecting the data on single plane:



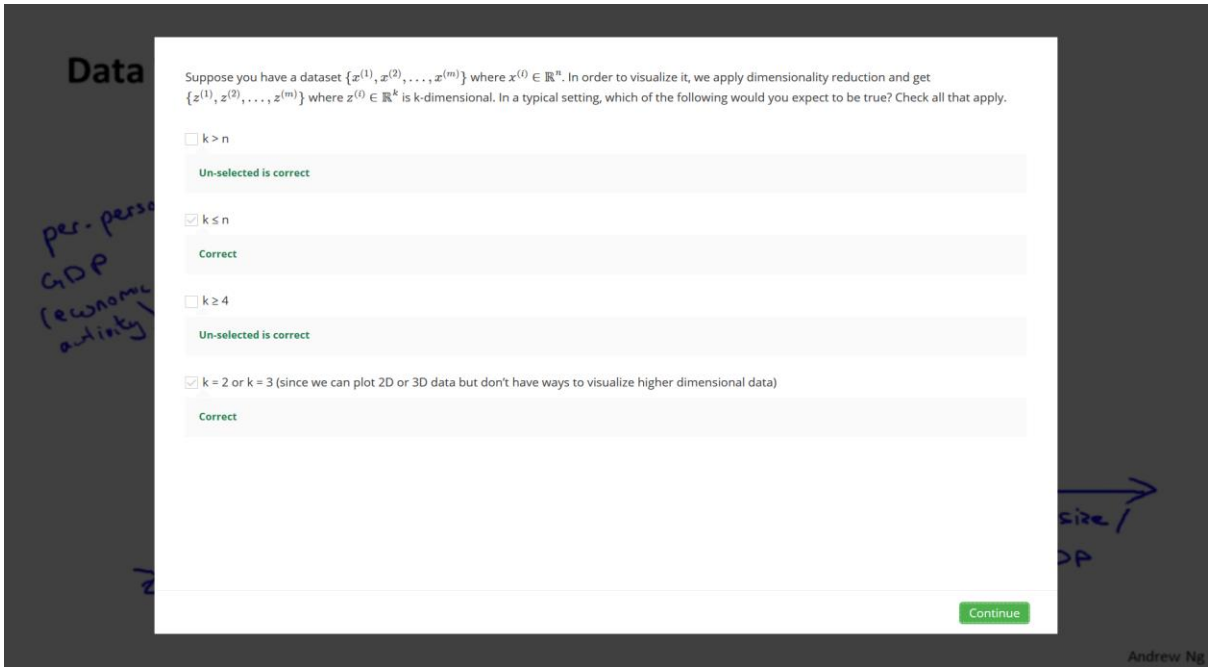
Projected dataset in 2D:



Questions:

Q1:

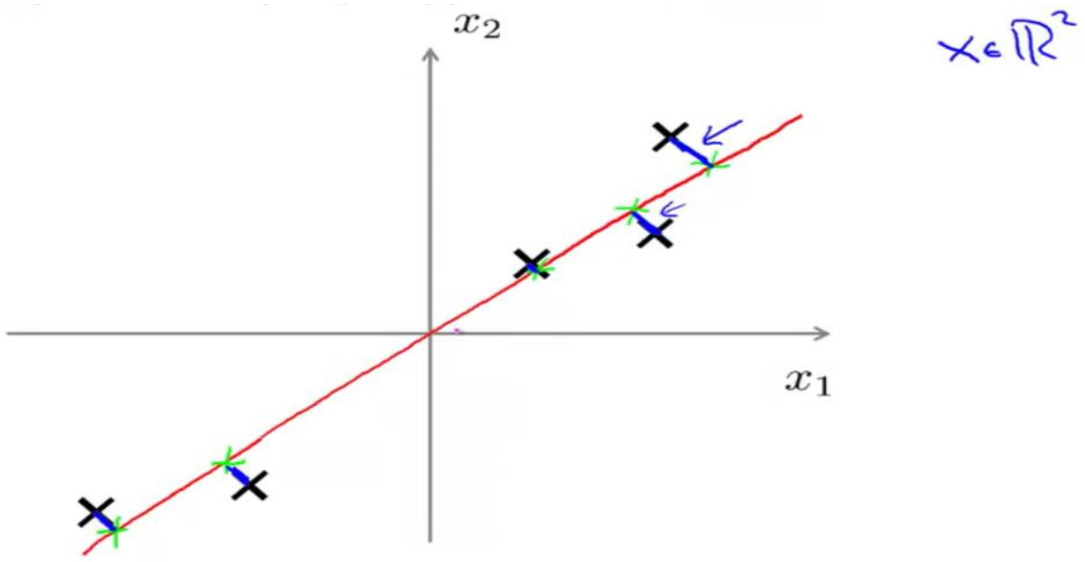
Q2:



Principal Component Analysis:

Principal component analysis is used to find the line/plane/any other $(n-1)^{th}$ dimensional component of a n dimensional data from which the points have the minimum perpendicular distance. Using PCA, we try to minimize the square of the perpendicular distance between the points and the $(n-1)^{th}$ dimensional surface.

Consider that we have a 2D dataset and want to convert it into 1D. PCA tries to find the best possible line to do this job. Now, the best possible line is the one from which the points have the minimum perpendicular distance:

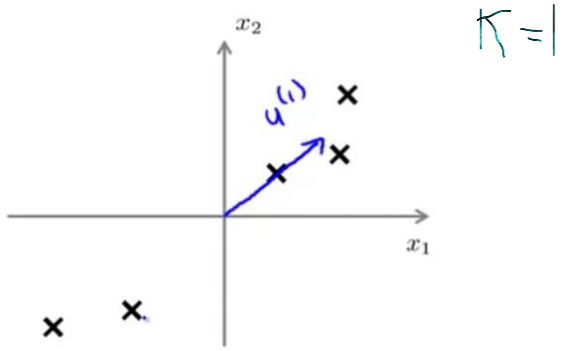


The lines projecting from the points to the red line depict the perpendicular distance of those points from the red line.

To find the best possible lower dimension surface we use vectors:

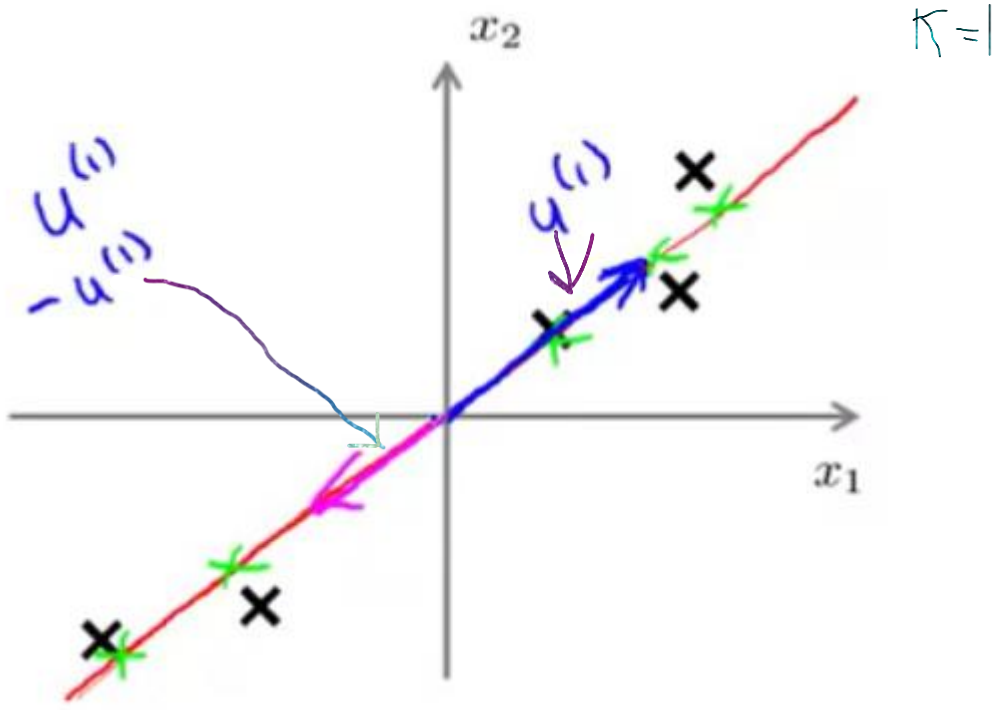
Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Example1: For example, in reduction of 2D to 1D, to find the best possible line, we take a vector u^1 which when extended back and forth forms the line we are looking for:

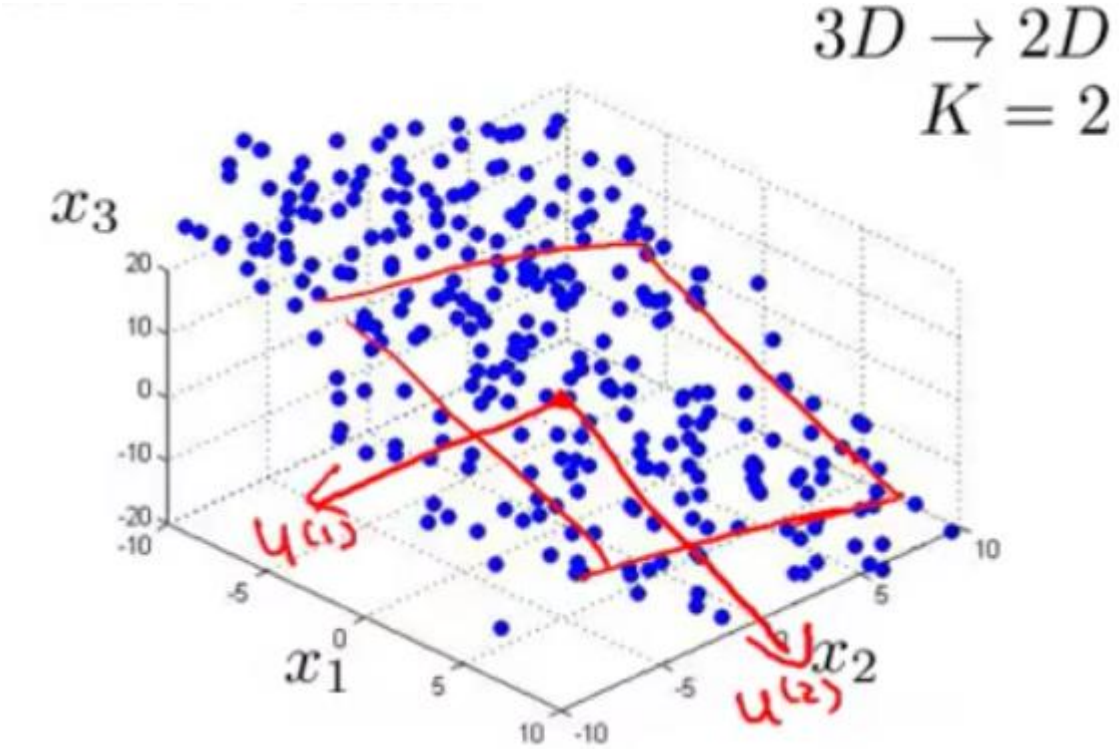


Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

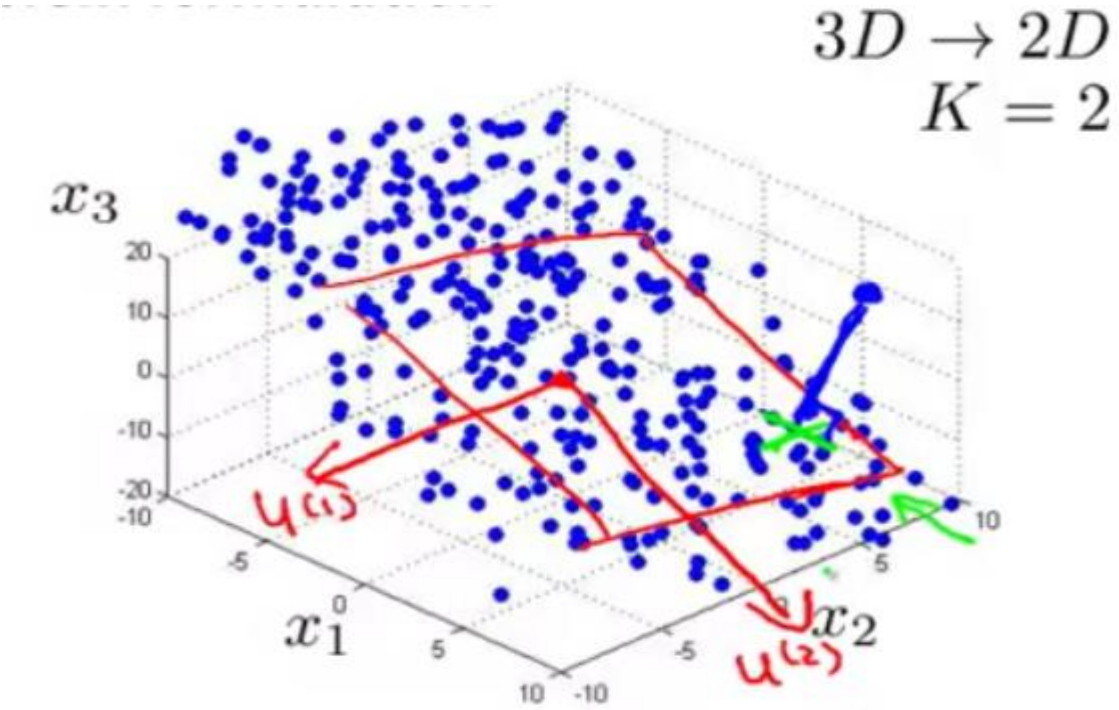
Note: Using $-u^1$ instead of u^1 does not matter since we have to extend the line in both directions.



Example 2: For example, in reduction of 3D to 2D, to find the best possible plane, we take vectors u^1 and u^2 which when extended back and forth form the plane we are looking for:



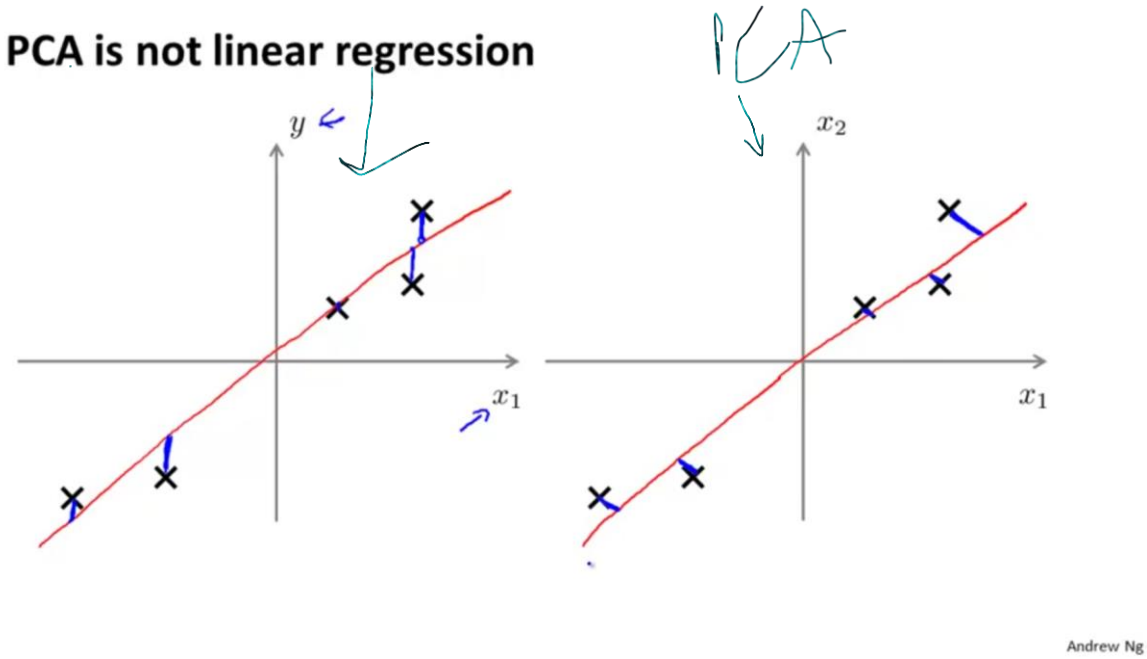
NOTE: We are trying to minimize the perpendicular distance of the points from the plane:



Difference between PCA and Linear Regression:

Though PCA seems to be the same as linear regression, the two have substantial differences:

1. In PCA, we minimize square of the perpendicular distance between the points and the lower dimension surface. Whereas in Linear Regression, we try to minimize the vertical distance between the points and the lower dimensional surface:



2.When you're doing linear regression, there is this distinguished variable y that we're trying to predict. In linear regression we take all the values of x and try to use that to predict y. Whereas in PCA, there is no distinguish, or there is no special variable y that we're trying to predict. And instead, we have a list of features, x_1, x_2…. and so on up to x_n , and all of these features are treated equally, so no one of them is special.

Applying the PCA Algorithm:

1. First, we perform 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.

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

2. Then we use the PCA algorithm by doing the following steps:

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

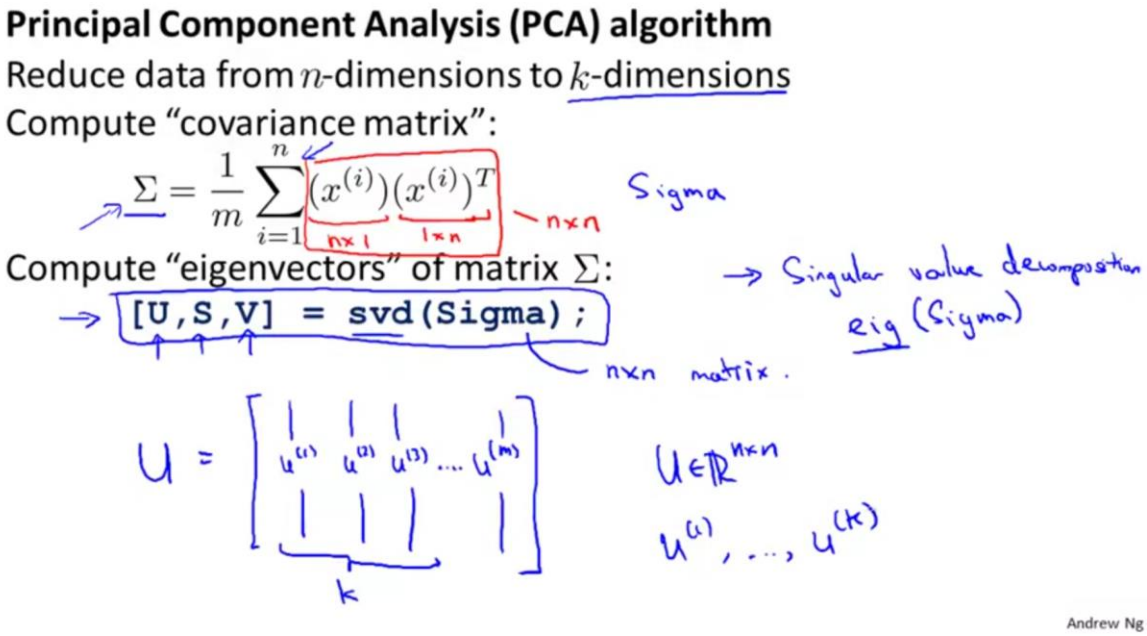
$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$ ($n \times 1$, $1 \times n$, $n \times n$)

Compute "eigenvectors" of matrix Σ:

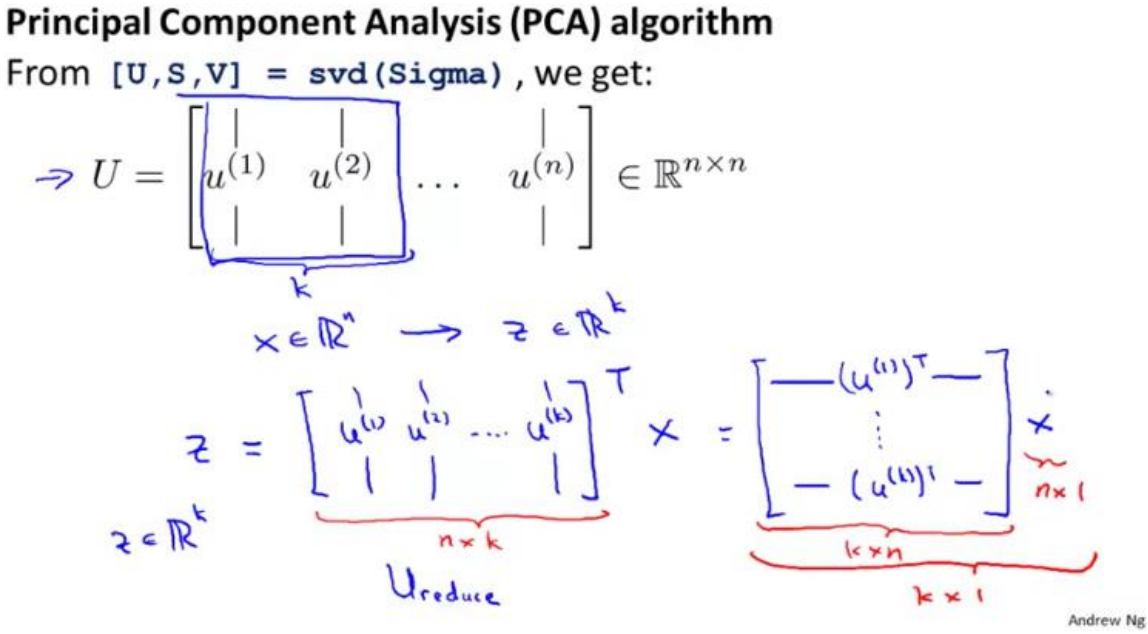
```
[U,S,V] = svd(Sigma);
```

Note: above is the code for computing eigenvectors in octave. We can also use eig(Sigma) instead of svd(Sigma).

Note: u which represents the vector was mentioned in the above pages, is a n*n dimensional vector. To obtain our lower k dimensional vector, we choose k columns from u.



We obtain values of z which is a k dimensional vector but multiplying the transpose of new u with n rows and k columns with X which is a n*n matrix (Note: the new vector is names as u_{reduce}):



Summary of using PCA:

Principal Component Analysis (PCA) algorithm summary

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

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

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

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

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

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

Question:

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

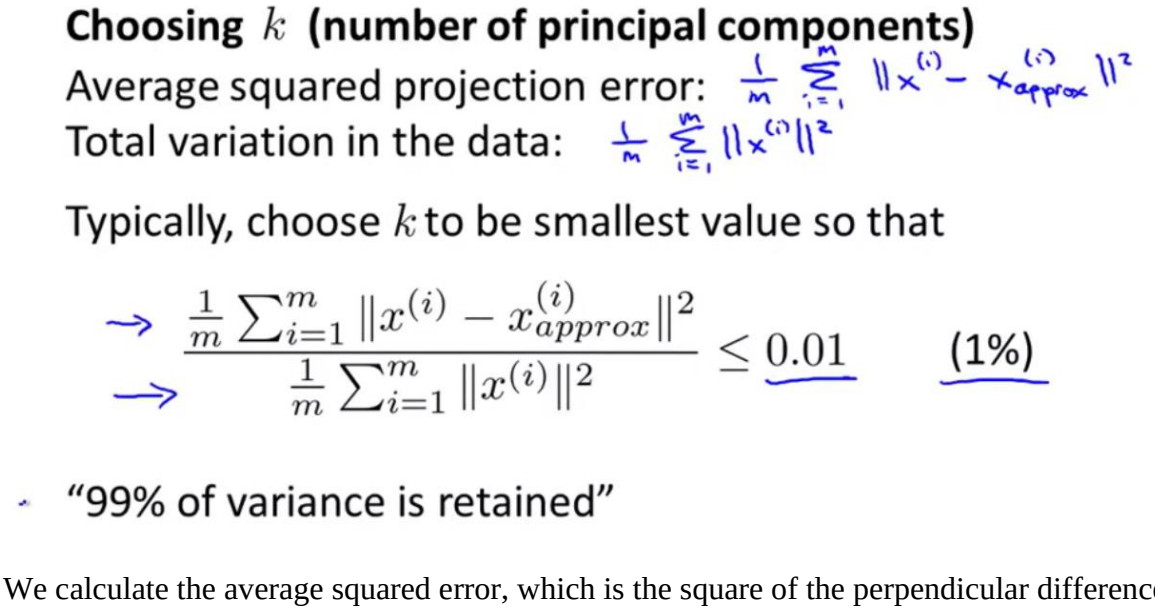
$$z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x = \begin{bmatrix} --- & (u^{(1)})^T & --- \\ --- & (u^{(2)})^T & --- \\ & \vdots & \\ --- & (u^{(k)})^T & --- \end{bmatrix} x$$

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

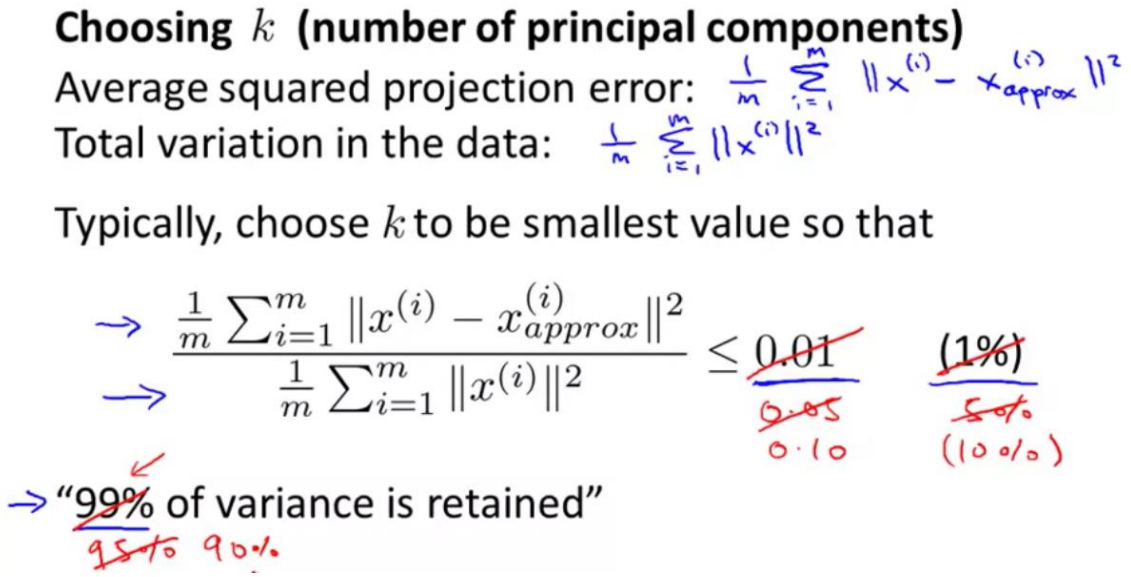
Choosing the value of K:



We calculate the average squared error, which is the square of the perpendicular difference between the actual x values and their projection.

Variance gives us an idea of how many features we have, more features => more variance. 99% variance retained simply means that even though we have used PCA to reduce the number of features, still 99% of the original variance is obtained.

So the 0.01 or 1% is the limit of variance sacrifice we have set ourselves meaning that we can allow maximum 1% variance to get removed in the process of data compression. Now, this limit can be 0.05 or 5% or 10% also:



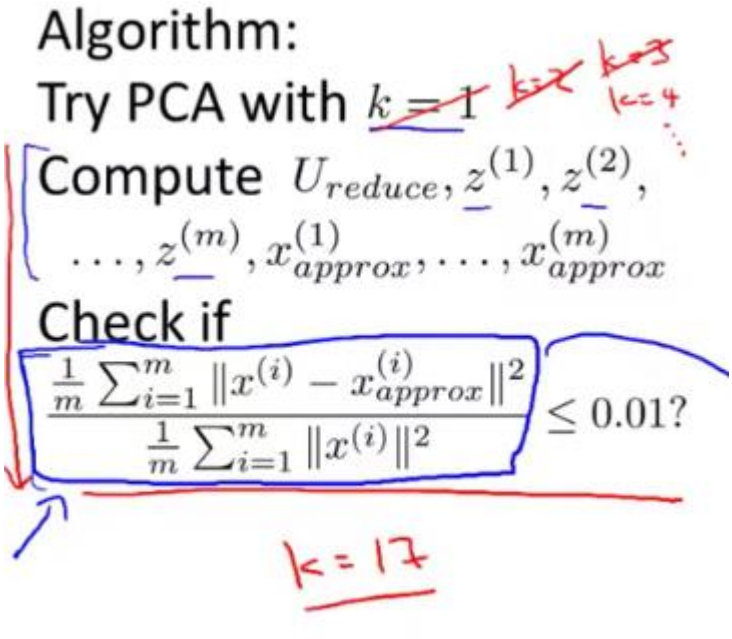
Methods for choosing k values:

We talk about 2 methods that can be used to get the best suited value of k (the one that minimizes average squared projection error).

Method-1:

We start from k=1 and keep on increasing the value of k until we get the one that gives us average squared projection error <=0.01.

With each new value of k , we calculate the n*k matrix u_{reduce} and then calculate average squared projection error.

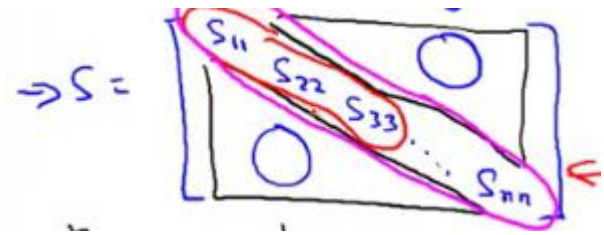


Let us assume that at k=17, we got average squared projection error<=0.01.

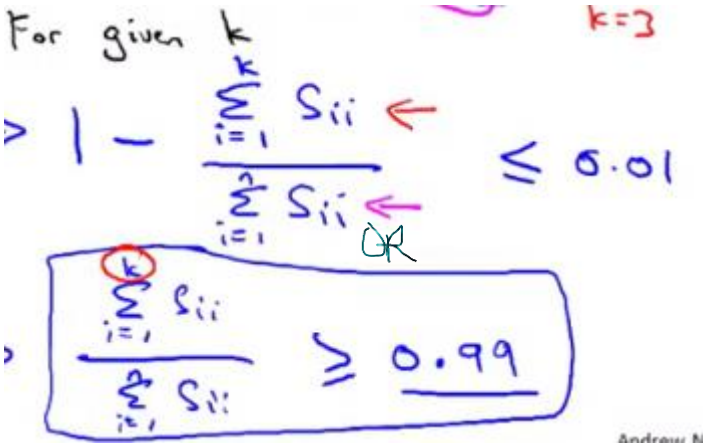
Method-2:

In this method we use the variable S which is obtained from svd(Sigma).

Note: The S matrix is a diagonal matrix and looks like this:

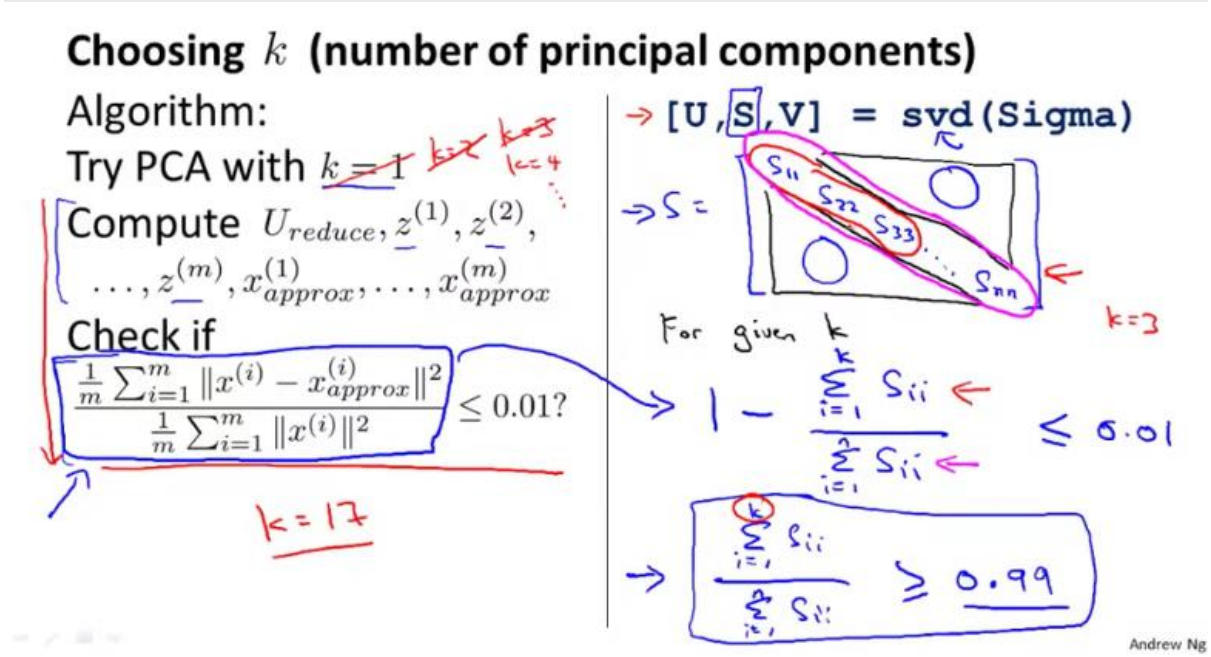


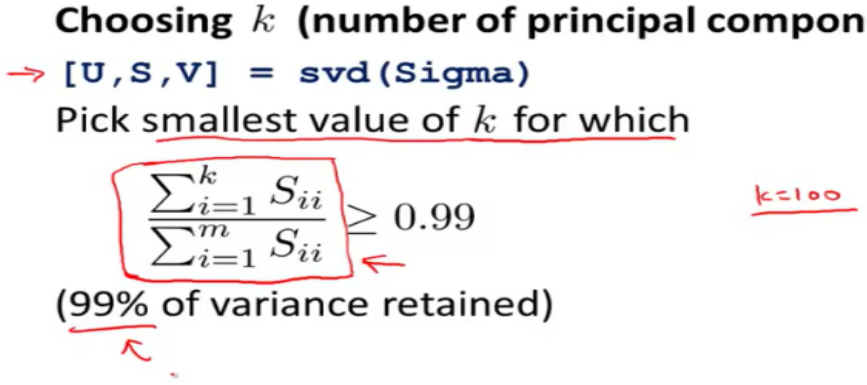
Now, we have to calculate the S matrix only once and its diagonal has all the values ranging from S_{11} to S_{nn}. We start from k=1 and keep on increasing k until for a given k:



Method 2 is faster than method 1 because in method 2 we don't have run PCA and calculate things over and over again, since the diagonal of S matrix has all the values ranging from S_{11} to S_{nn}.

To summarise:





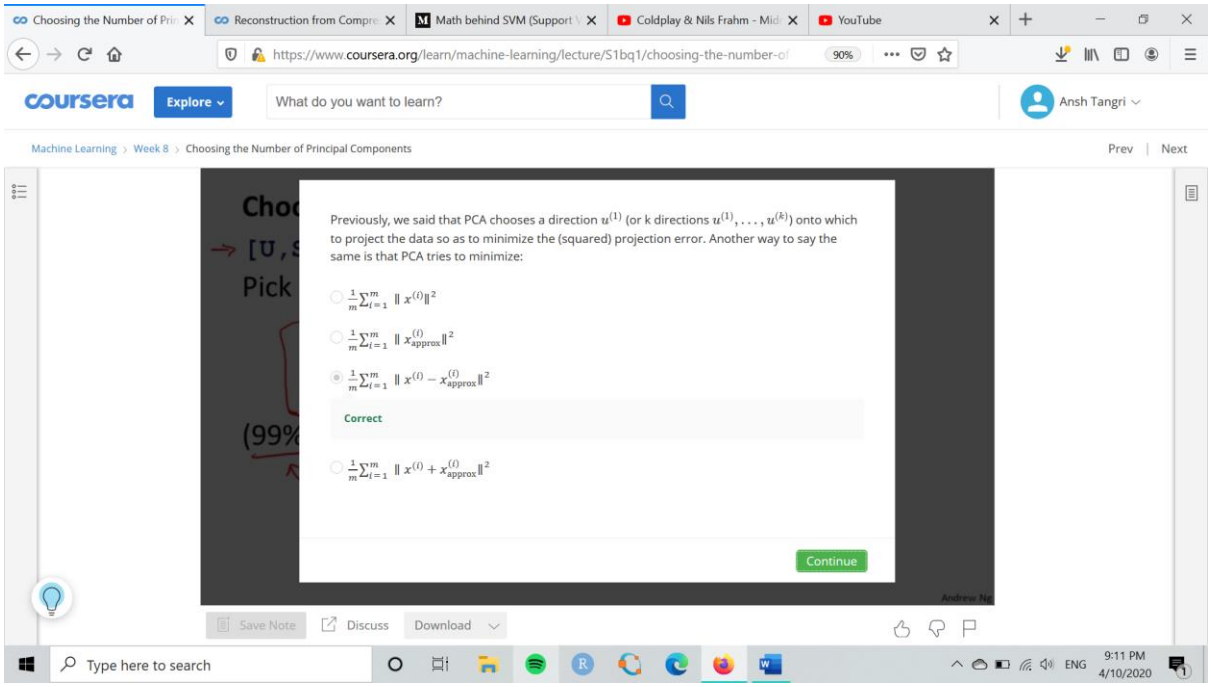
Even if we choose value of K manually, we can show the credibility of the chosen value by calculating:

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

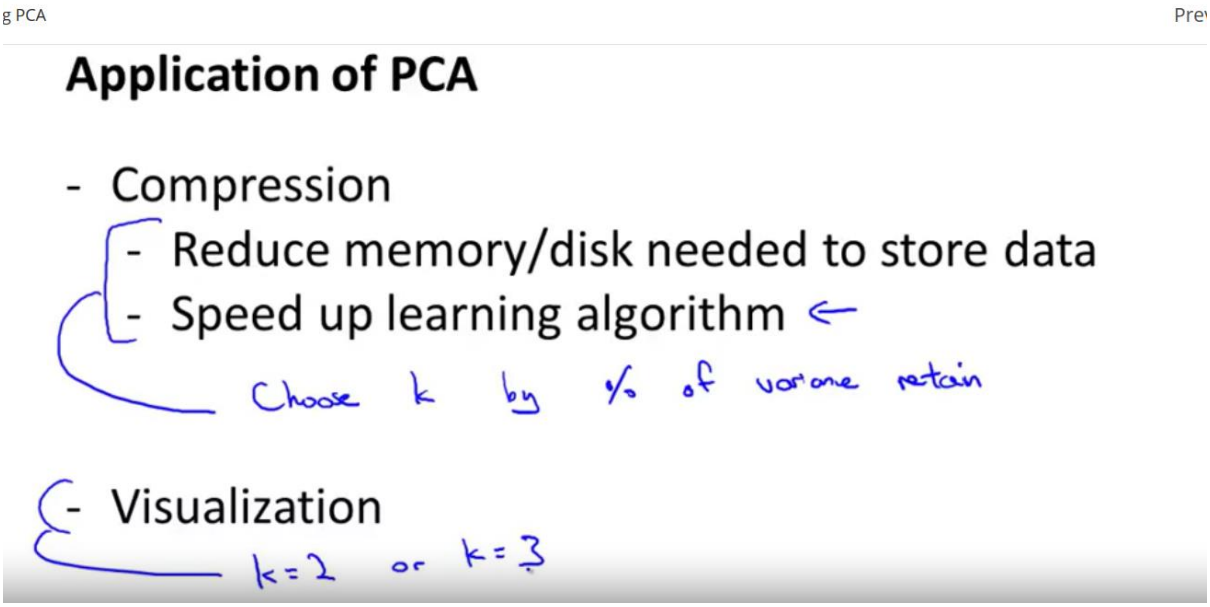
And checking if:

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

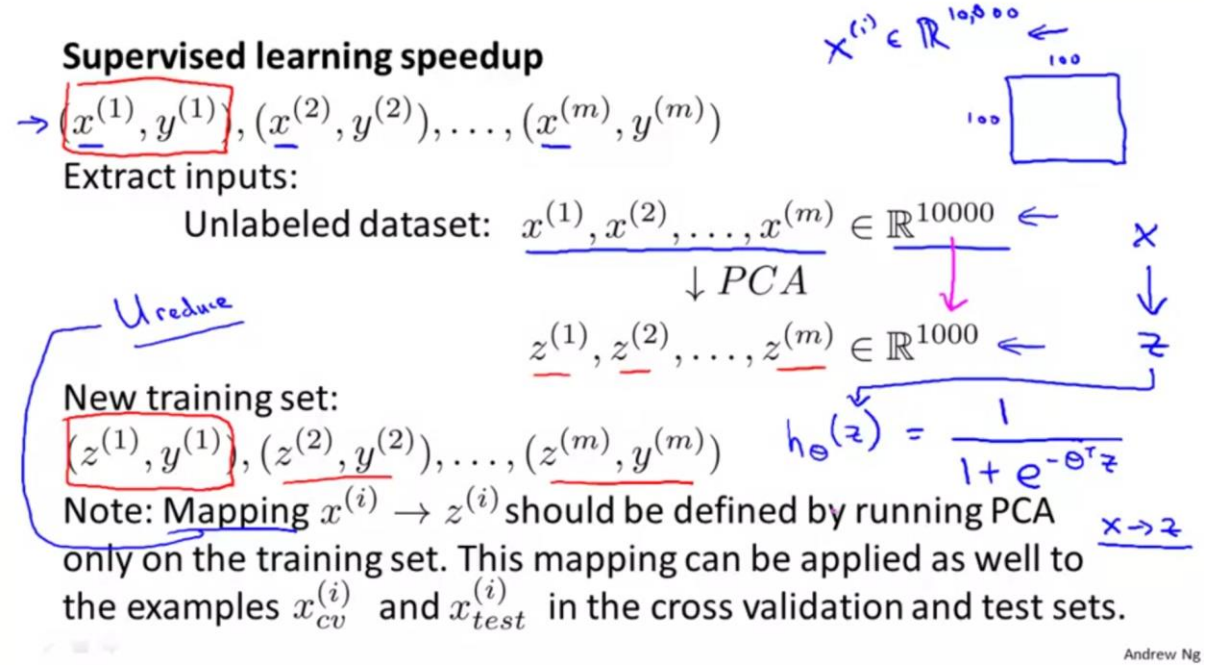
Question:



Applications of PCA:



1. Improving the speed of a supervised learning algorithm.

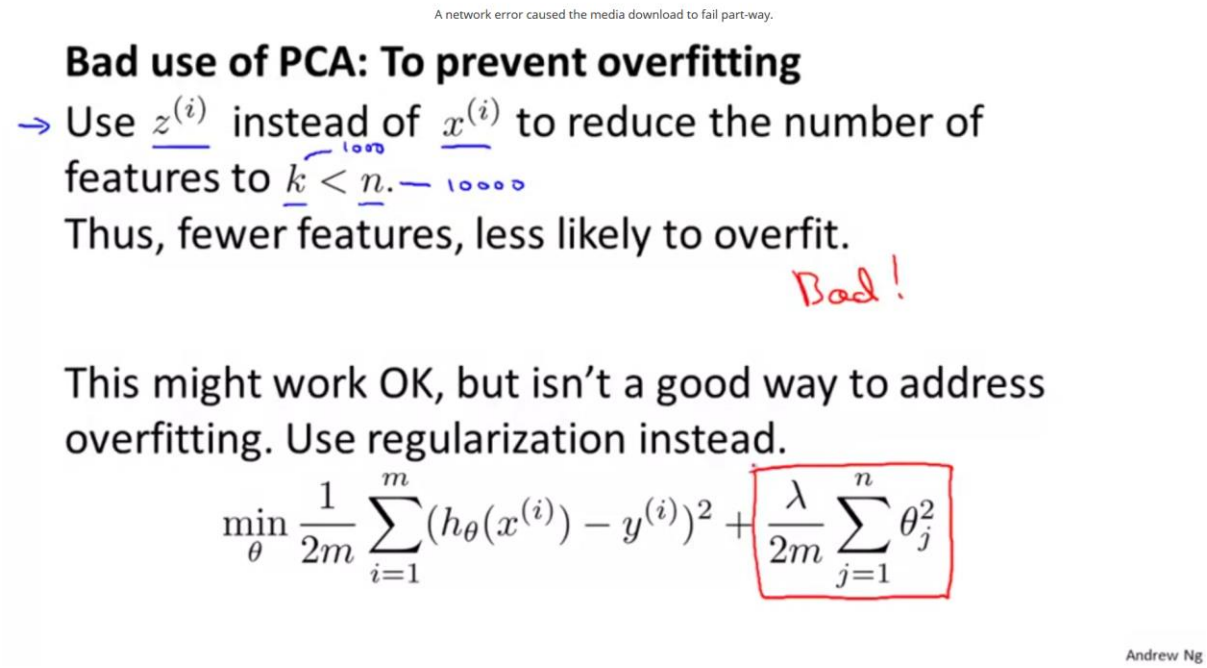


Note: The Mapping obtained from running PCA on the training set is applied on the testing and cross validation sets and not vice-versa.

2. We can reduce high dimensional data to 2D/3D, to effectively visualize it.

Bad uses of PCA:

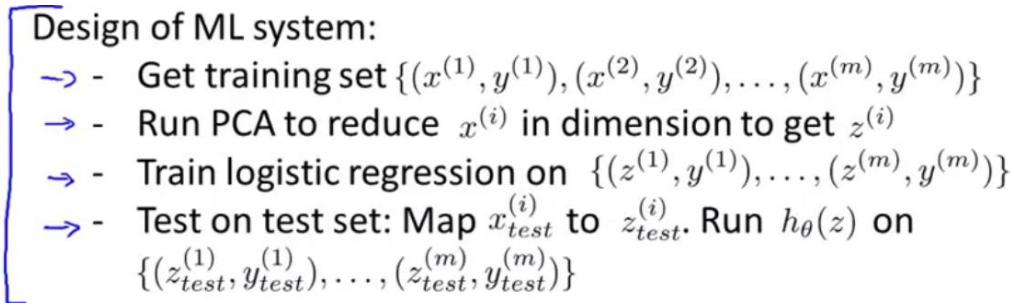
1. Using PCA to prevent overfitting.



Note: Using PCA to solve the problem of overfitting is harmful since it removes some of our features and this can also lead to some important features being removed, which is detrimental for our analysis. Using regularization is much more effective.

2. Using PCA where it shouldn't be used.

Shown below is the general layout of an ML system that people often use.



Now, step 2 i.e. running PCA to reduce $x^{(i)}$ to get $z^{(i)}$ is not always necessary.

Usually, we should train logistic regression using $x^{(i)}$ values and if there is a lot of problem, only then should we use PCA:

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 $z^{(i)}$.