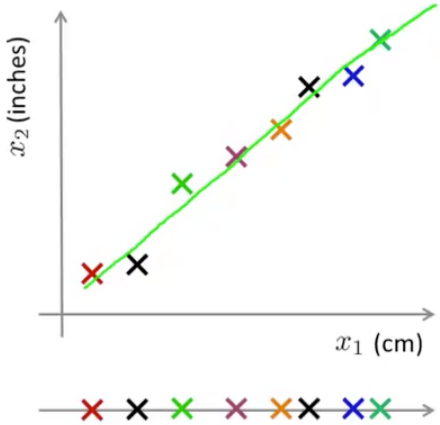
Principle Component Analysis (PCA)

Another unsupervised learning is dimensionality reduction. There are many reasons why we would like to reduce the dimensionality. The main reasons to use PCA are data compression, data visualization and speeding up the algorithm.

Compression

Compression allow us to speed up our learning algorithm. Imagine we have collected so many data with many features. For example, instead of having 2 highly correlated features, we may reduce it to one feature. In this case, we may find a direction on which most of the data seems to lie, then project the data on that line and finally come up with new feature Z_1. In real data, we normally reduce dimensionality from 1000 to 100-dimensions.



Suppose we apply dimensionality reduction to a dataset of m examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, *where* $x^{(i)} \in R^n$. *As a result, we get a lower dimensional dataset* $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ *of m examples where* $z^{(i)} \in R^k$ *for some values of k and* $K \leq n$.

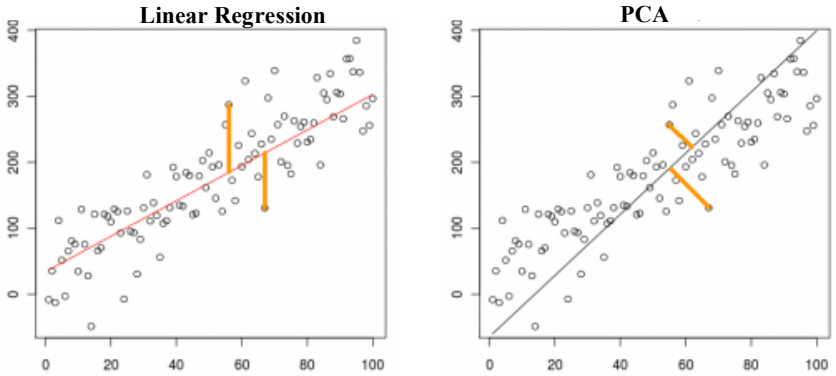
Data Visualization

It's useful to get better insight from data. Imagine, we have a table with about 50 features. It's hard to plot and look at 50-dimensional data. If we can reduce the data to like 2 or 3 dimensional, we can visualize data better.

Principle Component Analysis (PCA)

The most popular algorithm for data reduction is PCA. It tries to find a lower dimensional surface to project the data so the sum of square is minimized (the projection error is minimized).

PCA is not linear regression, even though there are some similarity. In linear regression (left plot), vertical distance between line and points is calculated and then minimizes the distance, but in PCA, it minimizes the shortest distance between the line and the data points.



Principle Component Analysis (PCA)

Which gives us very different effects. Moreover, in linear regression we want to predict y from x but in PCA we treat all x in same way and we would like to reduce dimensionality only.

Data Preprocessing and PCA Algorithm

Imagine we have a training set: $x^{(1)}, x^{(i2)}, \ldots, x^{(im)}$, *during preprocessing we do feature scaling, mean normalization*. So, we can make sure features have comparable range of values. Using this equation:

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

mean normalization: replace each $x_j^{(i)}$ with $x_j - \mu_j$

Feature scaling: replace each $x_j^{(i)}$ with $(x_j - \mu_j)/s_j$

Where s_j can be max-min or standard deviation (more common). After that we can apply PCA algorithm as follow to reduce data from n-dimensional to k- dimensions:

Compute covariance matrix:

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

vector version is:

$$\Sigma = \frac{1}{m} * x * x^T$$

Then compute the eigenvectors (n by n matrix)

We may use singular value decomposition "svd" in Matlab to calculate the eigenvectors. "svd" is more stable than "eig" in programming. SVD outputs [u, s, v], but we only need matrix u which is column of $[u^{(1)}, u^{(2)}, \ldots, u^{(n)}]$. We would like to find a lower dimension of the matrix. So, we take the first k column of $u^{(i)}$ and create a n by k matrix. Finally, z will be a k by 1 matrix:

$$Z=[u^{(1)}, u^{(2)}, \ldots, u^{(k)}]^T x$$

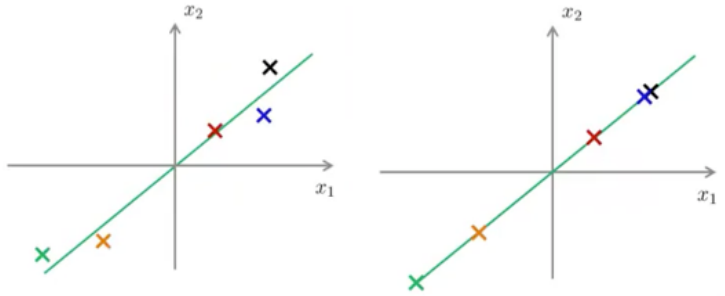
Reconstruction from Compressed Representation

With Simple math, we can use this equation to get the approximate value for x:

$$x=[u^{(1)}, u^{(2)}, \ldots, u^{(k)}] * z$$

Principle Component Analysis (PCA)

So, If the original points are as shown in left plot, we get approximation of the points that lays on the projection line as shown in the right plot.



Currently, we know given an unlabeled dataset, how we can reduce the dimensionality. Next, we learn about how to perform this algorithm to be more effective.

Choosing Number of PCA

K is known as number of components in PCA. Choose k to be the smallest values so that 99% of variance is retained:

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

The equation above is sum of mean squared error divided by variance. It's common to retain above 95% of the variance. In real life, most of the data are highly correlated and it's possible to compress the data lot. There are two ways how we can choose k:

Try PCA with k=1 Compute $U_{reduced}, z^1, z^2, \ldots$ Check if equation above stands? If not k=k+1	[u, s, v] =svd(sigma) for given k: use equation: $1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \leq 0.01$ when s=$\begin{bmatrix} s_{11} & 0 & 0 \\ 0 & \ldots & 0 \\ 0 & 0 & s_{nn} \end{bmatrix}$

The right algorithm is very time consuming, however in programming when it calculates svd, it gives a diagonal matrix s. Then we can calculate the equation by 1 minus summation of k values of s diagonally divided by summation of all diagonal values in matrix s. In the second algorithm, we calculate svm once and calculate values of equation for k=1, 2,.. till it satisfied the equation.

Principle Component Analysis (PCA)

Speed up the Algorithm

In supervised learning, imagine we have $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, …., $(x^{(m)}, y^{(m)})$. Let's say $x^{(i)}$ is high dimensional example, like 10,000 feature vectors, like image 100 by 100 pixels. Here the learning algorithm can be very slow. With PCA we can reduced the dimensionality and speed up the learning as follows:

Input: $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, …., $(x^{(m)}, y^{(m)})$
Extract inputs:
Unlabeled dataset: $x^{(1)}, x^{(2)}, \ldots\ldots, x^{(m)} \in R^{10,000}$
Apply PCA: $x^{(1)}, x^{(2)}, \ldots\ldots, x^{(m)} \in R^{1,000}$
New training set: $(z^{(1)}, y^{(1)})$, $(z^{(2)}, y^{(2)})$, …., $(z^{(m)}, y^{(m)})$
Then feed it into algorithm
For new testing example, we do the same with PCA

Note that mapping x to z should be only defined by running PCA only on training dataset. Then you can apply that mapping be applied in cross validation and testing dataset.

Misuse of PCA

Some may use it to avoid overfitting. This is not a great reason to use PCA. Even though, we are reducing number of PCA, and few features, likely to overfit. PCA should not be used for overfitting and must use regularization instead. Because PCA throws away some information that might hurt the model.

Another misuse is to include it in a plan for machine learning problems:

Design of ML system:
Get training set
Run PCA to reduce dimensionality
Train logistic regression
Test on test set

First do it without PCA, then if it does not work use PCA after because of computational or any other strong reasons, then you may include PCA.

So, in summary PCA should be used for compression (so it takes less computer memory), to reduce dimension of input so it speeds up the learning algorithm, and to visualize high dimensional data in 2D or 3D.