Principal Component Analysis

26 April 2025 10:34

A quick recap of Gradient Descent:

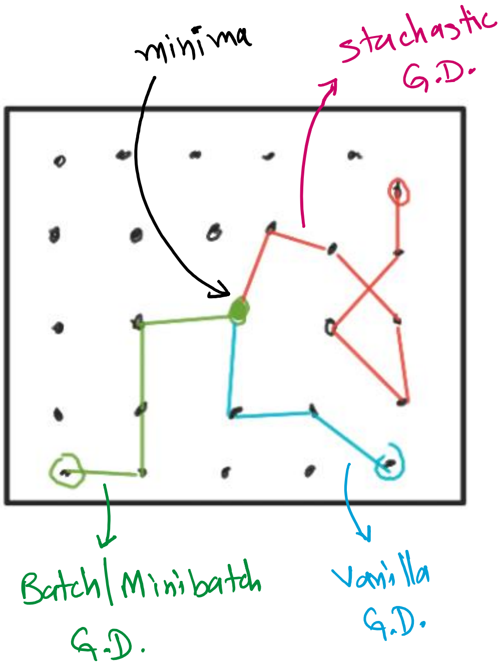
Core formula of GD:

$x_{t+1} = x_t - h*DL$

Where ΔL is:

$$\frac{\partial L}{\partial m} = \frac{-2}{n} \sum (y_i - (mx_i + c)) \cdot x_i$$

$$\frac{\partial L}{\partial c} = \frac{-2}{n} \sum (y_i - (mx_i + c))$$



1. **Vanilla G.D. -** In case of Vanilla GD, we consider all the datapoints from 1 to n to find the gradient ΔL in above equations. As we are taking all the points in account, we will get a very "perfect" gradient that will take us straight into the direction of the minima. Hence, we will need a very few such steps (or "leaps") to reach to the minima but the issue is that the each step or leap will need a lot of calculations in it making it time consuming.

2. **Stochastic G.D. -** In this variation of GD we just consider one random datapoint (x_k, y_k) to put into the above equations of ΔL and compute the gradient only using it making our steps very quick (using minimal computations). But here as we only take one point, our gradient will also be very much random and hence sometimes it may also take us away from the minima which will in turn, result into the need of taking a lot of steps (or leaps) to reach to the minima.

3. **Batch / Minibatch G.D. -** This version of GD takes "some" datapoints (any number between 1 to n) which is a "sample" from our entire dataset and use only those points to predict the gradient (using above equations). As an example, if we have 1 million datapoints then we may take 100 or 500 or 1000 datapoints from them at random and try to predict gradient using them. Suppose we are taking 100 datapoints then in each iteration we need to do 100 computations inside Σ and according to CLT, this sample of 100 datapoints will have the same mean as the entire dataset hence we will get a "decent" gradient which will not be converging (converging = moving towards the goal) as straight as Vanilla GD but it will not at least take you in the opposite direction. Therefore, this is the best balance of Vanilla & Stochastic GD. It will take less number of steps/leaps and each step will also not be very much time consuming.

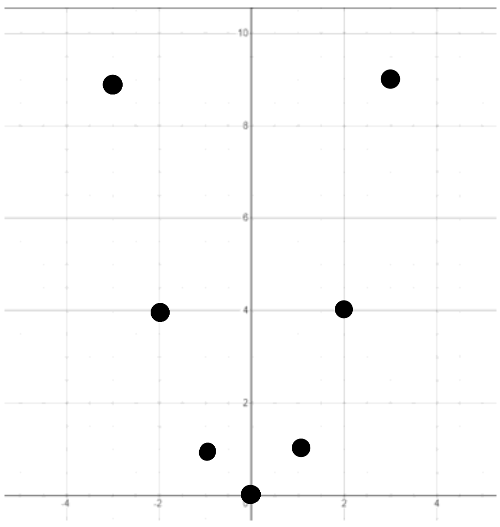
Why do we need PCA?
Ans: To reduce dimensionality of our data.

Why do we want to reduce the dimensionality (number of features)?

Ans:

1. To visualize the data - we need to get our data in 1D, 2D or at max 3D
2. To reduce the complexity in our formulae
    a. The components of a gradient are as many as the number of features/dimensions
    b. In each step of GD, X_i will be an array of the length equals to the number of the features/dimensions. For a 2D data, there will be two components of each X_i (that is X_i^1 and X_i^2)
3. To reduce the execution time (training time) - more complex formulae will take more time to execute because it needs more calculations to be done
4. To reduce "Curse of dimensionality"

Curse of dimensionality:



$y = x^2$

X	-3	-2	-1	0	1	2	3
Y	9	4	1	0	1	4	9

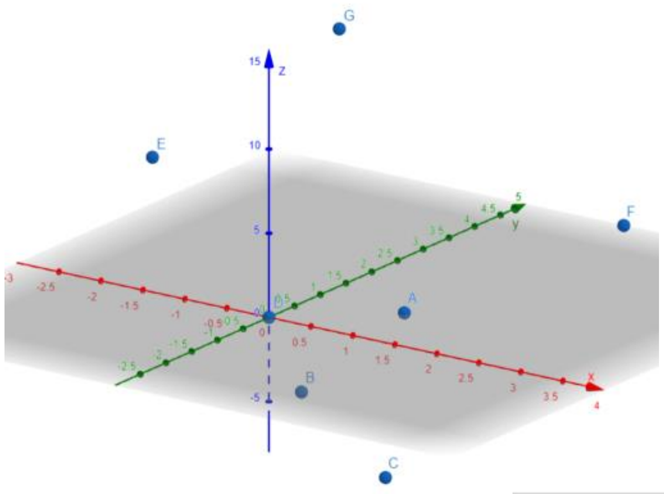
We can easily understand the behavior of this function just by plotting these 7 points **because it had only 2 dimensions.**

Now let's try to do the similar thing on a function that has just one more dimension i.e., 3D and let's try to understand the behavior of the following function by plotting some points:

let $\mathbf{z = x^2y - y^2x}$ be our function

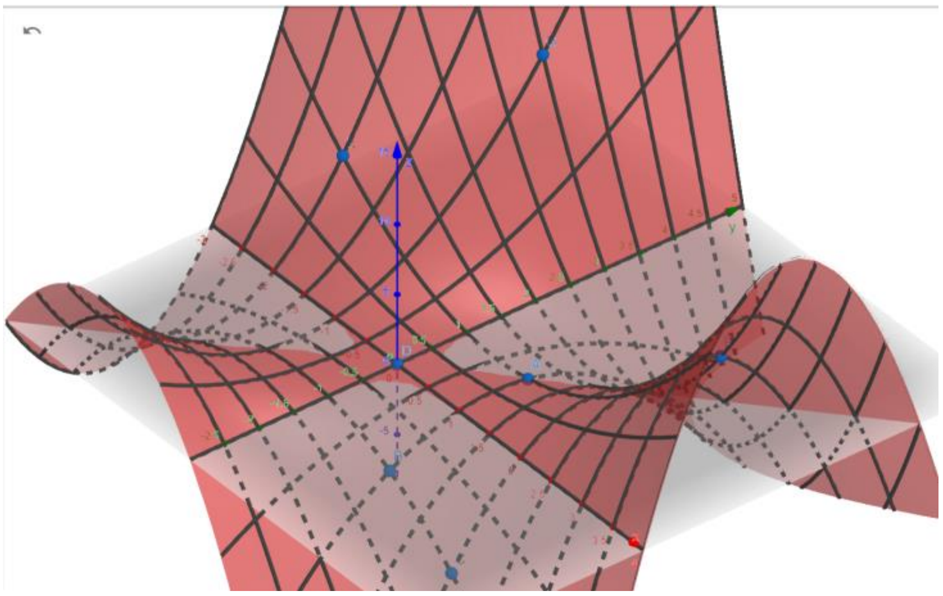
X	1	1	2	0	-2	3	-1
Y	1	-1	-1	0	1	2	3
Z	0	-2	-6	0	6	6	12

If we plot a graph of these points,
it will look like this -



As we see it is very difficult to imagine the nature of this graph by plotting these points. Hence we will need a lot more points to understand the behavior of this function.

The actual graph of this function is as below:



Imagine how many points you will need to imagine this graph!

The curse of dimensionality says that as the number of dimensions increases, the volume of the data space grows exponentially, making it harder to find patterns and relationships between the features & the target.

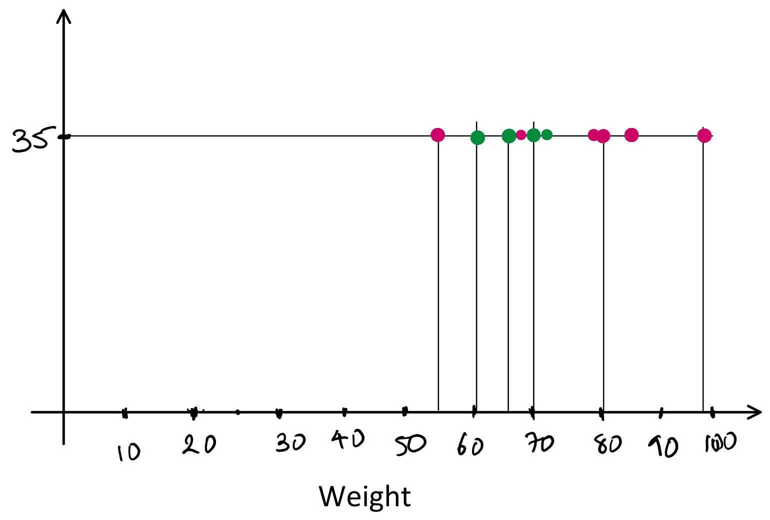
But as we remove a dimension (feature), some data is also lost along with it. Let's try to understand this fact by an example. Suppose we have got a data of rainfall.

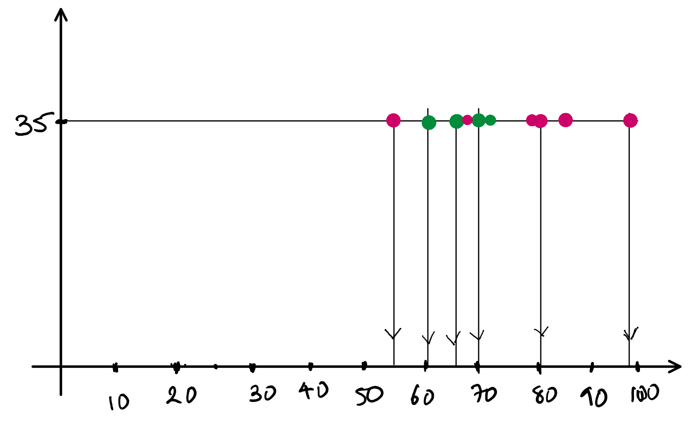
Humidity	Temperature	Wind Speed	Wind Direction	Month	Type of clouds	Cloud Density	Last precipitation	Rainfall
---	---	---	---	---	---	---	---	y_1

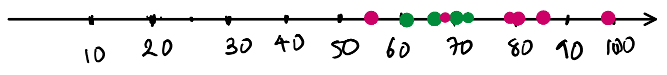
When we say we want to reduce the dimensionality, essentially we are going to remove some feature(s). What do you think, which feature could be the best choice to remove from the above dataset?
Ans: Month
But we don't take decisions based on intuition/logic/common sense etc., we take decisions those are data driven and we don't have put a single row of data in the above table! Hence, **ideally** we should find the correlation of each feature with the target variable & decide to drop the feature that has minimum correlation with the rainfall. But what if a feature is correlated with the target & we remove it then it will take some portion of information with it leaving us with a little less data. Now let's try to understand this mathematically with another example and also with 4 different cases.
Suppose we have the following data:
Case - 1: Removing Unnecessary Feature(s) - the feature that have no correlation with the target

Weight	80	60	55	65	70	68	72	98	79	85

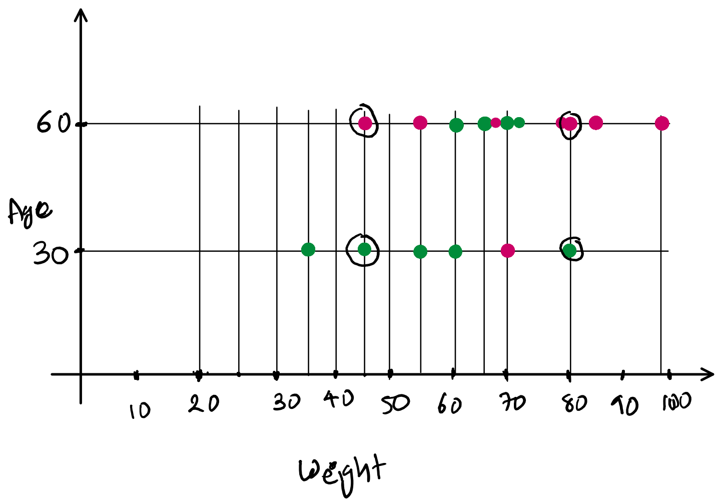
Age	35	35	35	35	35	35	35	35	35	35
Diabetic	1	0	1	0	0	1	0	1	1	1







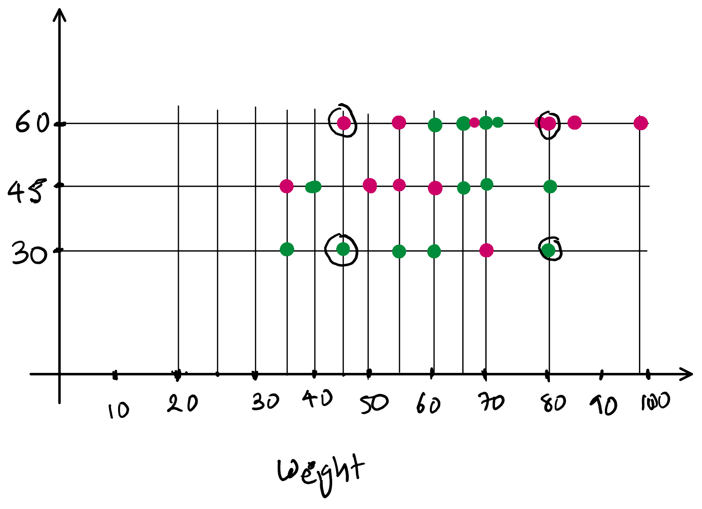
Case - 2: We have got 2 different values of age



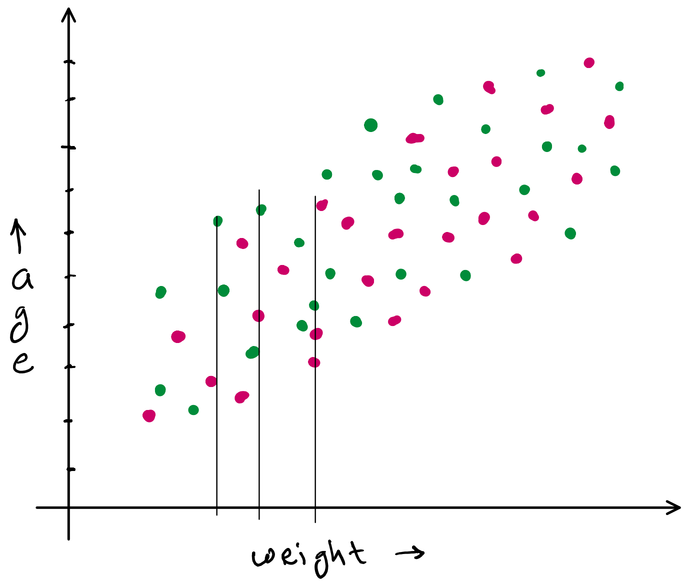
If we use the same method as above to remove the "age" dimension, we will have one person with weight 45 diabetic & one with the same weight non-diabetic and this will happen for many other points too e.g. weight 55, 70, 80 and so on.
This is the mathematical representation of "data loss/information loss".

Case - 3: What will happen if we have 3 unique values of age?

Now the information loss will be even more. Now we have 3 people of weight 55 out of which 2 are diabetic & 1 is not. Similarly 2 of 3 people of weight 60 are non-diabetic but 1 is diabetic. This will create more confusion for us to determine the relationship of weight & diabetes.



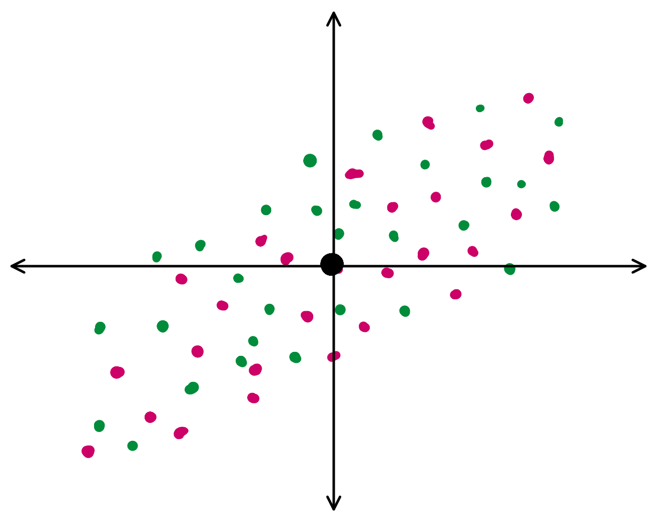
Case - 4: What if the data is as this?



If we directly remove "age" feature from this graph, we will lose a lot of information (data) hence at the first look, it seems that we cannot remove "age" dimension from this data.

But that is what PCA is about. Using PCA (applying some simple mathematical steps) we can still remove the age feature from our data with minimum loss of information!

Step - 1: Move (translate) the origin exactly on the mean of both the features



Step - 2: Rotate the axis as shown: