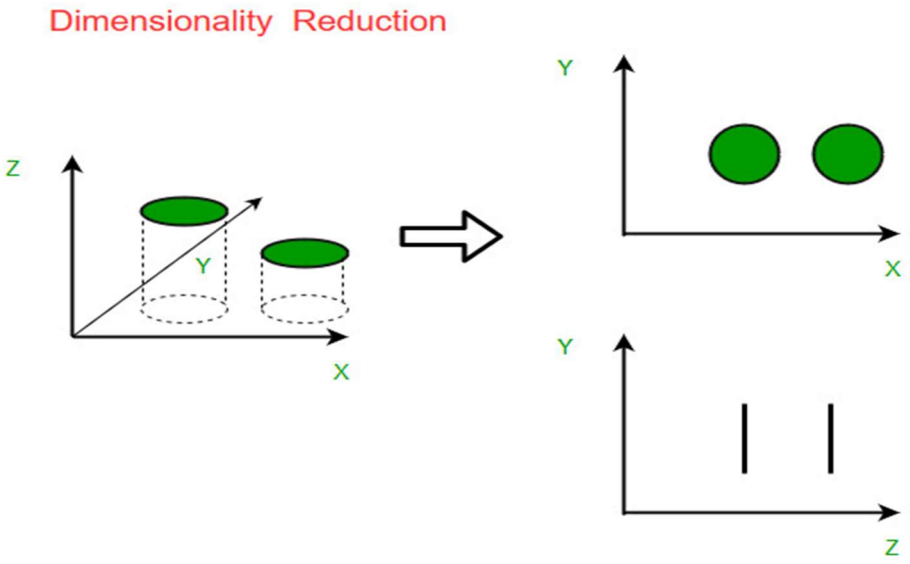
What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 2-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection/ Elimination:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
    1. Filter
    2. Wrapper
    3. Embedded

Feature elimination

- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.

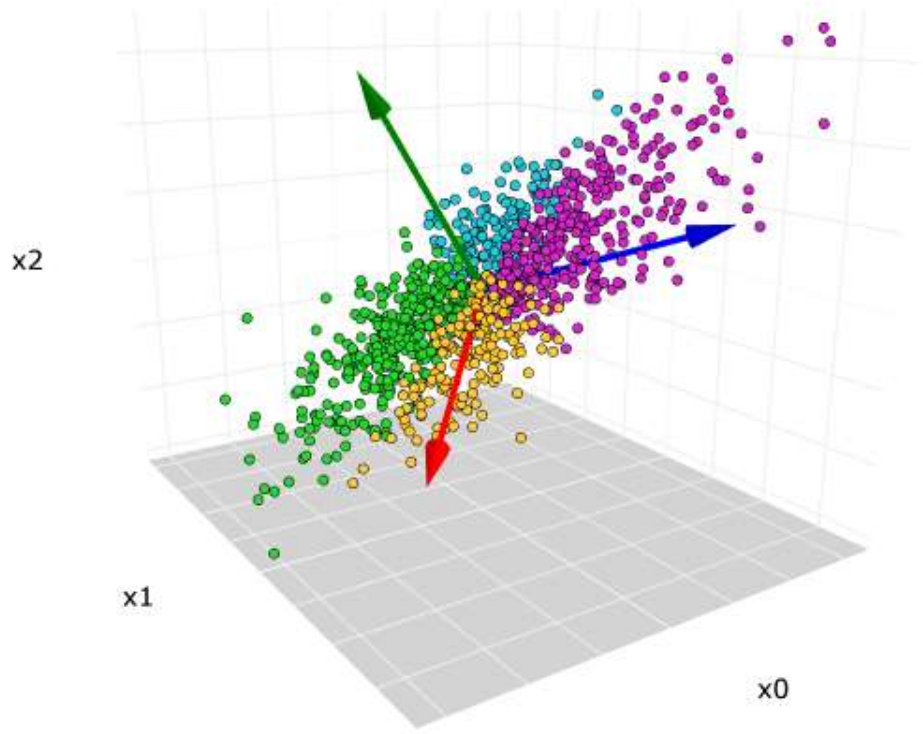
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

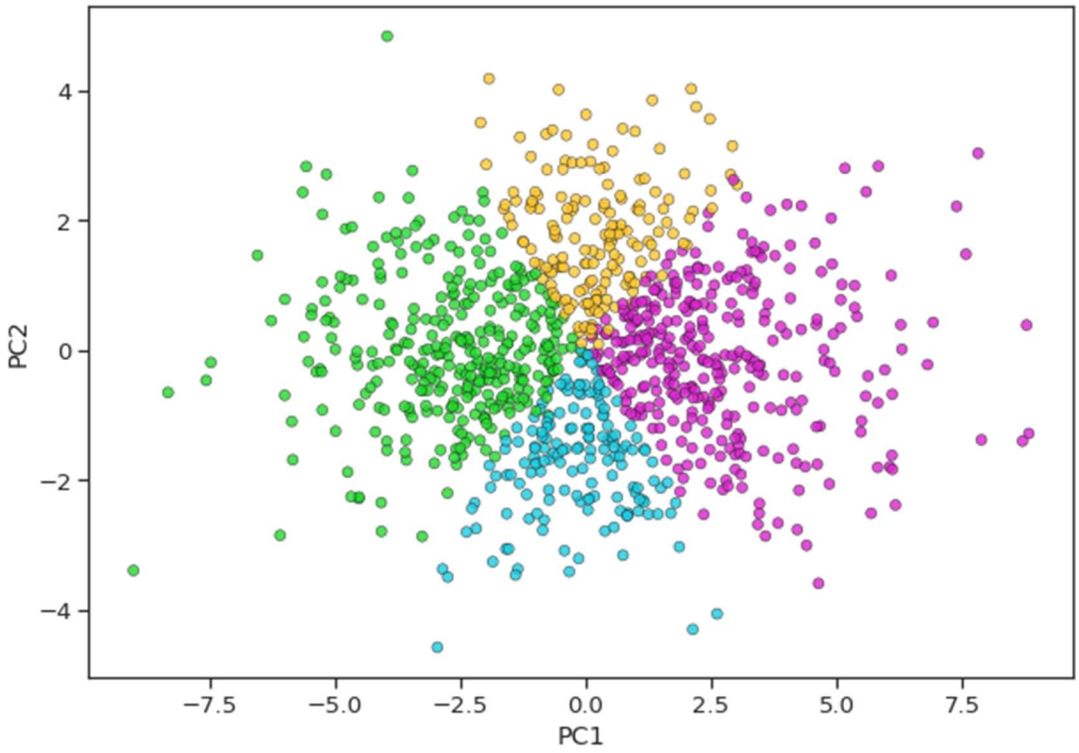
- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

What is PCA?

Principal component analysis (PCA) is a technique that transforms high-dimensions data into lower-dimensions while retaining as much information as possible.



The original 3-dimensional data set. The red, blue, green arrows are the direction of the first, second, and third principal components, respectively.



Scatterplot after PCA reduced from 3-dimensions to 2-dimensions.

PCA is extremely useful when working with data sets that have a lot of features. Common applications such as image processing, genome research always have to deal with thousands-, if not tens of thousands of columns.

While having more data is always great, sometimes they have so much information in them, we would have impossibly long model training time and the curse of dimensionality starts to become a problem. Sometimes, less is more.

How does PCA work?

It's a two-step process. We can't write a book summary if we haven't read or understood the content of the book.

PCA works the same way — understand, then summarize.

Understanding data the PCA way

Human understands the meaning of a storybook through the use of expressive language. Unfortunately, PCA doesn't speak English. It has to find meaning within our data through its preferred language, mathematics.

The million-dollar question here is…

- Can PCA understand which part of our data is important?
- Can we mathematically quantify the amount of information embedded within the data?

Well, *variance* can.

For most people out there, variance is not an unfamiliar term. We've learned in high school that variance measures the average degree to which each point differs from the mean.

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

The formula of variance.

But nowhere does it associate variance with information. So where does this association comes from? And why should it make sense?

Suppose that we are playing a guessing game with our friends. The game is simple. Our friends would cover their faces and we need to guess who's who based solely on their height. Being the good friends that we are, we remember how tall everyone is.

Person	Height (cm)
Alex	145
Ben	160
Chris	185

Our friends' heights from memory.

I'll go first.

The silhouette of three identical friends whom we need to identify based on their height differences

Without a doubt, I am going to say that Person A is Chris, Person B is Alex, and Person C is Ben.

Now, let's try and guess a different group of friends.

Person	Height (cm)
Daniel	172
Elsa	173
Fernandez	171

Another group of our friends' heights that we remember by heart.

Your turn.

The silhouette of three similarly-tall friends whom we need to identify.

Can you guess who's who? It's tough when they are very similar in height.

Earlier, we had no trouble differentiating a 185cm person from a 160cm and 145cm person because their height *varies* a lot.

In the same way, when our data has a higher variance, it holds more information. This is why we keep hearing PCA and maximum variance in the same sentence.

PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the **greatest variance** by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

In the eyes of PCA, variance is an objective and mathematical way to quantify the amount of information in our data.

Variance *is* information.

To drive the point home, I propose a rematch for the guessing game, only that this time, we get to guess who's who based on their height and weight.

Round two.

Person	Height (cm)	Weight (kg)
Alex	145	68
Ben	160	67
Chris	185	69

The same set of friends and their respective height and weight. Image by the author.

In the beginning, we only had height. Now, we have essentially doubled the amount of data on our friends. Would that change your guessing strategy?

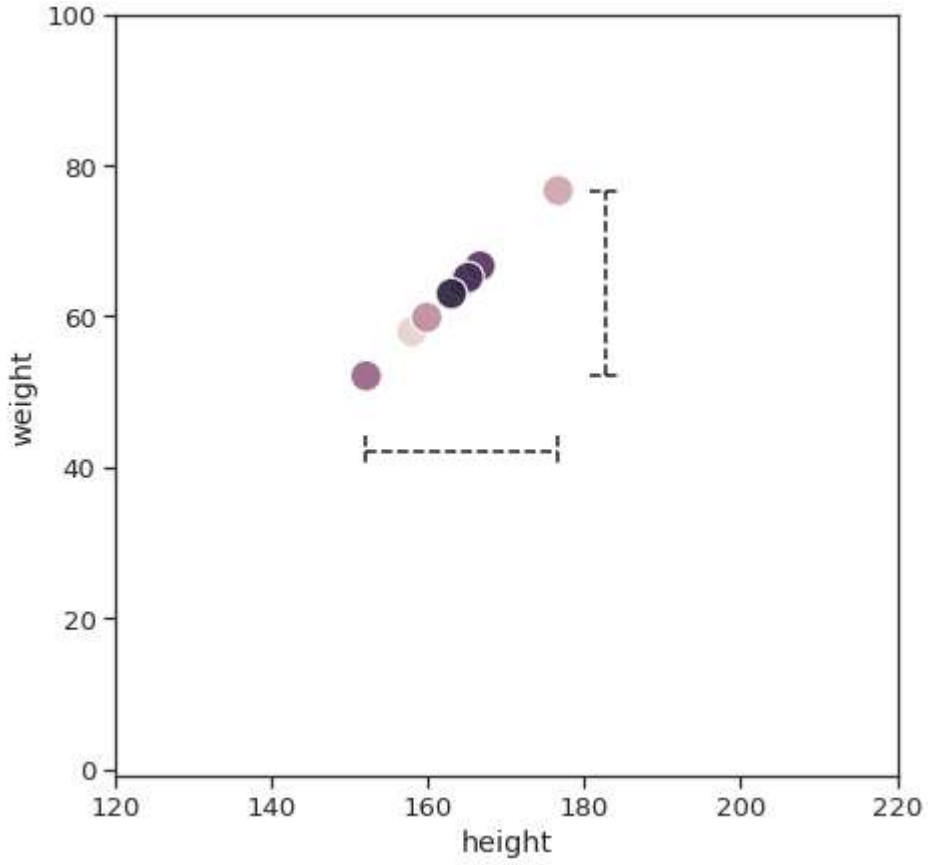
This is a nice little segue into the next section — how PCA summarizes our data, or more accurately, reduces dimensionality.

Summarizing data with PCA

Personally, the weight differences are so small (a.k.a small variance), it doesn't help me differentiate our friends at all. I still had to rely mostly on height to make my guesses.

Intuitively, we have just reduced our data from 2-dimensions to 1-dimension. The idea is that we can selectively keep the variables with higher variances and then forget about the variables with lower variance.

But what if-, just what if height and weight have the *same* variance? Does it mean we can no longer reduce the dimensionality of this data set? I'd like to illustrate this with a sample dataset.



The dotted line represents the variance of height and weight.

Feature	Variance
Height	1.11
Weight	1.11
TOTAL	**2.22**

All the features are standardized to the same scale for a fair comparison.

In this case, it's very difficult to choose the variables we want to delete. If I throw away either one of the variables, we are throwing away half of the information.

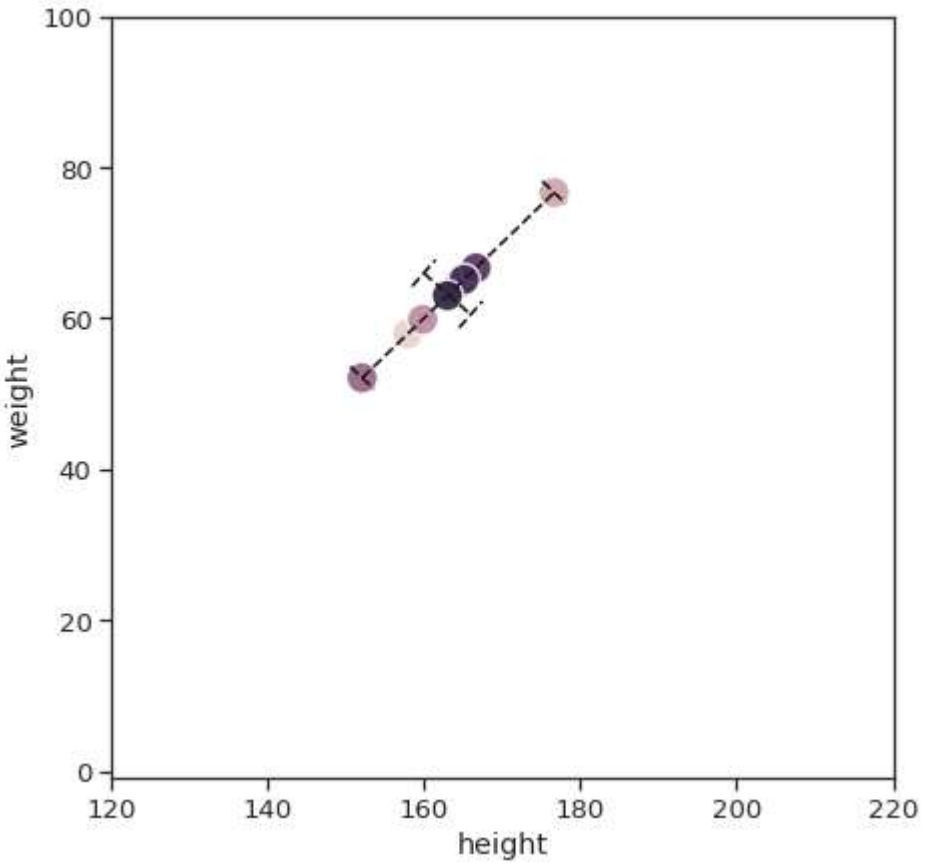
Can we keep *both*?

Perhaps, with a different perspective.

The best storybooks always have hidden themes that are not written but *implied*. Reading each chapter individually wouldn't make sense. But if we read all of it, it gives us enough context to piece the puzzles together — the underlying plot emerges.

Up until now, we have only been looking at the variance of height and weight individually. Instead of limiting ourselves to choose just one or the other, why not combine them?

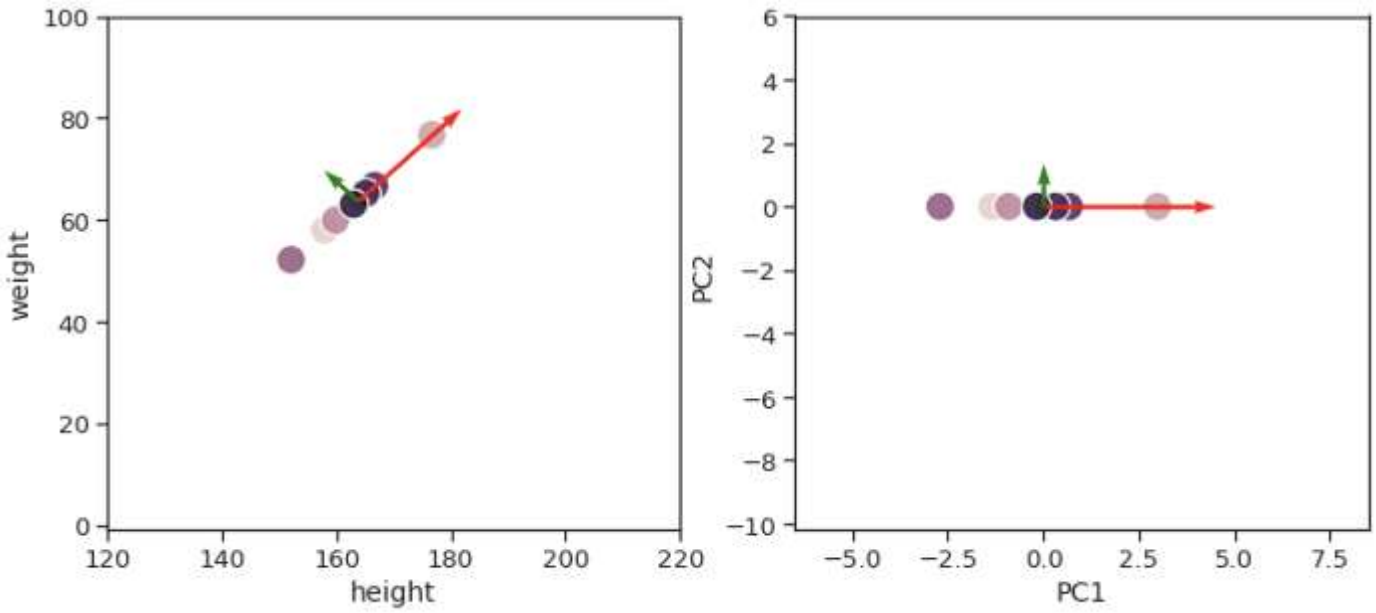
When we look closer at our data, the maximum amount of variance lies not in the x-axis, not in the y-axis, but a diagonal line across. The second-largest variance would be a line 90 degrees that cuts through the first.



The dotted line shows the direction of maximum variance.

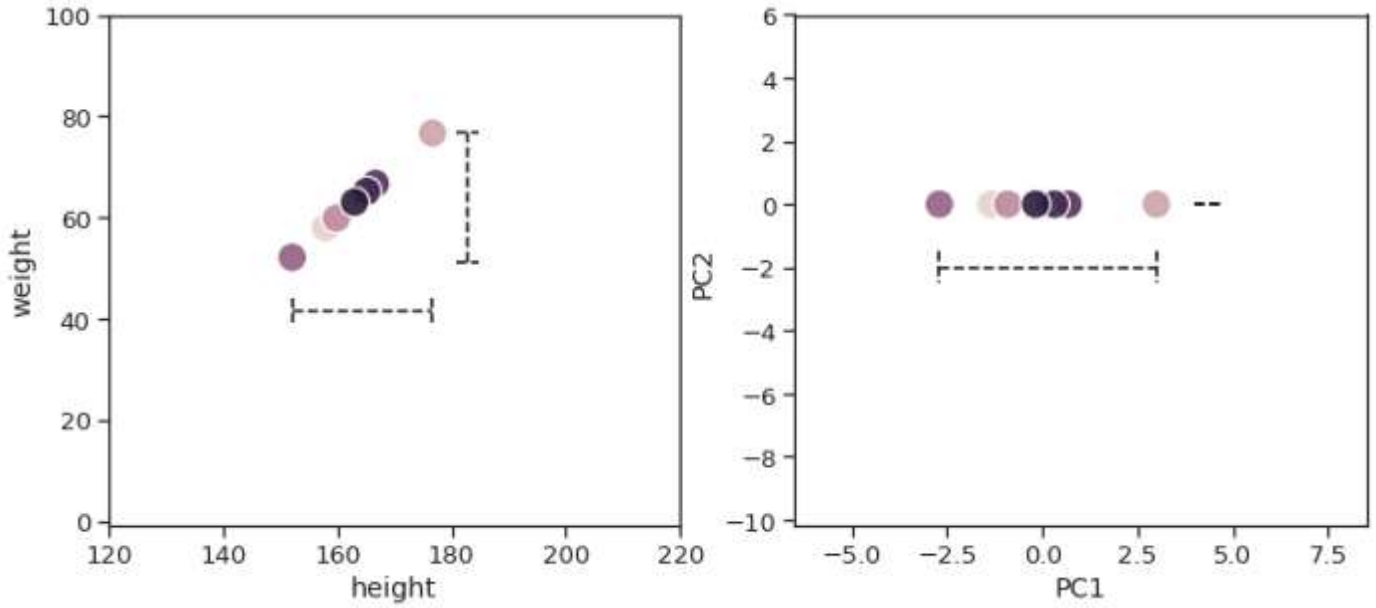
To represent these 2 lines, PCA combines both height and weight to create two brand new variables. It could be 30% height and 70% weight, or 87.2% height and 13.8% weight, or any other combinations depending on the data that we have.

These two new variables are called the **first principal component (PC1)** and the **second principal component (PC2)**. Rather than using height and weight on the two axes, we can use PC1 and PC2 respectively.



(**Left**) The red and green arrows are the principal axes in the original data. Image by the author. | (**Right**) The direction of the principal axes have been rotated to become the new x- and y-axis.

After all the shenanigans, let's take a look at the variances again.



Left) The variance of height and weight are similar in the original data. Image by the author. | (**Right**) After PCA transformation, all of the variances are shown in the PC1 axis.

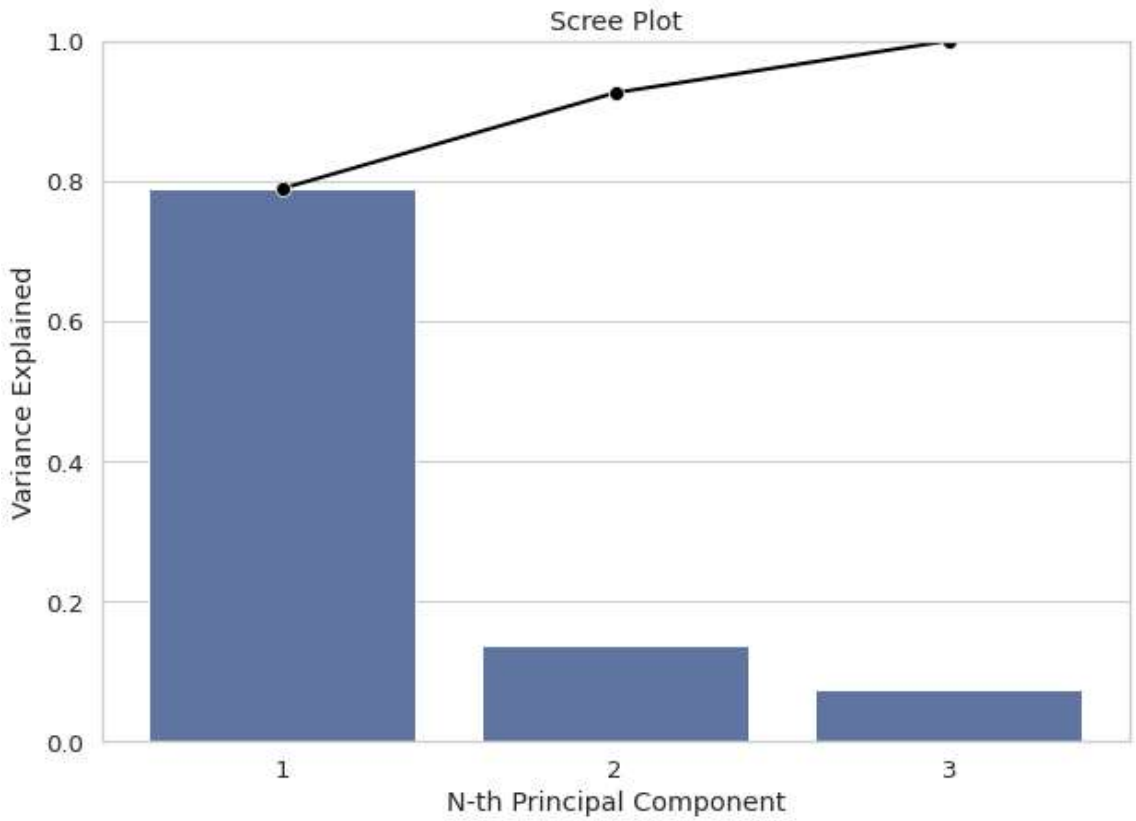
Feature	Variance	Feature	Variance
Height	1.11	PC1	2.22
Weight	1.11	PC2	0.00
TOTAL	**2.22**	**TOTAL**	**2.22**

All the variables are standardized to the same scale for a fair comparison.

PC1 alone can capture the total variance of Height and Weight combined. Since PC1 has all the information, you already know the drill — we can be very comfortable in removing PC2 and know that our new data is still representative of the original data.

When it comes to real data, more often than not, we won't get a principal component that captures 100% of the variances. Performing a PCA will give us N number of principal components, where N is equal to the dimensionality of our original data. From this list of principal components, we generally choose the least number of principal components that would explain the most amount of our original data.

A great visual aid that will help us make this decision is a **Scree Plot**.



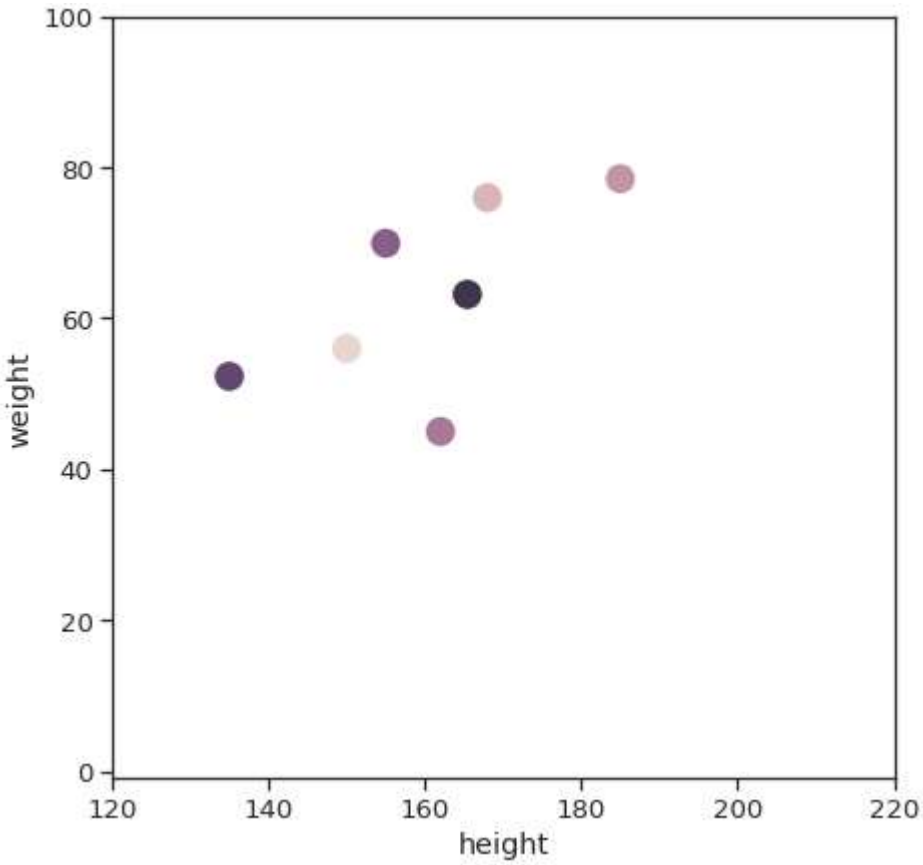
An example of a Scree Plot for a 3-dimensional data set.

The bar chart tells us the proportion of variance explained by each of the principal components. On the other hand, the superimposed line chart gives us the cumulative sum of explained variance up until N-th principal component. Ideally, we want to get at least 90% variance with just 2- to 3-components so that enough information is retained while we can still visualize our data on a chart.

Looking at the chart, I would feel comfortable using 2 principal components.

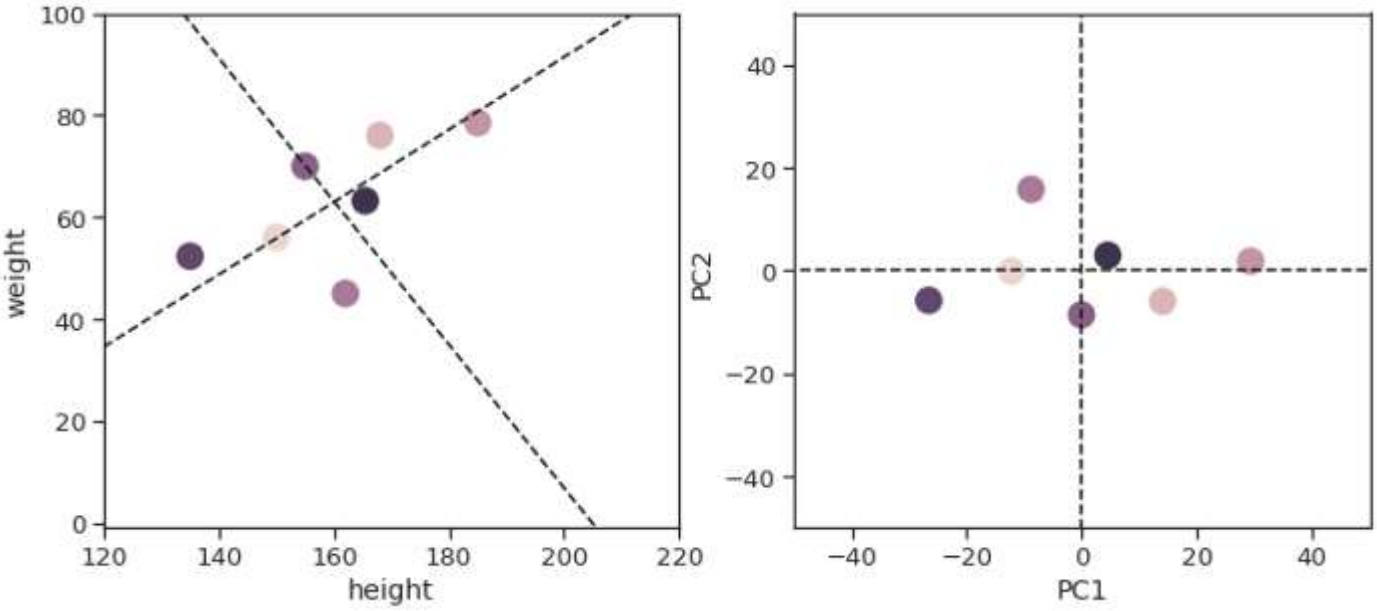
The PC That Got Away

Since we're not choosing all the principal components, we inevitably lose some information. But we haven't exactly described what we are losing. Let's dive deeper into that with a new toy example.



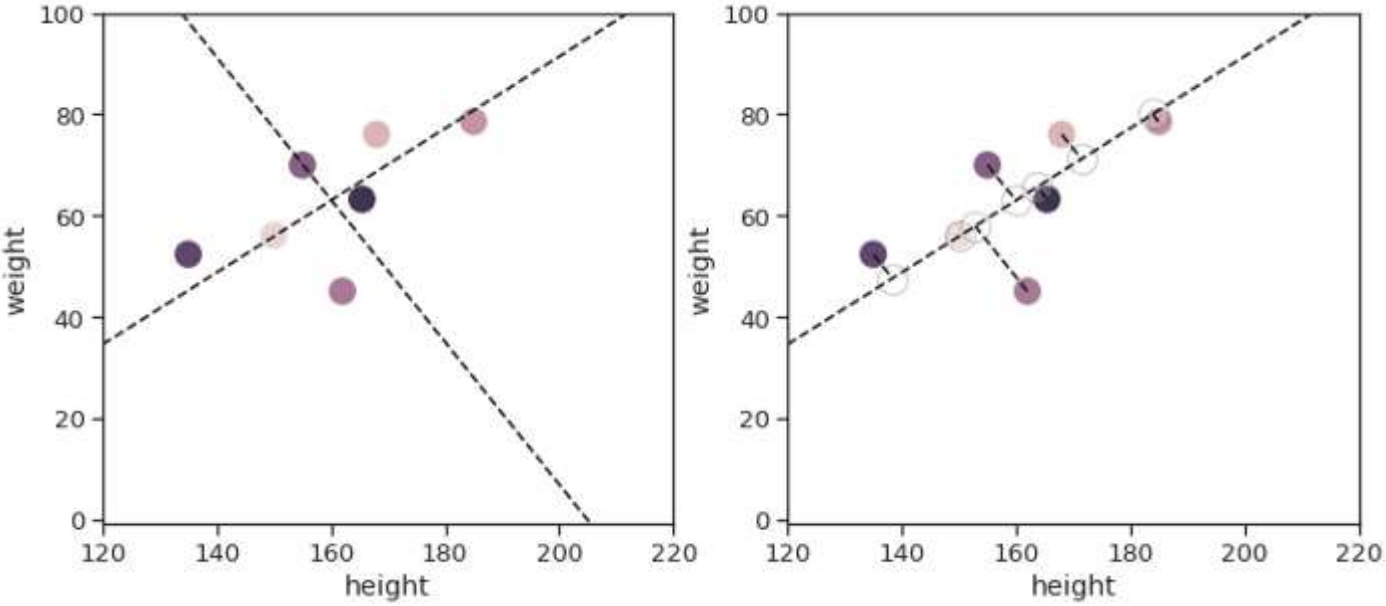
The points are scattered but we can still see some positive correlation in a diagonal line across.

If we feed our data through the PCA model, it would start by drawing the First Principal Component followed by the Second Principal Component. When we transform our original data from 2-dimensions to 2-dimensions, everything stays the same except the orientation. We just rotated our data so that the maximum variance is in PC1. Nothing new here.



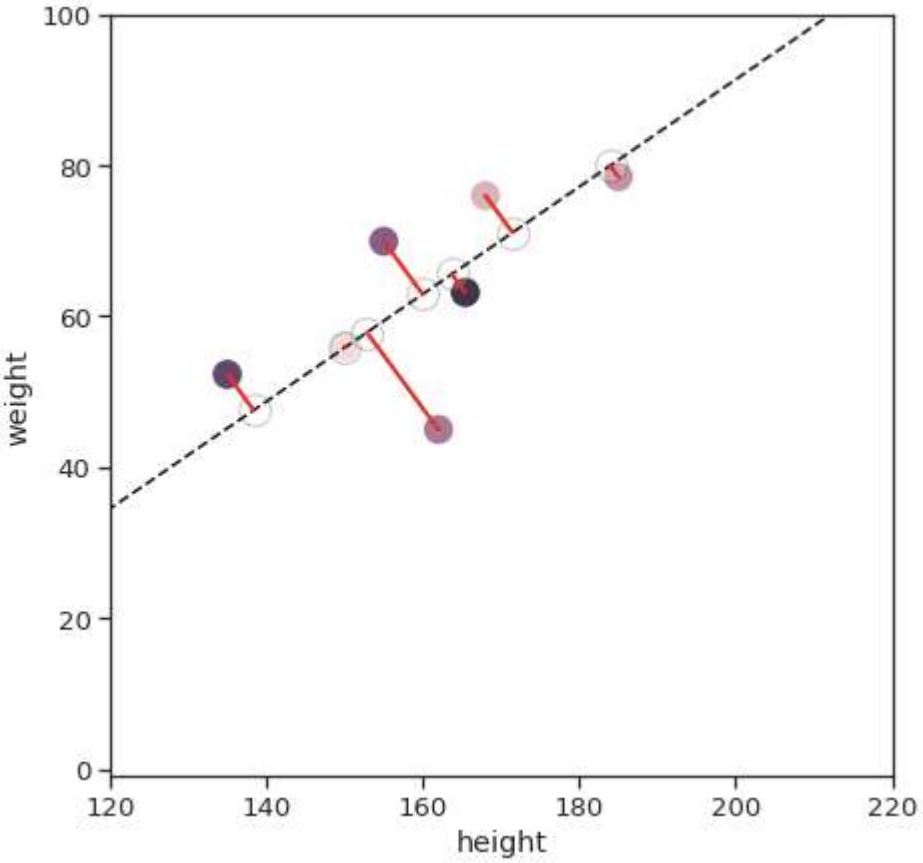
Left) The dotted lines are the direction of the first and second principal components. Image by the author. | (**Right**) PCA rotates the data hence putting the maximum variance on PC1, followed by PC2.

However, suppose that we have decided to keep only the First Principal Component, we would have to project all our data points onto the First Principal Component because we no longer have the y-axis.



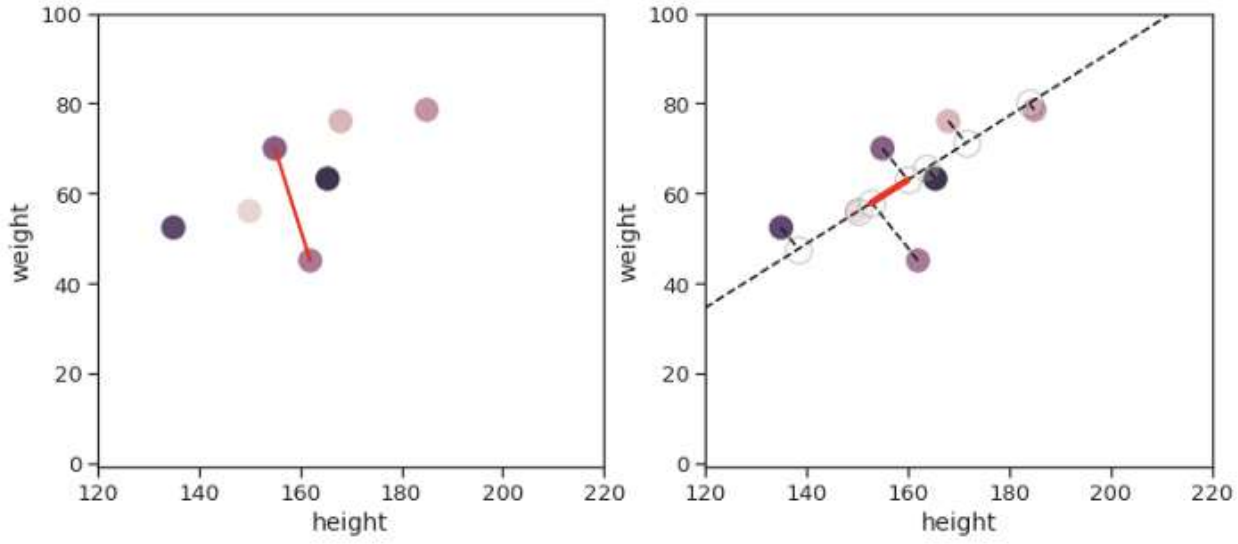
Left) The dotted lines are the direction of the first and second principal components. Image by the author. | (**Right**) All the dots now sit on the dotted line because we removed the 2nd principal component.

What we would lose is the distance in the Second Principal Component, highlighted with the red color line below.



All the red lines are values in the 2nd principal component and they have been removed

This has implications on the perceived distance of each data point. If we look at the Euclidean distance between two specific points (a.k.a pairwise distance), you will notice that some points are much farther in the original data than in the transformed data.

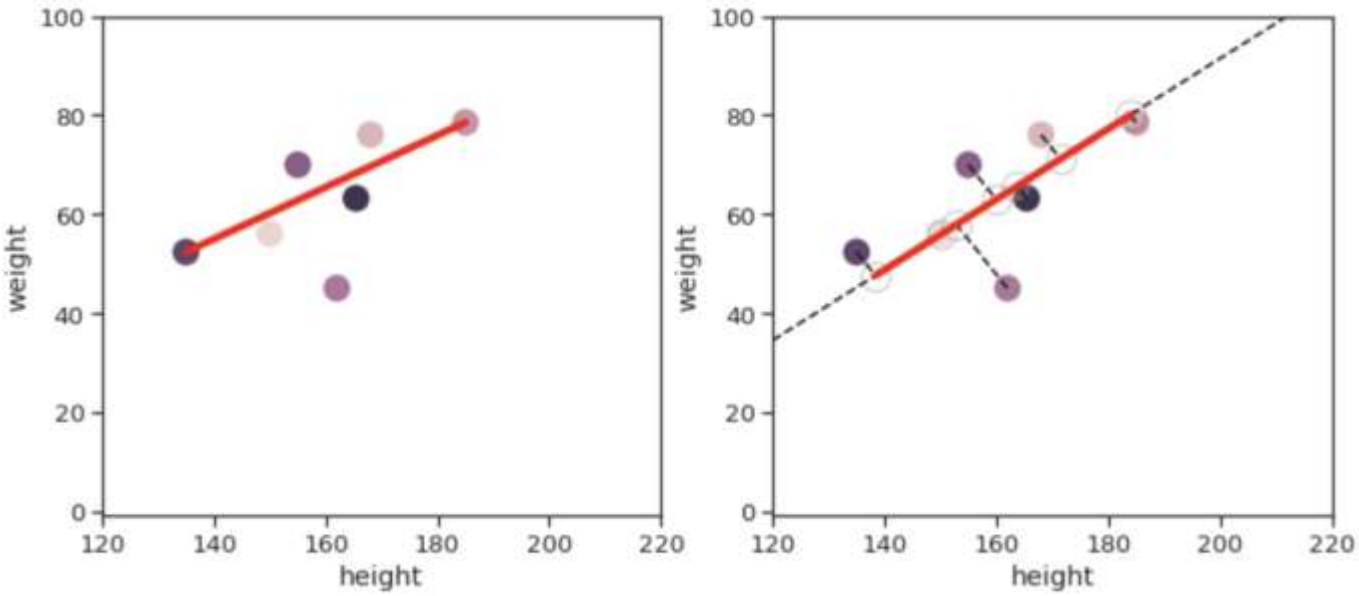


Left) and after (**Right**) dimensionality are reduced from 2- to 1-dimension.

The PCA is a linear transformation so in and of itself does not alter distances, but when we start removing dimensions, the distances get distorted.

It gets trickier— not all pairwise distance gets affected equally.

If we take the two furthest points, you will see that they are almost parallel to the principal axes. Although their Euclidean distance is still distorted, it is to a much lesser degree.



Left) and after (**Right**) dimensionality reduction remains fairly identical.

The reason is that principal component axes are drawn in the direction where we have the largest variance. By definition, variance increases when the data points are further apart. So naturally, the points furthest apart would align themselves better with the principal axes.

To sum it all up, reducing dimensions with PCA changes the distances of our data. It does so in a way that preserves large pairwise distance better than small pairwise distance.

This is one of the few drawbacks of reducing dimensions with PCA and we need to be aware of that, especially when working with Euclidean distance-based algorithm.

Dimension Reduction-

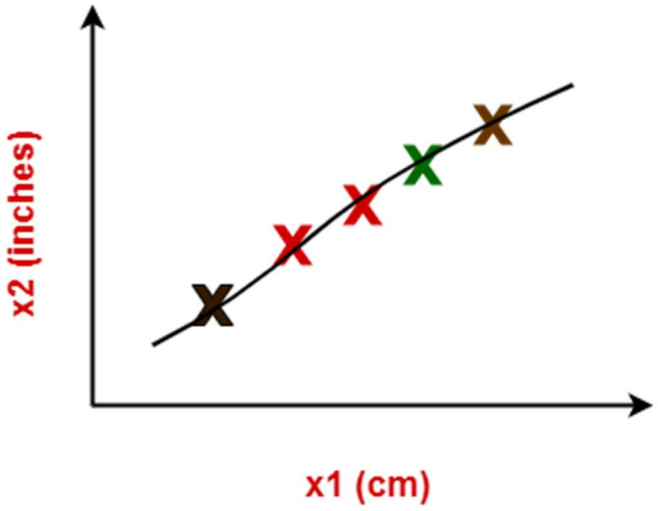
In pattern recognition, Dimension Reduction is defined as-

- It is a process of converting a data set having vast dimensions into a data set with lesser dimensions.
- It ensures that the converted data set conveys similar information concisely.

Example-

Consider the following example-

- The following graph shows two dimensions x1 and x2.
- x1 represents the measurement of several objects in cm.
- x2 represents the measurement of several objects in inches.



In machine learning,

- Using both these dimensions convey similar information.
- Also, they introduce a lot of noise in the system.

- So, it is better to use just one dimension.

Using dimension reduction techniques-

- We convert the dimensions of data from 2 dimensions (x1 and x2) to 1 dimension (z1).
- It makes the data relatively easier to explain.

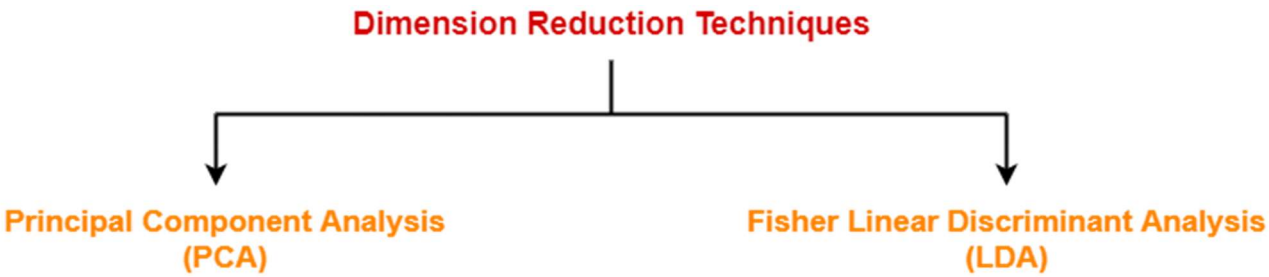
Benefits-

Dimension reduction offers several benefits such as-

- It compresses the data and thus reduces the storage space requirements.
- It reduces the time required for computation since less dimensions require less computation.
- It eliminates the redundant features.
- It improves the model performance.

Dimension Reduction Techniques-

The two popular and well-known dimension reduction techniques are-



1. Principal Component Analysis (PCA)
2. Fisher Linear Discriminant Analysis (LDA)

Principal Component Analysis-

- Principal Component Analysis is a well-known dimension reduction technique.
- It transforms the variables into a new set of variables called as principal components.
- These principal components are linear combination of original variables and are orthogonal.
- The first principal component accounts for most of the possible variation of original data.
- The second principal component does its best to capture the variance in the data.
- There can be only two principal components for a two-dimensional data set.

PCA Algorithm-

The steps involved in PCA Algorithm are as follows-

Step-01: Get data.

Step-02: Compute the mean vector (μ).

Step-03: Subtract mean from the given data.

Step-04: Calculate the covariance matrix.

Step-05: Calculate the eigen vectors and eigen values of the covariance matrix.

Step-06: Choosing components and forming a feature vector.

Step-07: Deriving the new data set.

PRACTICE PROBLEMS BASED ON PRINCIPAL COMPONENT ANALYSIS-

Problem-01:

Given data = { 2, 3, 4, 5, 6, 7 ; 1, 5, 3, 6, 7, 8 }.

Compute the principal component using PCA Algorithm.

OR

Consider the two dimensional patterns (2, 1), (3, 5), (4, 3), (5, 6), (6, 7), (7, 8).

Compute the principal component using PCA Algorithm.

OR

Compute the principal component of following data-

CLASS 1

X = 2 , 3 , 4

Y = 1 , 5 , 3

CLASS 2

X = 5 , 6 , 7

Y = 6 , 7 , 8

Solution-

We use the above discussed PCA Algorithm-

Step-01:

Get data.

The given feature vectors are-

- $x_1 = (2, 1)$
- $x_2 = (3, 5)$
- $x_3 = (4, 3)$
- $x_4 = (5, 6)$
- $x_5 = (6, 7)$
- $x_6 = (7, 8)$

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

Step-02:

Calculate the mean vector (μ).

Mean vector (μ)

= ((2 + 3 + 4 + 5 + 6 + 7) / 6, (1 + 5 + 3 + 6 + 7 + 8) / 6)

= (4.5, 5)

Thus,

$$\text{Mean vector } (\mu) = \begin{bmatrix} 4.5 \\ 5 \end{bmatrix}$$

Step-03:

Subtract mean vector (μ) from the given feature vectors.

- $x_1 - \mu = (2 - 4.5, 1 - 5) = (-2.5, -4)$
- $x_2 - \mu = (3 - 4.5, 5 - 5) = (-1.5, 0)$
- $x_3 - \mu = (4 - 4.5, 3 - 5) = (-0.5, -2)$
- $x_4 - \mu = (5 - 4.5, 6 - 5) = (0.5, 1)$
- $x_5 - \mu = (6 - 4.5, 7 - 5) = (1.5, 2)$
- $x_6 - \mu = (7 - 4.5, 8 - 5) = (2.5, 3)$

Feature vectors (x_i) after subtracting mean vector (μ) are-

$$\begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 2.5 \\ 3 \end{bmatrix}$$

Step-04:

Calculate the covariance matrix.

Covariance matrix is given by-

$$\text{Covariance Matrix} = \frac{\Sigma (x_i - \mu)(x_i - \mu)^t}{n}$$

Now,

$$m_1 = (x_1 - \mu)(x_1 - \mu)^t = \begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -2.5 & -4 \end{bmatrix} = \begin{bmatrix} 6.25 & 10 \\ 10 & 16 \end{bmatrix}$$

$$m_2 = (x_2 - \mu)(x_2 - \mu)^t = \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -1.5 & 0 \end{bmatrix} = \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix}$$

$$m_3 = (x_3 - \mu)(x_3 - \mu)^t = \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} -0.5 & -2 \end{bmatrix} = \begin{bmatrix} 0.25 & 1 \\ 1 & 4 \end{bmatrix}$$

$$m_4 = (x_4 - \mu)(x_4 - \mu)^t = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$m_5 = (x_5 - \mu)(x_5 - \mu)^t = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 1.5 & 2 \end{bmatrix} = \begin{bmatrix} 2.25 & 3 \\ 3 & 4 \end{bmatrix}$$

$$m_6 = (x_6 - \mu)(x_6 - \mu)^t = \begin{bmatrix} 2.5 \\ 3 \end{bmatrix} \begin{bmatrix} 2.5 & 3 \end{bmatrix} = \begin{bmatrix} 6.25 & 7.5 \\ 7.5 & 9 \end{bmatrix}$$

Now,

Covariance matrix

$= (m_1 + m_2 + m_3 + m_4 + m_5 + m_6) / 6$

On adding the above matrices and dividing by 6, we get-

$$\text{Covariance Matrix} = \frac{1}{6}\begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix}$$

$$\text{Covariance Matrix} = \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

Step-05:

Calculate the eigen values and eigen vectors of the covariance matrix.

λ is an eigen value for a matrix M if it is a solution of the characteristic equation $|M – \lambda I| = 0$.

So, we have-

$$\begin{vmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{vmatrix} - \begin{vmatrix} \lambda & 0 \\ 0 & \lambda \end{vmatrix} = 0$$

$$\begin{vmatrix} 2.92 - \lambda & 3.67 \\ 3.67 & 5.67 - \lambda \end{vmatrix} = 0$$

From here,

$(2.92 – \lambda)(5.67 – \lambda) – (3.67 \times 3.67) = 0$

$16.56 – 2.92\lambda – 5.67\lambda + \lambda^2 – 13.47 = 0$

$\lambda^2 – 8.59\lambda + 3.09 = 0$

Solving this quadratic equation, we get $\lambda = 8.22, 0.38$

Thus, two eigen values are $\lambda_1 = 8.22$ and $\lambda_2 = 0.38$.

Clearly, the second eigen value is very small compared to the first eigen value.

So, the second eigen vector can be left out.

Eigen vector corresponding to the greatest eigen value is the principal component for the given data set.

So. we find the eigen vector corresponding to eigen value λ_1.

We use the following equation to find the eigen vector-

$$MX = \lambda X$$

where-

- M = Covariance Matrix
- X = Eigen vector
- λ = Eigen value

Substituting the values in the above equation, we get-

$$\begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} = 8.22 \begin{bmatrix} X1 \\ X2 \end{bmatrix}$$

Solving these, we get-

$2.92X_1 + 3.67X_2 = 8.22X_1$

$3.67X_1 + 5.67X_2 = 8.22X_2$

On simplification, we get-

$5.3X_1 = 3.67X_2$ .....(1)

$3.67X_1 = 2.55X_2$ .....(2)

From (1) and (2), $\mathbf{X_1 = 0.69X_2}$

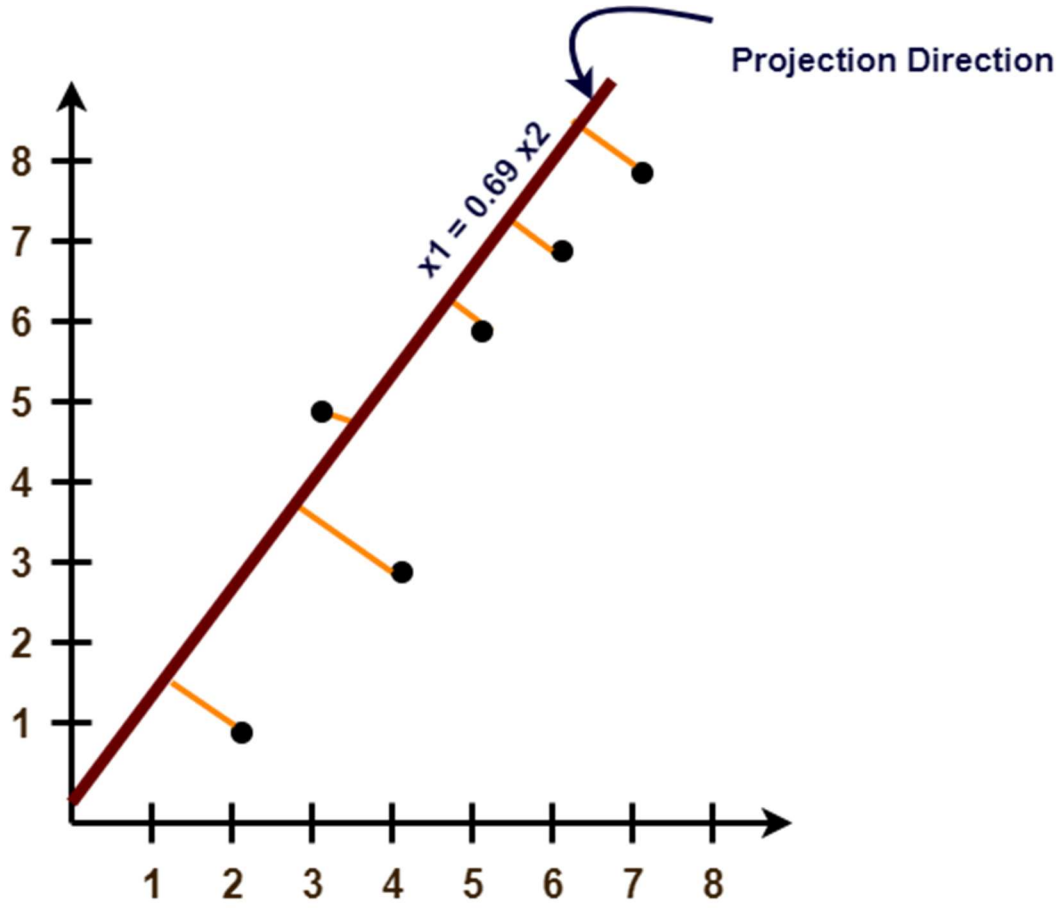
From (2), the eigen vector is-

$$\textbf{Eigen Vector :} \quad \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

Thus, principal component for the given data set is-

$$\textbf{Principal Component :} \quad \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

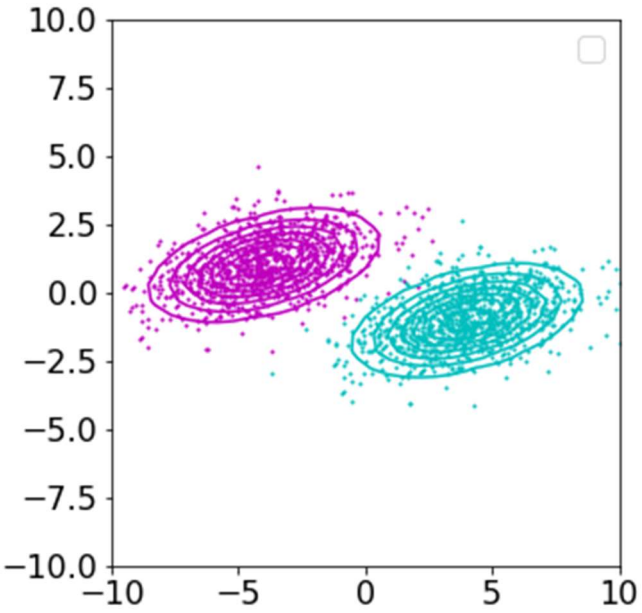
Lastly, we project the data points onto the new subspace as-



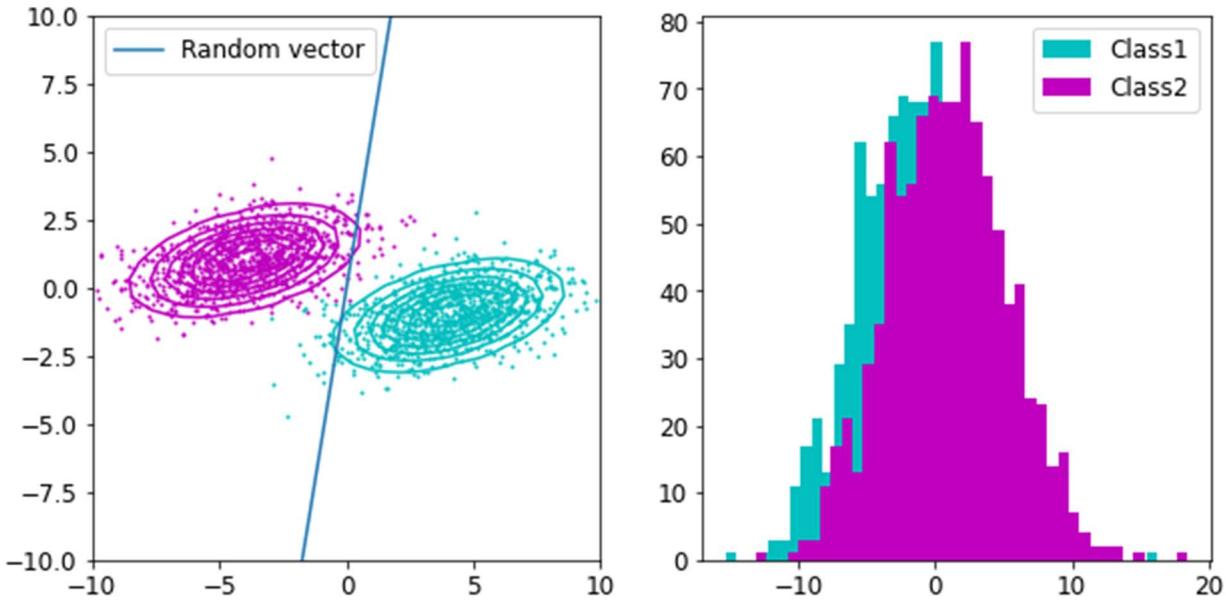
Fisher's Linear Discriminant

Fisher's linear discriminant can be used as a supervised learning classifier. Given labeled data, the classifier can find a set of weights to draw a decision boundary, classifying the data. **Fisher's linear discriminant attempts to find the vector that maximizes the separation between classes of the projected data.** Maximizing "separation" can be ambiguous. The criteria that Fisher's linear

discriminant follows to do this is to maximize the distance of the projected means and to minimize the projected within-class variance.

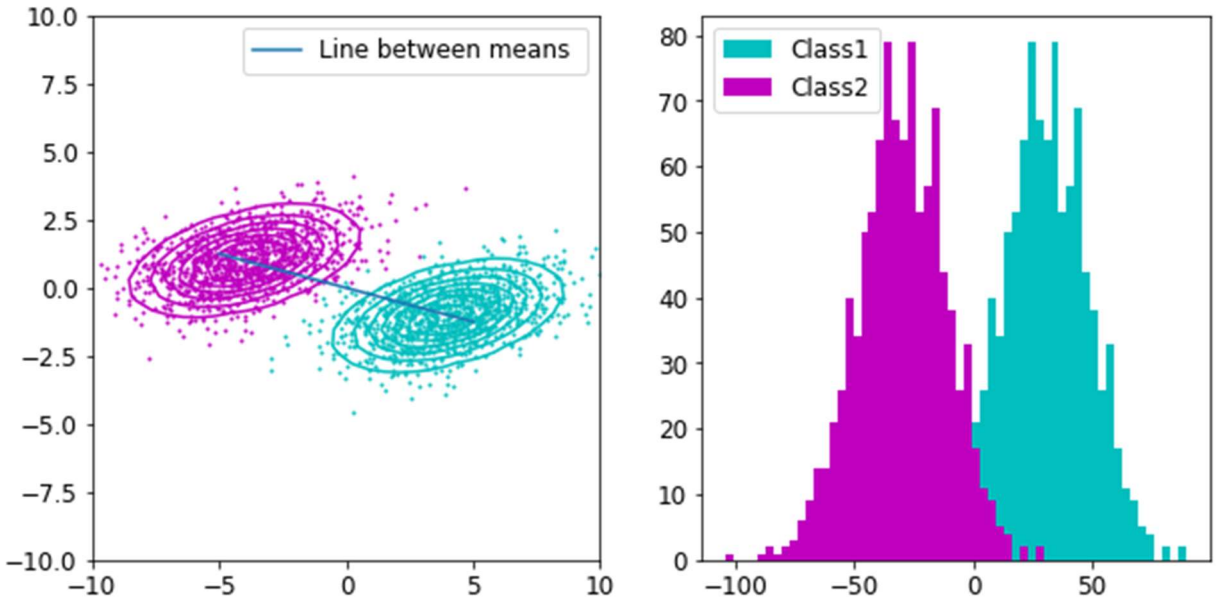


Here are two bivariate Gaussians with identical covariance matrices and distinct means. **We want to find the vector that best separates the projections of the data.** Let's draw a random vector and plot the projections.

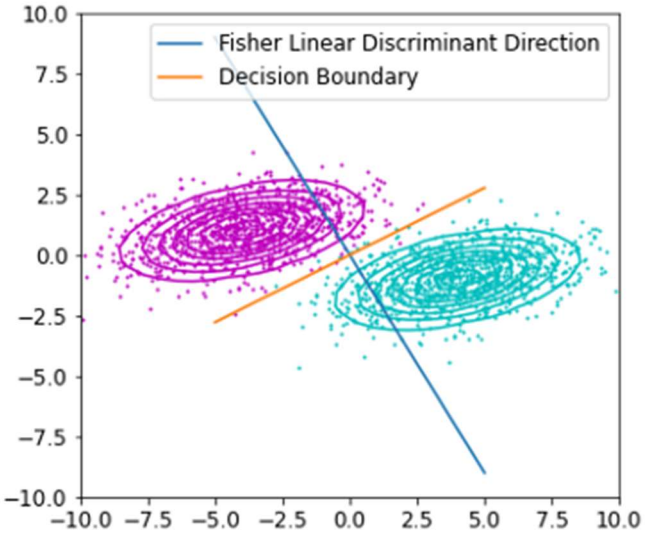


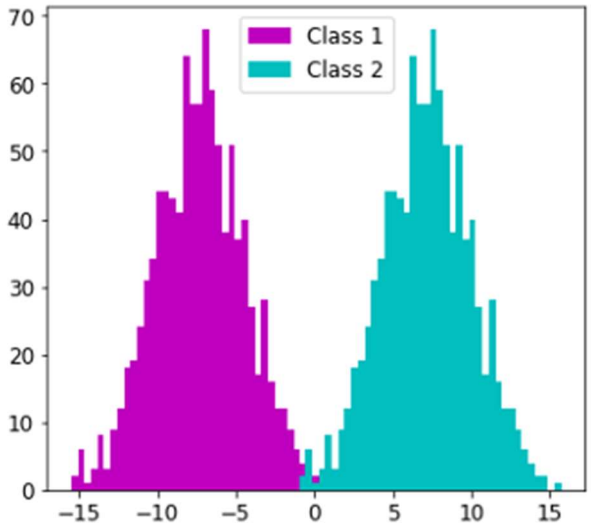
Remember we are looking at projections of the data onto the vector (the dot product of the weights vector and the data matrix) and not a decision boundary. The projections of the data onto this random weights vector can be plotted as a histogram (image on the right). As you can see when projecting the data onto the vector and drawing the histogram the two classes of data aren't well separated. The goal is to find the line that best separates the two distributions on the image on the right.

To separate the two distributions, we could first try to maximize the distance between the projected means, meaning the distributions are, on average, as far as possible from each other. Let's draw a line between the two means and plot the histogram of the projections onto that line.



That's quite a bit better, but the projections of the data are not fully separated yet. To fully separate them, Fisher's linear discriminant minimizes the within-class variance of the projections at the same time as maximizing the projections between the means. It tries to maximise the means as we discussed before to separate them, but also attempts to make the distributions as tight as possible. This allows for better separation as you'll see below.





As you can see the projections of the data are well separated. We can take an orthogonal vector from the weights vector to create a decision boundary. The decision boundary tells us that on either side of the boundary the data can be predicted to be one class or another. For multivariate gaussian distributions with identical covariance matrices, this yields an optimal classifier.

Fisher's criterion

Now that I visualized the problem we can formally express these concepts in equation form.

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

Where $m_1 - m_2 = \boldsymbol{w}^T(\boldsymbol{m_1} - \boldsymbol{m_2})$ and $s_k = \sum(\boldsymbol{w}^T\boldsymbol{x_n} - m_k)^2$

Maximising Fisher's criterion involves finding the weights vector that maximizes the equation above. Maximizing this equation is equivalent to what we saw before. Maximizing means maximizing the numerator (the distance between projected means) and minimizing the denominator (the within-class variance). We can substitute in the equations for the means and covariance and rewrite Fisher's criterion as follows.

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_w \boldsymbol{w}}$$

Where $S_B = (\boldsymbol{m_2} - \boldsymbol{m_1})(\boldsymbol{m_2} - \boldsymbol{m_1})^T$

And $S_w = \sum_{n \in C_1}(x_n - \boldsymbol{m_1})(x_n - \boldsymbol{m_1})^T + \sum_{n \in C_2}(x_n - \boldsymbol{m_2})(x_n - \boldsymbol{m_2})^T$

The above equation has a quadratic form in the numerator and denominator on the weights vector w. We can maximize J with respect to w by differentiating and equating to zero.

$$\frac{\partial J(w)}{\partial w} = \frac{(w^T S_w w)(2S_B w) - (w^T S_B w)(2S_w w)}{(w^T S_w w)^2} = 0$$

$$\leftrightarrow S_w \boldsymbol{w} = \alpha S_B \boldsymbol{w}$$

$$\leftrightarrow S_w \boldsymbol{w} = \alpha(\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T \boldsymbol{w}$$

$$\leftrightarrow w \propto S_w^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$$

We don't care about scalars since we can always normalize the vector later. The highlighted terms above are scalars and therefore are absorbed into the constant α or into the proportional symbol in the last line. From this, we know that the weights vector w maximizes fisher's criterion when it's proportional to the above expression. I used this proportionality to find Fisher's discriminant linear direction in the example earlier on.

Linear Discriminant Analysis (LDA)

Earlier on we projected the data onto the weights vector and plotted a histogram. This projection from a 2D space onto a line is reducing the dimensionality of the data, this is LDA. LDA uses Fisher's linear discriminant to reduce the dimensionality of the data whilst maximizing the separation between classes. It does this by maximizing the distance between means and minimizing within-class variance.

PCA vs LDA

So how does LDA compare to other dimensionality reduction techniques? Another very common way to reduce dimensionality is PCA, which maximizes the amount of information carried over onto smaller dimensions. Instead of Fisher's linear discriminant direction, PCA uses the principal components found through singular value decomposition. Principal components are the directions that maximize variation in the projected data (this does not take into account categories of data). LDA takes into account the categories in the data, whereas PCA does not.

Non Parametric Technique for Dimensionality reduction:-

A non-parametric numerosity reduction technique does not assume any model. The non-Parametric technique results in a more uniform reduction, irrespective of data size, but it may not achieve a high volume of data reduction like the parametric. There are at least four types of Non-Parametric data reduction techniques, Histogram, Clustering, Sampling, Data Cube Aggregation, and Data Compression.

PARZEN WINDOW

The main purpose of Parzen Window approach is to estimate a likelihood value, p(x), for each possible vector x. So that these likelihood values can be combined to produce a probability density function. At each iteration, a hypercube with edges of length **h_n** is placed and the center of the hypercube is set as the current **x** vector. After fitting the hypercube around x vector, every input x_i from the given samples is evaluated whether it's in the hypercube or not. This procedure is carried out by a window function **Φ**. The window function can be a discrete function such that it returns 1 if x_i is in the hypercube and 0 if not. And it can also be a continuous window such as Gaussian. In this way, the contribution of each sample will be continuous depending on the distance with respect to the center vector **x**. A simple formulation to find the likelihood p(x) for a given x vector is shown below:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\Phi\left(\frac{\vec{x}-\vec{x}_i}{h_n}\right)$$

Likelihood calculation for a given vector x

In the formula above; **Φ** stands for the window function, **x** is the vector that we seek to find the likelihood of, **x_i** is the **i**^th sample from the distribution, **h_n** is the edge length of the hypercube, **V_n** is the volume

of the hypercube and finally **n** is the number of samples in the dataset. If we summarize what's going on in this formula, we can say that the window function **Φ** is operated on each sample **x_i** and the output of the window is divided by the volume of the hypercube. At the end, each result is summed up and averaged by **n**, number of samples. This final value gives us the likelihood of the vector **x**!

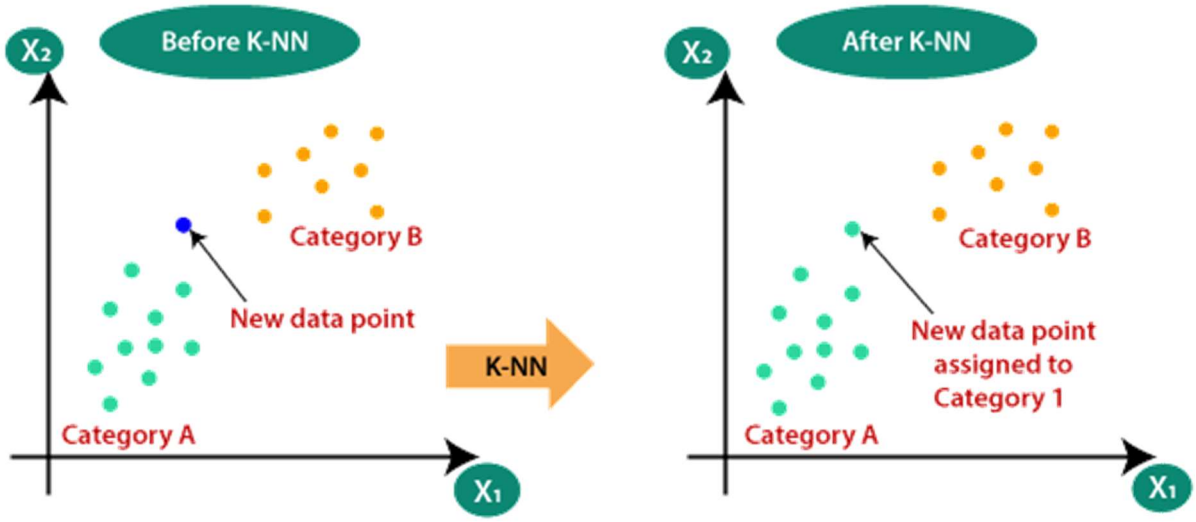
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

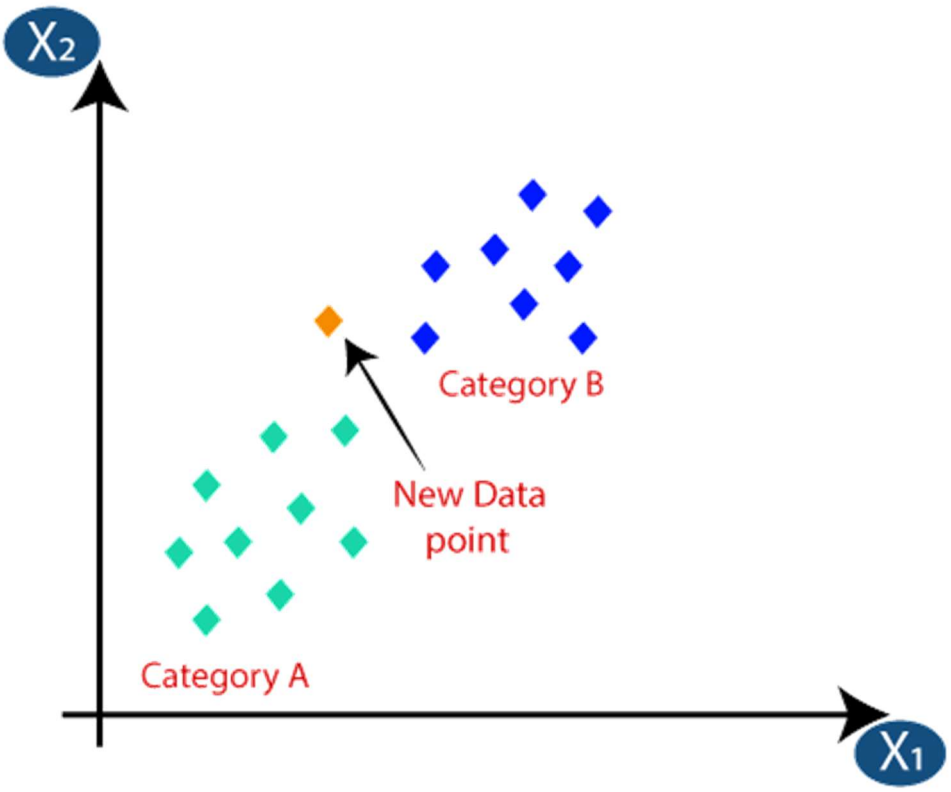


How does K-NN work?

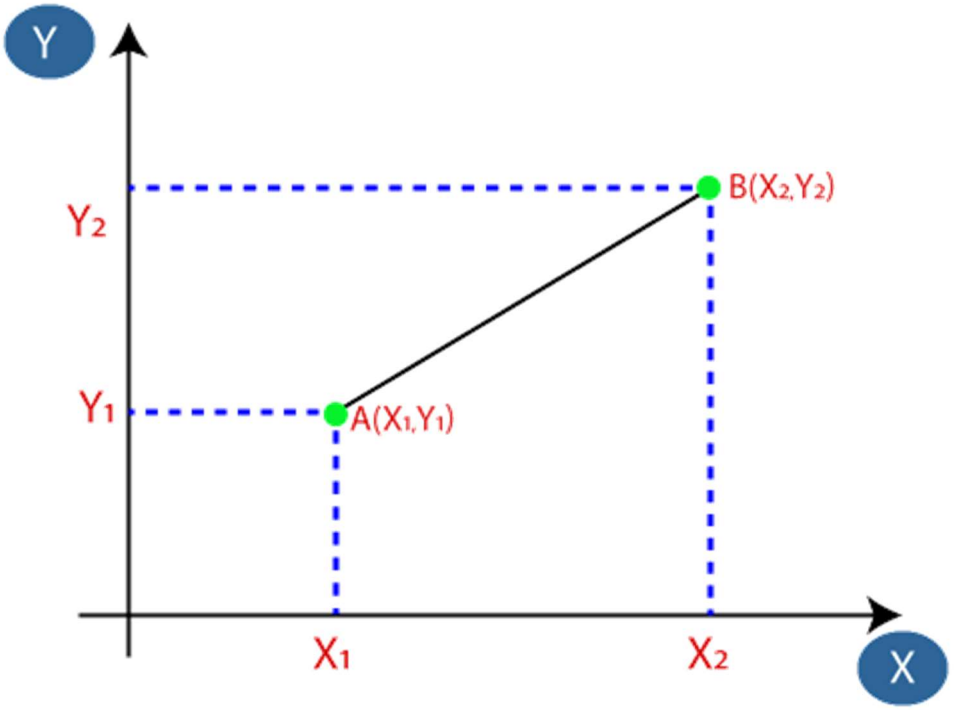
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

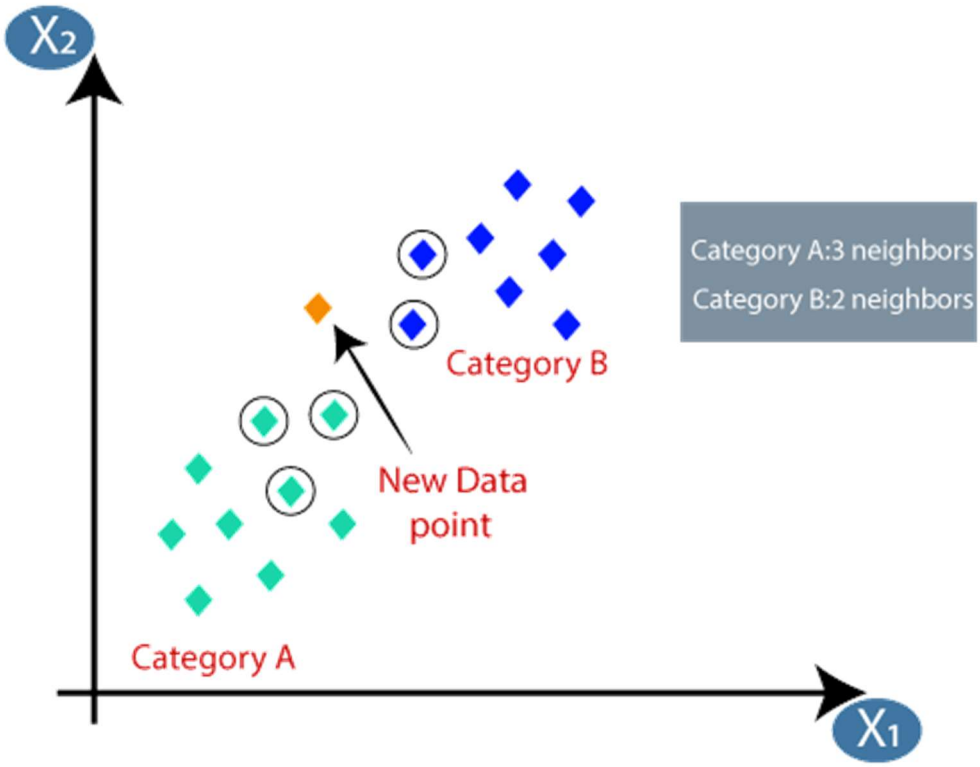


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.