**Data Transformation**

Data transformation is one of the fundamental steps in the part of data processing. When I first learnt the technique of feature scaling, the terms *scale*, *standardise*, and *normalis*e are often being used. However, it was pretty hard to find information about which of them I should use and also when to use.

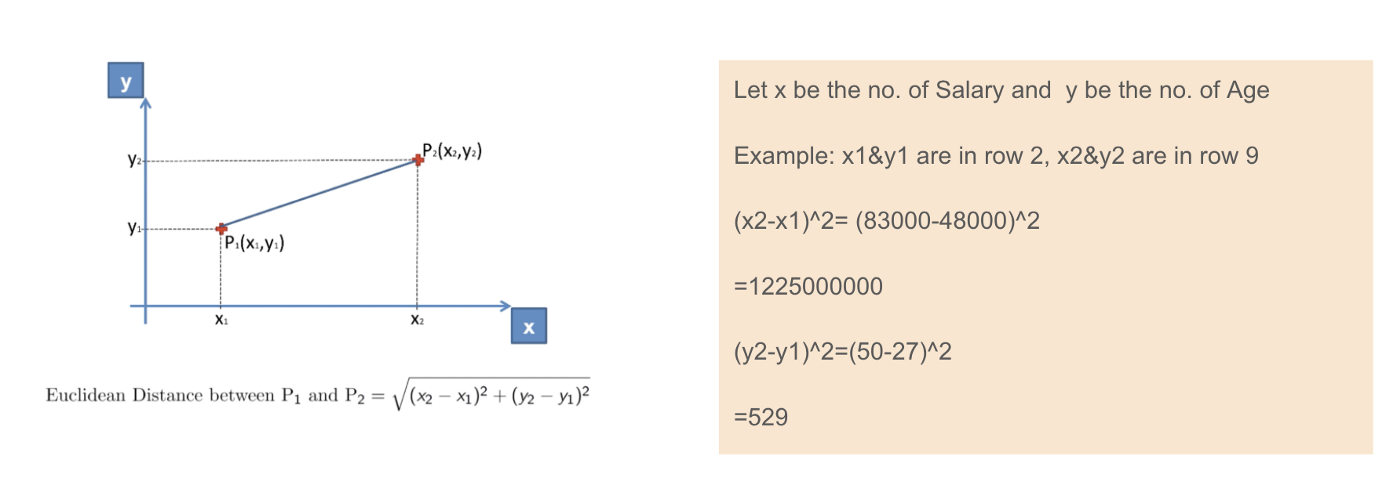
**What Does Feature Scaling Mean?**

 In practice, we often encounter different types of variables in the same dataset. A significant issue is that the range of the variables may differ a lot. Using the original scale may put more weights on the variables with a large range. In order to deal with this problem, we need to apply the technique of features rescaling to independent variables or features of data in the step of data pre-processing. The terms *normalisation* and *standardisation* are sometimes used interchangeably, but they usually refer to different things.

The goal of applying Feature Scaling is to make sure features are on almost the same scale so that each feature is equally important and make it easier to process by most ML algorithms

**Example**

 This is a dataset that contains an independent variable (Purchased) and 3 dependent variables (Country, Age, and Salary). We can easily notice that the variables are not on the same scale because the range of *Age* is from 27 to 50, while the range of *Salary* going from 48 K to 83 K. The range of *Salary* is much wider than the range of *Age*. This will cause some issues in our models since a lot of machine learning models such as k-means clustering and nearest neighbour classification are based on the Euclidean Distance.

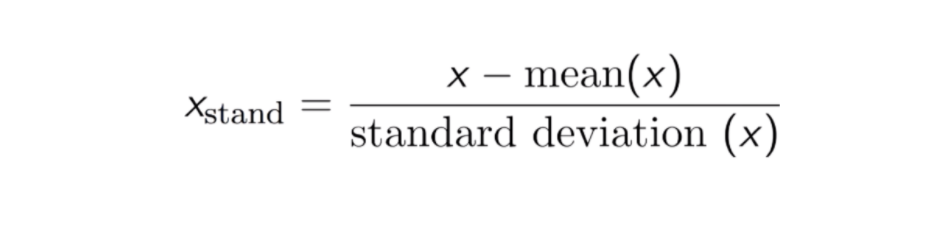
  
Euclidean distance application.

 Focusing on age and salary

When we calculate the equation of Euclidean distance, the number of (x2-x1)² is much bigger than the number of (y2-y1)² which means the Euclidean distance will be dominated by the salary if we do not apply feature scaling. The difference in Age contributes less to the overall difference. Therefore, we should use Feature Scaling to bring all values to the same magnitudes and, thus, solve this issue. To do this, there are primarily two methods called Standardisation and Normalisation.

**Standardisation**

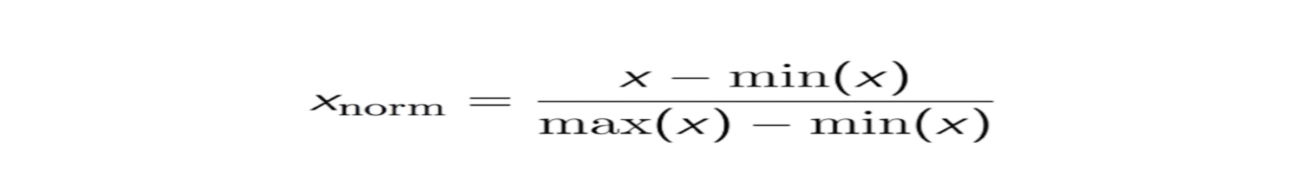
 The result of **standardization** (or **Z-score normalization**) is that the features will be rescaled to ensure the mean and the standard deviation to be 0 and 1, respectively. The equation is shown below:



This technique is to re-scale features value with the distribution value between 0 and 1 is useful for the optimization algorithms, such as gradient descent, that are used within machine learning algorithms that weight inputs (e.g., regression and neural networks). Rescaling is also used for algorithms that use distance measurements, for example, K-Nearest-Neighbours (KNN).

**Max-Min Normalization**

Another common approach is the so-called **Max-Min Normalization (**Min-Max scaling). This technique is to re-scales features with a distribution value between 0 and 1. For every feature, the minimum value of that feature gets transformed into 0, and the maximum value gets transformed into 1. The general equation is shown below:

  
The equation of Max-Min Normalization

**Standardisation vs Max-Min Normalization**

 In contrast to standardisation, we will obtain smaller standard deviations through the process of Max-Min Normalisation. Let me illustrate more in this area using the above dataset.



From the above graphs, we can clearly notice that applying Max-Min Nomaralisation in our dataset has generated smaller standard deviations (Salary and Age) than using Standardisation method. It implies the data are more concentrated around the mean if we scale data using Max-Min Nomaralisation.

As a result, if you have outliers in your feature (column), normalizing your data will scale most of the data to a small interval, which means all features will have the same scale but does not handle outliers well. Standardisation is more robust to outliers, and in many cases, it is preferable over Max-Min Normalisation.

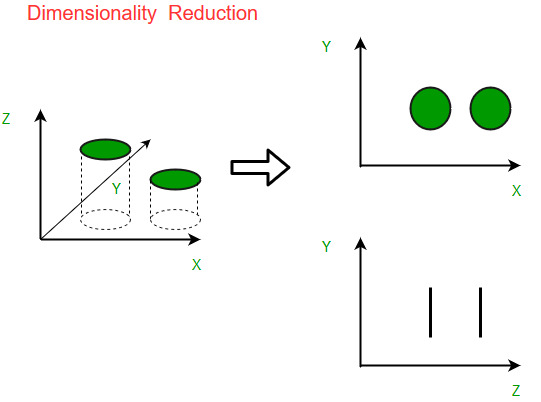
**Dimensionality Reduction**

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

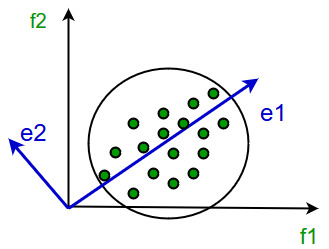
**Methods of Dimensionality Reduction**

The various methods used for dimensionality reduction include:

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

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

**Principal Component Analysis**

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.  It involves the following steps:

* Construct the covariance matrix of the data.
* Compute the eigenvectors of this matrix.
* Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

**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.

**Principal Component Analysis**

The main idea of [principal component analysis](https://www.projectpro.io/recipes/what-is-principal-component-analysis-statsmodels-library) (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

**How PCA works?**

**Step 1: Normalize the data**

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become 𝔁- and all Y become 𝒚-. This produces a dataset whose mean is zero.

**Step 2: Calculate the covariance matrix**

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

https://dezyre.gumlet.io/files.dezyre.com/images/Tutorials/Covariance+Matrix.JPG?w=750&dpr=1.0

Please note that Var[X1] = Cov[X1,X1] and Var[X2] = Cov[X2,X2].

**Step 3: Calculate the eigenvalues and eigenvectors**

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. ***ƛ*** is an eigenvalue for a matrix ***A*** if it is a solution of the characteristic equation:

***det( ƛI - A ) = 0***

Where, ***I*** is the identity matrix of the same dimension as ***A*** which is a required condition for the matrix subtraction as well in this case and ‘***det’***is the determinant of the matrix. For each eigenvalue ***ƛ***, a corresponding eigen-vector ***v***, can be found by solving:

***( ƛI - A )v = 0***

**Step 4: Choosing components and forming a feature vector:**

We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with *n* variables, then we have the corresponding *n* eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first *p* eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

*Feature Vector = (eig1, eig2)*

**Step 5: Forming Principal Components:**

This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

*NewData = FeatureVectorT x ScaledDataT*

Here,

*NewData*is the Matrix consisting of the principal components,

*FeatureVector*is the matrix we formed using the eigenvectors we chose to keep, and

*ScaledData*is the scaled version of original dataset

(‘T’ in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2x3 matrix has a transpose of size 3x2)