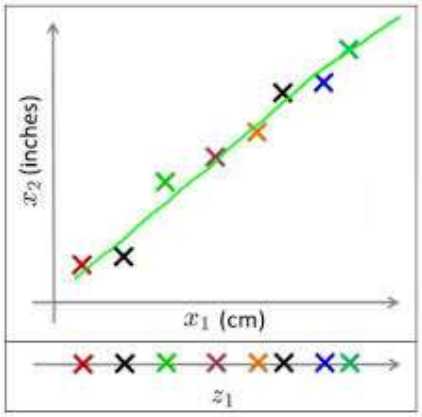
What are Dimension Reduction techniques?

Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely. These techniques are typically used while solving **machine learning problems** to obtain better features for a classification or regression task.

Let's look at the image shown below. It shows 2 dimensions x1 and x2, which are let us say measurements of several object in cm (x1) and inches (x2). Now, if you were to use both these dimensions in machine learning, they will convey similar information and introduce a lot of noise in system, so you are better of just using one dimension. Here we have converted the dimension of data from 2D (from x1 and x2) to 1D (z1), which has made the data relatively easier to explain.



In similar ways, we can reduce n dimensions of data set to k dimensions (k < n) . These k dimensions can be directly identified (filtered) or can be a combination of dimensions (weighted averages of dimensions) or new dimension(s) that represent existing multiple dimensions well.

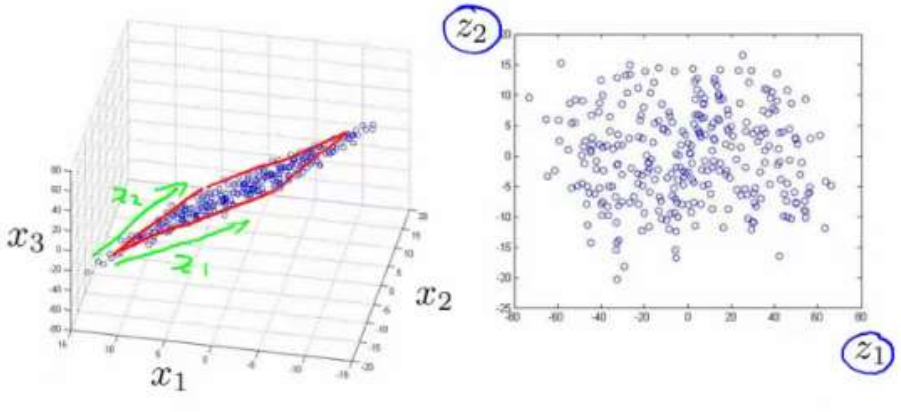
One of the most common application of this technique is **Image processing**. You might have come across this Facebook application – "**Which Celebrity Do You Look Like?**". But, have you ever thought about the algorithm used behind this?

Here's the answer: To identify the matched celebrity image, we use pixel data and each pixel is equivalent to one dimension. In every image, there are high number of pixels i.e. high number of dimensions. And every dimension is important here. You can't omit dimensions randomly to make better sense of your overall data set. In such cases, dimension reduction techniques help you to find the significant dimension(s) using various method(s).

What are the benefits of Dimension Reduction?

Let's look at the benefits of applying Dimension Reduction process:

- It helps in data compressing and reducing the storage space required
- It fastens the time required for performing same computations. Less dimensions leads to less computing, also less dimensions can allow usage of algorithms unfit for a large number of dimensions
- It takes care of multi-collinearity that improves the model performance. It removes redundant features. For example: there is no point in storing a value in two different units (meters and inches).
- Reducing the dimensions of data to 2D or 3D may allow us to plot and visualize it precisely. You can then observe patterns more clearly. Below you can see that, how a 3D data is converted into 2D. First it has identified the 2D plane then represented the points on these two new axis z1 and z2.



- It is helpful in noise removal also and as result of that we can improve the performance of models.

What are the common methods to perform Dimension Reduction?

There are many methods to perform Dimension reduction. I have listed the most common methods below:

1. Missing Values: While exploring data, if we encounter missing values, what we do? Our first step should be to identify the reason then impute missing values/ drop variables using appropriate methods. But, what if we have too many missing values? Should we impute missing values or drop the variables?

I would prefer the latter, because it would not have lot more details about data set. Also, it would not help in improving the power of model. Next question, is there any threshold of missing values for dropping a variable? It varies from case to case. If the information contained in the variable is not that much, you can drop the variable if it has more than ~40-50% missing values.

2. Low Variance: Let's think of a scenario where we have a constant variable (all observations have same value, 5) in our data set. Do you think, it can improve the power of model? Ofcourse NOT, because it has zero variance. In case of high number of dimensions, we should drop variables having low variance compared to others because these variables will not explain the variation in target variables.

3. Decision Trees: It is one of my favorite techniques. It can be used as a ultimate solution to tackle multiple challenges like missing values, outliers and identifying significant variables. It worked well in our Data Hackathon also. Several data scientists used decision tree and it worked well for them.

4. Random Forest: Similar to decision tree is Random Forest. I would also recommend using the in-built feature importance provided by random forests to select a smaller subset of input features. Just be careful that random forests have a tendency to bias towards variables that have more no. of distinct values i.e. favor numeric variables over binary/categorical values.

5. High Correlation: Dimensions exhibiting higher correlation can lower down the performance of model. Moreover, it is not good to have multiple variables of similar information or variation also known as **"Multicollinearity".** You can use *Pearson* (continuous variables) or *Polychoric* (discrete variables) correlation matrix to identify the variables with high correlation and select one of them using VIF (Variance Inflation Factor). Variables having higher value (VIF > 5) can be dropped.

6. Backward Feature Elimination: In this method, we start with all n dimensions. Compute the sum of square of error (SSR) after eliminating each variable (n times). Then, identifying variables whose removal has produced the smallest increase in the SSR and removing it finally, leaving us with *n-1* input features.

Repeat this process until no other variables can be dropped. Reverse to this, we can use "**Forward Feature Selection**" method. In this method, we select

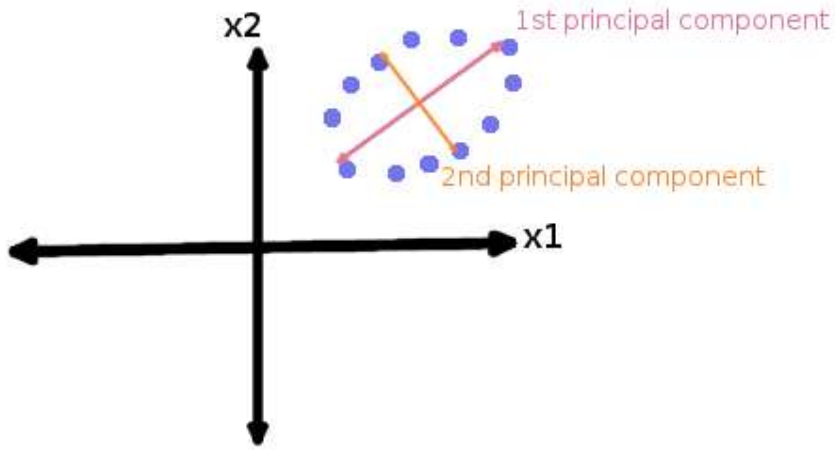
one variable and analyse the performance of model by adding another variable. Here, selection of variable is based on higher improvement in model performance.

7. Factor Analysis: Let's say some variables are highly correlated. These variables can be grouped by their correlations i.e. all variables in a particular group can be highly correlated among themselves but have low correlation with variables of other group(s). Here each group represents a single underlying construct or factor. These factors are small in number as compared to large number of dimensions. However, these factors are difficult to observe. There are basically two methods of performing factor analysis:

- EFA (Exploratory Factor Analysis)
- CFA (Confirmatory Factor Analysis)

8. Principal Component Analysis (PCA): In this technique, variables are transformed into a new set of variables, which are linear combination of original variables. These new set of variables are known as **principle components.** They are obtained in such a way that first principle component accounts for most of the possible variation of original data after which each succeeding component has the highest possible variance.

The second principal component must be orthogonal to the first principal component. In other words, it does its best to capture the variance in the data that is not captured by the first principal component. For two-dimensional dataset, there can be only two principal components. Below is a snapshot of the data and its first and second principal components. You can notice that second principle component is orthogonal to first principle component.



The principal components are sensitive to the scale of measurement, now to fix this issue we should always standardize variables before applying PCA. Applying PCA to your data set loses its meaning. If interpretability of the results is important for your analysis, PCA is not the right technique for your project.

Support Vector Machine

- Support vector machines are considered by some people to be the best stock classifier.
- This means you can take the classifier in its basic form and run it on the data, and the results will have low error rates.
- Support vector machines make good decisions for data points that are outside the training set.

Separating data with the maximum margin

To introduce the subject of support vector machines I need to explain a few concepts. Consider the data in frames A–D in figure 6.1; could you draw a straight line to put all of the circles on one side and all of the squares on another side? Now consider the data in figure 6.2, frame A. There are two groups of data, and the data points are separated enough that you could draw a straight line on the figure with all the points of one class on one side of the line and all the points of the other class on the other side of the line. If such a situation exists, we say the data is linearly separable. Don't worry if this assumption seems too perfect. We'll later make some changes where the data points can spill over the line.

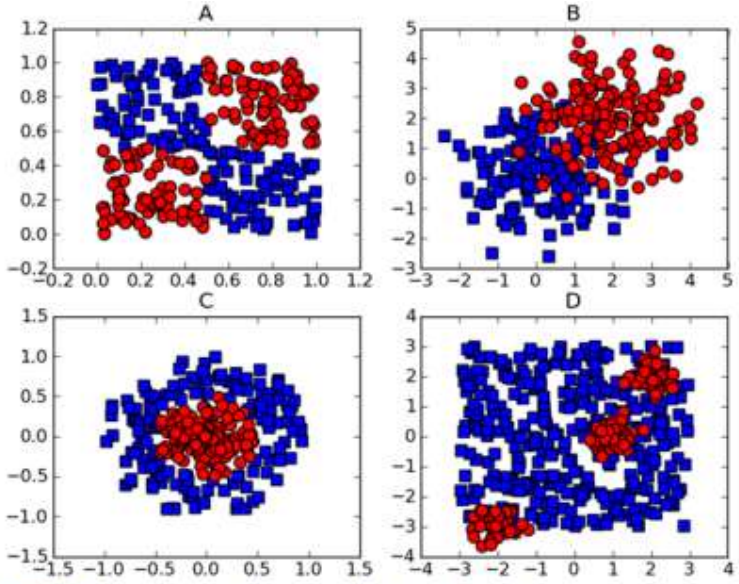


Figure 6.1 **Four examples of datasets that aren't linearly separable**

The line used to separate the dataset is called a separating hyperplane. In our simple 2D plots, it's just a line. But, if we have a dataset with three dimensions, we need a plane to separate the data; and if we have data with 1024 dimensions, we need something with 1023 dimensions to separate the data. What do you call something with 1023 dimensions? How about N-1 dimensions? It's called a hyperplane. The hyperplane is our decision boundary. Everything on one side belongs to one class, and everything on the other side belongs to a different class.

We'd like to make our classifier in such a way that the farther a data point is from the decision boundary, the more confident we are about the prediction we've made. Consider the plots in figure 6.2, frames B–D. They all separate the data, but which one does it best? Should we minimize the average distance to the separating hyperplane? In that case, are frames B and C any better than frame D in figure 6.2? Isn't something like that done with best-fit lines? Yes, but it's not the best idea here. We'd like to find the point closest to the separating hyperplane and make sure this is as far away from the separating line as possible. This is known as margin. We want to have the greatest possible margin, because if we made a mistake or trained our classifier on limited data, we'd want it to be as robust as possible.

The points closest to the separating hyperplane are known as support vectors. Now that we know that we're trying to maximize the distance from the separating line to the support vectors, we need to find a way to optimize this problem.

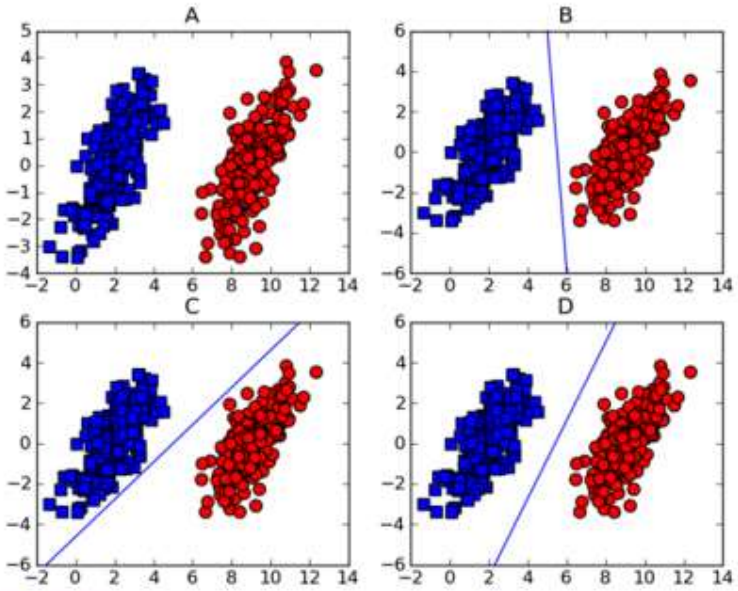


Figure 6.2 **Linearly separable data is shown in frame A. Frames B, C, and D show possible valid lines separating the two classes of data.**

Finding the maximum margin

How can we measure the line that best separates the data? To start with, look at figure 6.3. Our separating hyperplane has the form w^Tx+b. If we want to find the distance from A to the separating plane, we must measure normal or perpendicular to the line. This is given by $|w^T$x+b|/ ||w||. The constant b is just an offset like w0 in logistic regression. All this w and b stuff describes the separating line, or hyperplane, for our data. Now, let's talk about the classifier.

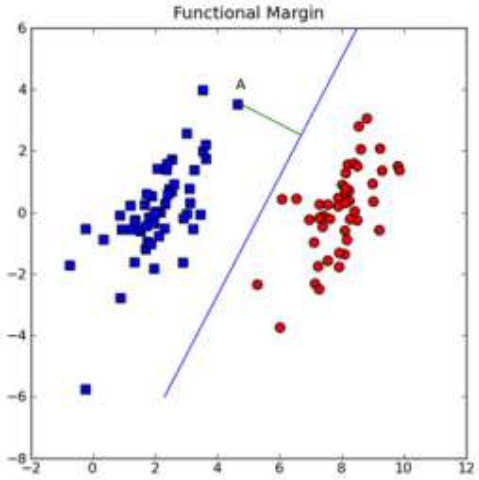


Figure 6.3 The distance from point A to the separating plane is measured by a line normal to the separating plane.

Temporal Difference Learning

Temporal difference (TD) learning is a prediction-based machine learning method. It has primarily been used for the reinforcement learning problem, and is said to be "a combination of Monte Carlo ideas and dynamic programming (DP) ideas."[1] TD resembles a Monte Carlo method because it learns by sampling the environment according to some *policy*, and is related to dynamic programming techniques as it approximates its current estimate based on previously learned estimates (a process known as bootstrapping). The TD learning algorithm is related to the temporal difference model of animal learning.[2]

As a prediction method, TD learning considers that subsequent predictions are often correlated in some sense. In standard supervised predictive learning, one learns only from actually observed values: A prediction is made, and when the observation is available, the prediction mechanism is adjusted to better match the observation. As elucidated by Richard Sutton, the core idea of TD learning is that one adjusts predictions to match other, more accurate, predictions about the future.[3] This procedure is a form of bootstrapping, as illustrated with the following example:

> "Suppose you wish to predict the weather for Saturday, and you have some model that predicts Saturday's weather, given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your models. However, when it is, for example, Friday, you should have a pretty good idea of what the weather would be on Saturday - and thus be able to change, say, Monday's model before Saturday arrives."