प्रशिक्षण मैनुअल

Training Manual

कृषि में डेटा साइंस प्रशिक्षण कार्यक्रम

(4 सितंबर - 15 सितंबर, 2023)

Data Science in Agriculture

(September 04 – 15, 2023)

हर कदम, हर डगर
किसानों का हमसफर
भारतीय कृषि अनुसंधान परिषद

Agrisearch with a human touch

Compiled and Edited by

Rajender Parsad, Alka Arora, Chandan Deb, Sapna Nigam, Upendra Kumar Pradhan & Mrinmoy Ray

भा.कृ.अनु.प. – भारतीय कृषि सांख्यिकी अनुसंधान संस्थान
लाइब्रेरी एवेन्यू, पूसा, नई दिल्ली-110 012
ICAR – Indian Agricultural Statistics Research Institute
Library Avenue, Pusa, New Delhi-110 012
www.iasri.res.in

2023

CONTENTS

Linear Regression, Multiple Linear Regression, Model Selection Criteria, Regularized Linear Modeling

Ramasubramanian V.
ICAR-National Academy of Agricultural Research Management, Hyderabad
r.subramanian@icar.gov.in

1. Introduction

Regression analysis is one of the most widely used techniques for studying relationships involving multiple variables for analysing data by expressing a relationship between a variable of interest (the response) and a set of related predictor variables. The regression models include both linear and non-linear approaches assuming appropriate functional forms. A good account on regression analysis and related topics can be found in Draper and Smith (1998), Montgomery *et al*. (2001), Chatterjee and Hadi (2006) etc.

Regression analysis is a statistical methodology that utilizes the relation between two or more quantitative variables so that one variable can be predicted from the other, or others. This methodology is widely used in business, the social and behavioral sciences, the biological sciences including agriculture and fishery research. For example, fish weight at harvest can be predicted by utilizing the relationship between fish weights and other growth affecting factors like water temperature, dissolved oxygen, free carbon dioxide etc.

Regression analysis serves three major purposes: (1) description (2) control and (3) prediction. We frequent use equations to summarize or describe a set of data. Regression analysis is helpful in developing such equations. For example, we may collect a considerable amount of fish growth data and data on a number of biotic and abiotic factors, and a regression model would probably be a much more convenient and useful summary of those data than a table or even a graph. Besides prediction, regression models may be used for control purposes. A cause and effect relationship may not be necessary if the equation is to be used only for prediction. In this case, it is only necessary that the relationships that existed in the original data used to build the regression equation are still valid.

A functional relation between two variables is expressed by a mathematical formula. If X denotes the independent variable and Y the dependent variable, a functional relation is of the form $Y = f(X)$. Given a particular value of X, the function f indicates the corresponding value of Y. A statistical relation, unlike a function is not a perfect one. In general, the observations for a statistical relation do not fall directly on the curve of relationship. Depending on the nature of the relationships between X and Y, regression approach may be classified into two broad categories viz., linear regression models and nonlinear regression models. The response variable is generally related to other causal variables through some parameters. The models that are linear in these parameters are known as linear models, whereas in nonlinear models parameters are appear nonlinearly. Linear models are generally satisfactory approximations for most regression applications. There are occasions, however, when an empirically indicated or a theoretically

justified nonlinear model is more appropriate. In the present lecture we shall consider fitting of linear models only.

In this write-up, regression model fitting, some of the detection techniques which are useful in detecting the problem of multicollinearity between the so-called 'independent variables' and also outlier detection in data are discussed. Linear regression with qualitative regressor variables is also discussed. Moreover, variable selection procedures, goodness of fit measures for model adequacy and validation are also discussed. In addition, the concept and purpose of regularization in linear modelling is also outlined.

2. Multiple linear regression modelling

Let the response variable (variable of interest) be denoted by Y and the set of predictor variables, by X_1 , X_2 , …, X_p, where p denotes the number of predictor variables. The true relationship between Y and (X_1, X_2 ,…,X_p) can be approximated by a multiple linear regression model given by $Y=\beta_0 +\beta_1 X_1 +\beta_2 X_2 +\ldots. +\beta_p X_p + \varepsilon$. Here ß$_0$ and ß$_i$ (i=1,2,…,p) are parameters to be estimated and ε is random error. Some assumptions are made about this model like the relationship of the response Y to the predictors X_1, X_2,…, X_p is linear in the regression parameters β_0, β_1,…, β_p , the errors are assumed to be independently and identically distributed (iid) normal random variables with mean zero and a common variance σ , the errors are independent of each other (their pair-wise covariances are zero) and that the predictor variables X_1, X_2,…, X_p are non-random and measured without error.

Francis Galton (in 1880's) coined the term 'regression' to refer to tall parents tending to beget offsprings which were not taller than their parents (also it was noted that parents who were not so taller in height got children who were somewhat taller than their shorter parents). Thus it was observed that mean filial regression towards mediocrity was directly proportional to the parental deviation from it. The cases in point were size of seedlings, Parents height (X) & Child's height (Y) etc. In today's model fitting situations, no "regression" in original sense but the same term prevailed. Thus regression analysis meant the average relationship between variables studied.

We consider a basic linear model where there is only one predictor variable and the regression function is linear. Model with more than one predictor variable is straight forward. The model can be stated as follows: $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$, where Y_i is the value of the response variable in the i^{th} trial; β_0 and β_1 are parameters, X_i is a known constant, namely, the value of the predictor variable in the i^{th} trial, ε_i is a random error term with mean zero and variance σ^2 and ε_i and ε_j are uncorrelated so that their covariance is zero. The above regression model is said to be simple, linear in the parameters, and linear in the predictor variable. It is "simple" in that there has only one predictor variable, "linear in the parameters" because no parameters appears as an exponent or its multiplied or divided by another parameter, and "linear in predictor variable" because this variable appears only in the first power. The parameters β_0 and β_1 in regression model are called regression coefficients, β_1 is the slope of the regression line. It indicates the change in the mean of the probability distribution of *Y* per unit increase in *X*. The parameter β_0 in *Y* intercept of the regression line. When the scope of the model includes $X = 0$, β_0 gives the mean of the probability distribution of *Y* at $X = 0$. When the scope of the model does not cover $X = 0$, β_0 does not have any particular meaning as a separate term in the regression model.

To find "good" estimates of the regression parameters β_0 and β_1, we employ the method of least squares. For each observation (X_i, Y_i), for each case, the method of least squares considers the deviation of Y from its expected value. In particular, the method of least squares requires that we consider the sum of such n squared deviations. According to the method of least squares, the estimators of β_0 and β_1 are those values b_0 and b_1, respectively, that minimize this criterion for the given observations.

Using the analytical approach, it can be shown for regression model (1) that the values of b_0 and b_1 that minimizes *this Error function* for any particular set of sample data are given by the following simultaneous equations:

$$\sum_{i=1}^{n} Y_i = nb_0 + b_1 \sum_{i=1}^{n} X_i$$

$$\sum_{i=1}^{n} X_i Y_i = b_0 \sum_{i=1}^{n} X_i + b_1 \sum_{i=1}^{n} X_i^2 .$$

These two equations are called normal equations and can be solved for b_0 and b_1:

$$b_1 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{n} (X_i - \overline{X})^2}$$

$$b_0 = \frac{1}{n}\left(\sum_{i=1}^{n} Y_i - b_1 \sum_{i=1}^{n} X_i\right) = \overline{Y} - b_1 \overline{X},$$

where $\overline{X}$ and $\overline{Y}$ are the means of the X_i and the Y_i observations, respectively.

Once the parameters estimates are obtained, the fitted line would be

$$\hat{Y}_i = b_0 + b_1 X_i$$

The ith residual is the difference between the observed value Y_i and the corresponding fitted value $\hat{Y}_i$, i.e., $e_i = Y_i - \hat{Y}_i$.

The estimated regression line fitted by the method of least squares has a number of properties worth noting.

1. The sum of the residuals is zero, $\sum_{i=1}^{n} e_i = 0$.
2. Sum of the squared residuals, $\sum_{i=1}^{n} e_i^2$, is a minimum.
3. Sum of the observed values Y_i equals the sum of the fitted values $\hat{Y}_i$, $\sum_{i=1}^{n} Y_i = \sum_{i=1}^{n} \hat{Y}_i$.
4. Sum of the weighted residuals is zero, weighted by the level of the predictor variable in the i^{th} trial: $\sum_{i=1}^{n} X_i e_i = 0$.

5. Sum of the weighted residuals is zero, weighted by the fitted value of the response variable in the i[th] trial: $\sum_{i=1}^{n} \hat{Y}_i e_i = 0$.
6. The regression line always goes through the points ($\overline{X}, \overline{Y}$).

The variance σ^2 of the error terms ε_i in regression model needs to be estimated to obtain an indication of the variability of the probability distribution of *Y*. In addition, a variety of inferences concerning the regression function and the prediction of *Y* require an estimate of σ^2. Denote by SSE = $\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} e_i^2$, is the error sum of squares or residual sum of squares. Then an estimate of σ^2 is given by $\hat{\sigma}^2 = \frac{SSE}{n-p}$, where p is the total number of parameters involved in the model. We also denote this quantity by MSE.

Frequently, we are interested in drawing inferences about β_1, the slope of the regression line. At times, tests concerning β_1 are of interest, particularly one of the form:

$H_0 = \beta_1 = 0$

$H_1 = \beta_1 \neq 0$

The reason for interest in testing whether or not $\beta_1 = 0$ is that, when $\beta_1 = 0$, there is no linear association between *Y* and *X*. For normal error regression model, the condition $\beta_1 = 0$ implies even more than no linear association between *Y* and *X*. $\beta_1 = 0$ for the normal error regression model implies not only that there is no linear association between *Y* and *X* but also that there is no relation of any kind between *Y* and *X*, since the probability distribution of *Y* are then identical at all levels of *X*.

An explicit test of the alternatives is based on the test statistic:

$$t = \frac{b_1}{s(b_1)},$$

where is the standard error of b_1 and calculated as $s(b_1) = \sqrt{\frac{MSE}{\sum_{i=1}^{n} (X_i - \overline{X})^2}}$.

The decision rule with this test statistic when controlling level of significance at α is

if $|t| \leq t(1-\alpha/2; n-p)$, conclude H_0,

if $|t| > t(1-\alpha/2; n-p)$, conclude H_1.

Similarly testing for other parameters can be carried out.

2.1 Adequacy and validation of regression models

As mentioned previously, many assumptions have to hold good in regression analysis such as the relationship between y and x's is linear., the errors have zero mean and constant variance, the

errors are uncorrelated, the errors are normally distributed etc. For checking whether these assumptions are adequately satisfied by any fitted regression model, residual analysis is resorted to. Residuals are nothing but differences between the observations and the corresponding fitted values. Residuals have zero mean and approximate average variance as "Error Mean Sum of Squares" from the regression ANOVA. Sometimes the standardised residuals are used. Residual plots are useful for detecting validity of assumptions on errors and model adequacy. Some important residual plots for detecting model inadequacies are stated in brief. Plot of residuals against fitted values - if this plot indicates that the residuals can be contained in a horizontal band, then there are no obvious model defects. If not so, transformations on the regressors and/or the response variable may be required. Plot of standardised residuals against independent variable with no apparent trend can also be taken as evidence of correct model specification.

Lack of normality and non-constant error variance frequently go hand in hand. Fortunately, it is often the case that the same transformation that helps stabilize the variance is also helpful in approximately normalizing the error terms. It is therefore, desirable that the transformation for stabilizing the error variance be utilized first, and then the residuals studied to see if serious departures from normality are still present.

We shall consider following six important types of departures from linear regression model with normal errors:

(i) Nonlinearity of Regression Model

Whether a linear regression function is appropriate for the data being analyzed can be studied from a residual plot against the predictor variable or equivalently from a residual plot against the fitted values. Figure 1(a) shows a prototype situation of the residual plot against X when a linear regression model is appropriate. The residuals then fall within a horizontal band centred around 0, displaying no systematic tendencies to be positive and negative.

Figure 1(b) shows a prototype situation of a departure from the linear regression model that indicates the need for a curvilinear regression function. Here the residuals tend to vary in a systematic fashion between being positive and negative.

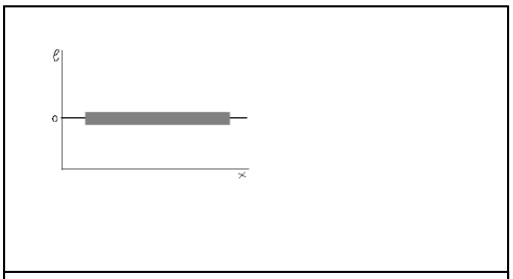


Fig. 1(a)

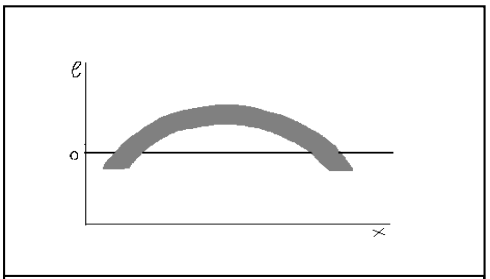
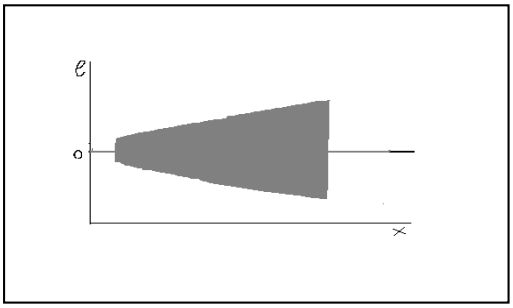
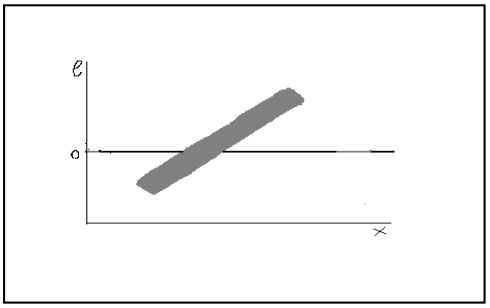
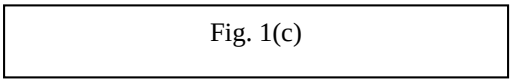
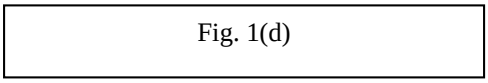


Fig. 1(b)









(ii) Non-constancy of Error Variance

Plots of residuals against the predictor variable or against the fitted values are not only helpful to study whether a linear regression function is appropriate but also to examine whether the variance of the error terms is constant. The prototype plot in Figure 1(a) exemplifies residual plots when error term variance is constant. Figure 1(c) shows a prototype picture of residual plot when the error variance increases with X. In many biological science applications, departures from constancy of the error variance tend to be of the "meghaphone" type.

(iii) Presence of Outliers

Outliers are extreme observations. Residual outliers can be identified from residual plots against X or $\hat{Y}$. Outliers can create great difficulty. When we encounter one, our first suspicion is that the observation resulted from a mistake or other extraneous effect. On the other hand, outliers may convey significant information, as when an outlier occurs because of an interaction with another predictor omitted from the model. A safe rule frequently suggested is to discard an outlier only if there is direct evidence that it represents in error in recording, a miscalculation, a malfunctioning of equipment, or a similar type of circumstances.

When outlying observations are present, use of the least squares and maximum likelihood estimates for regression model may lead to serious distortions in the estimated regression function. To test whether outlying observations are present, many measures such as elements of 'Hat Matrix', weighted sum of squared deviations, Cook's distance, 'DFFITS', 'DFBETAS', 'COVRATIO' etc. are employed. When the outlying observations do not represent recording errors and should not be discarded, it may be desirable to use an estimation procedure that places less emphasis on such outlying observations. Robust Regression falls under such methods.

Some tests for outlying observations are given below:

(a) *Elements of Hat Matrix* : The Hat matrix is defined as $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$, X is the matrix for explanatory variables. The larger values reflect data points are outliers.

(b) *$WSSD_i$*: $WSSD_i$ is an important statistic to locate points that are remote in *x*-space. $WSSD_i$ measures the weighted sum of squared distance of the i^{th} point from the center of the data. Generally if the $WSSD_i$ values progress smoothly from small to large, there are probably no extremely remote points. However, if there is a sudden jump in the magnitude of $WSSD_i$, this often indicates that one or more extreme points are present.

(c) *Cook's D_i:* Cook's D_i is designed to measure the shift in $\hat{y}$ when i^{th} obsevation is not used in the estimation of parameters. D_i follows approximately $F_{(p,n-p-1)}(1\text{-}\alpha)$. Lower 10% point

of this distribution is taken as a reasonable cut off (more conservative users suggest the 50% point). The cut off for D_i can be taken as $\frac{4}{n}$.

(d) *DFFITS$_i$: DFFIT* is used to measure difference in ith component of $\left(\hat{y} - \hat{y}_{(i)}\right)$. It is suggested that $DFFITS_i \geq 2\left(\frac{p+1}{n}\right)^{1/2}$ may be used to flag off influential observations.

(e) **DFBETAS** $_{j(i)}$: Cook's D_i reveals the impact of ith observation on the entire vector of the estimated regression coefficients. The influential observations for individual regression coefficient are identified by $DFBETAS_{j(i)}, j = 1,2,...,p+1$, where each $DFBETAS_{j(i)}$ is the standardized change in b_j when the ith observation is deleted.

(f) ***COVRATIO*** $_i$*:*The impact of the ith observation on variance-covariance matrix of the estimated regression coefficients is measured by the ratio of the determinants of the two variance-covariance matrices. Thus, COVRATIO reflects the impact of the ith observation on the precision of the estimates of the regression coefficients. Values near 1 indicate that the ith observation has little effect on the precision of the estimates. A value of COVRATIO greater than 1 indicates that the deletion of the ith observation decreases the precision of the estimates; a ratio less than 1 indicates that the deletion of the observation increases the precision of the estimates. Influential points are indicated by $\left|COVRATIO_i - 1\right| > \frac{3(p+1)}{n}$.

(g) ***FVARATIO*** $_i$***:*** The statistic detects change in variance of $\hat{y}_i$ when an observation is deleted. A value near 1 indicates that the ith observation has negligible effect on variance of y_i. A value greater than 1 indicates that deletion of the ith observation decreases the precision of the estimates, a value less than one increases the precision of the estimates.

(iv) Non-independence of Error Terms

Whenever data are obtained in a time sequence or some other type of sequence, such as for adjacent geographical areas, it is good idea to prepare a sequence plot of the residuals. The purpose of plotting the residuals against time or some other type of sequence is to see if there is any correlation between error terms that are near each other in the sequence. A prototype residual plot showing a time related trend effect is presented in Figure 1(d), which portrays a linear time related trend effect. When the error terms are independent, we expect the residuals in a sequence plot to fluctuate in a more or less random pattern around the base line 0.

(v) Non-normality of Error Terms

Small departures from normality do not create any serious problems. Major departures, on the other hand, should be of concern. The normality of the error terms can be studied informally by examining the residuals in a variety of graphic ways.

Comparison of frequencies: when the number of cases is reasonably large is to compare actual frequencies of the residuals against expected frequencies under normality. For example, one can determine whether, say, about 90% of the residuals fall between $\pm 1.645\ \sqrt{MSE}$.

Normal probability plot: Still another possibility is to prepare a normal probability plot of the residuals. Here each residual is plotted against its expected value under normality. A plot that is nearly linear suggests agreement with normality, whereas a plot that departs substantially from linearity suggests that the error distribution is not normal.

(vi) Omission of Important Predictor Variables

Residuals should also be plotted against variables omitted from the model that might have important effects on the response. The purpose of this additional analysis is to determine whether there are any key variables that could provide important additional descriptive and predictive power to the model. The residuals are plotted against the additional predictor variable to see whether or not the residuals tend to vary systematically with the level of the additional predictor variable.

Thus, if the simple regression model is not appropriate for a data set, employ some transformation on the data so that regression model is appropriate for the transformed data. When the regression function is not linear, a direct approach is to modify regression model by altering the nature of the regression function. For instance, a quadratic regression function might be used: $Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \varepsilon_i$

or an exponential regression function: $Y_i = \gamma_0 \gamma_1^{X_i} + \varepsilon_i$. When the nature of the regression function is not known, exploratory analysis that does not require specifying a particular type of function is often useful.

When the error variance is not constant but varies in a systematic fashion, a direct approach is to modify the method to allow for this and use the method of weighted least squares to obtain the estimates of the parameters.

Transformations is another way in stabilizing the variance. We first consider transformation for linearizing a nonlinear regression relation when the distribution of the error terms is reasonably close to a normal distribution and the error terms have approximately constant variance. In this situation, transformation on *X* should be attempted. The reason why transformation on *Y* may not be desirable here is that a transformation on *Y*, such as $Y' = \sqrt{Y}$, may materially change the shape of the distribution and may lead to substantially differing error term variance.

Following transformations are generally applied for stabilizing variance.
(1) when the error variance is rapidly increasing $Y' = \log_{10} Y$ or $Y' = \sqrt{Y}$
(2) when the error variance is slowly increasing, $Y' = Y^2$ or $Y' = Exp(Y)$
(3) when the error variance is decreasing, $Y' = 1/Y$ or $Y' = Exp(-Y)$.

It is difficult to determine, which transformation of Y is most appropriate for correcting skewness of the distributions of error terms, unequal error variance, and nonlinearity of the regression function. The Box-Cox transformation automatically identifies a transformation from the family of power transformations on Y. The family of power transformations is of the form: $Y' = Y^{\lambda}$, where is a parameter to be determined from the data. Using standard computer programme, it can be determined easily.

2.2 Detection of violations of assumptions in regression model, particularly multicollinearity

Regression models are fitted using ordinary least squares (OLS) technique for estimating parameters. The optimality parameters of these parameter estimates are described in an ideal setting which are not often realized in practice. The use and interpretation of a multiple regression model depends implicitly on the assumption that the explanatory variables are not strongly interrelated. In most regression applications the explanatory variables are not orthogonal. Usually the lack of orthogonality is not serious enough to affect the analysis. However, in some situations the explanatory variables are so strongly interrelated that the regression results are ambiguous. Typically, it is impossible to estimate the unique effects of individual variables in the regression equation. The estimated values of the coefficients are very sensitive to slight changes in the data and to the addition or deletion of variables in the equation. The regression coefficients have large sampling errors which affect both inference and forecasting that is based on the regression model. The condition of severe non-orthogonality is also referred to as the problem of multicollinearity.

It has been observed that regressions based on different subsets of data produce very different results, raising questions of model stability. Frequently, we do not have good data in the sense that errors are non–normal or the variance is non–homogeneous. When there are near linear dependencies among regressors, then the problem of multicollinearity is said to exist. The variable pool may not contain the right variables in the proper functional forms and we may have included variables with a high degree of multicollinearity, which may cause problems in estimation, prediction and interpretation. Strong multicollinearity between independent variables results in large variances and co-variances for the least squares estimators of the regression coefficients. Multicollinearity also tends to produce least squares estimates of regression coefficients that are too large in absolute value.

Detection of Multicollinearity can be done by certain ways. Let $R = \left(r_{ij}\right)$ and $R^{-1} = \left(r^{ij}\right)$ denote simple correlation matrix and its inverse. Let $\lambda_i, i = 1,2,...,p \ \left(\lambda_p \le \lambda_{p-1} \le ....\lambda_1\right)$ denote the eigen values of *R*. The following are common indicators of relationships among independent variables.

1. Simple pair-wise correlations
   $\left|r_{ij}\right| = 1$
2. The squared multiple correlation coefficients
   $R_i^2 = 1 - \frac{1}{r^{ii}} > 0.9$, where R_i^2 denote the squared multiple correlation coefficients for the regression of x_I on the remaining *x* variables.
3. The variance inflation factors, $VIF_i = r^{ii} > 10$ and

4. eigen values, $\lambda_i = 0$.

The first of these indicators, the simple correlation coefficients between pairs of independent variables r_{ij}, may detect a simple relationship between x_i and x_j. Thus $|r_{ij}| = 1$ implies that the ith and jth variables are nearly proportional.

The second set of indicators, R_i^2, the squared multiple correlation coefficient for the regression of x_i on the remaining x variables indicates the degree to which x_i is explained by a linear combination of all of the other input variables.

The third set of indicators, the diagonal elements of the inverse matrix, which have been labelled as the Variance Inflation Factors, VIF_i. The term arises by noting that with standardized data (mean zero and unit sum of squares), the variance of the least squares estimate of the ith coefficient is proportional to r^{ii}, $VIF_i > 10$ is probably based on the simple relation between R_i and VIF_i. That is $VIF_i > 10$ corresponds to $R_i^2 > 0.9$.

The remedial measures for presence of multicollinearity are given subsequently:

i) Collection of additional data: Collecting additional data has been suggested as one of the methods of combating multicollinearity. The additional data should be collected in a manner designed to break up the multicollinearity in the existing data.

ii) Model respecification: Multicollinearity is often caused by the choice of model, such as when two highly correlated regressors are used in the regression equation. In these situations, some respecification of the regression equation may lessen the impact of multicollinearity. One approach to respecification is to redefine the regressors. For example, if x_1, x_2 and x_3 are nearly linearly dependent it may be possible to find some function such as $x = (x_1+x_2)/x_3$ or $x = x_1x_2x_3$ that preserves the information content in the original regressors but reduces the multicollinearity.

iii) Ridge Regression: When method of least squares is used, parameter estimates are unbiased. A number of procedures have been developed for obtaining biased estimators of regression coefficients to tackle the problem of multicollinearity. One of these procedures is ridge regression. The ridge estimators are found by solving a slightly modified version of the normal equations. Each of the diagonal elements of $\mathbf{X'X}$ matrix are added a small quantity.

In order to pinpoint which variables contribute for the greater effect of multicollinearity, 'Belsley's procedure' is also employed. Estimation methods such as ridge regression and principal components regression in place of ordinary least squares regression are specifically resorted to combat the problems induced by multicollinearity. However, these procedures yield biased estimators of regression coefficients.

2.3 Variable selection

Variable selection is the process of determining the appropriate subset of that should be used in the model given a pool of candidate regressors that are the possible influential factors. For variable selection, either resort to subset regression models (all possible regressions) or use one of the three stepwise regression methods. viz. stepwise selection, forward selection, backward selection.

2.4 Multiple linear regression when some regressors are qualitative

In case the regressor variables are qualitative, dummy or indicator variables are employed. It there are 's' levels of a qualitative regressor variable, then (s-1) dummy variables need to be used with one of the levels to be mentioned as base or reference category. For e.g., if a qualitative regressor variable has four levels, the following dummy variables D_1, D_2 and D_3 are used taking values for the corresponding levels as follows (here, Level 1 has been taken as base):

	D_1	D_2	D_3
Level1	0	0	0
Level2	1	0	0
Level3	0	1	0
Level4	0	0	1

3. Model selection criteria

The criteria for evaluating subset regression models and hence the adequacy of a regression model are the coefficient of multiple determination R^2, adjusted R^2, Residual Mean Square, Mallow's C_p statistic and the PRediction Error Sum of Squares (PRESS). While comparing models (read model selection), these criteria can be used.

The coefficient of multiple determination R^2 is discussed in detail subsequently. Denote by SSTO $= \sum_{i=1}^{n}(Y_i - \bar{Y})^2$, total sum of squares which measures the variation in the observation Y_i, or the uncertainty in predicting *Y*, when no account of the predictor variable *X* is taken. Thus SSTO is a measure of uncertainty in predicting *Y* when *X* is not considered. Similarly, SSE measures the variation in the *Y*i when a regression model utilizing the predictor variable *X* is employed. A natural measure of the effect of *X* in reducing the variation in *Y*, i.e., in reducing the uncertaintity in predicting Y, is to express the reduction in variation (SSTO-SSE=SSR) as a proportion of the total variation: $R^2 = \frac{SSR}{SSTO} = 1 - \frac{SSE}{SSTO}$. The measure R^2 is called coefficient of determination, $0 \le R^2 \le 1$. In practice R^2 is not likely to be 0 or 1 but somewhere between these limits. The closer it is to 1, the greater is said to be the degree of linear association between *X* and *Y*.

4. Regularized Linear Modeling

When training a model (be it the usual multiple linear regression model or the machine learning model), the model can easily become either over-fitted or under-fitted. To circumvent this, regularization is employed to properly fit the model on the data set under consideration aiming to build an optimal model. To fit a best linear regression model, usually the method of least squares (often called as Ordinary Least Squares or OLS) is used. The process of arriving at a 'line of best

fit' is to understand the relationship between set of independent variables and the main or study or dependent variable. The fitted model is a best one, when the underlying pattern in the data is well represented covering most of the points on it by means of minimizing the 'error sum of squares', in case of OLS.

If one allows the model to look at the data too many times (repetition of same information again and again, if more of the same or similar kind of information is contained in the dataset), it will learn well on the given dataset and fit it very well but it may not be able to make predictions on other data sets in the same manner. That is, if new data is provided, the fitted model may not be able to understand the pattern in the new data, and in turn the model may not predict very well. Such a phenomenon is called 'over-fitting'. Conversely, in the scenario where the model has not been allowed to look at the data adequately, the model will not be able to find the underlying pattern in data set used for fitting. It may not fit the data set properly and may not certainly be able to work on any new data either. A scenario where a model can neither learn the relationship between variables in the test data nor predict or classify a new data point is called under-fitting.

Regularization refers to techniques used to calibrate statistical or machine learning models to minimize the adjusted loss (error) function and avoid over-fitting or under-fitting (Choudhary, 2022). According to Pandian (2022), regularization technique is an important step to improve the model prediction and reduce errors. This technique uses certain Shrinkage methods such as ridge regression to accomplish the task. For this, a penalty term is added to the Error Function (which needs to be minimised) to control the complex model to avoid overfitting by reducing the variance.

4.1 Ridge regularization

Also known as ridge regression, ridge regularization it adjusts models with overfitting or underfitting by adding a penalty equivalent to the sum of the squares of the magnitudes of the coefficients. This means that the mathematical function representing the linear regression model is minimized and the coefficients are calculated. Ridge Regression performs regularization by reducing the coefficients present (also known as shrinkage estimation).

In this regularization, the sum of squared errors added with sum of the squared coefficients (β) will be minimized. Usually, the coefficients (β_i's) with a large magnitude will generate the peaks in graph and also troughs which are deeper, hence to get a smooth plot covering most of the data points, a Penalty Factor named lambda (λ) is used. This is called "L2 regularization", since its adding a penalty equivalent to the Square-of-the magnitude of coefficients.

4.2 LASSO regularization

LASSO (Least Absolute Shrinkage and Selection Operator) regularization modifies overfitted or under-fitted models by adding a penalty equivalent to the sum of the absolute values of the coefficients. According to Kumar (2023), the primary goal of LASSO regression is to find a balance between model simplicity and accuracy. It achieves this by adding a penalty term to the traditional linear regression model, which encourages sparse solutions where some coefficients are forced to be exactly zero. This feature makes LASSO particularly useful for feature selection, as it can automatically identify and discard irrelevant or redundant variables. Lasso regression also performs coefficient minimization, but instead of squaring the magnitudes of the coefficients, it takes the

actual values of the coefficients. This means that the sum of the coefficients can also be 0 because there are negative coefficients. This regularization is very similar to ridge regularization, with little difference in Penalty Factor that coefficients considered is absolute value (magnitude) instead of squared terms. This is called "L1 regularization", because of adding the absolute value as penalty equivalent to the magnitude of coefficients. The following screenshot is taken from Pandian (2022), noting that in the Equation (1), instead of $\beta_n x_n$ at the end, it is $\beta_p x_p$, as there are p number of regressors (while n usually represents number of observations):

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots\ldots\ldots\ldots+\beta_n x_n \quad ------1$$

$$y_i=\beta_0+\Sigma\,\beta_i x_i \quad ------2$$

$$\Sigma\, y_i-\beta_0-\Sigma\,\beta_i x_i$$

$$\text{Cost/Loss function: } \Sigma\{y_i-\beta_0-\Sigma\,\beta_i x_{ij}\}^2 \quad ------3$$

$$\text{Regularized term: } \lambda\Sigma\,\beta_i^2 \quad ------4$$

$$\text{Ridge Regression = Loss function + Regularized term} \quad ------5$$

Put 3 and 4 in 5

$$\textbf{Ridge Regression} = \boldsymbol{\Sigma\{y_i-\beta_0-\Sigma\,\beta_i x_{ij}\}^2+\lambda\Sigma\,\beta_i^2}$$

$$\textbf{Lasso Regression} = \boldsymbol{\Sigma\{y_i-\beta_0-\Sigma\,\beta_i x_{ij}\}^2+\lambda\Sigma\,|\beta_i|}$$

- x ==> independent variables
- y ==> target variables
- β ==> coefficients
- λ ==> penalty-factor

4.3 Elastic Net

Elastic Net combines both Ridge and LASSO regularizations.

References

Chatterjee, S. and Hadi, A. S. (2006). *Regression analysis by example*, 4th Edition, John Wiley and Sons, New Jersey.

Choudhary, A.S. (2022). Regularization in Machine Learning, https://www.analyticsvidhya.com/blog/2022/08/regularization-in-machine-learning/, available for access on August 31, 2023.

Draper, N. R. and Smith, H. (1998). *Applied Regression Analysis, 3rd edition.* New York: Wiley.

Kumar, D. (2023). A complete understanding of LASSO regression, https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/#:~:text=Lasso%20regression%20is%20a%20regularization,i.e.%20models%20with%20fewer%20parameters), available for access on August 31, 2023.

Montgomery, D.C., Peck, E.A. and Vining, G.G. (2001). *Introduction to Linear Regression Analysis*. 3rd edition, John Wiley and Sons, New Delhi.

Pandian, S. (2022). Study of regularization techniques of linear models and its roles, https://www.analyticsvidhya.com/blog/2021/11/study-of-regularization-techniques-of-linear-model-and-its-roles/, available for access on August 31, 2023.

An Overview of Multivariate Statistical Analysis Techniques

Rahul Banerjee
ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110 012
rahul.banerjee@icar.gov.in

Introduction:

Multivariate analysis is a statistical technique used in data analysis to understand the relationships among multiple variables simultaneously. It is employed when you have a dataset with two or more variables, and you want to examine how these variables interact with each other or how they collectively influence an outcome or dependent variable. Multivariate analysis encompasses various statistical methods and techniques, including:

- **Multivariate Regression Analysis**: This includes techniques like Multiple Linear Regression and Multivariate Analysis of Variance (MANOVA), which deal with multiple predictor variables and one or more dependent variables.
- **Principal Component Analysis (PCA)**: PCA is used to reduce the dimensionality of data by transforming variables into a smaller set of uncorrelated variables called principal components. It is often used for data reduction and visualization.
- **Factor Analysis**: Factor analysis is used to identify underlying factors or latent variables that explain patterns in observed data. It's often used in fields like psychology to understand the structure of survey responses.
- **Cluster Analysis**: Cluster analysis groups similar data points or observations together into clusters or segments based on their similarities. It's often used for customer segmentation and pattern recognition.
- **Discriminant Analysis**: Discriminant analysis is used to distinguish between two or more groups based on their characteristics. It's commonly used in fields like marketing to identify factors that differentiate customer groups.
- **Canonical Correlation Analysis (CCA)**: CCA explores the relationships between two sets of variables and finds linear combinations of variables in each set that are maximally correlated with each other.
- **Multivariate Analysis of Covariance (MANCOVA)**: MANCOVA extends ANCOVA by allowing for the analysis of multiple dependent variables while controlling for one or more covariates.
- **Structural Equation Modeling (SEM)**: SEM is used to test and estimate complex relationships between observed and latent variables. It's often used in social sciences and psychology.
- **Multidimensional Scaling (MDS)**: MDS is a technique used to visualize the similarity or dissimilarity between data points in a lower-dimensional space, making it easier to interpret relationships.
- **Multivariate Time Series Analysis**: This involves analyzing multiple time series data simultaneously, which is common in fields like finance and economics.

The choice of which multivariate analysis technique to use depends on the nature of your data and research objectives. Multivariate analysis allows researchers and analysts to uncover

patterns, associations, and insights that may not be apparent when considering variables individually, making it a valuable tool in various fields, including social sciences, business, biology, and more.

1. Testing of mean vector - One Sample Case

This is useful for the situations where the data on the different variables are collected and it is required to test whether the sample mean vectors is equal to a specified mean vector. To be specific: Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a random sample of size *n* is drawn from the population with *p*-dimensional mean vector $\boldsymbol{\mu}_0$ and based on this sample we want to test $H_0 : \boldsymbol{\mu} = \boldsymbol{\mu_0}$ against $H_1 : \boldsymbol{\mu} \neq \boldsymbol{\mu_0}$.

If variance covariance matrix Σ is known or the sample is large, χ^2 test is used.

$$\chi^2 = n(\bar{\mathbf{x}} - \boldsymbol{\mu_0})' \Sigma^{-1} (\bar{\mathbf{x}} - \boldsymbol{\mu_0})$$

with *p* degrees of freedom where $\bar{\mathbf{x}} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$ is the sample mean calculated from the sample, *p* is number of variable in the study.

If Σ is not known and sample size is small. Hotelling T^2 is used.

$$T^2 = n(\bar{\mathbf{x}} - \boldsymbol{\mu_0})' \mathbf{s}^{-1} (\bar{\mathbf{x}} - \boldsymbol{\mu_0})$$

where $\bar{\mathbf{x}} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$, $\mathbf{s} = \frac{1}{n-1}\sum_{j=1}^{n} (\mathbf{x_j} - \bar{\mathbf{x}})(\mathbf{x_j} - \bar{\mathbf{x}})'$.

$$\frac{(n-p)}{(n-1)p} T^2 \approx F_{p,\, n-p}.$$

Example 1: {Example 5.2 in Johnson and Wichern, 2002}. Perspiration from 20 healthy females was analyzed. Three components, X_1 = sweat rate, X_2 = sodium content and X_3 = potassium content were measured and the results are presented in table 1.

Table 1: Sweat Data

Individual	X_1 **(sweat rate)**	X_2 **(sodium content)**	X_3 **(potassium content)**
1	3.7	48.5	9.3
2	5.7	65.1	8.0
3	3.8	47.2	10.9
4	3.2	53.2	12.0
5	3.1	55.5	9.7
6	4.6	36.1	7.9
7	2.4	24.8	14.0
8	7.2	33.1	7.6
9	6.7	47.4	8.5
10	5.4	54.1	11.3
11	3.9	36.9	12.7
12	4.5	58.8	12.3
13	3.5	27.8	9.8

14	4.5	40.2	8.4
15	1.5	13.5	10.1
16	8.5	56.4	7.1
17	4.5	71.6	8.2
18	6.5	52.8	10.9
19	4.1	44.1	11.2
20	5.5	40.9	9.4

Test the hypothesis, $H_0 : \boldsymbol{\mu} = \boldsymbol{\mu_0}$ given $\boldsymbol{\mu_0} = [4 \quad 50 \quad 10]$ against $H_1 : \mu \neq \mu_0$. From Table 1, we can calculate

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{j=1}^{n}\mathbf{X}_j = \begin{bmatrix} 4.640 \\ 45.400 \\ 9.965 \end{bmatrix}, \quad \mathbf{s} = \frac{1}{n-1}\sum_{j=1}^{n}(\mathbf{x_j} - \bar{\mathbf{x}})(\mathbf{x_j} - \bar{\mathbf{x}})' = \begin{bmatrix} 2.879368 & 10.01 & -1.80905 \\ 10.01 & 199.7884 & -5.64 \\ -1.80905 & -5.64 & 3.627658 \end{bmatrix}$$

and the observed T^2 value is

$$= n(\bar{\mathbf{x}} - \mu_{\mathbf{0}})'\mathbf{s}^{-1}(\bar{\mathbf{x}} - \mu_{\mathbf{0}})$$

$$= 20\begin{bmatrix} 4.640-4 & 45.400-50 & 9.965-10 \end{bmatrix}\begin{bmatrix} 2.879368 & 10.01 & -1.80905 \\ 10.01 & 199.7884 & -5.64 \\ -1.80905 & -5.64 & 3.627658 \end{bmatrix}^{-1}\begin{bmatrix} 4.640-4 \\ 45.400-50 \\ 9.965-10 \end{bmatrix}$$

$$= 20\begin{bmatrix} 0.640 & -4.600 & -0.035 \end{bmatrix}\begin{bmatrix} 0.467705 \\ -0.04199 \\ 0.158308 \end{bmatrix} = 9.738774$$

Comparing the observed T^2 =9.738774 with the critical value $\frac{(n-1)p}{(n-p)}F_{p,n-p(\alpha)} = 3.353 \times 3.20 = 10.73$ we see that T^2 = *9.74 < 10.73*, and consequently we accept H_0.

2. Testing of mean vectors - Two Sample Case

Consider that we have two independent random samples of sizes n_1 and n_2 with mean vectors $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ and sample dispersion matrices $\mathbf{s}_1$ and $\mathbf{s}_2$ respectively and want to test the hypothesis

$$H_0 : \boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 \quad \text{against} \quad H_1 : \boldsymbol{\mu}_1 \neq \boldsymbol{\mu}_2$$

$\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ are mean vectors of populations from which samples are drawn. If population dispersion matrices are unknown but same, we use

$$T^2 = \frac{n_1 n_2}{n_1 + n_2}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)\mathbf{s}_{pooled}^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$$

where $\mathbf{s}_{pooled} = \frac{(n_1 - 1)\mathbf{s}_1 + (n_2 - 1)\mathbf{s}_2}{n_1 + n_2 - 2}$.

T^2 is distributed as $\frac{(n_1 + n_2 - 2)p}{(n_1 + n_2 - p - 1)}F_{p,n_1+n_2-p-1}$

Example 2: {Example 6.4 in Johnson and Wichern, 2002}. Samples of sizes $n_1 = 45$ and $n_2 = 55$ were taken of homeowners with and without air conditioning respectively. Two measurements of electrical usage (is kilowatt-hours) were considered. The first is a measure of total on-peak consumption ($\mathbf{x}_1$) during July and the second is a measure of total off-peak consumption during July. Test weather there is a difference in electrical consumption between those with air conditioning and those without.

The summary statistics given are

$$\mathbf{x}_1 = \begin{bmatrix} 204.4 \\ 556.6 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 130.0 \\ 355.0 \end{bmatrix}$$

$$\mathbf{s}_1 = \begin{bmatrix} 13825.3 & 23823.4 \\ 23823.4 & 73107.4 \end{bmatrix}, \quad \mathbf{s}_2 = \begin{bmatrix} 8632.0 & 19616.7 \\ 19616.7 & 55964.5 \end{bmatrix}$$

$n_1 = 45, n_2 = 55$

Here the null hypothesis is $H_0 : \boldsymbol{\mu_1} = \boldsymbol{\mu_2}$ and alternate hypothesis is $H_1 : \boldsymbol{\mu_1} \neq \boldsymbol{\mu_2}$. To test the difference, first we calculate

$$\mathbf{s}_{pooled} = \frac{(n_1 - 1)\mathbf{s}_1 + (n_2 - 1)\mathbf{s}_2}{n_1 + n_2 - 2}$$

$$= \begin{bmatrix} 10963.7 & 21505.5 \\ 21505.5 & 63661.3 \end{bmatrix}.$$

Now $$T^2 = \frac{n_1 n_2}{n_1 + n_2}(\mathbf{x}_1 - \mathbf{x}_2)' \mathbf{s}_{pooled}^{-1}(\mathbf{x}_1 - \mathbf{x}_2)$$

$$= 16.22066$$

Comparing the observed T^2 with the critical value

$$\frac{(n_1 + n_2 - 2)p}{(n_1 + n_2 - p - 1)} F_{p, n_1 + n_2 - p - 1(\alpha)} = \frac{98(2)}{97} F_{2.97(0.05)} = 6.26.$$

We see that the observed $T^2 = 16.22066 > 6.26$, we reject the null hypothesis and conclude that there is a difference in electrical consumption between those with air conditioning and those without.

Note:

(i) For this testing, Mahalnobis D^2 can also be used which is a linear function of T^2

$$D^2 = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)' \mathbf{s}_{pooled}^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$$

$$= \frac{n_1 n_2}{n_1 + n_2} T^2$$

(ii) If $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$, the above test cannot be used. For large sample size or dispersion matrices known, χ^2 test can be used. However, test for small sample sizes and dispersion matrices not known to be equal is beyond the scope of discussion. Readers may go through the references given at the end.

Steps to carry out the Analysis: **Testing Mean Vector (s) (Using MS-EXCEL)**

We to use the inbuilt Functions of MS-EXCEL like *Average*: Mean; *VAR*: Variance and *COVAR*n/(n-1)*: Covariance. Correlation can be obtained using the function *CORREL*.

Matrix Inverse

Mark the area for the resultant matrix → Formula bar → =minverse (mark range of original matrix) → press control + shift + enter

Matrix multiplication

Mark the area for the resultant matrix → Formula bar → =mmult (mark range of first matrix, mark range of second matrix) → press control + shift + enter

Using the matrix multiplication and matrix inversion one can easily calculate Hotelling's T^2.

3. Multivariate Analysis of Variance (MANOVA)

One way Classified Data

Consider that the random samples from each of *g* (say) populations using are arranged as

Population 1: $\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{1n_1}$

Population 2: $\mathbf{x}_{21}, \mathbf{x}_{22}, \ldots, \mathbf{x}_{2n_2}$

:

:

:

Population g: $\mathbf{x}_{g1}, \mathbf{x}_{g2}, \ldots, \mathbf{x}_{gn_g}$

Multivariate analysis of variance is used first to investigate whether the populations mean vectors are the same and, if not, which mean components differ significantly. MANOVA is carried out under the following two assumptions: 1. Dispersion matrices of various populations are same. 2. Each population is multivariate normal. One-way Classified MANOVA Table for testing the equality of g-population mean Vectors is given below:

Source of variation	Degrees of freedom	SSP matrix
Population or treatment	g-1	$\mathbf{T} = \sum_{i=1}^{g} n_i (\bar{\mathbf{x}}_i - \bar{\mathbf{x}})(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})'$
Residual (error)	$\sum_{i=1}^{g} n_i - g$	$\mathbf{R} = \sum_{i=1}^{g} \sum_{j=1}^{n_i} (\mathbf{x}_{ij} - \bar{\mathbf{x}}_i)(\mathbf{x}_{ij} - \bar{\mathbf{x}}_i)'$
Total	$\sum_{i=1}^{g} n_i - 1$	$\mathbf{T} + \mathbf{R} = \sum_{i=1}^{g} \sum_{j=1}^{n_i} (\mathbf{x}_{ij} - \bar{\mathbf{x}})(\mathbf{x}_{ij} - \bar{\mathbf{x}})'$

We reject the null hypothesis of equal mean vectors if the ratio of generalized variance (*Wilk's lambda* statistic) $\Lambda^* = \dfrac{|\mathbf{R}|}{|\mathbf{T} + \mathbf{R}|}$ is too small. The distribution of Λ^* in different cases are as below.

$p = 1 \quad g \geq 2 \qquad \left(\dfrac{\sum n_i - g}{g - 1}\right)\left(\dfrac{1 - \Lambda^*}{\Lambda^*}\right) \sim F_{g-1,(\sum n_i - g)}(\alpha)$

$p = 2 \quad g \geq 2 \qquad \left(\frac{\sum n_i - g - 1}{g - 1}\right)\left(\frac{1 - \sqrt{\Lambda^*}}{\sqrt{\Lambda^*}}\right) \sim F_{2(g-1),2(\sum n_i - g - 1)}(\alpha)$

$p \geq 1 \quad g = 2 \qquad \left(\frac{\sum n_i - p - 1}{p}\right)\left(\frac{1 - \Lambda^*}{\Lambda^*}\right) \sim F_{p,(\sum n_i - p - 1)}(\alpha)$

$p \geq 1 \quad g = 3 \qquad \left(\frac{\sum n_i - p - 2}{p}\right)\left(\frac{1 - \sqrt{\Lambda^*}}{\sqrt{\Lambda^*}}\right) \sim F_{2p,2(\sum n_i - p - 2)}(\alpha)$

and for other cases $-\left(n - 1 - \frac{(p+g)}{2}\right) \text{ In } \Lambda^* \sim \chi^2_{p(g-1)}(\alpha)$ (approximate).

Example 3: {Example 6.8 in Johnson and Wichern, 2002}. Consider the following independent samples:

	R1	R2	R3	Total
Population 1	$\begin{bmatrix}9\\3\end{bmatrix}$	$\begin{bmatrix}6\\2\end{bmatrix}$	$\begin{bmatrix}9\\7\end{bmatrix}$	$\begin{bmatrix}24\\12\end{bmatrix}$
Population 2	$\begin{bmatrix}0\\4\end{bmatrix}$	$\begin{bmatrix}2\\0\end{bmatrix}$		$\begin{bmatrix}2\\4\end{bmatrix}$
Population 3	$\begin{bmatrix}3\\8\end{bmatrix}$	$\begin{bmatrix}1\\9\end{bmatrix}$	$\begin{bmatrix}2\\7\end{bmatrix}$	$\begin{bmatrix}6\\24\end{bmatrix}$
Grand Total				$\begin{bmatrix}32\\40\end{bmatrix}$

Due to variable 1

Sum of squares (Population) = $\frac{24^2}{3} + \frac{2^2}{2} + \frac{6^2}{3} - \frac{38^2}{8} = 78$

Sum of squares (Total) $= 9^2 + 6^2 + \ldots + 2^2 - \frac{38^2}{8} = 88$

Sum of squares (Residual) $= 88 - 78 = 10$ (by subtraction)

Due to variable 2

Sum of squares (Population) = $\frac{12^2}{3} + \frac{4^2}{2} + \frac{24^2}{3} - \frac{40^2}{8} = 48$

Sum of squares (Total) $= 3^2 + 2^2 + \ldots + 7^2 - \frac{40^2}{8} = 72$

Sum of squares (Residual) $= 72 - 48 = 24$ (by subtraction)

Due to variable 1 and 2

Sum of cross products (Population) = $\frac{24\times12}{3}+\frac{2\times4}{2}+\frac{6\times24}{3}-\frac{32\times40}{8}=-12$

Sum of cross products (Total) = $9\times3+6\times2+...+2\times7-\frac{32\times40}{8}=-11$

Sum of cross products (Residual) = $-11-(-12)=1$

MANOVA

Source of Variation	**Degrees of freedom**	**SSP matrix**
Population	$3-1=2$	$\begin{bmatrix} 78 & -12 \\ -12 & 48 \end{bmatrix}$
Residual (error)	$3+2+3-3=5$	$\begin{bmatrix} 10 & 1 \\ 1 & 24 \end{bmatrix}$
Total	$3+2+3-1=7$	$\begin{bmatrix} 88 & -11 \\ -11 & 72 \end{bmatrix}$

To test the hypothesis $H_0:\mu_1=\mu_2=\mu_3$. We use *Wilk's lambda* statistic

$$\Lambda^*=\frac{|\mathbf{R}|}{|\mathbf{T}+\mathbf{R}|}=\frac{\left\|\begin{bmatrix} 10 & 1 \\ 1 & 24 \end{bmatrix}\right\|}{\left\|\begin{bmatrix} 88 & -11 \\ -11 & 72 \end{bmatrix}\right\|}=\frac{10(24)-(1)^2}{88(72)-(-11)^2}=\frac{239}{6215}=0.0385$$

Since $p=2$ (variables) and $g=3$ (populations), we use the following $\left(\frac{\sum n_i-p-2}{p}\right)\left(\frac{1-\sqrt{\Lambda^*}}{\sqrt{\Lambda^*}}\right)=\left(\frac{8-3-1}{3-1}\right)\left(\frac{1-\sqrt{0.0385}}{\sqrt{0.0385}}\right)=8.19$ with a percentage point of an *F*-distribution having $n_1=4$ & $n_2=8$ d.f. Since $8.19>F_{4,8}(0.01)=7.01$, we reject the null hypothesis at 1% level of significance and conclude that there exists treatment differences. The pairwise comparisons can be done using the contrast analysis.

Remark: One complication of multivariate analysis that does not arise in the univariate case is due to the ranks of the matrices. The rank of **R** should not be smaller than *p* or in other words error degrees of freedom *s* should be greater than or equal to *p* ($s \geq p$).

For performing MANOVA using SAS, the following procedures/statements may be used.

PROC ANOVA and PROC GLM can be used to perform analysis of variance even for more than one dependent variables. PROC ANOVA performs the analysis of variance for balanced data whereas PROC GLM can analyze both balanced and unbalanced data. As ANOVA takes into account the special features of a balanced data, it is faster and uses less storage than PROC GLM for balanced data. The basic syntax of the ANOVA procedure is as given

```
PROC ANOVA <options>;
CLASS Variables;
MODEL dependents = independent variables (or effects)/ options;
MEANS effects / options,
```

ABSORB Variables;
FREQ Variable;
TEST H=effects E= effect M = equations/options;
REPEATED factor - name levels / options;
BY variables;
RUN;

The PROC ANOVA, CLASS and MODEL statements are must. The other statements are optional. The class statement defines the variables for classification (numeric or character variables - maximum characters = 16).

PROC GLM for analysis of variance is similar to PROC ANOVA. The statements listed for PROC ANOVA are also used for PROC GLM. The following more statements can be used with PROC GLM;

CONTRAST 'label' effect name < .... effect coefficients > / < options>;
ESTIMATE 'label' effect name < ... effect coefficients / <options>;
ID variables;
LSMEANS effects </options>;
OUTPUT <OUT = SAS-data-set > keyword = names< ... keyword=names;
RANDOM effects/ < options > ;
WEIGHT;

However, if the MODEL statement includes more than one dependent variable, additional multivariate statistics can be requested with the MANOVA statement.

When a MANOVA statement appears before the first RUN statement, GLM or ANOVA enters a multivariate mode with respect to the handling of missing values; observations with missing independent or dependent variables are excluded from the analysis. If you want to use this mode of handling missing values and do not need any multivariate analysis, specify the MANOVA option in the PROC GLM statement.

If both the CONTRAST and MANOVA statements are to be used, the MANOVA statement must appear after the CONTRAST statement. The basic syntax of MANOVA statement is

MANOVA;
MANOVA < H=effects | INTERCEPT |_ALL_ ><E=effect></options>;
MANOVA < H=effects | INTERCEPT |_ALL_><E=effect>
<M=equation,...,equation |(row-or-matrix,...,row-or-matrix)>
<MNAMES=names><PREFIX=name></options>;

The terms given in the MANOVA statement are specified as follows:

H=effects | INTERCEPT |_ALL_ : specifies effects in the preceding model to use as hypothesis matrices. For each H matrix (the SSCP matrix associated with that effects), the H=specification prints the characteristic roots and vectors of $\mathbf{E}^{-1}\mathbf{H}$ (where **E** is the matrix associated with the error effects), Hotelling-Lawley trace, Pillai's trace, Wilks' criterion, and Roy's maximum root criterion with approximate F statistic. Use the keyword INTERCEPT to print tests for the intercept. To print tests for all effects listed in the MODEL statement, use the keyword _ALL_ in place of a list of effects.

E=effect : specifies the error effect. If we omit the E=specification, GLM uses the error SSCP (residual) matrix from the analysis.

<M=equation, ..., equation |(row-or-matrix,...,row-or-matrix)> : specifies a transformation matrix for the dependent variables listed in the MODEL statement. The equations in the M=specification are of the form

$$C_1{}^{*}\text{dependent-variable} \pm C_2{}^{*}\text{dependent-variable} \pm C_n{}^{*}\text{dependent-variable}$$

where the C_i values are coefficients for the various dependent-variables. If the value of a given C_i is 1, it may be omitted; in other words, 1*Y is the same as Y. Equations should involve two or more dependent variables. Alternatively, we can input the transformation matrix directly by entering the elements of the matrix with commas separating the rows, and parentheses surrounding the matrix. When this alternate form of input is used, the number of elements in each row must equal the number of dependent variables. Although these combinations actually represent the columns of the M matrix, they are printed by rows.

When we include an M=specification, the analysis requested in the MANOVA statement is carried out for the variables defined by the equations in the specification, not the original dependent variables. If M=is omitted, the analysis is performed for the original dependent variables in the MODEL statement.

If an M=specification is included without either the MNAMES= or PREFIX=option, the variables are labelled by default as MVAR1, MVAR2, and so on.

MNAMES= names: provides names for the variables defined by the equations in the M=specification. Names in the list correspond to the M=equations or the rows of the M matrix (as it is entered).

PREFIX = name : is an alternative means of identifying the transformed variables defined by the M=specification. For example, if you specify PREFIX = DIFF, the transformed variables are labelled DIFF1, DIFF2, and so on.

The following options can be used in the MANOVA statement

CANONICAL : Prints a canonical analysis of the **H** and **E** matrices (transformed by the M matrix, if specified) instead of the default printout of characteristic roots and vectors.

ETYPE=n : specifies the type(1,2,3, or 4) of the **E** matrix. By default, the procedure uses an ETYPE=value corresponding to the highest type (largest n) used in the analysis.

HTYPE =n : specifies the type (1,2,3, or 4) of the H matrix.

ORTH : requests that the transformation matrix in the M=specification of the MANOVA statement be orthonormalized by rows before the analysis.

PRINTE : prints the **E** matrix. If the **E** matrix is the error SSCP (residual) matrix from the analysis, the partial correlations of the dependent variables given the independent variables are also printed. For example, the statement

```
manova / printe;
```

prints the error SSCP matrix and the partial correlation matrix computed from the error SSCP matrix.

PRINTH : prints the **H** matrix (the SSCP matrix) associated with each effect specified by the **H**=specification.

SUMMARY: produces analysis-of-variance tables for each dependent variable. When no **M** matrix is specified, a table is printed for each original dependent variable from the MODEL statement; with an **M** matrix other than the identity, a table is printed for each transformed variable defined by the **M** matrix.

Various ways of using a MANOVA statement are given as follows:

```
proc glm;
    class a b;
    model y1-y5=a b(a);
    manova h=a e=b(a) / printh printe htype=1 etype=1;
    manova h=b(a) / printe;
    manova h=a e=b(a) m=y1-y2, y2-y3, y3-y4, y4-y5  prefix=diff;
    manova h=a e=b(a) m=(1  -1  0  0  0,
                         0   1 -1  0  0,
                         0   0  1 -1  0,
                         0   0  0  1 -1) prefix=diff;
run;
```

Since this MODEL statement requests no options for type of sums of squares, GLM uses Type I and Type III. The first MANOVA statement specifies A as the hypothesis effect and B(A) as the error effect. As a result of PRINTH, the procedure prints the **H** matrix associated with the A effect; and, as a result of PRINTE, the procedure prints the **E** matrix associated with the B(A) effect. HTYPE=1 specifies a Type I **H** matrix, and ETYPE =1 specifies a Type I **E** matrix.

The second MANOVA statement specifies B(A) as the hypothesis effect. Since no error effect is specified, GLM uses the error SSCP matrix from the analysis as the **E** matrix. The PRINTE option prints this **E** matrix. Since the **E** matrix is the error SSCP matrix from the analysis, the partial correlation matrix computed from this matrix is also printed.

The third MANOVA statement requests the same analysis as the first MANOVA statement, but the analysis is carried out for variables transformed to be successive differences between the original dependent variables. PREFIX=DIFF labels the transformed variables as DIFF1, DIFF2, DIFF3, and DIFF4.

Finally, the fourth MANOVA statement has the identical effect as the third, but it uses an alternative form of the M=specification. Instead of specifying a set of equations, the fourth MANOVA statement specifies rows of a matrix of coefficients for the five dependent variables.

SPSS: To obtain MANOVA, from the menus choose Analyze → General Linear Models… → Multivariate…→ Select at least two dependent variables→ Optionally, one can specify Fixed Factor(s), Covariate(s), and WLS Weight.

4. Principal Component Analysis

Principal component analysis (PCA) is one among techniques for taking high dimensional data, and using the dependencies between the variables to represent it in a more tractable, lower-dimensional form, without losing too much information. It was invented by Pearson (1901) and Hotelling (1933) and first applied in ecology by Goodall (1954) under the name "Factor Analysis". During 1970 PCA was considered as the ordination method of choice for community data. Further, simulation studies made by Swan (1970), Austin and Noy-Meir (1971) demonstrated the horseshoe effect and showed that the linear assumption of PCA was not compatible with the non-linear structure of community data. Recently, it has stimulated the search for more appropriate ordination method and is most widely used as well as well-known of the "standard" multivariate methods. The purpose of principal component analysis is to derive a small number of linear combinations (principal components) of a set of variables that retain as much information in the original variables as possible. Often a small number of principal components can be used in place of the original variables for plotting, regression,

clustering and so on. Principal component analysis can also be viewed as a technique to remove multicollinearity in the data. PCA is a way of identifying patterns in the data; data is expressed in such a way that the similarities and differences are highlighted. Once the patterns are found in the data, it can be compressed (reduce the number of dimensions) without losing much information.

PCA is one of the simplest and most robust ways of dimensionality reduction. It is also one of the oldest methods, and has been rediscovered many times in many fields, so it is also known as the Karhunen-Loève transformation, the Hotelling transformation, the method of empirical orthogonal functions, and singular value decomposition. PCA is concerned with explaining the variance covariance structure of a set of variables through a few linear combinations of these variables. Mathematically it is orthogonal linear transformation of data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component (PC)), the second greatest variance on the second coordinate, and so on. PCA is an intermediate step for further treatment of data that includes regression analysis, indexing, assigning weights, etc.

In this technique, we transform the original set of variables to a new set of uncorrelated random variables. These new variables are linear combinations of the originals variables and are derived in decreasing order of importance so that the first principal component accounts for as much as possible of the variation in the original data.

PCA basically tries to explain the total data variability with the help of a fewer number of linear combinations of the original data called the Principal Components. The broad objective of PCA is the reduction in the data dimensions.

"*The information content of the new variables is as much as the information content of the original variables*."

Let $x_1, x_2, x_3, ..., x_p$ are variables under study, then first principal component may be defined as

$$z_1 = a_{11}x_1 + a_{12}x_2 + ... + a_{1p}x_p$$

such that variance of z_1 is as large as possible subject to the condition that

$$a_{11}^2 + a_{12}^2 + ... + a_{1p}^2 = 1$$

This constraint is introduced because if this is not done, then $Var(z_1)$ can be increased simply by multiplying any $a_{1j}'s$ by a constant factor. The second principal component is defined as

$$z_2 = a_{21}x_1 + a_{22}x_2 + ... + a_{2p}x_p$$

such that $Var(z_2)$ is as large as possible next to $Var(z_1)$ subject to the constraint that

$a_{21}^2 + a_{22}^2 + ... + a_{2p}^2 = 1$ and $Cov(z_1, z_2) = 0$ and so on.

It is quite likely that first few principal components account for most of the variability in the original data. If so, these few principal components can then replace the initial *p* variables in subsequent analysis, thus reducing the effective dimensionality of the problem. An analysis of principal components often reveals relationships that were not previously suspected and thereby allows interpretation that would not ordinarily result. However, Principal Components Analysis is more of a mean to an end rather than end in itself because this frequently serves as intermediate steps in much larger investigations by reducing the dimensionality of the problem

and providing easier interpretation. It is a mathematical technique, which does not require user to specify the statistical model or assumption about distribution of original variates. It may also be mentioned that principal components are artificial variables and often it is not possible to assign physical meaning to them. Further, since Principal Components Analysis transforms original set of variables to new set of uncorrelated variables. It is worth stressing that if the original variables are uncorrelated, then there is no point in carrying out the Principal Components Analysis. It is important to note here that the principal components depend on the scale of measurement. Conventional way of getting rid of this problem is to use the standardized variables with unit variances.

Example 4: Let us consider the following data on average minimum temperature (x_1), average relative humidity at 8 hrs. (x_2), average relative humidity at 14 hrs. (x_3) and total rainfall in cm. (x_4) pertaining to Raipur district from 1970 to 1986 for kharif season from 21st May to 7th Oct.

	x_1	x_2	x_3	x_4
	25.0	86	66	186.49
	24.9	84	66	124.34
	25.4	77	55	98.79
	24.4	82	62	118.88
	22.9	79	53	71.88
	7.7	86	60	111.96
	25.1	82	58	99.74
	24.9	83	63	115.20
	24.9	82	63	100.16
	24.9	78	56	62.38
	24.3	85	67	154.40
	24.6	79	61	112.71
	24.3	81	58	79.63
	24.6	81	61	125.59
	24.1	85	64	99.87
	24.5	84	63	143.56
	24.0	81	61	114.97
Mean	23.56	82.06	61.00	112.97
S.D.	4.13	2.75	3.97	30.06

with the variance co-variance matrix.

$$\sum = \begin{bmatrix} 17.02 & -4.12 & 1.54 & 5.14 \\ & 7.56 & 8.50 & 54.82 \\ & & 15.75 & 92.95 \\ & & & 903.87 \end{bmatrix}$$

Find the eigenvalues and eigenvectors of the above matrix. Arrange the eigenvalues in decreasing order. Let the eigenvalues in decreasing order and corresponding eigenvectors are

$\lambda_1 = 916.902 \quad \mathbf{a_1} = (0.006, 0.061, 0.103, 0.993)$

$\lambda_2 = 18.375 \quad \mathbf{a_2} = (0.955, -0.296, 0.011, 0.012)$

$\lambda_3 = 7.87 \quad \mathbf{a_3} = (0.141, 0.485, 0.855, -0.119)$

$\lambda_4 = 1.056 \qquad \mathbf{a_4} = (0.260,\ 0.820,\ -0.509,\ 0.001)$

The principal components for this data will be

$$z_1 = 0.006x_1 + 0.061x_2 + 0.103x_3 + 0.993x_4$$
$$z_2 = 0.955x_1 - 0.296x_2 + 0.011x_3 + 0.012x_4$$
$$z_3 = 0.141x_1 + 0.485x_2 + 0.855x_3 - 0.119x_4$$
$$z_4 = 0.26x_1 + 0.82x_2 - 0.509x_3 + 0.001x_4$$

The variance of principal components will be eigenvalues i.e.

$Var(z_1) = 916.902,\ Var(z_2) = 18.375,\ Var(z_3) = 7.87,\ Var(z_4) = 1.056$

The total variation explained by principal components is

$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 916.902 + 18.375 + 7.87 + 1.056 = 944.20$

As such, it can be seen that the total variation explained by principal components is same as that explained by original variables. It could also be proved mathematically as well as empirically that the principal components are uncorrelated.

The proportion of total variation accounted for by the principal components is

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4} = \frac{916.902}{944.203} = 0.97$$

Continuing, the first two components account for a proportion

$$\frac{\lambda_1 + \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4} = \frac{935.277}{944.203} = 0.99$$ of the total variance.

Hence, in further analysis, the first or first two principal components z_1 and z_2 could replace four variables by sacrificing negligible information about the total variation in the system. The scores of principal components can be obtained by substituting the values of x_i's in the equations of z_i's. For above data, the first two principal components for first observation i.e. for year 1970 can be worked out as

$$z_1 = 0.006 \times 25.0 + 0.061 \times 86 + 0.103 \times 66 + 0.993 \times 186.49 = 197.380$$
$$z_2 = 0.955 \times 25.0 - 0.296 \times 86 + 0.011 \times 66 + 0.012 \times 186.49 = 1.383$$

Similarly for the year 1971

$$z_1 = 0.006 \times 24.9 + 0.061 \times 84 + 0.103 \times 66 + 0.993 \times 124.34 = 135.54$$
$$z_2 = 0.955 \times 24.9 - 0.296 \times 84 + 0.011 \times 66 + 0.012 \times 124.34 = 1.134$$

Thus the whole data with four variables can be converted to a new data set with two principal components.

Example 5: Consider the same data as given in Example 1. The variance-covariance matrix was given as

$$\mathbf{\Sigma} = \begin{bmatrix} 2.879368 & 10.01 & -1.80905 \\ 10.01 & 199.7884 & -5.64 \\ -1.80905 & -5.64 & 3.627658 \end{bmatrix}$$

Now find the eigenvalues and eigenvectors of the above matrix. Arrange the eigenvalues in decreasing order. Let the eigenvalues in decreasing order and corresponding eigenvectors are

$\lambda_1 = 200.462 \quad \mathbf{a_1} = (0.0508,\ 0.9983,\ -0.0291)$

$\lambda_2 = 4.532 \quad \mathbf{a_2} = (-0.5737,\ 0.0530,\ \ 0.8173)$

$\lambda_3 = 1.301 \quad \mathbf{a_3} = (0.8175,\ -0.0249,\ 0.5754)$

The principal components for this data are

$$z_1 = 0.0508x_1 + 0.9983x_2 - 0.0291x_3$$
$$z_2 = -0.5737x_1 + 0.0530x_2 + 0.8173x_3$$
$$z_3 = 0.8175x_1 - 0.0249x_2 + 0.5754x_3$$

The variance of principal components will be eigenvalues i.e.

$Var(z_1) = 200.462,\ \ Var(z_2) = 4.532,\ \ Var(z_3) = 1.301$

The total variation explained by principal components is

$\lambda_1 + \lambda_2 + \lambda_3 = 200.462 + 4.532 + 1.301 = 206.295$

As such, it can be seen that the total variation explained by principal components is same as that explained by original variables. It could also be proved mathematically as well as empirically that the principal components are uncorrelated.

The proportion of total variation accounted for by the principal components is

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} = \frac{200.462}{206.295} = 0.9717 \text{ of the total variance.}$$

Continuing, the first two components account for a proportion

$$\frac{\lambda_1 + \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3} = \frac{204.994}{206.295} = 0.9937 \text{ of the total variance.}$$

Hence, in further analysis, the first or first two principal components z_1 and z_2 could replace four variables by sacrificing negligible information about the total variation in the system. The scores of principal components can be obtained by substituting the values of x_i's in the equations of z_i's. For above data, the first two principal components for first observation i.e. for first individual is

$$z_1 = 0.0508 \times 3.7 + 0.9983 \times 48.5 - 0.0291 \times 9.3$$
$$z_2 = -0.5737 \times 3.7 + 0.0530 \times 48.5 + 0.8173 \times 9.3$$

Similarly principal component scores for other individuals can be obtained. Thus the whole data with three variables can be converted to a new data set with two principal components.

Following steps of SAS may be used for performing the principal component analysis.

The PROC PRINCOMP can be used for performing principal component analysis. Raw data, a correlation matrix, a covariance matrix or a sum of squares and cross products (SSCP) matrix can be used as input data. The data sets containing eigenvalues, eigenvectors, and standardized or unstandardized principal component scores can be created as output. The basic syntax of PROC PRINCOMP is as follows:

PROC PRINCOMP Cov <options>;
BY variables;
FREQ Variable;
PARTIAL Variables;
VAR Variables;
WEIGHT Variable;
RUN;

The PROC PRINCOMP and RUN are must. However, the VAR statement listing the numeric variables to be analysed is usually used alongwith PROC PRINCOMP statement. If the DATA= data set is TYPE=SSCP, the default set of variables does not include intercept. Therefore, INTERCEPT may also be included in the VAR statement. The following options are available in PROC PRINCOMP.

A. DATA SETS SPECIFICATION

1. DATA= SAS-data-set : names the SAS data set to be analysed. This data set can be ordinary data set or a TYPE = CORR, COV, FACTOR, UCORR or UCOV data set.
2. OUT = SAS-data-set : creates an output data set containing original data alongwith principal component scores.
3. OUTSTAT-SAS-data-set : creates an output data set containing means, standard deviations, number of observations, correlations or covariances, eigenvalues and eigenvectors.

B. ANALYTICAL DETAILS SPECIFICATION

1. COV: computes the principal components from the covariance matrix. The default option is computation of principal components using a correlation matrix.
2. N=: the non-negative integer equal to the number of principal components to be computed.
3. NOINT : omits the intercept from the model
4. PREFIX= name: specifies a prefix for naming the principal components. The default option is PRIN1, PRIN2, ...
5. STANDARD (STD): standardizes the principal component scores to unit variance from the variance equal to corresponding eigenvalue.
6. VARDEF=DF | N | WDF | WEIGHT: specifies the divisor (error degree of freedom | number of observations | sun of weights | sum of weights-1) in calculating variances and standard deviations. The default option is DF.

Besides these options NOPRINT option suppresses the output. The other statements in PROC PRINCOMP are:

By variables: obtains the separate analysis on observations in groups defined by variables.

FREQ statement: It names a variable that provides frequencies of each observation in the data set. Specifically, if n is the value of the FREQ variable for a given observation, then that observation is used ‘n’ times.

PARTIAL Statement: used to analyze for a partial correlation or covariance matrix.

VAR statement: Lists the numeric variables to be analysed.

WEIGHT Statement: If we want to use relative weights for each observation in the input data set, place the weights in a variable in the data set and specify the name in a weight statement.

This is often done when the variance associated with each observation is different and the values of the weight variable are proportional to reciprocals of the variances. The observation is used in the analysis only if the value of the WEIGHT statement variable is non-missing and greater than zero.

The other closely related procedures with PROC PRINCOMP are

PROC PRINQUAL: It performs a principal component analysis of a qualitative data.

PROC CORRESP: performs correspondence analysis, which is a weighted principal component analysis of contingency tables.

For detailed steps for performing principal component analysis using SAS and SPSS, a reference may be made to link "Analysis of Data" at Design Resources Server. SAS and SPSS codes can be obtained from

http://www.iasri.res.in/design/Analysis of data/principal_component.html

5. Canonical Correlation Analysis

Canonical correlation is a technique for analyzing the relationship between two sets of variables. Each set can contain several variables. Simple and multiple correlation are special cases of canonical correlation in which one or both sets contain a single variable. This analysis actually focuses on the correlation between a linear combination of the variables in one set and a linear combination of the variables in the second set. The idea is first to determine the pair of linear combinations having the largest correlation. Next we determine the pair of linear combinations having the largest correlation among all pairs uncorrelated with the initially selected pair. This process continues until the number of pairs of canonical variables equals the number of variables in the smaller group. The pairs of linear combinations are called the **canonical variables** and their correlations are called **canonical correlations**. The canonical correlations measure the strength of association between the two sets of variables. The maximization aspect of the technique represents an attempt to concentrate a high-dimensional relationship between two sets of variables into a few pair of canonical variables.

The PROC CANCORR procedure tests a series of hypotheses that each canonical correlation and all smaller correlations are zero in population using an F-approximation. At least one of the two sets of the variables should have an approximate multivariate normal distribution. PROC CANCORR can also perform partial canonical correlation, a multivariate generalization of ordinary partial correlation. Most commonly used parametric statistical methods, ranging from t-tests to multivariate analysis of covariance are special cases of partial canonical correlations.

6. Discriminant Analysis

The term discriminant analysis refers to several types of analysis viz. classificatory discriminant analysis (used to classify observations into two or more known groups on the basis of one or more quantitative variables), Canonical discriminant analysis (a dimension reduction technique related to principal components and canonical correlation), Stepwise discriminant analysis (a variable selection technique i.e. to try to find a subset of quantitative variables that best reveals differences among the classes).

For classificatory discriminant analysis, Fisher's Discriminant function is generally used. It is described in the sequel.

Fisher's idea was to transform the multivariate observations **x** to univariate observations *y* such the *y*'s derived from the populations π_1 and π_2 were separated as much as possible. Fisher's

approach assumes that the populations are normal and also assumes the population covariance matrices are equal because a pooled estimate of common covariance matrix is used.

A fixed linear combination of the **x**'s takes the values $y_{11}, y_{12}, ..., y_{1n_1}$ for the observations from the first population and the values $y_{21}, y_{22}, ..., y_{2n_2}$ for the observations from the second population. The separation of these two sets of univariate *y*'s is assessed in terms of the differences between $\bar{y}_1$ and $\bar{y}_2$ expressed in standard deviation units. That is,

$$\text{separartion} = \frac{|\bar{y}_1 - \bar{y}_2|}{s_y}, \text{ where } s_y^2 = \frac{\sum_{j=1}^{n_1}(y_{1j} - \bar{y}_1)^2 + \sum_{j=1}^{n_2}(y_{2j} - \bar{y}_2)^2}{n_1 + n_2 - 2}$$

is the pooled estimate of the variance. The objective is to select the linear combination of the **x** to achieve maximum separation of the sample means $\bar{y}_1$ and $\bar{y}_2$.

Result: The linear combination $y = \hat{\mathbf{l}}'\mathbf{x} = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}\mathbf{x}$ maximizes the ratio

$$\frac{(\text{Squared distance between sample mean of y})}{(\text{Sample variance of y})} = \frac{(\bar{y}_1 - \bar{y}_2)^2}{s_y^2}$$

$$= \frac{(\hat{\mathbf{l}}\bar{\mathbf{x}}_1 - \hat{\mathbf{l}}'\bar{\mathbf{x}}_2)^2}{\hat{\mathbf{l}}'S_{pooled}\hat{\mathbf{l}}}$$

$$= \frac{(\hat{\mathbf{l}}'\mathbf{d})^2}{\hat{\mathbf{l}}'S_{pooled}\hat{\mathbf{l}}'}$$

over all possible coefficient vectors $\hat{\mathbf{l}}'$ where $\mathbf{d} = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$. The maximum of the above ratio is $\mathbf{D}^2 = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$, the Mahalanobis distance.

Fisher's solution to the separation problem can also be used to classify new observations. An allocation rule is as follows.

Allocate $\mathbf{x}_0$ to π_1 if $y_0 = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}\mathbf{x}_0 \geq \hat{\mathbf{m}} = \frac{1}{2}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2)$

and to π_2 if $y_0 < \hat{\mathbf{m}}$

If we assume the populations π_1 and π_2 are multivariate normal with a common covariance matrix, the a test of $\mathbf{H}_0 : \mu_1 = \mu_2$ versus $\mathbf{H}_1 : \mu_1 \neq \mu_2$ is accomplished by referring

$$\frac{(n_1 + n_2 - p - 1)}{(n_1 + n_2 - 2)p}\left(\frac{n_1 n_2}{n_1 + n_2}\right)\mathbf{D}^2$$

to an F-distribution with $\nu_1 = p$ and $\nu_2 = n_1 + n_2 - p - 1$ degrees of freedom. If $\mathbf{H}_0$ is rejected, we can conclude the separation between the two populations is significant.

Following procedure statements of SAS that can be used for above discriminant analyses.

PROC DISCRIM : Classificatory discriminant analysis

PROC CANDISC : Cannonical discriminant analysis

PROC STEPDISC : Stepwise discriminant analysis.

SPSS: To Obtain a Discriminant Analysis, from the menus choose: Analyze → Classify →Discriminant... → Select an integer-valued grouping variable and click Define Range to specify the categories of interest → Select the independent, or predictor, variables. (If the grouping variable does not have integer values, Automatic Recode on the Transform menu will create one that does.

Example 6: {Example 11.3 in Johnson and Wichern, 2002}. To construct a procedure for detecting potential hemophilia 'A' carriers, blood samples were analyzed for two groups of women and measurements on two variables, $x_1 = \log_{10}(AHF\ activity)$ and $x_2 = \log_{10}(AHF-like\ antigens)$ recorded. The first group of $n_1 = 30$ women were selected from a population who do not carry hemophilia gene (normal group). The second group of $n_2 = 22$ women were selected from known hemophilia 'A' carriers (obligatory group). The mean vectors and sample covariance matrix are given as

$$\bar{\mathbf{x}}_1 = \begin{bmatrix} -0.0065 \\ -0.0390 \end{bmatrix}, \quad \bar{\mathbf{x}}_2 = \begin{bmatrix} -0.2483 \\ 0.0262 \end{bmatrix} \text{ and } \mathbf{S}_{pooled}^{-1} = \begin{bmatrix} 131.158 & -90.423 \\ -90.423 & 108.147 \end{bmatrix}$$

Now the linear discriminant function is

$$y_0 = \hat{\mathbf{l}}'\mathbf{x}_0 = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}\mathbf{x}_0$$

$$= \begin{bmatrix} 0.2418 & -0.0652 \end{bmatrix} \begin{bmatrix} 131.158 & -90.423 \\ -90.423 & 108.147 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= 37.61\, x_1 - 28.92\, x_2$$

Moreover

$$\bar{y}_1 = \hat{\mathbf{l}}'\mathbf{x}_1 = \begin{bmatrix} 37.61 & -28.92 \end{bmatrix} \begin{bmatrix} -0.0065 \\ -0.0390 \end{bmatrix} = 0.88$$

$$\bar{y}_2 = \hat{\mathbf{l}}'\mathbf{x}_2 = \begin{bmatrix} 37.61 & -28.92 \end{bmatrix} \begin{bmatrix} -0.2483 \\ -0.0262 \end{bmatrix} = -10.10$$

and the mid-point between these means is

$$\hat{\mathbf{m}} = \frac{1}{2}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_{pooled}^{-1}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2) = \frac{1}{2}(\bar{y}_1 + \bar{y}_2) = -4.61$$

Now to classify a women who may be a hemophilia 'A' carrier with $x_1 = -0.210$ and $x_2 = -0.044$.

We calculate: $y_0 = \hat{\mathbf{l}}'\mathbf{x}_0 = 37.61\, x_1 - 28.92\, x_2 = -6.62$. Since $y_0 < \hat{\mathbf{m}}$ we classify the women in π_2 population, *i.e.*, to obligatory carrier group.

7. Factor Analysis

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called **factors**.
For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms. The factors typically are viewed as broad concepts or ideas that may describe an observed phenomenon. For example, a basic desire of obtaining a certain social level might explain most consumption behaviour. These unobserved factors are more interesting to the social scientist than the observed quantitative measurements.
Factor analysis is generally an exploratory/descriptive method that requires many subjective judgments. It is a widely used tool and often controversial because the models, methods, and subjectivity are so flexible that debates about interpretations can occur. The method is similar to principal components although, as the textbook points out, factor analysis is more elaborate. In one sense, factor analysis is an inversion of principal components. In factor analysis we model the observed variables as linear functions of the "factors." In principal components, we create new variables that are linear combinations of the observed variables. In both PCA and FA, the dimension of the data is reduced. Examples of fields where factor analysis is involved include physiology, health, intelligence, sociology, and sometimes ecology among others.

A common rationale behind factor analytic methods is that the information gained about the interdependencies between observed variables can be used later to reduce the set of variables in a dataset. It may help to deal with data sets where there are large numbers of observed variables that are thought to reflect a smaller number of underlying/latent variables. It is one of the most commonly used inter-dependency techniques and is used when the relevant set of variables shows a systematic inter-dependence and the objective is to find out the latent factors that create a commonality.

The PROC FACTOR can be used for several types of common factor and component analysis. Both orthogonal and oblique rotations are available. We can compute scoring coefficients by the regression method. All major statistics computed by PROC FACTOR can also be saved in an output DATA SET. The PROC FACTOR can be invoked by the following statements:

PROC FACTOR <options>;
VAR variables;
PRIORS Communalities;
PARTIAL Variables;
FREQ Variable;
WEIGHT Variable;
BY variables;
RUN;

Usually only the VAR statement is needed in addition to the PROC FACTOR statement. The some of the important options available with PROC FACTOR are:

METHOD=NAME : specifies the method of extracting factors. The default option is METHOD = PRINCIPAL, which yields principal component analysis if no PRIORS is used or if PRIORS = ONE is specified; if a PRIORS = value other than one is specified, a principal factor anlaysis is performed.

METHOD= PRINT : yields iterated principal factor analysis.

METHOD=ML : performs maximum- likelihood factor analysis.

METHOD = ALPHA : produced alpha factor analysis.

METHOD =ULS: produced unweighted least squares factor analysis.

NFACTORS=n | NFACT=n | N=n specifies the maximum number of factors to be extracted.

PRIORS =name: (ASMC | INPUT | MAX | ONE | RANDOM | SMC) : specifies a method for computing prior communality estimates

ROTATE=name: gives the rotation method. The default is ROTATE=NONE. FACTOR performs the following orthogonal rotation methods:

- EQUAMAX
- ORTHOMAX
- QUARTIMAX
- PARSIMAX
- VARIMAX

After the initial factor extraction, the common factors are uncorrelated with each other. If the factors are rotated by an orthogonal transformation, the rotated factors are uncorrelated. If the factors are rotated by an oblique transformation, the rotated factors become correlated. Oblique rotations often produce more useful patterns than do orthogonal rotations. However, a consequence of correlated factors is that there is no single unambiguous measure of the importance of a factor in explaining a variable. Thus, for oblique rotations, the pattern matrix doesn't provide all the necessary information for interpreting the factors.

SPSS: To Perform Factor Analysis. From the menus choose: Analyze → Data Reduction → Factor... → Select the variables for the factor analysis.

To understand the role of Factor Analysis, consider the following examples

Example 7: What underlying attitudes lead people to respond to the questions on a political survey as they do? Examining the correlations among the survey items reveals that there is significant overlap among various subgroups of items--questions about taxes tend to correlate with each other, questions about military issues correlate with each other, and so on. With factor analysis, you can investigate the number of underlying factors and, in many cases, you can identify what the factors represent conceptually. Additionally, you can compute factor scores for each respondent, which can then be used in subsequent analyses. For example, you might build a logistic regression model to predict voting behavior based on factor scores.

Example 8: A manufacturer of fabricating parts is interested in identifying the determinants of a successful salesperson. The manufacturer has on file the information shown in the following table. He is wondering whether he could reduce these seven variables to two or three factors, for a meaningful appreciation of the problem.

Data Matrix for Factor Analysis of seven variables (14 sales people)

Sales Person	**Height** (x_1)	**Weight** (x_2)	**Education** (x_3)	**Age** (x_4)	**No. of Children** (x_5)	**Size of Household** (x_6)	**IQ** (x_7)
1	67	155	12	27	0	2	102
2	69	175	11	35	3	6	92
3	71	170	14	32	1	3	111

4	70	160	16	25	0	1	115
5	72	180	12	30	2	4	108
6	69	170	11	41	3	5	90
7	74	195	13	36	1	2	114
8	68	160	16	32	1	3	118
9	70	175	12	45	4	6	121
10	71	180	13	24	0	2	92
11	66	145	10	39	2	4	100
12	75	210	16	26	0	1	109
13	70	160	12	31	0	3	102
14	71	175	13	43	3	5	112

Can we now collapse the seven variables into three factors? Intuition might suggest the presence of three primary factors: maturity revealed in age/children/size of household, physical size as shown by height and weight, and intelligence or training as revealed by education and IQ.

The sales people data have been analyzed by the SAS program. This program accepts data in the original units, automatically transforming them into standard scores. The three factors derived from the sales people data by principal component analysis (SAS program) are presented below:

Three-factor results with seven variables

Variable	Sales People Characteristics			Communality
	Factor I	Factor II	Factor III	
Height	0.59038	0.72170	-0.30331	0.96140 (sumsq I,II and III)
Weight	0.45256	0.75932	-0.44273	0.97738
Education	0.80252	0.18513	0.42631	0.86006
Age	-0.86689	0.41116	0.18733	0.95564
No. of Children	-0.84930	0.49247	0.05883	0.96730
Size of Household	-0.92582	0.30007	-0.01953	0.94756
IQ	0.28761	0.46696	0.80524	0.94918
Sum of squares	3.61007	1.85136	1.15709	
Variance summarized	0.51572	0.26448	0.16530	Average=0.94550

Factor Loadings

The coefficients in the factor equations are called "factor loadings". They appear above in each factor column, corresponding to each variable. The equations are:

$$\mathbf{F}_1 = 0.59038\, x_1 + 0.45256\, x_2 + 0.80252\, x_3 - 0.86689\, x_4 - 0.84930\, x_5 - 0.92582\, x_6 + 0.28761\, x_7$$

$$\mathbf{F}_2 = 0.72170\, x_1 + 0.75932\, x_2 + 0.18513\, x_3 + 0.41116\, x_4 + 0.49247\, x_5 + 0.30007\, x_6 + 0.46696\, x_7$$

$$\mathbf{F}_3 = -0.30331\, x_1 - 0.44273\, x_2 + 0.80252\, x_3 + 0.18733\, x_4 + 0.58830\, x_5 - 0.01953\, x_6 + 0.80524\, x_7$$

The factor loadings depict the relative importance of each variable with respect to a particular factor. In all the three equations, education (x_3) and IQ (x_7) have got positive loading factor indicating that they are variables of importance in determining the success of sales person.

Variance summarized

Factor analysis employs the criterion of maximum reduction of variance - variance found in the initial set of variables. Each factor contributes to reduction. In our example Factor I accounts for 51.6% of the total variance. Factor II for 26.4% and Factor III for 16.5%. Together the three factors "explain" almost 95% of the variance.

Communality

In the ideal solution the factors derived will explain 100% of the variance in each of the original variables, "Communality" measures the percentage of the variance in the original variables that is captured by the combinations of factors in the solution. Thus communality is computed for each of the original variables. Each variables communality might be thought of as showing the extent to which it is revealed by the system of factors. In our example the communality is over 85% for every variable. Thus the three factors seem to capture the underlying dimensions involved in these variables.

There is yet another analysis called varimax rotation, after we get the initial results. This could be employed if needed by the analyst. We do not intend to dwell on this and those who want to go into this aspect can use SAS program for varimax rotation.

8. Cluster Analysis

The basic aim of the cluster analysis is to find “natural” or “real” groupings, if any, of a set of individuals (or objects or points or units or whatever). This set of individuals may form a complete population or be a sample from a larger population. More formally, cluster analysis aims to allocate a set of individuals to a set of mutually exclusive, exhaustive groups such that individuals within a group are similar to one another while individuals in different groups are dissimilar. This set of groups is called partition or dissection. Cluster analysis can also be used for summarizing the data rather than finding natural or real groupings. Grouping or clustering is distinct from the classification methods in the sense that the classification pertains to a known number of groups, and the operational objective is to assign new observations to one of these groups. Cluster analysis is a more primitive technique in that no assumptions are made concerning the number of groups or the group structure. Grouping is done on the basis of similarities or distances (dissimilarities). Some of these distance criteria are:

Euclidean distance: This is probably the most commonly chosen type of distance. It is the geometric distance in the multidimensional space and is computed as:

$$d(\mathbf{x},\mathbf{y}) = \left[\sum_{i=1}^{p}(x_i - y_i)^2\right]^{1/2} = \sqrt{(\mathbf{x}-\mathbf{y})'(\mathbf{x}-\mathbf{y})}$$

where $\mathbf{x},\mathbf{y}$ are the p-dimensional vectors of observations.

Note that Euclidean (and squared Euclidean) distances are usually computed from raw data, and not from standardized data. This method has certain advantages (e.g., the distance between any two objects is not affected by the addition of new objects to the analysis, which may be outliers). However, the distances can be greatly affected by differences in scale among the dimensions from which the distances are computed. For example, if one of the dimensions denotes a measured length in centimeters, and you then convert it to millimeters (by

multiplying the values by 10), the resulting Euclidean or squared Euclidean distances (computed from multiple dimensions) can be greatly affected (i.e., biased by those dimensions which have a larger scale), and consequently, the results of cluster analyses may be very different. Generally, it is good practice to transform the dimensions so they have similar scales.

Squared Euclidean distance: This measure is used in order to place progressively greater weight on objects that are further apart. This distance is square of the Euclidean distance.

Statistical distance: The statistical distance between the two p-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ is $d(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})'\mathbf{s}^{-1}(\mathbf{x}-\mathbf{y})}$, where $\mathbf{s}$ is the sample variance-covariance matrix.

Many more distance measures are available in literature. For details, a reference may be made to Romesburg (1984).

Several types of clusters are possible using various PROC statements:

- Disjoint cluster place each object in one and only one cluster. (PROC FASTCLUS, PROC VARCLUS).
- Hierarchical clusters are organised so that one cluster may be entirely contained within another cluster, but no other kind of overlap between clusters is allowed. (PROC CLUSTER, PROC VARCLUS).
- Overlapping clusters can be constrained to limit the number of objects that belongs simultaneously to two clusters. (PROC OVERCLUS)
- Fuzzy clusters are defined by a probabilities or grade of membership of each object in each cluster. Fuzzy clusters can be disjoint, hierarchical or overlapping.

SPSS: To Obtain a Hierarchical Cluster Analysis, from the menus choose: Analyze → Classify → Hierarchical Cluster... → For clustering cases, select at least one numeric variable, For clustering variables, select at least three numeric variables → Optionally, one can select an identification variable to label cases.

References

Bryant, F. B., & Yarnold, P. R. (1995). Principal components analysis and exploratory and confirmatory factor analysis. In L. G. Grimm & P. R. Yarnold (Eds.), *Reading and understanding multivariate analysis*. Washington, DC: American Psychological Association

Joseph, F.H. and Anderson, R.E. (1995). *Multivariate data analysis: with readings*. 4th Edition, Prentice-Hall, Inc.

Johnson, R.A. and Wichern, D.W. (2002). *Applied Multivariate Statistical Analysis*. 5th Edition, Pearson Education Inc., New Delhi.

Romesburg, H.C. (1984). *Cluster Analysis for Researchers*. Lifetime Learning Publications, California.

Some E-learning Resources

kybele.psych.cornell.edu/~edelman/Psych-465-Spring-2003/**PCA-tutorial**.pdf

www.cs.princeton.edu/picasso/mats/**PCA-Tutorial**-Intuition_jp.pdf -

en.wikipedia.org/wiki/Principal_components_analysis

Linear and Integer programming: concept and its application in agriculture

HarishKumar H V, Bishal Gurung, and Achal Lama
ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012

I. Introduction

Linear programming (LP) is a mathematical modeling technique designed to optimize (maximize or minimize) the usage of limited resources. To define "LP is a mathematical technique of studying wherein we consider maximization (or minimization) of a linear expression (called the objective function) subjected to a number of linear equalities and inequalities (called linear restrictions)".

II. History and application of LP

In 1939, during World War II, a Soviet economist Leonid Kantorovich used LP to plan expenditures and returns in order to reduce costs of the army and to increase losses incurred to the enemy.

LP has wide applications in various fields like military, industry, agriculture, transportation, health system, economics and behavioral sciences etc., and is also utilized for some engineering problems. Transportation, energy, telecommunications, and manufacturing are the major industries that use linear programming models. LP has proven useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

III. Assumptions of linear programming

1. The LP models are "***deterministic***" in nature: Assumes everything is certain and equation is mathematical in nature.
2. The LP models are "***proportional***" in nature: This condition follows directly from linearity assumptions for objective function and constraints. This means that the objective function and constraints expand and contract proportionately to the level of each activity. This condition represents constant returns to scale rather than economies or diseconomies of scale.
3. The LP models are "***additive***" in nature: That is the Left Hand Side (LHS) should be equal to Right Hand Side (RHS). The assumption of proportionality guarantees linearity if and only if the joint effects or interactions are non-existent. That means the total contribution of all activities is identical to sum of the constraints per each activity individually.
4. The decision variables are ***"divisible"***: That is the fractional levels for decision variables are permissible, the objective function and constraints are continuous function.
5. Non-negativity: The value of variables must be zero or positive but not negative.

So LP is a special case of mathematical programming to achieve the best outcome (such as maximum profit or minimum cost) in a mathematical model whose requirements are represented by linear relationships.

Here is a simple example.

Reddy Mikks (R-M) company produce both interior and exterior paints from two raw materials M_1 and M_2. The following table provides the basic data of the problem

Table 1: Basic data of problem

Particulars	Tonnes of raw material required per tonne of		Maximum availability with R-M (tones)
	Exterior paint	Interior paint	
Raw material M_1	6	4	24
Raw material M_2	1	2	6
Profit per tonne ($ 000's)	5	4	-

- The market survey restricts maximum daily demand of interior paints to 2 tonnes.
- Additionally the daily demand for interior paint cannot exceed that of exterior paint more than 1 tonne.
- The R-M company wants to determine the optimum product mix of interior and exterior paints that maximizes total daily profit.

Let us formulate the problem

LP model includes three basic elements

1) **Decision variables** that we seek to determine
   X_1=Production of exterior paints (in tonnes)
   X_2=Production of interior paints (in tonnes)

2) **Objective function** that we aim to optimize
   Main objective is to maximize total daily profit
   Let Z represents total daily profit (in $ 000's)
   Max $Z = 5X_1+4X_2$

3) **The constraints** that we need to satisfy
   a) Restriction on raw material usage: The usage of raw material for production of both paints should not exceed raw material availability.
      - **Usage of raw material M_1: $6X_1+4X_2 \leq 24$**
      - **Usage of raw material M_2: $X_1+2X_2 \leq 6$**
   b) Demand restrictions
      - **Maximum daily demand of interior paint is limited to 2 tonnes: $X_2 \leq 2$**

- **Excess of daily production (**daily demand for interior paint cannot exceed that of exterior paint more than 1 tonne**):** $\mathbf{X_2 - X_1 \leq 1}$

So the LP model for above optimization problem looks like below

Max $Z = 5X_1+4X_2$

Subject to;

$6X_1+4X_2 \leq 24$

$X_1+2X_2 \leq 6$

$X_2 \leq 2$

$X_2 - X_1 \leq 1$

$X_1 \& X_2 \geq 0$

IV. Standard form of LP model

To solve LP problem manually it must be put in a common form which we call as standard form.

Properties of standard form are

- **All the constraints should be expressed as equations by adding slack or surplus and or artificial variables.**

  A constraint of the type $\leq (\geq)$ can be converted to an equation by adding ***slack*** variable to (subtracting ***surplus*** variable from) the left side of the constraint.

Ex 1: $3X_1+2X_2 \leq 6$

$3X_1+2X_2 +S_1=6$, where S_1 is a slack variable represents the unused amount of resources

Ex 2: $2X_1+X_2 \geq 6$

$2X_1+X_2 -S_2=6$, where S_2 is a surplus variable represents the excess amount of resources

Note: The introduction of slack and surplus variables alters neither the nature of the constraint nor the objective function. Accordingly such variables are incorporated into objective function with zero co-efficient.

- **The right hand side of each constraint should be made non-negative (if not).**

  The RHS of the equation can always be made non-negative by multiplying both the sides by -1.

  Ex: $2X_1+3X_2-7X_3=-5$ can be written as $-2X_1-3X_2+7X_3=5$

  $2X_1-X_2 \leq -5$ can be written as $-2X_1+X_2 \geq 5$ (the direction of inequality is reversed when both sides are multiplied by -1)

- **The objective function must be maximization type**

  Solving of maximization problem is easier than solving of minimization problem. So we can convert minimization form to maximization form for easy calculation and later

we can interpret it as minimization solution. The maximization of a function is equivalent to minimization of a negative of the same function and vice-versa.

For a given set of constraints,

Max $Z=5X_1+2X_2+3X_3$ is mathematically **equivalent** to Min $(-Z) = -5X_1-2X_2-3X_3$.

Equivalence means that for the same set of constraints the optimal values of X_1, X_2 and X_3 are the same in both cases. The only difference is that the values of the objective function, although equal numerically, will appear with opposite signs.

Table 2: General and standard form of LP model involving only less than or equal constraints (≤)

General form	Standard form
Max $Z = 5X_1+4X_2$ subject to; $6X_1+4X_2 \leq 24$ $X_1+2X_2 \leq 6$ $X_2 \leq 2$ $X_2 – X_1 \leq 1$ X_1 & $X_2 \geq 0$	Max $Z = 5X_1+4X_2+0S_1+0S_2+0S_3+0S_4$ subject to; $6X_1+4X_2 +S_1 =24$ $X_1+2X_2 + S_2 =6$ $X_2 +S_3=2$ $X_2 – X_1 + S_4=1$ X_1 , X_2, S_1, S_2, S_3 & $S_4 \geq 0$

V. Artificial variable (AV):

In case of problems with infeasible solution artificially we introduce a variable into objective function to obtain feasible solution. We use AV only to start solution and subsequently force them to be zero in the solution otherwise the resulting solution will be infeasible. To guarantee such assignments in the optimal solution, AVs are incorporated into objective function with very large positive co-efficient in minimization problem or very large negative co-efficient in maximization problem.

AVs do change the nature of constraint since they are added only to one side of inequality. That is if the original constraint is an equation (=) or of the type greater than or equal to (≥), then we have no longer basic starting feasible solution.

Table 3: General and standard form of LP model involving all kind of constraints (≤, =, ≥)

General form	Standard form
Max $Z = 5X_1+2X_2$ subject to; $6X_1+X_2 \leq 6$ $4X_1+3X_2 \geq 12$ $X_1+X_2 =1$ X_1 & $X_2 \geq 0$	Max $Z = 5X_1+2X_2+0S_1+0S_2-MA_1-MA_2$ subject to; $6X_1+X_2 +S_1 =6$ $4X_1+3X_2 - S_2+A_1 =12$ $X_1+X_2 +A_2=2$ X_1 , X_2, S_1, S_2, A_1 & $A_2 \geq 0$

VI. Solution to LP problem:

There are two approaches for solving LP problems.

1) Graphical approach and

2) Simplex technique

1) Graphical approach: LP problems which involve only two decision variables can be solved graphically. Since it is not possible to display the set of feasible solution for more than two variables in a graph for locating best optimal solution. There are two graphical solution methods namely, extreme point solution method and iso-profit (Cost) function line method. Of these, extreme point solution method is most commonly used method for solving LP problem involving two decision variables.

Extreme point solution method: Extreme point refers to corner of the feasible region i.e. the point lies at the intersection of two constraint equations. In this method, the co-ordinates of all corner or extreme points of the feasible region are determined and then value of the objective function at each of these points is computed and compared. The co-ordinates of an extreme point where the optimal (maximum or minimum) value of the objective function is found represent the solution of the given LP problem.

Example:

$$\text{Max } Z = 5X_1 + 7X_2$$

Subjected to:

$$X_1 \leq 6$$
$$2X_1 + 3X_2 \leq 19$$
$$X_1 + X_2 \leq 8$$
$$X_1, X_2 \geq 0$$

Solution:

Here we are not going to add any slack or surplus variable but we are just putting it into equation.

$$X_1 = 6$$
$$2X_1 + 3X_2 = 19$$
$$X_1 + X_2 = 8$$

$X_1 + X_2 = 8$ Extreme points: a) $X_1=0$ $X_2=8$, Co-ordinates:(0,8) b) $X_2=0$ $X_1=8$, Co-ordinates:(8,0)	$2X_1 + 3X_2 = 19$ Extreme points: a) $X_1=0$ $X_2=6.33$, Co-ordinates: (0,6.33) b) $X_2=0$ $X_1=9.5$, Co-ordinates: (9.5,0)	$X_1=6$ Extreme points: a) $X_1=6$, $X_2=0$, (6,0)

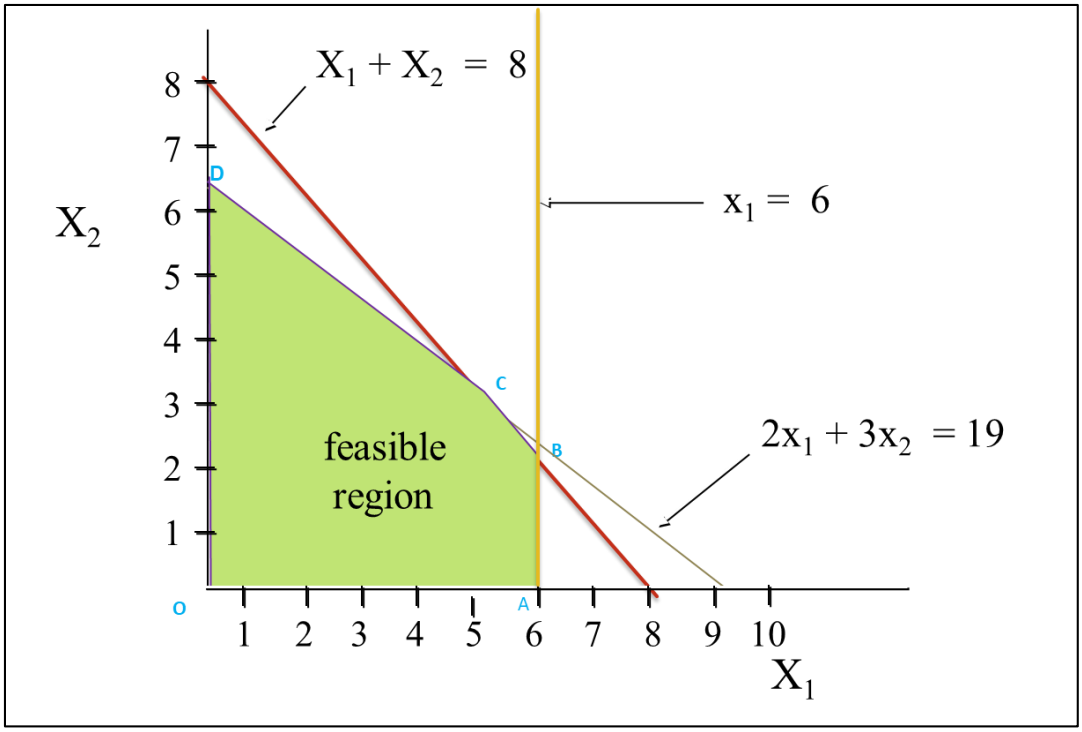


Figure 1: Combined-Constraint Graph Showing Feasible Region

The shaded zone is called feasible area where all the constraints holds good or this region satisfies all constraints so it is called **feasible region**.

- The corners or vertices of the feasible region are referred to as the extreme points.
- An optimal solution to an LP Maximization problem can be found at an extreme point of the feasible region.
- When looking for the optimal solution, you do not have to evaluate all feasible solution points.
- Consider only the extreme points of the feasible region.

Table 4: Value of objective function at extreme points of feasible region

Extreme Point	Co-Ordinates	Z value (**Z**=5X_1+7X_2)
O	(0,0)	0
A	(6,0)	30
B	(6,2)	44
C	**(5,3)**	**46**
D	(0,6.33)	44.31

At point C all constraints are satisfied and the Z value is highest hence it is optimal point.

Solution: At X_1=5 and X_2=3, Max Z=46

2) Simplex method:

It is an algorithm adopted to solve LP problem which employs an iterative procedure that starts at a feasible corner point, normally the origin and systematically moves from one feasible point to another point until it reaches optimum point. Dantzig introduced this method in 1947. Unique solution resulting from setting n-m variables equal to zero is called basic solution and the

basic solution which satisfies the non-negativity restriction is basic feasible solution. Simplex method deals with basic feasible solution and each basic feasible solution is associated with an iteration.

Computational details of Simplex method

Step 1: Using the standard form determine a starting basic feasible solution by setting n-m appropriate (non-basic) variables at zero level

Step 2: Select an entering variable from current non-basic variable which can improve the value of objective function.

If none exists then current solution is optimal otherwise go to step 3.

Step 3: Select a leaving variable from among the current basic variables that must be set to zero (becomes non-basic) when the entering variable becomes basic.

Step 4: Determine the new basic solution by making the entering variable basic and leaving variable as non-basic.

Go to step 2 to get optimum

To call a table as optimum table, the criteria is as below

a) Maximization: All values in Zj-Cj are positive

b) Minimization: All values in Zj-Cj are negative

Criteria for identification of entering variable

a) Maximization: Non basic variable with most negative coefficient in Zj-Cj index row

b) Minimization: Non basic variable with most positive coefficient in Zj-Cj index row

Criteria for identification of Leaving variable:

Current leaving variable associated with minimum ratio in both maximization and minimization cases

Number in the intersection of entering and leaving variable is called pivot element.

New basic solution in each iteration is obtained by applying Gauss-Jordon method which effects change by using 2 types of computations.

Type 1: New Pivot Equation (NPE)

NPE=(Old pivot equation/ Pivot element)

Type 2: All other equations

New equation= {Old equation-[(its entering column coefficient)*(NPE)}

Linear programming solvers are now part of many spreadsheet packages, such as Microsoft Excel. The leading commercial package is “LINDO”. We can solve LP problems in packages like “R” and “SAS” also.

VII. Special cases in simplex method of application

1. **Degeneracy:**

   In case of model consisting of at least one redundant (No longer needed or not useful) constraint then the optimum value won’t improve upon iterations instead same solution is generated over the iterations.

   Example:

Max $Z = 3X_1+9X_2$

Subject to;

$X_1+4X_2 \leq 8$

$X_1+2X_2 \leq 4$

$X_1 \& X_2 \geq 0$

In above case the first constraint is a redundant constraint.

2. **Alternative optima:**

Alternative optima exists when objective function running parallel to one of the constraints. Then the objective function will assume same optimal value at more than one solution point.

Example:

Max $Z = 2X_1+4X_2$

Subject to;

$X_1+2X_2 \leq 5$

$X_1+X_2 \leq 4$

$X_1 \& X_2 \geq 0$

In above case the objective function runs parallel to first constraint.

3. **Unbounded solutions**

The solution to a maximization LP problem is unbounded if the value of the solution may be made indefinitely large without violating any of the constraints. Sometimes feasible solution for the given LP problem exists and this has infinite values for the objective function. For real problems, this is the result of improper formulation.

4. **Infeasible or non-existent solutions**

No unique solution to the LP problem satisfies all the constraints, including the non-negativity conditions. Graphically, this means a feasible region does not exist. Causes includes formulation error, too high expectations by management or too many restrictions have been placed on the problem (i.e. the problem is over-constrained).

VIII. Integer programming:

In case of linear programming, the decision variables considered are supposed to take any real value. However in practical situations it makes no sense in assigning a real value to a variable where it has meaning only when it takes only integer values. To be clear let us consider a practical problem like optimum size of herd in a dairy project, it makes no sense if our optimal value from LP solution is 5.8.

In such situations, we naturally tend to round-off the optimal value to the nearest integer value say "6" in above example. However, the round-off may have following fundamental problems,

a) The round-off solution may not be feasible.

b) The objective function value given by the rounded-off solutions (even if some are feasible) may not be the optimal one.

c) Even if some of the rounded-off solutions are optimal, checking all the rounded-off solutions is computationally expensive.

So integer programming deals with the solution of mathematical programming problems in which some or all the variables can assume non-negative integer values only.

Types of integer programming problems

1) Pure integer programming problem: An integer programming problem in which all variables are required to be integers.

2) Mixed integer programming problem: If some variables are restricted to be integer and some are not restricted i.e. can be continuous or fractional.

3) Binary integer programming problem/ 0-1 programming problems: If some or all variables are restricted to be either "0" or "1". It can be pure or mixed.

The general form of integer programme is as below

Max $Z = 7X_1+9X_2$
subject to;

$-X_1+3X_2 \leq 6$

$7X_1+X_2 \leq 35$

X_1 & X_2 are non-negative integers.

IX. Applications of Linear Programming in agriculture

Case-1: Naidu Dairy farm uses at least 800 Kg's of *Special feed* daily. The *Special feed* is a mixture of corn silage and soybean meal with the following composition,

Table 5: Constituents of special feed

Feed stuff	In terms of Kg per every Kg of feed stuff		
	Protein	Fiber	Cost (Rs./Kg)
Corn silage	0.09	0.02	20
Soybean meal	0.60	0.06	62

The dietary requirements of *Special feed* must have at least 30 per cent protein and at most 5 per cent fiber. Now the Naidu Dairy farm wishes to determine the daily minimum cost of feed mix?

Solution:

Decision variables:

X_1= Quantity of corn silage to be used in feed mix (Kg's)

X_2=Quantity of soybean meal to be used in feed mix (Kg's)

Objective function

Min $Z=20X_1+62X_2$

Constraints

Demand constraint (Daily requirement): $X_1+X_2 \geq 800$

Protein constraint: $0.09X_1 +0.60X_2 \geq 0.30(X_1+X_2)$ on simplification $-0.21 X_1+0.30 X_2 \geq 0$

Fiber constraint: $0.02X_1 +0.06X_2 \leq 0.05 (X_1+X_2)$ on simplification $-0.03 X_1+0.01 X_2 \leq 0$

Overall the LP model looks like

Min $Z=20X_1+62X_2$
Subjected to,

$X_1+X_2 \geq 800$
$-0.21 X_1+0.30 X_2 \geq 0$
$-0.03 X_1+0.01 X_2 \leq 0$
$X_1 \& X_2 \geq 0$

R code for the above LP problem

```
library(lpSolve)
obj=c(20,62)
mat=matrix(c(1,1,-0.21,0.3,-0.03,0.01), nrow=3, byrow=TRUE)
rhs=c(800,0,0)
dir=c(">=",">=","<=")
prod.sol= lp("min", obj, mat, dir, rhs, compute.sens = TRUE)
prod.sol$status
prod.sol$objval
prod.sol$solution
prod.sol$duals
prod.sol$duals.from
prod.sol$duals.to
prod.sol$sens.coef.from
prod.sol$sens.coef.to
```

A) Optimal solution

Z	29835.29
X1	470.58
X2	329.41

The daily minimum cost of feed mix by using 470.58 Kg of corn silage and 329.41 Kg of soybean meal is Rs. 29835.29.

B) Sensitivity analysis

a) Maximum change in resource availability (RHS of binding constraints)

Binding Constraint	Shadow price	RHS	Sensitivity (Range)
Special feed	37.29	800	0 to $1*10^{30}$
Protein	82.35	0	-168 to 138

b) Maximum change in marginal cost (Co-efficients of DV's in objective function)

Variable	Value of DV's	Unit price	Sensitivity (Range)
Corn silage (X1)	470.58	20	-43.40 to 62.00
Soybean meal (X2)	329.41	62	20 to $1*10^{30}$

Case 2: Venkatesh, a Crop+Dairy farming system based farmer wishes to maximize the total revenue with the available resources. The below table provides the information on the resource availability and the information on resource requirement for the enterprises from his past experience. Ragi being the regular diet of Venkatesh's family he needs minimum 1 acre of his land to be under the same which also serves the fodder security of his dairy. Since the dairy is earning him the regular income for family maintenance he insists at least one cross breed (CB) cow in his farming system.

Resources	Availability	Per unit requirement			
		Tomato	Cabbage	Ragi	CB Cow
Land (Acres)	4	-	-	-	-
Labour (Man days)	350	180	65	32	38
Capital (Rs.)	250000	125000	65000	12500	33000
Water (acre inches)	100	24.5	17.8	9.4	0.5
Returns (Rs.)	-	280000	135000	19000	65000

Solution:

Decision variables:

X_1= Area under Tomato crop to be taken (Acres)

X_2= Area under Cabbage crop to be taken (Acres)

X_3= Area under Ragi crop to be taken (Acres)

X_4= Number of cross breed cows to be considered in his farming system

Objective function

Max $Z=280000X_1+135000X_2+19000\ X_3+65000\ X_4$

Constraints

Land constraint (Overall): $X_1+X_2+\ X_3\leq 4$

Labour constraint: $180X_1+65X_2+\ 32X_3+\ 38X_4 \leq 350$

Capital constraint: $125000X_1+65000X_2+\ 12500X_3+\ 33000X_4 \leq 250000$

Water constraint: $24.5X_1+17.8X_2+\ 9.4X_3+\ 0.5X_4 \leq 100$

Constraint for Ragi mandate: $X_3 \geq 1$

Constraint for Dairy mandate: $X_4 \geq 1$

Overall the LP model looks like

Max $Z=280000X_1+135000X_2+19000\ X_3+65000\ X_4$

Subjected to,

$X_1+X_2+\ X_3\leq 4$

$180X_1+65X_2+\ 32X_3+\ 38X_4 \leq 350$

$125000X_1+65000X_2+\ 12500X_3+\ 33000X_4 \leq 250000$

$24.5X_1+17.8X_2+\ 9.4X_3+\ 0.5X_4 \leq 100$

$X_3 \geq 1$

$X_4 \geq 1$

$X_1+X_2+\ X_3 \geq 0$ & X_4 is a non-negative integer

R code for the above LP problem

```
library(lpSolve)
obj=c(280000,135000,19000,65000)
mat=matrix(c(1,1,1,0,180,65,32,38,125000,65000,12500,33000,24.5,17.8,9.4,0.5,0,0,1,0,0,0,0,1)
, nrow=6, byrow=TRUE)
rhs=c(4,350,250000,100,1,1)
dir=c("<=","<=","<=","<=",">=",">=")
```

```
prod.sol= lp("max", obj, mat, dir, rhs, int.vec=4, compute.sens = TRUE)
prod.sol$status
prod.sol$objval
prod.sol$solution
prod.sol$duals
prod.sol$duals.from
prod.sol$duals.to
prod.sol$sens.coef.from
prod.sol$sens.coef.to
```

A) Optimal solution

Z	536713.28
X1	1.37
X2	0.50
X3	1
X4	1

The maximum total revenue that the farmer can achieve is Rs. 5,36,713.3/- by cultivating Tomato, Cabbage and Ragi in 1.37, 0.50 and 1 acre respectively, along with 1 CB cow.

B) Sensitivity analysis

a) Maximum change in resource availability (RHS of binding constraints)

Binding Constraint	Shadow price	RHS	Sensitivity (Range)
Labour	370.62	350	283.20 to 364.48
Capital	1.70	250000	239944.4 to 284847.8
Ragi	-14188.81	1	0 to 1.98
CB cow	-5391.60	1	0 to 2.52

b) Maximum change in marginal profit (Co-efficients of DV's in objective function)

Variable	Value of DV's	Unit price	Sensitivity (Range)
Tomato (X1)	1.37	280000	259615.4 to 373846.15
Cabbage (X2)	0.50	135000	118802.5 to 145600
Ragi (X3)	1	19000	$-1*10^{30}$ to 33188.81
CB cow (X4)	1	65000	$-1*10^{30}$ to 70391.61

X. References

Dorfman, R. 1996. Linear Programming & Economic Annalysis. McGraw-Hill. New York.

Hadley, G. 1997. Linear programming. Narosa publishing house. New Delhi.

Rao, S.S. 2007. Engineering Optimization: Theory and Practice. New Age International Publishers. New Delhi.

Taha, H.A. 2007. Operation Research: In Introduction. Seventh edition. Prentice Hall India. New Delhi.

https://nptel.ac.in/courses/105108127/

Python

Madhu

HarishKumar H V, Bishal Gurung, and Achal Lama

ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Characteristics of Python

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python Syntax

```
print('India is my country.')
```

Variables

```
x=2
y="India"
print(x)
print(y)
```

Type Casting

```
x=str(5)
y=int(5.0)
z=float(5)
print(x)
print(y)
print(z)
print(type(x))
print(type(y))
```

Single & Double Quotes

```
x="india"
y='india'
print(x)
print(y)
```

Multiline Strings

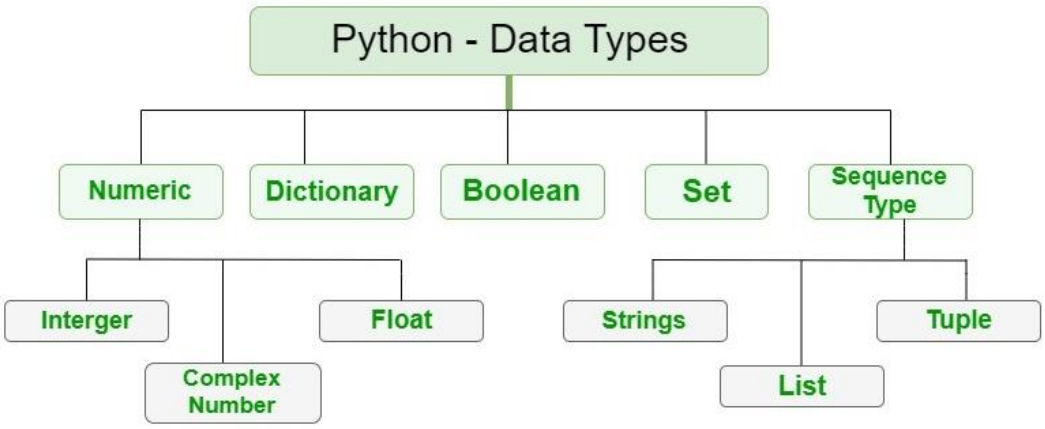
```
a = """hello,
good morning,
h r u,
all."""
print(a)

a = '''hello,
good morning,
h r u
all.'''
print(a)
```

Python Data Types

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



Numeric

In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Note – type() function is used to determine the type of data type.

Float

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

int

```
x = 1
y = 3565
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

complex

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Sequence Type

In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

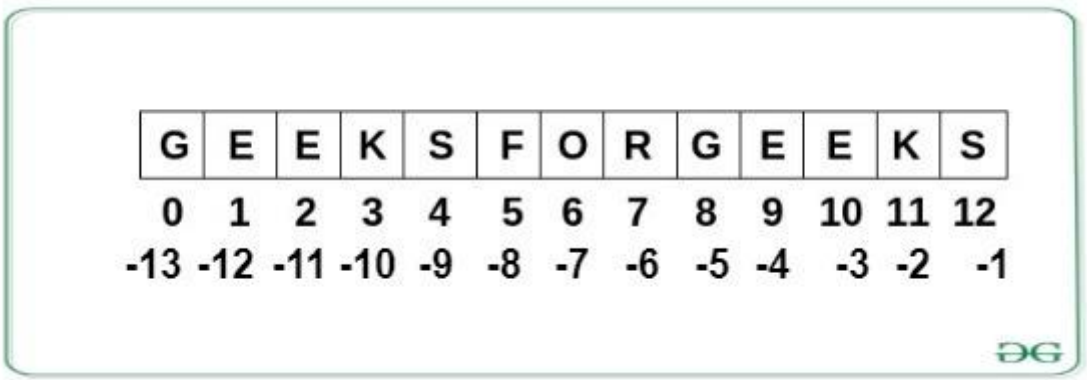
- String
- List
- Tuple

1) String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Creating String
Strings in Python can be created using single quotes or double quotes or even triple quotes.

Accessing elements of String
In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.




```
print("Hello")
print('Hello')

a = "Hello"
print(a)
```

Strings are Arrays

```
a = "Hello, World!"  # 0....n
print(a[1])
print(a[2])
print(a[7])
```

Looping Through a String

```
for x in "banana":
  print(x)
```

String Slicing

```
b = "Hello, World!"
print(b[2:5])

b = "Hello, World!"
print(b[:5])

b = "Hello, World!"
print(b[2:])

b = "Hello, World!"
print(b[-5:-2])
print(b[-1])
```

Strings Functions

```
a = "hello, World!"
print(a.upper())                #Converts a string into upper case
print(a.capitalize())           #Converts the first character to upper case
print(a.casefold())            #Converts string into lower case
print(a.split())             #Splits the string at the specified separator, and returns a list
print(a.lower())              #Converts a string into lower case
print(a.strip())                # Returns a trimmed version of the string
                              #returns "Hello, World!"
print(a.replace("h", "J"))           #Returns a string where a specified value is replaced with a
specified value
print(a.isdigit())                #Returns True if all characters in the string are digits
print(a.isupper())           #Returns True if all characters in the string are upper case
print(len(a))            #len() function returns the length of a string
```

String Concatenation

```
a = "Hello"
b = "World"
c = a + b
print(c)

a = "Hello"
b = "World"
c = a + "     "     + b
print(c)

"""we cannot combine strings and numbers"""
```

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

we can combine strings and numbers by using the **format() method!**

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {}

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders.

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

LIST [] O,C,DUPLICATE

TUPLES () O, NOT CHANGE,D

DICTIONARY KEY:VALUE

SET {} O,C,NOT DUPLICATE

2) List

Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating List
Lists in Python can be created by just placing the sequence inside the square brackets[].

Accessing elements of List
In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc

Lists are used to store multiple items in a single variable.

List items are ordered, changeable, and allow duplicate values. Lists are created using **square brackets.**

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
print(type(thislist))

thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)                                    # Allow Duplicates
```

Length of a List

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))                    #len function for finding the number of values in a tuple

list1 = ["apple", "banana", "cherry"]              #List items can be of any data type
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
print(type(list2))
print(type(list1))

list1 = ["abc", 34, True, 40, "male"]              #A list can contain different data types
print(type(list1))
```

Indexes of List items

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])

thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
print(thislist[-2])
print(thislist[-3])
print(thislist[0])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])

thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:       # true
  print("Yes, 'apple' is in the fruits list")

thislist = ["apple", "banana", "cherry"]
if "orange" in thislist:        #false
  print("no, 'orange' is not in the fruits list")    #not executed
```

Change Item Value

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]                    #Change the values "banana"
and "cherry" with the values "blackcurrant" and "watermelon
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the **insert() method**.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")          # insert() method inserts an item at the specified index

thislist = ["apple", "banana", "cherry"]
thislist.append("orange")                                  #append() method add an item to the
end of the list
```

```
print(thislist)

thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]

print(thislist+tropical)

thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)       #extend() method does not have to append lists only, you can
add any iterable object (tuples, sets, dictionaries etc.).
print(thislist)
print(type(thistuple))

thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")              #remove() method removes the specified item
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop(-1)                       #pop() method removes the specified index
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop()          #do not specify the index, the pop() method removes the last item
print(thislist)

thislist = ["apple", "banana", "cherry"]
del thislist[0]                        #del keyword also removes the specified index
print(thislist)

thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist)                                #Delete the entire list

thislist = ["apple", "banana", "cherry"]
thislist.clear()                         #list still remains, but it has no content
print(thislist)

thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)

thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])

thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):          #until true
  print(thislist[i])
```

```
  i = i + 1
```

Sorting List

```
thislist = ["orange", "mango", "apricot", "apple", "banana"]
thislist.sort()              #sort() method that will sort the list alphanumerically, ascending, by default
print(thislist)

thislist = [100, 50, 50, 82, 23]
thislist.sort()
print(thislist)

thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = False)         #To sort descending, use the keyword argument reverse = True
print(thislist)
```

Copy a List

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()      #to make a copy, one way is to use the built-in List method copy()
print(mylist)

thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)     #make a copy is to use the built-in method list()
print(mylist)
```

Join Two Lists

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2       #by using the + operator
print(list3)

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)           #Another way to join two lists is by appending all the items from list2 into list1, one by one

print(list1)
```

3) Tuple

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple

In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

Tuples are used to store multiple items in a single variable.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with **round brackets. ()**

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)

thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)                                    #allow duplicates

thistuple = ("apple", "banana", "cherry")
print(len(thistuple))          #len function for finding the number of values in a tuple

thistuple = ("apple",)         #use comma (,) if tuple having single value otherwise it is
considered as string
print(type(thistuple))
print(len(thistuple))

thistuple = ("apple")           #NOT a tuple
print(type(thistuple))
```

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
print(type(tuple1))
print(type(tuple2))

tuple1 = ("abc", 34, True, 40, "male")
print(type(tuple1))

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])           #access the elements of a tuple
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable.

You can convert the tuple into a list, change the list, and convert the list back into a tuple

```
x = ("apple", "banana", "cherry")   #can not change the tuple
y = list(x)               #convert tuple into list
y[1] = "kiwi"               # change list
x = tuple(y)              #convert list into tuple

print(x)

thistuple = ("apple", "banana", "cherry")               #Since tuples are immutable, they do not
have a build-in append() method
y = list(thistuple)
y.append("orange")                                  # list have append method
thistuple = tuple(y)
print(thistuple)

thistuple = ("apple", "banana", "cherry")
y = ("orange","mango")
thistuple += y          # allowed to add tuples to tuples

print(thistuple)

thistuple = ("apple", "banana", "cherry")        #  they do not have a build-in remove() method
y = list(thistuple)
y.remove("apple")                           # list have remove method
thistuple = tuple(y)
print(thistuple)

thistuple = ("apple", "banana", "cherry")
del thistuple
```

print(thistuple) #this will raise an error because the tuple no longer exists

Packing & Unpacking Tuples

When we create a tuple, we normally assign values to it. This is called **"packing"** a tuple.

In Python, we are also allowed to extract the values back into variables. This is called **"unpacking".**

a = ("Delhi", 5000, "Agriculture") #PACKS values into variable a

(City, student, type_ofcollege) = a #UNPACKS values of variable a

print(City,student,type_ofcollege)

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list.

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)

Join Tuples

tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)

fruits = ("apple", "banana", "cherry")

```
mytuple = fruits * 2

print(mytuple)
```

Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.
Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

Booleans represent one of two values: True or False.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)

print(bool("Hello"))        #Almost any value is evaluated to True if it has some sort of content.
print(bool(15))           #Any string is True, except empty strings.
print(bool(0))           # Any number is True, except 0.
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))            # Any list, tuple, set, and dictionary are True, except empty ones.
print(bool(False))
print(bool(None))
```

Set

In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets
Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by 'comma'. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Accessing elements of Sets

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Sets are used to store multiple items in a single variable.

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Sets are written with **curly brackets.**

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)

thisset = {"apple", "banana", "cherry"}
print(len(thisset))                  # len() function used to  find length

set1 = {"abc", 34, True, 40, "male"}
print(set1)                # A set with strings, integers and boolean values
print(type(set1))              # type function return data type

thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)

thisset = {"apple", "banana", "cherry"}
```

```
print("banana" in thisset)
print("kiwi" in thisset)
```

Add an Item

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")             #  to add one item to a set use the add() method.
print(thisset)

thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)                    #To add items from another set into the current set,
use the update() method
print(thisset)

thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)              #update() method does not have to be a set only, it can be any
iterable object (tuples, lists, dictionaries etc.)
print(thisset)
```

Remove Item

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("apple")
thisset.remove("kiwi")              #To remove an item in a set, use the remove(), or the
discard() method.
print(thisset)                      #If the item to remove does not exist, remove() will raise an
error

thisset = {"apple", "banana", "cherry"}
thisset.discard("kiwi")               #If the item to remove does not exist, discard() will NOT
raise an error.
print(thisset)

thisset = {"apple", "banana", "cherry"}
x = thisset.pop()                 #   pop() method to remove an item,but this method will remove
the last item
print(x)                          #Remember that sets are unordered, so you will not know what item
that gets removed.The return value of the pop() method is the removed item.
print(thisset)

thisset = {"apple", "banana", "cherry"}
thisset.clear()                   #   clear() method empties the set
print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
del thisset           #  del keyword will delete the set completely
print(thisset)
```

Join Two Sets

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)        # union() method returns a new set with all items from both sets
print(set3)

set1 = {"a", "b" ,"b" ,"c"}
set2 = {1, 2, 3}

set1.update(set2)          #update() method inserts the items of set2 into set1.
                        #Both union() and update() will exclude any duplicate items
print(set1)
```

Both union() and update() will exclude any duplicate items.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)          #intersection_update() method will keep only the items that are
present in both sets.

print(x)

x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)     #intersection() method will return a new set, that only contains
                        # the items that are present in both sets.

print(z)

x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)      #symmetric_difference_update() method will keep
only the elements that are
                                #NOT present in both sets.

print(x)
```

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)          # symmetric_difference() method will return a new set,
                                       # that contains only the elements that are NOT present in both sets.

print(z)
```

Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{}.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

Accessing elements of Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. There is also a method called get() that will also help in accessing the element from a dictionary.

Dictionaries are used to store data values in **key:value pairs**.

A dictionary is a collection which is ordered, changeable and do not allow duplicates.

Ordered

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key.

```
thisdict = {
  "brand": "Ford",

  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
print(len(thisdict))              # len () function returns length
print(type(thisdict))             # type() function show datatype

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020            #Dictionaries cannot have two items with the same key.
                        # Duplicate values will overwrite existing values
}
print(thisdict)
```

Accessing Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
y = thisdict.get("model")       #Get the value of the "model" key
z = thisdict.keys()         #keys() method will return a list of all the keys in the dictionary
b = thisdict.values()        #values() method will return a list of all the values in the dictionary
print(x)
print(y)
print(z)
print(b)
```

Change Dictionary Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})   #update() method will update the dictionary with the items from the given argument.
print(thisdict)
```

Adding Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})        #update() method will update the dictionary with the items from a given argument
print(thisdict)
```

Removing Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")     #pop() method removes the item with the specified key name
```

```
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()          #popitem() method removes the last inserted item
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]       #del keyword removes the item with the specified key name
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict                #del keyword can also delete the dictionary completely
print(thisdict)

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()          #clear() method empties the dictionary
print(thisdict)
```

Loop Through a Dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:        #Print all key names in the dictionary, one by one
  print(x)
for y in thisdict:
  print(thisdict[y])      #Print all values in the dictionary, one by one
for x in thisdict.values(): #values() method to return values of a dictionary
```

```
  print(x)
for x in thisdict.keys():
  print(x)                  #keys() method to return the keys of a dictionary
for x, y in thisdict.items(): #items methods to return values for both key and values
  print(x, y)
```

Copy a Dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()          # Make a copy of a dictionary with the copy() method
print(mydict)
print(thisdict)
mydict = dict(thisdict)     # to make a copy is to use the built-in function dict()
print(mydict)
```

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
```

```
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
print(myfamily)
```

Python Arrays

Array in Python can be created by importing array module. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

import array as arr

```
# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()
```

Accessing Python Array Elements

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])

print("First element:", a[0])
print("Second element:", a[1])
print("Last element:", a[-1])
```

Slicing Python Arrays

```
import array as arr

numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)
```

```
print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

Changing and Adding Elements

```
import array as arr

numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# changing first element
numbers[0] = 0
print(numbers)     # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)     # Output: array('i', [0, 2, 4, 6, 8, 10])
```

```
import array as arr

numbers = arr.array('i', [1, 2, 3])

numbers.append(4)
print(numbers)     # add one item to the array using the append() method

numbers.extend([5, 6, 7])
print(numbers)     # add several items using the extend() method
```

```
import array as arr

odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])

numbers = arr.array('i')   # concatenate two arrays using + operator
numbers = odd + even

print(numbers)
```

```
import array as arr

number = arr.array('i', [1, 2, 3, 3, 4])

del number[2]  # removing third element
print(number)  # Output: array('i', [1, 2, 3, 4])
```

```
del number  # deleting entire array
print(number)  # Error: array is not defined

import array as arr

numbers = arr.array('i', [10, 11, 12, 12, 13])

numbers.remove(12)    # remove() method to remove the given item
print(numbers)
print(numbers.pop(2))   # pop() method to remove an item at the given index
print(numbers)
```

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

1) Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

+ Add two operands or unary plus x + y+ 2:

- Subtract right operand from the left or unary minus   x - y- 2

* Multiply two operands x * y

/ Divide left operand by the right one (always results into float)   x / y

% Modulus - remainder of the division of left operand by the right   x % y (remainder of x/y)

// Floor division - division that results into whole number adjusted to the left in the number line   x // y

** Exponent - left operand raised to the power of right   x**y (x to the power y)

```
"""

x = 3
y = 2
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
```

```
print('x ** y =',x**y)
```

2) Comparison operators

Comparison operators are used to compare values. It returns either True or False according to the condition.

> Greater than - True if left operand is greater than the right x > y

< Less than - True if left operand is less than the right x < y

== Equal to - True if both operands are equal x == y

!= Not equal to - True if operands are not equal x != y

>= Greater than or equal to - True if left operand is greater than or equal to the right x >= y

<= Less than or equal to - True if left operand is less than or equal to the right x <= y

```
"""

x = 5
y = 10

print('x > y is',x>y)
print('x < y is',x<y)
print('x == y is',x==y)
print('x != y is',x!=y)
print('x >= y is',x>=y)
print('x <= y is',x<=y)
```

3) Logical operators

Logical operators are the and, or, not operators.

and True if both the operands are true x and y

or True if either of the operands is true x or y

not True if operand is false (complements the operand) not x

```
"""

x = True
y = False

print('x and y is',x and y)    #true true
```

```
print('x or y is',x or y)       #either true

print('not x is',not x)
```

4) Assignment operators

Assignment operators are used in Python to assign values to variables.

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5

```
a = 21
b = 10
c = 0

c = a + b
print ("Value of c is ", c)

c += a
print ("Value of c is ", c)

c *= a
print ("Value of c is ", c)

c /= a
print ("Value of c is ", c)

c  = 2
c %= a
print ("Value of c is ", c)

c **= a
print ("Value of c is ", c)
```

```
c //= a
print ("Value of c is ", c)
```

5) Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

& AND Sets each bit to 1 if both bits are 1

| OR Sets each bit to 1 if one of two bits is 1

^ XOR Sets each bit to 1 if only one of two bits is 1

~ NOT Inverts all the bits

```
a = 10          #1010    0101
b = 4           #0100

# Print bitwise AND operation
print("a & b =", a & b)         #0000

# Print bitwise OR operation
print("a | b =", a | b)       #1110

# Print bitwise NOT operation
print("~a =", ~a)               #  ~a = ~1010
                        #    = -(1010 + 1)
                        #    = -(1011)
                        #     = -11 (Decimal)

# print bitwise XOR operation
print("a ^ b =", a ^ b)         # Returns 1 if one of the bits is 1 and the other is 0 else returns
false.
```

6) Shift Operators

These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.

Bitwise right shift: Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result.

Bitwise left shift: Shifts the bits of the number to the left and fills 0 on voids right as a result.

```
a = 10
b = -10

# print bitwise right shift operator
print("a >> 1 =", a >> 1)
print("b >> 1 =", b >> 1)

a = 5
b = -10

# print bitwise left shift operator
print("a << 1 =", a << 1)
print("b << 1 =", b << 1)
```

7) Identity operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)     # Output: False

print(x2 is y2)         # Output: True

print(x3 is y3)       # Output: False
```

8) Membership operators

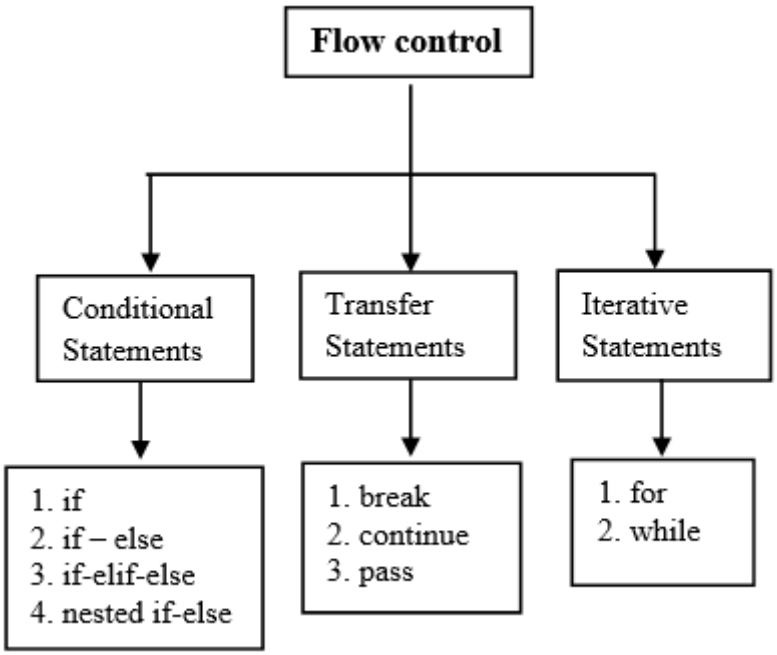
in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

```
x = 'Hello world'
y = {1:'a',2:'b'}

print('H' in x)         # Output: True

print('hello' not in x)   # Output: True

print(1 in y)            # Output: True
```

```
print('a' in y)          # Output: False
```

Control Flow Statements

The flow control statements are divided into three categories
1. Conditional statements
2. Iterative statements.
3. Transfer statements



Python If ... Else

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Short Hand If

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Nested If

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

The pass Statement

```
a = 33
b = 200

if b > a:
 pass
```

while Loop

```
i = 1
while i < 6:
  print(i)
  i += 1
```

break Statement

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

continue Statement

```
i = 0     #
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Looping Through a String

```
for x in "banana":
  print(x)
```

range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)

for x in range(2, 6):
  print(x)              # not including 6

for x in range(2, 30, 3):           #third parameter is the increment value
  print(x)

for x in range(6):
  print(x)
else:                 #else keyword in a for loop specifies a block of code to be executed when the
loop is finished
  print("Finally finished!")

for x in range(6):
  if x == 3: break                #else block will NOT be executed if the loop is stopped by a
break statement
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Functions

You use functions in programming to bundle a set of instructions that you want to use repeatedly. That means that a function is a piece of code written to carry out a specified task.

There are three types of functions in Python:

- **User-Defined Functions (UDFs),** which are functions that users create to help them out.

- **Anonymous functions,** which are also called **lambda functions** because they are not declared with the standard def keyword.

• **Built-in functions,** such as help() to ask for help, min() to get the minimum value, print() to print an object to the terminal.

Creating a Function

```
def my_function():
  print("Hello from a function")

"""**Calling** **a** **Function** """

def my_function():
  print("Hello from a function")

my_function()

def my_function(fname):                    #A parameter is the variable listed inside the parentheses in the function definition.
  print(fname + " Refsnes")

my_function("Emil")                        #An argument is the value that is sent to the function when it is called.
my_function("Tobias")
my_function("Linus")
```

Number of Arguments

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")

def my_function(*kids):                    #do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")

def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")          #send arguments with the key = value syntax

def my_function(**kid):                    #number of keyword arguments is unknown, add a double ** before the parameter name
  print("His last name is " + kid["lname"])
```

```
my_function(fname = "Tobias", lname = "Refsnes")
```

Default Parameter Value

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()                #If we call the function without argument, it uses the default value
my_function("Brazil")
```

Passing a List as an Argument

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

```
x = lambda a : a + 10
print(x(5))

Max = lambda a, b : a if(a > b) else b          # Example of lambda function using if-else

print(Max(1, 2))
```

Difference Between Lambda functions and def defined function

```
def cube(y):
    return y*y*y

lambda_cube = lambda y: y*y*y


print(cube(5))

print(lambda_cube(5))
```

Python Built-In Functions

all()

The python all() function accepts an iterable object (such as list, dictionary, etc.). It returns true if all items in passed iterable are true. Otherwise, it returns False. If the iterable object is empty, the all() function returns True.

```
k = [1, 3, 4, 6]             # all values true
   print(all(k))

   k = [0, False]               # all values false
   print(all(k))

   k = [1, 3, 7, 0]           # one false value
   print(all(k))

   k = [0, False, 5]         ## one true value
   print(all(k))

   k = []                    # empty iterable
print(all(k))

test1 = []
   print(test1,'is',bool(test1))
   test1 = [0]
   print(test1,'is',bool(test1))
   test1 = 0.0
   print(test1,'is',bool(test1))
   test1 = None
   print(test1,'is',bool(test1))
   test1 = True
   print(test1,'is',bool(test1))
   test1 = 'Easy string'
print(test1,'is',bool(test1))
```

```
x = 10
print('Absolute value of -40 is:', abs(x))    #abs() function is used to return the absolute value of a number
floating = -20.83
print('Absolute value of -20.83 is:', abs(floating))
y =  bin(x)  #bin() function is used to return the binary representation of a specified integer.
print (y)
s = sum([1, 2,4 ])              #sum() function is used to get the sum of numbers of an iterable, i.e., list.
print(s)
print(float(9))              # float() function change into float number
print(complex(9))              # complex() function change into complex number
```

Data Handling and Visualization using NumPy, Pandas, Matplotlib and Seaborn

Sanchita Naha

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012
sanchita.naha@icar.gov.in

Introduction:

Python offers many software packages for smooth data handling required for deploying Machine Learning or Deep Learning projects. Most important and very frequently used packages of them are NumPy and Pandas for data handling in array or tabular format and Matplotlib for data visualization.

Let us start with the NumPy library first. NumPy stands for Numerical Python. It is a library consisting of functions to handle multidimensional array objects and a collection of routines for processing arrays. NumPy was created in 2005 by Travis Oliphant. It is an open-source project, it can be used freely. NumPy array objects are 50x faster than traditional Python lists. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the code for declaration and manipulation of multidimensional arrays and its contents using NumPy. To install NumPy in local system use the following code in Python editor.

```
pip install numpy as np
```

After installation, run the following code:

```
import numpy as np

print(np.__version__)
```

Declare a one-dimensional array with the following code:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

Datatype of an array:

```
print(arr.dtype)
```

Create arrays of float and string type:

```
arr_float = np.array([10.2,23.0,68.5,98.7,5.0])

print(arr_float)

print(arr_float.dtype)
```

```
arr_string = np.array(['ramayanas','b','c','d','e'])
print(arr_string)
arr_string.dtype
```

Declare array of zeroes and ones:

```
arr = np.zeros(5)
arr = np.zeros([2,3]) arr
arr = np.ones(5) arr
```

Dimensions in Array:

0-D Arrays: 0-D arrays, or Scalars, are the elements in an array

```
import numpy as np
arr = np.array(42)
print(arr)
```

I-D Array: An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array. These are the most common and basic arrays.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D Arrays: An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

3-D arrays: An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
arr_0D = np.array(42)
arr_1D = np.array([1, 2, 3, 4, 5])
arr_2D = np.array([[1, 2, 3], [4, 5, 6]])
arr_3D = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print('Dimension of arr_oD is: ',arr_0D.ndim)
print('Dimension of arr_oD is: ',arr_1D.ndim)
print('Dimension of arr_oD is: ',arr_2D.ndim)
print('Dimension of arr_oD is: ',arr_3D.ndim)
```

Higher Dimensional Arrays An array can have any number of dimensions. When the array is created, you can define the number of dimensions by using the ndmi argument.

```
arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)

print('number of dimensions :', arr.ndim)
```

Indexing of an array:

Access Array Elements: Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

```
arr = np.array([1, 2, 3, 4])

print(arr[0])

Get third and fourth elements from the following array and add them.

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

Access 2-D Arrays: To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element. Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

```
# Access the element on the first row, second column:

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])

#Access the element on the 2nd row, 5th column:

print('5th element on 2nd row: ', arr[1, 4])
```

Access 3-D Arrays: To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

```
arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr_3d.ndim)

print(arr_3d[0, 1, 2])

# Printing size (total number of elements) of array

print("Size of array: ", arr.size)
```

The first number represents the first dimension, which contains two arrays: [[1, 2, 3], [4, 5, 6]] and: [[7, 8, 9], [10, 11, 12]] Since we selected 0, we are left with the first array: [[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays: [1, 2, 3] and: [4, 5, 6] Since we selected 1, we are left with the second array: [4, 5, 6]

The third number represents the third dimension, which contains three values: 4 5 6 Since we selected 2, we end up with the third value: 6

Negative Indexing Use negative indexing to access an array from the end.

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Array Slicing: Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: [start:end].We can also define the step, like this:[start:end:step]. If we don't pass start its considered 0 If we don't pass end its considered length of array in that dimension. If we don't pass step its considered 1 Note: The result includes the start index, but excludes the end index.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

Negative Slicing: Use the minus operator to refer to an index from the end.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

STEP Use the step value to determine the step of the slicing:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])

#Return every other element from the entire array:

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```

Slicing 2-D Arrays From the second element, slice elements from index 1 to index 4 (not included):

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

Checking the Data Type of an Array The NumPy array object has a property called dtype that returns the data type of the array:

```
arr = np.array([1, 2, 3, 4])

print(arr.dtype)

arr1 = np.array(['apple', 'banana', 'cherry'])

print(arr1.dtype)
```

Creating Arrays With a Defined Data Type We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

```
arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)

print(arr.dtype)

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)

print(arr.dtype)
```

Converting Data Type on Existing Arrays The best way to change the data type of an existing array, is to make a copy of the array with the astype() method. The astype() function creates a copy of the array, and allows you to specify the data type as a parameter. The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

```
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)

print(newarr.dtype)

arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)

print(newarr.dtype)
```

Shape of an Array The shape of an array is the number of elements in each dimension. Get the Shape of an Array NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
#Create an array with 5 dimensions using ndmin using a vector with val
ues 1,2,3,4 and verify that last dimension has value 4:

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)

print('shape of array :', arr.shape)
```

Reshaping arrays Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

print(arr.shape)

newarr = arr.reshape(4, 3)

print(newarr)
```

Reshape From 1-D to 3-D:

```
#Convert the following 1-D array with 12 elements into a 3-D array.

#The outermost dimension will have 2 arrays that contains 3 arrays, ea
ch with 2 elements:

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

print(arr)

newarr = arr.reshape(2, 3, 2)

print(newarr)

newarr1 = arr.reshape(-1, 1, 2)

print(newarr1)
```

Can We Reshape Into any Shape? Yes, if the elements required for reshaping are equal in both shapes. We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3-elements 3 rows 2D array as that would require 3x3 = 9 elements. Note: We cannot pass -1 to more than one dimension.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

Note: We can not pass -1 to more than one dimension.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr  =  arr.reshape(-1)

print(newarr)
```

Array Iteration:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:

  for y in x:

    print(y)
```

Iterating on Each Scalar Element In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):

  print(x)
```

Searching Arrays You can search an array for a certain value, and return the indexes that get a match. To search an array, use the where() method.

```
arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)
```

Addition, Subtraction, Division of elements of Matrix:

```
import numpy


# initializing matrices

x = numpy.array([[1, 2], [4, 5]])

y = numpy.array([[7, 8], [9, 10]])


print(x)

print(y)


# using add() to add matrices
```

```
print ("The element wise addition of matrix is : ")
print (numpy.add(x,y))

# using subtract() to subtract matrices
print ("The element wise subtraction of matrix is : ")
print (numpy.subtract(x,y))

# using divide() to divide matrices
print ("The element wise division of matrix is : ")
print (numpy.divide(x,y))
```

Array Multiplication:

```
import numpy

# initializing matrices
x = numpy.array([[1, 2], [4, 5]])
y = numpy.array([[7, 8], [9, 10]])

# using multiply() to multiply matrices element wise
print ("The element wise multiplication of matrix is : ")
print (numpy.multiply(x,y))

# using dot() to multiply matrices
print ("The product of matrices is : ")
print (numpy.dot(x,y))
```

Matrix transpose:

```
print ("The transpose of given matrix is : ")
print (x.T)
```

Matrix Multiplication:

```
# creating two matrices

p = [[1, 2], [2, 3]]

q = [[4, 5], [6, 7]]

print("Matrix p :")

print(p)

print("Matrix q :")

print(q)


# computing product

result = np.dot(p, q)


# printing the result

print("The matrix multiplication is :")

print(result)
```

The numpy.linspace() function returns number spaces evenly at a specified interval. Similar to numpy.arange() function but instead of step it uses sample number

```
# np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

#start = starting value; stop = end value; endpoint = true means include the last sample

# retstep = true ; stepping between samples a = np.linspace(1,10,10,retstep=True)

a = np.linspace(1,10,10,endpoint = False,retstep=True) a

#np.arange([start, ]stop, [step, ], dtype=None)

# starting number, ending position (excluding this value), step = spacing; by default it is set to 1

np.arange(1,100,9)
```

Generation of random numbers:

```
### Random Number

## random.randint(low, high=None, size=None, dtype=int)

## Return random integers from low (inclusive) to high (exclusive)
np.random.seed(10)
```

```
print(np.random.randint(100)) print(np.random.randint(100,1000,6))
print(np.random.randint(10,30,[5,4]))
```

Pandas is an open-source high-performance, easy-to-use Python library for data analysis. In this tutorial, working with Data Frame object has been illustrated. Install pandas as the following:

```
pip install pandas
```

For demonstration of pandas we have used two freely available dataset e.g., Iris.csv and titanic.csv.

```
#load Iris dataset

import pandas as pd

df = pd.read_csv('Iris.csv')

print(df.head())


#load titanic dataset

import pandas as pd

data = pd.read_csv('titanic.csv')

print(data.head())
```

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis", was created by Wes McKinney in 2008. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets and make them readable and relevant which is the most important requirement in data science. Pandas gives you answers about the data. Like: Is there a correlation between two or more columns? What is average value? Max value? Min value? Using Pandas, it is possible to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Create a dataframe with dictionary data structure:

```
import pandas as pd

mydataset = {

  'cars': ["BMW", "Volvo", "Ford"],

  'passings': [3, 7, 2]

}
```

```
myvar = pd.DataFrame(mydataset)

print(myvar)
```

Pandas as pd Pandas is usually imported under the pd alias. alias: In Python alias are an alternate name for referring to the same thing.

```
print(pd.__version__)
```

Pandas Series What is a Series? A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

```
#printing dataframe elements

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

print(myvar[0])


#printing dataframe elements

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)

print(myvar["y"])


#Note: The keys of the dictionary become the labels.

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)


calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

DataFrames Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table.

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
myvar = pd.DataFrame(data)
print(myvar)
```

A Pandas DataFrame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns. The DataFrame is like a table with rows and columns. Pandas use the *loc* attribute to return one or more specified row(s)

Locate Row:

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
print(df.loc[0])

#Return row 0 and 1:
#use a list of indexes:
print(df.loc[[0, 1]])
```

Named Indexes: With the index argument, you can name your own indexes.

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Max_rows: The number of rows returned is defined in Pandas option settings. You can check your system's maximum rows with the pd.options.display.max_rows statement.

```
import pandas as pd

print(pd.options.display.max_rows)
```

Generally in local systems the number is set to be 60, which means that if the DataFrame contains more than 60 rows, the print(df) statement will return only 60 rows. We can change the maximum number of rows number with the same statement.

```
pd.options.display.max_rows = 9999

df = pd.read_csv('Iris.csv')

print(df)

print (df.head())
```

Data Viewing: One of the most used method for getting a quick overview of the DataFrame, is the head() method. The head() method returns the headers and a specified number of rows, starting from the top.

```
#print top 7 rows of the data frame

df = pd.read_csv('Iris.csv')

print(df.head(7))
```

There is also a tail() method for viewing the last rows of the DataFrame. The tail() method returns the headers and a specified number of rows, starting from the bottom.

```
# print last 5 rows

print(df.tail())
```

Info About the Data: The DataFrame object has a method called info(), that gives you more information about the data set.

```
import pandas as pd

df = pd.read_csv('Iris.csv')

print(df.info())
```

Null Values: The info() method shows how many Non-Null values are present in each column, and in the data set. Empty values, or Null values, can be bad when analyzing data, and such rows should be removed with empty values. This is called data cleaning. Data Cleaning means fixing bad data in the data set. Bad data could be Empty cells, Data in wrong format, Wrong data, Duplicates. Empty cells can potentially give wrong result when you analyze data.

Removing Rows: One way to deal with empty cells is to remove rows that contain empty cells.

```
import pandas as pd

df = pd.read_csv('Iris.csv')

new_df = df.dropna()

print(new_df.head())

# Note: By default, the dropna() method returns a new DataFra
me, and will not change the original.

# If you want to change the original DataFrame, use the inpla
ce = True argument:

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df)
```

Replace Empty Values: Another way of dealing with empty cells is to insert a new value instead. This way no need to delete entire rows just because of some empty cells. The fillna() method allows to replace empty cells with a value:

```
df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Replace Only for Specified Columns: The example above replaces all empty cells in the whole Data Frame. To only replace empty values for one column, specify the column name for the DataFrame.

```
df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)
```

Matplotlib: Matplotlib is one of the most popular Python packages used for data visualization.It is a cross-platform library for making 2D plots from data in arrays. It provides an object- oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter. It can be used in Python and IPython shells, Jupyter notebook andweb application servers also.

Sample Codes:

```
import matplotlib.pyplot as plt
### Line Plot
x = [1,2,3,4,5,6,7,8,9,10]
y = [1,4,9,16,25,36,49,64,81,100]
plt.figure(figsize=(8,4))
```

```
plt.title("Line Plot
Graph",fontsize=15,color='red',fontweight='bold') plt.xlabel("X
Axis",fontsize=12,color='blue',fontweight='bold') plt.ylabel("Y
Axis",fontsize=12,color='blue',fontweight='bold')
## labels=[0,1,2,3,4,5,6,7,8,9,10]
##plt.xticks(labels,fontsize=15,color='green')
plt.xticks(fontsize=15,color='green')
plt.yticks(fontsize=15,color='green')
#plt.plot(x,y,'o-; o--;. ; --; --*; v; ^; o; -s; -*',label="Line
Plot",color="purple",lw=1)
plt.plot(x,y,'-o',label="Line Plot",color="purple",lw=1)
plt.legend(loc=2,fontsize=12)
plt.grid()
plt.show()
```

Output:

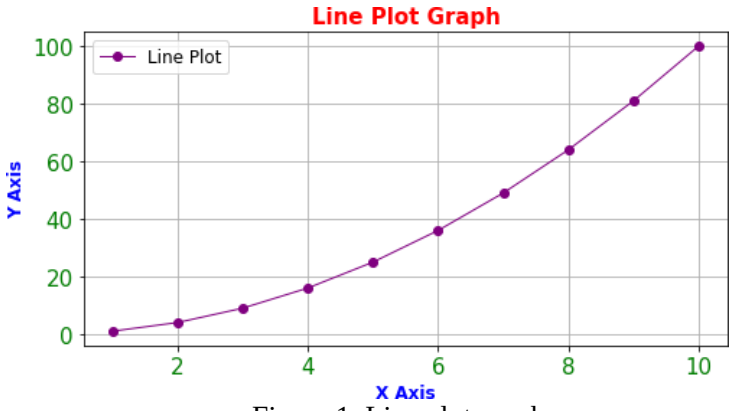


Figure 1: Line plot graph

```
# Bar Plot
x = ["A","B","C","D","E"] y = [10,20,40,30,50]
plt.figure(figsize=(8,4))
plt.title("Bar Plot Graph",fontsize=15,color='brown',
fontweight='bold') plt.xlabel("X Axis",fontsize=12,color='blue')
plt.ylabel("Y Axis",fontsize=12,color='blue')
plt.xticks(fontsize=15,color=orange)
plt.yticks(fontsize=15,color='green')
plt.bar(x,y,label="Bar Plot",color=["orange","green"],width=0.5)
plt.legend(loc=2,fontsize=12)
plt.show()
```

Output:

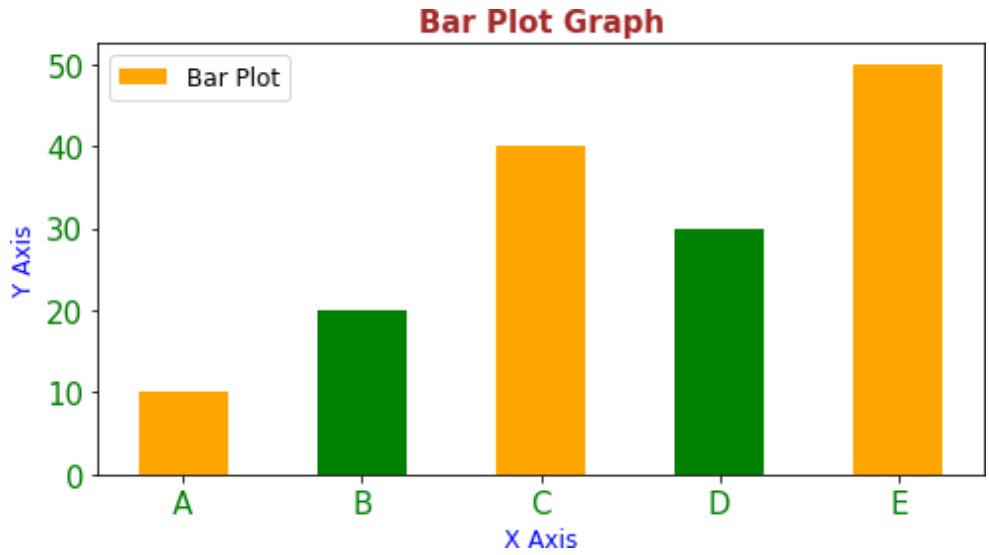


Figure 2: Bar plot graph

```
## Scatter Plot
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.figure(figsize=(6,4))
plt.title("Scatter Plot Graph",fontsize=15,color='red')
plt.xlabel("X Axis",fontsize=12,color='blue') plt.ylabel("Y
Axis",fontsize=12,color='blue')
plt.xticks(fontsize=15,color='green')
plt.yticks(fontsize=15,color='green')
plt.scatter(x,y,label="Scatter Plot",color="purple",s=40,marker =
"o") plt.legend(loc=2,fontsize=12)
plt.show()
```

Output:

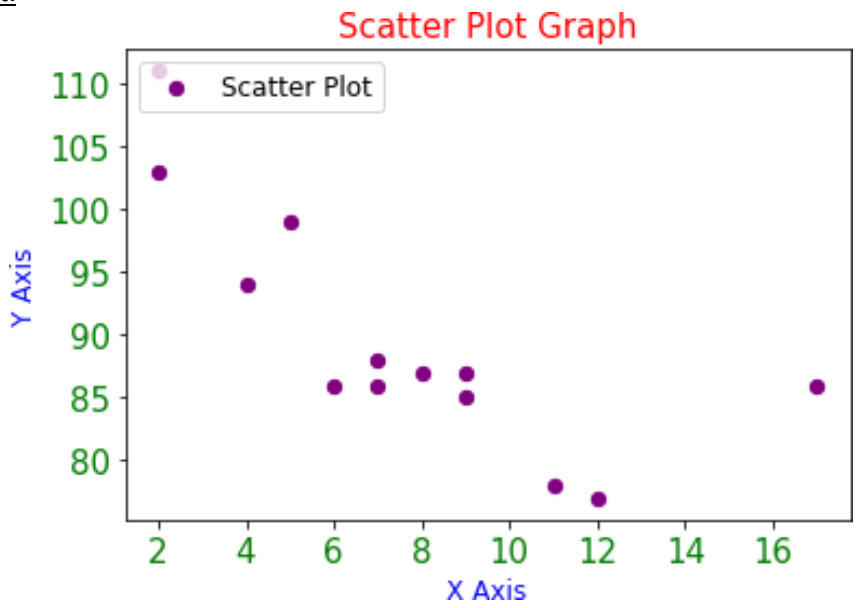


Figure 3: Scatter plot graph

Histogram

```
import numpy as np
np.random.seed(10)
data = np.random.randint(1,100,50) print(data)
plt.hist(data,rwidth=0.5,bins=5,color="pink") plt.show()
```

Output:

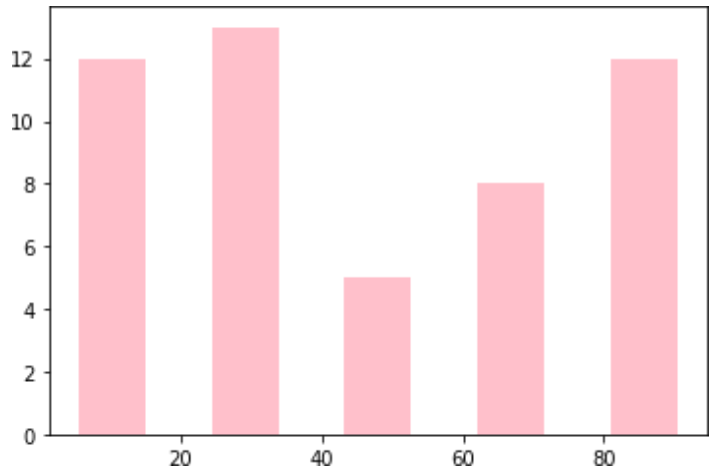


Figure 4: Histogram of a randomly generated set of numbers

simple pie chart

```
import numpy as np
import matplotlib.pyplot as plt
labels=['playing','sleeping','reading','eating'] sizes =
[25,25,25,25]
colors=['red','green','yellow','blue']
plt.pie(sizes, labels=labels, colors=colors,
autopct="%.2f%%",) plt.axis('equal')
plt.show()
```

Output:

Figure 5: Pie chart

```
### Pie Chart
plt.figure(figsize=(6,6)) slices = [90,80,30,70,10,100]
activities =
["Playing","Eating","Sleeping","Reading","Gyming","Gaming"]
cols = ["red","green","orange","purple","pink","yellow"]
plt.pie(slices,labels=activities,colors=cols,autopct="%1.2f%%
", explode=[0,0,0.3,0,0.3,0])
plt.show()
```

Output:

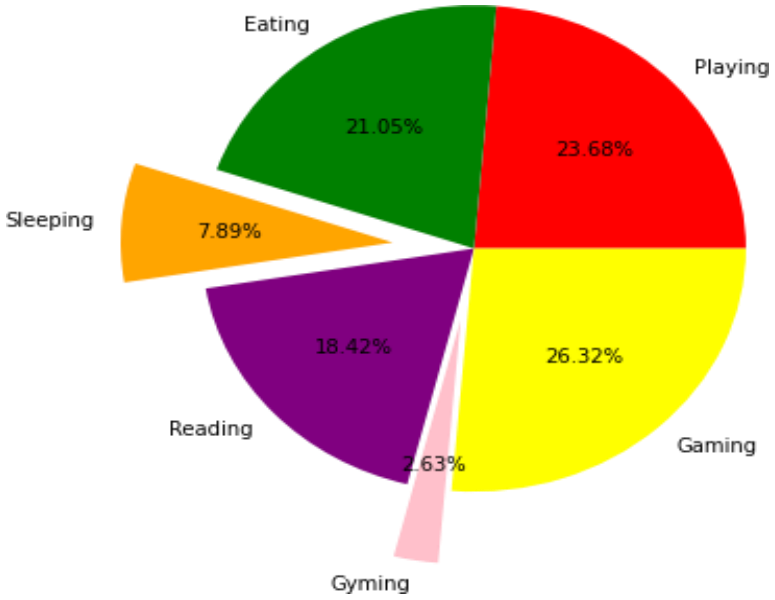


Figure 6: Pie chart with explode feature

```
### Line Plot with two lines
x = [1,2,3,4,5]
y1 = [10,20,40,30,50]
y2 = [5,15,25,35,45]
plt.figure(figsize=(8,4))
plt.title("Line Plot Graph",fontsize=15,color='red')
plt.xlabel("X Axis",fontsize=12,color='blue') plt.ylabel("Y
Axis",fontsize=12,color='blue')
plt.xticks(fontsize=15,color='green')
plt.yticks(fontsize=15,color='green')
plt.plot(x,y1,'--',label="Line Plot 1",color="purple",lw=1)
plt.plot(x,y2,'--',label="Line Plot 2",color="red",lw=1)
plt.legend(loc=2,fontsize=12)
plt.grid()
plt.show()
```

Output:

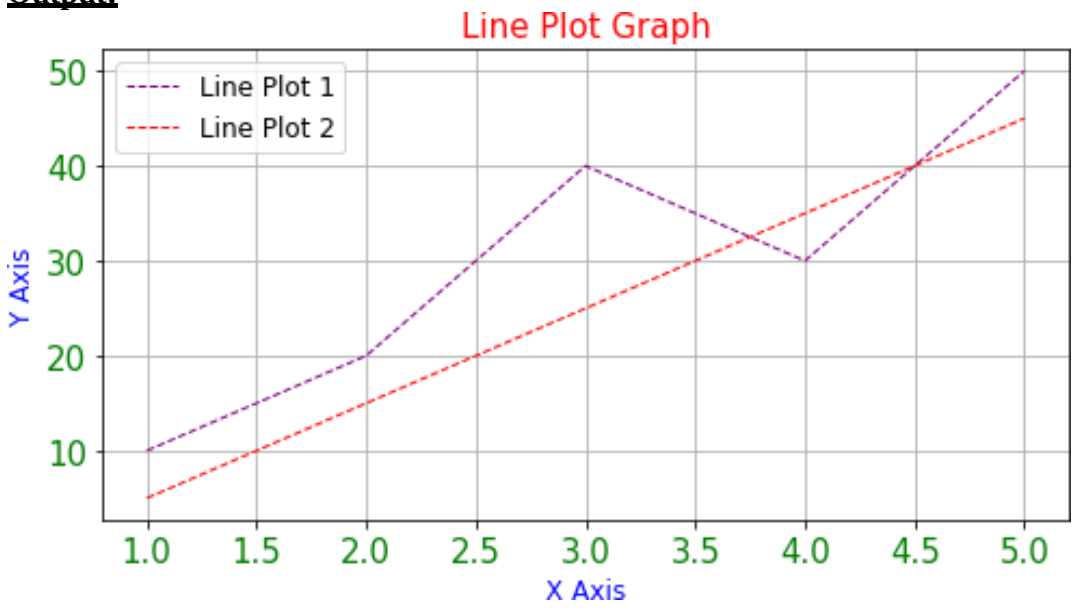


Figure 7: Line Plot with two lines in one diagram

```
#### Sub Plot
x = [1,2,3,4,5]
y1    =    [10,20,40,30,50]
y2    =    [5,15,25,35,45]
y3    =    [54,154,254,354,445]
y4    =    [25,215,225,325,425]
plt.figure(figsize=(10,8))
```

```
plt.subplot(2,2,1)
plt.plot(x,y1,label="Line                Plot
1",color="purple",lw=1) plt.title("Line   Plot
Graph             1",fontsize=15,color='red')
plt.subplot(2,2,2)
plt.plot(x,y2,label="Line Plot
2",color="purple",lw=1) plt.title("Line Plot
Graph 2",fontsize=15,color='blue')
plt.subplot(2,2,3)
plt.plot(x,y3,label="Line Plot
3",color="purple",lw=1) plt.title("Line Plot
Graph 3",fontsize=15,color='green')
plt.subplot(2,2,4)
plt.plot(x,y4,label="Line Plot
4",color="purple",lw=1) plt.title("Line Plot
Graph 4",fontsize=15,color='purple')
plt.savefig("graph.png")
plt.show()
```

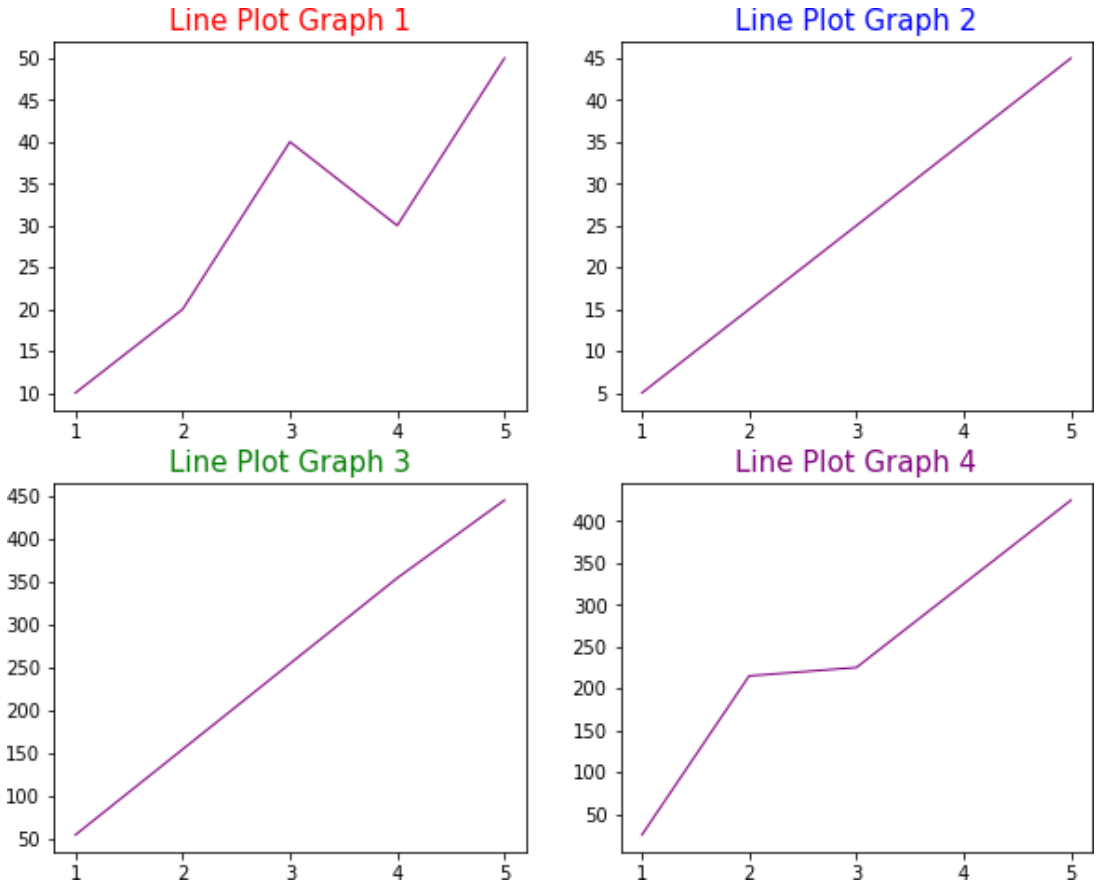


Figure 8: Sub plot feature with four different Line Plot graph

Support Vector Machine: A Non-Linear Machine Learning Technique

Amit Saha[1], K. N. Singh[2], Mrinmoy Ray[2] and Santosha Rathod[3]
[1]Central Silk Board, Ministry of Textiles, Government of India
[2]ICAR-IASRI, New Delhi
[3]ICAR-IIRR, Hyderabad
amits.csb@gov.in

1. Introduction:

Machine learning is a technique which allows the machine to learn by itself. Support Vector Machine (SVM) is one of the eminent supervised machine learning technique which was developed by Cortex and Vapnik (1995) for binary classification problems. In binary classification, the goal of the SVM is to find out a hyperplane that best separates a dataset into two classes. After two years of SVM's invention, support vector regression (SVR) based on similar principles as SVM classification was developed by Vapnik *et al*. (1997) to deal with the regression problems. Being a non-parametric method, SVR does not depend on assumptions like linear regression. Another benefit of using SVR is that it permits the construction of non-linear model. So, SVM is not only popular for the classification but also for its modelling and prediction ability. The performance of SVM is based upon proper selection of kernel. There are different types of kernel which can be used for the classification and prediction purposes. Since the last decade, the application of SVM has been extended to time series modelling and forecasting in various areas such as power load forecasting (Niu *et al*., 2010), rainfall forecasting (Ortiz-Garcia et al., 2014), wind power forecasting (De Giorgi *et al*., 2014) and agricultural forecasting (Kumar and Prajneshu, 2015).

2. Support Vector Machine (SVM) in time series:

Application of SVM in time series is generally utilized when the series shows non stationarity and non-linearity process. A tremendous advantage of SVM is that it is not model dependent as well as independent of stationarity and linearity. However, it may be computationally expensive during the training. The training of the data driven prediction process SVM is done by a function which is estimated utilizing the observed data. Let, a time series $y(t)$ which takes the data at time t $\{t = 0,1,2,3, \dots, N\}$.

Now, the prediction function for linear regression is defined as:

$$f(y) = (w.y) + c \tag{1}$$

Whereas, for non linear regression, it will be:

$$f(y) = \big(w.\emptyset(y)\big) + c \tag{2}$$

Where, w dentoes the weights, c represents threshold value and $\emptyset(y)$ is known as kernel function.

If the observed data is linear, then equation (1) will be used. But, for non-linear data, the mapping of $y(t)$ is done to the higher dimension feature space through some function which is denoted as $\emptyset(y)$ and eventually it is transformed into the linear process. Afer that, a linear regression will carry out in that feature space.

The first and foremost objective is to find out the value of w and c which will be optimal. In SVM, there are two things viz., flatness of weights and error after the estimation which are to be minimized. The flatness of the weights is denoted by $\|w\|^2$ which is the eucledian norm. Firstly, one has to concentrate on minimization the $\|w\|^2$. Second important thing is the minimization of the error. This is also called as empirical risk. However, the overall aim is to minimize the regularized risk which is sum of empirical risk and the half of the product of the flatness of weight and a constant term which is known as regularized constant. The regularized risk can be written as-

$$R_{reg}(f) = R_{emp}(f) + \frac{\tau}{2}\|w\|^2 \tag{3}$$

Where, $R_{reg}(f)$ is the regularized risk, $R_{emp}(f)$ denotes the empirical risk, τ is as constant which is called as regularized constant/capacity control term and $\|w\|^2$ is the flatness of weights.

The regularization constant has a significant impact on a better fitting of the data and it can also be useful for the minimization of bad generalization effects. In the other words, this constant deals with the problem of over-fitting. The overfitting of the data can be redued by the proper selection of this constant value. The empirical risk can be defined as:-

$$R_{emp}(f) = \frac{1}{N}\sum_{i=0}^{N-1} L\big(y(i), \alpha(i), f(y(i), w)\big) \tag{4}$$

Where, $\alpha(i)$ denotes the truth data of predicted value, $L(.)$ is known as loss function and i represents the index to the time series.

There are various types of loss function in literature. But, two functions viz., vapnik loss function and quadratic loss function are most popular and they are generally used. The quadratic programming problem has been made to minimize the regularised risk which is-

$$\text{Minimize, } \frac{1}{2}\|w\|^2 + D\sum_{i=1}^{n} L\big(\alpha(i), f(y(i), w)\big) \tag{5}$$

Where,

$$L\big(\alpha(i), f(y(i), w)\big) = |\alpha(i) - f(y(i), w)| - \in \text{ if } |\alpha(i) - f(y(i), w)| \geq \in$$

$$= 0; \text{ otherwise.}$$

Where, D is a constant which equals to the summation normalization factor and $\in$ represents the size of the tube.

The computation of $\in$ and D is done empirically because they are user defined. On has to choose proper value of D and $\in$. Now, dual optimization problem is formed using the lagrange multiplier which can be written as:

$$\text{Maximize, } -\frac{1}{2}\sum_{i,j=1}^{N}(\beta_i - \beta_i^*)\left(\beta_j - \beta_j^*\right)\langle y(i), y(j)\rangle - \in \sum_{i=1}^{N}(\beta_i - \beta_i^*) + \sum_{i=1}^{N}\alpha(i)\,(\beta_i - \beta_i^*) \tag{6}$$

$$\text{Subject to, } \sum_{i-1}^{N}(\beta_i - \beta_i^*) = 0 \; ; \beta_i, \beta_i^* \in [0, D]$$

The function $f(x)$ is defined as;

$$f(x) = \sum_{i=1}^{N}(\beta_i - \beta_i^*)\,\langle y, y(i)\rangle + C \tag{7}$$

KKT conditions are used to get the solution of the weights.

The significance of kernel function in non-linear support vector machine (NLSVR) is very much imporatnt for mapping the data $y(i)$ into higher dimension feature space $\emptyset(\, y(i))$ in which the data becomes linear. Generally notation for kernel function is given as;

$$k(y, y') = \langle \emptyset(y), \emptyset(\, y') \rangle; \tag{8}$$

There are many methods in literature to solve the quadartic programming. However, the most used method is sequential minimization optimization (SMO) algorithm.

3. Kernel function

SVM is a learning algorithm which is based on kernel. There are different types of kernel which can be used for the classification and prediction purpose. However, there is no such rule to make inference on which kernel should one use. All the kernels are used separately for the given datasets and whichever gives the better result, one should choose that one. Various types Kernel are listed below:

1. Non linear
2. Linear
3. Polynomial
4. Radial basis function: a) Gaussian Radial basis function b) Laplace Radial basis function.
5. Sigmoid kernel
6. Hyperbolic tangent kernel
7. Anova radial basis kernel
8. Multi-layer perceptron
9. Linear spline kernel.

Kernel function are used for the transformation of the given data into the required form. Kernel function is actually a mathematical function. RBF is mostly used kernel function. Some kernel functions are described in the following:

Polynomial kernel equation: Polynomial kernel is generally used in the image processing. It is useful for nonlinear modelling. This kernel function is very simple yet efficient method.

$$k(x, y) = (x.y + 1)^p \text{ ; } p\text{= degree of polynomial} \tag{9}$$

Gaussian kernel function:

$$k(x, y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{10}$$

Or $k(x, y) = exp(-\alpha\|x - y\|^2)$, Where, shape of hyperplane is controlled by σ.

Sigmoid kernel function:

Sigmoid function is used as the proxy of artificial neural network.

$$k(x, y) = tanh(\theta x^T.y + a) \tag{11}$$

Linear kernel function:

Sometimes, linear kernel gives better results as compared to complex and nonlinear kernels. Linear

classifier can be used to test the non-linearity of the datasets.

$$k(x, y) = x.y \tag{12}$$

4. Advantages of SVM:

1. It gives global optimum.
2. Training of SVM is comparatively easier than other machine learning techniques.
3. Well scaling for data with high dimensionality.
4. It can give a good prediction.
5. It is based on statistical learning theory.
6. Work on structural risk minimization.
7. Risk of overfitting problem may overcome by SVM.
8. It has good generalization property.
9. It is useful when there is no prior information about the data.
10. It also work on unstructured data.

5. Illustration:

Data Description:

Time series data on Cotton Production (Million Bales) of India from 1950 to 2016 were taken from the Ministry of Agriculture & Farmers Welfare, Government of India. The data from 1950-2011 have been utilized for model building purpose and the data from 2012 to 2016 were used to predict the cotton production for the validation purpose.

Support Vector Machine:

The most important part in SVM technique is the selection of parameters and kernel which have to be selected with utmost care to improve the performance of the model in order to get better accuracy in forecasting. The best parameters and kernel have been selected using "e1701" package (David, 2017) in R software.

The time series plot of cotton production is illustrated in Fig. 1. It can be seen from Table 1 that the time series show a high value of coefficient variation which reprsents the presence of highly heterogenous characteristic of the series.

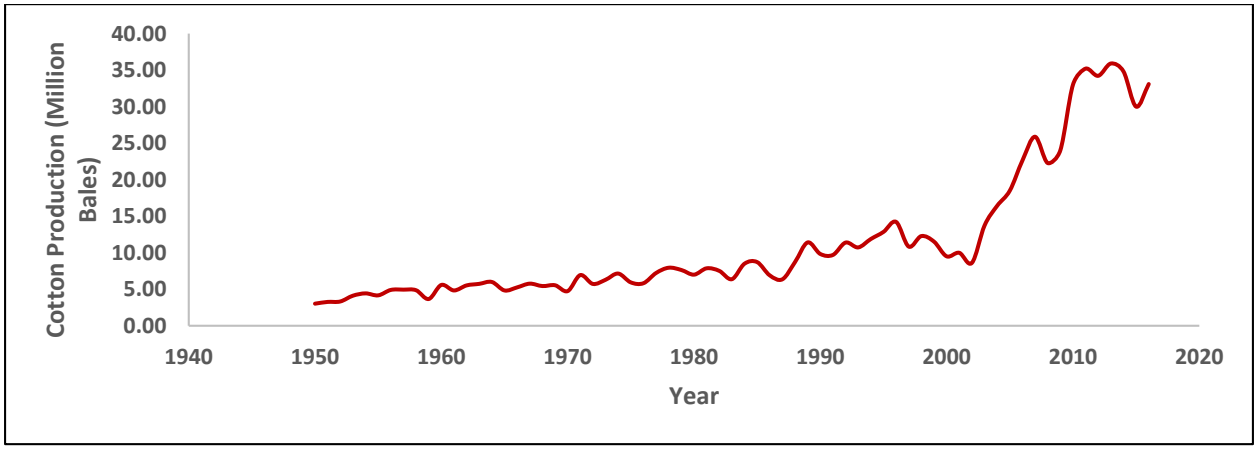


Fig. 1: Time Series Plot of Cotton Production

Table 1: Summary Statistics of Cotton Production

Statistic	Value	Statistic	Value
Minimum	3.04	Maximum	33.20
1st Quartile	5.54	Standard Deviation	6.81
Median	7.20	Skewness	2.05
Mean	9.60	Kurtosis	4.09
3rd Quartile	11.26	Coefficient of Variation	70.93

Table 2 displays the estimated best parameters of SVR after sufficient tuning of SVR model and these best parameters have been utilized to build the SVR model. It has been seen that the best SVM-kernel function is Radial basis function for SVR.

Table 2: Parameter estimation of SVR

Sampling method	10-fold cross validation
Epsilon (Best Parameter)	0.1
Cost (Best Parameter)	4
Gamma (Best Parameter)	1
Number of Support Vectors	39
SVM-Type	eps-regression
SVM-Kernel	Radial Basis Function

Fig. 2 shows the graphical representation of the performance of the models for Cotton Production series. Model performance in terms of MSE, MAE and MAPE has been shown in Table 3 and Table 4 for training and testing dataset respectively. Here, ARIMA (2, 2, 1) model has been fitted based on the lowest AIC values among various ARIMA models and the data of cotton production show the non-linearity pattern which is tested by Brock, Dechert and Scheinkman (BDS) test.

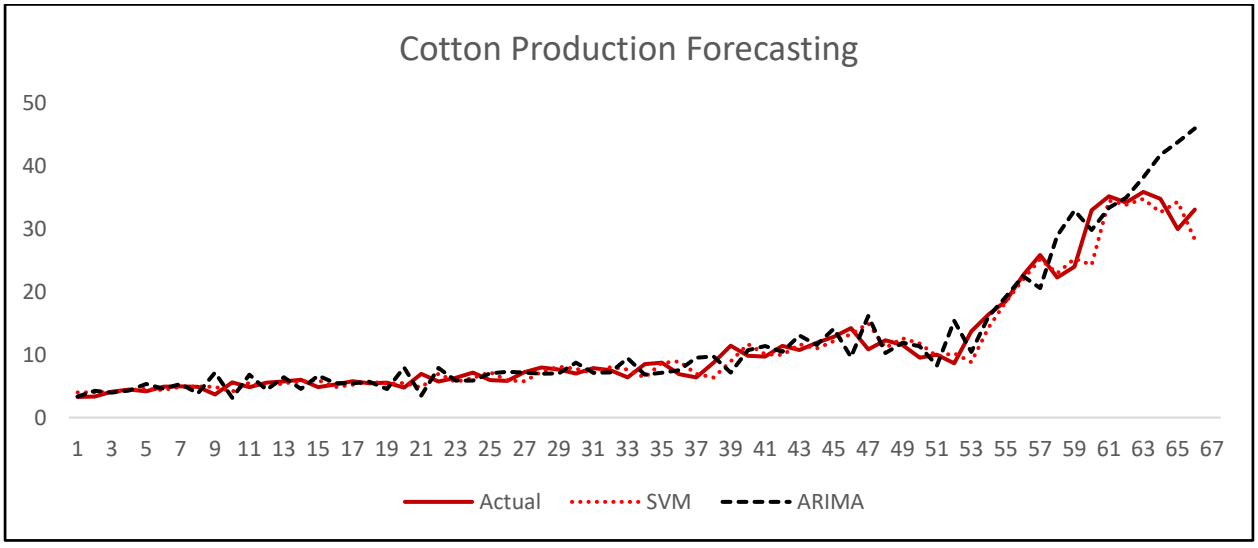


Fig. 2: Graphical representation of the performance of ARIMA and SVM models

Table 3: Model performance in training dataset using ARIMA and SVM

Model	MSE	MAE	MAPE
ARIMA	6.70	1.83	21.28
SVM	3.08	1.14	12.73

Table 4: Model performance in testing dataset using ARIMA and SVM

Model	MSE	MAE	MAPE
ARIMA	82.45	7.35	22.76
SVM	9.48	2.54	7.83

Table 5 displays the Out-of-Sample forecast values using ARIMA and SVM.

Table 5: Model performance in testing dataset using ARIMA and SVM

Year	Actual	ARIMA	SVM
2012	34.22	34.98	33.85
2013	35.9	38.21	34.78
2014	34.81	41.79	32.67
2015	30.0	43.82	34.33
2016	33.09	45.99	28.32

It has been seen from the Fig. 2 that the fitted graph of the SVM model is more close to the graph of original data as compare to ARIMA model both in training and forecasting. It is observed from Table 3 and Table 4 that the SVM has a lower MSE, MAE and MAPE compared to the ARIMA model in both training and testing dataset. It has also been seen from Table 5 that the forecasted values of the SVM are closer to the observed values compared to ARIMA. From the above results and discussion, it can be inferred that performance of the SVM model is better than the ARIMA model in terms of forecasting accuracy.

6. Conclusion

In reality, most of the time series data are non-linear in nature. In this study, the data of cotton production show non-stationary as well as non-linearity structure which were difficult to capture for the ARIMA models. However, SVM has shown its' tremendous performance due to the ability of capturing the non-linear pattern. Being a non-linear machine learning technique, SVM has well captured the heterogeneous trend of the given dataset. Based on the results, it can be inferred that SVM outperformed the ARIMA model. Therefore, it can be used in modeling and forecasting of time series to improve the forecasting accuracy in the presence of non-linear pattern.

7. Bibliography

Cortes, C. and Vapnik, V. (1995). Support-vector network. *Machine Learning*, 20, 1-25.

David, M. (2017). E1071: Misc Functions of the Department of Statistics. *Probability Theory Group R package version*, 1: 6–8.

De Giorgi, M.G., Campilongo, S., Ficarella, A. and Congedo, P.M. (2014). Comparison between wind power rediction models based on wavelet decomposition with least-squares support vector machine (LS-SVM) and artificial neural network (ANN). *Energies*, 7:5251-5272.

Kumar, T.L.M. and Prajneshu (2015). Development of Hybrid Models for Forecasting Time-Series Data Using Nonlinear SVR Enhanced by PSO. *Journal of Statistical Theory and Practice*, 9(4), 699-711.

Niu, D., Wang, Y. and Wu, D.D. (2010). Power load forecasting using support vector machine and ant colony optimization. *Expert Syst Appl*, 37:2531–2539.

Ortiz-Garcia, E.G., Salcedo-Sanz, S. and Casanova-Mateom, C. (2014). Accurate precipitation prediction with support vector classifiers: A study including novel predictive variables and observational data. *Atmos Res*, 139:128–136.

Vapnik, V., Golowich, S., and Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing, In Mozer, M., Jordan, M and Petsche, T. (Eds). *Advances in Neural Information Processing Systems*, 9:281-287, Cambridge, MA, MIT Press.

Foundations of Neural Networks Basics and Artificial Neural Networks Concepts

Anshu Bharadwaj
ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012

Introduction

The inspiration for artificial neural networks (ANN), or simply neural networks, resulted from the admiration for how the human brain computes complex processes, which is entirely different from the way conventional digital computers do this. The power of the human brain is superior to many information-processing systems, since it can perform highly complex, nonlinear, and parallel processing by organizing its structural constituents (neurons) to perform such tasks as accurate predictions, pattern recognition, perception, motor control, etc. It is also many times faster than the fastest digital computer in existence today. An example is the sophisticated functioning of the information-processing task called human vision. This system helps us to understand and capture the key components of the environment and supplies us with the information we need to interact with the environment. That is, the brain very often performs perceptual recognition tasks (e.g., voice recognition embedded in a complex scene) in around 100–200 ms, whereas less complex tasks many times take longer even on a powerful computer (Haykin 2009).

In general, the functioning of the brains of humans and other animals is intriguing because they are able to perform very complex tasks in a very short time and with high efficiency. For example, signals from sensors in the body convey information related to sight, hearing, taste, smell, touch, balance, temperature, pain, etc. Then the brain's neurons, which are autonomous units, transmit, process, and store this information so that we can respond successfully to external and internal stimuli (Dougherty 2013). The neurons of many animals transmit spikes of electrical activity through a long, thin strand called an axon. An axon is divided into thousands of terminals or branches, where depending on the size of the signal they synapse to dendrites of other neurons (Fig. 1). It is estimated that the brain is composed of around 10^{11} neurons that work in parallel, since the processing done by the neurons and the memory captured by the synapses are distributed together over the network. The amount of information processed and stored depends on the threshold firing levels and also on the weight given by each neuron to each of its inputs (Dougherty 2013).

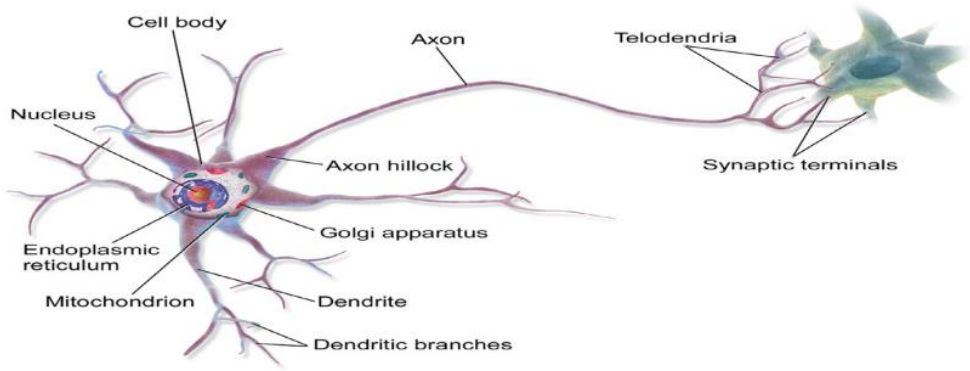


Figure 1: Graphical representation of Biological Neural Network

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

One of the characteristics of biological neurons, to which they owe their great capacity to process and perform highly complex tasks, is that they are highly connected to other neurons from which they receive stimuli from an event as it occurs, or hundreds of electrical signals with the information learned. When it reaches the body of the neuron, this information affects its behavior and can also affect a neighboring neuron or muscle (Francisco-Caicedo and López-Sotelo 2009). Francisco-Caicedo and López-Sotelo (2009) also point out that the communication between neurons goes through the so-called synapses. A synapse is a space that is occupied by chemicals called neurotransmitters. These neurotransmitters are responsible for blocking or passing on signals that come from other neurons. The neurons receive electrical signals from other neurons with which they are in contact. These signals accumulate in the body of the neuron and determine what to do. If the total electrical signal received by the neuron is sufficiently large, the action potential can be overcome, which allows the neuron to be activated or, on the contrary, to remain inactive. When a neuron is activated, it is able to transmit an electrical impulse to the neurons with which it is in contact. This new impulse, for example, acts as an input to other neurons or as a stimulus in some muscles (Francisco-Caicedo and López-Sotelo 2009). The architecture of biological neural networks is still the subject of active research, but some parts of the brain have been mapped, and it seems that neurons are often organized in consecutive layers, as shown in Fig. 2.

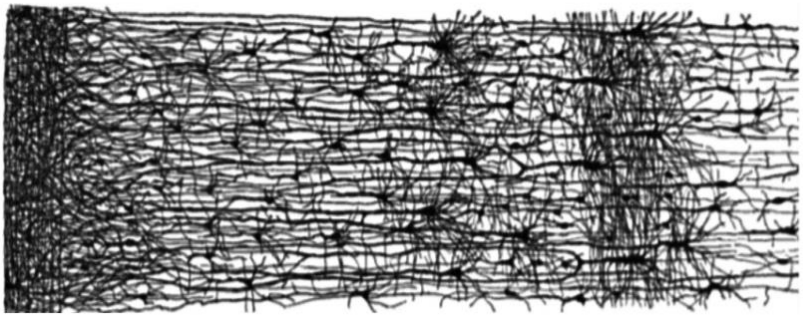


Figure 2: Multiple layers in a Biological Neural Network in a Human Cortex

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

ANN are machines designed to perform specific tasks by imitating how the human brain works, and build a neural network made up of hundreds or even thousands of artificial neurons or processing units. The artificial neural network is implemented by developing a computational learning algorithm that does not need to program all the rules since it is able to build up its own rules of behavior through what we usually refer to as “experience.” The practical implementation of neural networks is possible due to the fact that they are massively parallel computing systems made up of a huge number of basic processing units (neurons) that are interconnected and learn from their environment, and the synaptic weights capture and store the strengths of the interconnected neurons. The job of the learning algorithm consists of modifying the synaptic weights of the network in a sequential and supervised way to reach a specific objective (Haykin 2009). There is evidence that neurons working together are able to learn complex linear and nonlinear input–output relationships by using sequential training procedures. It is important to point out that even though the inspiration for these models was quite different from what inspired statistical models, the building blocks of both types of models are quite similar. Anderson et al. (1990) and Ripley (1993) pointed out that neural networks are simply no more than *generalized nonlinear statistical models*. Anderson et al.

(1990) pointed out that "ANN are statistics for amateurs since most neural networks conceal the statistics from the user."

The Building Blocks of Artificial Neural Networks

To get a clear idea of the main elements used to construct ANN models, Fig.3 provides a general artificial neural network model that contains the main components for this type of models.

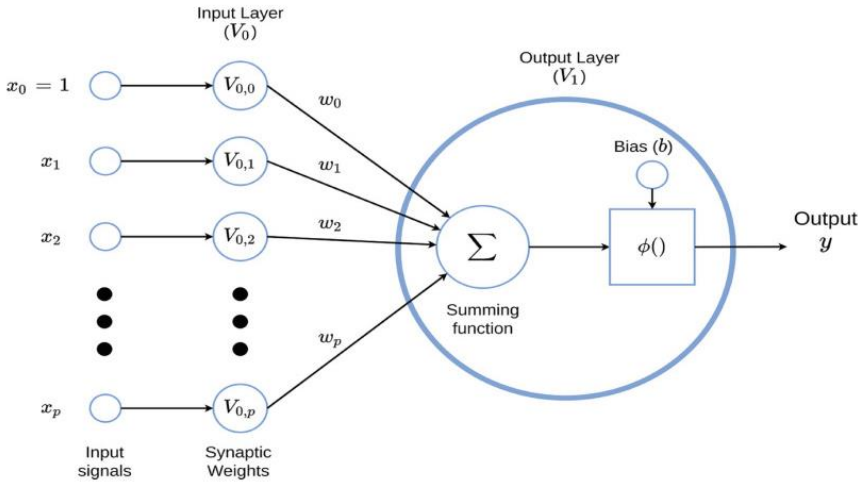


Figure 3: General Artificial Neural Network

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

$x_1, \ldots, x_p$ represents the information (input) that the neuron receives from the external sensory system or from other neurons with which it has a connection. $\boldsymbol{w} = (w_1, \ldots, w_p)$ is the vector of synaptic weights that modifies the received information emulating the synapse between the biological neurons. These can be interpreted as gains that can attenuate or amplify the values that they wish to propagate toward the neuron. Parameter b_j is known as the bias (intercept or threshold) of a neuron. Here in ANN, learning refers to the method of modifying the weights of connections between the nodes (neurons) of a specified network.

The different values that the neuron receives are modified by the synaptic weights, which then are added together to produce what is called the *net input*. In mathematical notation, that is equal to

$$v_j = \sum_{j=1}^{p} \omega_{ij} x_j$$

This net input (v_j) is what determines whether the neuron is activated or not. The activation of the neuron depends on what we call the *activation function*. The net input is evaluated in this function and we obtain the output of the network as shown next:

$$y_j = g(v_j),$$

where *g* is the activation function. For example, if we define this function as a unit step (also called threshold), the output will be 1 if the net input is greater than zero; otherwise the output will be 0. Although there is no biological behavior indicating the presence of something similar to the neurons of the brain, the use of the activation function is an artifice that allows applying ANN to a great diversity of real problems. According to what has been mentioned, output y_j of the neuron is generated when evaluating the net input (v_j) in the activation function. We can propagate the output of the neuron to other neurons or it can be the output of the network, which, according to the application, will have an interpretation for the user. In general, the job of an artificial neural network model is done by simple elements called neurons. The signals are passed between neurons through connection links. Each connection link has an associated weight, which, in a typical neuronal network, multiplies the transmitted signal. Each neuron applies an activation function (usually nonlinear) to the network inputs (sum of the heavy input signals) for determining its corresponding sign. Later in this chapter, we describe the many options for activation functions and the context in which they can be used.

A unilayer ANN like that in Fig. 3 has a low processing capacity by itself and its level of applicability is low; its true power lies in the interconnection of many ANNs, as happens in the human brain. This has motivated different researchers to propose various topologies (architectures) to connect neurons to each other in the context of ANN. Next, we provide two definitions of ANN and one definition of deep learning:

Definition 1. An artificial neural network is a system composed of many simple elements of processing which operate in parallel and whose function is determined by the structure of the network and the weight of connections, where the processing is done in each of the nodes or computing elements that has a low processing capacity (Francisco-Caicedo and López-Sotelo 2009).

Definition 2. An artificial neural network is a structure containing simple elements that are interconnected in many ways with hierarchical organization, which tries to interact with objects in the real world in the same way as the biological nervous system does (Kohonen 2000).

Perceptron

The first application of the neuron replicated a logic gate, where there were one or two binary inputs, and a boolean function that only gets activated given the right inputs and weights.

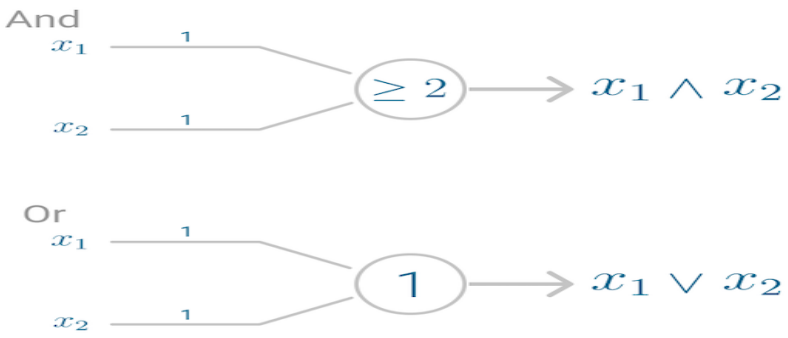


Figure 4: Example of the AND and OR logic gates

However, this model had a problem. It couldn't *learn* like the brain. The only way to get the desired output was if the weights, working as catalyst in the model, were set beforehand.

It was only a decade later that Frank Rosenblatt extended this model, and created an algorithm that could *learn* the weights in order to generate an output. Building onto McCulloch and Pitt's neuron, Rosenblatt developed the **Perceptron**.

Perceptron

Although today the **Perceptron** is widely recognized as an algorithm, it was initially intended as an image recognition machine. It gets its name from performing the human-like function of *perception*, seeing and recognizing images.

Rosenblatt's perceptron machine relied on a basic unit of computation, the **neuron**. Just like in previous models, each neuron has a cell that receives a series of pairs of inputs and weights. The major difference in Rosenblatt's model is that **inputs are combined in a weighted sum** and, if the weighted sum exceeds a predefined threshold, the neuron fires and produces an output.

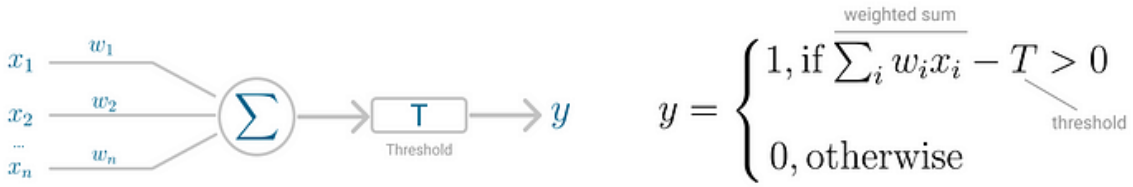


Figure 5: Perceptrons neuron model (left) and threshold logic (right).

Threshold *T* represents the **activation function**. If the weighted sum of the inputs is greater than zero the neuron outputs the value 1, otherwise the output value is zero.

Perceptron for Binary Classification

With this discrete output, controlled by the activation function, the perceptron can be used as a **binary classification model**, defining a **linear decision boundary**. It finds the separating hyperplane that minimizes the distance between misclassified points and the decision boundary.

Figure 6: Perceptron's loss function

To minimize this distance, Perceptron uses Stochastic Gradient Descent as the optimization function. If the data is linearly separable, it is guaranteed that Stochastic Gradient Descent will converge in a finite number of steps. The last piece that Perceptron needs is the **activation function**, the function that determines if the neuron will fire or not. Initial Perceptron models used sigmoid function, and just by looking at its shape, it makes a lot of sense!

The sigmoid function maps any real input to a value that is either 0 or 1, and encodes a non-linear function. The neuron can receive negative numbers as input, and it will still be able to produce an output that is either 0 or 1.

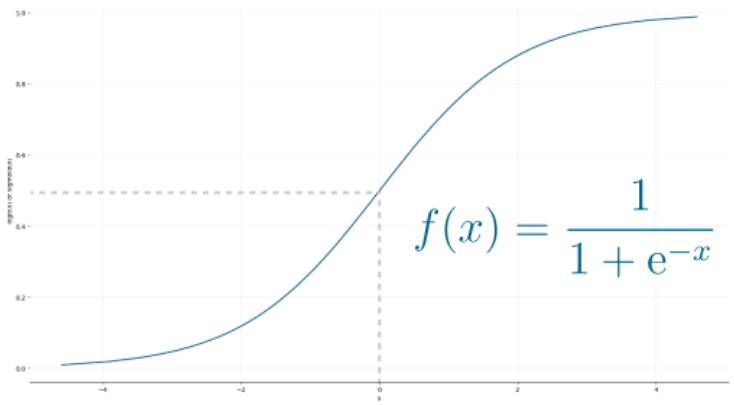


Figure 7: Sigmoid function

Putting it all together

The neuron receives inputs and picks an initial set of weights a random. These are combined in weighted sum and then ReLU, the activation function, determines the value of the output.

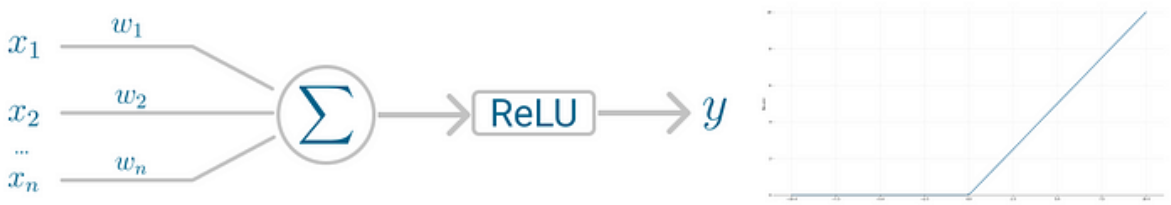


Figure 8: Perceptrons neuron model (left) and activation function (right)

But we wonder, *Doesn't Perceptron actually learn the weights?*

It does! Perceptron uses Stochastic Gradient Descent to find, or you might say *learn*, the set of weight that minimizes the distance between the misclassified points and the decision boundary.

Once Stochastic Gradient Descent converges, the dataset is separated into two regions by a linear hyperplane.

Although it was said the Perceptron could represent any circuit and logic, the biggest criticism was that it couldn't represent the XOR gate, *exclusive OR*, where the gate only returns *1* if the inputs are different. This was proved almost a decade later by Minsky and Papert, in 1969[5] and highlights the fact that Perceptron, with only one neuron, can't be applied to non-linear data.

Backpropagation

Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function. There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in MultiLayer Perceptron.

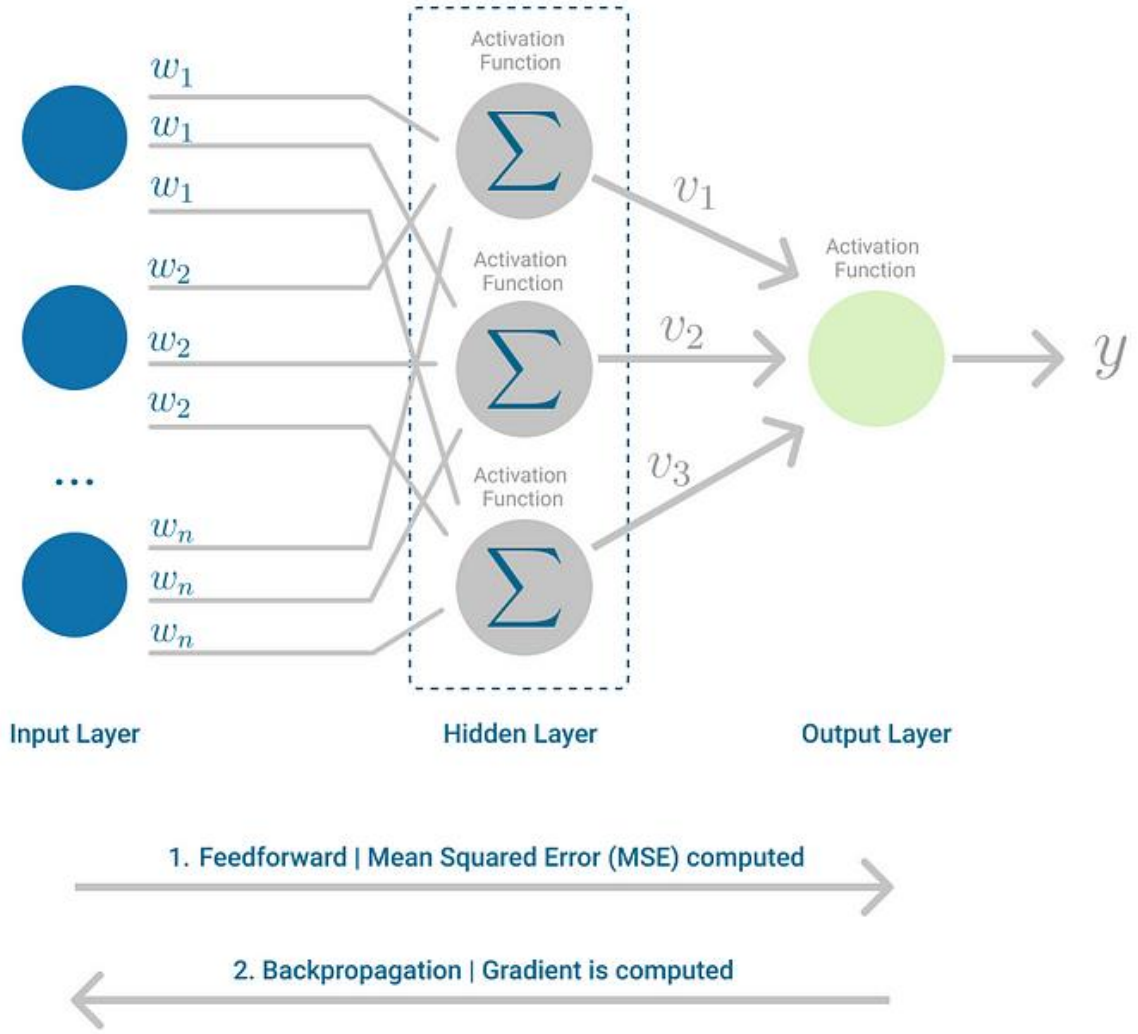


Figure 9: Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps.

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

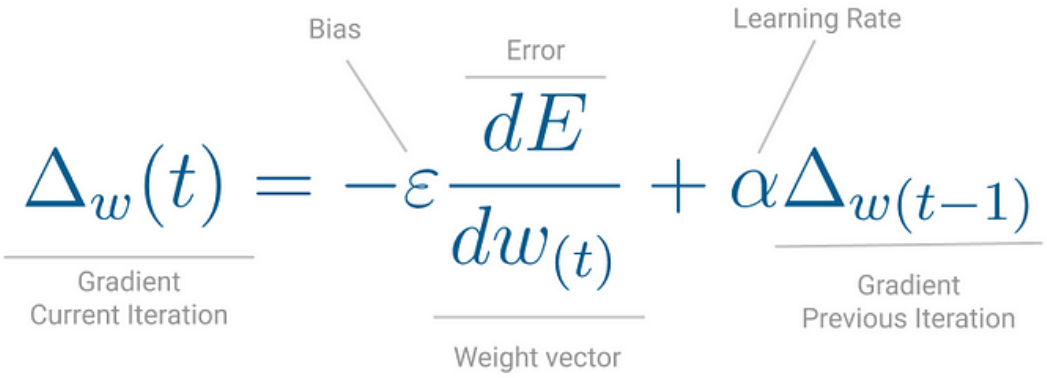


Figure 10: One iteration of Gradient Descent

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.

Multilayer perceptron (MLP)

The **Multilayer Perceptron** was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.

A Multilayer Perceptron has input and output layers, and one or more **hidden layers** with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.

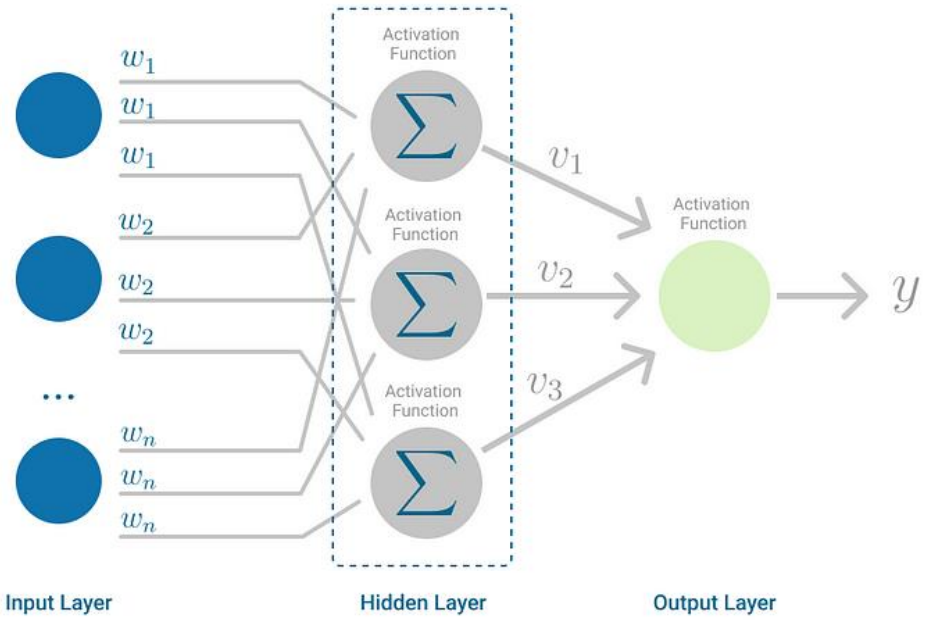


Figure 11: Multilayer Perceptron

Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer. Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer. But it has more to it.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

This is where **Backpropagation** comes into play.

Backpropagation

Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in MultiLayer Perceptron.

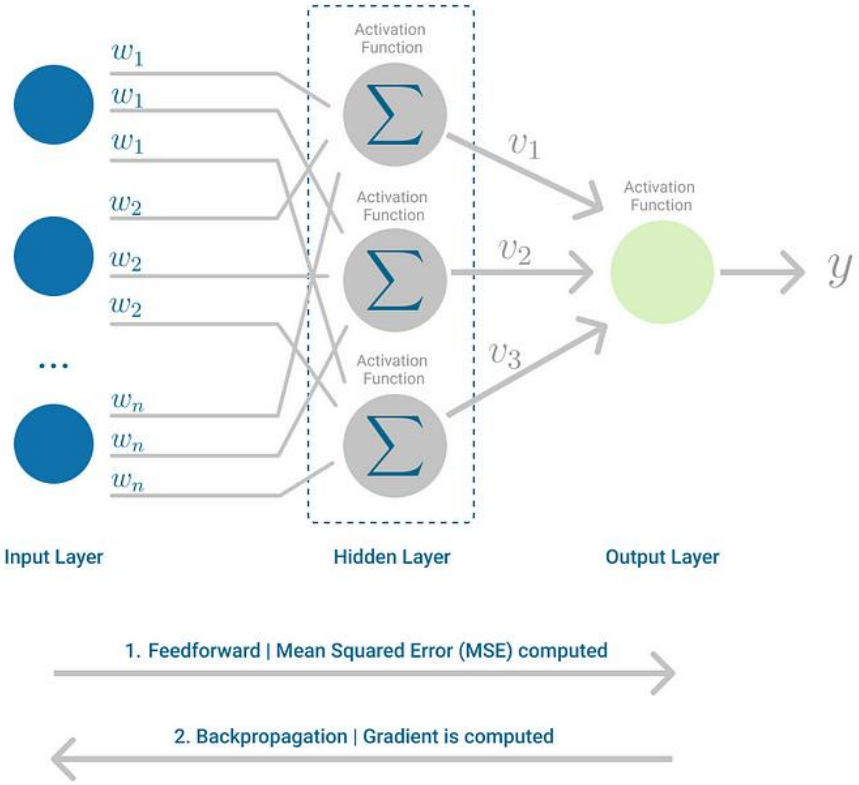


Figure 12: Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps.

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

To better understand the elements of the model depicted in Fig. 12, it is important to distinguish between the types of layers and the types of neurons; for this reason, next we will explain the type of layers and then the type of neurons in more detail.

(a) **Input layer**: It is the set of neurons that directly receives the information coming from the external sources of the network.

(b) **Hidden layers**: Consist of a set of internal neurons of the network that do not have direct contact with the outside. The number of hidden layers can be 0, 1, or more. In general, the neurons of each hidden layer share the same type of information; for this reason, they are called hidden layers. The neurons of the hidden layers can be interconnected in different ways; this determines, together with their number, the different topologies of ANN. The learned information extracted from the training data is stored and captured by the weight values of the connections between the layers of the artificial neural network.
(c) **Output layer**: It is a set of neurons that transfers the information that the network has processed to the outside (Francisco-Caicedo and López-Sotelo 2009). In Fig. 10.4 the output neurons correspond to the output variables *y*1, *y*2, *y*3, and *y*4. This means that the output layer gives the answer or prediction of the artificial neural network model based on the input from the input layer. The final output can be continuous, binary, ordinal, or count depending on the setup of the ANN which is controlled by the activation (or inverse link in the statistical

domain) function we specified on the neurons in the output layer (Patterson and Gibson 2017).

Next, we define the types of neurons: (1) *input neuron.* A neuron that receives external inputs from outside the network; (2) *output neuron.* A neuron that produces some of the outputs of the network; and (3) *hidden neuron.* A neuron that has no direct interaction with the "outside world" but only with other neurons within the network.

Activation Functions

The mapping between inputs and a hidden layer in ANN and DNN is determined by activation functions. Activation functions propagate the output of one layer's nodes forward to the next layer (up to and including the output layer). Activation functions are scalar-to-scalar functions that provide a specific output of the neuron. Activation functions allow nonlinearities to be introduced into the network's modeling capabilities (Wiley 2016). The activation function of a neuron (node) defines the functional form for how a neuron gets activated. For example, if we define a linear activation function as $g(z) = z$, in this case the value of the neuron would be the raw input, x, times the learned weight, that is, a linear model. Next, we describe the most popular activation functions.

Linear

Figure 13 shows a linear activation function that is basically the identity function. It is defined as $g(z) = Wz$, where the dependent variable has a direct, proportional relationship with the independent variable. In practical terms, it means the function passes the signal through unchanged. The problem with making activation functions linear is that this does not permit any nonlinear functional forms to be learned (Patterson and Gibson 2017).

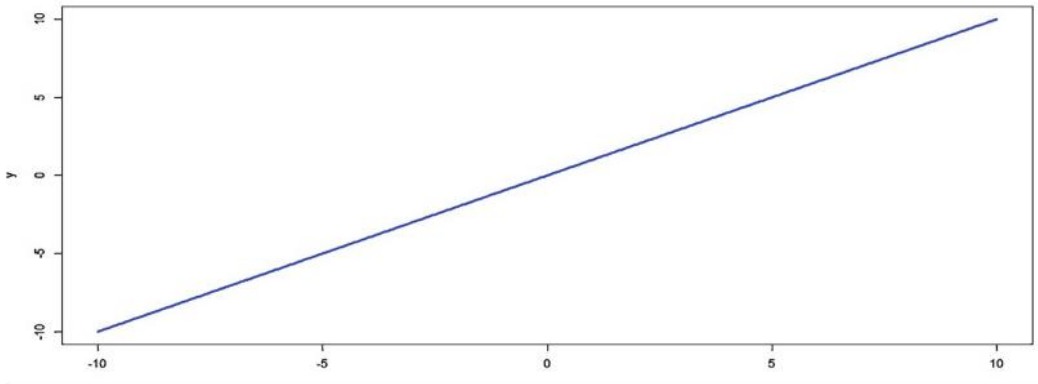


Figure 13: Linear Activation Function

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

Rectifier Linear Unit (ReLU)

The rectifier linear unit (ReLU) activation function is one of the most popular. The ReLU activation function is flat below some threshold (usually the threshold is zero) and then linear.

The ReLU activates a node only if the input is above a certain quantity. When the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable $g(z) = \max(0, z)$, as demonstrated in Fig. 14. Despite its simplicity, the ReLU activation function provides nonlinear transformation, and enough linear rectifiers can be used to approximate arbitrary nonlinear functions, unlike when only linear activation functions are used (Patterson and Gibson 2017). ReLUs are the current state of the art because they have proven to work in many different situations. Because the gradient of a ReLU is either zero or a constant, it is not easy to control the vanishing exploding gradient issue, also known as the "dying ReLU" issue. ReLU activation functions have been shown to train better in practice than sigmoid activation functions. This activation function is the most used in hidden layers and in output layers when the response variable is continuous and larger than zero.

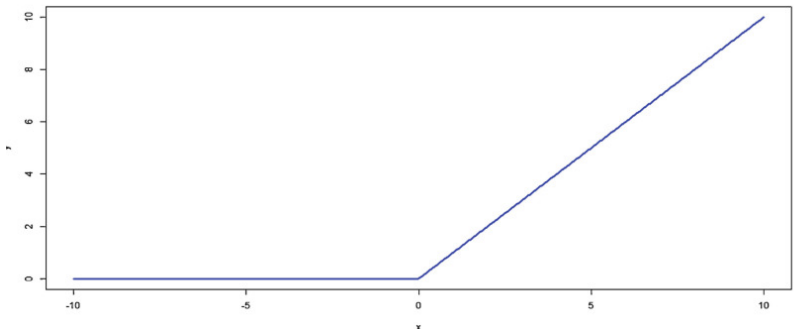


Figure 14: ReLU Activation Function

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

Sigmoid

A sigmoid activation function is a machine that converts independent variables of near infinite range into simple probabilities between 0 and 1, and most of its output will be very close to 0 or 1. Like all logistic transformations, sigmoids can reduce extreme values or outliers in data without removing them. This activation function resembles an S (Wiley 2016; Patterson and Gibson 2017) and is defined as $g(z) = (1 + e^{-z})^{-1}$. This activation function is one of the most common types of activation functions used to construct ANNs and DNNs, where the outcome is a probability or binary outcome. This activation function is a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior but has the propensity to get "stuck," i.e., the output values would be very close to 1 or 0 when the input values are strongly positive or negative (Fig. 15). By getting "stuck" we mean that the learning process is not improving due to the large or small values of the output values of this activation function.

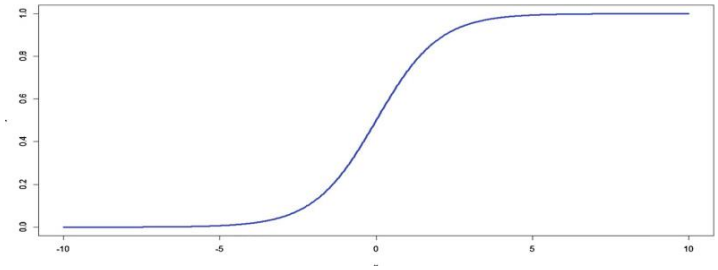


Figure 15: Sigmoid Activation Function

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

Softmax

Softmax is a generalization of the sigmoid activation function that handles multinomial labeling systems, that is, it is appropriate for categorical outcomes. Softmax is the function you will often find in the output layer of a classifier with more than two categories. The softmax activation function returns the probability distribution over mutually exclusive output classes. To further illustrate the idea of the softmax output layer and how to use it, let's consider two types of uses. If we have a multiclass modeling problem we only care about the best score across these classes, we'd use a softmax output layer with an *argmax()* function to get the highest score across all classes. For example, let us assume that our categorical response has ten classes; with this activation function we calculate a probability for each category (the sum of the ten categories is one) and we classify a particular individual in the class with the largest probability. It is important to recall that if we want to get binary classifications per output (e.g., "diseased and not diseased"), we do not want softmax as an output layer. Instead, we will use the sigmoid activation function explained before. The softmax function is defined as

$$g(z_j) = \frac{\exp(z_j)}{1 + \sum_{c=1}^{C} \exp(z_c)}, \quad j = 1, .., C$$

This activation function is a generalization of the sigmoid activation function that squeezes (force) a C dimensional vector of arbitrary real values to a C dimensional vector of real values in the range [0,1] that adds up to 1. A strong prediction would have a single entry in the vector close to 1, while the remaining entries would be close to 0. A weak prediction would have multiple possible categories (labels) that are more or less equally likely. The sigmoid and softmax activation functions are suitable for probabilistic interpretation because the output is a probabilistic distribution of the classes. This activation function is mostly recommended for output layers when the response variable is categorical.

Tanh

The hyperbolic tangent (Tanh) activation function is defined as $\tanh(z) = \sinh(z)/\cosh(z) = \exp(z) - \exp(-z)\exp(z) + \exp(-z)$. The hyperbolic tangent works well in some cases and, like the sigmoid activation function, has a sigmoidal ("S" shaped) output, with the advantage that it is less likely to get "stuck" than the sigmoid activation function since its output values are between −1 and 1, as shown in Fig. 16. For this reason, for hidden layers should be preferred the Tanh activation function. Large negative inputs to the tanh function will give negative outputs, while large positive inputs will give positive outputs (Patterson and Gibson 2017). The advantage of tanh is that it can deal more easily with negative numbers.

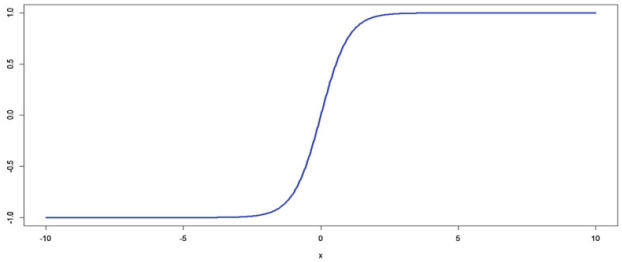


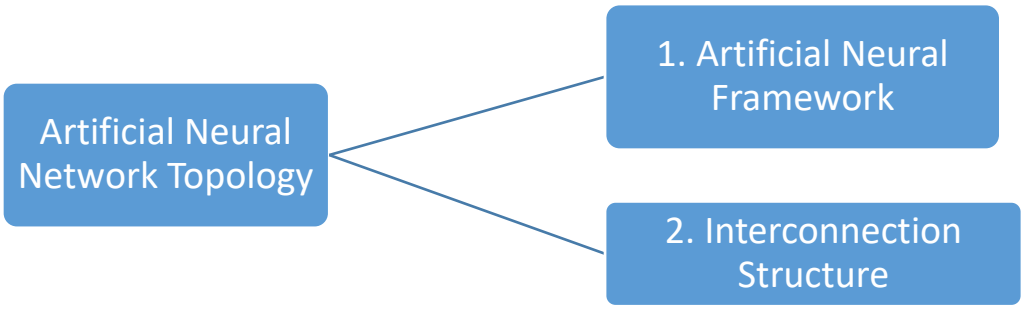
Figure 16: Tanh Activation Function

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

Artificial Neural Network Topologies

In this subsection, we describe the most popular network topologies. An artificial *neural network topology* represents the way in which neurons are connected to form a network. In other words, the neural network topology can be seen as the relationship between the neurons by means of their connections. The topology of a neural network plays a fundamental role in its functionality and performance, as illustrated throughout this chapter. The generic terms *structure* and *architecture* are used as synonyms for network topology.

the topology of a neural network consists of its *frame* or *framework* of neurons, together with its *interconnection structure* or *connectivity:*



Artificial Neural Framework

Most neural networks, including many biological ones, have a layered topology. There are a few exceptions where the network is not explicitly layered, but those can usually be interpreted as having a layered topology, for example, in some *associative memory networks*, which can be seen as one-layer neural networks where all neurons function both as input and output units. At the framework level, neurons are considered abstract entities, therefore possible differences between them are not considered. The framework of an artificial neural network can therefore be described by the number of neurons, number of layers (denoted by L), and the size of the layer, which consists of the number of neurons in each of the layers.

Interconnection structure

The interconnection structure of an artificial neural network determines the way in which the neurons are linked. Based on a layered structure, several different kinds of connections can be distinguished (see Fig. 10.11): (a) *Interlayer connection:* This connects neurons in adjacent layers whose layer indices differ by one; (b) *Intralayer connection:* This is a connection between neurons in the same layer; (c) *Self-connection:* This is a special kind of intralayer connection that connects a neuron to itself; (d) *Supralayer connection:* This is a connection between neurons that are in distinct nonadjacent layers; in other words, these connections "cross" or "jump" at least one hidden layers.

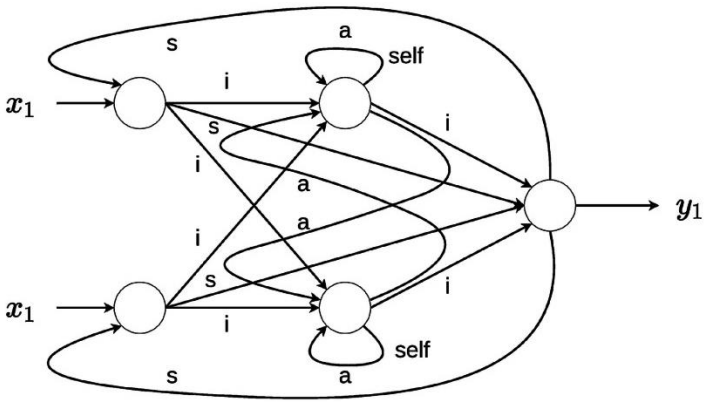


Figure 17: Network topology with two layers. (i) denotes the six interlayer connections, (s) denotes the four supralayered connections, and (a) denotes four intralayer connections of which two are self-connections

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

With each connection *(interconnection*), a *weight (strength)* is associated which is a weighting factor that reflects its importance. This weight is a scalar value (a number), which can be positive *(excitatory)* or negative *(inhibitory).* If a connection has zero weight, it is considered to be nonexistent at that point in time.

Note that the basic concept of layeredness is based on the presence of interlayer connections. In other words, every layered neural network has at least one interlayer connection between adjacent layers. If interlayer connections are absent between any two adjacent clusters in the network, a spatial reordering can be applied to the topology, after which certain connections become the interlayer connections of the transformed, layered network.

Now that we have described the two key components of an artificial neural network topology, we will present two of the most commonly used topologies.

Feedforward network

In this type of artificial neural network, the information flows in a single direction from the input neurons to the processing layer or layers (only interlayer connections) for monolayer and multilayer networks, respectively, until reaching the output layer of the neural network. This means that there are no connections between neurons in the same layer (no intralayer), and there are no connections that transmit data from a higher layer to a lower layer, that is, no supralayer connections (Fig. 19). This type of network is simple to analyze, but is not restricted to only one hidden layer.

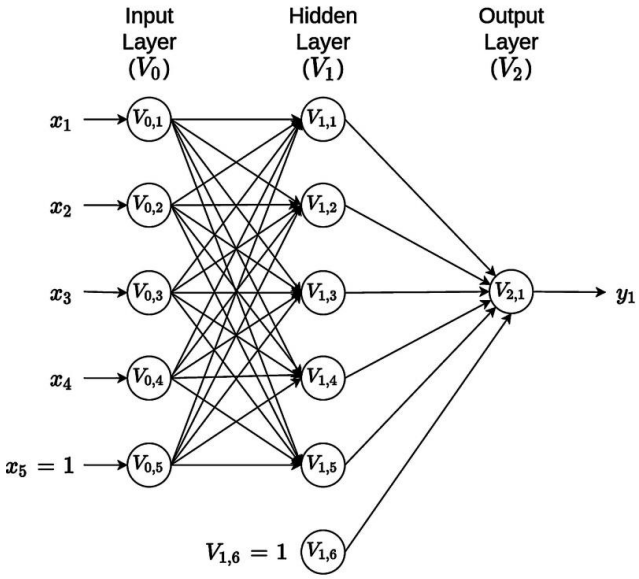


Figure 19: A simple two layered feed forward artificial neural network

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

Recurrent Networks

In this type of neural network, information does not always flow in one direction, since it can feed back into previous layers through synaptic connections. This type of neural network can be monolayer or multilayer. In this network, all the neurons have (1) incoming connections emanating from all the neurons in the previous layer, (2) ongoing connections leading to all the neurons in the subsequent layer, and (3) recurrent connections that propagate information between neurons of the same layer. Recurrent neural networks (RNNs) are different from a feedforward neural network in that they have at least one feedback loop since the signals travel in both directions. This type of network is frequently used in time series prediction since short-term memory, or delay, increases the power of recurrent networks immensely. In this case, we present an example of a recurrent two-layer neural network. The output of each neuron is passed through a delay unit and then taken to all the neurons, except itself. It is seen that only one input variable is presented to the input units, the feedforward flow is computed, and the outputs are fed back as auxiliary inputs. This leads to a different set of hidden unit activations, new output activations, and so on. Ultimately, the activations stabilize, and the final output values are used for predictions.

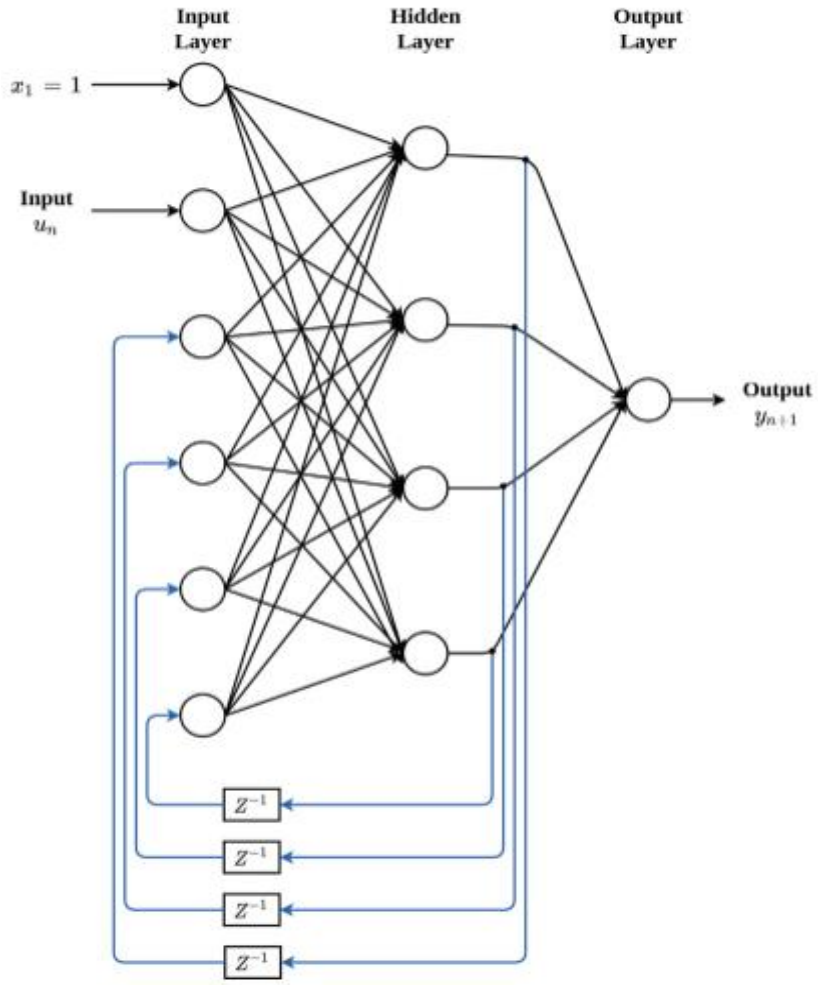


Figure 20: A simple two-layer recurrent artificial neural network with univariate output

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

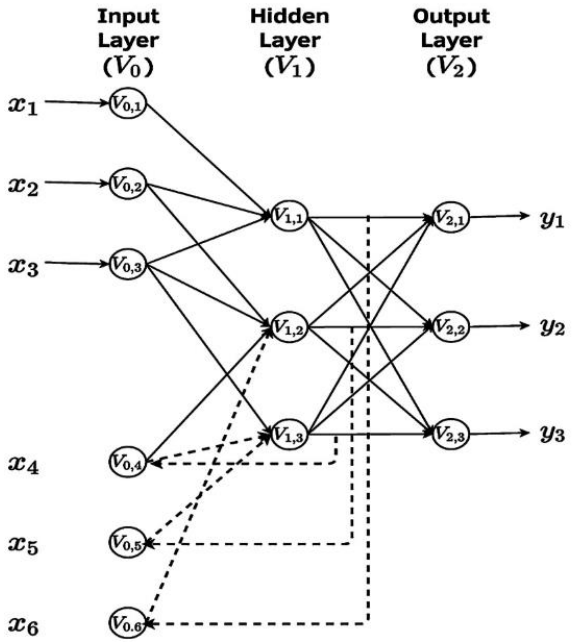


Figure 21: A two-layer recurrent artificial neural network with multivariate outputs

Source: https://www.ncbi.nlm.nih.gov/books/NBK583971/

However, it is important to point out out that despite the just mentioned virtues of recurrent artificial neural networks, they are still largely theoretical and produce mixed results (good and bad) in real applications. On the other hand, the feedforward networks are the most popular since they are successfully implemented in all areas of domain; the multilayer perceptron (MLP; that is, onother name give to feedforward networks) is the de facto standard artificial neural network topology (Lantz 2015).

Successful Applications of ANN and DL

The success of ANN and DL is due to remarkable results on perceptual problems such as seeing and hearing—problems involving skills that seem natural and intuitive to humans but have long been elusive for machines. Some of these successful applications:

(a) Near-human-level image classification, speech recognition, handwriting transcription, autonomous driving (Chollet and Allaire 2017)

(b) Automatic translation of text and images (LeCun et al. 2015)

(c) Improved text-to-speech conversion (Chollet and Allaire 2017)

(d) Digital assistants such as Google Now and Amazon Alexa

(e) Improved ad targeting, as used by Google, Baidu, and Bing

(f) Improved search results on the Web (Chollet and Allaire 2017)

(g)Ability to answer natural language questions (Goldberg 2016)

(h)In games like chess, Jeopardy, GO, and poker (Makridakis et al. 2018)

(i)Self-driving cars (Liu et al. 2017),

(j)Voice search and voice-activated intelligent assistants (LeCun et al. 2015)

(k)Automatically adding sound to silent movies (Chollet and Allaire 2017)

(l)Energy market price forecasting (Weron 2014)

(m)Image recognition (LeCun et al. 2015)

(n)Prediction of time series (Dingli and Fournier 2017)

(o)Predicting breast, brain (Cole et al. 2017), or skin cancer

(p)Automatic image captioning (Chollet and Allaire 2017)

(q)Predicting earthquakes (Rouet-Leduc et al. 2017)

(r)Genomic prediction (Montesinos-López et al. 2018a, b)

References

- Alipanahi B, Delong A, Weirauch MT, Frey BJ (2015) Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotechnol 33:831–838.
- Anderson J, Pellionisz A, Rosenfeld E (1990) Neurocomputing 2: directions for research. MIT, Cambridge.
- Angermueller C, Pärnamaa T, Parts L, Stegle O (2016) Deep learning for computational biology. Mol Syst Biol 12(878):1–16.
- Chollet F, Allaire JJ (2017) Deep learning with R. Manning Publications, Manning Early Access Program (MEA), 1st edn.
- Cole JH, Rudra PK, Poudel DT, Matthan WA, Caan CS, Tim D, Spector GM (2017) Predicting brain age with deep learning from raw imaging data results in a reliable and heritable biomarker. NeuroImage 163(1):115–124. https://doi.org/10.1016/j .neuroimage.2017.07.059.
- Cybenko G (1989) Approximations by superpositions of sigmoidal functions. Math Control Signal Syst 2:303–314.
- Dingli A, Fournier KS (2017) Financial time series forecasting—a deep learning approach. Int J Mach Learn Comput 7(5):118–122.
- Dougherty G (2013) Pattern recognition and classification-an introduction. Springer Science + Business Media, New York.
- Efron B, Hastie T (2016) Computer age statistical inference. Algorithms, evidence, and data science. Cambridge University Press, New York.
- Francisco-Caicedo EF, López-Sotelo JA (2009) Una approximación práctica a las redes neuronales artificiales. Universidad del Valle, Cali.
- Goldberg Y (2016) A primer on neural network models for natural language processing. J Artif Intell Res 57(345):420.
- Haykin S (2009) Neural networks and learning machines, 3rd edn. Pearson Prentice Hall, New York.
- Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257. [CrossRef]
- James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning with applications in R. Springer, New York.
- Kohonen T (2000) Self-organizing maps. Springer, Berlin.
- Lantz B (2015) Machine learning with R, 2nd edn. Packt Publishing Ltd, Birmingham.
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444.
- Lewis ND (2016) Deep learning made easy with R. A gentle introduction for data science. CreateSpace Independent Publishing Platform.
- Liu S, Tang J, Zhang Z, Gaudiot JL (2017) CAAD: computer architecture for autonomous driving. ariv preprint ariv:1702.01894.
- Ma W, Qiu Z, Song J, Cheng Q, Ma C (2017) DeepGS: predicting phenotypes from genotypes using Deep Learning. bioRxiv 241414. https://doi.org/10.1101/241414.
- Makridakis S, Spiliotis E, Assimakopoulos V (2018) Statistical and Machine Learning forecasting methods: concerns and ways forward. PLoS One 13(3):e0194889. https://doi.org/10.1371/journal.pone.0194889. [PMC free article]
- McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5:115–133.

- McDowell R, Grant D (2016) Genomic selection with deep neural networks. Graduate Theses and Dissertations, p 15973. https://lib.dr.iastate.edu/etd/15973.
- Menden MP, Iorio F, Garnett M, McDermott U, Benes CH et al (2013) Machine learning prediction of cancer cell sensitivity to drugs based on genomic and chemical properties. PLoS One 8:e61318. [PMC free article]
- Montesinos-López A, Montesinos-López OA, Gianola D, Crossa J, Hernández-Suárez CM (2018a) Multi-environment genomic prediction of plant traits using deep learners with a dense architecture. G3: Genes, Genomes, Genetics 8(12):3813–3828. https://doi .org/10.1534/g3.118.200740. [PMC free article]
- Montesinos-López OA, Montesinos-López A, Crossa J, Gianola D, Hernández-Suárez CM et al (2018b) Multi-trait, multi-environment deep learning modeling for genomic-enabled prediction of plant traits. G3: Genes, Genomes, Genetics 8(12):3829–3840. https://doi.org/10.1534/g3.118.200728. [PMC free article]
- Montesinos-López OA, Vallejo M, Crossa J, Gianola D, Hernández-Suárez CM, Montesinos-López A, Juliana P, Singh R (2019a) A benchmarking between deep learning, support vector machine and bayesian threshold best linear unbiased prediction for predicting ordinal traits in plant breeding. G3: Genes, Genomes, Genetics 9(2):601–618. [PMC free article]
- Montesinos-López OA, Martín-Vallejo J, Crossa J, Gianola D, Hernández-Suárez CM, Montesinos-López A, Juliana P, Singh R (2019b) New deep learning genomic prediction model for multi-traits with mixed binary, ordinal, and continuous phenotypes. G3: Genes, Genomes, Genetics 9(5):1545–1556. [PMC free article]
- Montesinos-López OA, Montesinos-López A, Pérez-Rodríguez P, Barrón-López JA, Martini JWR, Fajardo-Flores SB, Gaytan-Lugo LS, Santana-Mancilla PC, Crossa J (2021) A review of deep learning applications for genomic selection. BMC Genomics 22:19. [PMC free article]
- Patterson J, Gibson A (2017) Deep learning: a practitioner's approach. O'Reilly Media.
- Ripley B (1993) Statistical aspects of neural networks. In: Bornndorff-Nielsen U, Jensen J, Kendal W (eds) Networks and chaos—statistical and probabilistic aspects. Chapman and Hall, London, pp 40–123.
- Rouet-Leduc B, Hulbert C, Lubbers N, Barros K, Humphreys CJ et al (2017) Machine learning predicts laboratory earthquakes. Geophys Res Lett 44(28):9276–9282. [CrossRef]
- Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by backpropagating errors. Nature 323:533–536.
- Shalev-Shwartz, Ben-David (2014) Understanding machine learning: from theory to algorithms. Cambridge University Press, New York.
- Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(6):1929–1958.
- Tavanaei A, Anandanadarajah N, Maida AS, Loganantharaj R (2017) A deep learning model for predicting tumor suppressor genes and oncogenes from PDB structure. bioRiv 177378. https://doi.org/10.1101/177378.
- Weron R (2014) Electricity price forecasting: a review of the state-of-the-art with a look into the future. Int J Forecast 30(4):1030–1081.
- Wiley JF (2016) R deep learning essentials: build automatic classification and prediction models using unsupervised learning. Packt Publishing, Birmingham, Mumbai.

Functions, Module, File Handling in Python

Functions

In Python, the **function is a block of code defined with a name**. We use functions whenever we need to perform the same task multiple times without writing the same code again. It can take arguments and returns the value.

Python has a DRY principle like other programming languages. DRY stands for Don't Repeat Yourself. Consider a scenario where we need to do some action/task many times. We can define that action only once using a function and call that function whenever required to do the same activity.

Function improves efficiency and reduces errors because of the reusability of a code. Once we create a function, we can call it anywhere and anytime. The benefit of using a function is reusability and modularity.

Types of Functions

Python support two types of functions

1. Built-in function
2. User-defined function

Built-in function

The functions which are come along with Python itself are called a built-in function or **predefined function**. Some of them are listed below. range(), type(), input(), eval() etc.

Example: Python range() function generates the immutable sequence of numbers starting from the given start integer to the stop integer.

```
for i in range(1, 10):
    print(i, end=' ')
# Output 1 2 3 4 5 6 7 8 9
```

User-defined function

Functions which are created by programmer explicitly according to the requirement are called a user-defined function.

Creating a Function

Use the following steps to define a function in Python.

- Use the def keyword with the function name to define a function.
- Next, pass the number of parameters as per your requirement. (Optional).
- Next, define the function body with a **block of code**. This block of code is nothing but the action you want to perform.

In Python, no need to specify curly braces for the function body. The only **indentation** is essential to separate code blocks. Otherwise, you will get an error.

Syntax of creating a function

```
def function_name(parameter1, parameter2):
    # function body
    # write some action
return value
```

Here,

- function_name: Function name is the name of the function. We can give any name to function.
- parameter: Parameter is the value passed to the function. We can pass any number of parameters. Function body uses the parameter's value to perform an action
- function_body: The function body is a block of code that performs some task. This block of code is nothing but the action you wanted to accomplish.
- return value: Return value is the output of the function.

Note: While defining a function, we use two keywords, def (mandatory) and return (optional).

Creating a function without any parameters

```
# function
def message():
   print("Welcome participants to ICAR-IASRI")

# call function using its name
message()
```

Creating a function with parameters

Let's create a function that takes two parameters and displays their values.

In this example, we are creating function with two parameters ' name' and 'age'.

```
# function
def training_func(name, training_name):
    print("Hello", name, "Welcome to ICAR-IASRI")
    print("Your training title is", training_name)

# call function
training_func ('XYZ', 'Python')
```

Creating a function with parameters and return value

Functions can return a value. The return value is the output of the function. Use the return keyword to return value from a function.

```
# function
def calculator(a, b):
    add = a + b
    # return the addition
    return add

# call function
# take return value in variable
res = calculator(20, 5)

print("Addition :", res)
# Output Addition : 25
```

Calling a function

Once we defined a function or finalized structure, we can call that function by using its name. We can also call that function from another function or program by importing it.

To call a function, use the name of the function with the parenthesis, and if the function accepts parameters, then pass those parameters in the parenthesis.

```
# function
def even_odd(n):
    # check number is even or odd
    if n % 2 == 0:
```

```
        print('Even number')
    else:
        print('Odd Number')

# calling function by its name
even_odd(19)
# Output Odd Number
```

Calling a function of a module

You can take advantage of the built-in module and use the functions defined in it. For example, Python has a random module that is used for generating random numbers and data. It has various functions to create different types of random data.

Let's see how to use functions defined in any module.

- First, we need to use the import statement to import a specific function from a module.
- Next, we can call that function by its name.

```
# import randint function
from random import randint

# call randint function to get random number
print(randint(10, 20))
# Output 14
```

Docstrings

In Python, the documentation string is also called a **docstring**. It is a descriptive text (like a comment) written by a programmer to let others know what block of code does.

We write docstring in source code and define it immediately after module, class, function, or method definition

It is being declared using triple single quotes (''' ''') or triple-double quote(""" """).

We can access docstring using doc attribute (__doc__) for any object like list, tuple, dict, and user-defined function, etc.

Single-Line Docstring

The single-line docstring is a docstring that fits in one line. We can use the triple single or triple-double quotes to define it. The Opening and closing quotes need to be the same. By convention, we should use to use the triple-double quotes to define docstring.

```
def factorial(x):
    """This function returns the factorial of a given number."""
    return x
# access doc string
print(factorial.__doc__)
```

When you use the help function to get the information of any function, it returns the docstring.

```
# pass function name to help() function
print(help(factorial))
```

```
Help on function factorial in module main:
factorial(x)
    This function returns the factorial of a given number.
None
```

Multi-Line Docstring

A multi-line Docstrings is the same single-line Docstrings, but it is followed by a single blank line with the descriptive text.

The general format of writing a multi-line Docstring is as follows:

Example

```
def any_fun(parameter1):
"""
  Description of function

  Arguments:
  parameter1(int):Description of parameter1

  Returns:
  int value
"""
print(any_fun.__doc__)
```

Output

```
Description of function

Arguments
parameter1(int):Description of parameter1

Returns:
int value
```

Return Value From a Function

In Python, to return value from the function, a return statement is used. It returns the value of the expression following the returns keyword.

Syntax of return statement

```
def fun():
   statement-1
   statement-2
   statement-3
   .
   .
   return [expression]
```

The return value is nothing but a outcome of function.

- The return statement ends the function execution.
- For a function, it is not mandatory to return a value.
- If a return statement is used without any expression, then the None is returned.
- The return statement should be inside of the function block.

```
def is_even(list1):
   even_num = []
   for n in list1:
      if n % 2 == 0:
         even_num.append(n)
   # return a list
   return even_num

# Pass list to the function
even_num = is_even([2, 3, 42, 51, 62, 70, 5, 9])
print("Even numbers are:", even_num)
```

Output

Even numbers are: [2, 42, 62, 70]

Return Multiple Values

You can also return multiple values from a function. Use the return statement by separating each expression by a comma.

Example: –

In this example, we are returning three values from a function. We will also see how to process or read multiple return values in our code.

```
def arithmetic(num1, num2):
    add = num1 + num2
    sub = num1 - num2
    multiply = num1 * num2
    division = num1 / num2
    # return four values
    return add, sub, multiply, division

# read four return values in four variables
a, b, c, d = arithmetic(10, 2)

print("Addition: ", a)
print("Subtraction: ", b)
print("Multiplication: ", c)
print("Division: ", d)
```

Scope and Lifetime of Variables

When we define a function with variables, then those variables' scope is limited to that function. In Python, the scope of a variable is an area where a variable is declared. It is called the variable's local scope.

We cannot access the local variables from outside of the function. Because the scope is local, those variables are not visible from the outside of the function.

Note: The inner function does have access to the outer function's local scope.

When we are executing a function, the life of the variables is up to running time. Once we return from the function, those variables get destroyed. So function does no need to remember the value of a variable from its previous call.

The following code shows the scope of a variable inside a function.

Example

```
global_lang = 'DataScience'

def var_scope_test():
    local_lang = 'Python'
    print(local_lang)
```

```
var_scope_test()
# Output 'Python'

# outside of function
print(global_lang)
# Output 'DataScience'

# NameError: name 'local_lang' is not defined
print(local_lang)
```

In the above example, we print the local and global variable values from outside of the function. The global variable is accessible with its name global_lang.

But when we try to access the local variable with its name local_lang, we got a NameError, because the local variable is not accessible from outside of the function.

Example

```
def function1():
   # local variable
   loc_var = 888
   print("Value is :", loc_var)

def function2():

   print("Value is :", loc_var)

function1()
function2()
```

Output

```
Value is : 888
print("Value is :", loc_var) # gives error,
NameError: name 'loc_var' is not defined
```

Global Variable in function

A Global variable is a variable that is declared outside of the function. The scope of a global variable is broad. It is accessible in all functions of the same module.

Example

```
global_var = 999

def function1():
```

```
    print("Value in 1nd function :", global_var)

def function2():
    print("Value in 2nd function :", global_var)

function1()
function2()
```

Output

```
Value in 1nd function : 999
Value in 2nd function : 999
```

Global Keyword in Function

In Python, global is the keyword used to access the actual global variable from outside the function. we use the global keyword for two purposes:

1. To declare a global variable inside the function.
2. Declaring a variable as global, which makes it available to function to perform the modification.

Let's see what happens when we don't use global keyword to access the global variable in the function

```
# Global variable
global_var = 5

def function1():
    print("Value in 1st function :", global_var)

def function2():
    # Modify global variable
    # function will treat it as a local variable
    global_var = 555
    print("Value in 2nd function :", global_var)

def function3():
    print("Value in 3rd function :", global_var)

function1()
function2()
function3()
```

Output

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 5
```

As you can see, function2() treated global_var as a new variable (local variable). To solve such issues or access/modify global variables inside a function, we use the global keyword.

```
# Global variable
x = 5

# defining 1st function
def function1():
   print("Value in 1st function :", x)

# defining 2nd function
def function2():
   # Modify global variable using global keyword
   global x
   x = 555
   print("Value in 2nd function :", x)

# defining 3rd function
def function3():
   print("Value in 3rd function :", x)

function1()
function2()
function3()
```

Output

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 555
```

Python Function Arguments

The argument is a value, a variable, or an object that we pass to a function or method call. In Python, there are four types of arguments allowed.

1. Positional arguments
2. keyword arguments
3. Default arguments
4. Variable-length arguments

Positional Arguments

Positional arguments are arguments that are pass to function in **proper positional order**. That is, the 1st positional argument needs to be 1st when the function is called. The 2nd positional argument needs to be 2nd when the function is called, etc. See the following example for more understanding.

Example

```
def add(a, b):
    print(a - b)

add(50, 10)
# Output 40
add(10, 50)
# Output -40
```

Keyword Arguments

A keyword argument is an argument value, passed to function preceded by the variable name and an equals sign.

Example

```
def message(name, surname):
    print("Hello", name, surname)

message(name="A", surname="B")
message(surname="C", name="D")
```

Output

```
Hello A B
Hello C D
```

In keyword arguments order of argument is not matter, but the number of arguments must match. Otherwise, we will get an error.

While using keyword and positional argument simultaneously, we need to pass 1st arguments as positional arguments and then keyword arguments. Otherwise, we will get SyntaxError. See the following example.

Example

```
def message(first_nm, last_nm):
    print("Hello..!", first_nm, last_nm)
```

```
# correct use
message("A", "B")
message("A", last_nm="B")

# Error
# SyntaxError: positional argument follows keyword argument
message(first_nm="A", "B")
```

Default Arguments

Default arguments take the default value during the function call if we do not pass them. We can assign a default value to an argument in function definition using the = assignment operator.

Example

```
# function with default argument
def message(name="Guest"):
    print("Hello", name)

# calling function with argument
message("John")

# calling function without argument
message()
```

Output

```
Hello John
Hello Guest
```

Variable-length Arguments

In Python, sometimes, there is a situation where we need to pass multiple numbers of arguments to the function. Such types of arguments are called **variable-length arguments**. We can declare a variable-length argument with the * **(asterisk)** symbol.

```
def fun(*var):
    function body
```

We can pass any number of arguments to this function. Internally all these values are represented in the form of a **tuple**.

Example

```
def addition(*numbers):
    total = 0
    for no in numbers:
        total = total + no
    print("Sum is:", total)

# 0 arguments
addition()

# 5 arguments
addition(10, 5, 2, 5, 4)

# 3 arguments
addition(78, 7, 2.5)
```

Output

```
Sum is: 0
Sum is: 26
Sum is: 87.5
```

Recursive Function

A recursive function is a function that calls itself, again and again.

Consider, calculating the factorial of a number is a repetitive activity, in that case, we can call a function again and again, which calculates factorial.

```
def factorial(no):
    if no == 0:
        return 1
    else:
        return no * factorial(no - 1)

print("factorial of a number is:", factorial(8))
```

Output

```
factorial of a number is: 40320
```

The advantages of the recursive function are:

1. By using recursive, we can reduce the length of the code.
2. The readability of code improves due to code reduction.
3. Useful for solving a complex problem

The disadvantage of the recursive function:

1. The recursive function takes more memory and time for execution.
2. Debugging is not easy for the recursive function.

Python Anonymous/Lambda Function

Sometimes we need to declare a function without any name. The nameless property function is called an **anonymous function** or **lambda function.**

The reason behind the using anonymous function is for instant use, that is, one-time usage. Normal function is declared using the def function. Whereas the anonymous function is declared using the lambda keyword.

A Python lambda function is a single expression. But, in a lambda body, we can expand with expressions over multiple lines using parentheses or a multiline string.
ex : lambda n:n+n

Syntax of lambda function:

lambda: argument_list:expression

When we define a function using the lambda keyword, the code is very concise so that there is more readability in the code. A lambda function can have any number of arguments but return only one value after expression evaluation.

Let's see an example to print even numbers without a lambda function and with a lambda function. See the difference in line of code as well as readability of code.

Example 1: Program for even numbers without lambda function

```
def even_numbers(nums):
   even_list = []
   for n in nums:
      if n % 2 == 0:
         even_list.append(n)
   return even_list


num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = even_numbers(num_list)
print("Even numbers are:", ans)
```

Output

Even numbers are: [10, 12, 78, 6]

Example 2: Program for even number with a lambda function

```
l = [10, 5, 12, 78, 6, 1, 7, 9]
even_nos = list(filter(lambda x: x % 2 == 0, l))
print("Even numbers are: ", even_nos)
```

Output

Even numbers are: [10, 12, 78, 6]

We are not required to write **explicitly return statements** in the lambda function because the lambda internally returns expression value.

Lambda functions are more useful when we pass a function as an argument to another function. We can also use the lambda function with built-in functions such as filter, map, reduce because this function requires another function as an argument.

filter() function in Python

In Python, the filter() function is used to return the filtered value. We use this function to filter values based on some conditions.

Syntax of filter() function:

filter(funtion, sequence)

where,

- function – Function argument is responsible for performing condition checking.
- sequence – Sequence argument can be anything like list, tuple, string

Example: lambda function with filter()

```
l = [-10, 5, 12, -78, 6, -1, -7, 9]
positive_nos = list(filter(lambda x: x > 0, l))
print("Positive numbers are: ", positive_nos)
```

Output

Positive numbers are: [5, 12, 6, 9]

Python Modules

In Python, modules refer to the Python file, which contains Python code like Python statements, classes, functions, variables, etc. A file with Python code is defined with extension.py

For example: In Test.py, where the test is the module name.

In Python, large code is divided into small modules. The benefit of modules is, it provides a way to share reusable functions.

Types of modules

In Python, there are two types of modules.

1. Built-in Modules
2. User-defined Modules

Built-in modules

Built-in modules come with default Python installation. One of Python's most significant advantages is its rich library support that contains lots of built-in modules. Hence, it provides a lot of reusable code.

Some commonly used Python built-in modules are datetime, os, math, sys, random, etc.

User-defined modules

The modules which the user defines or create are called a **user-defined module**. We can create our own module, which contains classes, functions, variables, etc., as per our requirements.

How to import modules?

In Python, the import statement is used to import the whole module. Also, we can import specific classes and functions from a module.

import module name.

When the interpreter finds an import statement, it imports the module presented in a search path. The module is loaded only once, even we import multiple times.

To import modules in Python, we use the Python import keyword. With the help of the import keyword, both the **built-in** and **user-defined** modules are imported. Let's see an example of importing a math module.

```
import math
# use math module functions
print(math.sqrt(5))
# Output 2.23606797749979
```

Import multiple modules

If we want to use more than one module, then we can import multiple modules. This is the simplest form of import statement that we already used in the above example.

Syntax of import statement:

import module1,module2,.. moduleN

Example

```
# Import two modules
import math, random
```

```
print(math.factorial(5))
print(random.randint(10, 20))
```

Output

```
120
18
```

Import only specific classes or functions from a module

To import particular classes or functions, we can use the from...import statement. It is an alternate way to import. Using this way, we can import individual attributes and methods directly into the program.

In this way, we are not required to use the module name. See the following example.

Syntax of from...import statement:

```
from <module_name> import <name(s)>
```

Example

```
# import only factorial function from math module
from math import factorial
print(factorial(5))
```

Output

```
120
```

Import with renaming a module

If we want to use the module with a different name, we can use from..import…as statement.

It is also possible to import a particular method and use that method with a different name. It is called **aliasing**. Afterward, we can use that name in the entire program.

Syntax of from..import ..as keyword:

```
from <module_name> import <name> as <alternative_name>
```

Example 1: Import a module by renaming it

```
import random as rand
print(rand.randrange(10, 20, 2))
```

Output

```
16
```

Example 2: Import a method by renaming it

```
# rename randint as random_number
from random import randint as random_number
```

```
# Gives any random number from range(10, 50)
print(random_number(10, 50))
```

Output
32

Import all names

If we need to import all functions and attributes of a specific module, then instead of writing all function names and attribute names, we can import all using an **asterisk** *.

Syntax of import * statement:
import *

Example

```
from math import *
print(pow(4,2))
print(factorial(5))

print(pi*3)
print(sqrt(100))
```

Output
16.0
120
9.42477796076938
10.0

Create Module

In Python, to create a module, write Python code in the file, and save that file with the.py extension. Here our module is created.

Example

```
def my_func():
   print("Welcome to ICAR-IASRI")
```

Output
Welcome to ICAR-IASRI

Variables in Module

In Python, the module contains Python code like classes, functions, methods, but it also has variables. A variable can list, tuple, dict, etc.

Let’s see this with an example:

First, create a Python module with the name test_module.py and write the below code in that file.

Example

```
cities_list = ['Mumbai', 'Delhi', 'Bangalore', 'Karnataka', 'Hyderabad']
```

Now, create a Python file with the name test_file.py, write the below code and import the above module test_module.py in that file. See the following code.

```
import test_module
# access first city
city = test_module.cities_list[1]
print("Accessing 1st city:", city)

# Get all cities
cities = test_module.cities_list
print("Accessing All cities :", cities)
```

When we execute this test_file.py, the variable of test_module.py is accessible using the dot(.)operator.

Output

```
Accessing 1st city: Delhi
Accessing All cities : ['Mumbai', 'Delhi', 'Bangalore', 'Karnataka', 'Hyderabad']
```

File Handling in Python

File handling is an important part of any web application. Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

Types of File

- **Binary File**: The binary files are used to store binary data such as images, video files, audio files, etc.
- **Text File**: Text file usually we use to store character data. For example, test.txt

Binary files in Python

Most of the files that we see in our computer system are called binary files.

Example:

1. **Document files:** .pdf, .doc, .xls etc.
2. **Image files:** .png, .jpg, .gif, .bmp etc.
3. **Video files:** .mp4, .3gp, .mkv, .avi etc.
4. **Audio files:** .mp3, .wav, .mka, .aac etc.
5. **Database files:** .mdb, .accde, .frm, .sqlite etc.
6. **Archive files:** .zip, .rar, .iso, .7z etc.
7. **Executable files:** .exe, .dll, .class etc.

Text files in Python

Text files don't have any specific encoding and it can be opened in normal text editor itself.

Example:

- **Web standards:** html, XML, CSS, JSON etc.
- **Source code:** c, app, js, py, java etc.
- **Documents:** txt, tex, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

Python File Handling Operations

Most importantly there are 4 types of operations that can be handled by Python on files:

- Open
- Read
- Write
- Close

Other operations include:

- Rename
- Delete

Python Create and Open a File

Python has an in-built function called open() to open a file. It takes a minimum of one argument as mentioned in the below syntax. The open method returns a file object which is used to access the write, read and other in-built methods.

Here, file_name is the name of the file or the location of the file that you want to open, and file_name should have the file extension included as well. Which means in **test.txt** – the term test is the name of the file and .txt is the extension of the file.

The mode in the open function syntax will tell Python as what operation you want to do on a file.

- **'r' – Read Mode:** Read mode is used only to read data from the file.
- **'w' – Write Mode:** This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.
- **'a' – Append Mode:** Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
- **'r+' – Read or Write Mode:** This mode is used when we want to write or read the data from the same file.
- **'a+' – Append or Read Mode:** This mode is used when we want to read data from the file or append the data into the same file.

Note: The above-mentioned modes are for opening, reading or writing text files only.

While using binary files, we have to use the same modes with the letter **'b'** at the end. So that Python can understand that we are interacting with binary files.

- **'wb' –** Open a file for write only mode in the binary format.
- **'rb' –** Open a file for the read-only mode in the binary format.
- **'ab' –** Open a file for appending only mode in the binary format.
- **'rb+' –** Open a file for read and write only mode in the binary format.
- **'ab+' –** Open a file for appending and read-only mode in the binary format.

Example 1:

```
fo=open("/Users/akshaydheeraj/Desktop/Python_Practice/test.txt","r+")
```

In the above example, we are opening the file named 'test.txt' present at the location 'C:/Documents/Python/' and we are opening the same file in a read-write mode which gives us more flexibility.

Let's create the file named test.txt with the sample text as

Hello everyone! Welcome to ICAR-IASRI

Good morning.

How are you?

Python Read From File

In order to read a file in python, we must open the file in read mode.

There are three ways in which we can read the files in python.

- read([n])
- readline([n])
- readlines()

Here, n is the number of bytes to be read.

Here we are opening the file test.txt in a read-only mode and are reading only the first 5 characters of the file using the **fo.read(5)** method.

print(fo.read(5))

Output:

Hello

Here we have not provided any argument inside the **read()** function. Hence it will read all the content present inside the file.

print(fo.read())

Output:

Hello everyone! Welcome to ICAR-IASRI

Good morning.

How are you?

The **readline()** method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

print(fo.readline())

Output:

Hello everyone! Welcome to ICAR-IASRI

Python provides also the **readlines()** method which is used for the reading lines. It returns the list of the lines till the end of **file(EOF)** is reached.

print(fo.readlines())

Output:

['Hello everyone! Welcome to ICAR-IASRI\n', 'Good morning.\n', 'How are you?\n', '\n']

Python Write to File

In order to write data into a file, we must open the file in write mode.

We need to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.

We have two methods for writing data into a file as shown below.

- write(string)
- writelines(list)

Example 1:

my_file = open("C:/Documents/Python/test.txt", "w")

my_file.write("Hello World")

The above code writes the String 'Hello World' into the 'test.txt' file.

Example 2:

my_file = open("C:/Documents/Python/test.txt", "w")

my_file.write("Hello World\n")

my_file.write("Hello Python")

The first line will be 'Hello World' and as we have mentioned \n character, the cursor will move to the next line of the file and then write 'Hello Python'.

Remember if we don't mention \n character, then the data will be written continuously in the text file like 'Hello WorldHelloPython'

fruits = ["Apple\n", "Orange\n", "Grapes\n", "Watermelon"]

my_file = open("C:/Documents/Python/test.txt", "w")

my_file.writelines(fruits)

Python Append to File

To append data into a file we must open the file in 'a+' mode so that we will have access to both the append as well as write modes.

Example 1:

my_file = open("C:/Documents/Python/test.txt", "a+")

my_file.write ("Strawberry")

The above code appends the string 'Apple' at the **end** of the 'test.txt' file.

Python Close File

In order to close a file, we must first open the file. In python, we have an in-built method called close() to close the file which is opened.

Whenever you open a file, it is important to close it, especially, with write method. Because if we don't call the close function after the write method then whatever data we have written to a file will not be saved into the file.

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.read())
my_file.close()
```

Python Rename or Delete File

Python provides us with an "os" module which has some in-built methods that would help us in performing the file operations such as renaming and deleting the file.

In order to use this module, first of all, we need to import the "os" module in our program and then call the related methods.

rename() method:

This rename() method accepts two arguments i.e. the current file name and the new file name.

Syntax:

```
os.rename(current_file_name, new_file_name)
```

Example 1:

```
import os
os.rename("test.txt", "test1.txt")
```

Here 'test.txt' is the current file name and 'test1.txt' is the new file name.

You can specify the location as well as shown in the below example.

Example 2:

```
import os
os.rename("C:/Documents/Python/test.txt", "C:/Documents/Python/test1.txt")
```

Deep Learning and Convolutional Neural Networks Architectures

Dr Sapna Nigam
ICAR – IASRI, New Delhi – 110012

1. Artificial intelligence

Artificial intelligence was born in the 1950s when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to "think". A concise definition of the field would be as follows: the effort to automate intellectual tasks normally performed by humans. As such, AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don't involve any learning. Early chess programs, for instance, only involved hardcoded rules crafted by programmers and didn't qualify as machine learning. For a fairly long time, many experts believed that human-level artificial intelligence could be achieved by having programmers handcraft a sufficiently large set of explicit rules for manipulating knowledge. This approach is known as symbolic AI, and it was the dominant paradigm in AI from the 1950s to the late 1980s. It reached its peak popularity during the expert systems boom of the 1980s. Although symbolic AI proved suitable to solve well-defined, logical problems, such as playing chess, it turned out to be intractable to figure out explicit rules for solving more complex, fuzzy problems, such as image classification, speech recognition, and language translation. A new approach arose to take symbolic AI's place that is machine learning.

2. Machine learning

Machine learning arises from this question: could a computer go beyond "what we know how to order it to perform" and learn on its own how to perform a specified task? Could a computer surprise us? Rather than programmers crafting data-processing rules by hand, could a computer automatically learn these rules by looking at data? This question opens the door to a new programming paradigm. In classical programming, the paradigm of symbolic AI, humans input rules (a program) and data to be processed according to these rules, and outcome answers (Figure 1). With machine learning, humans input data as well as the answers expected from the data, and outcome of the rules. These rules can then be applied to new data to produce original answers.

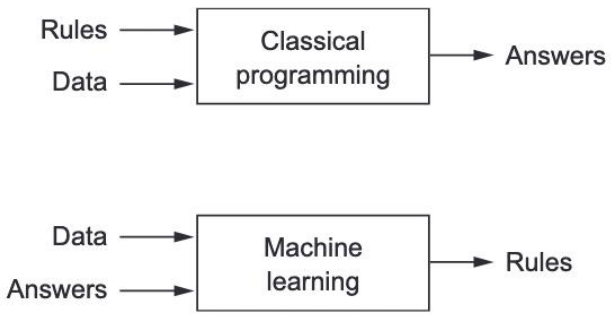


Figure 1: Machine learning new paradigm

A machine-learning system is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that

eventually allows the system to come up with rules for automating the task. Although machine learning only started to flourish in the 1990s, it has quickly become the most popular and most successful subfield of AI, a trend driven by the availability of faster hardware and larger datasets. Machine learning is tightly related to mathematical statistics, but it differs from statistics in several important ways. Unlike statistics, machine learning tends to deal with large, complex datasets (such as a dataset of millions of images, each consisting of tens of thousands of pixels) for which classical statistical analysis such as Bayesian analysis would be impractical. As a result, machine learning, and especially deep learning, exhibits comparatively little mathematical theory—maybe too little—and is engineering-oriented. It's a hands-on discipline in which ideas are proven empirically more often than theoretically.

3. Learning representations from data

Machine learning discovers rules to execute a data-processing task, giving examples of what's expected. So, to do machine learning, the following things are required:

- Input data points —For instance, if the task is speech recognition, these data points could be sound files of people speaking. If the task is image tagging, they could be pictures.
- Examples of the expected output —In a speech-recognition task, these could be human-generated transcripts of sound files. In an image task, expected outputs could be tags such as "dog," "cat," and so on.
- A way to measure whether the algorithm is doing a good job —This is necessary in order to determine the distance between the algorithm's current output and its expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call learning.

A machine-learning model transforms its input data into meaningful outputs, a process that is "learned" from exposure to known examples of inputs and outputs. Therefore, the central problem in machine learning and deep learning is to meaningfully transform data : in other words, to learn useful representations of the input data at hand—representations that get us closer to the expected output.

4. The "deep" in deep learning

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations. The deep in deep learning isn't a reference to any kind of deeper understanding achieved by the approach; rather, it stands for this idea of successive layers of representations. How many layers contribute to a model of the data is called the depth of the model. Other appropriate names for the field could have been layered representations learning and hierarchical representations learning. Modern deep learning often involves tens or even hundreds of successive layers of representations—and they're all learned automatically from exposure to training data. Meanwhile, other approaches to machine learning tend to focus on learning only one or two layers of representations of the data; hence, they're sometimes called shallow learning. In deep learning, these layered representations are (almost always) learned via models called neural networks, structured in literal layers stacked on top of each other. The term neural network is a reference to neurobiology, but although some of the central

concepts in deep learning were developed in part by drawing inspiration from our understanding of the brain, deep-learning models are not models of the brain. There's no evidence that the brain implements anything like the learning mechanisms used in modern deep-learning models.

What do the representations learned by a deep-learning algorithm look like? Let's examine how a network several layers deep (Figure 2) transforms an image of a digit in order to recognize what digit it is.

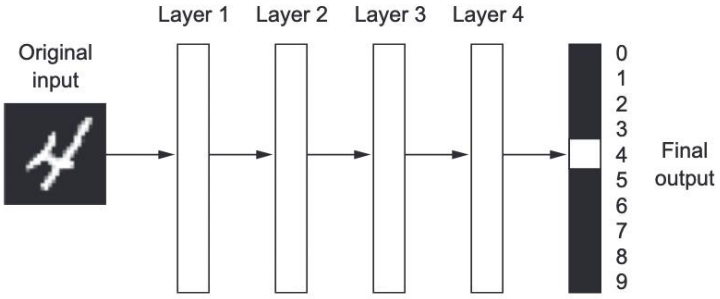


Figure 2: Digit Classification example

As you can see in Figure, the network transforms the digit image into representations that are increasingly different from the original image and increasingly informative about the final result. You can think of a deep network as a multistage information-distillation operation, where information goes through successive filters and comes out increasingly purified (that is, useful with regard to some task).

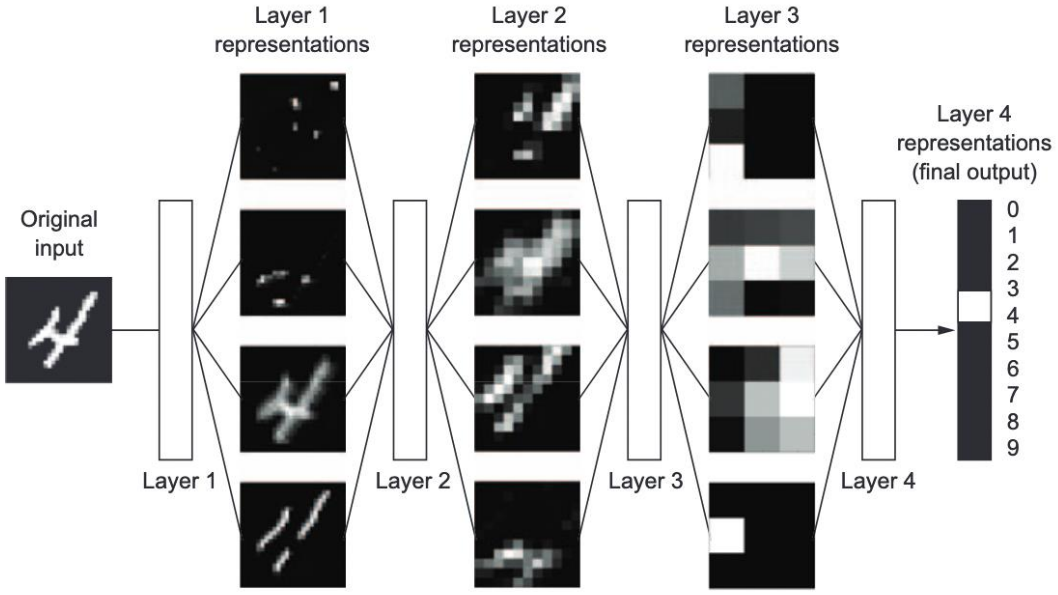


Figure 3: Deep representations learned by a digit classification model

So that's what deep learning is, technically: a multistage way to learn data representations. It's a simple idea—but, as it turns out, very simple mechanisms, sufficiently scaled, can end up looking like magic.

5. Understanding how deep learning works

The specification of what a layer does to its input data is stored in the layer's weights, which in essence are a bunch of numbers. Weights are also sometimes called the parameters of a layer. In this context, learning means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated targets. But here's the thing: a deep neural network can contain tens of millions of parameters. Finding the correct value for all of them may seem like a daunting task, especially given that modifying the value of one parameter will affect the behaviour of all the others!

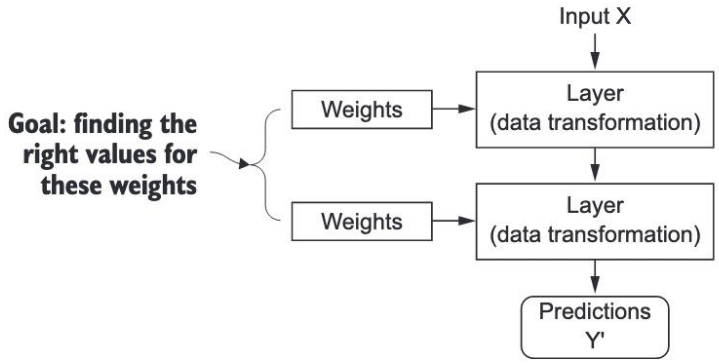


Figure 4: A neural network is parameterized by its weights

The loss function takes the prediction s of the network and the true target (what you wanted the network to output) and computes a distance score, capturing how well the network has done on this specific example (figure 5).

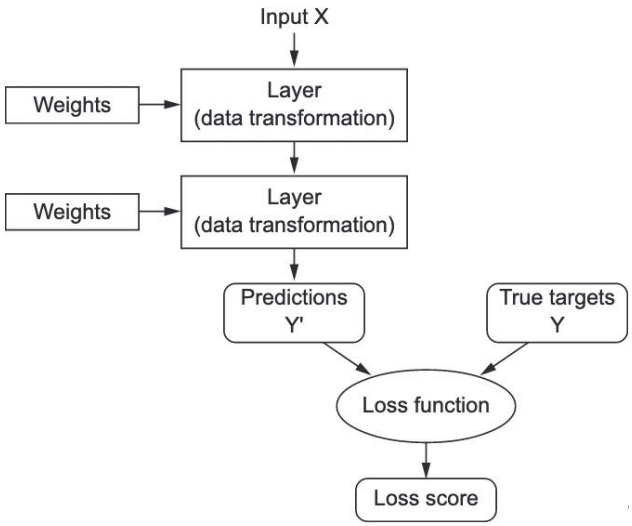


Figure 5: A loss function measures the quality of the network's output

The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current example This adjustment is the job of the optimizer, which implements what's called the

Backpropagation algorithm: the central algorithm in deep learning. The next chapter explains in more detail how backpropagation works.

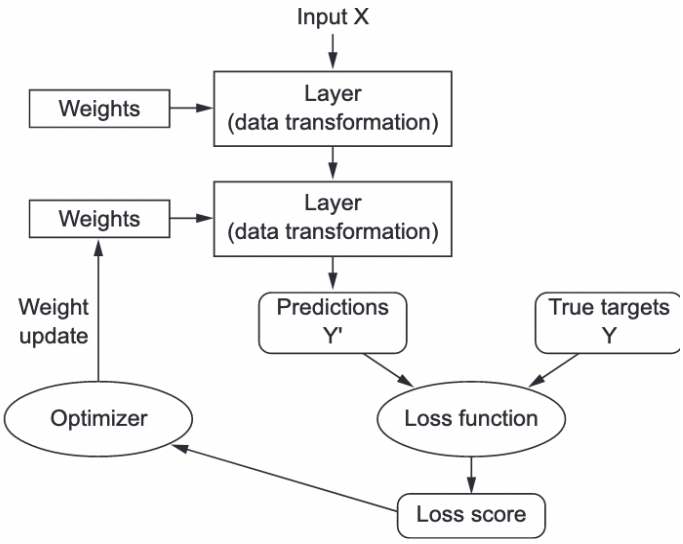


Figure 6: The loss score is used as a feedback signal to adjust the weights

Initially, the weights of the network are assigned random values, so the network merely implements a series of random transformations. Naturally, its output is far from what it should ideally be, and the lo ss score is accordingly very high. But with every example the network processes, the weights are adjusted a little in the correct direction, and the loss score decreases. This is the training loop, which, repeated a sufficient number of times (typically tens of iterations over thousands of examples), yields weight values that minimize the loss function. A network with a minimal loss is one for which the outputs are as close as they can be to the targets: a trained network. Once again, it's a simple mechanism that, once scaled, ends up looking like magic.

6. What deep learning has achieved so far

In particular, deep learning has achieved the following breakthroughs, all in historically difficult areas of machine learning:

Near-human-level image classification
Near-human-level speech recognition
Near-human-level handwriting transcription
Improved machine translation
Improved text-to-speech conversion
Digital assistants such as Google Now and Amazon Alexa
Near-human-level autonomous driving
Improved ad targeting, as used by Google, Baidu, and Bing
Improved search results on the web
Ability to answer natural-language questions
Superhuman Go playing

7. What makes deep learning different

Deep learning, on the other hand, completely automates this step: with deep learning, you learn all features in one pass rather than having to engineer them yourself. This has greatly simplified machine-learning workflows, often replacing sophisticated multistage pipelines with a single, simple, end-to-end deep-learning model.
In practice, there are fast-diminishing returns to successive applications of shallow-learning methods, because the optimal first representation layer in a three-layer model isn't the optimal first layer in a one-layer or two-layer model. What is transformative about deep learning is that it allows a model to learn all layers of representation jointly, at the same time, rather than in succession (greedily, as it's called). With joint feature learning, whenever the model adjusts one of its internal features, all other features that depend on it automatically adapt to the change, without requiring human intervention. Everything is supervised by a single feedback signal: every change in the model serves the end goal. This is much more powerful than greedily stacking shallow models because it allows for complex, abstract representations to be learned by breaking them down into long series of intermediate spaces (layers); each space is only a simple transformation away from the previous one.

These are the two essential characteristics of how deep learning learns from data: the incremental, layer-by-layer way in which increasingly complex representations are developed, and the fact that these intermediate incremental representations are learned jointly, each layer being updated to follow both the representational needs of the layer above and the needs of the layer below. Together, these two properties have made deep learning vastly more successful than previous approaches to machine learning.

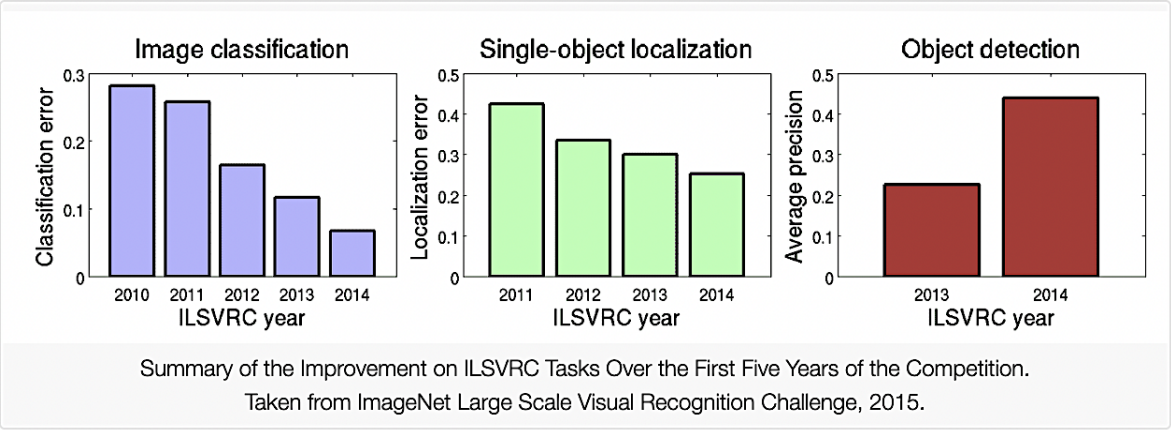
8. The limitations of deep learning

The space of applications that can be implemented with deep learning is nearly infinite. And yet, many applications are completely out of reach for current deep--learning techniques—even given vast amounts of human-annotated data. Say, for instance, that you could assemble a data set of hundreds of thousands—even millions—of English-language descriptions of the features of a software product, written by a product manager, as well as the corresponding source code developed by a team of engineers to meet these requirements. Even with this data, you could not train a deep-learning model to read a product description and generate the appropriate codebase. That's just one example among many. In general, anything that requires reasoning—like programming or applying the scientific method—long-term planning, and algorithmic data manipulation is out of reach for deep-learning models, no matter how much data you throw at them. Even learning a sorting algorithm with a deep neural network is tremendously difficult.

9. Convolutional Neural Networks Architectures

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The data is available for free to researchers for non-commercial use. The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection. The ImageNet Large Scale Visual Recognition Challenge or ILSVRC for short is an annual competition helped between 2010 and 2017 in which challenge tasks use subsets of the

ImageNet dataset. The goal of the challenge was to both promote the development of better computer vision techniques and to benchmark the state of the art. The annual challenge focuses on multiple tasks for "*image classification*" which includes both assigning a class label to an image based on the main object in the photograph and "*object detection*" which involves localizing objects within the photograph. State-of-the-art accuracy has improved significantly from ILSVRC2010 to ILSVRC2014, showcasing the massive progress that has been made in large-scale object recognition over the past five years.



Summary of the Improvement on ILSVRC Tasks Over the First Five Years of the Competition.
Taken from ImageNet Large Scale Visual Recognition Challenge, 2015.

A Convolutional Neural Network (**CNN**, or **ConvNet**) are a special kind of multi-layer neural networks, designed to recognize visual patterns directly from pixel images with minimal pre-processing. It consist of following architectures:

a. LeNet-5 (1998)

LeNet is the first CNN architecture. It was developed in 1998 by Yann LeCun, Corinna Cortes, and Christopher Burges for handwritten digit recognition problems. The model has five convolution layers followed by two fully connected layers. LeNet was the beginning of CNNs in deep learning for computer vision problems. However, LeNet could not train well due to the vanishing gradients problem. To solve this issue, a shortcut connection layer known as max-pooling is used between convolutional layers to reduce the spatial size of images which helps prevent overfitting and allows CNNs to train more effectively. The ability to process higher resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources. Convolutional layers use a subset of the previous layer's channels for each filter to reduce computation and force a break of symmetry in the network. The subsampling layers use a form of average pooling.

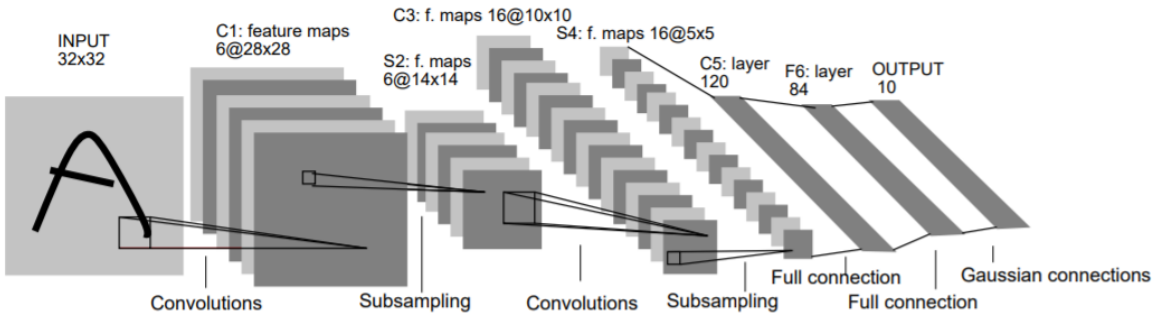


Figure 7: LeNet Architecture

b. AlexNet (2012)

AlexNet was designed by the SuperVision group in 2012 , consisting of Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever to compete in the ImageNet competition. The general architecture is quite similar to LeNet-5, although this model is considerably larger. In 2012, AlexNet significantly outperformed all the prior competitors and won the challenge by reducing the top-5 error from 26% to 15.3%. The Alexnet model has eight CNN layers and three fully-connected layers. It was the first CNN model to have over 100 million parameters with a 60MB training set, which is considered large for deep learning models at that time. The network had a very similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer.

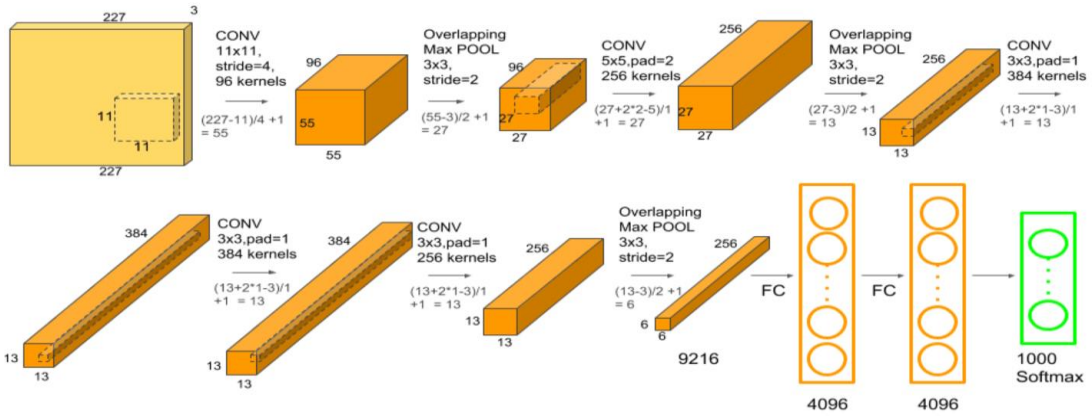


Figure 8: AlexNet Architecture

c. VGGNet (2014)

VGGNet is the CNN architecture that was developed by Karen Simonyan, Andrew Zisserman et al. at Oxford University. VGGNet is a 16-layer CNN with up to 95 million parameters and trained on over one billion images (1000 classes). It can take large input images of 224 x 224-pixel size for which it has 4096 convolutional features. CNNs with such large filters are expensive to train and require a lot of data, which is the main reason why CNN architectures like GoogLeNet (AlexNet architecture) work better than VGGNet for most image classification tasks where input images have a size between 100 x 100-pixel and 350 x 350 pixels. Real-world applications / examples of VGGNet CNN architecture include the ILSVRC 2014 classification task, which was also won by GoogleNet CNN architecture. The VGG CNN model is computationally efficient and serves as a strong baseline for many applications in computer vision due to its applicability on numerous tasks including object detection. The runner-up at the ILSVRC 2014 competition is dubbed VGGNet by the community and was developed by Simonyan and Zisserman. Its deep feature representations are used across multiple neural network architectures like YOLO, SSD etc. VGGNet consists of 16 convolutional layers and is very appealing because of its very uniform architecture. Similar to AlexNet, only 3x3 convolutions, but lots of filters. It is currently the most preferred choice in the community for extracting features from images.

The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor.

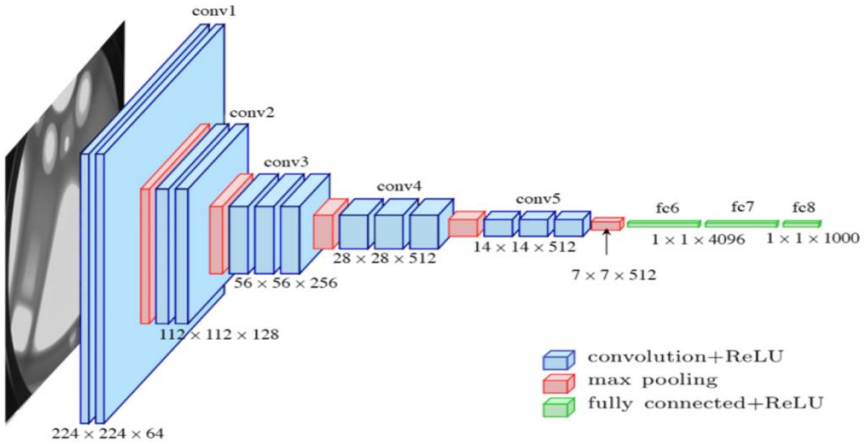


Figure 9: VGGNet Architecture

d. GoogLeNet/Inception(2014)

The winner of the ILSVRC 2014 competition was GoogLeNet(a.k.a. Inception V1) from Google. It achieved a top-5 error rate of 6.67%! This was very close to human level performance which the organisers of the challenge were now forced to evaluate.The network used a CNN inspired by LeNet but implemented a novel element which is dubbed an inception module. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million. GoogLeNet is the CNN architecture used by Google to win ILSVRC 2014 classification task. It was developed by Jeff Dean, Christian Szegedy, Alexandro Szegedy et al.. It achieves deeper architecture by employing a number of distinct techniques, including 1×1 convolution and global average pooling. GoogleNet CNN architecture is computationally expensive. To reduce the parameters that must be learned, it uses heavy unpooling layers on top of CNNs to remove spatial redundancy during training and also features shortcut connections between the first two convolutional layers before adding new filters in later CNN layers. Real-world applications / examples of GoogLeNet CNN architecture include Street View House Number (SVHN) digit recognition task, which is often used as a proxy for roadside object detection.

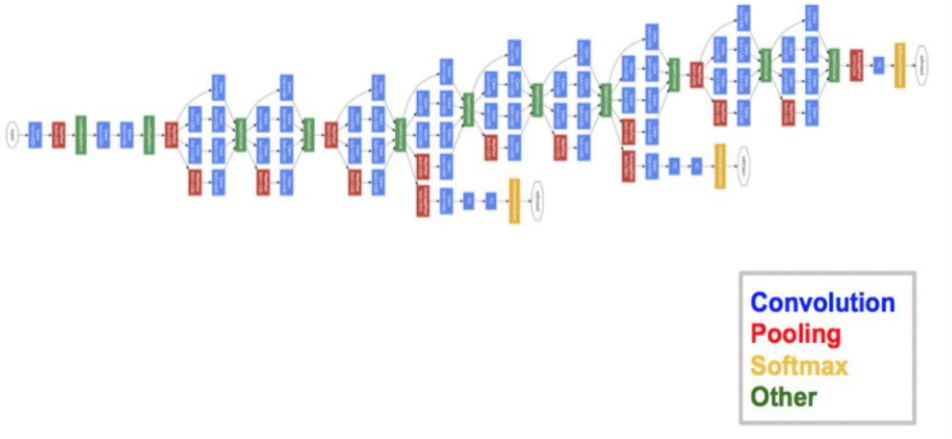


Figure 10: GoogleNet Architecture

e. ResNet(2015)

ResNet is the CNN architecture that was developed by Kaiming He et al. to win the ILSVRC 2015 classification task with a top-five error of only 15.43%.The network has 152 layers and over one million parameters, which is considered deep even for CNNs because it would have taken more than 40 days on 32 GPUs to train the network on the ILSVRC 2015 dataset. CNNs are mostly used for image classification tasks with 1000 classes, but ResNet proves that CNNs can also be used successfully to solve natural language processing problems like sentence completion or machine comprehension. Real-life applications / examples of ResNet CNN architecture include Microsoft's machine comprehension system, which has used CNNs to generate the answers for more than 100k questions in over 20 categories. The CNN architecture ResNet is computationally efficient and can be scaled up or down to match computational power of GPUs. In the residual module, the identity mapping allows to reuse the activations of the previous layer until the adjacent layers learns the weights. This identity mapping solves the problem of vanishing gradient while training a very deep CNN network

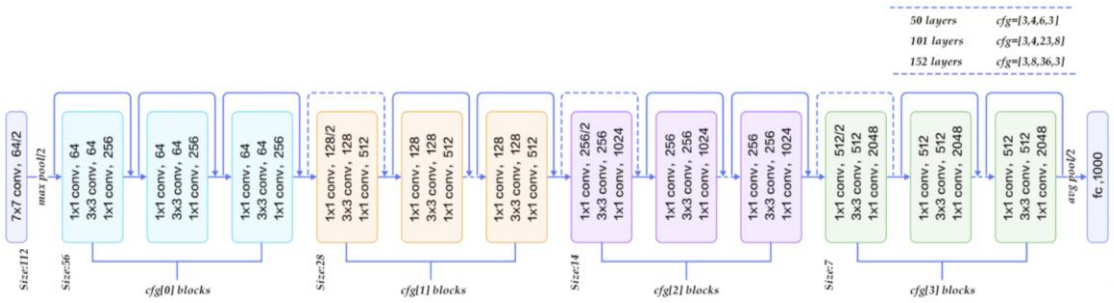


Figure 11: ResNet Architecture

Transfer Learning

Transfer Learning is a machine learning method where we reuse a pre-trained model as the starting point for a model on a new task. To put it simply—a model trained on one task is repurposed on a second, related task as an optimization that allows rapid progress when modeling the second task. By applying transfer learning to a new task, one can achieve significantly higher performance than training with only a small amount of data. ImageNet, AlexNet, and Inception are typical examples of models that have the basis of Transfer learning. Two common approaches are as follows:

- Develop Model Approach
- Pre-trained Model Approach

Develop Model Approach

- *Select Source Task.* You must select a related predictive modelling problem with an abundance of data where there is some relationship in the input data, output data, and/or concepts learned during the mapping from input to output data.
- *Develop Source Model.* Next, you must develop a skilful model for this first task. The model must be better than a naive model to ensure that some feature learning has been performed.
- *Reuse Model.* The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modelling technique used.
- *Tune Model.* Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

Pre-trained Model Approach

- *Select Source Model.* A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
- *Reuse Model.* The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modelling technique used.
- *Tune Model.* Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

References:

Chollet, F. (2021). Deep learning with Python. Simon and Schuster.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publishing Ltd.

Deep Learning using Python Software

Upendra Kumar Pradhan[1] and Ritwika Das[1]
ICAR – IASRI, New Delhi – 110012
Email: upendra.pradhan@icar.gov.in

- **Artificial Intelligence (AI)**

  The term Artificial Intelligence (AI) denotes the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

  Need for AI:

  - ✓ To create expert systems which exhibit intelligent behaviour with the capability to learn, demonstrate, explain and advice its users.
  - ✓ To help machines in finding solutions to complex problems like humans do and applying them as algorithms in a computer-friendly manner.

- **Deep Learning (DL)**

  Now-a-days, Machine Learning (ML) is being applied in various sectors like, medical image analysis, root zone soil moisture estimation, estimation of soil organic matter variability with environmental factors, forecasting of commodity prices, rainfall, automated video surveillance, online customer support, bioinformatics *etc.*

  Machine Learning is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data. It is a subset of AI which allows a machine to automatically learn from past data without programming explicitly.

  Deep Learning (DL) is a subfield of machine learning (Figure 1) which imitates the activities of the human brain in data processing and creating patterns for use in decision making. Deep learning model architectures are derived for ANNs where large number of hidden layers are introduced each of them performing some specific functionalities.

- **Advantages of Deep Learning Algorithms**

  - ✓ High volume data processing (Big data)
  - ✓ Parallel computing
  - ✓ High computation speed
  - ✓ No need for feature selection separately

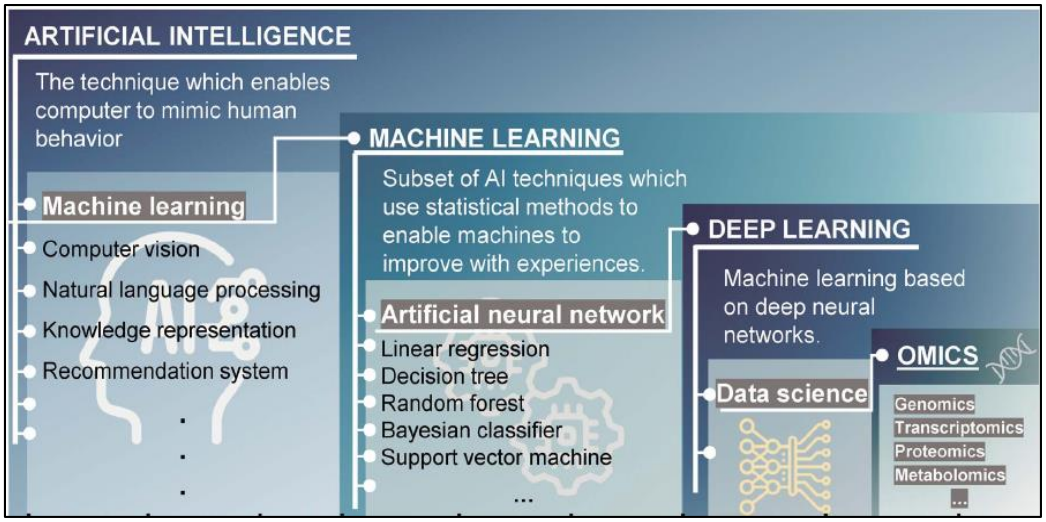


Figure 1: AI vs ML vs DL

- **Types of Deep Learning**

  Depending on the types of hidden layers and learning methods, various deep learning models have been introduced (Figure 2). Some significant DL models are:

  ✓ Convolutional Neural Network (CNN)
  ✓ Recurrent Neural Network (RNN)
  ✓ Deep Belief Network (DBN)
  ✓ Autoencoder
  ✓ Generative Adversarial Network (GAN) *etc*.

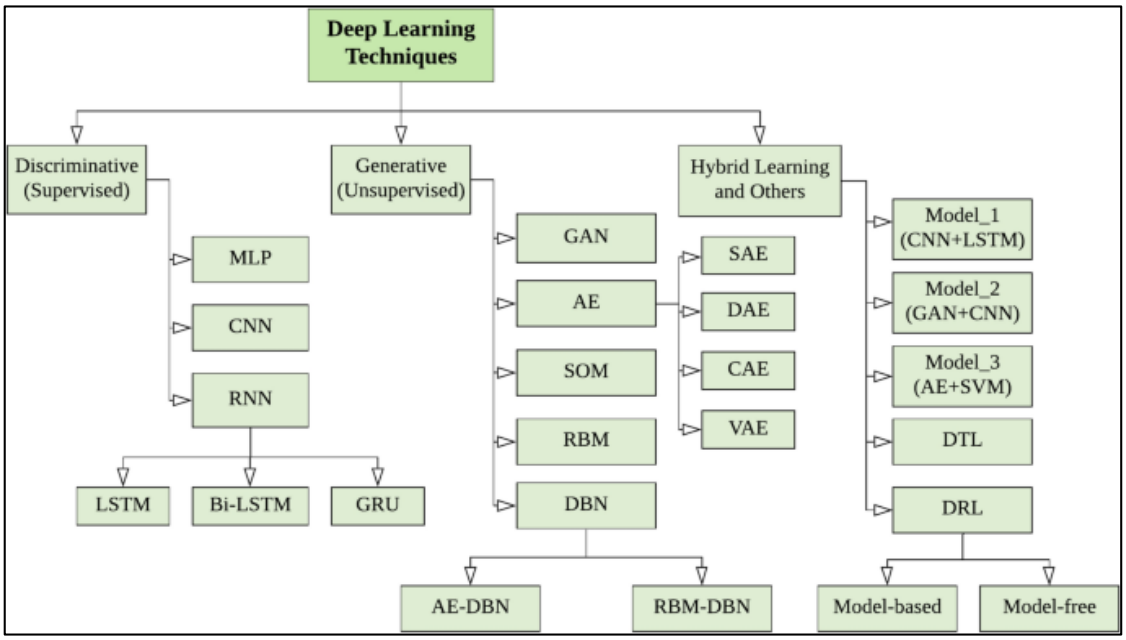


Figure 2: Types of deep learning algorithms

In this lecture, CNN and RNN variant LSTM networks are demonstrated with an example feature set AADP_PSSM derived from RNA binding protein sequences for classifying the dataset into RNA binding and non-binding proteins.

- **Python Programming Language**

  Python combines the power of general-purpose programming languages as well as the user-friendliness of domain-specific scripting languages like MATLAB or R. It has numerous modules for data loading, visualization, statistical analysis, natural language processing, image processing *etc*. Advantages of Python programming languages are:

  ✓ Ability to interact directly with the code, using a terminal or other tools like the Jupyter Notebook
  ✓ Open source, scalable
  ✓ Quick iteration and easy interaction
  ✓ Availability of an extensive range of machine learning libraries, such as Scikit-Learn, TensorFlow, Keras, PyTorch *etc.*
  ✓ Easy to create complex graphical user interfaces (GUIs) and web applications
  ✓ Easy to integrate into existing software.

- **Installation of Python**

  Python can be installed in two ways:

  1. **Install Python individually:** Download Python source codes from the site https://www.python.org/downloads/ based on the particular operation system
  2. **Install pre-packaged Python distribution:** Anaconda (https://www.anaconda.com/download/)

- **Jupyter Notebook**

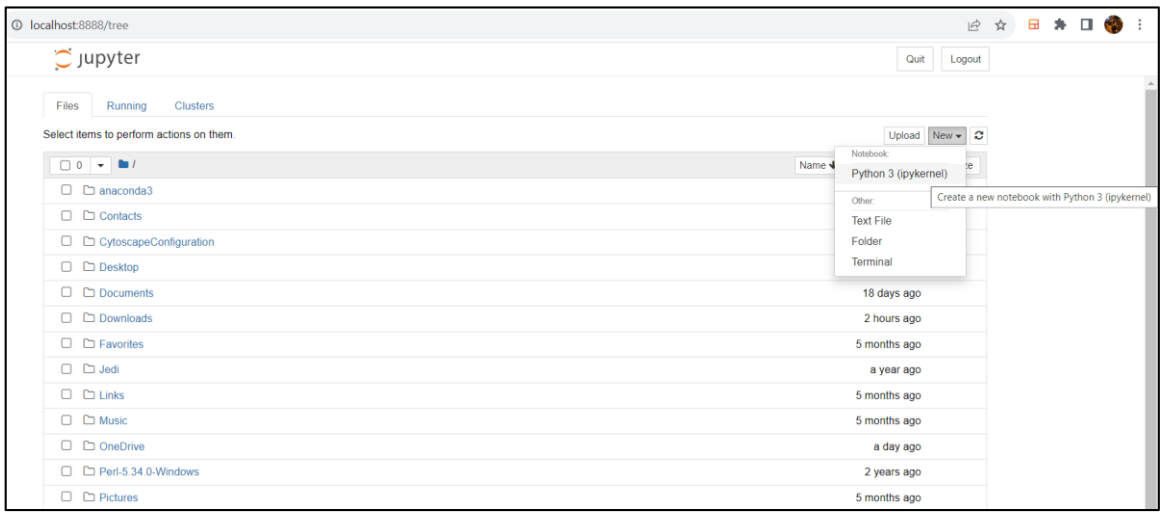
  It provides an interactive computational environment for developing Python based data science applications. It facilitates the arrangement of codes in various blocks so that changes in codes can be done easily and analysis results, *viz*., images, text, output *etc*. can be viewed in a step-by-step manner.

  ✓ **Installation of Jupyter Notebook**

  ```
  C:\>pip install notebook
  ```

  ✓ **Running Jupyter Notebook**

  ```
  C:\>jupyter notebook
  ```



(A)

(B)

Figure 3: (A) Running Jupyter Notebook in the local browser, (B) Starting a new Python3 kernel

- **Python libraries used for Deep Learning:**

For developing machine learning models, following Python libraries are needed to be installed:

✓ **NumPy:** It facilitates the data representation in the form of one-dimensional NumPy arrays and also contains functionality for multidimensional arrays, high-level mathematical functions such as linear algebra operations, Fourier transform, pseudorandom number generators *etc*.

✓ **Pandas:** It represents the tabular data as DataFrame similar to R Dataframe and Excel spreadsheet. The input data file of various formats like SQL, CSV, Excel *etc*. can be loaded to the Python programme using Pandas. It also facilitates easy data manipulation, SQL like queries and joins of tables.

✓ **Keras**: It is a high-level neural networks API written in Python. It includes functions required for developing deep learning models such as fully connected (dense) layers, convolutional layers, recurrent layers, normalization layers, *etc*. Keras uses different deep learning frameworks as backend engines, such as TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK).

✓ **Tensorflow**: TensorFlow is a widely used Python package for deep learning and machine learning tasks. It provides a framework to efficiently build, train, and deploy neural network models. With its powerful tools and libraries, TensorFlow simplifies the development of complex neural architectures while offering GPU acceleration for faster computations.

✓ **Scikit-Learn:** Most important Python library for data science and machine learning. It contains a wide range of machine learning algorithms like data preprocessing, classification, clustering, regression, dimensionality reduction, model selection, model evaluation *etc*.

✓ **Matplotlib:** This library is useful for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on.

Installation of these Python libraries can be done using the following code:

```
C:\>pip install <library_name>
```

➢ **Dataset**

Briefly write about **AADP_PSSM** feature set

1. Convolutional Neural Network (CNN)

Convolutional neural network is a deep learning algorithm whose function is similar to the visual cortex of the human brain. CNN consists of an input layer, multiple hidden layers and an output layer. The major architectural difference between CNN and ANN is that the former consists of 4 different types of hidden layers (Figure 4) performing specific functions like feature extraction, dimension reduction etc. before the final classification or prediction task. The first hidden layer of CNN is convolution layer which contains various convolution kernels or filters made up of matrices having sets of learnable parameters. CNN processes the data in the hierarchical pattern by segregating the large input matrix into smaller receptive fields by passing the convolution kernels in a sliding window approach. Dot products are performed between the kernel and the corresponding receptive field to generate the output feature map. Due to the selection of a specific kernel size, the chance of losing border information gets increased which is mitigated by introducing zero padding. Therefore, the dimension of the convolved feature map depends on the kernel size, stride length (i.e., number of steps the sliding window takes at a time) as well as the type of padding applied and it is calculated as,

$$w_{out} = \frac{w - m + 2p}{s} + 1 \qquad (1)$$

Where, $w_{out} \times w_{out} \times n$ = size of the convolved feature map, $w \times w \times 1$ = size of input feature matrix, w = length and width of input feature matrix, w_{out} = length and width of the output feature map, m = convolution kernel size, p = degree of padding, s = stride length and n = number of convolution kernels.

Convolution operation helps to extract high level, informative features by successfully capturing the special and temporal dependencies among the initial input features. After convolution operation, an activation function is applied on the convolved features to introduce non-linearity in the output feature map. Some widely used activation functions include sigmoid, hyperbolic tangent, Rectified Linear Unit (ReLU) *etc*. among which ReLU is highly preferred as it facilitates the training of the model without a significant reduction in classification accuracy. Features are further passed on to the pooling layer which facilitates the down-sampling of the feature matrix by aggregating the outputs from a set of neurons into a single neuron and thereby retaining significantly useful information. There are two most commonly used methods of pooling, *viz*., max pooling and average pooling. Apart from dimension reduction, pooling layer also helps to reduce complexity, increase the model efficiency and decrease the risk of overfitting. The final hidden layer of CNN is the fully connected dense layer which is similar to the hidden layer of a feed forward neural network where, neurons are fully connected with all the neurons of the preceding layer and

the succeeding output layer. The feature map generated from previous layers are flattened before sending to this layer and the classification task is performed. Depending on the type of task, i.e., regression or classification, specific loss functions such as mean absolute error (MAE), mean square error (MSE), binary cross entropy, categorical cross-entropy etc. are used and simultaneously optimizers like Root-mean-square prop (RMSprop), adaptive moment estimation (Adam) etc. are introduced in the model for optimizing the network parameters while minimizing the loss function in a short period of time. CNNs are considered as the regularized version of traditional multilayer perceptron. It can perform regularization in various ways such as penalization of parameters by weight decay during model training, skipping inter-node connectivity by setting higher dropout rates etc. and thereby can prevent the problem of model overfitting to some extent.

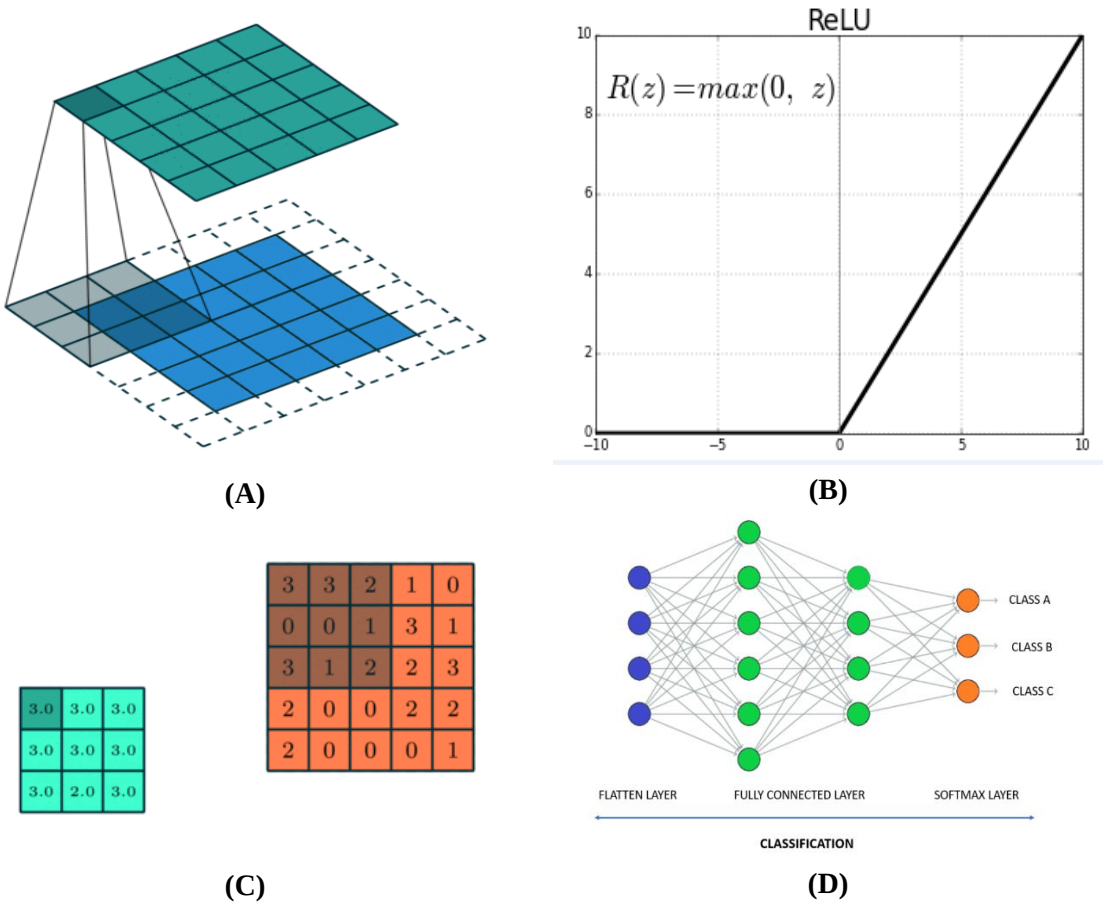


Figure 4: Hidden layers of CNN: (A) Convolution layer, (B) ReLU activation layer, (C) Max Pooling layer and (D) Fully connected dense layer

For classification tasks, various CNN-based classification models have been developed including LeNet-5, AlexNet, VGGNet, GoogLeNet, ResNet, MobileNet, EfficientNet *etc*. which are highly efficient in processing pixel data, image recognition, image segmentation, face and video recognition, natural language processing *etc*.
Steps of developing a CNN classifier for **AADP_PSSM** dataset are given below:

Step1: Load required libraries

```
In [1]: ### Required Python modules and functions
from __future__ import division, print_function
import _ctypes
import numpy as np
import pandas as pd
import os, time, math, statistics
import tensorflow as tf
from keras.utils import np_utils
from keras.utils.generic_utils import get_custom_objects
from keras.preprocessing import sequence
from keras import initializers
from keras.models import Sequential, load_model, Model
from keras.layers import ZeroPadding1D, Conv1D, MaxPooling1D, Activation, Dense, Flatten, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import matthews_corrcoef, auc, precision_recall_curve, roc_auc_score, roc_curve
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import h5py
```

Step 2: Load the data and save the features and labels in different variables

```
In [6]: # Input the dataset
data=pd.read_csv("C:/Users/ritwi/Desktop/AADP_PSSM.csv", header = None)
data1 = data.iloc[:,]
print (data1)

# Features = X and Labels = Y
X = data1.iloc[:, 1:]
Y = data1.iloc[:, :1]
print ("Shape of feature matrix: ", X.shape)
print ("Shape of label vector: ", Y.shape)
```

```
      0         1         2         3         4         5         6    \
0       1 -0.810384 -0.747178 -0.712190 -0.788939 -2.193002 -0.460497
1       1 -0.370876 -0.475540 -0.472127 -0.513083 -1.510808 -0.273038
2       1 -0.645062  0.236111 -0.825617 -0.600309 -1.695988 -0.745370
3       1 -0.502825 -0.435028 -0.875706 -0.966102 -1.977401 -0.672316
4       1 -0.756705 -0.423372 -0.950192 -1.191571 -1.770115 -0.680077
...   ...       ...       ...       ...       ...       ...       ...
4987    0 -0.749226 -2.095975 -1.730650 -2.250774 -3.625387 -1.789474
4988    0 -1.340050 -0.496222 -0.994962 -0.858942  0.161209 -1.269521
4989    0 -0.582278 -1.097046 -1.489451 -1.607595 -1.556962 -1.388186
4990    0 -0.770588 -1.305882 -0.294118 -1.247059 -2.547059 -0.570588
4991    0 -0.538462 -1.863462 -1.015385 -1.611538 -2.103846 -1.601923

           7         8         9    ...       411       412       413  \
0    -0.353273 -1.404063 -0.965011  ...  1.821469  0.974011  1.520904
1    -0.245734 -1.238908 -0.659841  ...  1.194761  0.137813  0.533030
2    -0.703704 -0.976852 -0.580247  ...  2.630603  1.333849  1.585781
3    -0.548023 -1.259887 -1.084746  ...  1.869318  1.000000  0.971591
4    -0.831418 -1.191571 -0.906130  ...  1.875240  0.804223  1.717850
...        ...       ...       ...  ...       ...       ...       ...
4987 -2.167183 -1.789474 -2.842105  ...  3.313665  3.872671  3.105590
4988 -1.173804 -1.614610  0.030227  ...  1.126263  1.492424  0.853535
4989 -1.443038 -1.666667 -1.181435  ...  1.677966  1.161017  1.597458
4990 -0.876471 -2.517647 -1.005882  ...  3.656805  1.183432  3.426036
4991 -1.634615 -2.180769 -1.130769  ...  1.876686  4.364162  3.529865

           414       415       416       417       418       419       420
0     2.102825  1.708475  0.882486  1.381921  4.024859  2.102825  1.392090
1     1.316629  0.961276  0.429385  0.373576  2.349658  0.992027  0.774487
2     2.826893 -0.177743 -0.429675  1.208655  3.802164  1.030912  2.197836
3     2.289773  1.357955  0.255682  1.107955  3.198864  1.829545  1.335227
4     1.637236  1.838772  0.445298  1.253359  2.761996  1.706334  1.850288
...        ...       ...       ...       ...       ...       ...       ...
4987  3.450311  3.375776  1.959627  2.667702  5.639752  3.630435  2.767081
4988  1.378788  1.439394  0.148990  0.871212  1.295455  1.707071  1.904040
4989  3.131356 -0.326271 -0.322034  0.432203  3.207627  2.279661  1.669492
4990  4.242604  7.763314  2.952663  4.307692  6.479290  0.733728  5.461538
4991  4.109827  5.778420  2.599229  4.082852  4.741811  2.641618  3.373796

[4992 rows x 421 columns]
Shape of feature matrix:  (4992, 420)
Shape of label vector:  (4992, 1)
```

Step 3: Compute the number of distinct classes and features

In [3]:
```
### Feature = X, Labels = Y...Label index starts from 0

Y=Y.to_numpy()
classes=np.unique(Y)
counter=0
for i in classes:
    Y[np.where(Y==i)]=counter
    counter+=1
features = X.shape[1]
### Counter = Total number of species
### Features = Total number of columns in the feature matrix

print ("Total number of classes: ",counter)
print ("Total number of features: ",features)
```

```
Total number of classes:  2
Total number of features:  420
```

Step 4: Define the CNN model architecture

In [5]:
```
### CNN architecture
def create_model(nb_classes,input_length):
    model = Sequential()
    model.add(Conv1D(5, 5, input_shape=(input_length, 1), padding="valid")) #input_dim
    model.add(Activation('relu'))
    model.add(MaxPooling1D(pool_size=2, padding="valid"))
    model.add(Conv1D(10, 5, padding="valid"))
    model.add(Activation('relu'))
    model.add(MaxPooling1D(pool_size=2, padding="valid"))
    model.add(Flatten())
    ##
    ##MLP
    model.add(Dropout(0.2))
    model.add(Dense(500, activation='relu'))
    #model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))
    model.compile(optimizer='adam',
      loss='binary_crossentropy',
      metrics=['accuracy'])
    return model
```

Step 5: Define the model training-validation validation procedure and set hyperparameters

In [6]:
```
### CNN Model training and validation specifications with validation split within model fit

epochs=100
def train_and_evaluate_model (model, train_feature, train_label, validation_feature, validation_label, nb_classes):
    global epochs

    train_feature = train_feature.reshape(train_feature.shape + (1,))
    train_label = np_utils.to_categorical(train_label, nb_classes)
    validation_label_bin = np_utils.to_categorical(validation_label, nb_classes)
    validation_feature = validation_feature.reshape(validation_feature.shape + (1,))

    start = time.time()
    history=model.fit(train_feature, train_label, epochs=epochs, batch_size=20, validation_data=(validation_feature, validation_
    total_time = time.time()-start
    #print("training time", total_time)



    training_scores = model.evaluate(train_feature,train_label,verbose=1)
    #print ("training loss and accuracy=", training_scores) #to print training accuracy

    prediction = model.predict(validation_feature,verbose = 1)

    validation_scores = model.evaluate(validation_feature, validation_label_bin,verbose= 1)
    #print ("validation loss and accuracy=", validation_scores) #to print validation accuracy
    print()


    return prediction, validation_label, history, total_time, model.evaluate(train_feature,train_label)[1]
```

Step 6: Train the model using training dataset and evaluate using validation set based on 5 fold cross-validation

```
### Model training with training and validation datasets and evaluation on test dataset done
### Dataframes to save fold wise average model performance using training and validation datasets
training_acc=[]
validation_acc=[]
precision=[]
recall=[]
specificity = []
F1_value=[]
train_time=[]
MCC_score=[]
AUPRC = []
AUROC = []


if __name__ == "__main__":
    n_folds = 5
    nb_classes=counter
    input_length = features
    X = np.array(X,dtype=float)
    Y = np.array(Y, dtype=int)
    print (X.shape)
    print (Y.shape)
    print ()

    ### K-fold Cross-Validation with Training and Validation dataset
    i = 1
    kfold = StratifiedKFold(n_splits=n_folds, shuffle=True)
    for train, validation in kfold.split(X, Y):
        print('fold {} is running'.format(i))
        print()
        model = None  # Clearing the NN.
        model = create_model(nb_classes, input_length)
        pred, validation_label, history,total_time,train_ac = train_and_evaluate_model(model, X[train], Y[train], X[validation],
        print ()

        ### K-fold Cross-Validation with Training and Validation dataset
        i = 1
        kfold = StratifiedKFold(n_splits=n_folds, shuffle=True)
        for train, validation in kfold.split(X, Y):
            print('fold {} is running'.format(i))
            print()
            model = None  # Clearing the NN.
            model = create_model(nb_classes, input_length)
            pred, validation_label, history,total_time,train_ac = train_and_evaluate_model(model, X[train], Y[train], X[validation],
            print ()

            #print('training accuracy is:{}'.format(train_ac))
            training_acc.append(train_ac)

            ### Saving predicted class probability matrix for Validation Dataset
            pred
            predict_probability = pd.DataFrame(pred)
            predict_probability.to_csv("C:/Users/ritwi/Desktop/Results_CNN/Validation Set_Prediction Probabilities_for fold_"+str(i)+

            ### Calculating actual and predicted class labels for Validation Dataset
            pred_1 = np.argmax(pred, axis=1)
            pred_1 = pred_1.reshape(pred_1.shape+(1,))
            print("predicted"+"\n", pred_1.shape)
            print("Actual"+"\n", Y[validation].shape)

            Y[validation] = np.array(Y[validation], dtype=int)
            Y[validation].shape
            pred_1 = np.array(pred_1, dtype=int)
            pred_1.shape

            #print ('Validation Accuracy:', accuracy_score(Y_train[validation], pred))
            validation_acc.append(accuracy_score(Y[validation], pred_1))

            #print ('Precision:', precision_score(Y_train[test], pred, average='macro'))
            precision.append(precision_score(Y[validation], pred_1))

            #print ('Recall:', recall_score(Y_train[test], pred, average='macro'))
            recall.append(recall_score(Y[validation], pred_1))

            # calculate and print specificity
            tn, fp, fn, tp = confusion_matrix(Y[validation], pred_1).ravel()
            print ("tn, fp, fn, tp:", tn, fp, fn, tp)
            specificity_score = tn / (tn+fp)
            specificity.append(specificity_score)

            #print ('F1 score:', f1_score(Y_train[test], pred, average='macro'))
            F1_value.append(f1_score(Y[validation], pred_1))

            #print ('MCC score:', matthews_corrcoef(actual, pred))
            MCC_score.append(matthews_corrcoef(Y[validation], pred_1))

            #print('time taken during training is:{}'.format(total_time))
            train_time.append(total_time)

            # Area under the Precision-Recall curve
            positive_class_probs = pred[:, 1]
            positive_class_precision, positive_class_recall, _ = precision_recall_curve(Y[validation], positive_class_probs)
            AUPRC_score = auc(positive_class_recall, positive_class_precision)
            #print ("AUPRC score", AUPRC)
            AUPRC.append(AUPRC_score)

            # Area under the ROC curve
            AUROC_score = roc_auc_score(Y[validation], positive_class_probs)
            #print ("AUROC score", AUROC)
            AUROC.append(AUROC_score)

            cr = classification_report(Y[validation], pred_1, output_dict=True)
            cr_report = pd.DataFrame(cr).transpose()
            cr_report.to_csv("CNN_07.08.2023/"+str(j)+"/Validation Set_classification report for fold_"+str(i)+".csv")

            cm = confusion_matrix(Y[validation], pred_1)
            conf_matrix = pd.DataFrame(cm).transpose()
            conf_matrix.to_csv("CNN_07.08.2023/"+str(j)+"/Validation Set_confusion matrix for fold_"+str(i)+".csv", index = False, he
```

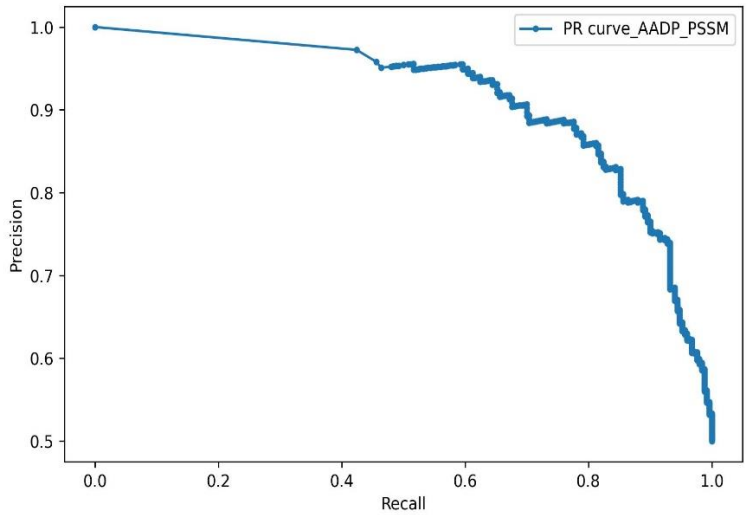
```
(4992, 420)
(4992, 1)

fold 1 is running

Epoch 1/100
200/200 [==============================] - 2s 7ms/step - loss: 0.5589 - accuracy: 0.7263 - val_loss: 0.5586 - val_accuracy:
0.7027
Epoch 2/100
200/200 [==============================] - 1s 6ms/step - loss: 0.4761 - accuracy: 0.7891 - val_loss: 0.4863 - val_accuracy:
0.7688
Epoch 3/100
200/200 [==============================] - 1s 5ms/step - loss: 0.4392 - accuracy: 0.8074 - val_loss: 0.4356 - val_accuracy:
0.8148
Epoch 4/100
200/200 [==============================] - 1s 5ms/step - loss: 0.4180 - accuracy: 0.8137 - val_loss: 0.4229 - val_accuracy:
0.8128
Epoch 5/100
200/200 [==============================] - 1s 6ms/step - loss: 0.4006 - accuracy: 0.8189 - val_loss: 0.4274 - val_accuracy:
```

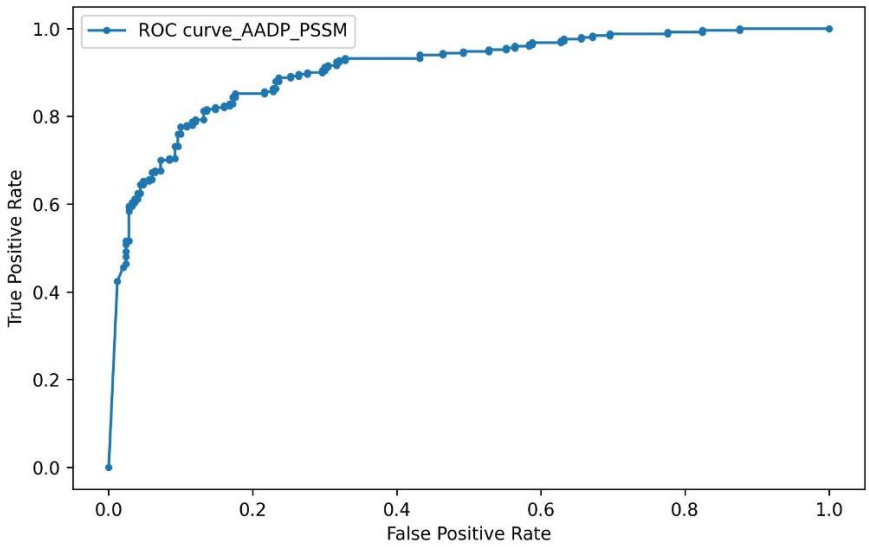
Step 7: Saving 5 fold evaluation results

```
In [9]: ### Fold Wise results saving

        training_data=pd.DataFrame()
        training_data['Fold No.']=range(1,n_folds+1)
        training_data['Training Time (In Seconds)']=train_time
        training_data['Training Accuracy']=training_acc
        training_data['Validation Accuracy']=validation_acc
        training_data['Precision']=precision
        training_data['Recall']=recall
        training_data['Specificity']=specificity
        training_data['F1 Score']=F1_value
        training_data['MCC Score']=MCC_score
        training_data['AUPRC']=AUPRC
        training_data['AUROC']=AUROC

        training_data.to_csv("C:/Users/ritwi/Desktop/Results_CNN/Validation Set_Evaluation metrices_5folds.csv",index=False)
```

Fold No.	Training Ti	Training Ac	Validation	Precision	Recall	Specificity	F1 Score	MCC Score
1	112.6659	0.978569	0.817575	0.832558	0.795556	0.839644	0.813636	0.635797
2	130.9579	0.980796	0.813126	0.795789	0.841871	0.784444	0.818182	0.627325
3	133.0448	0.979132	0.826281	0.859951	0.77951	0.873051	0.817757	0.655435
4	141.6041	0.979688	0.832962	0.834452	0.830735	0.835189	0.832589	0.665931
5	148.2354	0.978297	0.85412	0.856502	0.85078	0.857461	0.853631	0.708256

Step 8: PR and ROC curve for a particular fold





Step 9: Save the trained model

```
In [10]: ### Saving the CNN Model
         model.save("C:/Users/ritwi/Desktop/Results_CNN/CNN_model.h5")
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_20 (Conv1D)          (None, 416, 5)            30

 activation_30 (Activation)  (None, 416, 5)            0

 max_pooling1d_20 (MaxPoolin  (None, 208, 5)           0
 g1D)

 conv1d_21 (Conv1D)          (None, 204, 10)           260

 activation_31 (Activation)  (None, 204, 10)           0

 max_pooling1d_21 (MaxPoolin  (None, 102, 10)          0
 g1D)

 flatten_10 (Flatten)        (None, 1020)              0

 dropout_20 (Dropout)        (None, 1020)              0

 dense_20 (Dense)            (None, 500)               510500

 dropout_21 (Dropout)        (None, 500)               0

 dense_21 (Dense)            (None, 2)                 1002

 activation_32 (Activation)  (None, 2)                 0

=================================================================
Total params: 511,792
Trainable params: 511,792
Non-trainable params: 0
```

2. Long Short Term Memory (LSTM)

Recurrent Neural Network (RNN) is a type of DL network used to capture temporal dependency present in the sequencial data such as speech, text, time series, weather *etc.* RNN is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence.

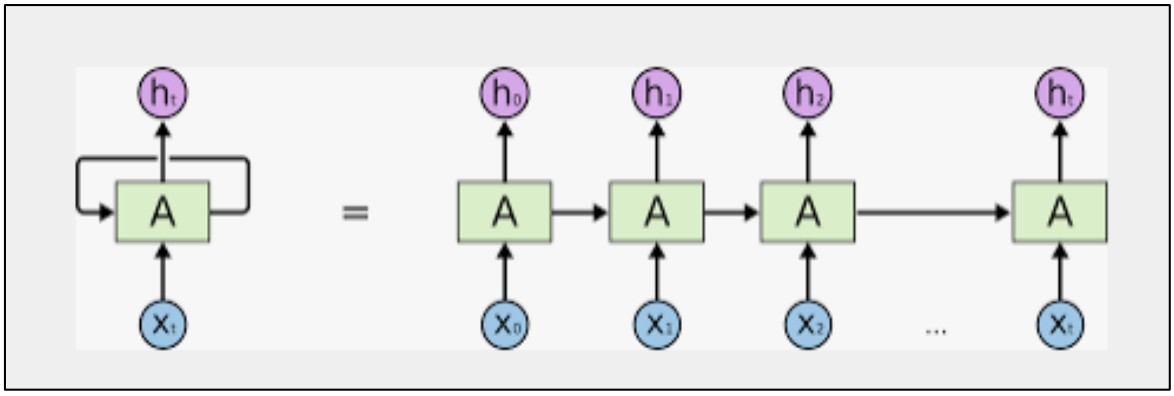


Figure 5: Architecture of RNN

In RNN, a sequence of input vectors x is processed by applying a recurrence formula at every time step:

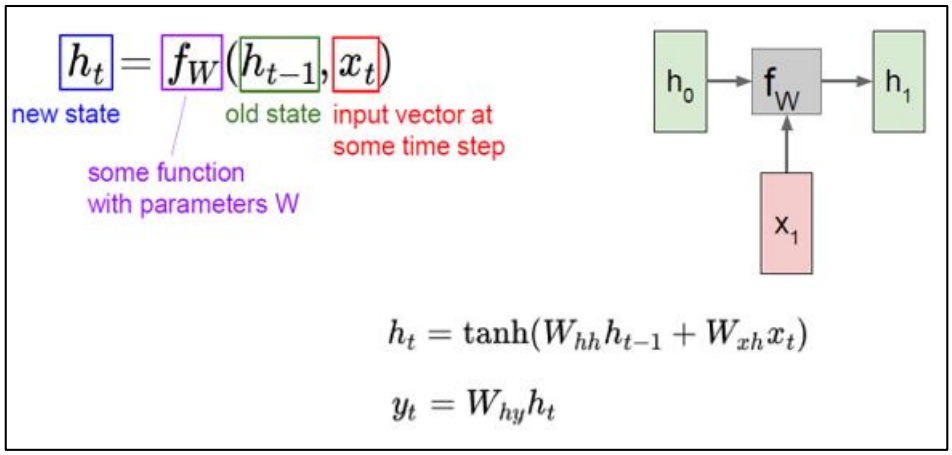


Figure 6: Input and output from RNN at time point *t*

LSTM is a variant of recurrent neural network (RNN) which is widely used for processing sequential information and has high potential in handing long-range dependencies among the input data. The traditional RNN model stores the information learnt from the previous outputs in the memory and gives prediction for the new input data. However, it is not capable of storing past information for a longer period of time to accurately predict the future instance. Similarly, RNN does not possess a clearly defined control mechanism about the amount of past information to be carried forward in the future. It is known as the "Vanishing-Gradient" problem and LSTM model is primarily designed to handle this major drawback by retaining and updating the important information over a long period of time. A LSTM unit consists of a cell known as "cell state" which functions as a conveyor belt and carries the information through the entire network. Apart from the cell, LSTM consists of three

gates (Figure 7), *viz*., input gate, forget gate and output gate, which regulates the flow of information by deciding which information would be passed into the cell and which part will be removed.

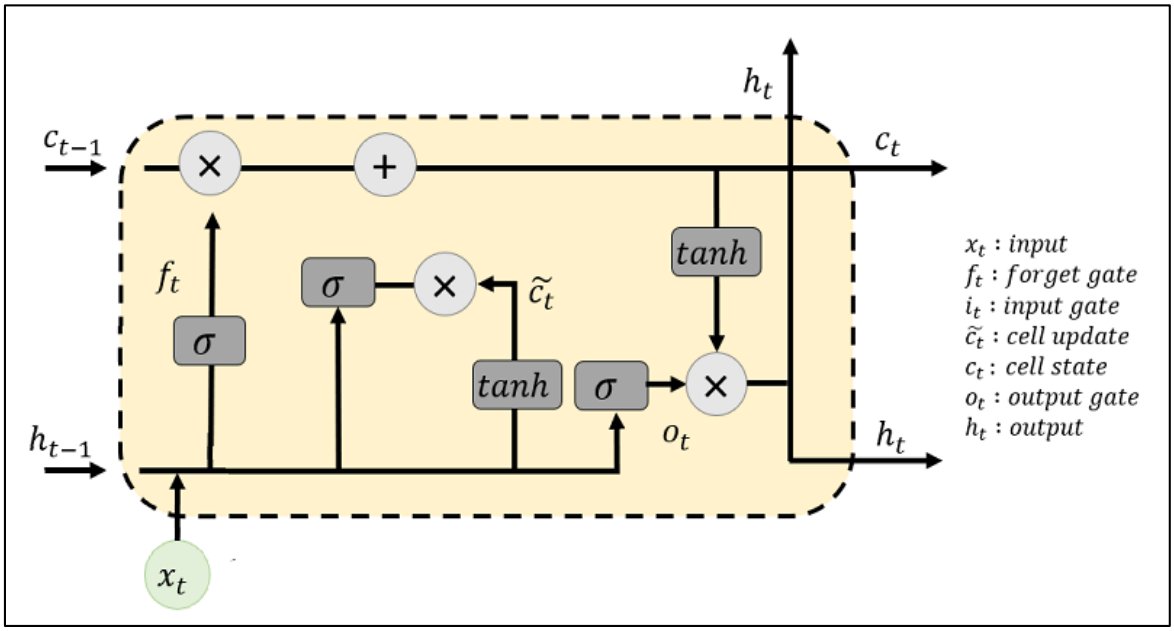


Figure 7: Gates of a LSTM unit

Input gate controls the inclusion of new information from the present input to the cell. It takes both current input and output from the previous hidden state, multiplies them and applies a sigmoid activation function to generate an output value between 0 and 1 which indicates the amount of information to be passed on. The forget gate performs similar functions as the input gate. However, it determines which information from the previous cell state will be removed. The output gate regulates the output from the LSTM cell. At first, it generates an activation value between 0 and 1 by employing a sigmoid activation function over the multiplication between the current input and the output from the previous hidden state. After that, the activation value is multiplied with the updates cell value to produce the final output from the LSTM model, thus effectively capturing the long-range dependencies of the periodical/ sequential data.

Due to the ability of handing sequential information, LSTM is highly efficient for speech recognition, text translation, sentiment analysis, language modelling, time series data analysis etc. LSTM models are also being widely used in various omics areas such as gene expression analysis, protein structure prediction, epigenetics, metagenomics, metabolomics etc.

Steps of developing a LSTM classifier for **AADP_PSSM** dataset are given below:

Step1: Load required libraries

```
### Required Python modules and functions
from __future__ import division, print_function
import _ctypes
import numpy as np
import pandas as pd
import os, time, math, statistics
import tensorflow as tf
from keras.utils import np_utils
from keras.utils.generic_utils import get_custom_objects
from keras.preprocessing import sequence
from keras import initializers
from keras.models import Sequential, load_model, Model
from keras.layers import Embedding, LSTM, Activation, Dense, Flatten, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import matthews_corrcoef, auc, precision_recall_curve, roc_auc_score, roc_curve
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import h5py
```

Step 2: Load the data and save the features and labels in different variables

(Same as CNN)

Step 2: Load the data and save the features and labels in different variables

(Same as CNN)

Step 4: Define the LSTM model architecture

```
In [4]: ### LSTM architecture
        def create_model(nb_classes, input_length):
            model = Sequential()
            model.add((LSTM(64, input_shape = (10, input_length), dropout = 0.2, recurrent_dropout=0.2)))
            model.add(Dropout(rate=0.5))
            model.add(Dense(500,activation='relu'))
            model.add(Dropout(rate=0.5))
            model.add(Dense(nb_classes, activation='softmax'))
            model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
            return model
```

Step 5: Define the model training-validation validation procedure and set hyperparameters

```
### LSTM Model training and validation specifications with validation split within model fit
epochs=100
def train_and_evaluate_model (model, train_feature, train_label, validation_feature, validation_label, nb_classes):
    global epochs

    a = train_feature.shape[1]/10
    b = validation_feature.shape[1]/10
    a = np.array(a, dtype=int)
    b = np.array(b, dtype=int)

    train_feature = train_feature.reshape(train_feature.shape[0], 10, a)
    train_label = np_utils.to_categorical(train_label, nb_classes)
    validation_label_bin = np_utils.to_categorical(validation_label, nb_classes)
    validation_feature = validation_feature.reshape(validation_feature.shape[0], 10, b)

    start = time.time()
    history=model.fit(train_feature, train_label, epochs=epochs, batch_size=20, validation_data=(validation_feature, validation_l
    total_time = time.time()-start
    #print("training time", total_time)



    training_scores = model.evaluate(train_feature,train_label,verbose=1)
    #print ("training loss and accuracy=", training_scores) #to print training accuracy

    prediction = model.predict(validation_feature,verbose = 1)

    validation_scores = model.evaluate(validation_feature, validation_label_bin,verbose= 1)
    #print ("validation loss and accuracy=", validation_scores) #to print validation accuracy
    print()


    return prediction, validation_label, history, total_time, model.evaluate(train_feature,train_label)[1]
```

Step 6: Train the model using training dataset and evaluate using validation set based on 5 fold cross-validation

(Same as CNN)

Some minor changes are needed in the respective code block.

```
if __name__ == "__main__":
    n_folds = 5
    nb_classes=counter
    input_length = features
    input_length_1 = features/10
    input_length_1 = np.array(input_length_1, dtype = int)

    X = np.array(X,dtype=float)
    Y = np.array(Y, dtype=int)
    print (X.shape)
    print (Y.shape)
    print ()

    ### K-fold Cross-Validation with Training and Validation dataset
    i = 1
    kfold = StratifiedKFold(n_splits=n_folds, shuffle=True)
    for train, validation in kfold.split(X, Y):
        print('fold {} is running'.format(i))
        print()
        model = None  # Clearing the NN.
        model = create_model(nb_classes, input_length_1)
        pred, validation_label, history,total_time,train_ac = train_and_evaluate_model(model, X[train], Y[train], X[validation],
        print ()
```

Step 7: Saving 5 fold evaluation results

(Same as CNN)

Step 8: PR and ROC curve for a particular fold

(Same as CNN)

Step 9: Save the trained model

```
In [7]: ### Saving the LSTM Model
        model.save("C:/Users/ritwi/Desktop/Results_LSTM/LSTM_model.h5")
```

```
Model: "sequential_84"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_84 (LSTM)               (None, 64)                27392

 dropout_168 (Dropout)        (None, 64)                0

 dense_168 (Dense)            (None, 500)               32500

 dropout_169 (Dropout)        (None, 500)               0

 dense_169 (Dense)            (None, 2)                 1002

=================================================================
Total params: 60,894
Trainable params: 60,894
Non-trainable params: 0
_________________________________________________________________
```

AI-DISC (Artificial Intelligence Based Disease Identification for Crops): A case study in Data Science

Chandan Kumar Deb

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012
chandan.deb@icar.gov.in

Introduction

Artificial Intelligence based Disease Identification System for Crops (AI-DISC) is a compressive mobile application developed by Division of Computer Application, ICAR-IASRI, New Delhi under NAHEP Component 2 project "Investment in ICAR Leadership for Agriculture Higher Education under National Agricultural Higher Education Project Component 2" and NASF Project "Artificial intelligence based mobile app for identification and advisory of maize diseases and insect pests" in collaboration of State Agricultural Universities for collecting, validating, annotating the images and identification of disease and pest of different crops. AI-DISC app uses National Image Base for Plant Protection (NIBPP) as its knowledge and image base and is hosted on Krishi Megh cloud infrastructure. It has different modules and types of users such as administrator, validator, domain experts and farmers. The AI-DISC mobile application facilitates the image-based automated plant disease and pest identification using artificial intelligence techniques. The AI-DISC mobile application is aimed to provide real-time crop protection solutions to the farmers on their fingertips.

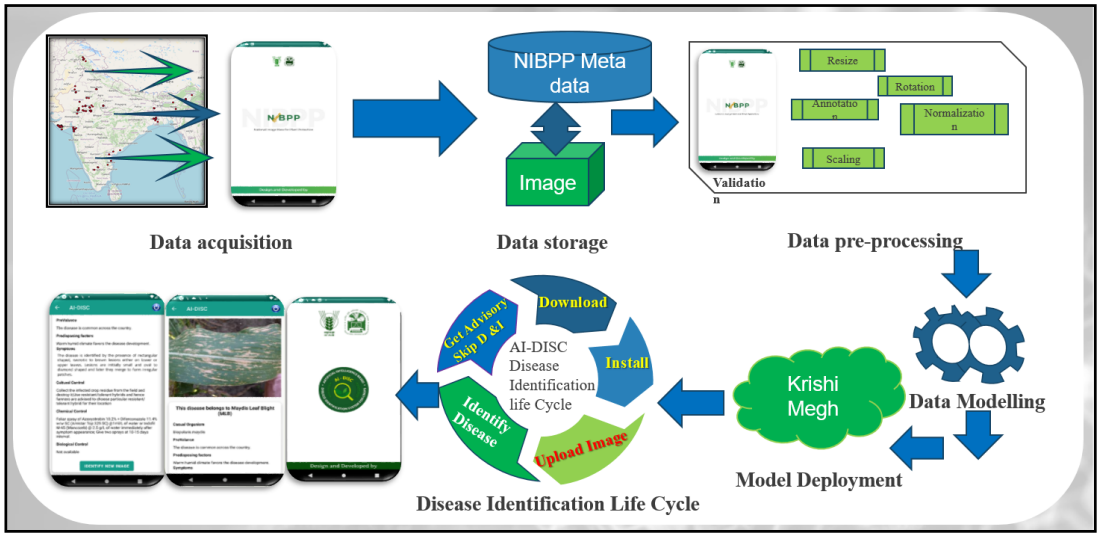


Fig 1: Development journey of AI-DISC

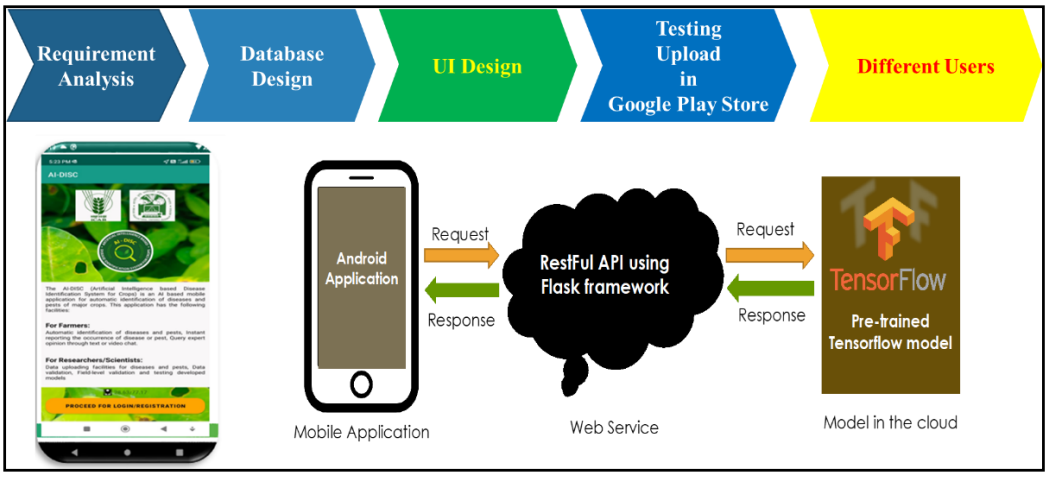


Fig. 2: Software development lifecycle of AI-DISC

Modules in AI-DISC:

- **Disease and Pest Identification Module (DPIM)**

Disease and Pest Identification Module (DPIM) is the main module of AI-DISC consisting of two manifestations: one for the farmers and another for the subject matter specialists. The farmers can use this module for uploading disease or pest infected images from the fields and easily diagnose the respective diseases or pests with the proper management practices. On the other hand, subject matter specialists can use this module to validate the developed models in the field conditions.

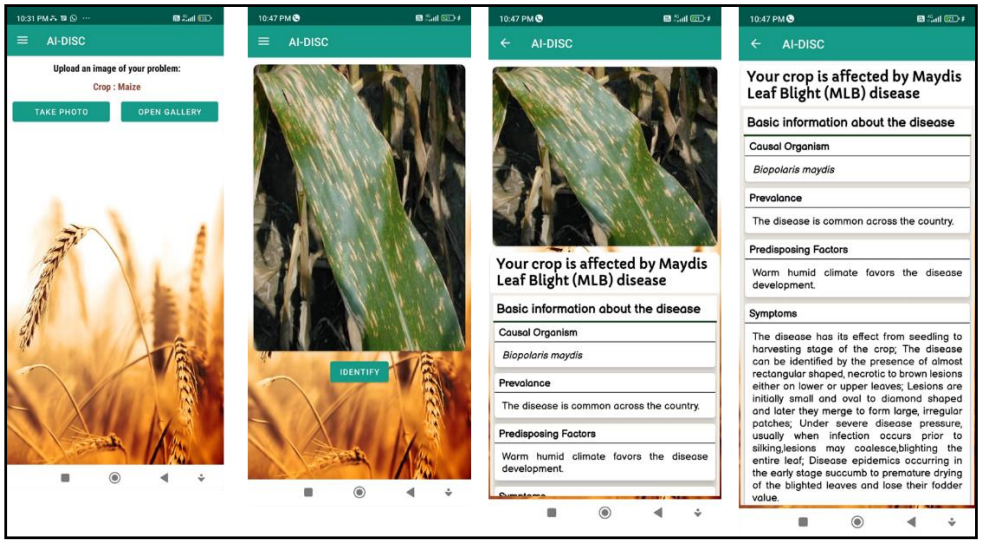


Fig 3. Disease Identification Module

- **User Management Module (UMM)**

User Management Module **(**UMM) has been created for providing access of the AI-DISC mobile application to the different types and levels of users. There are six types of the accounts: Farmer's account, Administrator account, Validator account, Data Entry Operator Account, Tester account and Domain Expert account. Each level and type of the users have a distinct responsibility.

Simple Steps to Identify Crop Diseases

Download AI-DISC android mobile app
(https://play.google.com/store/apps/details?id=com.ai.ai_disc)

Upload images with visible symptoms

Get the disease and advisory automatically

Fig 4. Login module of AI-DISC

- **Expert Forum for Farmers Module (EFFM)**

Expert forum for Farmers Module (EFFM) has been developed for the real-time communications between farmers and the domain experts. This module provides a discussion forum to the farmers for discussing crop protection related issues with the image uploading facility. It also has a video consultation interface with the domain experts for solutions.

- **Dashboard**

The Dashboard is being integrated with all types of account for visualizing the different activities and data status in a holistic manner. The module is very useful for the managers and policy maker.

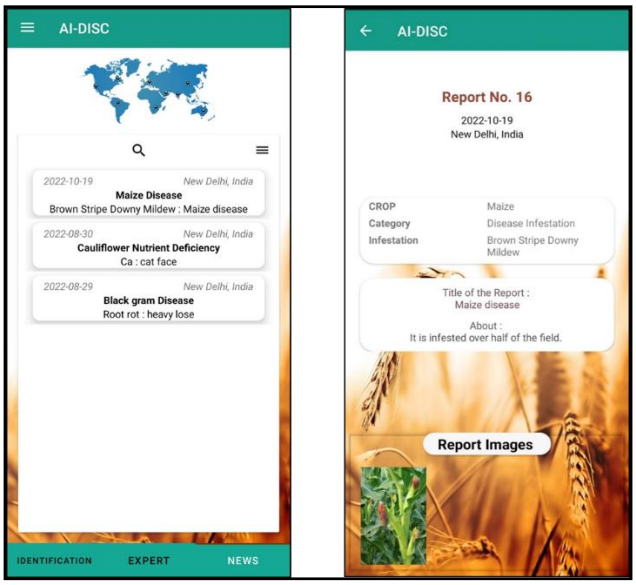


Fig. 5: Reporting module of disease occurrence

User types in AI-DISC

Type 1: Farmer's account

Farmers can create their account by registering themselves in a fast and easy way in the application. After registration, they can identify the diseases and pests by clicking and uploading the image of infected crop. The AI-DISC app will provide the management practices, cure and preventive measures for the identified disease/pest. In case, the app is not able to automatically identify the disease/pest, the query will be routed to the domain experts already registered in the app. The AI-DISC app also provide video consultation interface through Expert Forum for Farmers that allows the farmers to interact with the domain experts and to get the solutions regarding crop protection related issues faced by them.

Type 2: Administrator Account

The administrator account in AI-DISC is a top-level account consisting of master privileges. New information regarding crops, diseases, pests can be incorporated through this account. Other type of accounts can be created in AI-DISC through this admin account. A dashboard facility is available in this account from which status of uploaded data can be viewed.

Type 3: Validator Account:

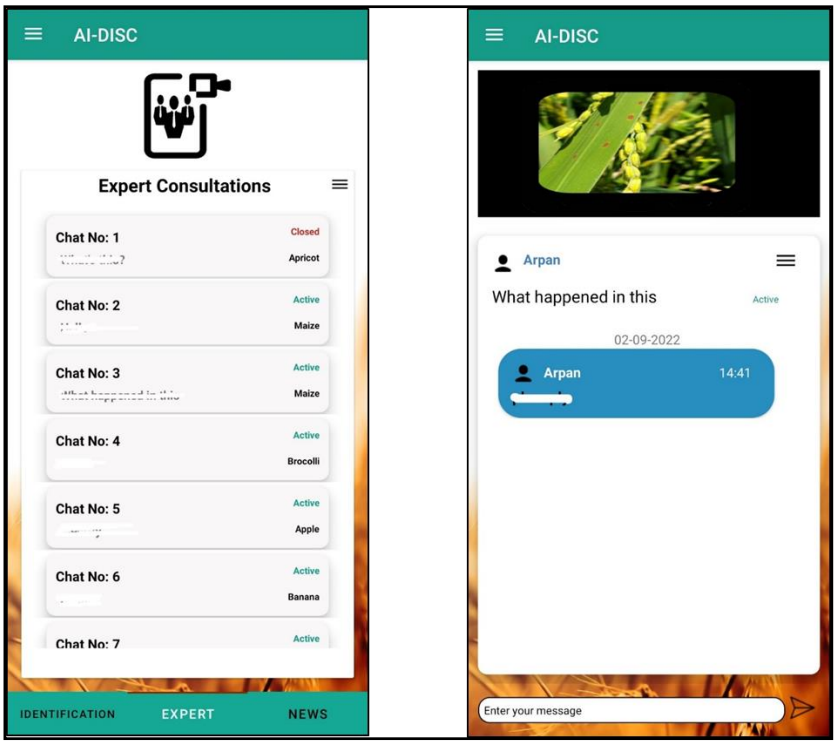
Validator account is one of the most important account of AI-DISC application. This account acts as a supervisor that can create the data entry operator account and can also approve or disapprove the images uploaded in the mobile application.

Type 5: Tester Account

Tester account has been created for the subject matter specialists for validating the developed models in the field conditions.

Type 6: Domain Expert Account

Domain Expert account has been created for the crop specific experts across the country as the part of the Expert Forum for resolving farmers crop protection related issues in the AI-DISC application.



Key Features of AI-DISC Mobile application

- Automatic image-based disease and pest identification module using deep learning models
- Advisory for crop protection related issues from domain experts via Expert Forum.
- Uploading facility of diseases and pests infected images along with the accurate metadata
- Annotation of the disease lesions and pests in the uploaded images
- Validation of the uploaded images by domain experts
- Efficient user management

Utility

- Automated image-based disease and pest identification in farmer's field
- National level repository of disease and pest infected images of crops
- Advisory for crop protection related issues from domain experts via Expert Forum.
- Location wise hotspot mapping for the diseases and pests and other policy-making tasks.

Case study in Data Science (Machine Learning)

Pankaj Das

Division of Sample Surveys

ICAR-Indian Agricultural Statistics Research Institute

Library avenue, Pusa, New Delhi-110012

Introduction:

Machine learning (ML) has found extensive applications across various industries, revolutionizing the way organizations make decisions and optimize processes. This case study focuses on the implementation of machine learning in the context of predictive maintenance, showcasing its impact on efficiency, cost savings, and overall operational effectiveness.

Objective:

The primary objective of this case study is to demonstrate how machine learning algorithms can be employed to predict equipment failures and schedule maintenance activities proactively.

Case study:

The study shows how ML models are applied to predict crop yield using their morphological characters. The resin yield data were used for the demonstration of the study. Artificial neural network (ANN) model was taken as a prediction model in the study. Height, canopy, tree girth, FD, FDI and CD were the morphological characters that used as explanatory variables used for ANN model building. R code was written to apply ANN model.

R Code:

Data should be incorporated in the R environment. There are many methods for uploading in R. here one of the most common method i.e. file upload in .csv format was used. The syntax for this

```
#Dataset upload

Data =read.csv(file.choose())

#to see the variables

colnames(Data)
```

One of the crucial steps once the dataset has been uploaded is to check it. It is essential to determine whether the dataset has any discrepancies. A visual method was employed to examine the dataset. Descriptive statistics may also be employed in addition to this.

```
# To plot data in boxplot

boxplot(Data)

#To see Descriptive statistics

Summary(Data)
```

Data splitting is used in the construction of machine learning models to generate prediction models and assess the accuracy of the created models. Data are typically divided into training (80% of data) and testing sets (20% data). Random splitting is used to remove any type of bias.

```
# Random sampling for data splitting

samplesize = 0.80 * nrow(Data)

set.seed(100)

index = sample( seq_len ( nrow ( Data) ), size = samplesize )

# Creation of training and test set

datatrain = Data [ index, ]

datatest = Data [ -index, ]
```

Normalization in machine learning is essential to ensure that all input features are on a similar scale, preventing algorithms from being influenced by the magnitude of individual features. It facilitates efficient convergence in optimization algorithms, enhances model interpretability, and addresses issues related to sensitivity, distance calculations, and numerical stability. Minimax normalization is used for the study.

```
#Scale data for neural network (minimax normalization)

max = apply(Data, 2 , max)

min = apply(Data, 2 , min)

scaled = as.data.frame(scale(Data, center = min, scale = max - min))
```

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain. Comprising interconnected nodes organized into layers, ANNs are designed for machine learning tasks, learning patterns and relationships within data through iterative training processes. Input nodes receive data, hidden layers process it through weighted connections, and output nodes produce predictions or classifications. The network adjusts its weights during training, optimizing its ability to generalize to new, unseen data. ANNs find extensive applications in tasks such as image recognition, natural language processing, and predictive analytics. The neuralnet R package is used to fit ANN models.

```
## Fit neural network

#installation of R package

install.packages("neuralnet")

# load library

library(neuralnet)

# creating training and test set

trainNN = scaled[index , ]

testNN = scaled[-index , ]
```

In artificial neural network (ANN) model fitting, the process involves adjusting the weights and biases of the network to minimize the difference between predicted and actual outputs. This optimization, often achieved through techniques like backpropagation and gradient descent, aims to ensure the neural network learns to accurately represent the underlying patterns in the training data. In present study, one hidden layer with 3 hidden neurons is used for yield prediction. The resilient backpropagation is used as training algorithm.

```
# Model building of ANN

set.seed(100)

Model_NN = neuralnet(RESIN.Y ~ TREE.D+ HT + BH + NOB + AGE, trainNN, hidden =3, linear.output = T )

# plot neural network

plot(Model_NN)
```

To evaluate the effectiveness of the model, the weights between the inputs, hidden layers, and outputs are generated once the neural network model's errors have been fitted.

```
#Generate the error of the neural network model, along with the weights between the inputs, hidden layers, and outputs:

Model_NN $result.matrix[1,]

#Test the resulting output

temp_test <- subset(testNN, select = c("TREE.D","HT", "BH", "NOB", "AGE"))

head(temp_test)

nn.results <- compute(Model_NN, temp_test)

results <- data.frame(actual = testNN$RESIN.Y, prediction = nn.results$net.result)

#plot of results

plot(results)

cor(results)

#confussion matrix

roundedresults<-sapply(results,round, digits=0)

roundedresultsdf=data.frame(roundedresults)

attach(roundedresultsdf)

#prediction table

table(actual,prediction)
```

Checking in-sample performance evaluates how well a model fits the training data, assessing its ability to capture patterns within the dataset. However, this alone does not indicate the model's ability to generalize to new, unseen data. Out-sample performance, evaluated on a separate dataset, provides a realistic measure of a model's generalization capability and helps detect issues such as overfitting. Both

in-sample and out-sample evaluations are essential for tuning hyperparameters during model development and selecting the final model based on its real-world performance. The process ensures that a model not only fits the training data effectively but also has the capacity to make accurate predictions on novel instances. Ultimately, assessing both in-sample and out-sample performance is fundamental for creating reliable and generalizable machine learning models. Root Mean Square Error (RMSE), Mean absolute deviation (MAD), Mean absolute percent error (MAPE), Maximum error (ME), Coefficient of determination (R square) and accuracy are computed both training and testing dataset as performance measure.

Insample performance check

```
predict_trainNN = compute(Model_NN, trainNN[,c(1:5)])

predict_trainNN2 = (predict_trainNN$net.result * (max(Data$RESIN.Y) - min(Data$RESIN.Y))) + min(data_sl$RESIN.Y)

#Accuracy measure in traindata

actual_train=datatrain$RESIN.Y

comparison2=data.frame(predict_trainNN2 ,actual_train)

deviation2=((actual_train-predict_trainNN2)/actual_train)

comparison2=data.frame(predict_trainNN2 ,actual_train,deviation2)

accuracy_train=1-abs(mean(deviation2))

accuracy_train

#Calculate Root Mean Square Error (RMSE)

RMSE.NN_Tr = (sum((datatrain$RESIN.Y - predict_trainNN2)^2) / nrow(datatrain)) ^ 0.5

# Calculate Mean absolute deviation (MAD)

MAD_Tr=(sum(abs(datatrain$RESIN.Y-predict_trainNN2))/nrow(datatrain))

# Calculate Mean absolute percent error (MAPE)

d3_tr=sum((abs(datatrain$RESIN.Y-predict_trainNN2))/datatrain$RESIN.Y)

MAPE_Tr=d3_tr/nrow(datatrain)

# Calculate maximum error

ME_tr=max(abs(datatrain$RESIN.Y-predict_trainNN2))

# Calculate coefficient of determintaion

RSquare.NTr=(cor(datatrain$RESIN.Y,predict_trainNN2))^2

#Print of the computed performance measures

print(paste(RMSE.NN_Tr,MAD_Tr,MAPE_Tr,ME_tr,RSquare.NTr,accuracy_train,NN3$result.matrix[1,]))
```

Accuracy measure in test data

```
## Prediction using neural network

predict_testNN3 = compute(Model_NN, testNN[,c(1:5)])

#Rescaling of output

predict_testNN3 = (predict_testNN3$net.result * (max(data_sl$RESIN.Y) - min(data_sl$RESIN.Y))) + min(data_sl$RESIN.Y)

#Accuracy calculation

actual1=datatest$RESIN.Y

comparison=data.frame(predict_testNN3,actual1)

deviation=((actual1-predict_testNN3)/actual1)

comparison2=data.frame(predict_testNN3,actual1,deviation)

accuracy=1-abs(mean(deviation))

#Plot the model

plot(Model_NN)

#Plot predict vs actual value of testing dataset

plot(datatest$RESIN.Y, predict_testNN3, col='blue', pch=16, ylab = "Predicted value of ANN model", xlab = "real value of yield")

abline(0,1)

# Calculate Root Mean Square Error (RMSE)

RMSE.NN3 = (sum((datatest$RESIN.Y - predict_testNN3)^2) / nrow(datatest)) ^ 0.5

#Mean absolute deviation (MAD)

MAD3=(sum(abs(datatest$RESIN.Y-predict_testNN3))/nrow(datatest))

#Mean absolute percent error (MAPE)

d3=sum((abs(datatest$RESIN.Y-predict_testNN3))/datatest$RESIN.Y)

MAPE3=d3/nrow(datatest)

#maximum error

ME3=max(abs(datatest$RESIN.Y-predict_testNN3))

#R square calculation

RSquare.N=(cor(datatest$RESIN.Y,predict_testNN3))^2

#Printing all calculated measures

print(paste(RMSE.NN3,MAD3,MAPE3,ME3,RSquare.N,accuracy,NN3$result.matrix[1,]))
```

Case study in Data Science (Statistical modelling)

Pankaj Das

Division of Sample Surveys

ICAR-Indian Agricultural Statistics Research Institute

Library avenue, Pusa, New Delhi-110012

Introduction:

Modeling in time series analysis involves applying mathematical and statistical techniques to understand, represent, and forecast patterns in sequential data points over time. Here is a concise overview of the key steps in time series modeling:

Data Collection:

- Gather time-stamped data, where each observation is associated with a specific time or temporal sequence. This can include financial data, stock prices, temperature readings, or any other information that varies over time.

Exploratory Data Analysis (EDA):

Conduct EDA to understand the underlying patterns, trends, and seasonality in the time series. Visualization tools such as line plots, histograms, and autocorrelation functions can be used to gain insights.

Data Preprocessing:

Clean the data by handling missing values, outliers, and inconsistencies. Normalize or scale the data if necessary. Create lag features or rolling averages to capture temporal dependencies and make the series stationary if needed.

Model Selection:

Choose an appropriate time series model based on the characteristics of the data. Common models include:

- Autoregressive Integrated Moving Average (ARIMA):Suitable for stationary time series data.

- Seasonal Decomposition of Time Series (STL): Useful for decomposing time series into trend, seasonal, and residual components.

- Machine Learning Models (e.g., LSTM): Effective for capturing long-term dependencies in sequential data.

Besides this, ARCH (Autoregressive Conditional Heteroskedasticity) and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models are statistical models used to analyze and model time series data with changing volatility over time. These models are particularly prevalent in financial econometrics for modeling the volatility of financial returns.

- ARCH Model (Autoregressive Conditional Heteroskedasticity): ARCH models are designed to capture the changing variance (heteroskedasticity) in a time series.

- GARCH Model (Generalized Autoregressive Conditional Heteroskedasticity): GARCH models extend ARCH models by introducing lagged conditional variances in addition to lagged squared errors.

Training and Validation:

- Split the dataset into training and validation sets. Train the selected model on the training set and validate its performance on the validation set. Adjust model hyperparameters to optimize forecasting accuracy.

Model Evaluation:

- Assess the model's performance using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or others, depending on the specific characteristics and goals of the analysis.

Forecasting/Prediction:

- Deploy the trained model to make future predictions. Forecasting can be done for a short-term horizon or extended to longer-term predictions, depending on the objectives.

Interpretability and Insights:

- Interpret the model's outputs to gain insights into the factors influencing the time series. Understand how external factors, seasonality, or trends contribute to the forecasted values.

Challenges and Limitations:

- Address challenges encountered during the modeling process, such as data quality issues, unforeseen events, or limitations in long-term forecasting accuracy.

Inshort the time series modeling process, emphasizing the success of the model in capturing and forecasting patterns in the sequential data. Discuss the practical applications and potential avenues for further improvement.

In the present case ARIMA and GARCH model was fitted in a time series data.

ARIMA model fitting:

Fitting an ARIMA (AutoRegressive Integrated Moving Average) model involves selecting appropriate model parameters and estimating coefficients to capture the temporal patterns and dynamics of a time series. Here are the key steps in ARIMA model fitting:

1. Stationarity Assessment:

- Check if the time series is stationary. Stationarity is a key assumption for ARIMA models. If the series is non-stationary, perform differencing until stationarity is achieved. The number of differences needed is denoted as \(d\).

2. Identification of Model Order (p, d, q):

- Autoregressive Order (p): Determine the number of autoregressive terms by examining the autocorrelation function (ACF) plot. Significant lags suggest potential autoregressive terms.

- Integrated Order (d): The differencing order determined in step 1.

- Moving Average Order (q): Determine the number of moving average terms by examining the partial autocorrelation function (PACF) plot. Significant lags suggest potential moving average terms.

3. Model Specification:

- Define the ARIMA model based on the identified order (p, d, q). The notation is ARIMA(p, d, q).

4. Parameter Estimation:

- Use methods like maximum likelihood estimation (MLE) to estimate the model parameters. This involves finding the values for the autoregressive (AR) coefficients, differencing term, and moving average (MA) coefficients that maximize the likelihood of observing the given data.

5. Model Diagnostic Checking:

- Conduct diagnostic tests to assess the adequacy of the model. Common checks include examining residuals for randomness, performing Ljung-Box tests for autocorrelation in residuals, and ensuring that residuals are normally distributed.

6. Model Refinement:

- If diagnostic checks reveal issues, consider refining the model. This may involve adjusting the model order, incorporating exogenous variables, or exploring alternative models.

7. Forecasting:

- Once the ARIMA model is fitted and validated, use it for forecasting future values. Forecast intervals can be generated to indicate the uncertainty associated with predictions.

8. Model Evaluation:

- Evaluate the performance of the ARIMA model using appropriate metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or others depending on the specific goals of the analysis.

9. Iterative Process:

- ARIMA model fitting can be an iterative process. If initial model results are not satisfactory, refine the model based on diagnostic checks and re-estimate parameters.

10. Deployment and Monitoring:

- Deploy the final ARIMA model for real-time forecasting if applicable. Implement monitoring to track its performance over time and update the model as needed.

Fitting an ARIMA model requires a balance between capturing the temporal dynamics of the time series and avoiding overfitting. Regular diagnostic checks and model refinement contribute to building a robust and reliable ARIMA model for time series analysis and forecasting.

R code for ARIMA model

```
#Data generation
set.seed(250)
timeseries=arima.sim(list(order = c(1,1,2), ma=c(0.32,0.47), ar=0.8), n = 50)+20
plot(timeseries)
#Model Specification
acf(timeseries, lag.max=100)
pacf(timeseries, lag.max=100)
#Differencing of data to remove nonstationary behaviour
diff(timeseries)
plot(diff(timeseries),type="b")
#ARMA(0,2)
#ARMA(1,0)
#ARMA(1,2)
#That is, for the original time series, we propose three ARIMA models, ARIMA(0,1,2) ARIMA(1,1,0) and ARMA(1,1,2)
#Model building and diagnostic
## Partition into train and test
#The retain last 10 observaitons for forecasting and use first 40 observations to fit the models
train_series=timeseries[1:40]
test_series=timeseries[41:50]
## make ARIMA models
arimaModel_1=arima(train_series, order=c(0,1,2))
arimaModel_2=arima(train_series, order=c(1,1,0))
arimaModel_3=arima(train_series, order=c(1,1,2))
## look at the parameters
print(arimaModel_1);print(arimaModel_2);print(arimaModel_3)
#third is best based on likelihood and aic.
#Prediction
Predict1=predict(arimaModel_1, 10)
```

```
Preidict2=predict(arimaModel_2, 10)
predict3=predict(arimaModel_3, 10)
#For Forecasting
forecast1=forecast(arimaModel_1, 10)
forecast2=forecast(arimaModel_2, 10)
forecast3=forecast(arimaModel_3, 10)
#Plot of forecasted value
plot(forecast1, main = "Graph with forecasting",
    col.main = "darkgreen")
#Auto ARIMA model fitting
library(forecast)
fit3 = auto.arima(timeseries)
tsdiag(fit3)
```

GARCH model fitting

GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model fitting involves preparing time series data, identifying the model order, estimating parameters through techniques like maximum likelihood estimation, and performing diagnostic checks. The model is then used for forecasting future volatility, with evaluation metrics assessing its performance. Refinement may be needed based on diagnostics, and the final GARCH model can be deployed for real-time forecasting, monitored, and updated as necessary. GARCH models are especially valuable for capturing the changing nature of volatility, commonly applied in financial time series analysis. The process of GARCH model fitting is similar to ARIMA model.

Data Preparation:

Collect time series data, typically financial returns, where volatility varies over time.

Model Identification:

Examine the time series to identify the order of the GARCH model (p, q), where p is the order of autoregressive conditional heteroskedasticity, and q is the order of moving average conditional heteroskedasticity.

Parameter Estimation:

Use maximum likelihood estimation (MLE) or other optimization techniques to estimate the model parameters, including the ARCH and GARCH coefficients.

Model Checking:

Perform diagnostic tests, such as the Ljung-Box test, to ensure that the model adequately captures volatility patterns. Examine residuals for autocorrelation and heteroskedasticity.

Forecasting:

Utilize the fitted GARCH model for forecasting future volatility. Forecast intervals can be generated to quantify the uncertainty associated with volatility predictions.

Model Evaluation:

Assess the performance of the GARCH model using appropriate metrics, such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE), to evaluate how well the model captures volatility dynamics.

Refinement and Iteration:

If model diagnostics reveal issues, refine the model by adjusting the order or exploring alternative specifications. Iteratively re-estimate parameters and check model adequacy.

Deployment and Monitoring:

Deploy the final GARCH model for real-time volatility forecasting if applicable. Implement monitoring to track its performance over time and update the model as needed.

R Code for GARCH model fitting

```
###installing and loading multiple packages

list.packages<-c("fGarch", "PerformanceAnalytics","rugarch","tseries","xts","FinTS")

new.packages <- list.packages[!(list.packages %in% installed.packages()[,"Package"])]

if(length(new.packages)) install.packages(new.packages)

#Loading Packages

invisible(lapply(list.packages, require, character.only = TRUE))

library(quantmod)

library (rugarch)

library (xts)

library(PerformanceAnalytics)

library(tidyverse)

library (dplyr)

library (tseries)

#Data upload

df <- getSymbols("TSLA", from="2010-01-01", to="2020-12-31")

chartSeries(TSLA)

head(TSLA)

#for specific period

chartSeries(TSLA["2020-12"])
```

```
#the calculation of the daily return of the price and display it. For the return calculation we use the function CalculateReturns().

return=CalculateReturns(TSLA$TSLA.Adjusted)

#remove first row as it doesnot contain any value

return=return[-c(1),]

#plot the result

plot(return)

chart.Histogram(return, methods = c('add.density','add.normal'),colorset = c('blue','red','black'))

legend("topright",legend = c("return","kernel","normal dist"),fill=c('blue','red','black'))

#volatility

sd(return)

sqrt(252)*sd(return["2020"])

chart.RollingPerformance(R=return["2010::2020"],width=22,FUN="sd.annualized",scale=252,main= "TESLA's monthly volatility")

#Garch model fitting arma varaince (0,0)

garch_spec = ugarchspec(variance.model=list(model="sGARCH", garchOrder=c(1,1)), mean.model=list(armaOrder=c(0,0)))

#fit model

fit_garch <- ugarchfit(data=return, spec = garch_spec, out.sample = 20)

fit_garch
```

Hands on Image Classification using Convolutional Neural Networks

Md. Ashraful Haque

Scientist, ICAR-IASRI

```
# TensorFlow and tf.keras
import tensorflow as tf
# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import utils
from sklearn.metrics import classification_report, confusion_matrix
```

This practical uses the Fashion MNIST dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as seen here:

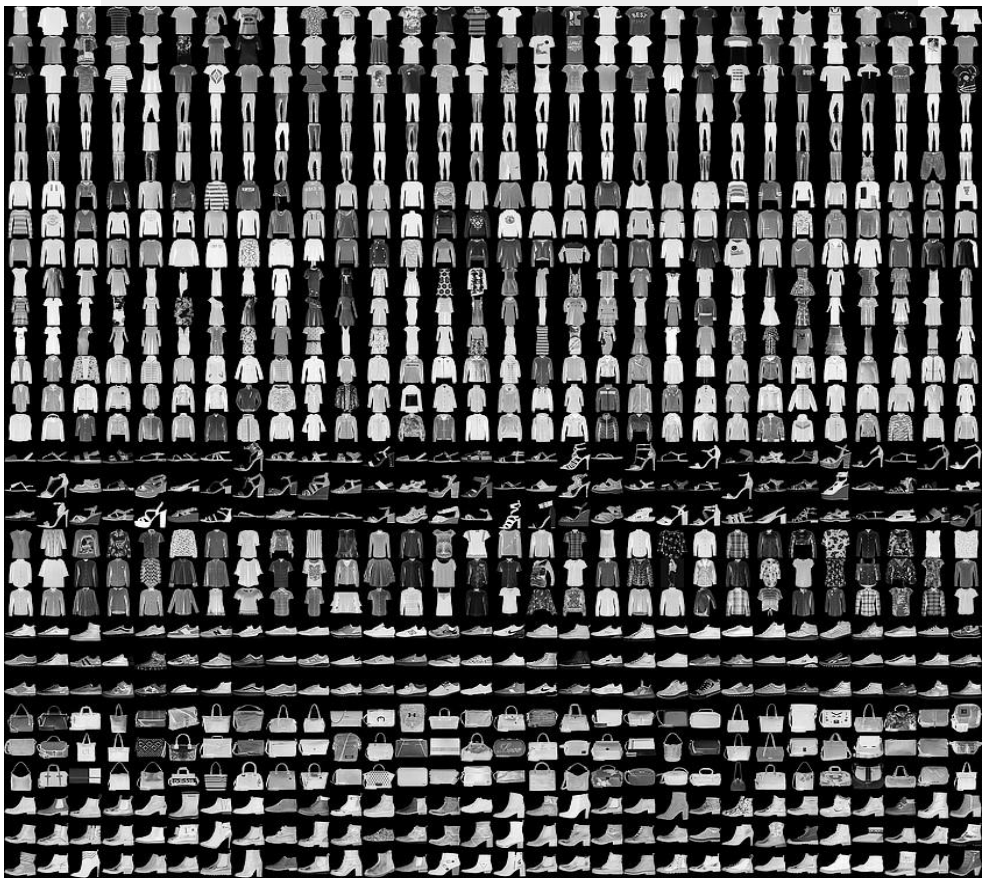


Figure 1: Fashion-MNIST sample

Fashion MNIST is intended as a drop-in replacement for the classic MNIST dataset—often used as the "Hello, World" of machine learning programs for computer vision. The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing you'll use here.

This practical uses Fashion MNIST for variety, and because it's a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. You can access the Fashion MNIST directly from TensorFlow. Import and load the Fashion MNIST data directly from TensorFlow:

```
## Loading the data
fashion_mnist = tf.keras.datasets.fashion_mnist
```

```
## traning and testing dataset
(train_images, train_labels),(test_images, test_labels) =
fashion_mnist.load_data()
```

```
## Explore the dataset
print(train_images.shape)
print(test_images.shape)
print(len(train_labels))
print(len(test_labels))
```

```
  (60000, 28, 28)
  (10000, 28, 28)
  60000
  10000
```

Loading the dataset returns four NumPy arrays:

- The train_images and train_labels arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the test_images , and test_labels arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The *labels* are an array of integers, ranging from 0 to 9. These correspond to the *class* of clothing the image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

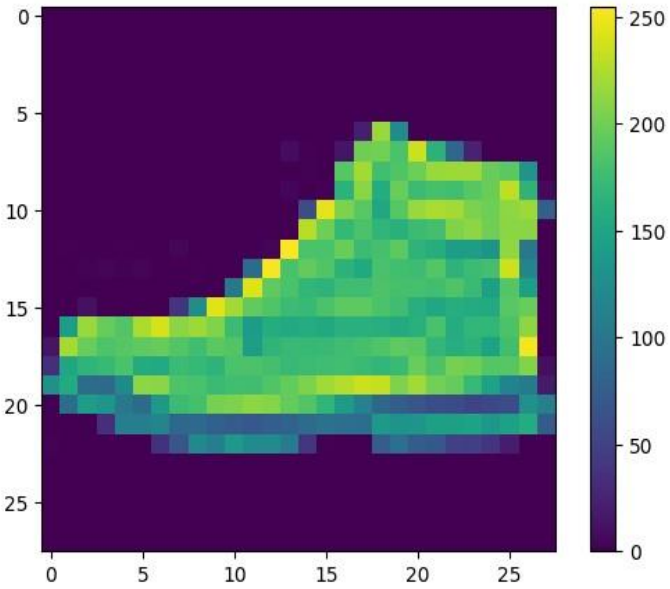
Each image is mapped to a single label. Since the *class names* are not included with the dataset, store them here to use later when we evaluate the model.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Preprocess the data:

The data must be pre-processed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255:

```
## plotting single image
plt.figure()
plt.imshow(train_images[89])
plt.colorbar()
plt.grid(False)
plt.show()
```



To verify that the data is in the correct format and that you're ready to build and train the network, let's display the first 25 images from the *training set* and display the class name below each image.

```
## plotting images along with respective classes
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
    plt.show()
```


```
train_images.shape
```

```
    (60000, 28, 28)
```

```
# type(train_images)
```

numpy.ndarray

```
## reshaping training and testing images
train_images =train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
```

```
## Explore the dataset
print(train_images.shape)
print(test_images.shape)
```

```
    (60000, 28, 28, 1)
    (10000, 28, 28, 1)
```

```
## reshaping training and testing lables
```

```
n_classes=10 train_labels = utils.to_categorical(train_labels, n_classes)
test_labels = utils.to_categorical(test_labels, n_classes)
print(len(train_labels)) print(len(test_labels))
   60000
   10000
```

Scale these values to a range of 0 to 1 before feeding them to the Convolutional neural network model. To do so, divide the values by 255. It's important that the *training set* and the *testing set* be preprocessed in the same way:

```
## rescaling
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Build the model:

Building the neural network requires configuring the layers of the model, then compiling the model.

Set up the layers:

The basic building block of a neural network is the layer. Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers, such as tf.keras.layers.Dense , have parameters that are learned during training.

```
## define Model
model = tf.keras.Sequential()
## add layers
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3,3),strides=(1, 1),padding='valid',
activation='relu', input_shape=(28,28, 1)))
model.add(tf.keras.layers.MaxPool2D(pool_size=(1,1)))
model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3),strides=(1, 1),padding='valid',
activation='relu')) model.add(tf.keras.layers.MaxPool2D(pool_size=(1,1)))
# model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3),strides=(1,
1),padding='valid', activation='relu'))
# model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
# model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3),strides=(1,
1),padding='valid', activation='relu')) #
model.add(tf.keras.layers.MaxPool2D(pool_size=(1,1)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(100, activation='relu'))
model.add(tf.keras.layers.Dropout(0.4))
# model.add(tf.keras.layers.Dense(128, activation='relu'))
# model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Dense(10,
activation='softmax'))
## Compile the model model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
## check model Summary model.summary()
```

```
_________________________________________________________________
__
 Layer (type)                 Output Shape              Param #
=================================================================
===  conv2d_3 (Conv2D)              (None, 26, 26, 32)        320
max_pooling2d_3(MaxPooling2D)  (None, 26, 26, 32)        0
conv2d_4 (Conv2D)              (None, 24, 24, 64)        18496
max_pooling2d_4 (MaxPoolin     (None, 24, 24, 64)        0
flatten_2 (Flatten)            (None, 36864)             0
dense_4 (Dense)                (None, 100)               3686500
dropout_1 (Dropout)            (None, 100)               0
dense_5 (Dense)                (None, 10)                1010
=================================================================
==
Total params: 37,06,326 (14.14 MB)
Trainable params: 37,06,326 (14.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
## fit the model and start training
history = model.fit(train_images,
                    train_labels,
                    epochs=5,
                    validation_data = (test_images, test_labels))
```

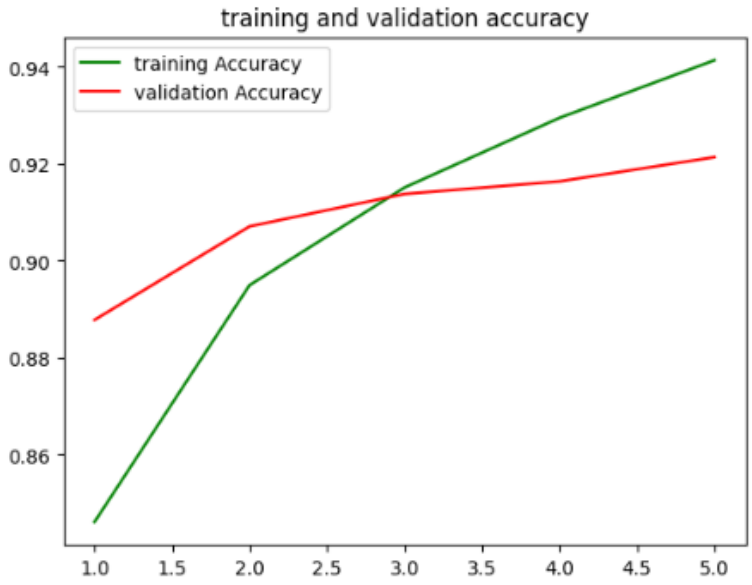
```
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.4366 -
accuracy: 0.8461 - val_loss: 0.3080 - val_accuracy: 0.88
Epoch 2/5
1875/1875 [==============================] - 10s 6ms/step - loss: 0.2876 -
accuracy: 0.8950 - val_loss: 0.2633 - val_accuracy: 0.90
Epoch 3/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2308 -
accuracy: 0.9151 - val_loss: 0.2392 - val_accuracy: 0.91
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1881 -
accuracy: 0.9295 - val_loss: 0.2344 - val_accuracy: 0.91
Epoch 5/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1572 -
accuracy: 0.9414 - val_loss: 0.2396 - val_accuracy: 0.92
```

```
### history variables
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
EPOCHS = range(1,len(train_acc)+1
```

```
## ploting graphs for viewing the training and testing performance
## accuracies
plt.figure()
```

```
plt.plot(EPOCHS, train_acc, 'b', color= 'green', label ='training
Accuracy')
plt.plot(EPOCHS, val_acc, 'b', color ='red', label ='validation Accuracy')
plt.title('training and validation accuracy')
plt.legend()
plt.show()
```

```
## loss
plt.figure()
plt.plot(EPOCHS, train_loss, 'b', color= 'green', label ='training Loss')
plt.plot(EPOCHS, val_loss, 'b', color ='red', label ='validation Loss')
plt.title('training and validation loss')
plt.legend()
plt.show()
```



```
### model evaluation
test_loss, test_acc = model.evaluate(test_images,  test_labels)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.4466 -
accuracy: 0.8663
Test accuracy: 0.8662999868392944
Test loss: 0.44664257764816284
```

```
## measuring the model performace
test_pred = model.predict(test_images)
test_labels = np.argmax(test_labels, axis=1)
test_pred = np.argmax(test_pred, axis=1)
# confusion matrix
print(confusion_matrix(test_labels, test_pred))
```

```
313/313 [==============================] - 1s 3ms/step
[[886   3  12  18   2   2  67   0  10   0]
 [  7 963   1  20   2   0   5   0   2   0]
 [ 25   1 857   7  48   0  62   0   0   0]
 [ 58   5  10 852  27   0  41   2   3   2]
 [  6   3 213  36 660   0  81   0   1   0]
 [  0   0   0   0   0 944   0  24   0  32]
 [195   2  99  23  52   1 608   0  20   0]
 [  0   0   0   0   0  13   0 949   0  38]
 [  7   1   1   4   2   1  10   6 966   2]
 [  0   0   0   0   0   0   0  22   0 978]]
```

```
# classification report
print('Classification Report')
print(classification_report(test_labels, test_pred,
target_names=class_names))
```

```
Classification Report
              precision    recall  f1-score   support

 T-shirt/top       0.75      0.89      0.81      1000
     Trouser       0.98      0.96      0.97      1000
    Pullover       0.72      0.86      0.78      1000
       Dress       0.89      0.85      0.87      1000
        Coat       0.83      0.66      0.74      1000
      Sandal       0.98      0.94      0.96      1000
       Shirt       0.70      0.61      0.65      1000
     Sneaker       0.95      0.95      0.95      1000
         Bag       0.96      0.97      0.97      1000
  Ankle boot       0.93      0.98      0.95      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.86     10000
weighted avg       0.87      0.87      0.86     10000
```