SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CAMPUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2021-22(EVEN SEMESTER)

18CSE479T - Statistical Machine Learning

Unit - I

Statistical terminology for model building and validation-Machine Learning, Major differences between statistical modeling and machine learning- Steps in machine learning model development and deployment- Statistical fundamentals and terminology for model building and validation- Bias versus variance trade-off, Train and test data- Linear regression versus gradient descent Machine learning losses- When to stop tuning machine learning models- Train, validation, and test data Cross-validation- Grid Search- Machine learning model overview.

I INTRODUCTION

Definition

- Statistical modeling is the use of mathematical models and statistical assumptions to generate sample data and make predictions about the real world. A statistical model is a collection of probability distributions on a set of all possible outcomes of an experiment.
- Machine learning is a branch of study in which a model can learn automatically from the experiences based on data without exclusively being modeled like in statistical models.

II STATISTICAL TERMINOLOGY FOR MODEL BUILDING AND VALIDATION

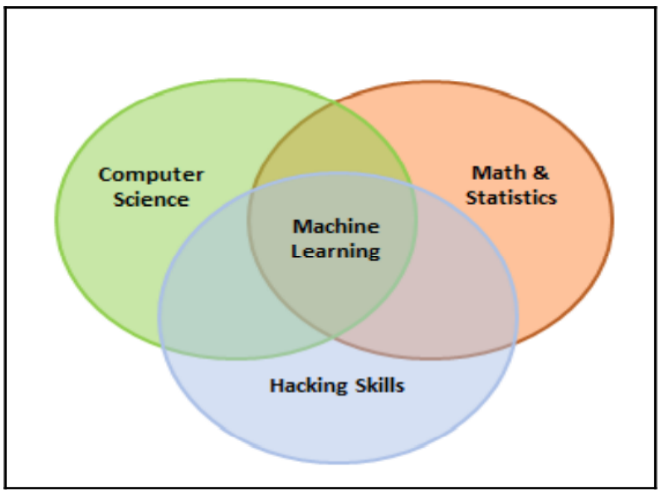
- Statistics is the branch of mathematics dealing with the collection, analysis, interpretation, presentation, and organization of numerical data.
- Statistics are mainly classified into two subbranches
    - **Descriptive statistics**: These are used to summarize data, such as the mean, standard deviation for continuous data types (such as age), whereas frequency and percentage are useful for categorical data (such as gender).

- **Inferential statistics**: A subset of the data points is collected, also called a sample, and conclusions about the entire population will be drawn, which is known as inferential statistics. Inferences are drawn using hypothesis testing, the estimation of numerical characteristics, the correlation of relationships within data, and so on

- Statistical modeling is applying statistics on data to find underlying hidden relationships by analyzing the significance of the variables

III Machine learning

- Machine learning is the branch of computer science that utilizes past experience to learn from and use its knowledge to make future decisions.
- Machine learning is at the intersection of computer science, engineering, and statistics.
- The goal of machine learning is to generalize a detectable pattern or to create an unknown rule from given examples.
- An overview of machine learning landscape is as follows:



- Machine learning is broadly classified into three categories but nonetheless, based on the situation, these categories can be combined to achieve the desired results for particular applications:
  - Supervised Learning
  - Un Supervised Learning
  - Reinforcement Learning
- **Supervised learning**

- This is teaching machines to learn the relationship between other variables and a target variable, similar to the way in which a teacher provides feedback to students on their performance.
- The major segments within supervised learning are as follows
  - ✔ Classification problem
  - ✔ Regression problem

- **Unsupervised learning**
  - In unsupervised learning, algorithms learn by themselves without any supervision or without any target variable provided.
  - It is a question of finding hidden patterns and relations in the given data.
  - The categories in unsupervised learning are as follows
    - ✔ Dimensionality reduction
    - ✔ Clustering
- **Reinforcement Learning**

- This allows the machine or agent to learn its behavior based on feedback from the environment.
- In reinforcement learning, the agent takes a series of decisive actions without supervision and, in the end, a reward will be given, either +1 or -1.
- Based on the final payoff/reward, the agent reevaluates its paths.
- Reinforcement learning problems are closer to the artificial intelligence methodology rather than frequently used machine learning algorithms.

IV MAJOR DIFFERENCES BETWEEN STATISTICAL MODELING AND MACHINE LEARNING

Statistical modeling	**Machine learning**
Formalization of relationships between variables in the form of mathematical equations.	Algorithm that can learn from the data without relying on rule-based programming.

Required to assume shape of the model curve prior to perform model fitting on the data (for example, linear, polynomial, and so on).	Does not need to assume underlying shape, as machine learning algorithms can learn complex patterns automatically based on the provided data.
Statistical model predicts the output with accuracy of 85 percent and having 90 percent confidence about it.	Machine learning just predicts the output with accuracy of 85 percent.
In statistical modeling, various diagnostics of parameters are performed, like p-value, and so on.	Machine learning models do not perform any statistical diagnostic significance tests.
Data will be split into 70 percent - 30 percent to create training and testing data. Model developed on training data and tested on testing data.	Data will be split into 50 percent - 25 percent - 25 percent to create training, validation, and testing data. Models developed on training and hyperparameters are tuned on validation data and finally get evaluated against test data.
Statistical models can be developed on a single dataset called training data, as diagnostics are performed at both overall accuracy and individual variable level.	Due to lack of diagnostics on variables, machine learning algorithms need to be trained on two datasets, called training and validation data, to ensure two-point validation.
Statistical modeling is mostly used for research purposes.	Machine learning is very apt for implementation in a production environment.
From the school of statistics and mathematics.	From the school of computer science.

V STEPS IN MACHINE LEARNING MODEL DEVELOPMENT AND DEPLOYMENT

The development and deployment of machine learning models involves a series of steps that are almost similar to the statistical modeling process, in order to develop, validate, and implement machine learning models. The steps are as follows

1. **Collection of data**: Data for machine learning is collected directly from structured source data, web scrapping, API, chat interaction, and so on, as machine learning can work on both structured and unstructured data (voice, image, and text).
2. **Data preparation and missing/outlier treatment**: Data is to be formatted as per the chosen machine learning algorithm; also, missing value treatment needs to be performed by

replacing missing and outlier values with the mean/median, and so on.

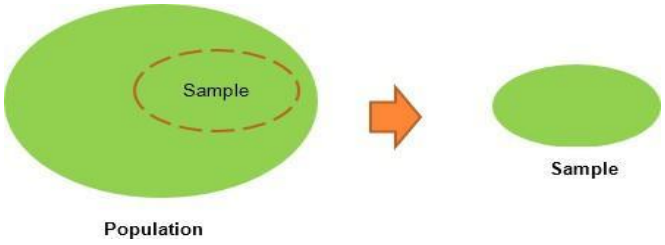
3. **Data analysis and feature engineering**: Data needs to be analyzed in order to find any hidden patterns and relations between variables, and so on. Correct feature engineering with appropriate business knowledge will solve 70 percent of the problems. Also, in practice, 70 percent of the data scientist's time is spent on feature engineering tasks.
4. **Train algorithm on training and validation data**: Post feature engineering, data will be divided into three chunks (train, validation, and test data) rather than two (train and test) in statistical modeling. Machine learning are applied on training data and the hyperparameters of the model are tuned based on validation data to avoid overfitting
5. **Test the algorithm on test data**: Once the model has shown a good enough performance on train and validation data, its performance will be checked against unseen test data. If the performance is still good enough, we can proceed to the next and final step.
6. **Deploy the algorithm**: Trained machine learning algorithms will be deployed on live streaming data to classify the outcomes. One example could be recommender systems implemented by e-commerce websites.

VI STATISTICAL FUNDAMENTALS AND TERMINOLOGY FOR MODEL BUILDING AND VALIDATION

- Statistical models are a class of mathematical models that are usually specified by mathematical equations that relate one or more variables to approximate reality.
- Assumptions embodied by statistical models describe a set of probability distributions, which distinguishes it from non-statistical, mathematical, or machine learning models.
- **The various fundamentals of Statistical models are**

i. **Population**: This is the totality, the complete list of observations, or all the data points about the subject under study.

ii. **Sample**: A sample is a subset of a population, usually a small portion of the population that is being analyzed

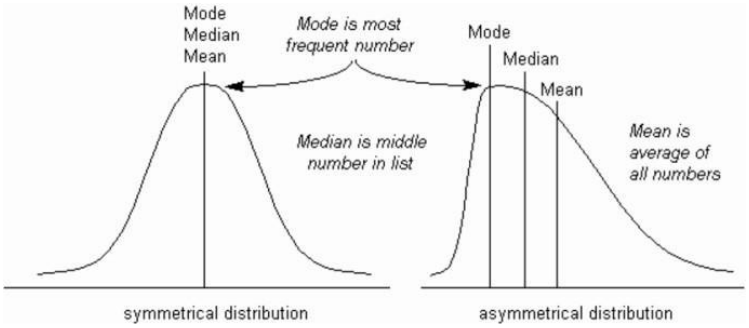


iii. Parameter versus statistic: Any measure that is calculated on the population is a parameter, whereas on a sample it is called a **statistic**

iv. Mean: This is a simple arithmetic average, which is computed by taking the aggregated sum of values divided by a count of those values. The mean is sensitive to outliers in the data. An outlier is the value of a set or column that is highly deviant from the many other values in the same data; it usually has very high or low values

v. **Median**: This is the midpoint of the data, and is calculated by either arranging it in ascending or descending order. If there are *N* observations

vi. **Mode**: This is the most repetitive data point in the data



The Python code for the calculation of mean, median, and mode using a numpy array and the stats package is as follows:

```
>>> import numpy as np
>>> from scipy import stats

>>> data = np.array([4,5,1,2,7,2,6,9,3])

# Calculate Mean
>>> dt_mean = np.mean(data) ; print ("Mean :",round(dt_mean,2))

# Calculate Median
>>> dt_median = np.median(data) ; print ("Median :",dt_median)

# Calculate Mode
>>> dt_mode = stats.mode(data); print ("Mode :",dt_mode[0][0])
```

The output of the preceding code is as follows:

```
Mean : 4.33333333333
Median : 4.0
Mode : 2
[Finished in 0.3s]
```

scikit-learn package built on top of NumPy array in which all statistical models and machine learning algorithms have been built on NumPy array itself. The mode function is not implemented in the numpy package, hence we have used SciPy's stats package. SciPy is also built on top of NumPy arrays.

vi. Measure of variation: Dispersion is the variation in the data, and measures the inconsistencies in the value of variables in the data. Dispersion actually provides an idea about the spread rather than central values

vii. Range: This is the difference between the maximum and minimum of the value.

viii. **Variance**: This is the mean of squared deviations from the mean (*xi* = data points, μ = mean of the data, *N* = number of data points). The dimension of variance is the square of the actual values. The reason to use denominator *N-1* for a sample instead of *N* in the population is due the

$$population\ variance = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \quad sample\ variance = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$$

degree of freedom.

ix. Standard deviation: This is the square root of variance. By applying the square root on variance, we measure the dispersion with respect to the original variable

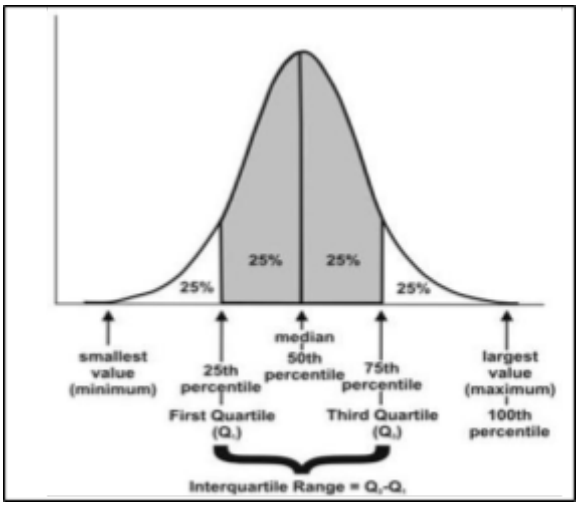
$$population\ standard\ deviation\ (\sigma) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad sample\ standard\ deviation\ (s) = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$$

rather than square of the dimension

x. Quantiles: These are simply identical fragments of the data. Quantiles cover percentiles, deciles, quartiles, and so on. These measures are calculated after

arranging the data in ascending order

- **Percentile**: This is nothing but the percentage of data points below the value of the original whole data. The median is the 50th percentile, as the number of data points below the median is about 50 percent of the data.
- **Decile**: This is 10th percentile, which means the number of data points below the decile is 10 percent of the whole data.
- **Quartile**: This is one-fourth of the data, and also is the 25th percentile. The first quartile is 25 percent of the data, the second quartile is 50 percent of the data, the third quartile is 75 percent of the data. The second quartile is also known as the median or 50th percentile or 5th decile.
- **Interquartile range**: This is the difference between the third quartile and first quartile. It is effective in identifying outliers in data. The interquartile range describes the middle 50 percent of the data points



The Python code is as follows:

```
>>> from statistics import variance, stdev
>>> game_points = np.array([35,56,43,59,63,79,35,41,64,43,93,60,77,24,82])

# Calculate Variance
>>> dt_var = variance(game_points) ; print ("Sample variance:", round(dt_var,2))

# Calculate Standard Deviation
>>> dt_std = stdev(game_points) ; print ("Sample std.dev:", round(dt_std,2))
```

```
# Calculate Range
>>> dt_rng = np.max(game_points,axis=0) - np.min(game_points,axis=0) ; print ("Range:",dt_rng)

#Calculate percentiles
>>> print ("Quantiles:")
>>> for val in [20,80,100]:
>>>         dt_qntls = np.percentile(game_points,val)
>>>         print  (str(val)+"%"  ,dt_qntls)  #  Calculate IQR
>>> q75, q25 = np.percentile(game_points, [75 ,25]); print ("Inter quartile range:",q75-q25)
```

The output of the preceding code is as follows:

```
Sample variance: 400.64
Sample std.dev: 20.02
Range: 69
Quantiles:
20% 39.8
80% 77.4
100% 93.0
Inter quartile range: 28.5
[Finished in 0.2s]
```

xi. **Hypothesis testing**: This is the process of making inferences about the overall population by conducting some statistical tests on a sample. Null and alternate hypotheses are ways to validate whether an assumption is statistically significant or not.

xii. **P-value**: The probability of obtaining a test statistic result is at least as extreme as the one that was actually observed, assuming that the null hypothesis is true (usually in modeling, against each independent variable, a p-value less than 0.05 is considered significant and greater than 0.05 is considered insignificant;

The steps involved in hypothesis testing are as follows:

1. Assume a null hypothesis (usually no difference, no significance, and so on; a null hypothesis always tries to assume that there is no anomaly pattern and is always homogeneous, and so on).
2. Collect the sample.
3. Calculate test statistics from the sample in order to verify whether the hypothesis is statistically significant or not.

4. Decide either to accept or reject the null hypothesis based on the test statistic.

Example of hypothesis testing: A chocolate manufacturer who is also your friend claims that all chocolates produced from his factory weigh at least 1,000 g and you have got a funny feeling that it might not be true; you both collected a sample of 30 chocolates and found that the average chocolate weight as 990 g with sample standard deviation as 12.5 g. Given the 0.05 significance level, can we reject the claim made by your friend?

The null hypothesis is that $\mu 0 \geq 1000$ (all chocolates weigh more than 1,000 g). Collected sample:

$$\bar{x} = 990, s = 12.5, n = 30$$

Calculate test statistic:

$$t = \frac{(\bar{x} - \mu_0)}{(\frac{s}{\sqrt{n}})}$$

t = (990 - 1000) / (12.5/sqrt(30)) = - 4.3818

Critical t value from t tables = t0.05, 30 = 1.699 => - t0.05, 30 = -1.699

P-value = 7.03 e-05

Test statistic is *-4.3818*, which is less than the critical value of *-1.699*. Hence, we can reject the null hypothesis (your friend's claim) that the mean weight of a chocolate is above 1,000 g.

A p-value less than *0.05* means both claimed values and distribution mean values are significantly different, hence we can reject the null hypothesis:

The Python code is as follows:

```
>>> from scipy import stats
>>> xbar = 990; mu0 = 1000; s = 12.5; n = 30
```

```
# Test Statistic

>>> t_smple = (xbar-mu0)/(s/np.sqrt(float(n))); print ("Test Statistic:",round(t_smple,2))


# Critical value from t-table
>>> alpha = 0.05
>>> t_alpha = stats.t.ppf(alpha,n-1); print ("Critical value from t-table:",round(t_alpha,3))


#Lower tail p-value from t-table

>>> p_val = stats.t.sf(np.abs(t_smple), n-1); print ("Lower tail p-value from t-table", p_val)
```

xiii. Type I and II error:

Hypothesis testing is usually done on the samples rather than the entire population, due to the practical constraints of available resources to collect all the available data. However, performing inferences about the population from samples comes with its own costs, such as rejecting good results or accepting false results, not to mention separately, when increases in sample size lead to minimizing type I and II errors:

Type I error: Rejecting a null hypothesis when it is true

Type II error: Accepting a null hypothesis when it is false

xiv. Normal distribution:

This is very important in statistics because of the central limit theorem, which states that the population of all possible samples of size *n* from a population with mean μ and variance $\sigma 2$ approaches a normal distribution:

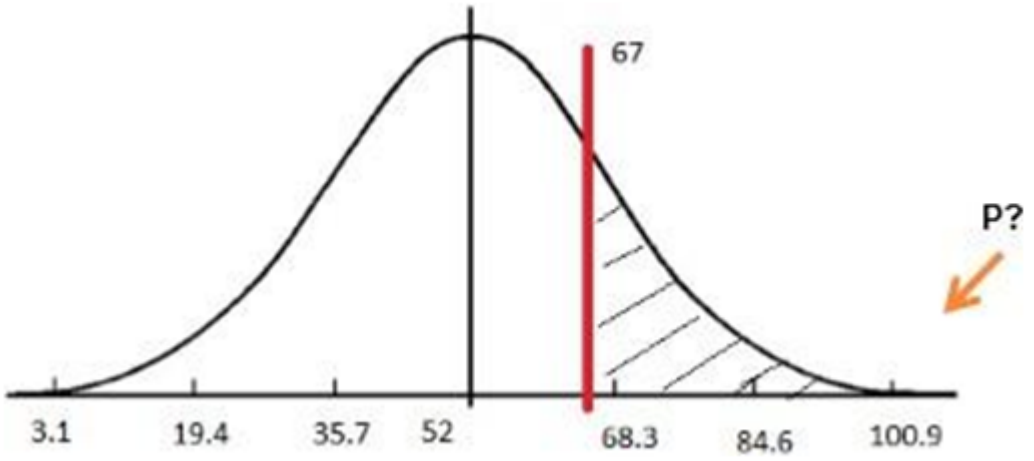
$$f(x) = \frac{1}{\sigma\sqrt{2\Pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad X \sim N(\mu, \sigma^2)$$

Example: Assume that the test scores of an entrance exam fit a normal distribution. Furthermore, the mean test score is *52* and the standard deviation is *16.3*. What is the percentage of students scoring *67* or more in the exam?

$$pr(X \geq 67) = pr\left(Z \geq \frac{(67-\mu)}{\sigma}\right) = 1 - pr\left(Z \leq \frac{(67-\mu)}{\sigma}\right)$$

$$= 1 - pr\left(Z \leq \frac{(67-52)}{16.3}\right) = 1 - 0.8212 = 0.1788$$

$$probability\ of\ students\ scoring\ more\ than\ 67\ marks\ is = 17.88\ \%$$



The Python code is as follows:

```
>>> from scipy import stats
>>> xbar = 67; mu0 = 52; s = 16.3


# Calculating z-score

>>> z = (67-52)/16.3


# Calculating probability under the curve
>>> p_val = 1- stats.norm.cdf(z)
>>> print ("Prob. to score more than 67 is
",round(p_val*100,2),"%")
```

xv. Chi-square:

This test of independence is one of the most basic and common hypothesis tests in the statistical analysis of categorical data. Given two categorical random variables *X* and *Y*, the chi-square test of independence determines whether or not there exists a statistical dependence between them

The test is usually performed by calculating $\chi 2$ from the data and $\chi 2$ with (*m-1*, *n-1*) degrees from the table. A decision is made as to whether both variables are independent based on the actual value and table value, whichever is higher

$$\chi^2 = \sum_i \frac{(o_i - e_i)^2}{e_i} \qquad o_i = observed\ , e_i = expected$$

Example: In the following table, calculate whether the smoking habit has an impact on exercise behavior

	Exercise: Frequent	Exercise: None	Exercise: Sometimes
Smoke: Heavy	7	1	3
Smoke: Never	87	18	84
Smoke: Occasional	12	3	4
Smoke: Regularly	9	1	7

The Python code is as follows:

```
>>> import pandas as pd
>>> from scipy import stats
```

```
>>> survey = pd.read_csv("survey.csv")
```

Tabulating 2 variables with row & column variables respectively

```
>>> survey_tab = pd.crosstab(survey.Smoke, survey.Exer, margins
= True)
```

While creating a table using the crosstab function, we will obtain both row and column totals fields extra. However, in order to create the observed table, we need to extract the variables part and ignore the totals:

Creating observed table for analysis

```
>>> observed = survey_tab.ix[0:4,0:3]
```

The chi2_contingency function in the stats package uses the observed table and subsequently calculates its expected table, followed by calculating the p-value in order to check whether two variables are dependent or not. If *p-value < 0.05*, there is a strong dependency between two variables, whereas if *p-value > 0.05*, there is no dependency between the variables:

```
>>> contg = stats.chi2_contingency(observed= observed)
>>> p_value = round(contg[1],3)
>>> print ("P-value is: ",p_value)
```

The p-value is 0.483, which means there is no dependency between the smoking habit and exercise behavior.

xvi. ANOVA:

Analyzing variance tests the hypothesis that the means of two or more populations are

equal. ANOVAs assess the importance of one or more factors by comparing the response variable means at the different factor levels. The null hypothesis states that all population means are equal while the alternative hypothesis states that at least one is different.

Example: A fertilizer company developed three new types of universal fertilizers after research that can be utilized to grow any type of crop. In order to find out whether all three have a similar crop yield, they randomly chose six crop types in the study. In accordance with the randomized block design, each crop type will be tested with all three types of fertilizer separately. The following table represents the yield in g/m^2. At the 0.05 level of significance, test whether the mean yields for the three new types of fertilizers are all equal.

Fertilizer 1	Fertilizer 2	Fertilizer 3
62	54	48
62	56	62
90	58	92
42	36	96
84	72	92
64	34	80

The Python code is as follows:

```
>>> import pandas as pd
>>> from scipy import stats
>>> fetilizers = pd.read_csv("fetilizers.csv")
```

Calculating one-way ANOVA using the stats package:

```
>>> one_way_anova = stats.f_oneway(fetilizers["fertilizer1"], fetilizers["fertilizer2"],
fetilizers["fertilizer3"])
```

```
>>> print ("Statistic :", round(one_way_anova[0],2),", p-value
:",round(one_way_anova[1],3))
```

Result: The p-value did come as less than 0.05, hence we can reject the null hypothesis that the mean crop yields of the fertilizers are equal. Fertilizers make a significant difference to crops

xvi. Confusion matrix:

This is the matrix of the actual versus the predicted. This concept is better explained with the example of cancer prediction using the model.

	Predicted: Yes	Predicted: No
Actual: Yes	TP	FN
Actual: No	FP	TN

Some terms used in a confusion matrix are:

- True positives (TPs): True positives are cases when we predict the disease as yes when the patient actually does have the disease.
- True negatives (TNs): Cases when we predict the disease as no when the patient actually does not have the disease.
- False positives (FPs): When we predict the disease as yes when the patient actually does not have the disease. FPs are also considered to be type I errors.
- False negatives (FNs): When we predict the disease as no when the patient actually does have the disease. FNs are also considered to be type II errors.

xvii. Performance Measures

Precision (P): When yes is predicted, how often is it correct?

(TP/TP+FP)

Recall (R)/sensitivity/true positive rate: Among the actual yeses, what fraction was predicted as yes?

(TP/TP+FN)

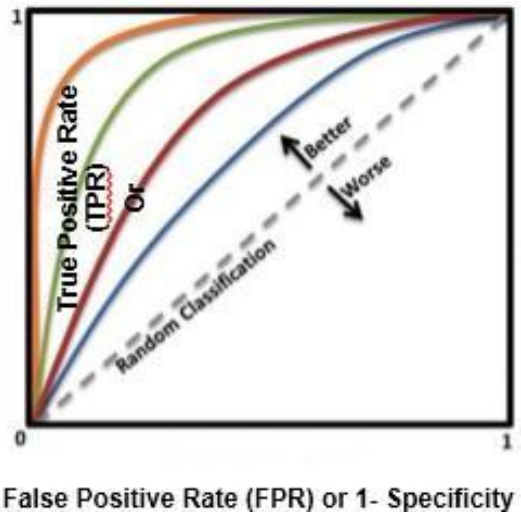
F1 score (F1): This is the harmonic mean of the precision and recall. Multiplying the constant of *2* scales the score to *1* when both precision and recall are *1*

$$F_1 = \frac{2}{\frac{1}{P}+\frac{1}{R}} \quad \gg \quad F_1 = \frac{2*P*R}{P+R}$$

Specificity: Among the actual nos, what fraction was predicted as no? Also equivalent to *1- false positive rate*

(TN/TN+FP)

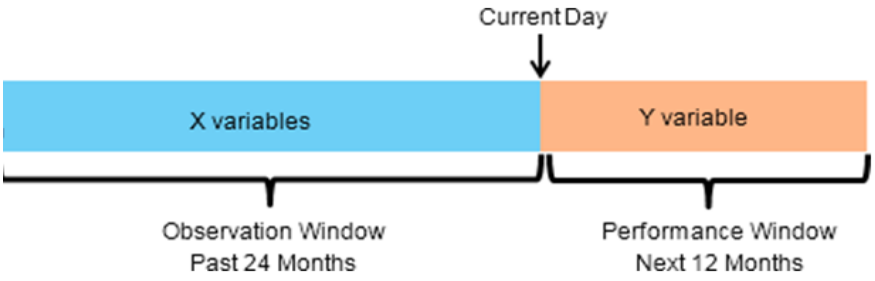
Area under curve (ROC): Receiver operating characteristic curve is used to plot between true positive rate (TPR) and false positive rate (FPR), also known as a sensitivity and *1- specificity* graph



xviii. Observation and performance window:

In statistical modeling, the model tries to predict the event in advance rather than at the moment, so that some buffer time will exist to work on corrective actions. For example, a question from a credit card company would be, for example, what is the probability that a particular customer will default in the coming 12-month period.

A probability of default model (or behavioral scorecard in technical terms) needs to be developed by using independent variables from the past 24 months and a dependent variable from the next 12 months. After preparing data with *X* and *Y* variables, it will be split into 70 percent - 30 percent as train and test data randomly; this method is called in- time validation as both train and test samples are from the same time period.



xix. In-time and out-of-time validation:

In-time validation implies obtaining both a training and testing dataset from the same period of time, whereas out-of-time validation implies training and testing datasets drawn from different time periods. Usually, the model performs worse in out-of-time validation rather than in-time due to the obvious reason that the characteristics of the train and test datasets might differ.

xx. R-squared (coefficient of determination):

This is the measure of the percentage of the response variable variation that is explained by a model. It also a measure of how well the model minimizes error compared with just utilizing the mean as an estimate. In some extreme cases, R-squared can have a value less than zero also, which means the predicted values from the model perform worse than just taking the simple mean as a prediction for all the observations.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \;;\; SS_{tot} = \sum_i (y_i - \bar{y})^2 \;;\; SS_{reg} = \sum_i (f_i - \bar{y})^2 \;;\; SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}}$$

xxi. Adjusted R-squared:

The explanation of the adjusted R-squared statistic is almost the same as R-squared but it penalizes the R-squared value if extra variables without a strong correlation are included in the model

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}$$

Here, *R2* = sample R-squared value, *n* = sample size, *k* = number of predictors (or) variables.

Adjusted R-squared value is the key metric in evaluating the quality of linear regressions. Any linear regression model having the value of *R2 adjusted* >= *0.7* is considered as a good enough model to implement.

Example: The R-squared value of a sample is *0.5,* with a sample size of *50* and the independent variables are *10* in number. Calculated adjusted R-squared

$$R^2_{adjusted} = 1 - \frac{(1 - 0.5)(50 - 1)}{50 - 10 - 1} = 0.402$$

xxii. Maximum likelihood estimate (MLE):

This is estimating the parameter values of a statistical model (logistic regression, to be precise) by finding the parameter values that maximize the likelihood of making the observations.

xxiii. Akaike information criteria (AIC): This is used in logistic regression, which is similar to the principle of adjusted R-square for linear regression. It measures the relative quality of a model for a given set of data

$$AIC = -2 * \ln(L) + 2 * k$$

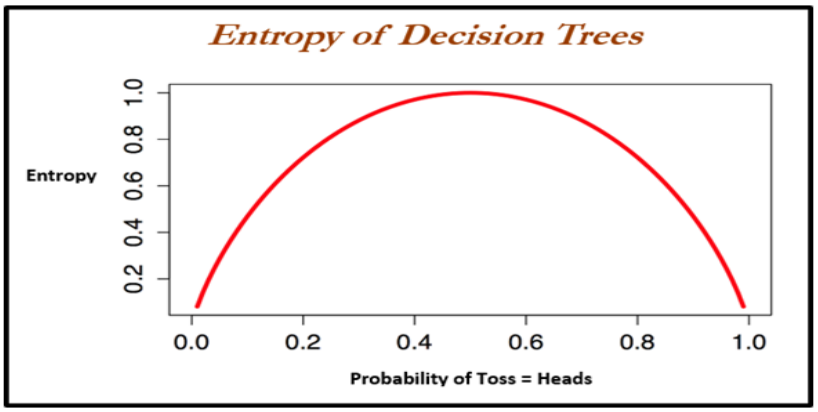
Here, k = number of predictors or variables

xxiv. Entropy:

This comes from information theory and is the measure of impurity in the data. If the sample is completely homogeneous, the entropy is zero and if the sample is equally divided, it has an entropy of *1*.

$$Entropy = -p_1 * log_2 p_1 - \ldots - -p_n * log_2 p_n$$

Here, n = number of classes. Entropy is maximal at the middle, with the value of *1* and minimal at the extremes as *0*. A low value of entropy is desirable as it will segregate classes better



Example: Given two types of coin in which the first one is a fair one (*1/2* head and *1/2* tail probabilities) and the other is a biased one (*1/3* head and *2/3* tail probabilities), calculate the entropy for both and justify which one is better with respect to modeling:

$$Entropy\ of\ a\ Fair\ Coin = -\frac{1}{2} * log_2 \frac{1}{2} - \frac{1}{2} * log_2 \frac{1}{2} = 1\ bits$$

$$Entropy\ of\ a\ Biased\ Coin = -\frac{1}{3} * log_2 \frac{1}{3} - \frac{2}{3} * log_2 \frac{2}{3} = 0.9183\ bits$$

From both values, the decision tree algorithm chooses the biased coin rather than the fair coin as an observation splitter due to the fact the value of entropy is less.

xxv. Information gain:

This is the expected reduction in entropy caused by partitioning the examples according to a given attribute. The idea is to start with mixed classes and to keep partitioning until each node reaches its observations of the purest class

*Information gain = Entropy of parent - sum (weighted % * Entropy of child)*

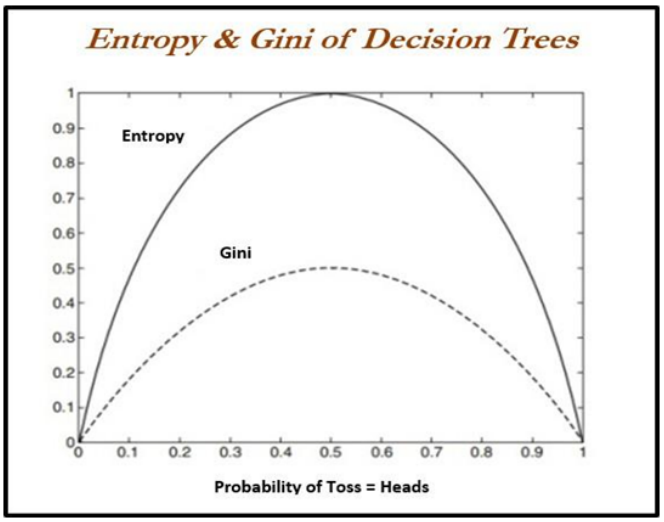
Weighted % = Number of observations in particular child / sum (observations in all child nodes)

xxvi. Gini:

Gini impurity is a measure of misclassification, which applies in a multiclass classifier context. Gini works almost the same as entropy, except Gini is faster to calculate

$$Gini = 1 - \sum_i p_i^2$$

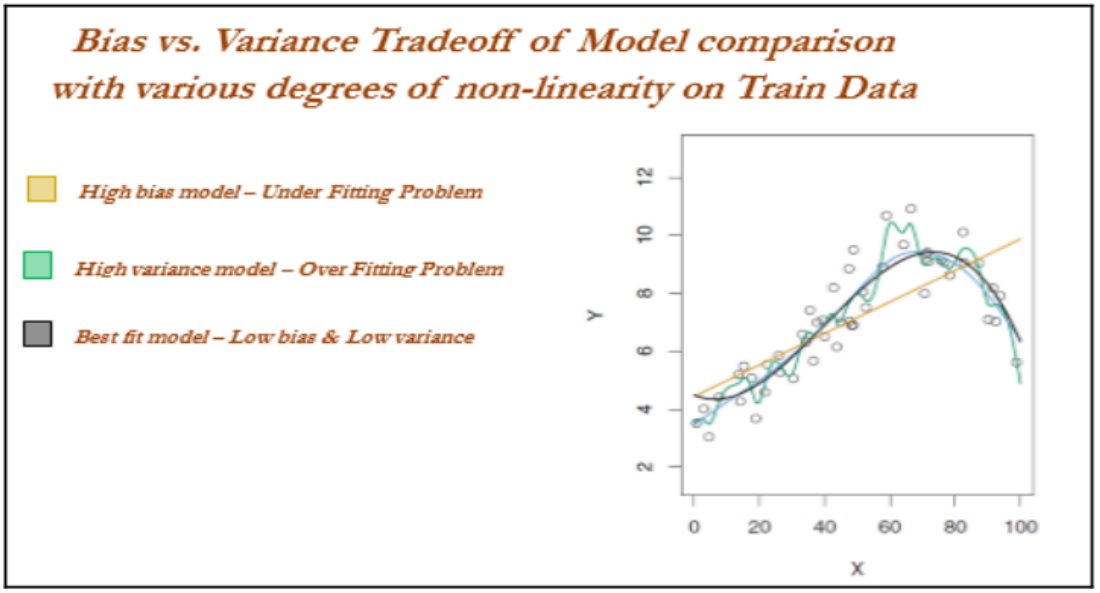
Here, *i* = number of classes. The similarity between Gini and entropy is shown as follows



VII BIAS VERSUS VARIANCE TRADE-OFF

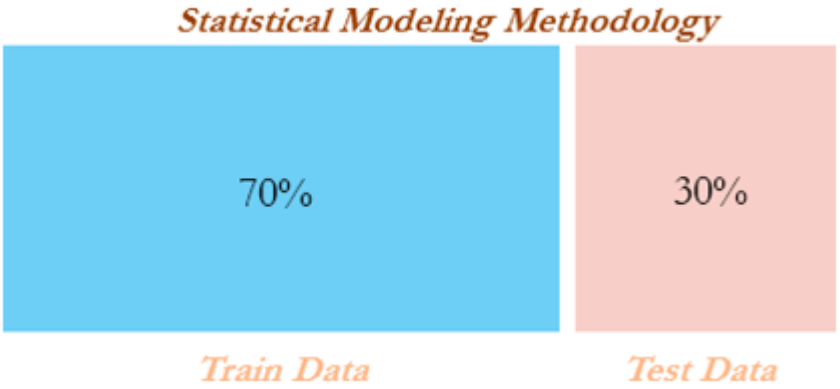
- Every model has both bias and variance error components in addition to white noise.
- Bias and variance are inversely related to each other; while trying to reduce one component, the other component of the model will increase.
- The true art lies in creating a good fit by balancing both. The ideal model will have both low bias and low variance.
- Errors from the bias component come from erroneous assumptions in the underlying learning algorithm.High bias can cause an algorithm to miss the relevant relations between features and target outputs; this phenomenon causes an underfitting problem.
- On the other hand, errors from the variance component come from sensitivity to change in the fit of the model, even a small change in training data; high variance can cause an overfitting problem

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = Var\left(\hat{f}(x_0)\right) + \left[Bias(\hat{f}(x_0))\right]^2 + Var(\varepsilon)$$



VIII TRAIN AND TEST DATA

- In practice, data usually will be split randomly 70-30 or 80-20 into train and test datasets respectively in statistical modeling, in which training data utilized for building the model and its effectiveness will be checked on test data



- In the following code, we split the original data into train and test data by 70 percent - 30 percent.
- Random state is seed in this process of generating pseudo-random numbers, which makes the results reproducible by splitting the exact same observations while running every time

```
# Train & Test split
>>> import pandas as pd
>>> from sklearn.model_selection import train_test_split
>>> original_data = pd.read_csv("mtcars.csv")

>>> train_data,test_data = train_test_split(original_data,train_size = 0.7,random_state=42)
```

IX LINEAR REGRESSION VERSUS GRADIENT DESCENT

Linear Regression

- Linear regression analysis is used to predict the value of a variable based on the value of another variable.
- The variable you want to predict is called the dependent variable.
- The variable you are using to predict the other variable's value is called the independent variable

Gradient Descent

- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
- Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates

In the following code, a comparison has been made between applying linear regression in a statistical way and gradient descent in a machine learning way on the same dataset:

```
>>> import numpy as np
>>> import pandas as pd
```

The following code describes reading data using a pandas DataFrame:

```
>>> train_data = pd.read_csv("mtcars.csv")
```

Converting DataFrame variables into NumPy arrays in order to process them in scikit learn packages, as scikit-learn is built on NumPy arrays itself, is shown next:

```
>>> X = np.array(train_data["hp"]) ; y = np.array(train_data["mpg"])

>>> X = X.reshape(32,1); y = y.reshape(32,1)
```

Importing linear regression from the scikit-learn package; this works on the least squares method:

```
>>> from sklearn.linear_model import LinearRegression

   >>> model = LinearRegression(fit_intercept = True)
```

Fitting a linear regression model on the data and display intercept and coefficient of single variable (hp variable):

```
>>> model.fit(X,y)
>>> print ("Linear Regression Results" )
>>> print ("Intercept",model.intercept_[0] ,"Coefficient", model.coef_[0])
```

Defining the gradient descent function gradient_descent with the following:

x: Independent variable.

y: Dependent variable.

learn_rate: Learning rate with which gradients are updated; too low causes slower convergence and too high causes overshooting of gradients. batch_size: Number of observations considered at each iteration for updating gradients; a high number causes a lower number of iterations and a lower

number causes an erratic decrease in errors. Ideally, the batch size should be a minimum value of 30 due to statistical significance. However, various settings need to be tried to check which one is better.

max_iter: Maximum number of iteration, beyond which the algorithm will get auto-terminated:

```
>>> def gradient_descent(x, y,learn_rate,
conv_threshold,batch_size, max_iter):
...     converged = False
...     iter = 0
...     m = batch_size
...     t0 = np.random.random(x.shape[1])
...     t1 = np.random.random(x.shape[1])
```

The following code states, run the algorithm until it does not meet the convergence criteria:

```
... while not converged:
...     grad0 = 1.0/m * sum([(t0 + t1*x[i] - y[i]) for i in range(m)])
...     grad1 = 1.0/m * sum([(t0 + t1*x[i] - y[i])*x[i] for i in range(m)])
...     temp0 = t0 - learn_rate * grad0
...     temp1 = t1 - learn_rate * grad1
...     t0 = temp0
...     t1 = temp1
```

Calculate a new error with updated parameters, in order to check whether the new error changed more than the predefined convergence threshold value; otherwise, stop the iterations and return parameters:

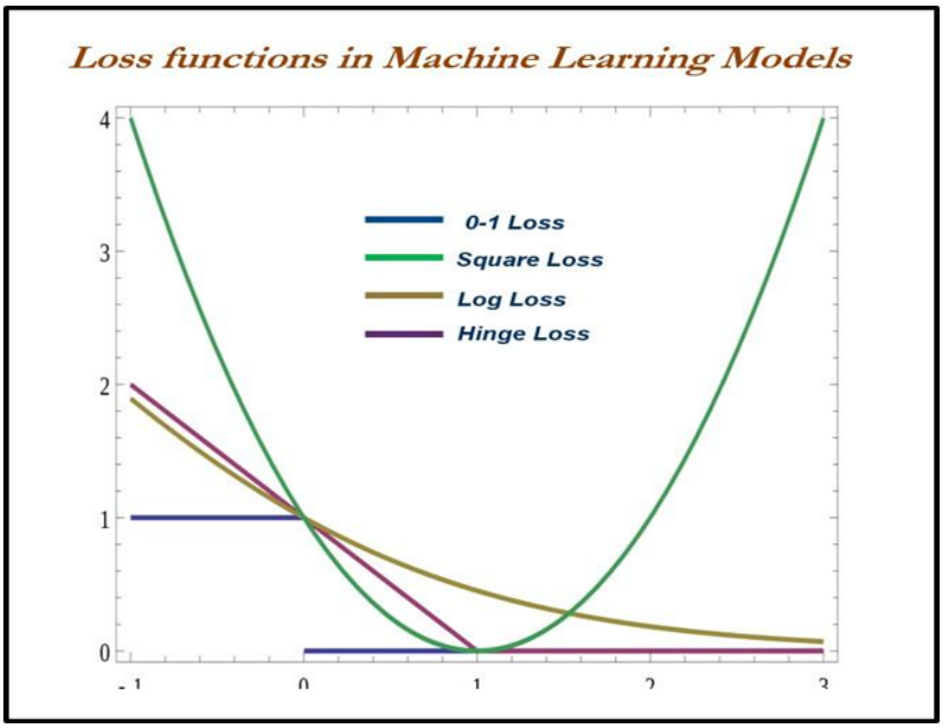
```
...     MSE_New = (sum( [ (t0 + t1*x[i] - y[i])**2 for i in range(m)]
) / m)
...     if abs(MSE - MSE_New ) <= conv_threshold:
...     print 'Converged, iterations: ', iter
...     converged = True
...     MSE = MSE_New
...     iter += 1
...     if iter == max_iter:
...     print 'Max interactions reached'
...     converged = True
...     return t0,t1
```

The following code describes running the gradient descent function with defined values. Learn rate = 0.0003, convergence threshold = 1e-8, batch size = 32, maximum number of iteration = 1500000:

```
>>> if__name_== '_main_':
... Inter, Coeff = gradient_descent(x = X,y = y,learn_rate=0.00003 , conv_threshold = 1e-8, batch_size=32,max_iter=1500000)
... print ('Gradient Descent Results')
... print (('Intercept = %s Coefficient = %s') %(Inter, Coeff))
```

X MACHINE LEARNING LOSSES

- The loss function or cost function in machine learning is a function that maps the values of variables onto a real number intuitively representing some cost associated with the variable values.
- Optimization methods are applied to minimize the loss function by changing the parameter values, which is the central theme of machine learning.
- Zero-one loss is *L0-1 = 1 (m <= 0)*; in zero-one loss, value of loss is *0* for *m >= 0* whereas *1* for *m < 0*.
- Surrogate losses used for machine learning in place of zero-one loss are given as follows. The zero-one loss is not differentiable, hence approximated losses are being used instead:
    - Squared loss (for regression)
    - Hinge loss (SVM)
    - Logistics /Log Loss(Logistic Regression)



$$f_W(x) = sign\left(w.\phi(x)\right)$$

$$Loss_{0-1}(x, y, w) = 1\left[f_W(x) \neq y\right] = 1\left[\underbrace{\left(w.\phi(x)\right)y}_{\text{Margin}} \leq 0\right]$$

$$Loss_{squared}(x, y, w) = \underbrace{\left(f_W(x) - y\right)^2}_{\text{Residual}}$$

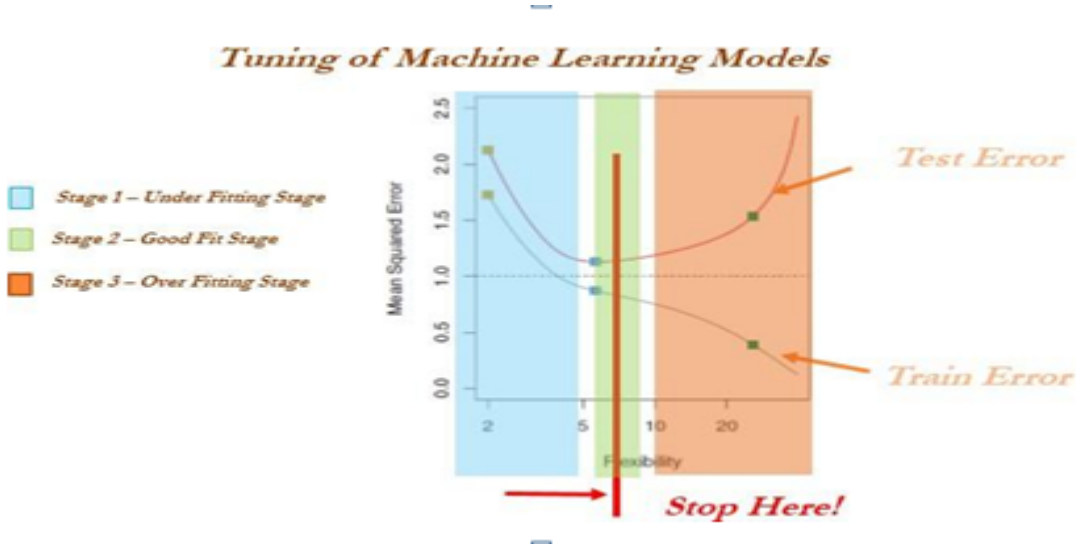
$$Loss_{hinge}(x, y, w) = \max\left\{1 - \left(w.\phi(x)\right)y, 0\right\}$$

$$Loss_{logistic}(x, y, w) = \log\left(1 + e^{-\left(w.\phi(x)\right)y}\right)$$

XI. WHEN TO STOP TUNING MACHINE LEARNING MODELS

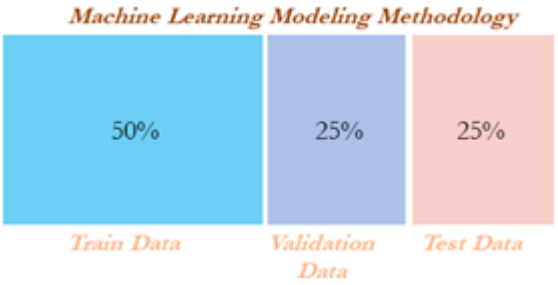
While increasing the complexity of a model, the following stages occur:

- **Stage 1**: Underfitting stage - high train and high test errors (or low train and low test accuracy)
- **Stage 2:** Good fit stage (ideal scenario) - low train and low test errors (or high train and high test accuracy)
- **Stage 3**: Overfitting stage - low train and high test errors (or high train and low test accuracy)



XI TRAIN, VALIDATION, AND TEST DATA

- Cross-validation is not popular in the statistical modeling world for many reasons;
- Statistical models are linear in nature and robust, and do not have a high variance/overfitting problem. Hence, the model fit will remain the same either on train or test data, which does not hold true in the machine learning world.
- Also, in statistical modeling, lots of tests are performed at the individual parameter level apart from aggregated metrics, whereas in machine learning we do not have visibility at the individual parameter level



- The default parameters are 50 percent for train data, 25 percent for validation data, and 25 percent for the remaining test data.
- Python implementation has only one train and test split functionality, hence we have used it twice and also used the number of observations to split rather than the percentage (as shown in the previous train and test split example).
- Hence, a customized function is needed to split into three datasets:

```
>>> import pandas as pd

>>> from sklearn.model_selection import train_test_split
>>> original_data = pd.read_csv("mtcars.csv")
>>> def data_split(dat,trf = 0.5,vlf=0.25,tsf = 0.25):
...  nrows = dat.shape[0]
...  trnr = int(nrows*trf)
...  vlnr = int(nrows*vlf)
```

- The following Python code splits the data into training and the remaining data. The remaining data will be further split into validation and test datasets:

```
...  tr_data,rmng = train_test_split(dat,train_size = trnr,random_state=42)
...  vl_data, ts_data = train_test_split(rmng,train_size = vlnr,random_state=45)

...  return (tr_data,vl_data,ts_data)
```

- Implementation of the split function on the original data to create three datasets (by 50 percent, 25 percent, and 25 percent splits) is as follows:

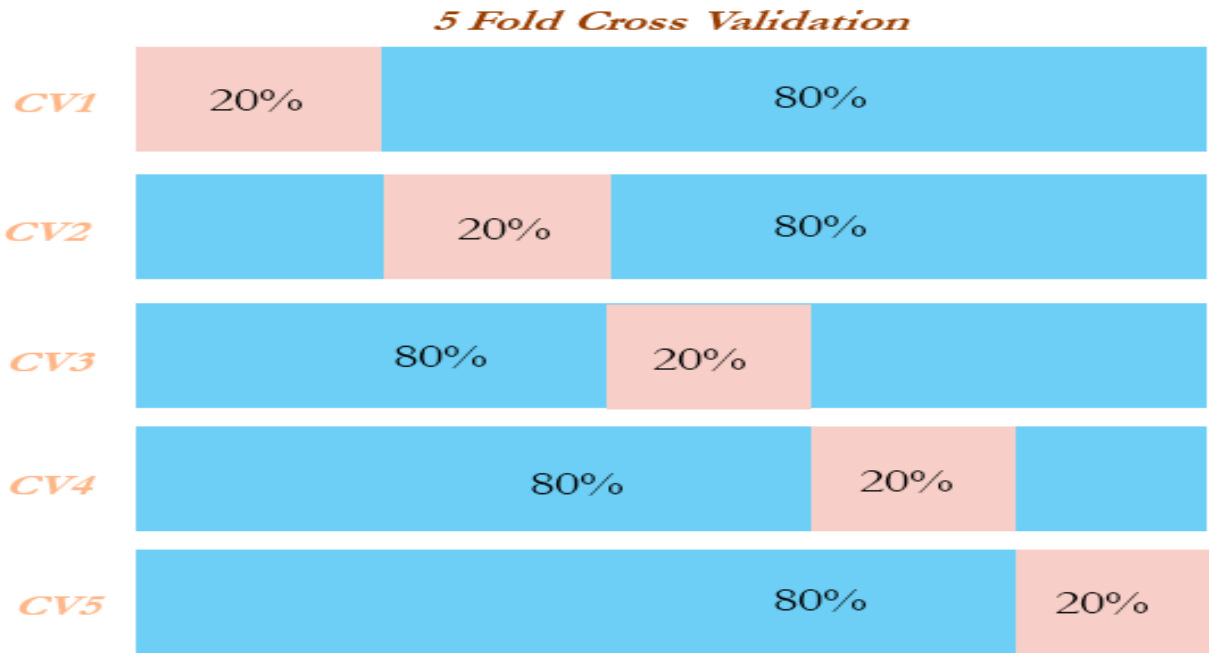
```
>>> train_data, validation_data, test_data = data_split (original_data

,trf=0.5, vlf=0.25,tsf=0.25)
```

- The R code for the train, validation, and test split is as follows:

```
# Train Validation & Test samples
  trvaltest <- function(dat,prop = c(0.5,0.25,0.25)){ nrw = nrow(dat)
  trnr = as.integer(nrw *prop[1]) vlnr =
  as.integer(nrw*prop[2]) set.seed(123)
  trni = sample(1:nrow(dat),trnr) trndata =
  dat[trni,]

  rmng = dat[-trni,]

  vlni = sample(1:nrow(rmng),vlnr) valdata =
  rmng[vlni,]

  tstdata = rmng[-vlni,]

  mylist = list("trn" = trndata,"val"= valdata,"tst" = tstdata)

  return(mylist)
}
outdata = trvaltest(mtcars,prop = c(0.5,0.25,0.25))
train_data = outdata$trn; valid_data = outdata$val; test_data = outdata$tst
```

XII CROSS-VALIDATION

- Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice
- **Example**: In five-fold cross-validation, data will be divided into five parts, subsequently trained on four parts of the data, and tested on the one part of the data. This process will run five times, in order to cover all points in the data. Finally, the error calculated will be the average of all the errors



XIII GRID SEARCH

- Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary.
- Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model.
- Grid search in machine learning is a popular way to tune the hyperparameters of the model in order to find the best combination for determining the best fit.
- Grid search has been implemented using a decision tree classifier for classification purposes.
- Tuning parameters are the depth of the tree, the minimum number of observations in terminal node, and the minimum number of observations required to perform the node split

```
# Grid search
>>> import pandas as pd
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
>>> from sklearn.pipeline import Pipeline
```

```
>>> from sklearn.grid_search import GridSearchCV

>>> input_data = pd.read_csv("ad.csv",header=None)

>>> X_columns = set(input_data.columns.values)
>>> y = input_data[len(input_data.columns.values)-1]
>>> X_columns.remove(len(input_data.columns.values)-1)
>>> X = input_data[list(X_columns)]
```

- Split the data into train and testing:

  ```
  >>> X_train, X_test,y_train,y_test = train_test_split(X,y,train_size = 0.7,random_state=33)
  ```

  - Create a pipeline to create combinations of variables for the grid search:

    ```
    >>> pipeline = Pipeline([
    ...     ('clf', DecisionTreeClassifier(criterion='entropy')) ])
    ```

- Combinations to explore are given as parameters in Python dictionary format:

  ```
  >>> parameters = {
  ...     'clf max_depth': (50,100,150),
  ...     'clf min_samples_split': (2, 3),
  ...     'clf min_samples_leaf': (1, 2, 3)}
  ```

  - The n_jobs field is for selecting the number of cores in a computer; -1 means it uses all the cores in the computer. The scoring methodology is accuracy, in which many other options can be chosen, such as precision, recall, and f1:

    ```
    >>> grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, scoring='accuracy')

    >>> grid_search.fit(X_train, y_train)
    ```

  - Predict using the best parameters of grid search:

```
>>> y_pred = grid_search.predict(X_test)
```

- The output is as follows:

```
>>> print ('\n Best score: \n', grid_search.best_score_)
>>> print ('\n Best parameters set: \n')
>>> best_parameters = grid_search.best_estimator_.get_params()
>>> for param_name in sorted(parameters.keys()):
>>>         print ('\t%s: %r' % (param_name, best_parameters[param_name]))
>>> print ("\n Confusion Matrix on Test data
\n",confusion_matrix(y_test,y_pred))
>>> print ("\n Test Accuracy \n",accuracy_score(y_test,y_pred))
>>> print ("\nPrecision Recall f1 table \n",classification_report(y_test, y_pred))
```

XIV MACHINE LEARNING MODEL OVERVIEW

- Supervised learning: This is where an instructor provides feedback to a student on whether they have performed well in an examination or not. In which target variable do present and models do get tune to achieve it. Many machine learning methods fall in to this category:
    - Classification problems
    - Logistic regression
    - Lasso and ridge regression Decision trees (classification trees) Bagging classifier
    - Random forest classifier
    - Boosting classifier (adaboost, gradient boost, and xgboost)
    - SVM classifier
    - Recommendation engine Regression problems
    - Linear regression (lasso and ridge regression) Decision trees (regression trees)
    - Bagging regressor Random forest regressor
    - Boosting regressor - (adaboost, gradient boost, and xgboost)
    - SVM regressor
- Unsupervised learning: Similar to the teacher-student analogy, in which the instructor does not present and provide feedback to the student and who needs to prepare on his/her own. Unsupervised learning does not have as many are in supervised learning:
    - Principal component analysis (PCA)

- K-means clustering

- Reinforcement learning: This is the scenario in which multiple decisions need to be taken by an agent prior to reaching the target and it provides a reward, either +1 or -1, rather than notifying how well or how badly the agent performed across the path:
  - Markov decision process
  - Monte Carlo methods Temporal difference learning

The different Machine Learning algorithms are

1. **Logistic regression**:
   - This is the problem in which outcomes are discrete classes rather than continuous values.
   - For example, a customer will arrive or not, he will purchase the product or not, and so on.
   - In statistical methodology, it uses the maximum likelihood method to calculate the parameter of individual variables. In contrast, in machine learning methodology, log loss will be minimized with respect to β coefficients (also known as weights).
   - Logistic regression has a high bias and a low variance error.
2. **Linear regression**:
   - This is used for the prediction of continuous variables such as customer income and so on.
   - It utilizes error minimization to fit the best possible line in statistical methodology. However, in machine learning methodology, squared loss will be minimized with respect to β coefficients.
   - Linear regression also has a high bias and a low variance error
3. **Lasso and ridge regression**:
   - This uses regularization to control overfitting issues by applying a penalty on coefficients.
   - In ridge regression, a penalty is applied on the sum of squares of coefficients, whereas in lasso, a penalty is applied on the absolute values of the coefficients.
   - The penalty can be tuned in order to change the dynamics of the model fit. Ridge regression tries to minimize the magnitude of coefficients, whereas lasso tries to eliminate them
4. **Decision trees**:
   - Recursive binary splitting is applied to split the classes at each level to classify observations to their purest class.

- The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class.
- Decision trees have an overfitting problem due to their high variance in a way to fit; pruning is applied to reduce the overfitting problem by growing the tree completely.
- Decision trees have low a bias and a high variance error

5. **Bagging**:
    - This is an ensemble technique applied on decision trees in order to minimize the variance error and at the same time not increase the error component due to bias.
    - In bagging, various samples are selected with a subsample of observations and all variables (columns), subsequently fit individual decision trees independently on each sample and later ensemble the results by taking the maximum vote (in regression cases, the mean of outcomes calculated)
6. **Random forest**:
    - This is similar to bagging except for one difference. In bagging, all the variables/columns are selected for each sample, whereas in random forest a few subcolumns are selected.
    - The reason behind the selection of a few variables rather than all was that during each independent tree sampled, significant variables always came first in the top layer of splitting which makes all the trees look more or less similar and defies the sole purpose of ensemble: that it works better on diversified and independent individual models rather than correlated individual models.
    - Random forest has both low bias and variance errors.
7. **Boosting**:
    - This is a sequential algorithm that applies on weak classifiers such as a decision stump (a one-level decision tree or a tree with one root node and two terminal nodes) to create a strong classifier by ensembling the results.
    - The algorithm starts with equal weights assigned to all the observations, followed by subsequent iterations where more focus was given to misclassified observations by increasing the weight of misclassified observations and decreasing the weight of properly classified observations. In the end, all the individual classifiers were combined to create a strong classifier.
    - Boosting might have an overfitting problem, but by carefully tuning the parameters, we can obtain the best of the self machine learning model
8. **Support vector machines (SVMs)**:

- This maximizes the margin between classes by fitting the widest possible hyperplane between them.
- In the case of non-linearly separable classes, it uses kernels to move observations into higher-dimensional space and then separates them linearly with the hyperplane there

9. **Recommendation engine**:
    - This utilizes a collaborative filtering algorithm to identify high-probability items to its respective users, who have not used it in the past, by considering the tastes of similar users who would be using that particular item.
    - It uses the **alternating least squares** (**ALS**) methodology to solve this problem
10. **Principal component analysis (PCA)**: This is a dimensionality reduction technique in which principal components are calculated in place of the original variable. Principal components are determined where the variance in data is maximum; subsequently, the top *n* components will be taken by covering about 80 percent of variance and will be used in further modeling processes, or exploratory analysis will be performed as unsupervised learning
11. **K-means clustering**: This is an unsupervised algorithm that is mainly utilized for segmentation exercise. K-means clustering classifies the given data into *k* clusters in such a way that, within the cluster, variation is minimal and across the cluster, variation is maximal
12. **Markov decision process (MDP)**:
    - In reinforcement learning, MDP is a mathematical framework for modeling decision-making of an agent in situations or environments where outcomes are partly random and partly under control. In this model, environment is modeled as a set of states and actions that can be performed by an agent to control the system's state.
    - The objective is to control the system in such a way that the agent's total payoff is maximized
13. **Monte Carlo method**:
    - Monte Carlo methods do not require complete knowledge of the environment, in contrast with MDP.
    - Monte Carlo methods require only experience, which is obtained by sample sequences of states, actions, and rewards from actual or simulated interaction with the environment.
    - Monte Carlo methods explore the space until the final outcome of a chosen sample sequences and update estimates accordingly
14. **Temporal difference learning**:

- This is a core theme in reinforcement learning. Temporal difference is a combination of both Monte Carlo and dynamic programming ideas.
- Similar to Monte Carlo, temporal difference methods can learn directly from raw experience without a model of the environment's dynamics.
- Like dynamic programming, temporal difference methods update estimates based in part on other learned estimates, without waiting for a final outcome.
- Temporal difference is the best of both worlds and is most commonly used in games such as AlphaGo and so on

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CAMPUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2021-22(EVEN SEMESTER)

18CSE479T - Statistical Machine Learning

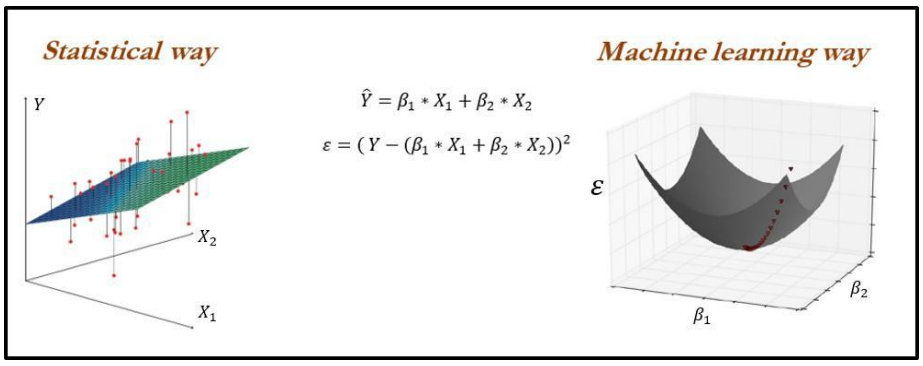
Unit - II

Comparison between regression and machine learning models-Compensating factors in machine learning models- Assumptions of linear regression Steps applied in linear regression modeling- Example of simple linear regression from first principles- Machine learning models - ridge and lasso regression-Example of ridge regression machine learning, Example of lasso regression machine learning model- Logistic Regression Versus Random Forest-Maximum likelihood estimation- Terminology involved in logistic regression- Applying steps in logistic regression modeling - Random forest-Example of random forest using German credit data -Grid search on random forest - Variable importance plot- Comparison of logistic regression with random forest.

I PARALLELISM OF STATISTICS AND MACHINE LEARNING

Comparison between regression and machine learning models

- Linear regression and machine learning models both try to solve the same problem in different ways.
- In the following simple example of a two-variable equation fitting the best possible plane, regression models try to fit the best possible hyperplane by minimizing the errors between the hyperplane and actual observations. However, in machine learning, the same problem has been converted into an optimization problem in which errors are modeled in squared form to minimize errors by altering the weights



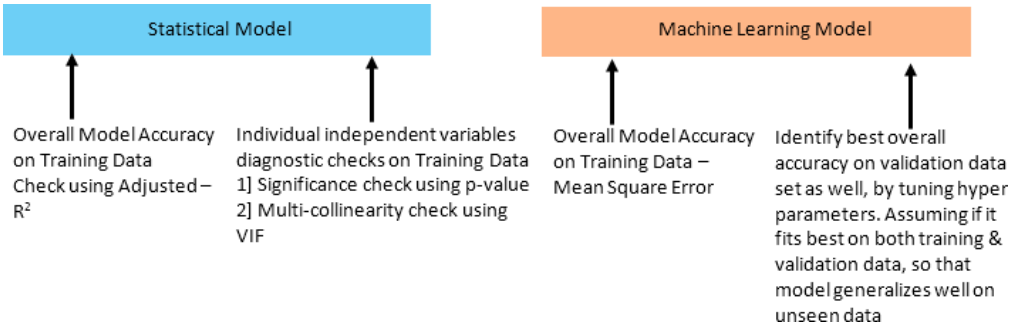
- In statistical modeling, samples are drawn from the population and the model will be fitted on sampled data. However, in machine learning, even small numbers such as 30 observations would be good enough to update the weights at the end of each iteration; in a few cases, such as online learning, the model will be updated with even one observation
- Statistical models are parametric in nature, which means a model will have parameters on which diagnostics are performed to check the validity of the model. Whereas machine learning models are non-parametric, do not have any parameters, or curve assumptions; these models learn by themselves based on provided data and come up with complex and intricate functions rather than predefined function fitting.
- Multi-collinearity checks are required to be performed in statistical modeling. Whereas, in machine learning space, weights automatically get adjusted to compensate the multicollinearity problem

Note : Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. This means that an independent variable can be predicted from another independent variable in a regression model

II Compensating Factors In Machine Learning Model

- Compensating factors in machine learning models to equate statistical diagnostics is explained with the example of a beam being supported by two supports. If one of the supports doesn't exist, the beam will eventually fall down by moving out of balance.
- A similar analogy is applied for comparing statistical modeling and machine learning methodologies here.
- The two-point validation is performed on the statistical modeling methodology on training data using overall model accuracy and individual parameters significance test. Due to the fact that either linear or logistic regression has less variance by shape of the model itself, hence there would be very little chance of it working worse on unseen data. Hence, during deployment, these models do not incur too many deviated results.

- However, in the machine learning space, models have a high degree of flexibility which can change from simple to highly complex.
- On top, statistical diagnostics on individual variables are not performed in machine learning. Hence, it is important to ensure the robustness to avoid overfitting of the models, which will ensure its usability during the implementation phase to ensure correct usage on unseen data.
- In machine learning, data will be split into three parts (train data - 50 percent, validation data - 25 percent, testing data - 25 percent) rather than two parts in statistical methodology.
- Machine learning models should be developed on training data, and its hyperparameters should be tuned based on validation data to ensure the two-point validation equivalence; this way, the robustness of models is ensured without diagnostics performed at an individual variable level



III ASSUMPTIONS OF LINEAR REGRESSION

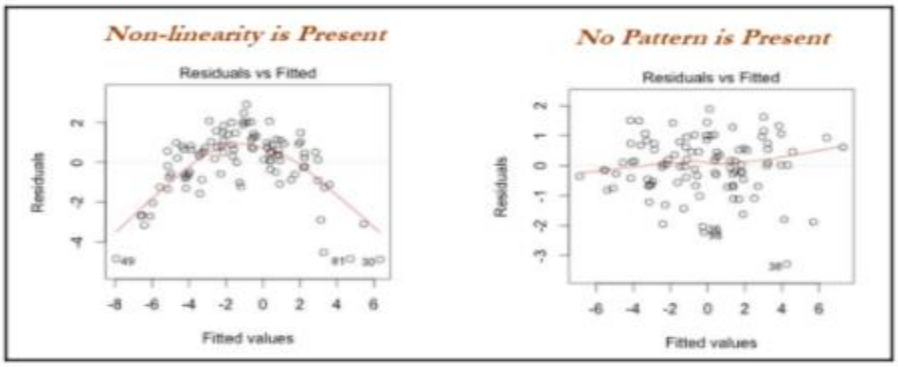
Linear regression has the following assumptions, failing which the linear regression model does not hold true:

- The dependent variable should be a linear combination of independent variables
- No autocorrelation in error terms
- Errors should have zero mean and be normally distributed
- No or little multi-collinearity
- Error terms should be homoscedastic

1.The dependent variable should be a linear combination of independent variables:

Y should be a linear combination of *X* variables. Please note, in the following equation, *X2* has raised to the power of *2*, the equation is still holding the assumption of a linear combination of variables

$$Y = A_0 + (\beta_1 * x_1) + (\beta_2 * x_2^2)$$



In the preceding sample graph, initially, linear regression was applied and the errors seem to have a pattern rather than being pure white noise; in this case, it is simply showing the presence of non-linearity. After increasing the power of the polynomial value, now the errors simply look like white noise.

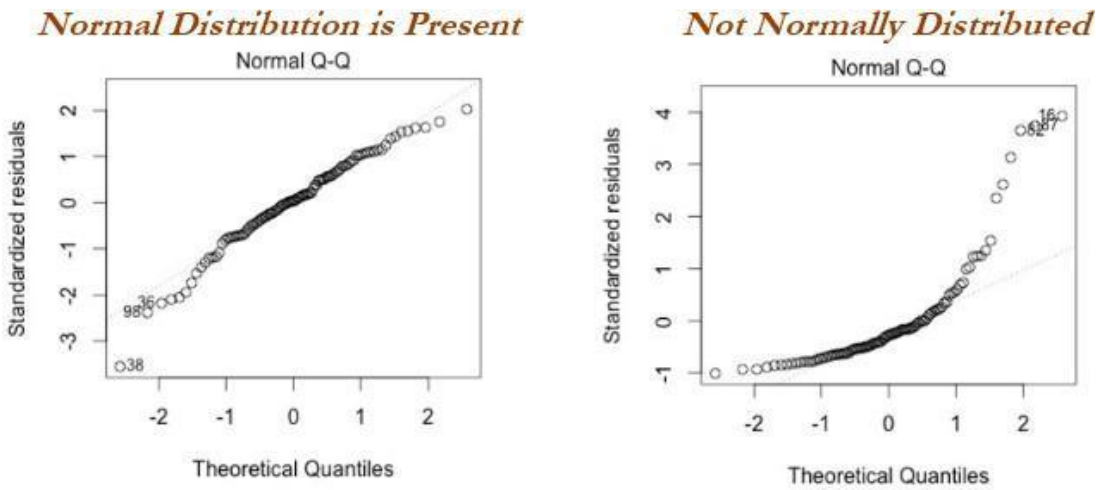
2.No autocorrelation in error terms

Presence of correlation in error terms penalized model accuracy

To diagnose: Look for the Durbin-Watson test. Durbin-Watson's *d* tests the null hypothesis that the residuals are not linearly auto correlated. While *d* can lie between *0* and *4*, if $d \approx 2$ indicates no autocorrelation, $0<d<2$ implies positive autocorrelation, and $2<d<4$ indicates negative autocorrelation

3.Error should have zero mean and be normally distributed

Errors should have zero mean for the model to create an unbiased estimate. Plotting the errors will show the distribution of errors. Whereas, if error terms are not normally distributed, it implies confidence intervals will become too wide or narrow, which leads to difficulty in estimating coefficients based on minimization of least squares:



To diagnose: Look into Q-Q plot and also tests such as Kolmogorov- Smirnov tests will be helpful. By looking into the above Q-Q plot, it is evident that the first chart shows errors are normally distributed, as the residuals do not seem to be deviating much compared with the diagonal-like line, whereas in the right-hand chart, it is clearly showing that errors are not normally distributed; in these scenarios, we need to reevaluate the variables by taking log transformations and so on to make residuals look as they do on the left-hand chart.

4.No or little multi-collinearity:

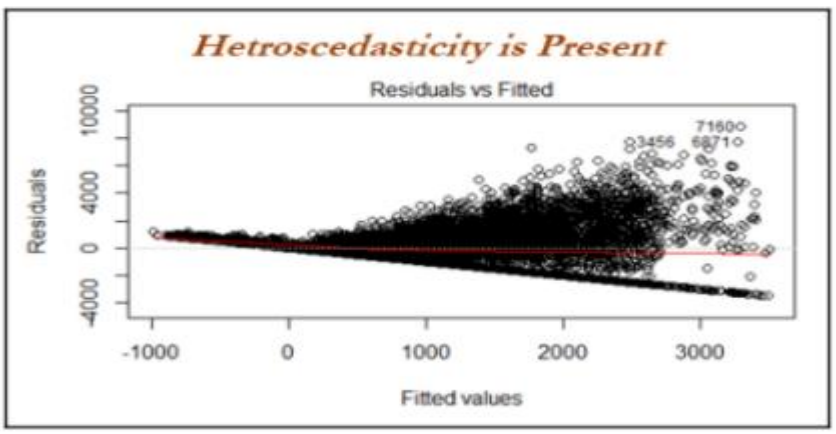
Multi-collinearity is the case in which independent variables are correlated with each other and this situation creates unstable models by inflating the magnitude of coefficients/estimates. It also becomes difficult to determine which variable is contributing to predict the response variable. *VIF* is calculated for each independent variable by calculating the R-squared value with respect to all the other independent variables and tries to eliminate which variable has the highest *VIF* value one by one:

$$VIF = \frac{1}{1 - R^2}$$

To diagnose: Look into scatter plots, run correlation coefficient on all the variables of data. Calculate the **variance inflation factor** (**VIF**). If *VIF <= 4* suggests no multi-collinearity, in banking scenarios, people use *VIF <= 2* also.

5.Errors should be homoscedastic

Errors should have constant variance with respect to the independent variable, which leads to impractically wide or narrow confidence intervals for estimates, which degrades the model's performance. One reason for not holding homoscedasticity is due to the presence of outliers in the data, which drags the model fit toward them with higher weights



To diagnose: Look into the residual versus dependent variables plot; if any pattern of cone or divergence does exist, it indicates the errors do not have constant variance, which impacts its predictions.

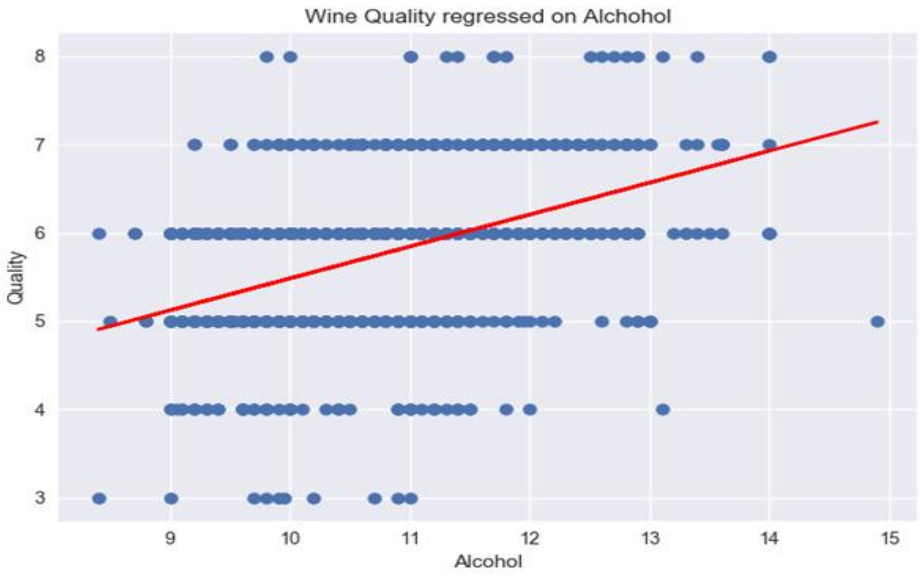
IV STEPS APPLIED IN LINEAR REGRESSION MODELING

The following steps are applied in linear regression modeling in industry:

1. Missing value and outlier treatment
2. Correlation check of independent variables
3. Train and test random classification
4. Fit the model on train data
5. Evaluate model on test data

V EXAMPLE OF SIMPLE LINEAR REGRESSION FROM FIRST PRINCIPLES

- UCI machine learning repository at https://archive.ics.uci.edu/ml/datasets/Wine+Quality.
- Simple linear regression is a straightforward approach for predicting the dependent/response variable *Y* given the independent/predictor variable *X*. It assumes a linear relationship between *X* and *Y*
- $\beta 0$ and $\beta 1$ are two unknown constants which are intercept and slope parameters respectively. Once we determine the constants, we can utilize them for the prediction of the dependent variable
- Residuals are the differences between the *i*th observed response value and the *i*th response value that is predicted from the model. Residual sum of squares is shown. The least squares approach chooses estimates by minimizing errors



$$Y \approx \beta_0 + \beta_1 X\,; \quad \hat{y} = \beta_0 + \beta_1 X$$

$$\hat{y}_i = \widehat{\beta_0} + \widehat{\beta_1} x_i \Rightarrow e_i = y_i - \hat{y}_i$$

$$RSS\ (Residual\ sum\ of\ squares) = e_1^2 + e_2^2 + \cdots + e_n^2$$

$$RSS = (y_1 - \widehat{\beta_0} - \widehat{\beta_1} x_1)^2 + (y_2 - \widehat{\beta_0} - \widehat{\beta_1} x_2)^2 + \cdots + (y_n - \widehat{\beta_0} - \widehat{\beta_1} x_n)^2$$

$$\widehat{\beta_1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{cov(x,y)}{var(x)}; \widehat{\beta_0} = \bar{y} - \widehat{\beta_1}\,\bar{x}$$

$$\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i \;;\; \bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$$

- In order to prove statistically that linear regression is significant, we have to perform hypothesis testing.
- Let's assume we start with the null hypothesis that there is no significant relationship between *X* and *Y*
- Since, if $\beta 1 = 0$, then the model shows no association between both variables ($Y = \beta 0 + \varepsilon$), these are the null hypothesis assumptions;
- In order to prove this assumption right or wrong, we need to determine $\beta 1$ is sufficiently far from *0 so* that we can be confident that $\beta 1$ is nonzero and have a significant relationship between both variables

H_0: *There is no relationship between X and Y*

H_a: *There is relationship between X and Y*

$$H_0: \beta_1 = 0 \; ; \quad H_a: \beta_1 \neq 0$$

- It depends on the distribution of *β1*, which is its mean and standard error (similar to standard deviation).
- In some cases, if the standard error is small, even relatively small values may provide strong evidence that *β1 ≠ 0*, hence there is a relationship between *X* and *Y*.
- In contrast, if *SE(β1)* is large, then *β1* must be large in absolute value in order for us to reject the null hypothesis.

$$t = \frac{\widehat{\beta_1} - 0}{SE(\widehat{\beta_1})}$$

- We usually perform the t test to check how many standard deviations *β1* is away from the value *0*:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \; ; \; SS_{tot} = \sum_i (y_i - \bar{y})^2 \; ; \; SS_{reg} = \sum_i (f_i - \bar{y})^2 \; ; \; SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}}$$

- With this *t* value, we calculate the probability of observing any value equal to |*t*| or larger, assuming *β1 = 0*; this probability is also known as the p-value.
- If *p-value < 0.05*, it signifies that *β1* is significantly far from *0*, hence we can reject the null hypothesis and agree that there exists a strong relationship if *p-value > 0.05*, we accept the null hypothesis and conclude that there is no significant relationship between both variables
- To predict the dependent value and check for the R-squared value;
- if the value is *>= 0.7*, it means the model is good enough to deploy on unseen data
- if it is not such a good value (*<0.6*), we can conclude that this model is not good enough to deploy

VI MACHINE LEARNING MODELS – INTRODUCTION TO RIDGE AND LASSO REGRESSION

- In linear regression, only the **residual sum of squares** (**RSS**) is minimized
- In ridge and lasso regression, a penalty is applied (also known as **shrinkage penalty**) on coefficient values to regularize the coefficients with the tuning parameter λ
- When $\lambda=0$, the penalty has no impact, ridge/lasso produces the same result as linear regression, whereas λ -> ∞ will bring coefficients to zero.

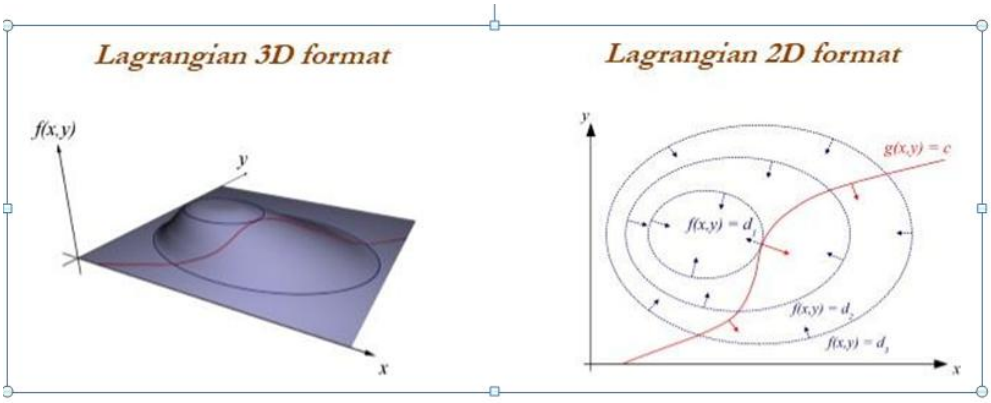
$$RSS\ (\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$$

Lagrangian multipliers

- **Defintion of Lagrangian** : a function that describes the state of a dynamic system in terms of position coordinates and their time derivatives
- The Lagrange multiplier, λ, measures the increase in the objective function (f(x, y) that is obtained through a marginal relaxation in the constraint (an increase in k). For this reason, the Lagrange multiplier is often termed a shadow price.
- The method of Lagrange multipliers in Machine learning is a simple and elegant method of finding the local minima or local maxima of a function subject to equality or inequality constraints. Lagrange multipliers are also called undetermined multipliers.
- The objective is RSS subjected to cost constraint (*s*) of budget.
- For every value of λ, there is an *s* such that will provide the equivalent equations, as shown for the overall objective function with a penalty factor

$$Objective\ function\ to\ minimize\ in\ Ridge\ Regression = RSS(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$Objective\ function\ to\ minimize\ in\ Lasso\ Regression = RSS(\beta) + \lambda \sum_{j=1}^{p} |\beta_j|$$



- For any fixed value of λ, ridge regression only fits a single model and the model-fitting procedure can be performed very quickly
- One disadvantage of ridge regression
    - Given a situation where the number of predictors is significantly large, using ridge may provide accuracy, but it includes all the variables, which is not desired in a compact representation of the model;
- But in lasso, it will set the weights of unnecessary variables to zero

VII RIDGE REGRESSION

What is Ridge Regression?

- Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.
- The cost function for ridge regression:

  Min(||Y – X(theta)||^2 + λ||theta||^2)

  Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the values of alpha, the bigger is the penalty and therefore the magnitude of coefficients is reduced.
- It shrinks the parameters. Therefore, it is used to prevent multicollinearity
- It reduces the model complexity by coefficient shrinkage

Lasso Meaning

- The word "LASSO" stands for Least Absolute Shrinkage and Selection Operator. It is a statistical formula for the regularisation of data models and feature selection.

Regularization

- Regularization is an important concept that is used to avoid overfitting of the data, especially when the trained and test data are much varying.
- Regularization is implemented by adding a "penalty" term to the best fit derived from the trained data, to achieve a lesser variance with the tested data and also restricts the influence of predictor variables over the output variable by compressing their coefficients.

Mathematical equation of Lasso Regression

Residual Sum of Squares + λ * (Sum of the absolute value of the magnitude of coefficients)

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where,

- λ denotes the amount of shrinkage.
- λ = 0 implies all features are considered and it is equivalent to the linear regression where only the residual sum of squares is considered to build a predictive model
- λ = ∞ implies no feature is considered i.e, as λ closes to infinity it eliminates more and more features
- The bias increases with increase in λ
- variance increases with decrease in λ

The key difference is in how they assign penalty to the coefficients:

1. Ridge Regression:
    - Performs L2 regularization, i.e. adds penalty equivalent to square of the magnitude of coefficients
    - Minimization objective = LS Obj + α * (sum of square of coefficients)
2. Lasso Regression:

- Performs L1 regularization, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

- Ridge regression is a machine learning model in which we do not perform any statistical diagnostics on the independent variables and just utilize the model to fit on test data and check the accuracy of the fit. Here, we have used the scikit-learn package

```
>>> from sklearn.linear_model import Ridge

>>> wine_quality = pd.read_csv("winequality-red.csv",sep=';')
>>> wine_quality.rename(columns=lambda x: x.replace(" ", "_"),
inplace=True)


>>> all_colnms = ['fixed_acidity', 'volatile_acidity', 'citric_acid',
'residual_sugar', 'chlorides', 'free_sulfur_dioxide',
'total_sulfur_dioxide', 'density', 'pH', 'sulphates', 'alcohol']

>>> pdx = wine_quality[all_colnms]
>>> pdy = wine_quality["quality"]

>>> x_train,x_test,y_train,y_test = train_test_split(pdx,pdy,train_size =
0.7,random_state=42)
```

A simple version of a grid search from scratch is described as follows, in which various values of alphas are to be tested in a grid search to test the model's fitness:

```
>>> alphas = [1e-4,1e-3,1e-2,0.1,0.5,1.0,5.0,10.0]
```

Initial values of R-squared are set to 0 in order to keep track of its updated value and to print whenever the new value is greater than the existing value:

```
>>> initrsq = 0

>>> print ("\nRidge Regression: Best Parameters\n")
>>> for alph in alphas:
...       ridge_reg = Ridge(alpha=alph)
...       ridge_reg.fit(x_train,y_train)    0
...       tr_rsqrd = ridge_reg.score(x_train,y_train)
...       ts_rsqrd = ridge_reg.score(x_test,y_test)
```

The following code always keeps track of the test R-squared value and prints if the new value is greater than the existing best value:

```
>>>      if ts_rsqrd > initrsq:
...           print ("Lambda: ",alph,"Train R-Squared
value:",round(tr_rsqrd,5),"Test R-squared value:",round(ts_rsqrd,5))
...           initrsq = ts_rsqrd
```

This is shown in the following screenshot:

```
Ridge Regression: Best Parameters

Lambda:  0.0001 Train R-Squared value: 0.3612 Test R-squared value: 0.35135
```

Lasso regression is a close cousin of ridge regression, in which absolute values of coefficients are minimized rather than the square of values

The following implementation is similar to ridge regression apart from penalty application on mod/absolute value of coefficients:

```
>>> from sklearn.linear_model  import Lasso

>>> alphas = [1e-4,1e-3,1e-2,0.1,0.5,1.0,5.0,10.0]
>>> initrsq = 0
>>> print ("\nLasso Regression: Best Parameters\n")

>>> for alph in alphas:
...       lasso_reg = Lasso(alpha=alph)
...       lasso_reg.fit(x_train,y_train)
...       tr_rsqrd = lasso_reg.score(x_train,y_train)
...       ts_rsqrd = lasso_reg.score(x_test,y_test)

...       if ts_rsqrd > initrsq:
...           print ("Lambda: ",alph,"Train R-Squared
value:",round(tr_rsqrd,5),"Test  R-squared value:",round(ts_rsqrd,5))
...           initrsq = ts_rsqrd
```

```
Lasso Regression: Best Parameters

Lambda:  0.0001 Train R-Squared value: 0.36101 Test R-squared value: 0.35057
```

```
>>> ridge_reg = Ridge(alpha=0.001)
>>> ridge_reg.fit(x_train,y_train)
>>> print ("\nRidge Regression coefficient values of Alpha = 0.001\n")
>>> for i in range(11):
...     print (all_colnms[i],": ",ridge_reg.coef_[i])

>>> lasso_reg = Lasso(alpha=0.001)
>>> lasso_reg.fit(x_train,y_train)
>>> print ("\nLasso Regression coefficient values of Alpha = 0.001\n")
>>> for i in range(11):
...      print (all_colnms[i],": ",lasso_reg.coef_[i])
```

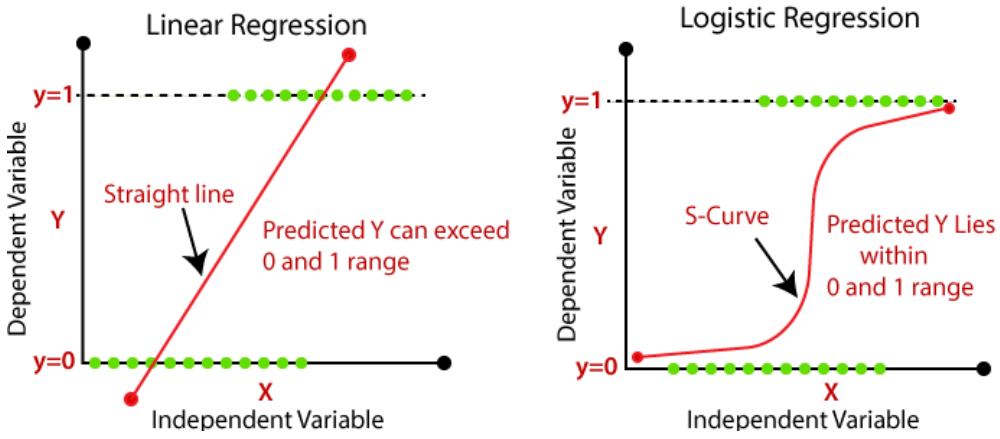
The following results show the coefficient values of both methods; the coefficient of density has been set to 0 in lasso regression, whereas the density value is -5.5672 in ridge regression; also, none of the coefficients in ridge regression are zero value

```
Ridge Regression coeficcient values of Alpha = 0.001

fixed_acidity :  0.015506587508
volatile_acidity :  -1.10509823549
citric_acid :  -0.248798655324
residual_sugar :  0.00401889539284
chlorides :  -1.68438396209
free_sulfur_dioxide :  0.00463690171096
total_sulfur_dioxide :  -0.00328376790411
density :  -5.5672717468
pH :  -0.362480017204
sulphates :  0.800919122803
alcohol :  0.299918244295
```

```
Lasso Regression coeficcient values of Alpha = 0.001

fixed_acidity :  0.0141495463691
volatile_acidity :  -1.09062360905
citric_acid :  -0.185295150047
residual_sugar :  -0.000136610246787
chlorides :  -1.05877579704
free_sulfur_dioxide :  0.00483164817515
total_sulfur_dioxide :  -0.00326722885596
density :  -0.0
pH :  -0.256901925871
sulphates :  0.694487540316
alcohol :  0.307756149124
```

VIII LOGISTIC REGRESSION

- Logistic regression is a process of modeling the probability of a discrete outcome given an input variable.
- It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation. This type of analysis can help you predict the likelihood of an event happening or a choice being made



Maximum Likelihood

- Maximum likelihood estimation is a method of estimating the parameters of a model given observations, by finding the parameter values that maximize the likelihood of making the observations, this means finding parameters that maximize the probability *p* of event *1* and *(1-p)* of non-event *0*

  probability (event + non-event) = 1

Example: Sample *(0, 1, 0, 0, 1, 0)* is drawn from binomial distribution. What is the maximum likelihood estimate of μ?

Solution: Given the fact that for binomial distribution $P(X=1) = \mu$ and $P(X=0) = 1- \mu$ where μ is the parameter:

$$\begin{aligned} Likelihood\ of\ \mu = L(\mu) \\ &= P(x=0) * P(x=1) * P(x=0) * P(x=0) * P(x=1) * P(x=0) \\ &= (1-\mu) * \mu * (1-\mu) * (1-\mu) * \mu * (1-\mu) \\ &L(\mu) = (1-\mu)^4 \mu^2 \end{aligned}$$

Here, *log* is applied to both sides of the equation for mathematical convenience; also, maximizing likelihood is the same as the maximizing log of likelihood:

$$\log(L(\mu)) = \log((1-\mu)^4 \mu^2)$$
$$\log(L(\mu)) = 4 * \log(1-\mu) + 2 * \log(\mu)$$

Determining the maximum value of μ by equating derivative to zero:

$$\frac{\partial}{\partial \mu} \log(L(\mu)) = 0$$
$$\Rightarrow 4 * \frac{1}{1-\mu} * (-1) + 2 * \frac{1}{\mu} = 0$$
$$-4 * \mu + 2 * (1-\mu) = 0$$
$$\mu = \frac{1}{3}$$

However, we need to do double differentiation to determine the saddle point obtained from equating derivative to zero is maximum or minimum. If the μ value is maximum; double differentiation of *log(L(μ))* should be a negative value:

$$\frac{\partial^2}{\partial \mu^2} \log(L(\mu)) = -4 * \frac{1}{(1-\mu)^2} - \frac{2}{\mu^2}$$

Even without substitution of μ value in double differentiation, we can determine that it is a negative value, as denominator values are squared and it has a negative sign against both terms. Nonetheless, we are substituting and the value is:

$$\frac{\partial^2}{\partial \mu^2}\log(L(\mu)) = -4 * \frac{1}{\left(1-\frac{1}{3}\right)^2} - \frac{2}{\left(\frac{1}{3}\right)^2} = -9 - 18 = -27$$

Hence it has been proven that at value $\mu = 1/3$, it is maximizing the likelihood. If we substitute the value in the log likelihood function, we will obtain:

$$L(\mu) = \left(1-\frac{1}{3}\right)^4 \frac{1}{3}^2 = 0.021948$$
$$\ln(L(\mu)) = \ln(0.021948) = -3.819$$
$$-2\ln(L(\mu)) = -2 * -3.819 = 7.63$$

The reason behind calculating *-2*ln(L)* is to replicate the metric calculated in proper logistic regression. In fact:

*AIC = -2*ln(L) + 2*k*

So, Logistic regression tries to find the parameters by maximizing the likelihood with respect to individual parameters.

IX TERMINOLOGY INVOLVED IN LOGISTIC REGRESSION

1.Information value (IV)

- This is very useful in the preliminary filtering of variables prior to including them in the model.
- IV is mainly used by industry for eliminating major variables in the first step prior to fitting the model, as the number of variables present in the final model would be about 10.
- Hence, initial processing is needed to reduce variables from 400+ in number or so.

$$Information\ Value = \ln\left(\frac{\%good}{\%bad}\right) * (\%good - \%bad)$$

$$Weight\ Of\ Evidence = \ln\left(\frac{\%good}{\%bad}\right)$$

Information Value	Predictive Power
<0.02	Useless for prediction
0.02 to 0.1	Weak predictor
0.1 to 0.3	Medium predictor
0.3 to 0.5	Strong predictor
> 0.5	Suspicious or too good predictor

Example: In the following table, continuous variable (price) has been broken down into deciles (10 bins) based on price range and the counted number of events and non-events in that bin, and the information value has been calculated for all the segments and added together. We got the total value as *0.0356*, meaning it is a weak predictor to classify events.

Range	Bin Number	Events	Non Events	%of Events [E] (Events/Total Events)	% of Non-Events [NE] (Non Events/Total Non Events)	[E] -[NE]	WOE = ln(E/NE)	IV = WOE * (E-NE)
0-50	1	40	394	5%	5%	0.003	0.062	0.0002
51-100	2	68	900	9%	12%	-0.024	-0.234	0.0056
101-150	3	78	984	10%	13%	-0.021	-0.186	0.0040
151-200	4	102	1194	14%	15%	-0.016	-0.111	0.0018
201-250	5	108	1218	15%	16%	-0.011	-0.074	0.0008
251-300	6	110	1164	15%	15%	-0.001	-0.010	0.0000
301-350	7	82	772	11%	10%	0.011	0.107	0.0012
351-400	8	46	330	6%	4%	0.019	0.379	0.0074
401-450	9	42	368	6%	5%	0.009	0.179	0.0017
>451	10	68	470	9%	6%	0.031	0.416	0.0129
	Total	**744**	**7794**				**I.V.Total**	**0.0356**

2.Akaike information criteria (AIC):

- The Akaike information criterion (AIC) is a mathematical method for evaluating how well a model fits the data it was generated from. In statistics, AIC is used to compare different possible models and determine which one is the best fit for the data.
- This measures the relative quality of a statistical model for a given set of data.
- During a comparison between two models, the model with less AIC is preferred over higher value

$$AIC = -2*ln(L) + 2*k$$

L = Maximum value of Likelihood (log transformation applied for mathematical convenience)

k = Number of variables in the model

3. **Receiver operating characteristic (ROC) curve**:

- This is a graphical plot that illustrates the performance of a binary classifier as its discriminant threshold is varied.
- The curve is created by plotting **true positive rate** (**TPR**) against **false positive rate** (**FPR**) at various threshold values.
- A threshold is a real value between *0* and *1*, used to convert the predicted probability of output into class.
- Ideally, the threshold should be set in a way that trade-offs value between both categories and produces higher overall accuracy
- *Optimum threshold = Threshold where maximum (sensitivity + specificity) is possible*

4. ***Confuion Matrix***

	Predicted: YES	Predicted: NO
Actual: YES	TP (True Positive)	FN (False Negative)
Actual: NO	FP (False Positive)	TN (True Negative)

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

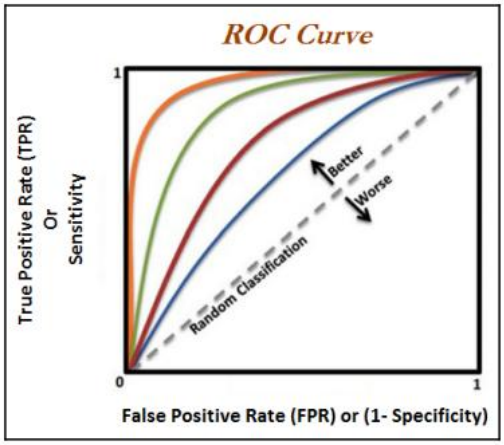
$$Precision = \frac{TP}{TP + FP}$$

$$Recall\ (Sensitivity\ or\ TPR) = \frac{TP}{TP + FN}$$

$$f1\ score = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$FPR\ (1 - Specificity) = \frac{FP}{FP + TN}$$

$$Specificity\ (TNR) = \frac{TN}{FP + TN}$$



5. **Rank ordering**:
    - After sorting observations in descending order by predicted probabilities, deciles are created (10 equal bins with 10 percent of total observations in each bin).
    - By adding up the number of events in each decile, we will get aggregated events for each decile and this number should be in decreasing order, else it will be in serious violation of logistic regression methodology
6. **Concordance/c-statistic**:
    - The C-statistic (sometimes called the "concordance" statistic or C-index) is a measure of goodness of fit for binary outcomes in a logistic regression model.
    - This is a measure of quality of fit for a binary outcome in a logistic regression model.
    - It is a proportion of pairs in which the predicted event probability is higher for the actual event than non-event
    - In the following table, both actual and predicted values are shown with a sample of seven rows. Actual is the true category, either default or not; whereas predicted is predicted probabilities from the logistic regression model. Calculate the concordance value

Actual	Predicted
1	0.92
0	0.34
0	0.12
1	0.4
1	0.64
0	0.82
1	0.84

Actual	Predicted
1	0.92
1	0.4
1	0.64
1	0.84

Actual	Predicted
0	0.34
0	0.12
0	0.82

- For calculating concordance, we need to split the table into two (each table with actual values as *1* and *0*) and apply the Cartesian product of each row from both tables to form pairs
- The complete Cartesian product has been calculated and has classified the pair as a concordant pair whenever the predicted probability for *1* category is higher than the predicted probability for *0* category

Actual	Predicted	Actual	Predicted	Concordant pair	Discordant pair
1	0.92	0	0.34	✓	
1	0.92	0	0.12	✓	
1	0.92	0	0.82	✓	
1	0.4	0	0.34	✓	
1	0.4	0	0.12	✓	
1	0.4	0	0.82		✓
1	0.64	0	0.34	✓	
1	0.64	0	0.12	✓	
1	0.64	0	0.82		✓
1	0.84	0	0.34	✓	
1	0.84	0	0.12	✓	
1	0.84	0	0.82	✓	

$$\%\ concordant\ pairs = \frac{Number\ of\ Concordant\ pairs}{Total\ pairs} = \frac{10}{12} = 83.3\%$$

$$\%\ discordant\ pairs = \frac{Number\ of\ Discordant\ pairs}{Total\ pairs} = \frac{2}{12} = 16.67\%$$

$$\%\ tied\ pairs = \frac{Number\ of\ Tied\ pairs}{Total\ pairs} = \frac{0}{12} = 0\%$$

$$c\ statistic\ or\ c\ index = 0.5 + \frac{(\%Concordant\ pairs - \%Discordant\ pairs)}{2}$$
$$= 0.5 + \frac{(0.833 - 0.1667)}{2} = 0.83315 = 83.315\ \%$$

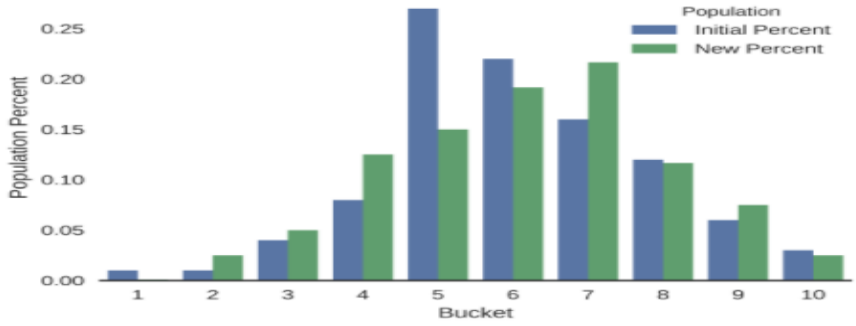
$$Somers'D = (\%\ Concordant\ pairs - \%\ Discordant\ pairs) = (0.833 - 0.1667)$$
$$= 0.663$$

- **C-statistic**: This is 0.83315 percent or 83.315 percent, and any value greater than 0.7 percent or 70 percent is considered a good model to use for practical purposes

7. **Divergence**: The distance between the average score of default accounts and the average score of non-default accounts. The greater the distance, the more effective the scoring system is at segregating good and bad observations.
8. **K-S statistic**: This is the maximum distance between two population distributions. It helps with discriminating default accounts from non-default accounts
9. **Population stability index (PSI)**:

- This is the metric used to check that drift in the current population on which the credit scoring model will be used is the same as the population with respective to development time:
- *PSI <= 0.1*: This states no change in characteristics of the current population with respect to the development population
- *0.1 < PSI <= 0.25*: This signifies some change has taken place and warns for attention, but can still be used
- *PSI >0.25*: This indicates a large shift in the score distribution of the current population compared with development time
- To calculate the PSI we first divide the initial population range into 10 buckets (an arbitrary number I chose), and count the number of values in each of those buckets for the initial and new populations, and then divide those by the total values in each

population to get the percents in each bucket. As expected, plotting the percents ends up looking like a discretized version of the original chart:



From here, we perform the actual PSI calculation for each bucket, and them sum them all up to get the overall PSI values for the distributions.

PSI Formula:

$$PSI = \sum \left((Actual\% - Expected\%) \times ln(\frac{Actual\%}{Expected\%}) \right)$$

Breakpoint Value	Bucket	Initial Count	New Count	Initial Percent	New Percent	PSI
-2.330642	1	1	0	0.01	0.001000	0.020723
-1.801596	2	1	3	0.01	0.025000	0.013744
-1.272550	3	4	6	0.04	0.050000	0.002231
-0.743504	4	8	15	0.08	0.125000	0.020083
-0.214458	5	27	18	0.27	0.150000	0.070534
0.314588	6	22	23	0.22	0.191667	0.003906
0.843633	7	16	26	0.16	0.216667	0.017181
1.372679	8	12	14	0.12	0.116667	0.000094
1.901725	9	6	9	0.06	0.075000	0.003347
2.430771	10	3	3	0.03	0.025000	0.000912

Interpretation:

We get a final PSI value of **0.153**, which indicates that there's a chance our population is shifting, and we may want to monitor it going forwards. Of course, this is just one way of calculating PSI by using equal size binning of 10 buckets. If we keep the 10 buckets but change our binning strategy to quantile bins, we end up with a different percent distribution and an overall lower estimate of **0.129**.

X APPLYING STEPS IN LOGISTIC REGRESSION MODELING

The following steps are applied in linear regression modeling in industry:

1. Exclusion criteria and good-bad definition finalization
2. Initial data preparation and univariate analysis
3. Derived/dummy variable creation
4. Fine classing and coarse classing

5. Fitting the logistic model on the training data
6. Evaluating the model on test data

XI RANDOM FOREST

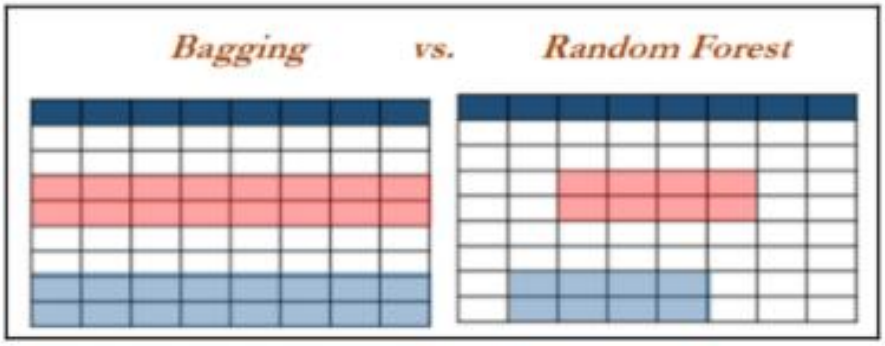
- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.
- It can be used for both Classification and Regression problems in ML.
- It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model
- Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
- Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.
- RF focuses on sampling both observations and variables of training data to develop independent decision trees and take majority voting for classification and averaging for regression problems respectively
- In contrast, bagging samples only observations at random and selects all columns that have the deficiency of representing significant variables at root for all decision trees.
- This way makes trees that are dependent on each other, for which accuracy will be penalized.
- The following are a few rules of thumb when selecting sub-samples from observations using random forest

About 2/3 of observations in training data for each individual tree

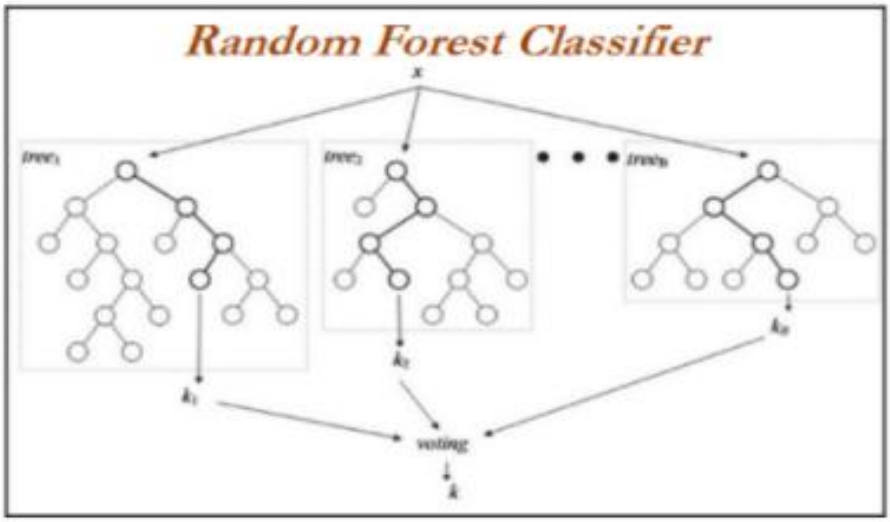
Select columns sqrt(p) -> For classification problem if p is total columns in training data

p/3 -> for regression problem if p is number of columns

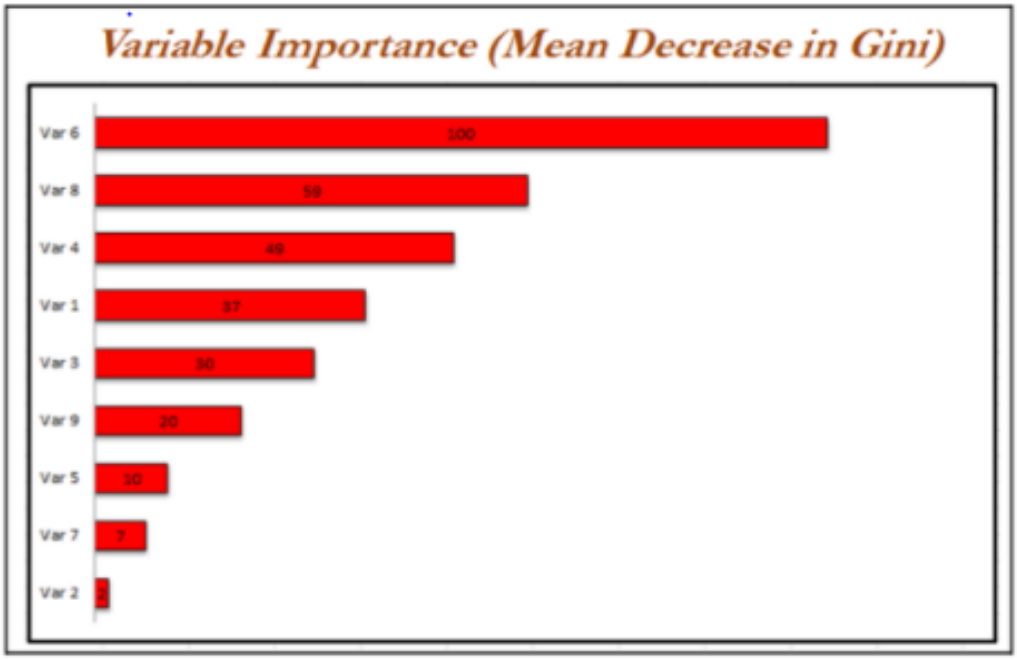
In the following diagram, two samples were shown with blue and pink colors, where, in the bagging scenario, a few observations and all columns are selected. Whereas, in random forest, a few observations and columns are selected to create uncorrelated individual trees.



In the following diagram, a sample idea shows how RF classifier works. Each tree has grown separately, and the depth of each tree varies as per the selected sample, but in the end, voting is performed to determine the final class.



Due to the ensemble of decision trees, RF suffered interpretability and could not determine the significance of each variable; only variable importance could be provided instead. In the following graph, a sample of variable performance has been provided, consisting of a mean decrease in *Gini*:



XII EXAMPLE OF RANDOM FOREST USING GERMAN CREDIT DATA

- The same German credit data is being utilized to illustrate the random forest model
- A very significant difference anyone can observe compared with logistic regression is that effort applied on data preprocessing drastically decreases.
- In RF, we have not removed variables one by one from analysis based on significance and VIF values, as significance tests are not applicable for ML models. However five-fold cross validation has been performed on training data to ensure the model's robustness
- In RF we have not removed the extra dummy variable from the analysis, as the latter automatically takes care of multi-collinearity.
- Random forest requires much less human effort and intervention to train the model.

Random forest applied on German credit data:

```
>>> import pandas as pd
>>> from sklearn.ensemble import RandomForestClassifier

>>> credit_data = pd.read_csv("credit_data.csv")
>>> credit_data['class'] = credit_data['class']-1
```

The creation of dummy variables step is similar to the logistic regression preprocessing step:

```
>>> dummy_stseca =
pd.get_dummies(credit_data['Status_of_existing_checking_account'],
prefix='status_exs_accnt')
>>> dummy_ch = pd.get_dummies(credit_data['Credit_history'],
prefix='cred_hist')
>>> dummy_purpose = pd.get_dummies(credit_data['Purpose'],
prefix='purpose')
>>> dummy_savacc = pd.get_dummies(credit_data['Savings_Account'],
prefix='sav_acc')
>>> dummy_presc = pd.get_dummies(credit_data['Present_Employment_since'],
prefix='pre_emp_snc')
>>> dummy_perssx = pd.get_dummies(credit_data['Personal_status_and_sex'],
prefix='per_stat_sx')
>>> dummy_othdts = pd.get_dummies(credit_data['Other_debtors'],
prefix='oth_debtors')
>>> dummy_property = pd.get_dummies(credit_data['Property'],
prefix='property')
>>> dummy_othinstpln =
pd.get_dummies(credit_data['Other_installment_plans'],
prefix='oth_inst_pln')
>>> dummy_housing = pd.get_dummies(credit_data['Housing'],
prefix='housing')
>>> dummy_job = pd.get_dummies(credit_data['Job'], prefix='job')
>>> dummy_telephn = pd.get_dummies(credit_data['Telephone'],
prefix='telephn')
>>> dummy_forgnwrkr = pd.get_dummies(credit_data['Foreign_worker'],
prefix='forgn_wrkr')
>>> continuous_columns = ['Duration_in_month', 'Credit_amount',
'Installment_rate_in_percentage_of_disposable_income',
'Present_residence_since','Age_in_years','Number_of_existing_credits_at_thi
s_bank',
'Number_of_People_being_liable_to_provide_maintenance_for']
>>> credit_continuous = credit_data[continuous_columns]
```

In the following example, data has been split 70-30. The reason is due to the fact that we would be performing five-fold cross-validation in grid search during training, which produces a similar effect of splitting the data into 50-25-25 of train, validation, and test datasets respectively.

```
>>> x_train,x_test,y_train,y_test = train_test_split( credit_data_new.drop(
['class'],axis=1),credit_data_new['class'],train_size =
0.7,random_state=42)
```

The random forest ML model is applied with assumed hyperparameter values, as follows:

- Number of trees is 1000
- Criterion of slitting is gini
- Maximum depth each decision tree can grow is 100
- Minimum observations required at each not to be eligible for splitting is 3
- Minimum number of observations in tree node should be 2

However, optimum parameter values needs to be tuned using grid search:

```
>>> rf_fit = RandomForestClassifier( n_estimators=1000, criterion="gini",
max_depth=100, min_samples_split=3,min_samples_leaf=2)
>>> rf_fit.fit(x_train,y_train)

>>> print ("\nRandom Forest -Train Confusion Matrix\n\n",
pd.crosstab(y_train, rf_fit.predict( x_train),rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\n Random Forest - Train accuracy",round(accuracy_score(
y_train, rf_fit.predict(x_train)),3))

>>> print ("\nRandom Forest - Test Confusion
Matrix\n\n",pd.crosstab(y_test, rf_fit.predict(x_test),rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\nRandom Forest - Test accuracy",round(accuracy_score(y_test,
rf_fit.predict(x_test)),3))
```

The test accuracy produced from random forest is 0.855

```
Random Forest - Train Confusion Matrix

Predicted     0     1
Actuall
0           501     0
1            18   185

Random Forest - Train accuracy 0.974


Random Forest - Test Confusion Matrix

Predicted     0    1
Actuall
0           210    1
1            43   49

Random Forest - Test accuracy 0.855
```

XIII GRID SEARCH ON RANDOM FOREST

Grid search has been performed by changing various hyperparameters with the following settings. However, readers are encouraged to try other parameters to explore further in this space.

Number of trees is (1000,2000,3000)

Maximum depth is (100,200,300)

Minimum samples per split are (2,3)

Minimum samples in leaf node are (1,2)

Import `Pipeline` as follows:

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.model_selection import train_test_split,GridSearchCV
```

The `Pipeline` function creates the combinations which will be applied one by one sequentially to determine the best possible combination:

```
>>> pipeline = Pipeline([
('clf',RandomForestClassifier(criterion='gini'))])
>>> parameters = {
  ...     'clf__n_estimators':(1000,2000,3000),
  ...     'clf__max_depth':(100,200,300),
  ...     'clf__min_samples_split':(2,3),
  ...     'clf__min_samples_leaf':(1,2) }
```

In the following, grid search utilizes cross-validation of five to ensure robustness in the model, which is the ML way of creating two-point validation of the model:

```
>>> grid_search = GridSearchCV(pipeline,parameters,n_jobs=-1, cv=5,
verbose=1, ... scoring='accuracy')
>>> grid_search.fit(x_train,y_train)

>>> print ('Best Training score: %0.3f' % grid_search.best_score_)
>>> print ('Best parameters set:')
>>> best_parameters = grid_search.best_estimator_.get_params()
>>> for param_name in sorted(parameters.keys()):
  ...     print ('\t%s: %r' % (param_name, best_parameters[param_name]))


>>> predictions = grid_search.predict(x_test)

>>> print ("Testing accuracy:",round(accuracy_score(y_test,
predictions),4))
>>> print ("\nComplete report of Testing
data\n",classification_report(y_test, ... predictions))

>>> print ("\n\nRandom Forest Grid Search- Test Confusion Matrix\n\n",
pd.crosstab(y_test, predictions,rownames = ["Actuall"],colnames =
["Predicted"]))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   47.7s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  4.0min finished
Best Training score: 0.820
Best parameters set:
        clf__max_depth: 300
        clf__min_samples_leaf: 1
        clf__min_samples_split: 3
        clf__n_estimators: 1000
Testing accuracy: 0.8911

Complete report of Testing data
             precision    recall  f1-score   support

          0       0.87      1.00      0.93       211
          1       0.98      0.65      0.78        92

avg / total       0.90      0.89      0.88       303
```

```
Random Forest Grid Search- Test Confusion Matrix

Predicted    0   1
Actuall
0          210   1
1           32  60
```

XIV VARIABLE IMPORTANCE PLOT

- Variable importance plot provides a list of the most significant variables in descending order by a mean decrease in Gini.
- The top variables contribute more to the model than the bottom ones and also have high predictive power in classifying default and non-default customers
- Grid search does not have variable importance functionality in Python scikit- learn, hence we are using the best parameters from grid search and plotting the variable importance graph with simple random forest scikit-learn function.
- Whereas, in R programming, we have that provision, hence R code would be compact here.

```
>>> import matplotlib.pyplot as plt
>>> rf_fit = RandomForestClassifier(n_estimators=1000, criterion="gini",
max_depth=300, min_samples_split=3,min_samples_leaf=1)
>>> rf_fit.fit(x_train,y_train)
>>> importances = rf_fit.feature_importances_
>>> std = np.std([tree.feature_importances_ for tree in
rf_fit.estimators_], axis=0)
>>> indices = np.argsort(importances)[::-1]

>>> colnames = list(x_train.columns)
# Print the feature ranking
>>> print("\nFeature ranking:\n")
>>> for f in range(x_train.shape[1]):
...     print ("Feature", indices[f], ",", colnames[indices[f]],
round(importances [indices[f]],4))

>>> plt.figure()
>>> plt.bar(range(x_train.shape[1]), importances[indices], color="r", yerr=
std[indices], align="center")
>>> plt.xticks(range(x_train.shape[1]), indices)
>>> plt.xlim([-1, x_train.shape[1]])
>>> plt.show()
```

XV COMPARISON OF LOGISTIC REGRESSION WITH RANDOM FOREST

- In the following table, both models explanatory variables have been put in descending order based on the importance of them towards the model contribution.
- In the logistic regression model, it is the p-value (minimum is a better predictor), and for random forest it is the mean decrease in Gini (maximum is a better predictor).
- Many of the variables are very much matching in importance like,
    - status_exs_accnt_A14
    - credit_hist_A34
    - installment_rate_in_percentage_of_disposable_income
    - property_A_24
    - Credit_amount
    - Duration_in_month

Logistic Regression - Summary		
Variable	Co-efficient	p-value
const	-2.1782	0
status_exs_accnt_A14	-1.5561	0
cred_hist_A34	-0.9717	0
per_stat_sx_A93	-0.7401	0
Installment_rate_in_percentage_of_disposable_income	0.3783	0
purpose_A41	-1.3009	0.002
property_A124	0.7233	0.007
status_exs_accnt_A13	-1.1011	0.009
oth_inst_pln_A142	1.1535	0.009
purpose_A43	-0.5359	0.017
Credit_amount	0.0001	0.026
Number_of_existing_credits_at_this_bank	0.3876	0.041
cred_hist_A31	0.9028	0.042
Duration_in_month	0.0185	0.057

Variable Importance - Random Forest		
Feature Number	Variable Name	Mean Decrese in Gini
55	Credit_amount	0.1075
58	Age_in_years	0.0826
54	Duration_in_month	0.0729
3	status_exs_accnt_A14	0.0446
56	Installment_rate_in_percentage_of_disposable_income	0.0355
57	Present_residence_since	0.0315
0	status_exs_accnt_A11	0.0307
8	cred_hist_A34	0.0231
1	status_exs_accnt_A12	0.0191
59	Number_of_existing_credits_at_this_bank	0.0191
19	sav_acc_A61	0.0172
9	purpose_A40	0.017
39	property_A124	0.0157
30	per_stat_sx_A92	0.0155
31	per_stat_sx_A93	0.0155
38	property_A123	0.0155
42	oth_inst_pln_A143	0.0152
44	housing_A152	0.0149
36	property_A121	0.0149
25	pre_emp_snc_A72	0.0147
6	cred_hist_A32	0.0147
48	job_A173	0.014
5	cred_hist_A31	0.0139
26	pre_emp_snc_A73	0.0137
13	purpose_A43	0.0131
49	job_A174	0.0131
51	telephn_A192	0.0128
50	telephn_A191	0.0127
12	purpose_A42	0.0127
27	pre_emp_snc_A74	0.0127
28	pre_emp_snc_A75	0.0126
4	cred_hist_A30	0.0123
23	sav_acc_A65	0.0122
40	oth_inst_pln_A141	0.0115

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CAMPUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2021-22(EVEN SEMESTER)

18CSE479T - Statistical Machine Learning

Unit - III

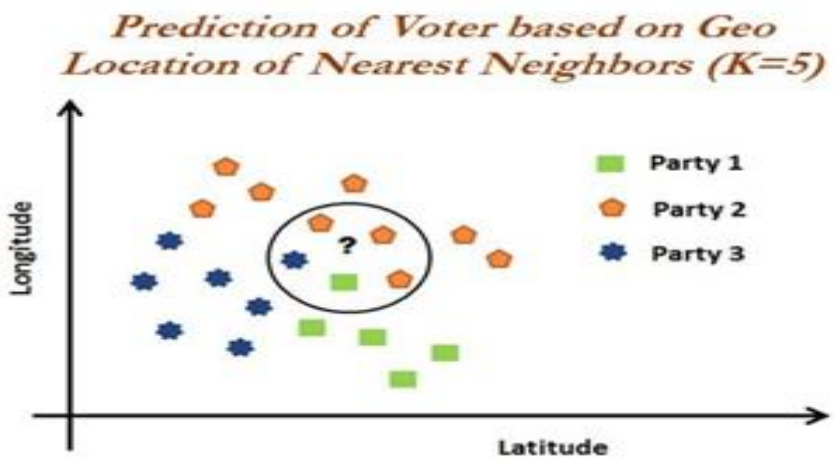
K-nearest neighbors-KNN voter example -Curse of dimensionality-Curse of dimensionality with 1D, 2D, and 3D example-Curse of dimensionality with 3D example-KNN classifier with breast cancer Wisconsin data example.Naive Bayes-Probability fundamentals - Joint probability-Understanding Bayes theorem with conditional probability-Naive Bayes classification -Laplace estimator-Naive Bayes SMS spam classification example

I K-nearest neighbors

- Definition - K-nearest neighbors is a **non-parametric machine learning model** in which the model memorizes the training observation for classifying the unseen test data.
- It is also called as **Instance-based learning**
- This model is often termed as **lazy learning**, as it does not learn anything during the training phase like regression, random forest, and so on.
- It starts working only during the testing/evaluation phase to compare the given test observations with nearest training observations

II KNN voter example

- Objective is to predict the party for which voter will vote based on their neighborhood, precisely geolocation (latitude and longitude).
- Assumption - Identify the potential voter to which political party they would be voting based on majority voters did voted for that particular party.
- Tuning the k-value (number to consider, among which majority should be counted) is the million- dollar question (as same as any machine learning algorithm)



- In the preceding diagram, we can see that the voter of the study will vote for **Party 2**.
- As within the vicinity, one neighbor has voted for **Party 1** and the other voter voted for P**arty 3**.
- But three voters voted for **Party 2**.
- In fact, by this way KNN solves any given classification problem.
- Regression problems are solved by taking mean of its neighbors within the given circle or vicinity or k-value

III Curse of dimensionality

Curse of dimensionality

- The curse of dimensionality refers to the phenomena that occur when classifying, organizing, and analyzing high dimensional data that does not occur in low dimensional spaces, specifically the issue of data sparsity and "closeness" of data.
- Points in high-dimensional spaces tend to be dispersing from each other more compared with the points in low-dimensional space.
- uniform random values between zero and one generated for 1D, 2D, and 3D space to validate this hypothesis

Example

- In the following code, mean distance between 1,000 observations have been calculated with the change in dimensions.
- It is apparent that with the increase in dimensions, distance between points increases logarithmically, which gives us the hint that we need to have exponential increase in data points with increase in dimensions in order to make machine learning algorithms work correctly

```
>>> import numpy as np
>>> import pandas as pd
>>> import random,math
```

- The following code generates random numbers between zero and one from uniformdistribution with the given dimension, which is equivalent of length of array or list:

```
>>> def random_point_gen(dimension):
...         return [random.random() for _ in range(dimension)]
```

- The following function calculates root mean sum of squares of Euclidean distances (2-norm) between points by taking the difference between points and sum the squares and finally takes square root of total distance:

```
>>> def distance(v,w):
...         vec_sub = [v_i-w_i for v_i,w_i in zip(v,w)]
...         sum_of_sqrs = sum(v_i*v_i for v_i in vec_sub)
...         return math.sqrt(sum_of_sqrs)
```

- Both dimension and number of pairs are utilized for calculating the distances with the following code:

```
>>> def random_distances_comparison(dimension,number_pairs):
... return[distance(random_point_gen(dimension),random_point_gen(dimension))
for _ in range(number_pairs)]
>>> def mean(x):
...         return sum(x) / len(x)
```

- changing dimensions from 1 to 201 with the increase of 5 dimensions to check the increase in distance:

```
>>> dimensions = range(1, 201, 5)
```

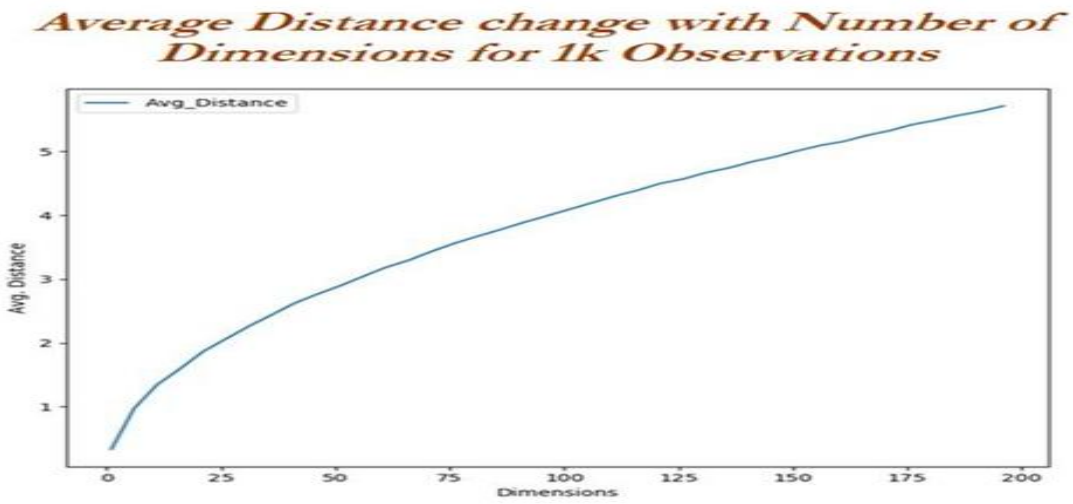
- Both minimum and average distances have been calculated to check,

```
>>> avg_distances = []
>>> min_distances = []
>>> dummyarray = np.empty((20,4))
```

```
>>> dist_vals = pd.DataFrame(dummyarray)
>>> dist_vals.columns = ["Dimension","Min_Distance","Avg_Distance","Min/Avg_Distance”]
>>> random.seed(34)
>>> i = 0
>>> for dims in dimensions:
...     distances = random_distances_comparison(dims, 1000)
...     avg_distances.append(mean(distances))
...     min_distances.append(min(distances))
...     dist_vals.loc[i,"Dimension"] = dims
...     dist_vals.loc[i,"Min_Distance"] = min(distances)
...     dist_vals.loc[i,"Avg_Distance"] = mean(distances)
...     dist_vals.loc[i,"Min/Avg_Distance"] = min(distances)/mean(distances)
...     print(dims, min(distances), mean(distances), min(distances)*1.0 / mean(distances))
...     i = i+1
```

- **Plotting Average distances for Various Dimensions**

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plt.xlabel('Dimensions')
>>> plt.ylabel('Avg. Distance')
>>> plt.plot(dist_vals["Dimension"],dist_vals["Avg_Distance"])
>>> plt.legend(loc='best')
>>> plt.show()
```



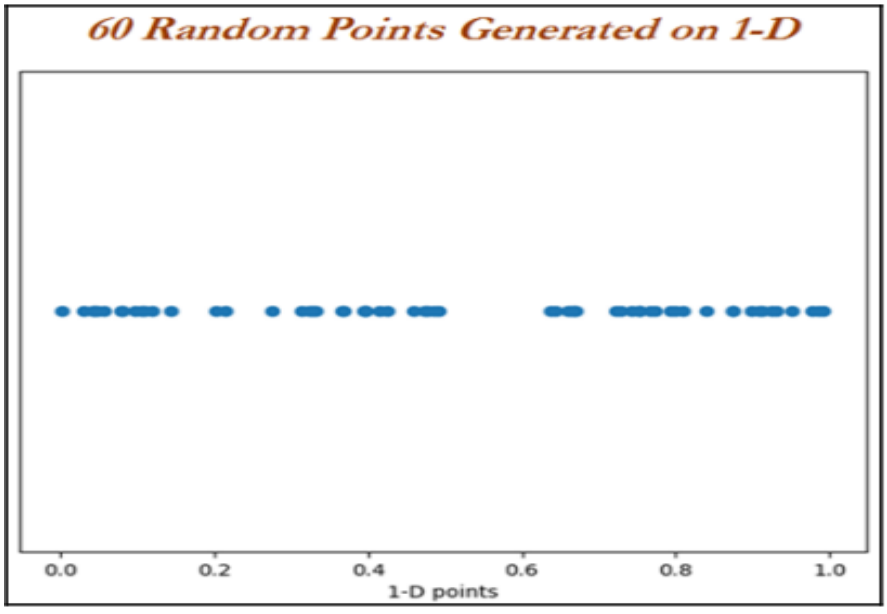
- From the graph, it is proved that with the increase in dimensions, mean distance increases logarithmically.
- Hence the higher the dimensions, the more data is needed to overcome the curse of dimensionality!

IV Curse of dimensionality with 1D

- distance of 60 random points are expanding with the increase in dimensionality.

 # 1-Dimension Plot

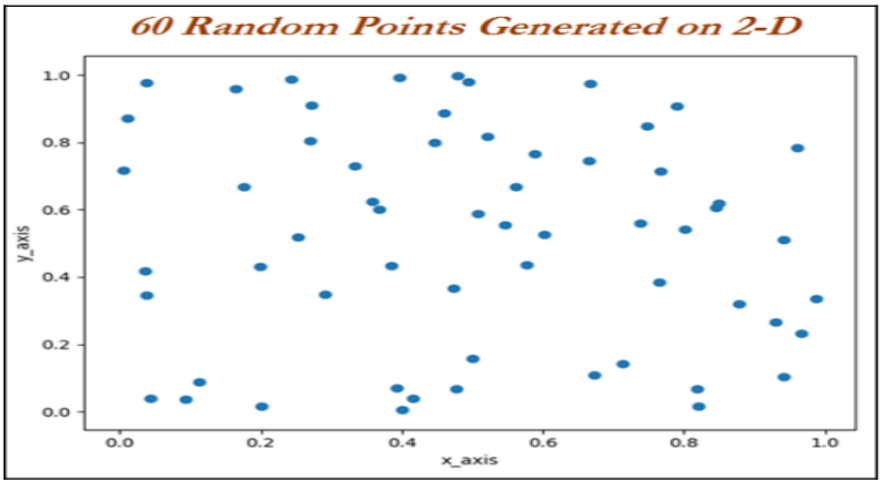
```
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> one_d_data = np.random.rand(60,1)
>>> one_d_data_df = pd.DataFrame(one_d_data)
>>> one_d_data_df.columns = ["1D_Data"]
>>> one_d_data_df["height"] = 1
>>> plt.figure()
>>> plt.scatter(one_d_data_df['1D_Data'],one_d_data_df["height"])
>>> plt.yticks([])
>>> plt.xlabel("1-D points")
>>> plt.show()
```



V Curse of dimensionality with 2D

- 60 random numbers with x and y co-ordinate space

```
# 2- Dimensions Plot
>>> two_d_data = np.random.rand(60,2)
>>> two_d_data_df = pd.DataFrame(two_d_data)
>>> two_d_data_df.columns = ["x_axis","y_axis"]
 >>> plt.figure()
>>> plt.scatter(two_d_data_df['x_axis'],two_d_data_df["y_axis"])
>>> plt.xlabel("x_axis");plt.ylabel("y_axis")
>>> plt.show()
```

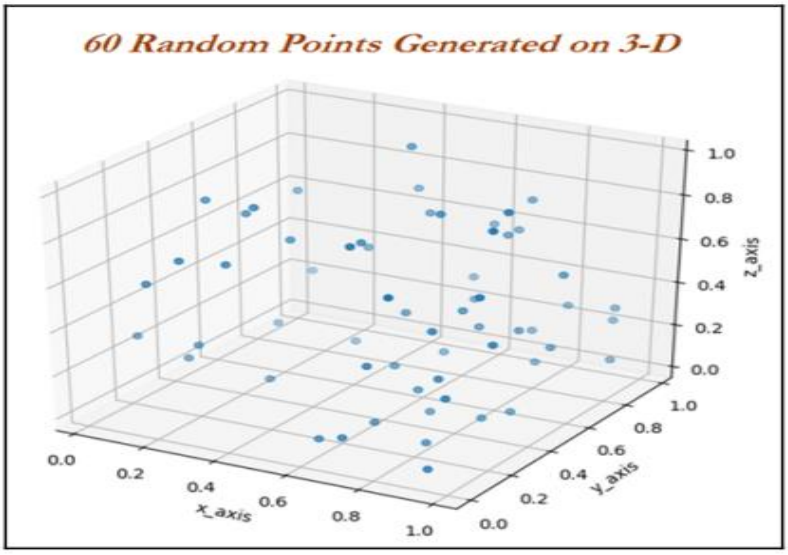


VI Curse of dimensionality with 3D example

- 60 data points are drawn for 3D space
- increase in spaces, which is very apparent.
- This has proven to us visually that with the increase in dimensions, it creates lot of space, which makes a classifier weak to detect the signal

```
# 3- Dimensions Plot
>>> three_d_data = np.random.rand(60,3)
>>> three_d_data_df = pd.DataFrame(three_d_data)
>>> three_d_data_df.columns = ["x_axis","y_axis","z_axis"]
 >>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
```

```
>>> ax = fig.add_subplot(111, projection='3d')
>>>
ax.scatter(three_d_data_df['x_axis'],three_d_data_df["y_axis"],three_d_data
_df ["z_axis"])
>>> plt.show()
```



VII KNN classifier with breast cancer Wisconsin data example

Breast cancer data has been utilized from the UCI machine learning repository http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

To find whether the cancer is malignant or benign based on various collected features such as clump thickness and so on using the KNN classifier

 # KNN Classifier - Breast Cancer

```
>>> import numpy as np
>>> import pandas as pd
>>> from sklearn.metrics import accuracy_score,classification_report
>>> breast_cancer = pd.read_csv("Breast_Cancer_Wisconsin.csv")
```

- The Class value has class 2 and 4. Value 2 and 4 represent benign and malignant class, respectively.
- Whereas all the other variables do vary between value 1 and 10, which are very much categorical in nature

ID_Number	Clump_Thickness	Unif_Cell_Size	Unif_Cell_Shape	Marg_Adhesion	Single_Epith_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2
1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1	1	2

- the Bare_Nuclei variable has some missing values, here we are replacing them with the most frequent value (category value 1) in the following code:

  >>>breast_cancer['Bare_Nuclei']= breast_cancer['Bare_Nuclei'].replace('?', np.NAN)

  >>>breast_cancer['Bare_Nuclei'] = breast_cancer['Bare_Nuclei'].fillna(breast_cancer['Bare_Nuclei'].value_counts().index[0])

- Use the following code **to convert the classes to a 0 and 1** indicator for using in the classifier:

  >>> breast_cancer['Cancer_Ind'] = 0

  >>>breast_cancer.loc[breast_cancer['Class']==4,'Cancer_Ind '] = 1

- we are dropping non-value added variables from analysis:

  >>> x_vars = breast_cancer.drop(['ID_Number','Class','Cancer_Ind'],axis=1)

  >>> y_var = breast_cancer['Cancer_Ind']

  >>> from sklearn.preprocessing import StandardScaler

  >>> x_vars_stdscle = StandardScaler().fit_transform(x_vars.values)

  >>> from sklearn.model_selection import train_test_split

- As KNN is very sensitive to distances, here we are standardizing all the columns before applying algorithms:

  >>> x_vars_stdscle_df = pd.DataFrame(x_vars_stdscle, index=x_vars.index, columns=x_vars.columns)

  >>> x_train,x_test,y_train,y_test = train_test_split(x_vars_stdscle_df,y_var, train_size = 0.7,random_state=42)

- KNN classifier is being applied with neighbor value of 3 and p value indicates it is 2-norm, also known as Euclidean distance for computing classes:

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn_fit = KNeighborsClassifier(n_neighbors=3,p=2,metric='minkowski')
```

```
>>> knn_fit.fit(x_train,y_train)
>>> print ("\nK-Nearest Neighbors - Train Confusion Matrix\n\n",pd.crosstab(y_train, knn_fit.predict(x_train),rownames = ["Actuall"],colnames = ["Predicted"]) )
>>> print ("\nK-Nearest Neighbors - Train accuracy:",round(accuracy_score(y_train, knn_fit.predict(x_train)),3))
>>> print ("\nK-Nearest Neighbors - Train Classification Report\n", classification_report( y_train,knn_fit.predict(x_train)))
>>> print ("\n\nK-Nearest Neighbors - Test Confusion Matrix\n\n",pd.crosstab(y_test, knn_fit.predict(x_test),rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nK-Nearest Neighbors - Test accuracy:",round(accuracy_score( y_test,knn_fit.predict(x_test)),3))
>>> print ("\nK-Nearest Neighbors - Test Classification Report\n", classification_report(y_test,knn_fit.predict(x_test)))
```

```
K-Nearest Neighbors - Train Confusion Matrix

 Predicted    0    1
Actuall
0          309    6
1            4  170

K-Nearest Neighbors - Train accuracy: 0.98

K-Nearest Neighbors - Train Classification Report
              precision    recall  f1-score   support

          0       0.99      0.98      0.98       315
          1       0.97      0.98      0.97       174

avg / total       0.98      0.98      0.98       489


K-Nearest Neighbors - Test Confusion Matrix

 Predicted    0   1
Actuall
0          141   2
1            3  64

K-Nearest Neighbors - Test accuracy: 0.976

K-Nearest Neighbors - Test Classification Report
              precision    recall  f1-score   support

          0       0.98      0.99      0.98       143
          1       0.97      0.96      0.96        67

avg / total       0.98      0.98      0.98       210
```

- From the results, it is appearing that KNN is working very well in classifying malignant and benign classes well
- Obtains test accuracy of 97.6 percent with 96 percent of recall on malignant class.

- The only deficiency of KNN classifier would be, it is computationally
- intensive during test phase, as each test observation will be compared with all the available observations in train data, which practically KNN does not learn a thing from training data. Hence, we are also calling it a lazy classifier!

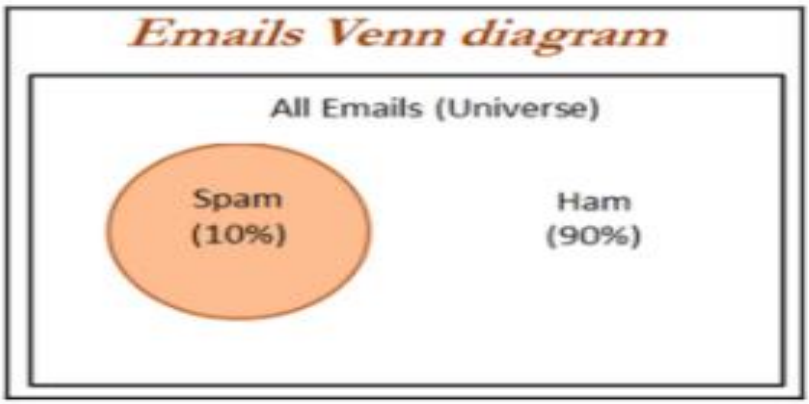
VIII Naive Bayes - Introduction

- The Naive Bayes is a **classification algorithm** that is suitable for **binary and multiclass classification.**
- Naïve Bayes performs well in cases of **categorical input variables** compared to numerical variables.
- It is useful for making predictions and forecasting data based on historical results.
- The distinction between Bayes theorem and Naive Bayes is that **Naive Bayes assumes conditional independence** where **Bayes theorem does not.**
- This means **the relationship between all input features are independent**. Maybe not a great assumption, but this is is why the algorithm is called "naive
- **Thomas** developed the foundational mathematical principles for **determining the probability of unknown events from the known events.**
- For example, if all apples are red in color and average diameter would be about 4 inches then, if at random one fruit is selected from the basket with red color and diameter of 3.7 inch, what is the probability that the particular fruit would be an apple?
- **Naive term does assume independence of particular features in a class with respect to others**.
- In this case, there would be no dependency between color and diameter.
- This independence assumption makes the Naive Bayes classifier most effective in terms of computational ease for particular tasks
- **Bayesian classifiers are best applied to problems in which information from a very high number of attributes should be considered simultaneously to estimate the probability of final outcome.**

IX Probability fundamentals

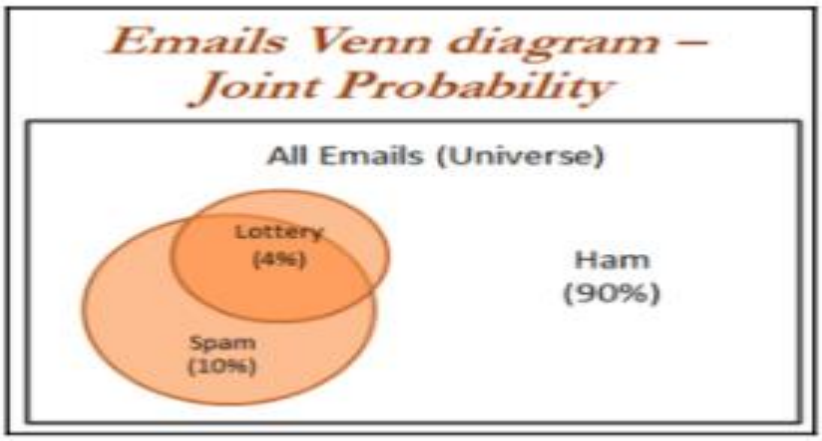
- Probability of an event can be estimated from observed data by dividing the number of trails in which an event occurred with total number of trails.

- For instance, if a bag contains red and blue balls and randomly picked 10 balls one by one with replacement and out of 10, 3 red balls appeared in trails we can say that
- probability of red is 0.3, pred = 3/10 = 0.3.
- Total probability of all possible outcomes must be 100 percent.
- **If a trail has two outcomes such as email classification either it is spam or ham and both cannot occur simultaneously, these events are considered as mutually exclusive with each other.**
- In addition, if those outcomes cover all possible events, it would be called as **exhaustive events**.
- For example, in email classification if *P (spam) = 0.1*, we will be able to calculate *P (ham) = 1- 0.1 = 0.9*, these two events are mutually exclusive

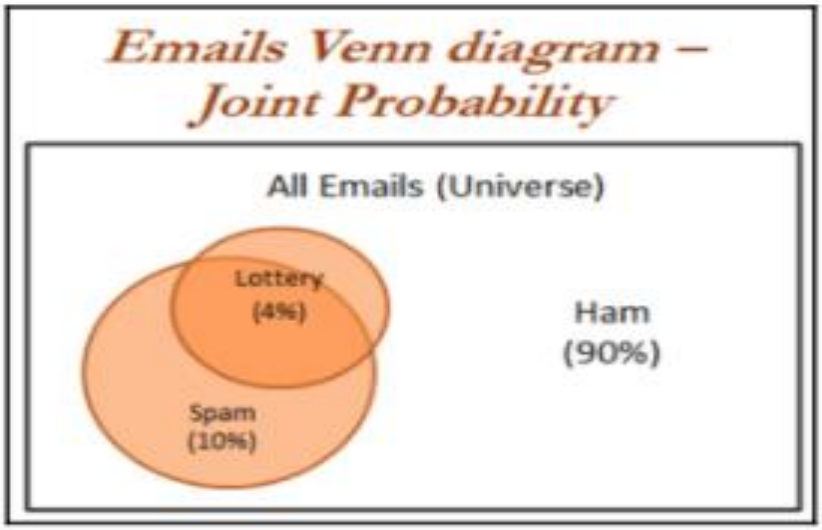


X Joint probability

- Though mutually exclusive cases are simple to work upon, most of the actual problems do fall under the category of non-mutually exclusive events.
- **By using the joint appearance, we can predict the event outcome.**
- For example, if emails messages present the word like lottery, which is very highly likely of being spam rather than ham.
- The spam messages contain the word lottery and not every email with the word lottery is spam.



- the joint probability of both *p(spam)* and *p(lottery)* occurring, which can be written as *p(spam ∩ lottery)*.
- In case if both the events are totally unrelated, they are called **independent events** and their respective value is *p(spam ∩ lottery) = p(spam) * p(lottery) = 0.1 * 0.04 = 0.004*, which is 0.4 percent of all messages are spam containing the word Lottery.
- In general, for independent events *P(A∩ B) = P(A) * P(B)*.

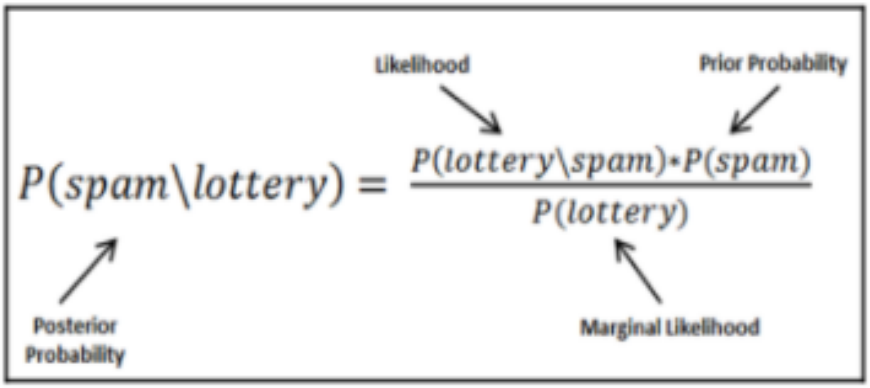


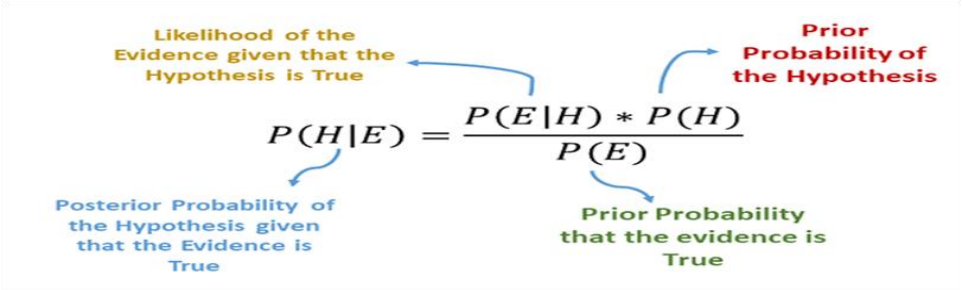
XI Understanding Bayes theorem with conditional probability

- **Conditional probability provides a way of calculating relationships between dependent events using Bayes theorem.**
- For example, A and B are two events and we would like to calculate **P(A\B) can be read as the probability of event occurring A given the fact that event B already occurred**, in fact this is known as conditional probability,
- The equation can be written as follows:

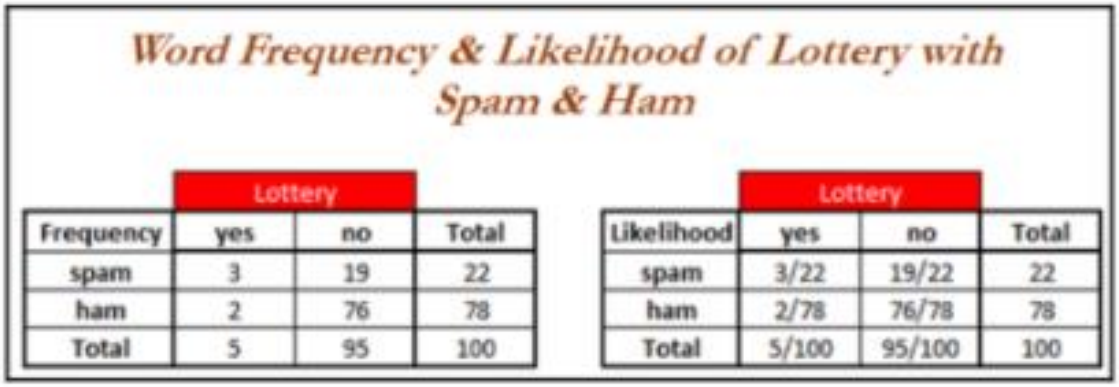
$$P(A\backslash B) = \frac{P(B\backslash A) * P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)}$$

- Email classification example - Objective is to predict whether email is spam given the word lottery and some other clues.
- In this case we already knew the overall probability of spam, which is 10 percent also known as prior probability.
- Now suppose you have obtained an additional piece of information that probability of word lottery in all messages, which is 4 percent, also known as marginal likelihood.
- Now, we know the probability that lottery was used in previous spam messages and is called the likelihood.





- By applying the Bayes theorem to the evidence, we can calculate the posterior probability that calculates the probability that the message is how likely being a spam; given the fact that lottery was appearing in message.
- On average if the probability is greater than 50 percent it indicates that the message is spam rather than ham.



Word Frequency & Likelihood of Lottery with Spam & Ham

	Lottery		
Frequency	yes	no	Total
spam	3	19	22
ham	2	76	78
Total	5	95	100

	Lottery		
Likelihood	yes	no	Total
spam	3/22	19/22	22
ham	2/78	76/78	78
Total	5/100	95/100	100

- sample frequency table that records the number of times *Lottery* appeared in spam and ham messages and its respective likelihood has been shown.
- **Likelihood table reveals that *P(Lottery\Spam)= 3/22 = 0.13,*** indicating that probability is 13 percent that a spam message contains the term *Lottery*.
- Subsequently we can calculate the ***P(Spam ∩ Lottery) = P(Lottery\Spam) * P(Spam) = (3/22) * (22/100) = 0.03*.**
- In order to **calculate the posterior probability, we divide *P(Spam ∩ Lottery)* with *P(Lottery)*, which means *(3/22)*(22/100) / (4/100) = 0.75***
- Therefore, the probability is 75 percent that a message is spam, given that message contains the word *Lottery*.

XII Naive Bayes classification

- Let us construct the likelihood table for the appearance of the three words (W1, W2, and W3), as shown in the following table for 100 emails:

	Lottery (W1)		Million (W2)		Unsubscribe (W3)		
Likelihood	yes	no	yes	no	yes	no	Total
spam	3/22	19/22	11/22	11/22	13/22	9/22	22
ham	2/78	76/78	15/78	63/78	21/78	57/78	78
Total	5/100	95/100	26/100	74/100	34/100	66/100	100

- When a new message is received, the posterior probability will be calculated to determine that email message is spam or ham. Let us assume that we have an email with terms *Lottery* and *Unsubscribe*, but it does not have word *Million* in it, with this details, what is the probability of spam?
- By using Bayes theorem, we can define the problem as *Lottery = Yes*, *Million = No* and *Unsubscribe = Yes*:
- Solving the preceding equations will have high computational complexity due to the dependency of words with each other. As more number of words are added, this will even explode and also huge memory will be needed for processing all possible intersecting events. This finally leads to intuitive turnaround with independence of words (**cross- conditional independence**)

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \cap \neg W_2 \cap W_3 \mid Spam) * P(Spam)}{P(W_1 \cap \neg W_2 \cap W_3)}$$

- When both events are independent we can write $P(A \cap B) = P(A) * P(B)$. In fact, this equivalence is much easier to compute with less memory requirement:

$$P(Spam \mid W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \mid Spam) * P(\neg W_2 \mid Spam) * P(W_3 \mid Spam) * P(Spam)}{P(W_1) * P(\neg W_2) * P(W_3)}$$

- In a similar way, we will calculate the probability for ham messages as well, as follows:

$$P(Ham \mid W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \mid Ham) * P(\neg W_2 \mid Ham) * P(W_3 \mid Ham) * P(Ham)}{P(W_1) * P(\neg W_2) * P(W_3)}$$

- By substituting the preceding likelihood table in the equations, due to the ratio of spam/ham we can just simply ignore the denominator terms in both the equations. Overall likelihood of spam is:

	Lottery (W1)		Million (W2)		Unsubscribe (W3)		
Likelihood	yes	no	yes	no	yes	no	Total
spam	3/22	19/22	11/22	11/22	13/22	9/22	22
ham	2/78	76/78	15/78	63/78	21/78	57/78	78
Total	5/100	95/100	26/100	74/100	34/100	66/100	100

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{3}{22}\right) * \left(\frac{11}{22}\right) * \left(\frac{13}{22}\right) * \left(\frac{22}{100}\right) = 0.008864$$

$$P(Ham|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{2}{78}\right) * \left(\frac{63}{78}\right) * \left(\frac{21}{78}\right) * \left(\frac{78}{100}\right) = 0.004349$$

- After calculating ratio, *0.008864/0.004349 = 2.03*, which means that this message is two times more likely to be spam than ham. But we can calculate the probabilities as follows

$$P(Spam) = 0.008864/(0.008864+0.004349) = 0.67$$

$$P(Ham) = 0.004349/(0.008864+0.004349) = 0.33$$

XIII Laplace estimator

- In the previous calculation, all the values are nonzeros, which makes calculations well. Whereas in practice some words never appear in past for specific category and suddenly appear at later stages, which makes entire calculations as zeros
- For example, in the previous equation *W3* did have a *0* value instead of *13*, and it will convert entire equations to *0* altogether

$$P(Spam\ |W_1 \cap \neg W_2 \cap W_3) = \left(\frac{3}{22}\right) * \left(\frac{11}{22}\right) * \left(\frac{0}{22}\right) * \left(\frac{22}{100}\right) = 0$$

- In order to avoid this situation, Laplace estimator essentially adds a small number to each of the counts in the frequency table, which ensures that each feature has a nonzero probability of occurring with each class. Usually Laplace estimator is set to *1*, which ensures that each class-feature combination is found in the data at least once

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{4}{25}\right) * \left(\frac{12}{25}\right) * \left(\frac{1}{25}\right) * \left(\frac{22}{100}\right) = 0$$

XIV Naive Bayes SMS spam classification example

- Data set : SMS spam collection
- NLP techniques to preprocess prior to build the Naive Bayes model

```
>>> import csv
>>> smsdata = open('SMSSpamCollection.txt','r')
>>> csv_reader = csv.reader(smsdata,delimiter='\t')
```

```
>>> smsdata_data = []
>>> smsdata_labels = []

>>> for line in csv_reader:
...     smsdata_labels.append(line[0])
...     smsdata_data.append(line[1])

>>> smsdata.close()
```

The following code prints the top 5 lines:

```
>>> for i in range(5):
...     print (smsdata_data[i],smsdata_labels[i])
```

```
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore
wat... ham
Ok lar... Joking wif u oni... ham
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry
question(std txt rate)T&C's apply 08452810075over18's spam
U dun say so early hor... U c already then say... ham
Nah I don't think he goes to usf, he lives around here though ham
```

After getting preceding output run following code:

```
>>> from collections import Counter
>>> c = Counter( smsdata_labels )
>>> print(c)
```

```
Counter({'ham': 4825, 'spam': 747})
```

Out of 5,572 observations, 4,825 are ham messages, which are about 86.5 percent and 747 spam messages are about remaining 13.4 percent.

Preprocessing stages are

- **Removal of punctuations**
- **Word tokenization**: Words are chunked from sentences based on white space for further processing
- **Converting words into lower case**: Converting to all lower case provides removal of duplicates, such as *Run* and *run*, where the first one comes at start of the sentence and the later one comes in the middle of the sentence, and so on
- **Stop word removal**: Stop words are the words that repeat so many times in literature and yet are not much differentiator in explanatory power of sentences. For example: *I*, *me*, *you*, *this*, *that*, and so on, which needs to be removed before further processing
- **of length at least three**: Here we have removed words with length less than three

- Stemming of words: Stemming process stems the words to its respective root words. Example of stemming is bringing down running to run or runs to run. By doing stemming we reduce duplicates and improve the accuracy of the model
- **Part-of-speech (POS) tagging**: This applies the speech tags to words, such as noun, verb, adjective, and so on. For example, POS tagging for *running* is verb, whereas for *run* is noun. In some situation *running* is noun and lemmatization will not bring down the word to root word *run*, instead it just keeps the *running* as it is.
- **Lemmatization of words**: Lemmatization is another different process to reduce the dimensionality. In lemmatization process, it brings down the word to root word rather than just truncating the words. For example, bring *ate* to its root word as *eat* when we pass the *ate* word into lemmatizer with the POS tag as verb.

The `nltk` package has been utilized for all the preprocessing steps, as it consists of all the necessary NLP functionality in one single roof:

```
>>> import nltk
>>> from nltk.corpus import stopwords
>>> from nltk.stem import WordNetLemmatizer
>>> import string
>>> import pandas as pd
>>> from nltk import pos_tag
>>> from nltk.stem import PorterStemmer
```

Function has been written (preprocessing) consists of all the steps for convenience. However, we will be explaining all the steps in each section:

```
>>> def preprocessing(text):
```

The following line of the code splits the word and checks each character if it is in standard punctuations, if so it will be replaced with blank and or else it just does not replace with blanks:

```
...     text2 = " ".join("".join([" " if ch in string.punctuation else ch
for ch in text]).split())
```

The following code tokenizes the sentences into words based on white spaces and put them together as a list for applying further steps:

```
...     tokens = [word for sent in nltk.sent_tokenize(text2) for word in
            nltk.word_tokenize(sent)]
```

Converting all the cases (upper, lower, and proper) into lowercase reduces duplicates in corpus:

```
...     tokens = [word.lower() for word in tokens]
```

As mentioned earlier, stop words are the words that do not carry much weight in understanding the sentence; they are used for connecting words, and so on. We have removed them with the following line of code:

```
...     stopwds = stopwords.words('english')
...     tokens = [token for token in tokens if token not in stopwds]
```

Keeping only the words with length greater than `3` in the following code for removing small words, which hardly consists of much of a meaning to carry:

```
...     tokens = [word for word in tokens if len(word)>=3]
```

Stemming is applied on the words using `PorterStemmer` function, which stems the extra suffixes from the words:

```
...     stemmer = PorterStemmer()
...     tokens = [stemmer.stem(word) for word in tokens]
```

POS tagging is a prerequisite for lemmatization, based on whether the word is noun or verb, and so on, it will reduce it to the root word:

```
...     tagged_corpus = pos_tag(tokens)
```

The `pos_tag` function returns the part of speed in four formats for noun and six formats for verb. `NN` (noun, common, singular), `NNP` (noun, proper, singular), `NNPS` (noun, proper, plural), `NNS` (noun, common, plural), `VB` (verb, base form), `VBD` (verb, past tense), `VBG` (verb, present participle), `VBN` (verb, past participle), `VBP` (verb, present tense, not third person singular), `VBZ` (verb, present tense, third person singular):

```
...     Noun_tags = ['NN','NNP','NNPS','NNS']
...     Verb_tags = ['VB','VBD','VBG','VBN','VBP','VBZ']
...     lemmatizer = WordNetLemmatizer()


...     def prat_lemmatize(token,tag):
...         if tag in Noun_tags:
...             return lemmatizer.lemmatize(token,'n')
...         elif tag in Verb_tags:
...             return lemmatizer.lemmatize(token,'v')
...         else:
...             return lemmatizer.lemmatize(token,'n')
```

After performing tokenization and applied all the various operations, we need to join it back to form stings and the following function performs the same:

```
...     pre_proc_text =  " ".join([prat_lemmatize(token,tag) for token,tag
in tagged_corpus])
...     return pre_proc_text
```

The following step applies the preprocessing function to the data and generates new corpus:

```
>>> smsdata_data_2 = []
>>> for i in smsdata_data:
...     smsdata_data_2.append(preprocessing(i))
```

```
>>> import numpy as np
>>> trainset_size = int(round(len(smsdata_data_2)*0.70))
>>> print ('The training set size for this classifier is ' +
str(trainset_size) + '\n')
>>> x_train = np.array([''.join(rec) for rec in
smsdata_data_2[0:trainset_size]])
>>> y_train = np.array([rec for rec in smsdata_labels[0:trainset_size]])
>>> x_test = np.array([''.join(rec) for rec in
smsdata_data_2[trainset_size+1:len( smsdata_data_2)]])
>>> y_test = np.array([rec for rec in smsdata_labels[trainset_size+1:len(
smsdata_labels)]])
```

The following code converts the words into a vectorizer format and applies **term frequency-inverse document frequency** (**TF-IDF**) weights, which is a way to increase weights to words with high frequency and at the same time penalize the general terms such as *the*, *him*, *at*, and so on. In the following code, we have restricted to most frequent 4,000 words in the vocabulary, none the less we can tune this parameter as well for checking where the better accuracies are obtained:

```
# building TFIDF vectorizer
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer(min_df=2, ngram_range=(1, 2),
stop_words='english',
    max_features= 4000,strip_accents='unicode',  norm='l2')
```

The TF-IDF transformation has been shown as follows on both train and test data. The `todense` function is used to create the data to visualize the content:

```
>>> x_train_2 = vectorizer.fit_transform(x_train).todense()
>>> x_test_2 = vectorizer.transform(x_test).todense()
```

```
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB().fit(x_train_2, y_train)

>>> ytrain_nb_predicted = clf.predict(x_train_2)
>>> ytest_nb_predicted = clf.predict(x_test_2)

>>> from sklearn.metrics import classification_report,accuracy_score
```

```
>>> print ("\nNaive Bayes - Train Confusion
Matrix\n\n",pd.crosstab(y_train, ytrain_nb_predicted,rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\nNaive Bayes- Train accuracy",round(accuracy_score(y_train,
ytrain_nb_predicted),3))
>>> print ("\nNaive Bayes  - Train Classification
Report\n",classification_report(y_train, ytrain_nb_predicted))

>>> print ("\nNaive Bayes - Test Confusion Matrix\n\n",pd.crosstab(y_test,
ytest_nb_predicted,rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nNaive Bayes- Test accuracy",round(accuracy_score(y_test,
ytest_nb_predicted),3))
>>> print ("\nNaive Bayes  - Test Classification
Report\n",classification_report( y_test, ytest_nb_predicted))
```

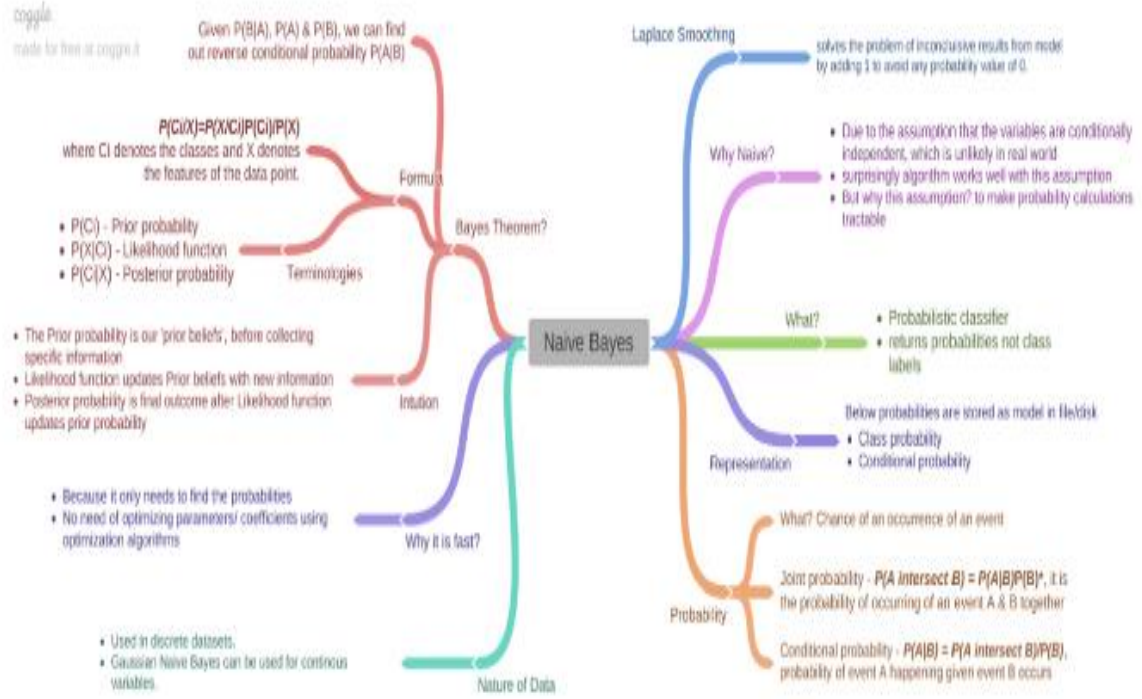
```
Naive Bayes - Train Confusion Matrix

 Predicted    ham  spam
Actuall
ham         3381     0
spam          78   441

Naive Bayes- Train accuracy 0.98

Naive Bayes   - Train Classification Report
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      3381
        spam       1.00      0.85      0.92       519

avg / total       0.98      0.98      0.98      3900


Naive Bayes - Test Confusion Matrix

 Predicted    ham  spam
Actuall
ham         1440     3
spam          54   174

Naive Bayes- Test accuracy 0.966

Naive Bayes   - Test Classification Report
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98      1443
        spam       0.98      0.76      0.86       228

avg / total       0.97      0.97      0.96      1671
```

However, if we would like to check what are the top 10 features based on their coefficients from Naive Bayes, the following code will be handy for this:

```
# printing top features
>>> feature_names = vectorizer.get_feature_names()
>>> coefs = clf.coef_
>>> intercept = clf.intercept_
>>> coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))

>>> print ("\n\nTop 10 features - both first & last\n")
>>> n=10
>>> top_n_coefs = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
>>> for (coef_1, fn_1), (coef_2, fn_2) in top_n_coefs:
...     print('\t%.4f\t%-15s\t\t%.4f\t%-15s' % (coef_1, fn_1, coef_2,
fn_2))
```

```
Top 10 features - both first & last

        -8.7128 1hr                     -5.5773 free
        -8.7128 1st love                -5.7141 txt
        -8.7128 2go                     -5.8715 text
        -8.7128 2morrow                 -6.0127 claim
        -8.7128 2mrw                    -6.0740 stop
        -8.7128 2nd inning              -6.0809 mobil
        -8.7128 2nd sm                  -6.1059 repli
        -8.7128 30ish                   -6.1593 prize
        -8.7128 3rd                     -6.1994 servic
        -8.7128 3rd natur               -6.2101 tone
```

Naive Bayes - Mind Map



Unit - IV
Support Vector Machines and Artificial Neural Networks

Dr.S.Veena,Associate Professor/CSE

Unit IV

Support Vector Machines

- Support vector machines working principles
- Maximum margin classifier
- Support vector classifier
- Support vector machines
- Kernel functions

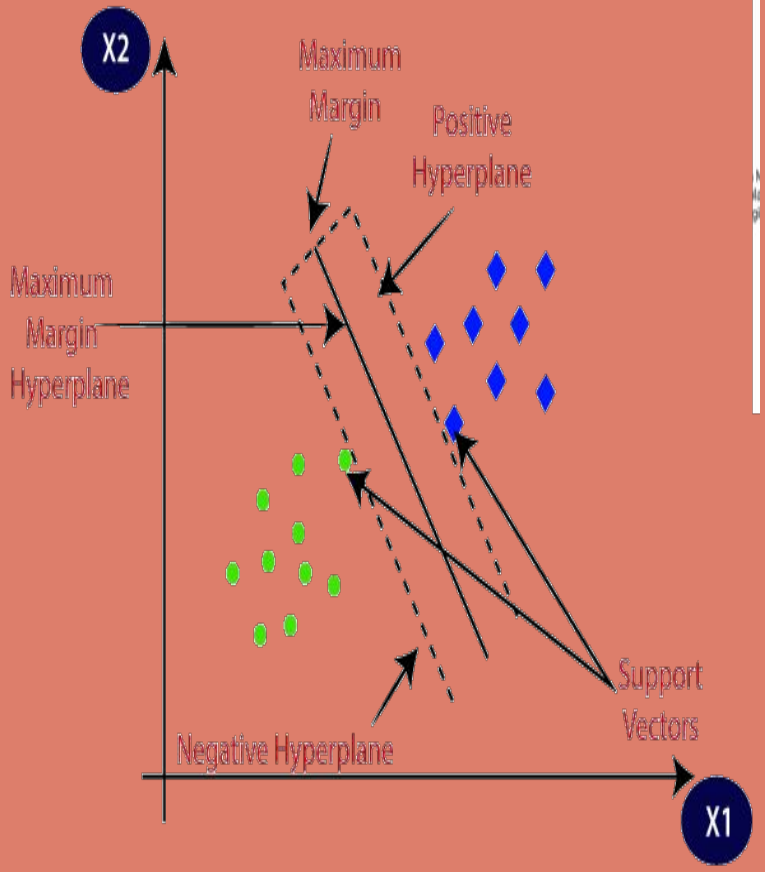
Artificial Neural Networks

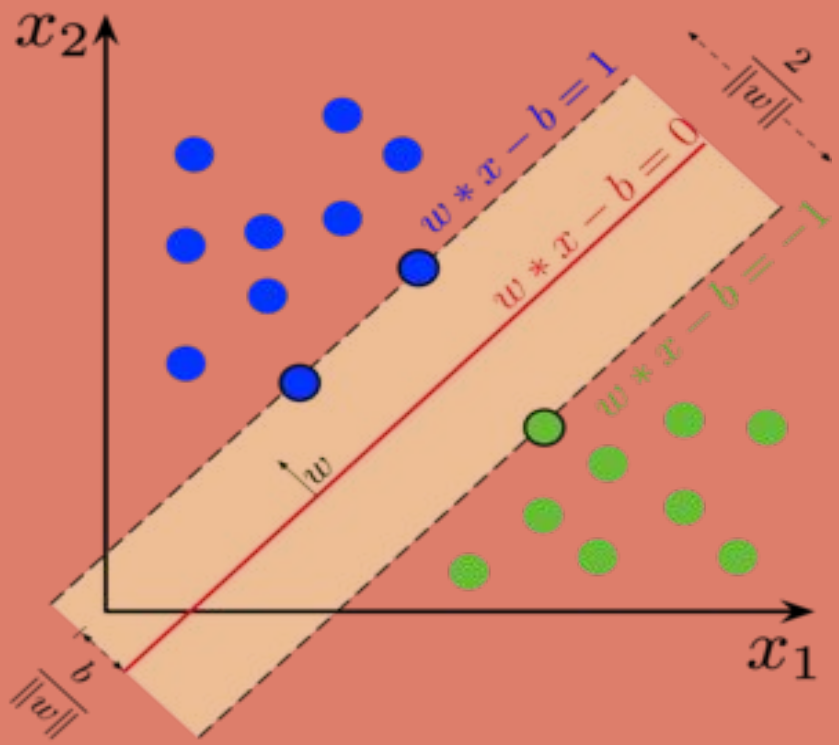
- Forward propagation and back propagation
- Optimization of neural networks
- Stochastic gradient descent - SGD
- Introduction to deep learning
- Solving methodology
- Deep learning software

Support Vector Machines

- Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.
- A **support vector machine** (**SVM**) can be imagined as a surface that maximizes the boundaries between various types of points of data that is represent in multidimensional space, also known as a hyperplane, which creates the most homogeneous points in each subregion.
- Support vector machines can be used on any type of data, but have special extra advantages for data types with very high dimensions relative to the observations.
- For example:
  - Text classification, in which language has the very dimensions of word vectors
  - For the quality control of DNA sequencing by labeling chromatograms correctly

Support Vector Machines





Support Vector Machines

Support vector machines working principles

- Support vector machines are mainly classified into three types based on their working principles:
  - Maximum margin classifiers
  - Support vector classifiers
  - Support vector machines

Support Vector Machines

Maximum margin classifier

- It is feasible to draw infinite hyperplanes to classify the same set of data
- But to consider as an ideal hyperplane, The maximum margin classifier considers the hyperplane with the maximum margin of separation width.

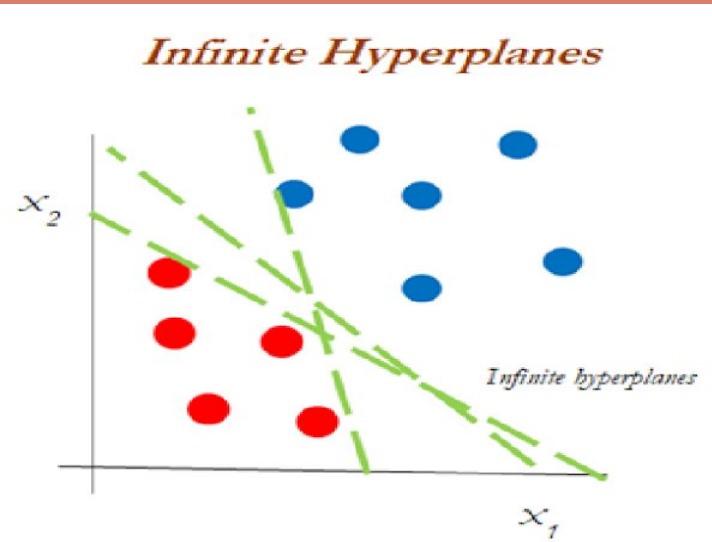
Hyperplanes:

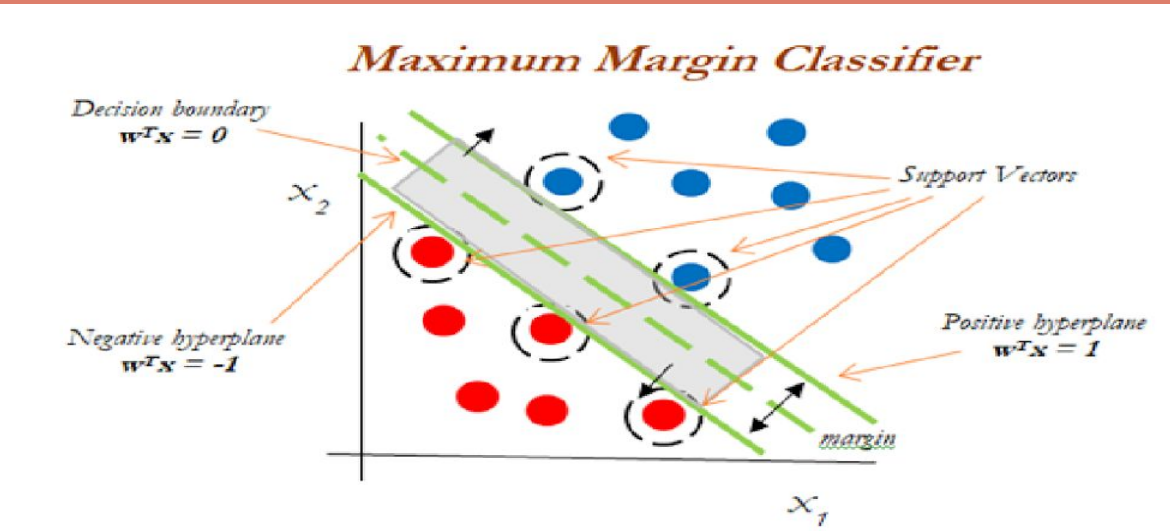
- In *n- dimensional* space, a hyperplane is a flat affine subspace of dimension n-1.
- In 2- dimensional space, the hyperplane is a straight line which separates the 2-dimensional space into two halves.
- The hyperplane is defined by the following equation.

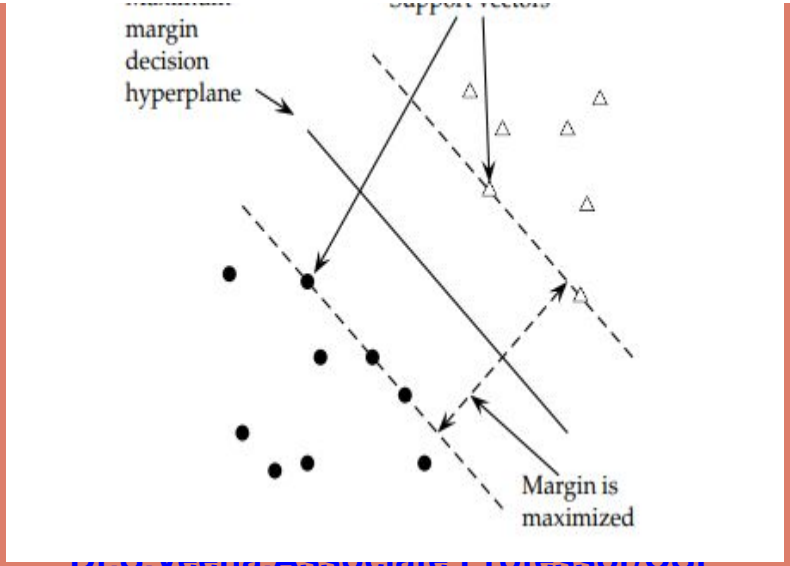
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

- Points which lay on the hyperplane have to follow the above equation.

Support Vector Machines







Support Vector Machines

- However, there are regions above and below as well. This means observations could fall in either of the regions, also called the region of classes

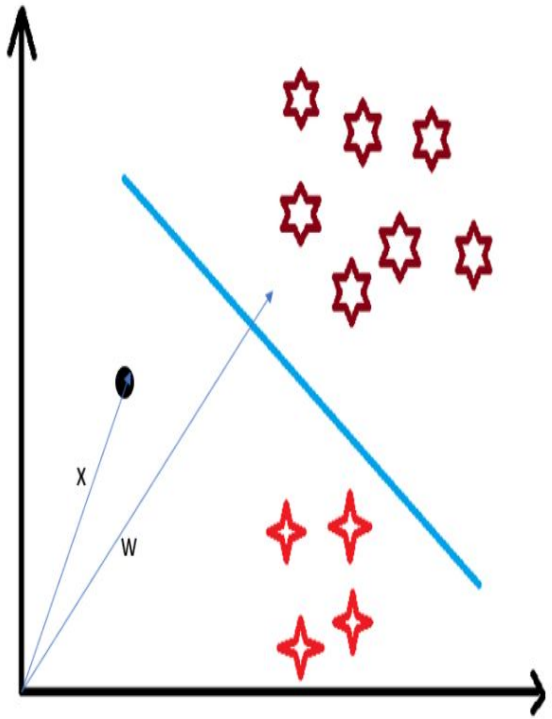
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$$

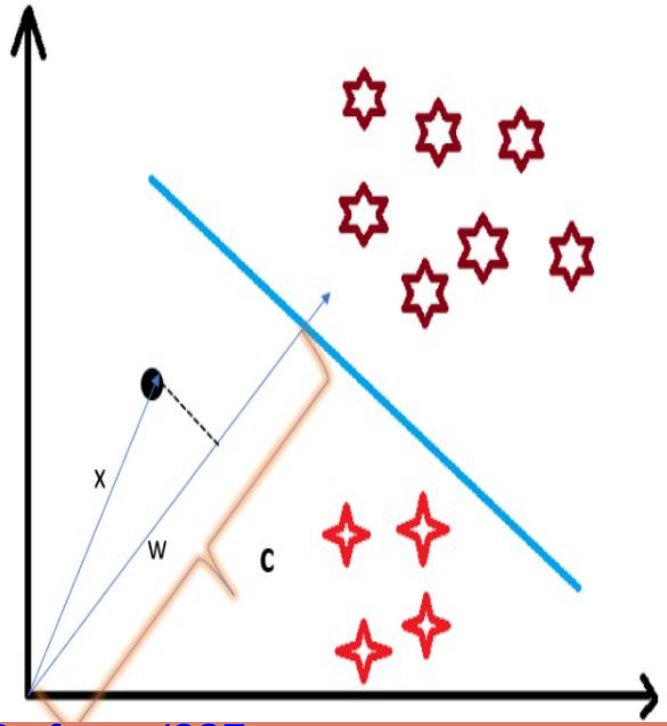
Support Vector Machines

Use of Dot Product in SVM:

Consider a random point X and we want to know whether it lies on the right side of the plane or the left sic
the plane (positive or negative).



To find this first we assume this point is a vector (X) and then we make a vector (w) which is perpendicular to the hyperplane. Let's say the distance of vector w from origin to decision boundary is 'c'. Now we take the projection of X vector on w.



Support Vector Machines

We already know that projection of any vector or another vector is called dot-product. Hence, we take the dot product of x and w vectors. If the dot product is greater than 'c' then we can say that the point lies on the right side. If the dot product is less than 'c' then the point is on the left side and if the dot product is equal to 'c' then the point lies on the decision boundary.

$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c \ (positive\ samples)$$

$$\vec{X}.\vec{w} < c \ (negative\ samples)$$

Support Vector Machines

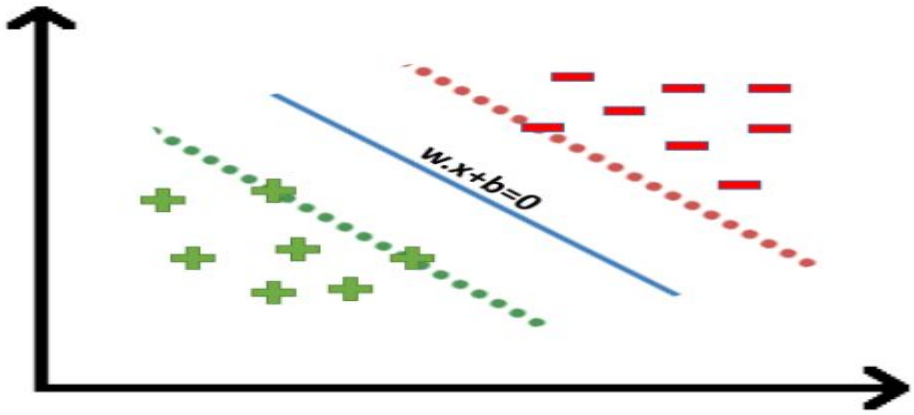
why did we take this perpendicular vector w to the hyperplane?

So what we want is the distance of vector X from the decision boundary and there can be infinite points on the boundary to measure the distance from. So that's why we come to standard, we simply take perpendicular and use it as a reference and then take projections of all the other data points on this perpendicular vector and then compare the distance.

Support Vector Machines

Margin in Support Vector Machine

We all know the equation of a hyperplane is w.x+b=0 where w is a vector normal to hyperplane and b is an offset.



To classify a point as negative or positive we need to define a decision rule. We can define decision rule as:

$$\vec{X}.\vec{w} - c \geq 0$$

putting $-c$ as b, we get

$$\vec{X}.\vec{w} + b \geq 0$$

hence

$$y = \begin{cases} +1 & \text{if } \vec{X}.\vec{w} + b \geq 0 \\ -1 & \text{if } \vec{X}.\vec{w} + b < 0 \end{cases}$$

If the value of w.x+b>0 then we can say it is a positive point otherwise it is a negative point. Now we need (w,b) such that the margin has a maximum distance. Let's say this distance is 'd'.

Support Vector Machines

- The mathematical representation of the maximum margin classifier is as follows, which is an optimization problem

$$\textit{Objective function:} \quad \underset{\beta_0,\beta_1,\ldots,\beta_n}{\text{maximize}}\ M$$

$$\textit{Constraint 1:} \quad \textit{subject to} \sum_{j=1}^{n} \beta_j^2 = 1$$

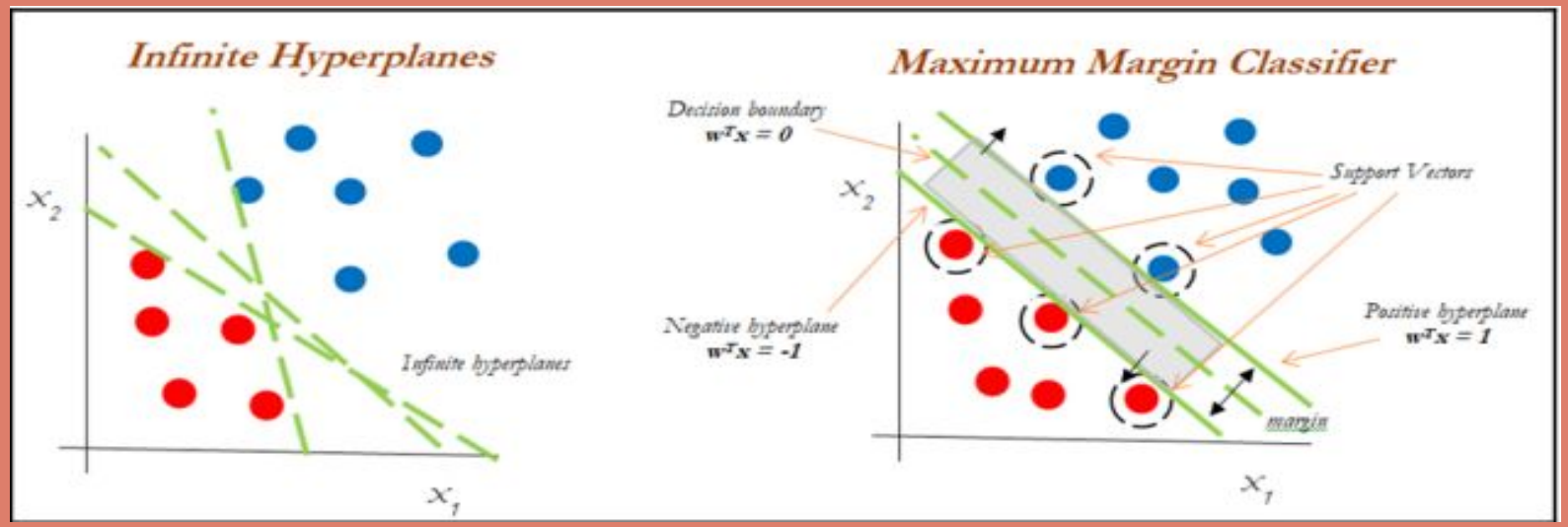
$$\textit{Constraint 2:}\ y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in}) \geq M\ \forall\ i = 1, \ldots, m$$

$$\textit{class labels:}\ y_1, \ldots y_m \in \{-1, 1\}$$

- Constraint 2 ensures that observations will be on the correct side of the hyperplane by taking the product of coefficients with x variables and finally, with a class variable indicator.

Support Vector Machines

- Infinite separate hyperplanes can be drawn to separate the two classes (blue and red).
- However, the maximum margin classifier attempts to fit the widest slab (maximize the margin between positive and negative hyperplanes) between two classes and the observations touching both the positive and negative hyperplanes called support vectors:
- Classifier performance purely depends on the support vectors and any changes to observation values which are not support vectors do not impact any change in the performance of the Maximum Margin Classifier, as only extreme points are considered in the algorithm.



Support Vector Machines

Support vector classifier

- Support vector classifiers are an extended version of maximum margin classifiers, in which some violations are tolerated for non-separable cases in order to create the best fit, even with slight errors within the threshold limit.
- In fact, in real-life scenarios, we hardly find any data with purely separable classes; most classes have a few or more observations in overlapping classes.
- The mathematical representation of the support vector classifier is as follows, a slight correction to the constraints to accommodate error terms

$$Objective\ function: \quad \underset{\beta_0,\beta_1,\ldots,\beta_n}{\text{maximize}}\ M$$

Support Vector Machines

Support vector classifier

- The mathematical representation of the support vector classifier is as follows, a slight correction to the constraints to accommodate error terms

$$Objective\ function: \quad \underset{\beta_0,\beta_1,\dots,\beta_n}{\text{maximize}}\ M$$

$$Constraint\ 1: \quad subject\ to \sum_{j=1}^{n} \beta_j^2 = 1$$

$$Constraint\ 2: \quad y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in}) \geq M(1 - \varepsilon_i)\ \forall\ i = 1, \dots., m$$

$$Constraint\ 3: \quad \varepsilon_i \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C$$

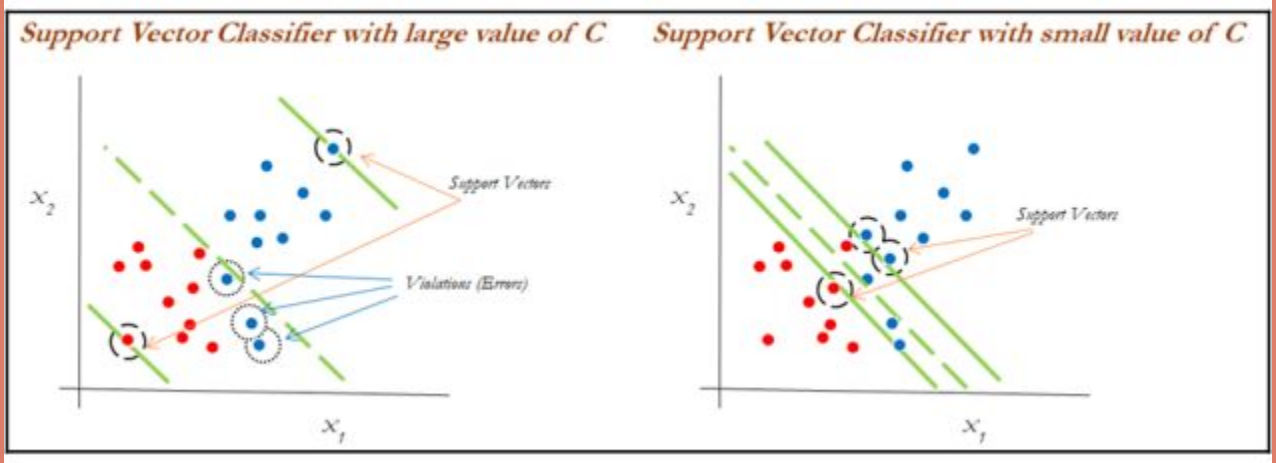
$$class\ labels: \quad y_1, \dots y_m \in \{-1, 1\}$$

- In constraint 3, the C value is a non-negative tuning parameter to either accommodate more or fewer overall errors in the model.

Support Vector Machines

Support vector classifier

- C is a tuning parameter in Support Vector Classifiers
- The impact of changing the C value on margins is shown in the following diagram:

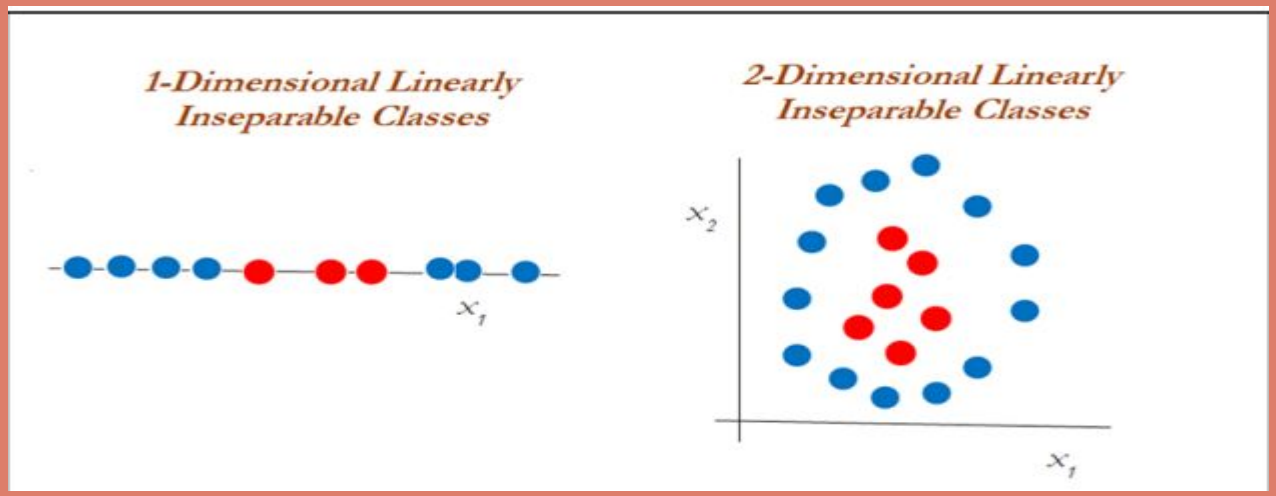


- High value of C - the model would be more tolerating and also have space for violations (errors) in the left diagram,
- Lower value of C, - no scope for accepting violations leads to a reduction in margin width.

Support Vector Machines

Support vector machines

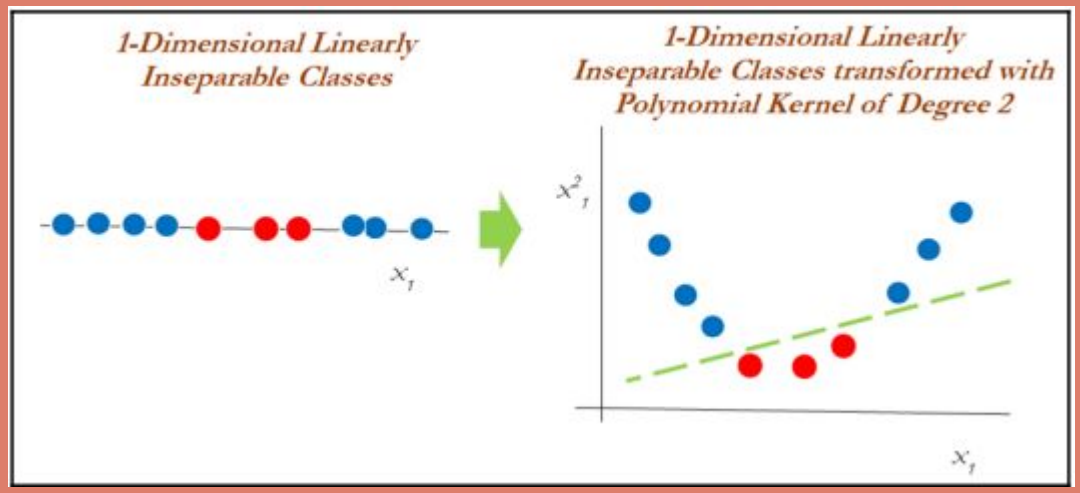
- Support vector machines are used when the decision boundary is non-linear and would not be separable with support vector classifiers whatever the cost function is.
- The following diagram explains the non-linearly separable cases for both 1-dimension and 2-dimensions
- We need to use another way of handling the data, called the kernel trick, using the kernel function to work with non-linearly separable data.



Support Vector Machines

Support vector machines

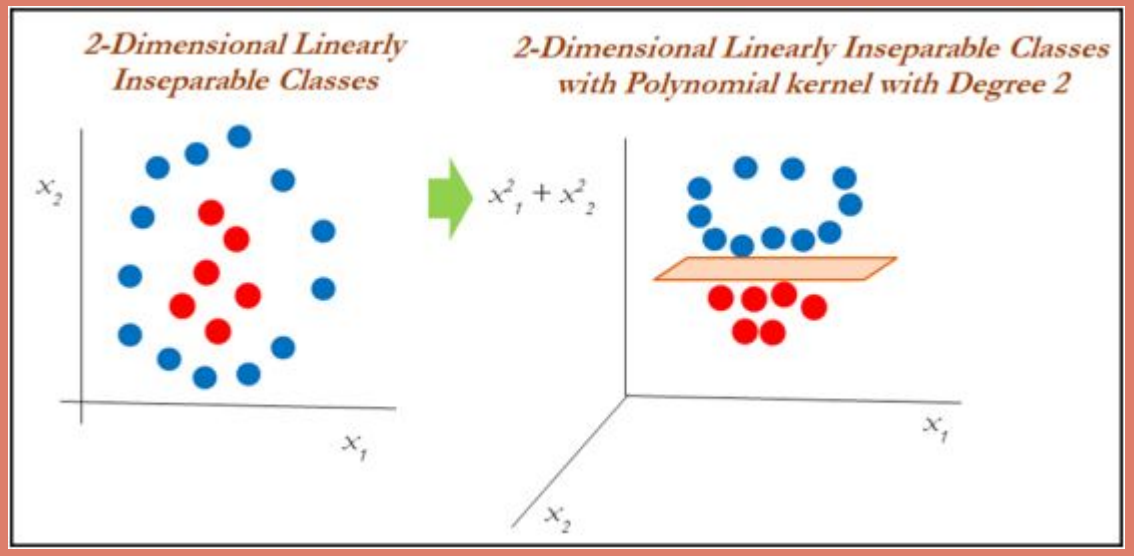
- A polynomial kernel with degree 2 has been applied in transforming the data from 1-dimensional to 2-dimensional data.
- In the left diagram, different classes (red and blue) are plotted on X1 only, whereas after applying degree 2, we now have 2- dimensions, X1 and $X^2 1$ (the original and a new dimension).
- The degree of the polynomial kernel is a tuning parameter;
- the practitioner needs to tune them with various values to check where higher accuracies are possible with the model:



Support Vector Machines

Support vector machines

- In the 2-dimensional case, the kernel trick is applied with the polynomial kernel with degree 2.



- It seems that observations have been classified successfully using a linear plane after projecting the data into higher dimensions:

Support Vector Machines

Kernel functions

- Kernel functions are the functions that, given the original feature vectors, return the same value as the dot product of its corresponding mapped feature vectors.
- The main reason for using kernel functions is to eliminate the computational requirement to derive the higher-dimensional vector space from the given basic vector space, so that observations be separated linearly in higher dimensions.
- Need - The derived vector space will grow exponentially with the increase in dimensions and it will become almost too difficult to continue computation, even when you have a variable size of 30 or so.

Support Vector Machines

Kernel functions

- Example - shows how the size of the variables grows
- When we have two variables such as *x* and *y*, with a polynomial degree kernel, it needs to compute x^2, y^2, and *xy* dimensions in addition.
- Whereas, if we have three variables x, y, and z, then we need to calculate the x^2, y^2, z^2, *xy*, *yz*, *xz*, and *xyz* vector spaces.
- The increase of one more dimension creates so many combinations. Hence, care needs to be taken to reduce its computational complexity;
- Thus Kernels are used and defined more formally in the following equation:

$$K(x,z) = < \Phi(x), \Phi(z) >$$

Support Vector Machines

Kernel functions

- **Polynomial Kernel**:
- Polynomial kernels are popularly used, especially with degree 2.
- In fact, the inventor of support vector machines, *Vladimir N Vapnik*, developed using a degree 2 kernel for classifying handwritten digits.
- Polynomial kernels are given by the following equation:

$$K(x, x') = (1 + x * x')^k$$

For reference : SVM | Support Vector Machine Algorithm in Machine Learning (analyticsvidhya.com)

Support Vector Machines

Kernel functions

- **Radial Basis Function (RBF) / Gaussian Kernel**:
- RBF kernels are a good first choice for problems requiring nonlinear models.
- A decision boundary that is a hyperplane in the mapped feature space is similar to a decision boundary that is a hypersphere in the original space.
- The feature space produced by the Gaussian kernel can have an infinite number of dimensions, a feat that would be impossible otherwise. RBF kernels are represented by the following equation

$$K(x, x') = e(-\,||x - x'\,||^2\ /\ \sigma^2)$$

- This is often simplified as the following equation:

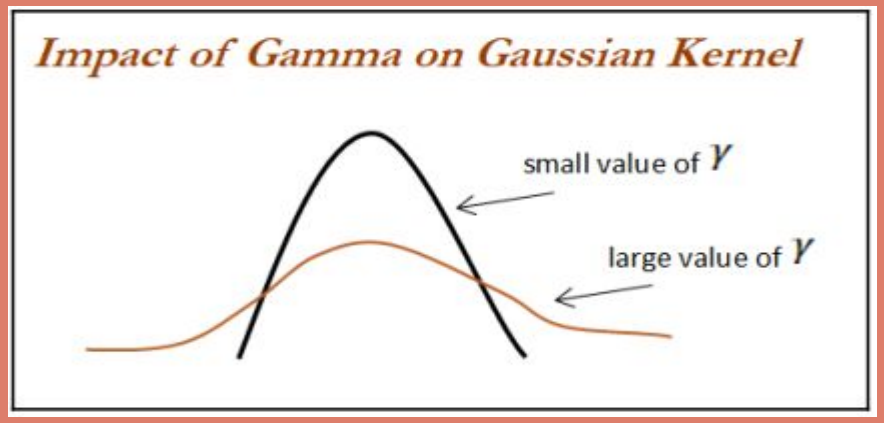
$$K(x, x') = e(-\,\gamma\,||x - x'\,||^2\ );\ \ \gamma\ = hyperparameter$$

Support Vector Machines

Kernel functions

$$K(x, x') = e(-\gamma \, ||x - x' \, ||^2 \,) \; ; \;\; \gamma \; = hyperparameter$$

- When the value of the gamma value is small, it gives you a pointed bump in the higher dimensions; a larger value gives you a softer, broader bump.
- A small gamma will give you low bias and high variance solutions;
- a high gamma will give high bias and low variance solutions



Support Vector Machines

SVM multilabel classifier with letter recognition data example

- Dataset : https://archive.ics.uci.edu/ml/datasets/letter+recognition.
- The task is to identify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters in the English alphabet (from A to Z; 26 classes altogether) based on a few characteristics in integers, such as x-box (horizontal position of box), y-box (vertical position of box), width of the box, height of the box, and so on:

```
>>> import os

""" First change the following directory link to
where all input files do exist """
>>> os.chdir("D:\\Book writing\\Codes\\Chapter 6")
>>> import pandas as pd
>>> letterdata = pd.read_csv("letterdata.csv")
>>> print (letterdata.head())
```

Support Vector Machines

SVM multilabel classifier with letter recognition data example

letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

Following code is used to remove the target variable from x variables and at the same time create new y variable for convenience:

```
>>> x_vars = letterdata.drop(['letter'],axis=1)
>>> y_var = letterdata["letter"]
```

Support Vector Machines

SVM multilabel classifier with letter recognition data example

- As scikit-learn does not directly support characters, we need to convert them into number mappings. Here, we have done so with the dictionary:

```
>>> y_var = y_var.replace({'A':1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7,
'H':8,'I':9,
'J':10,'K':11,'L':12,'M':13,'N':14,'O':15,'P':16,'Q':17,'R':18,'S':19,'
T':2 0,'U':21, 'V':22, 'W':23,'X':24,'Y':25,'Z':26})

>>> from sklearn.metrics import accuracy_score,classification_report

>>> from sklearn.model_selection import train_test_split
>>> x_train,x_test,y_train,y_test =
train_test_split(x_vars,y_var,train_size = 0.7,random_state=42)


# Linear Classifier

    >>> from sklearn.svm import SVC
```

Support Vector Machines

Maximum margin classifier - linear kernel

The following code shows a linear classifier (also known as a maximum margin classifier) with cost value as 1.0:

```
>>> svm_fit = SVC(kernel='linear',C=1.0,random_state=43)

>>> svm_fit.fit(x_train,y_train)
>>> print ("\nSVM Linear Classifier - Train Confusion    Matrix\n\n",

pd.crosstab(y_train, svm_fit.predict(x_train),rownames = ["Actuall"],colnames =
["Predicted"]) )

>>> print ("\nSVM Linear Classifier - Train
accuracy:",round(accuracy_score(y_train, svm_fit.predict(x_train)),3))
>>> print ("\nSVM Linear Classifier - Train Classification Report\n",
classification_report(y_train,svm_fit.predict(x_train)))
```

Support Vector Machines

Maximum margin classifier - linear kernel

SVM Linear Classifier - Train accuracy: 0.876

SVM Linear Classifier - Train Classification Report

	precision	recall	f1-score	support
1	0.93	0.97	0.95	557
2	0.82	0.90	0.86	537
3	0.89	0.91	0.90	535
4	0.82	0.92	0.87	555
5	0.81	0.84	0.83	530
6	0.84	0.89	0.86	564
7	0.76	0.80	0.78	543
8	0.75	0.71	0.73	516
9	0.92	0.88	0.90	534
10	0.89	0.90	0.90	519
11	0.84	0.84	0.84	551
12	0.91	0.89	0.90	530
13	0.93	0.92	0.93	540
14	0.94	0.93	0.94	552
15	0.89	0.78	0.83	535
16	0.96	0.89	0.92	555
17	0.88	0.84	0.86	530
18	0.81	0.85	0.82	524
19	0.75	0.73	0.74	513
20	0.92	0.90	0.91	564
21	0.95	0.94	0.94	552
22	0.92	0.92	0.92	527
23	0.93	0.95	0.94	539
24	0.89	0.89	0.89	542
25	0.94	0.91	0.92	535
26	0.90	0.84	0.87	521
avg / total	0.88	0.88	0.88	14000

Support Vector Machines

Maximum margin classifier - linear kernel

Following code used for printing the test accuracy values:

```
>>> print ("\n\nSVM Linear Classifier - Test Confusion Matrix\n\n",pd.crosstab(y_test, svm_fit.predict(x_test),rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nSVM Linear Classifier - Test accuracy:",round(accuracy_score( y_test,svm_fit.predict(x_test)),3))
>>> print ("\nSVM Linear Classifier - Test Classification Report\n", classification_report(y_test,svm_fit.predict(x_test)))
```

Support Vector Machines

Maximum margin classifier - linear kernel

From the above results, we can see that test accuracy for the linear classifier is 85%

SVM Linear Classifier - Test accuracy: 0.85

SVM Linear Classifier - Test Classification Report

	precision	recall	f1-score	support
1	0.87	0.94	0.90	232
2	0.80	0.86	0.83	229
3	0.86	0.86	0.86	201
4	0.77	0.89	0.83	250
5	0.81	0.88	0.84	238
6	0.81	0.89	0.85	211
7	0.74	0.75	0.74	230
8	0.70	0.63	0.67	218
9	0.89	0.85	0.87	221
10	0.85	0.85	0.85	228
11	0.77	0.79	0.78	188
12	0.93	0.88	0.90	231
13	0.95	0.93	0.94	252
14	0.91	0.90	0.91	231
15	0.86	0.79	0.82	218
16	0.97	0.83	0.89	248
17	0.84	0.77	0.81	253
18	0.74	0.82	0.78	234
19	0.73	0.72	0.73	235
20	0.90	0.89	0.89	232
21	0.94	0.90	0.92	261
22	0.89	0.93	0.91	237
23	0.89	0.94	0.91	213
24	0.91	0.88	0.89	245
25	0.94	0.88	0.91	251
26	0.84	0.82	0.83	213
avg / total	0.85	0.85	0.85	6000

Support Vector Machines

Polynomial kernel

- A polynomial kernel with degree of 2 has been used to check whether any improvement in accuracy is possible.
- The cost value has been kept constant with respect to the linear classifier in order to determine the impact of the non-linear kernel:

```
#Polynomial Kernel
>>> svm_poly_fit = SVC(kernel='poly',C=1.0,degree=2)
>>> svm_poly_fit.fit(x_train,y_train)
>>> print ("\nSVM Polynomial Kernel Classifier - Train Confusion
Matrix\n\n",pd.crosstab(y_train,svm_poly_fit.predict(x_train),rowname
s = ["Actuall"],colnames = ["Predicted"]) )
>>> print ("\nSVM Polynomial Kernel Classifier - Train
accuracy:",round(accuracy_score(
y_train,svm_poly_fit.predict(x_train)),3))
>>> print ("\nSVM Polynomial Kernel Classifier - Train Classification
Report\n",
classification_report(y_train,svm_poly_fit.predict(x_train)))
```

Support Vector Machines

Polynomial kernel

SVM Polynomial Kernel Classifier - Train accuracy: 0.989

SVM Polynomial Kernel Classifier - Train Classification Report

	precision	recall	f1-score	support
1	1.00	1.00	1.00	557
2	0.98	0.98	0.98	537
3	1.00	0.99	0.99	535
4	0.97	0.99	0.98	555
5	0.99	0.98	0.98	530
6	0.98	0.99	0.98	564
7	0.98	0.99	0.98	543
8	0.98	0.95	0.96	516
9	0.98	0.98	0.98	534
10	0.99	0.98	0.98	519
11	0.99	0.98	0.98	551
12	1.00	0.99	1.00	530
13	1.00	1.00	1.00	540
14	1.00	0.99	0.99	552
15	0.99	1.00	0.99	535
16	0.99	0.97	0.98	555
17	1.00	1.00	1.00	530
18	0.96	0.98	0.97	524
19	0.99	0.99	0.99	513
20	1.00	0.99	1.00	564
21	0.99	1.00	1.00	552
22	0.99	0.99	0.99	527
23	1.00	1.00	1.00	539
24	0.99	1.00	1.00	542
25	1.00	1.00	1.00	535
26	1.00	1.00	1.00	521
avg / total	0.99	0.99	0.99	14000

Support Vector Machines

Polynomial kernel

>>> print ("\n\nSVM Polynomial Kernel Classifier - Test Confusion Matrix\n\n", pd.crosstab(y_test,svm_poly_fit.predict(x_test),rownames = ["Actuall"],colnames = ["Predicted"]))

>>> print ("\nSVM Polynomial Kernel Classifier - Test accuracy:",round(accuracy_score(y_test,svm_poly_fit.predict(x_test)),3))

>>> print ("\nSVM Polynomial Kernel Classifier - Test Classification Report\n", classification_report(y_test,svm_poly_fit.predict(x_test)))

```
SVM Polynomial Kernel Classifier - Test accuracy: 0.954

SVM Polynomial Kernel Classifier - Test Classification Report
             precision    recall  f1-score   support

          1       0.98      0.99      0.99       232
          2       0.92      0.96      0.94       229
          3       0.92      0.95      0.93       201
          4       0.91      0.94      0.93       250
          5       0.93      0.96      0.95       238
          6       0.92      0.97      0.94       211
          7       0.94      0.93      0.94       230
          8       0.92      0.87      0.89       218
          9       0.96      0.95      0.96       221
         10       0.96      0.94      0.95       228
         11       0.90      0.93      0.91       188
         12       0.98      0.95      0.97       231
         13       0.98      0.98      0.98       252
         14       0.96      0.94      0.95       231
         15       0.94      0.95      0.94       218
         16       0.99      0.95      0.97       248
         17       0.98      0.93      0.95       253
         18       0.91      0.94      0.92       234
         19       0.98      0.98      0.98       235
         20       0.98      0.97      0.97       232
         21       0.98      0.97      0.98       261
         22       0.98      0.96      0.97       237
         23       0.96      0.97      0.96       213
         24       0.95      0.96      0.96       245
         25       0.97      0.98      0.97       251
         26       0.98      0.98      0.98       213

avg / total       0.95      0.95      0.95      6000
```

Support Vector Machines

RBF kernel

The cost value is kept constant with respective other kernels but the gamma value has been chosen as 0.1 to fit the model:

```
#RBF Kernel
>>> svm_rbf_fit = SVC(kernel='rbf',C=1.0, gamma=0.1)
>>> svm_rbf_fit.fit(x_train,y_train)
>>> print ("\nSVM RBF Kernel Classifier - Train Confusion Matrix\n\n",pd.crosstab( y_train,svm_rbf_fit.predict(x_train),rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nSVM RBF Kernel Classifier - Train accuracy:",round(accuracy_score( y_train, svm_rbf_fit.predict(x_train)),3))
>>> print ("\nSVM RBF Kernel Classifier - Train Classification Report\n", classification_report(y_train,svm_rbf_fit.predict(x_train)))
```

Support Vector Machines

RBF kernel

SVM RBF Kernel Classifier - Train accuracy: 0.998

SVM RBF Kernel Classifier - Train Classification Report

	precision	recall	f1-score	support
1	1.00	1.00	1.00	557
2	1.00	1.00	1.00	537
3	1.00	1.00	1.00	535
4	1.00	1.00	1.00	555
5	1.00	0.99	1.00	530
6	0.99	1.00	0.99	564
7	0.99	1.00	0.99	543
8	0.99	0.99	0.99	516
9	1.00	0.99	0.99	534
10	0.99	1.00	0.99	519
11	1.00	1.00	1.00	551
12	1.00	1.00	1.00	530
13	1.00	1.00	1.00	540
14	1.00	1.00	1.00	552
15	1.00	1.00	1.00	535
16	1.00	0.99	1.00	555
17	1.00	1.00	1.00	530
18	0.99	1.00	1.00	524
19	1.00	1.00	1.00	513
20	1.00	1.00	1.00	564
21	1.00	1.00	1.00	552
22	0.99	1.00	1.00	527
23	1.00	1.00	1.00	539
24	1.00	1.00	1.00	542
25	1.00	1.00	1.00	535
26	1.00	1.00	1.00	521
avg / total	1.00	1.00	1.00	14000

Support Vector Machines

RBF kernel

```
>>> print ("\n\nSVM RBF Kernel Classifier - Test Confusion Matrix\n\n",
pd.crosstab(y_test,svm_rbf_fit.predict(x_test),rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\nSVM RBF Kernel Classifier - Test accuracy:",round(
accuracy_score( y_test,svm_rbf_fit.predict(x_test)),3))
>>> print ("\nSVM RBF Kernel Classifier - Test Classification Report\n",
classification_report(y_test,svm_rbf_fit.predict(x_test)))
```

Support Vector Machines

RBF kernel

SVM RBF Kernel Classifier - Test accuracy: 0.969

SVM RBF Kernel Classifier - Test Classification Report

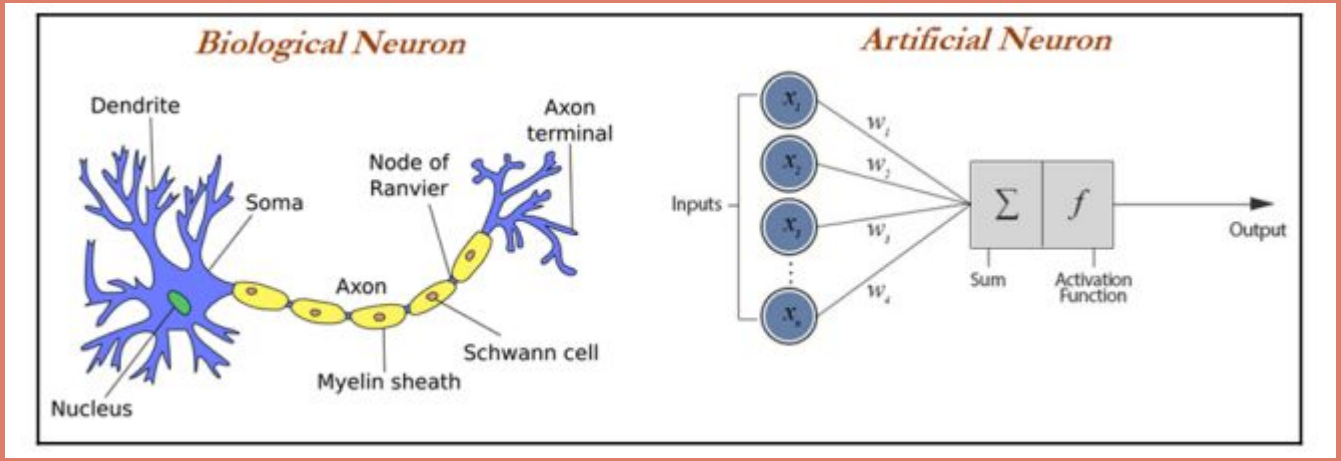
	precision	recall	f1-score	support
1	0.99	1.00	0.99	232
2	0.93	0.97	0.95	229
3	0.98	0.93	0.96	201
4	0.95	0.97	0.96	250
5	0.98	0.98	0.98	238
6	0.96	0.98	0.97	211
7	0.97	0.96	0.96	230
8	0.94	0.87	0.90	218
9	0.98	0.95	0.97	221
10	0.95	0.96	0.96	228
11	0.93	0.94	0.93	188
12	0.99	0.98	0.98	231
13	0.96	1.00	0.98	252
14	0.99	0.95	0.97	231
15	0.94	0.97	0.95	218
16	0.99	0.96	0.97	248
17	0.99	0.97	0.98	253
18	0.89	0.96	0.92	234
19	0.99	0.99	0.99	235
20	0.98	0.99	0.99	232
21	1.00	0.99	0.99	261
22	0.97	0.97	0.97	237
23	0.99	1.00	0.99	213
24	0.98	0.96	0.97	245
25	0.98	0.99	0.99	251
26	1.00	0.99	0.99	213
avg / total	0.97	0.97	0.97	6000

Artificial Neural Networks - ANN

- **Artificial neural networks** (**ANN**s) model the relationship between a set of input signals and output signals using a model derived from a replica of the biological brain, which responds to stimuli from its sensory inputs.
- The human brain consists of about 90 billion neurons, with around 1 trillion connections between them;
- ANN methods try to model problems using interconnected artificial neurons (or nodes) to solve machine learning problems
- In Human Brain
  - Incoming signals are received by the cell's dendrites through a biochemical process that allows the impulses to be weighted according to their relative importance.
  - As the cell body begins to accumulate the incoming signals, a threshold is reached, at which the cell fires and the output signal is then transmitted via an electrochemical process down the axon.
  - At the axon terminal, an electric signal is again processed as a chemical signal to be passed to its neighboring neurons, which will be dendrites to some other neuron.

Artificial Neural Networks - ANN

- A similar working principle is loosely used in building an artificial neural network, in which each neuron has a set of inputs, each of which is given a specific weight.
- The neuron computes a function on these weighted inputs.
- A linear neuron takes a linear combination of weighted input and applies an activation function (sigmoid, tanh, relu, and so on) on the aggregated sum.
- The network feeds the weighted sum of the input into the logistic function (in case of sigmoid function).
- The logistic function returns a value between 0 and 1 based on the set threshold; for example, here we set the threshold as 0.7.
- Any accumulated signal greater than 0.7 gives the signal of 1 and vice versa; any accumulated signal less than 0.7 returns the value of 0:



Artificial Neural Networks - ANN

- A typical artificial neuron with n input dendrites can be represented by the following formula.
- The w weights allow each of the n inputs of x to contribute a greater or lesser amount to the sum of input signals.
- The accumulated value is passed to the activation function, f(x), and the resulting signal, y(x), is the output axon:

$$y(x) = f(\sum_{i=1}^{n} w_i x_i)$$

Artificial Neural Networks - ANN

The parameters required for choosing for building neural networks are the following:

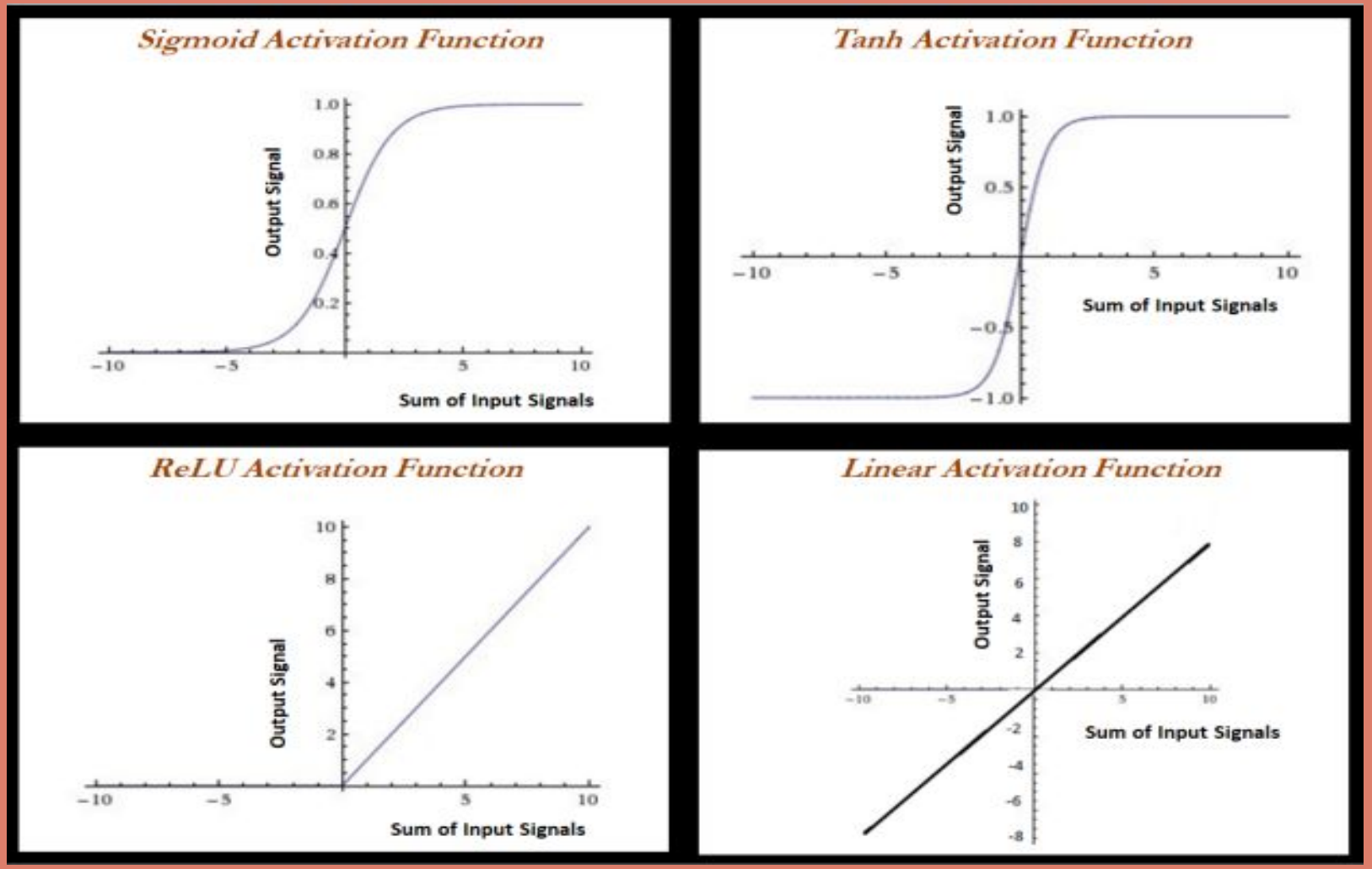
- **Activation function:** Choosing an activation function plays a major role in aggregating signals into the output signal to be propagated to the other neurons of the network.
- **Network architecture or topology:** This represents the number of layers required and the number of neurons in each layer. More layers and neurons will create a highly non-linear decision boundary, whereas if we reduce the architecture, the model will be less flexible and more robust.
- **Training optimization algorithm:** The selection of an optimization algorithm plays a critical role as well, in order to converge quickly and accurately to the best optimal solutions
- **Applications of Neural Networks:**
  - **Images and videos:** To identify an object in an image or to classify whether it is a dog or a cat
  - **Text processing (NLP):** Deep-learning-based chatbot and so on
  - **Speech:** Recognize speech
  - **Structured data processing:** Building highly powerful models to obtain a non-linear decision boundary

Artificial Neural Networks - ANN

Activation functions

- Activation functions are the mechanisms by which an artificial neuron processes information and passes it throughout the network.
- The activation function takes a single number and performs a certain fixed mathematical functional mapping on it.
- Different types of activation functions are :
- **Sigmoid function:** Sigmoid has the mathematical form $\sigma(x) = 1 / (1+e-x)$. It takes a real- valued number and squashes it into a range between 0 and 1. Sigmoid is a popular choice, which makes calculating derivatives easy and is easy to interpret.
- **Tanh function:** Tanh squashes the real-valued number into the range $[-1, 1]$. The output is zero-centered. In practice, tanh non-linearity is always preferred to sigmoid non-linearity. Also, it can be proved that tanh is scaled sigmoid neuron $\tanh(x) = 2\sigma(2x) - 1$.
- **Rectified Linear Unit (ReLU) function:** ReLU has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. Activation is simply thresholds at zero.
- **Linear function:** The linear activation function is used in linear regression problems, where it always provides a derivative as 1 due to the function used being $f(x) = x$.
- **Relu** is now popularly being used in place of **Sigmoid** or **Tanh** due to its better convergence property.

Artificial Neural Networks - ANN



Artificial Neural Networks - ANN

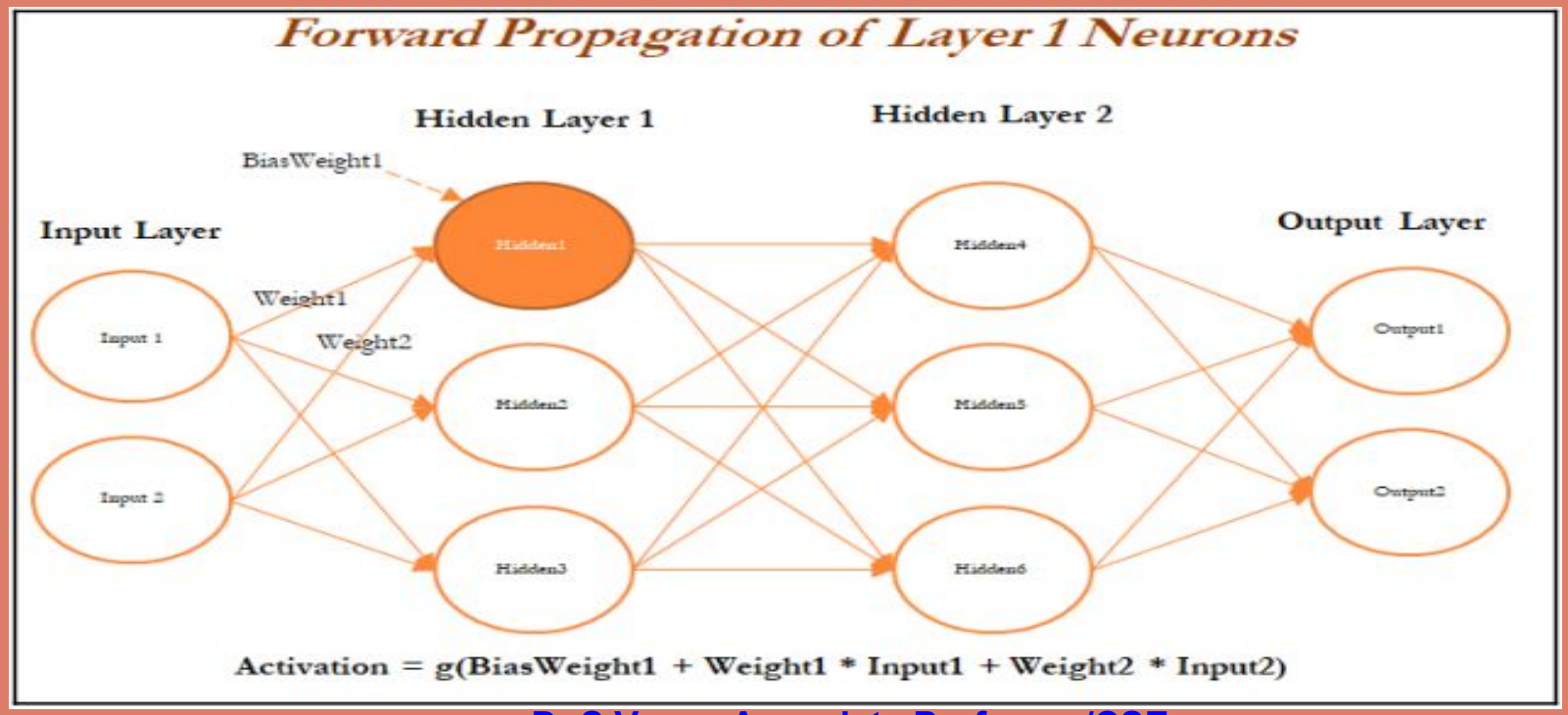
Forward propagation and backpropagation

- Forward propagation and backpropagation are illustrated with the two hidden layer deep neural networks in the following example, in which both layers get three neurons each, in addition to input and output layers.
- The number of neurons in the input layer is based on the number of x (independent) variables, whereas the number of neurons in the output layer is decided by the number of classes the model needs to be predicted.
- For ease, we have shown only one neuron in each layer;
- Weights and biases are initiated from some random numbers, so that in both forward and backward passes, these can be updated in order to minimize the errors altogether.

Artificial Neural Networks - ANN

Forward propagation and backpropagation

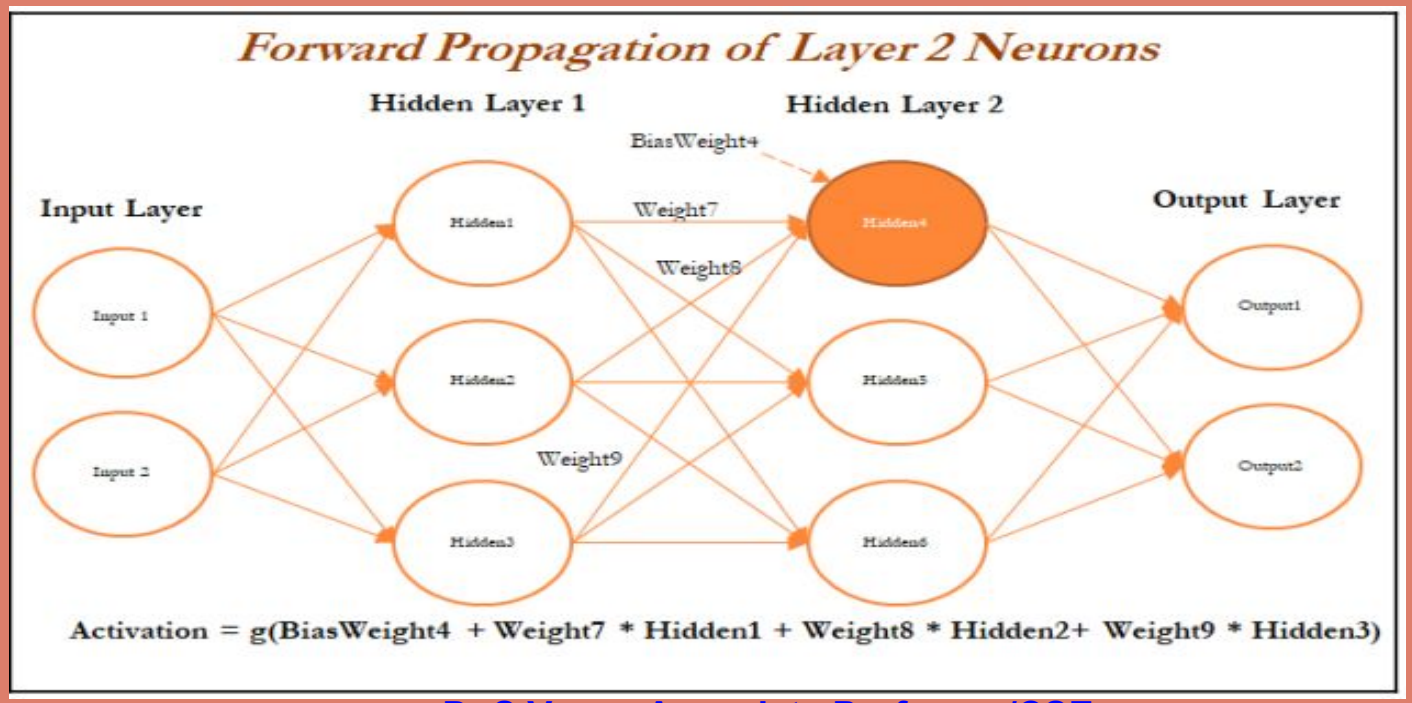
- During forward propagation, features are input to the network and fed through the following layers to produce the output activation.
- In the hidden layer 1, the activation obtained is the combination of bias weight 1 and weighted combination of input values; If the overall value crosses the threshold, it will trigger to the next layer, else the signal will be 0 to the next layer values. Bias values are necessary to control the trigger points.
- In some cases, the weighted combination signal is low; in those cases, bias will compensate the extra amount for adjusting the aggregated value, which can trigger for the next level



Artificial Neural Networks - ANN

Forward propagation and backpropagation

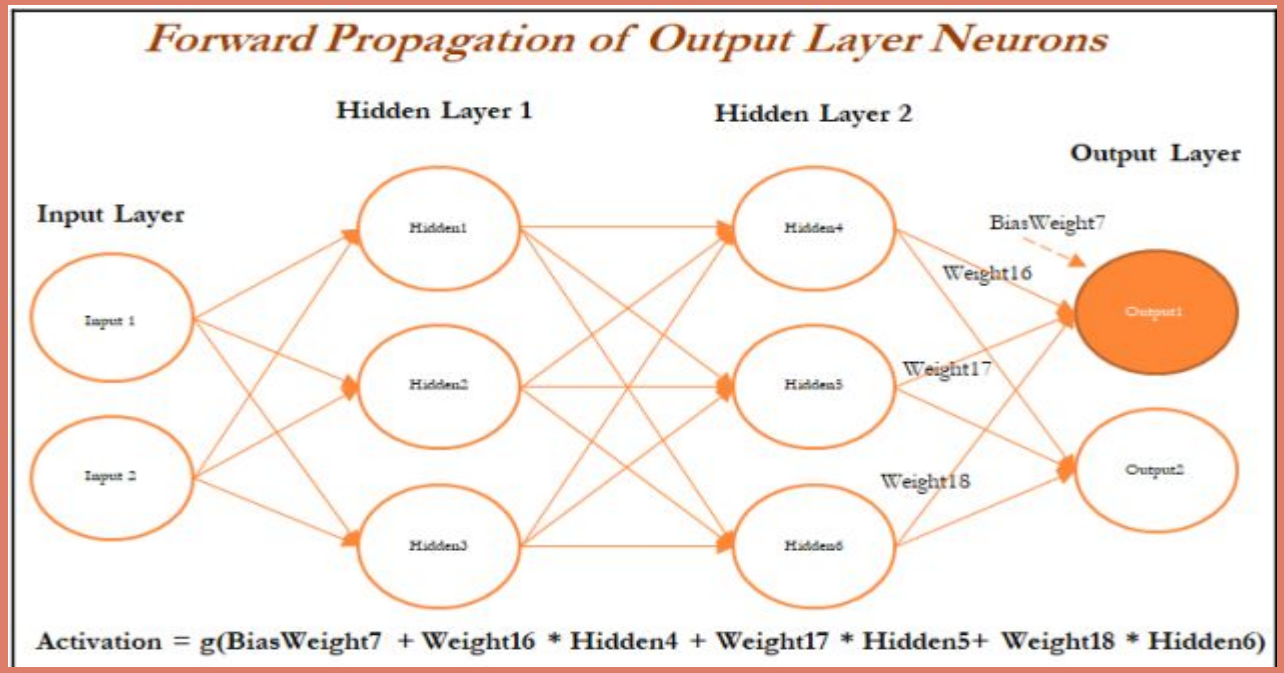
- Once all the neurons are calculated in Hidden Layer 1 (Hidden1, Hidden2, and Hidden3 neurons), the next layer of neurons needs to be calculated in a similar way from the output of the hidden neurons from the first layer with the addition of bias (bias weight 4).
- The following figure describes the hidden neuron 4 shown in layer 2:



Artificial Neural Networks - ANN

Forward propagation and backpropagation

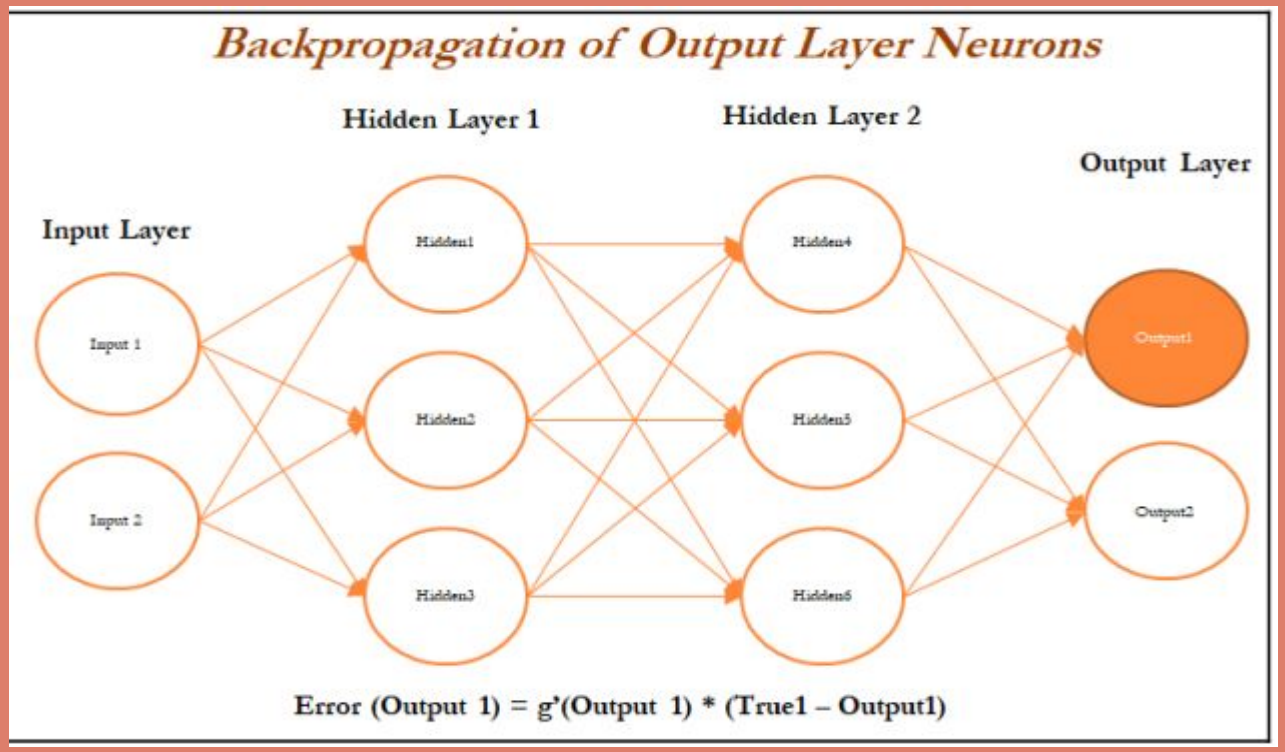
- In the last layer (also known as the output layer), outputs are calculated in the same way from the outputs obtained from hidden layer 2 by taking the weighted combination of weights and outputs obtained from hidden layer 2.
- Once we obtain the output from the model, a comparison needs to be made with the actual value and we need to backpropagate the errors across the net backward in order to correct the weights of the entire neural network:



Artificial Neural Networks - ANN

Forward propagation and backpropagation

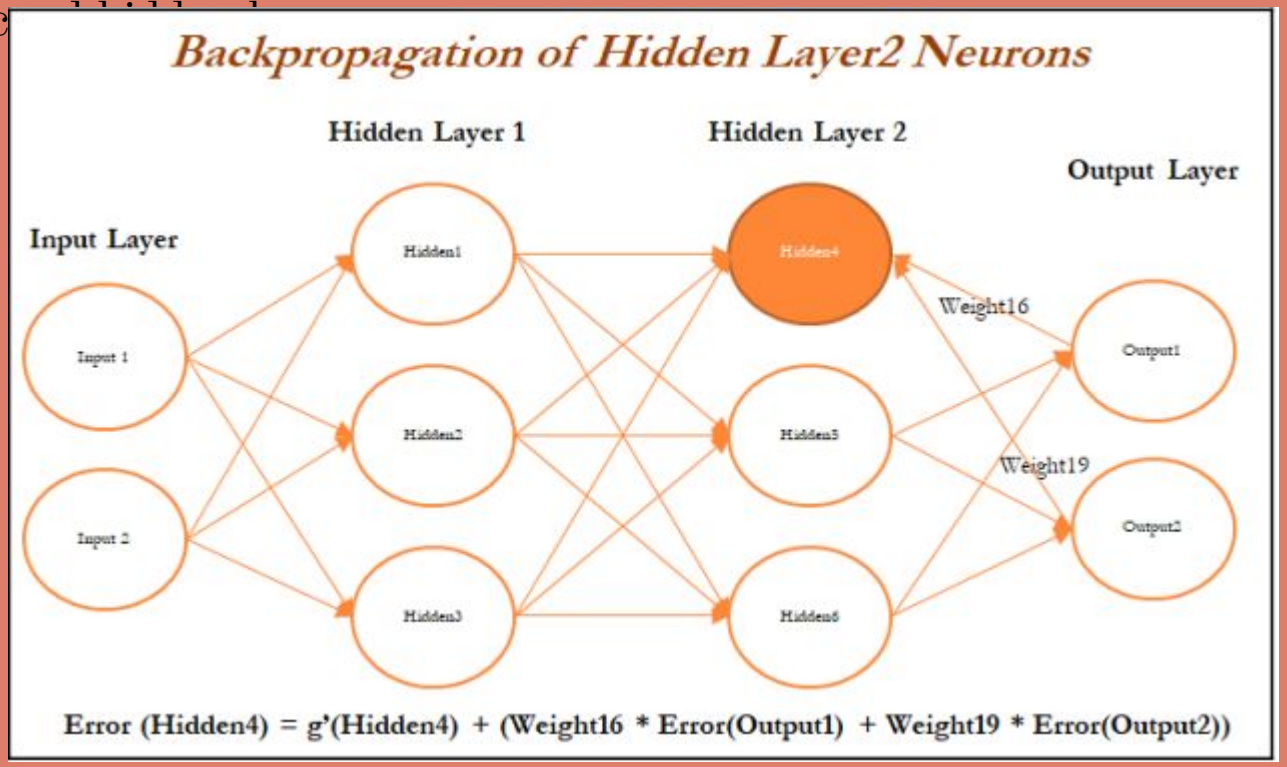
- In the following diagram, we have taken the derivative of the output value and multiplied by that much amount to the error component, which was obtained from differencing the actual value with the model output:
-



Artificial Neural Networks - ANN

Forward propagation and backpropagation

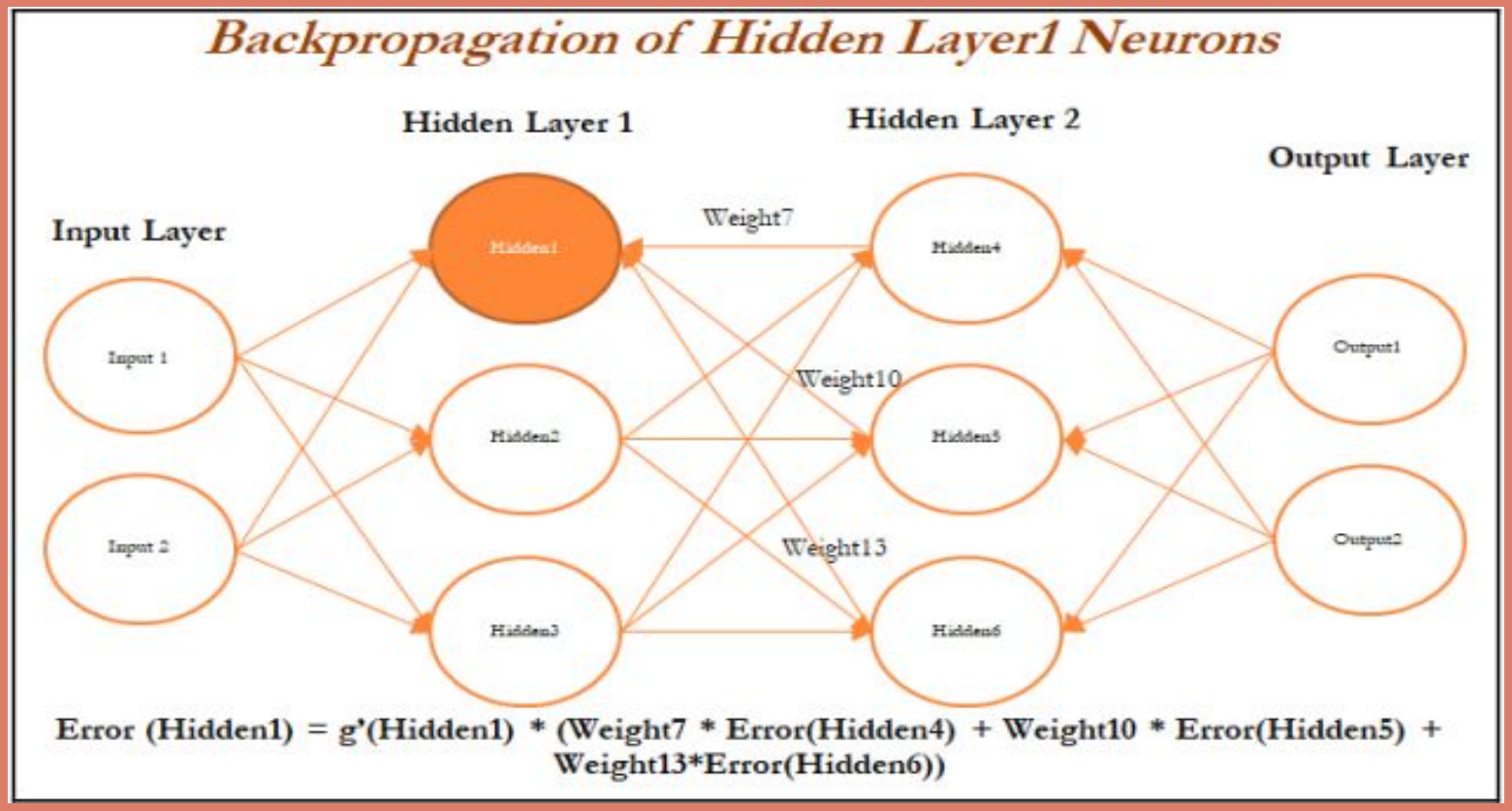
- In a similar way, we will backpropagate the error from the second hidden layer as well.
- In the following diagram, errors are computed from the Hidden 4 neuron in the sec



Artificial Neural Networks - ANN

Forward propagation and backpropagation

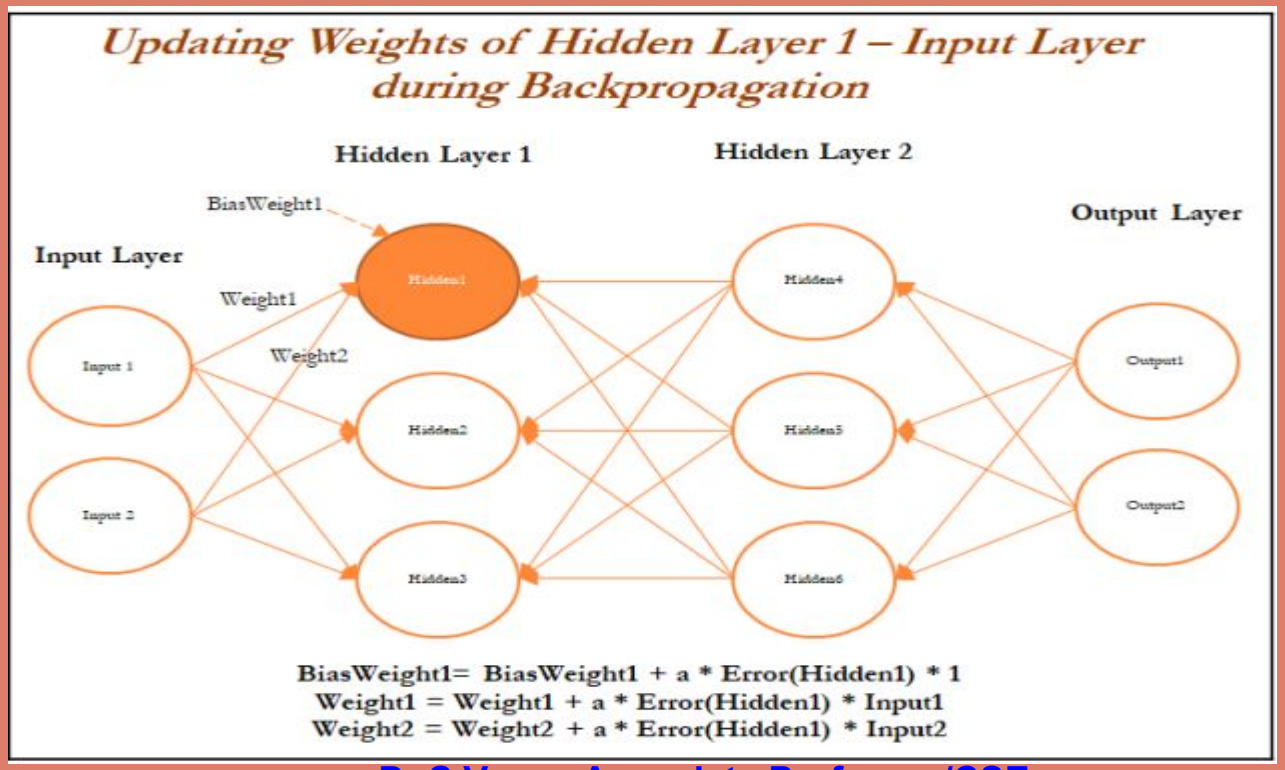
- In the following diagram, errors are calculated for the Hidden 1 neuron in layer 1 based on errors obtained from all the neurons in layer 2:



Artificial Neural Networks - ANN

Forward propagation and backpropagation

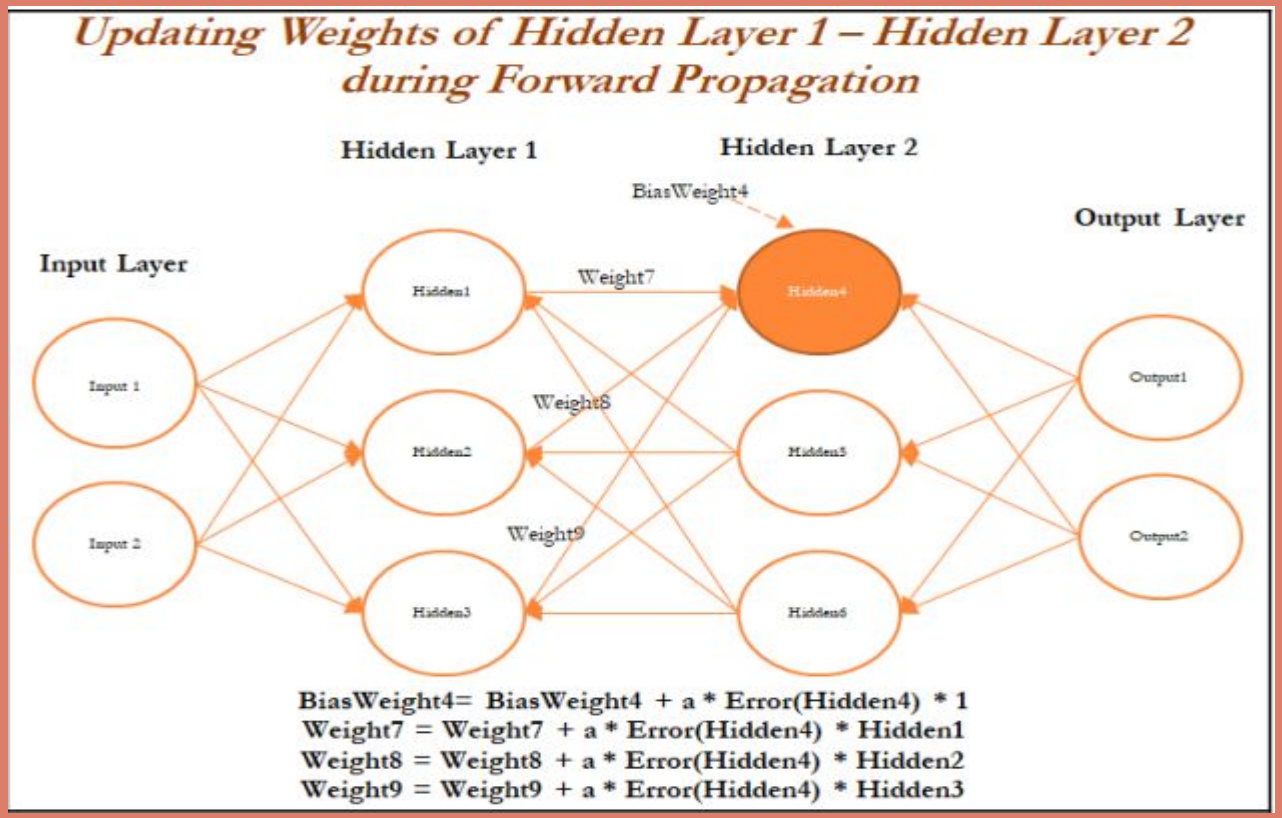
- Once all the neurons in hidden layer 1 are updated, weights between inputs and the hidden layer also need to be updated,
- In the following diagram, we will be updating the weights of both the inputs and also, at the same time, the neurons in hidden layer 1, as neurons in layer 1 utilize the weights from input only:



Artificial Neural Networks - ANN

Forward propagation and backpropagation

- Finally, in the following figure, layer 2 neurons are being updated in the forward propagation pass:



Artificial Neural Networks - ANN

Optimization of neural networks

- Various techniques have been used for optimizing the weights of neural networks:
  - Stochastic gradient descent (SGD)
  - Momentum
  - Nesterov accelerated gradient (NAG)
  - Adaptive gradient (Adagrad)
  - Adadelta
  - RMSprop
  - Adaptive moment estimation (Adam)
  - Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

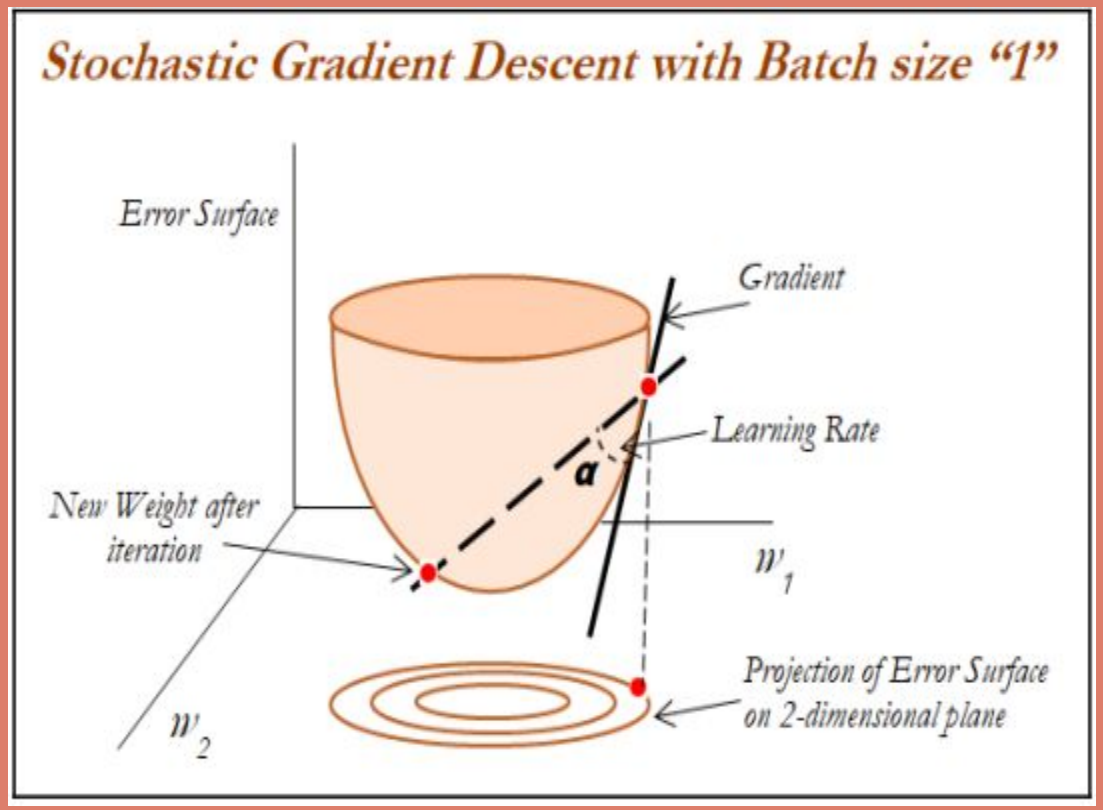
Artificial Neural Networks - ANN

Optimization of neural networks -Stochastic gradient descent - SGD

- Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameter $\theta \varepsilon R^d$ by updating the parameters in the opposite direction of the gradient of the objective function with regard to the parameters.
- The learning rate determines the size of the steps taken to reach the minimum
  - Batch gradient descent (all training observations utilized in each iteration)
  - SGD (one observation per iteration)
  - Mini batch gradient descent (size of about 50 training observations for each iteration)

Artificial Neural Networks - ANN

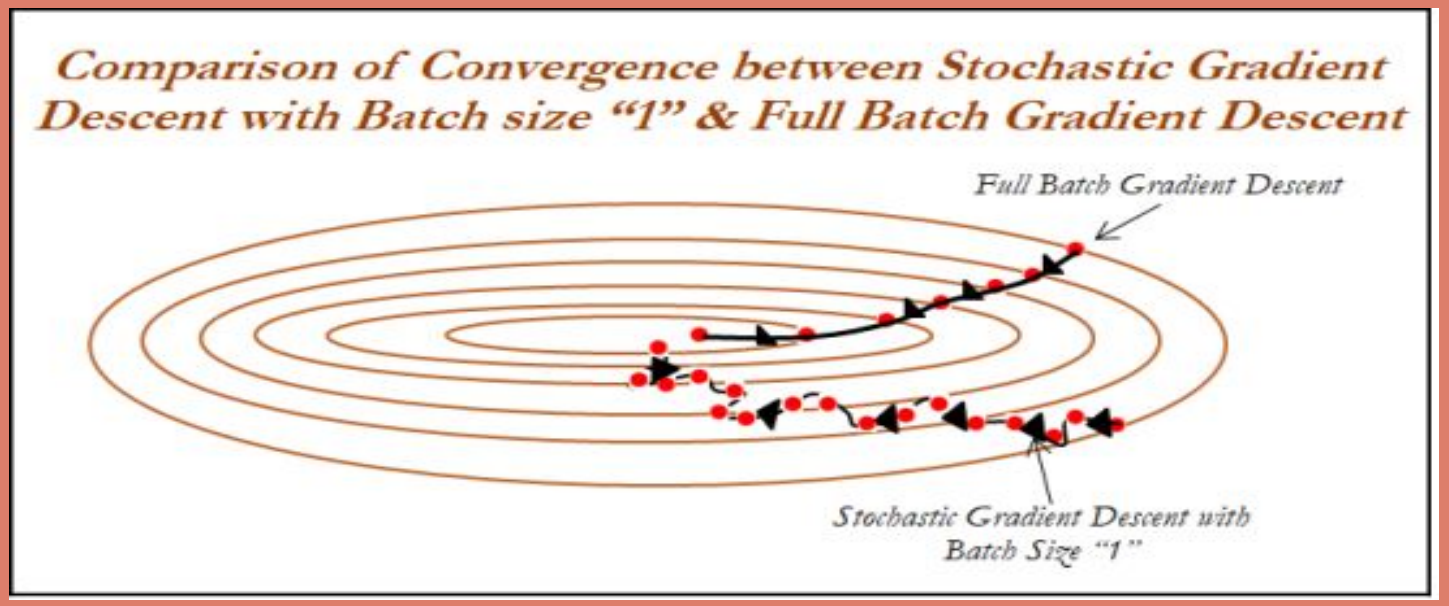
Optimization of neural networks -Stochastic gradient descent - SGD



Artificial Neural Networks - ANN

Optimization of neural networks -Stochastic gradient descent - SGD

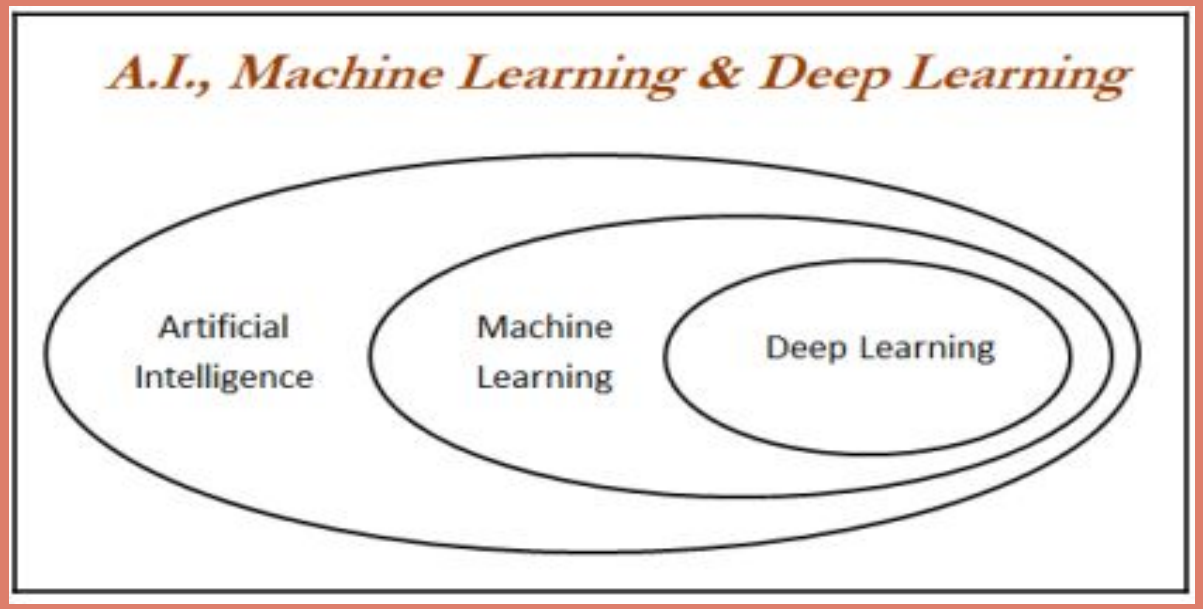
- In the following image 2D projection convergence characteristics of both full batch and stochastic gradient descent with batch size 1 has been compared.
- If we see, full batch updates, are more smoother due to the consideration of all the observations.
- Whereas, SGD have wiggly convergence characteristics due to the reason of using 1 observation for each update:



Artificial Neural Networks - ANN

Introduction to deep learning

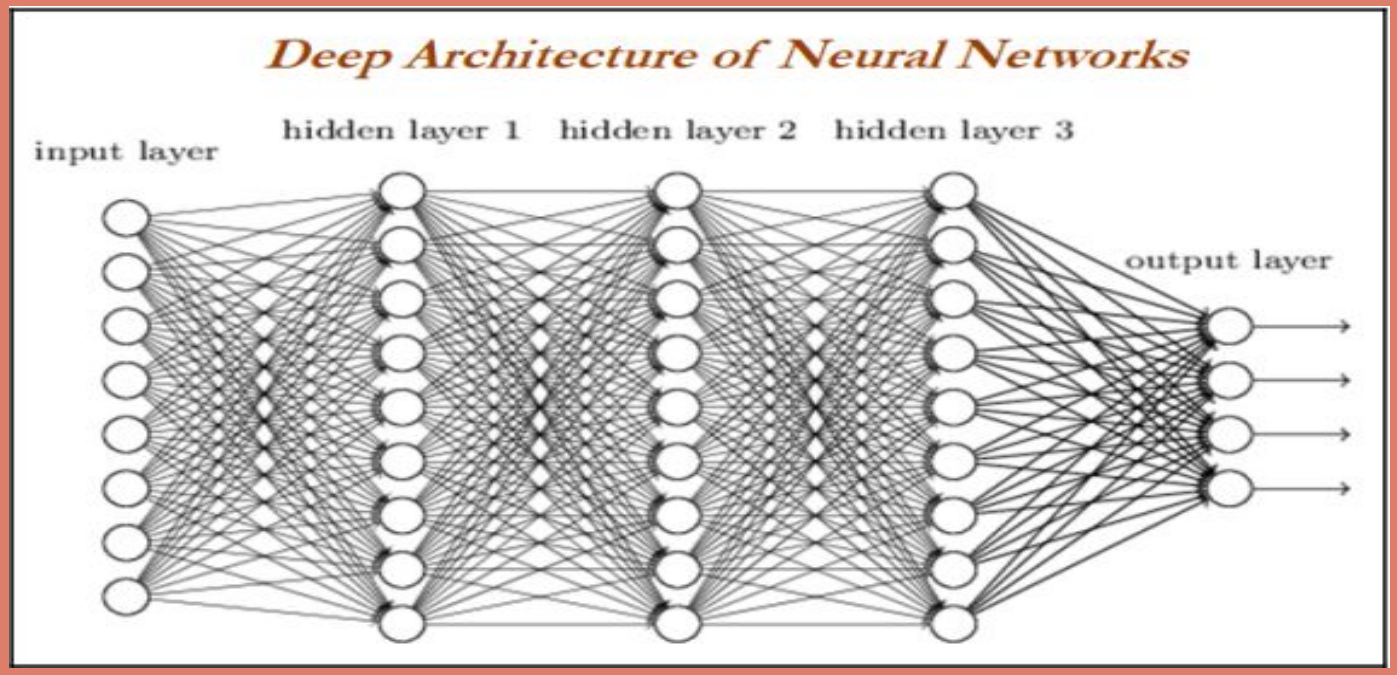
Deep learning is a class of machine learning algorithms which utilizes neural networks for building models to solve both supervised and unsupervised problems on structured and unstructured datasets such as images, videos, NLP, voice processing, and so on:



Artificial Neural Networks - ANN

Introduction to deep learning

- Deep neural network/deep architecture consists of multiple hidden layers of units between input and output layers.
- Each layer is fully connected with the subsequent layer.
- The output of each artificial neuron in a layer is an input to every artificial neuron in the next layer towards the output:

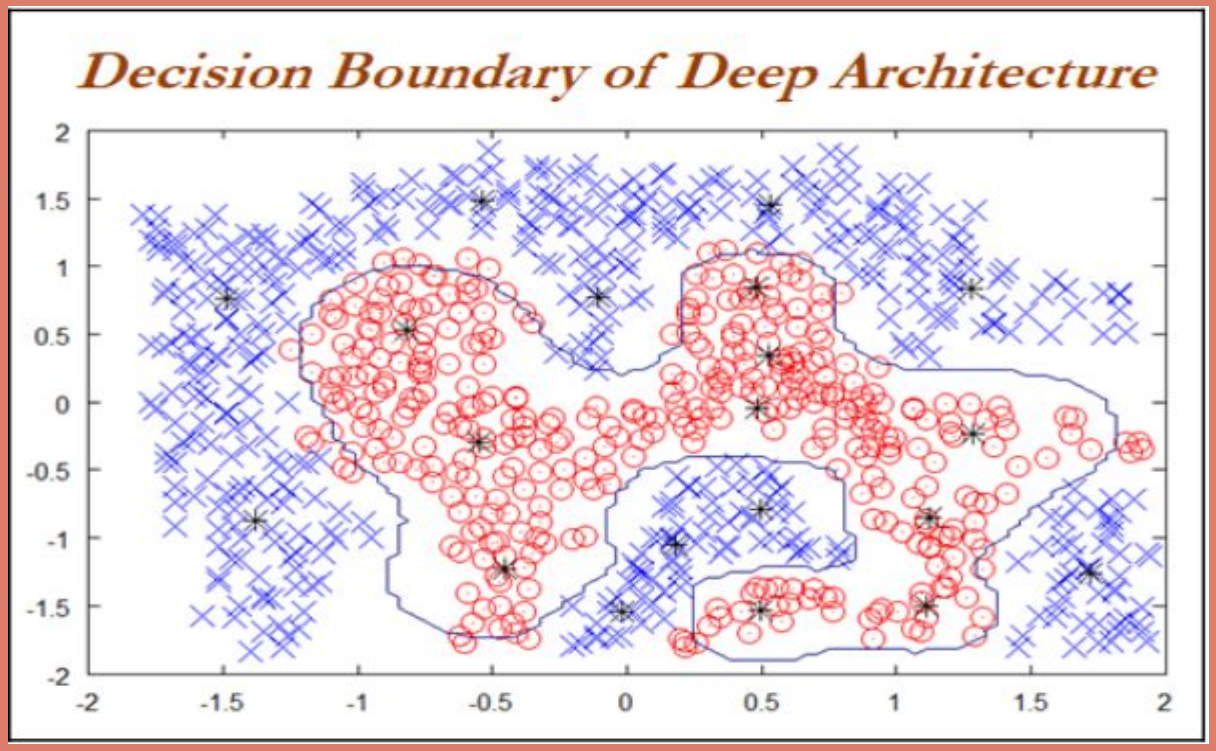


Artificial Neural Networks - ANN

Introduction to deep learning

With the more number of hidden layers are being added to the neural network, more complex decision boundaries are being created to classify different categories.

Example of complex decision boundary can be seen in the following graph:



Artificial Neural Networks - ANN

Solving methodology

- Backpropagation is used to solve deep layers by calculating the error of the network at output units and propagate back through layers to update the weights to reduce error terms.
- Thumb rules in designing deep neural networks:
  - All hidden layers should have the same number of neurons per layer
  - Typically, two hidden layers are good enough to solve the majority of problems
  - Using scaling/batch normalization (mean 0, variance 1) for all input variables after each layer improves convergence effectiveness
  - Reduction in step size after each iteration improves convergence, in addition to the use of momentum and dropout
-

Artificial Neural Networks - ANN

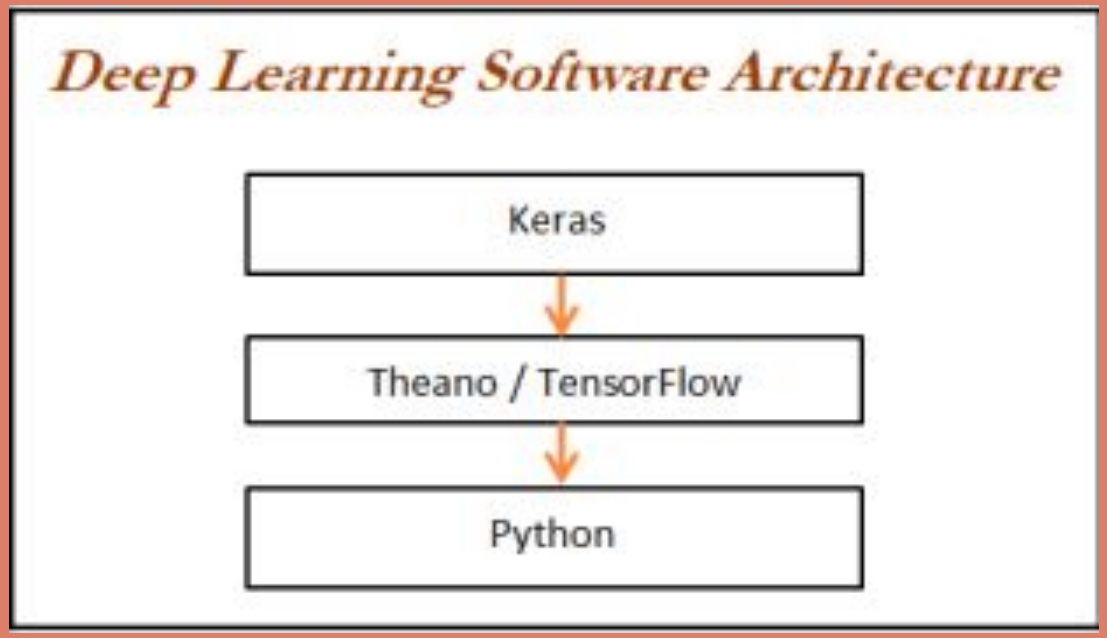
Deep learning software

- Deep learning software has evolved multi-fold in recent times.
- Different types of Deep Learning software are
  - Theano: Python-based deep learning library developed by the University of Montreal
  - TensorFlow: Google's deep learning library runs on top of Python/C++
  - Keras / Lasagne: Lightweight wrapper which sits on top of Theano/TensorFlow and enables faster model prototyping
  - Torch: Lua-based deep learning library with wide support for machine learning algorithms
  - Caffe: deep learning library primarily used for processing pictures

Artificial Neural Networks - ANN

Deep learning software

TensorFlow is recently picking up momentum among the deep learning community, as it is being backed up by Google and also has good visualization capabilities using TensorBoard:



Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

We are using the same data as we trained the model on previously using scikit-learn in order to perform apple-to-apple comparison between scikit-learn and the deep learning software Keras.

Data loading steps

```
>>> import numpy as np

>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> from sklearn.datasets import load_digits
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.metrics import
accuracy_score,classification_report
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

Keras Library modules

```
>>> from keras.models import Sequential
>>> from keras.layers.core import Dense, Dropout, Activation
>>> from keras.optimizers import Adadelta,Adam,RMSprop
>>> from keras.utils import np_utils
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

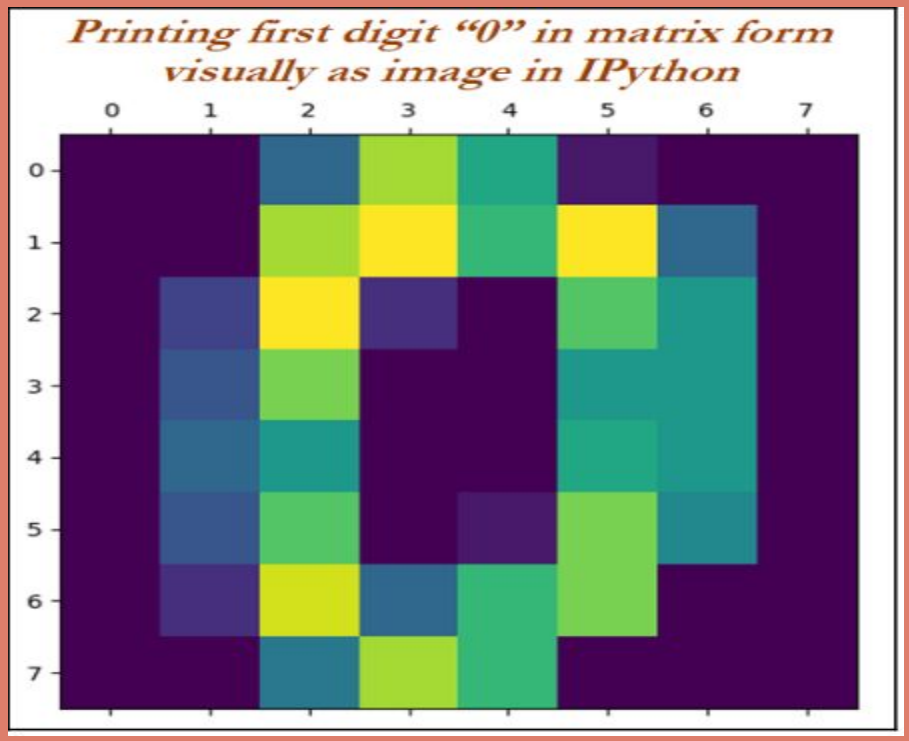
The following code loads the digit data from scikit-learn datasets. A quick piece of code to check the shape of the data, as data embedded in numpy arrays itself, hence we do not need to change it into any other format, as deep learning models get trained on numpy arrays:

```
>>> digits = load_digits()
>>> X = digits.data
>>> y = digits.target
>>> print (X.shape)
>>> print (y.shape)
>>> print ("\nPrinting first digit")
>>> plt.matshow(digits.images[0])
>>> plt.show()
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

The previous code prints the first digit in matrix form. It appears that the following digit looks like a 0:



Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

We are performing the standardizing of data with the following code to demean the series, followed by standard deviation to put all the 64 dimensions in a similar scale.

```
>>> x_vars_stdscle = StandardScaler().fit_transform(X)
```

The following section of the code splits the data into train and test based on a 70-30 split:

```
>>> x_train,x_test,y_train,y_test =
train_test_split(x_vars_stdscle,y,train_size = 0.7,random_state=42)
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

We have used nb_classes as 10, due to the reason that the digits range from 0-9; batch_size as 128, which means for each batch, we utilize 128 observations to update the weights; and finally, we have used nb_epochs as 200, which means the number of epochs the model needs to be trained is 200

```
# Defining hyper parameters
>>> np.random.seed(1337)
>>> nb_classes = 10
>>> batch_size = 128
>>> nb_epochs = 200
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

The following code actually creates the n-dimensional vector for multiclass values based on the nb_classes value. Here, we will get the dimension as 10 for all train observations for training using the softmax classifier:

```
>>> Y_train = np_utils.to_categorical(y_train, nb_classes)
```

The core model building code, which looks like Lego blocks, is shown as follows. Here we, initiate the model as sequential rather than parallel and so on:

```
#Deep Layer Model building in Keras
>>> model = Sequential()
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

In the first layer, we are using 100 neurons with input shape as 64 columns (as the number of columns in X is 64), followed by relu activation functions with dropout value as 0.5

```
>>> model.add(Dense(100,input_shape= (64,)))
>>> model.add(Activation('relu'))
>>> model.add(Dropout(0.5))
```

In the second layer, we are using 50 neurons (to compare the results obtained using the scikit-learn methodology, we have used a similar architecture):

```
>>> model.add(Dense(50))
>>> model.add(Activation('relu'))
>>> model.add(Dropout(0.5))
```

In the output layer, the number of classes needs to be used with the softmax classifier:

```
>>> model.add(Dense(nb_classes))
>>> model.add(Activation('softmax'))
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

Here, we are compiling with categorical_crossentropy, as the output is multiclass; whereas, if we want to use binary class, we need to use binary_crossentropy instead:

```
>>> model.compile(loss='categorical_crossentropy', optimizer='adam')
```

The model is being trained in the following step with all the given batch sizes and number of epochs:

```
#Model training
>>> model.fit(x_train, Y_train, batch_size=batch_size,
nb_epoch=nb_epochs,verbose=1)
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

```
#Model Prediction
>>> y_train_predclass =
model.predict_classes(x_train,batch_size=batch_size)
>>> y_test_predclass = model.predict_classes(x_test,batch_size=batch_size)
>>> print ("\n\nDeep Neural Network - Train accuracy:"),
(round(accuracy_score(y_train,y_train_predclass),3))
>>> print ("\nDeep Neural Network - Train Classification Report")
>>> print classification_report(y_train,y_train_predclass)
>>> print ("\nDeep Neural Network - Train Confusion Matrix\n")
>>> print (pd.crosstab(y_train,y_train_predclass,rownames =
["Actuall"],colnames = ["Predicted"]) )
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

```
Deep Neural Network  - Train accuracy: 1.0

Deep Neural Network  - Train Classification Report
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       125
          1       1.00      1.00      1.00       132
          2       1.00      1.00      1.00       130
          3       1.00      1.00      1.00       129
          4       1.00      1.00      1.00       121
          5       1.00      1.00      1.00       116
          6       1.00      1.00      1.00       128
          7       1.00      1.00      1.00       124
          8       1.00      1.00      1.00       131
          9       1.00      1.00      1.00       121

avg / total       1.00      1.00      1.00      1257


Deep Neural Network - Train Confusion Matrix

Predicted    0    1    2    3    4    5    6    7    8    9
Actuall
0          125    0    0    0    0    0    0    0    0    0
1            0  132    0    0    0    0    0    0    0    0
2            0    0  130    0    0    0    0    0    0    0
3            0    0    0  129    0    0    0    0    0    0
4            0    0    0    0  121    0    0    0    0    0
5            0    0    0    0    0  116    0    0    0    0
6            0    0    0    0    0    0  128    0    0    0
7            0    0    0    0    0    0    0  124    0    0
8            0    0    0    0    0    0    0    0  131    0
9            0    0    0    0    0    0    0    0    0  121
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

Testing

```
>>> print ("\nDeep Neural Network - Test
accuracy:"),(round(accuracy_score(y_test, y_test_predclass),3))
>>> print ("\nDeep Neural Network - Test Classification Report")
>>> print (classification_report(y_test,y_test_predclass))
>>> print ("\nDeep Neural Network - Test Confusion Matrix\n")
>>> print (pd.crosstab(y_test,y_test_predclass,rownames =
["Actuall"],colnames = ["Predicted"]) )
```

Artificial Neural Networks - ANN

Deep learning software - Deep neural network classifier applied on handwritten digits using Keras

```
Deep Neural Network  - Test accuracy: 0.976

Deep Neural Network  - Test Classification Report
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        53
          1       0.98      0.98      0.98        50
          2       0.94      0.98      0.96        47
          3       0.98      0.94      0.96        54
          4       0.98      1.00      0.99        60
          5       0.97      0.97      0.97        66
          6       0.98      0.98      0.98        53
          7       0.98      0.98      0.98        55
          8       0.98      0.95      0.96        43
          9       0.97      0.97      0.97        59

avg / total       0.98      0.98      0.98       540


Deep Neural Network - Test Confusion Matrix

Predicted   0   1   2   3   4   5   6   7   8   9
Actuall
0          53   0   0   0   0   0   0   0   0   0
1           0  49   1   0   0   0   0   0   0   0
2           0   0  46   1   0   0   0   0   0   0
3           0   0   2  51   0   1   0   0   0   0
4           0   0   0   0  60   0   0   0   0   0
5           0   0   0   0   0  64   1   0   0   1
6           0   0   0   0   1   0  52   0   0   0
7           0   0   0   0   0   0   0  54   0   1
8           0   1   0   0   0   1   0   0  41   0
9           0   0   0   0   0   0   0   1   1  57
```

Unit V

K-means Clustering, Principal Component Analysis and Singular Value Decomposition

Dr.S.Veena,Associate Professor/CSE

Unit V

K-means Clustering

- Introduction to K-means Clustering
- K-means working methodology from first principles
- Optimal number of clusters and cluster evaluation
- The elbow method
- K-means clustering with the iris data example

Principal Component Analysis(PCA)

- Introduction to PCA
- PCA working methodology from first principles
- PCA applied on handwritten digits using scikit-learn

Singular Value Decomposition(SVD)

- Introduction to SVD
- SVD applied on handwritten digits using scikit-learn

K-means Clustering

Introduction to K-Means Clustering

- Clustering is the task of grouping observations in such a way that members of the same cluster are more similar to each other and members of different clusters are very different from each other.
- Examples
  - In anti-money laundering measures, suspicious activities and individuals can be identified using anomaly detection
  - In biology, clustering is used to find groups of genes with similar expression
  - patterns
  - In marketing analytics, clustering is used to find segments of similar customers so that different marketing strategies can be applied to different customer segments accordingly
-

K-means Clustering

Introduction to K-Means Clustering

- K-means clustering algorithm
  - an iterative process of moving the centers of clusters or centroids to the mean position of their constituent points
  - reassigning instances to their closest clusters iteratively until there is no significant change in the number of cluster centers possible or number of iterations reached.

- The cost function of k-means is determined by the Euclidean distance (square-norm) between the observations belonging to that cluster with its respective centroid value.

K-means Clustering

Introduction to K-Means Clustering

- An intuitive way to understand the equation is, if there is only one cluster (k=1), then the distances between all the observations are compared with its single mean.
- if number of clusters increases to 2 (k= 2), then two-means are calculated and a few of the observations are assigned to cluster 1 and other observations are assigned to cluster two-based on proximity.
- Subsequently, distances are calculated in cost functions by applying the same distance measure, but separately to their cluster centers:

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} \| x_i - \mu_k \|^2$$

K-means Clustering

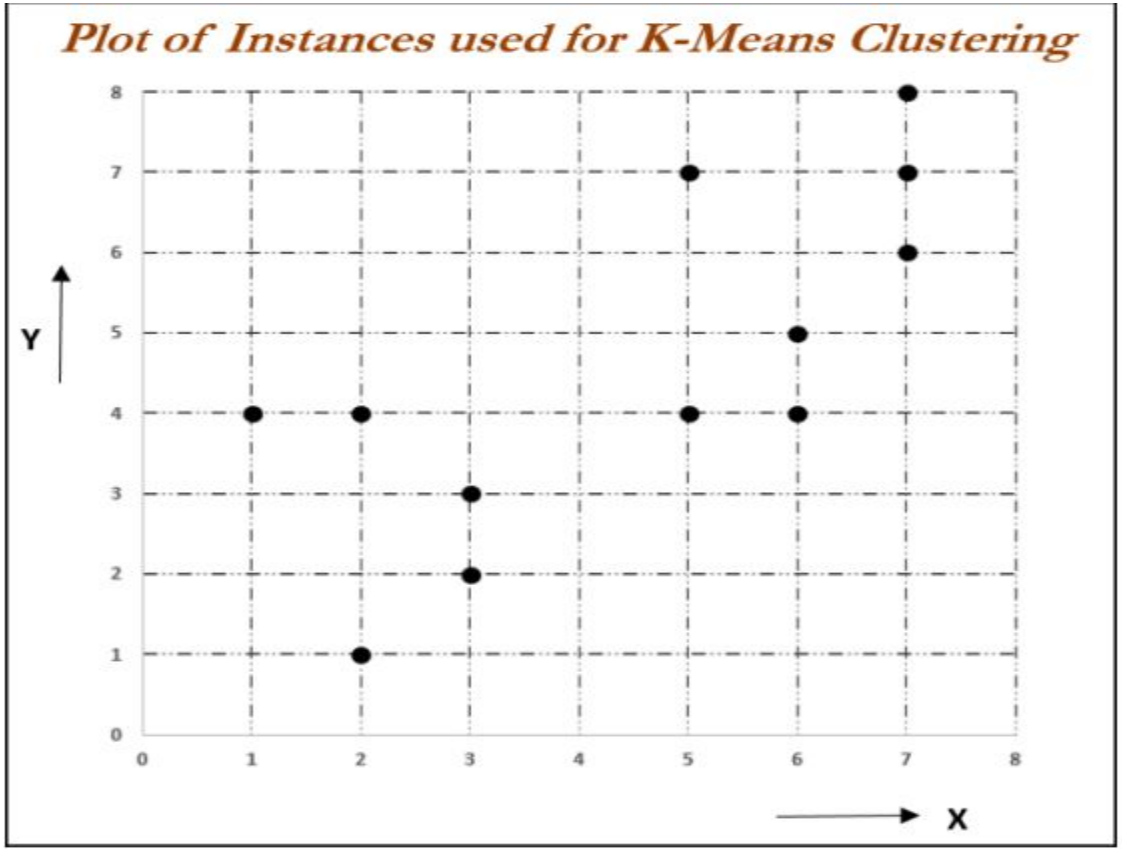
K-means working methodology from first principles

The k-means working methodology is illustrated in the following example in which 12 instances are considered with their X and Y values. The task is to determine the optimal clusters out of the data.

Instance	X	Y
1	7	8
2	2	4
3	6	4
4	3	2
5	6	5
6	5	7
7	3	3
8	1	4
9	5	4
10	7	7
11	7	6
12	2	1

K-means Clustering

K-means working methodology from first principles



K-means Clustering

K-means working methodology from first principles

Iteration 1:

- Let us assume two centers from two instances out of all the 12 instances.
- chosen instance 1 (X = 7, Y = 8) and instance 8 (X = 1, Y = 4),
- For each instance, calculate its Euclidean distances with respect to
- both centroids and assign it to the nearest cluster center.

The Euclidean distance between two points A (X1, Y1) and B (X2, Y2) is shown as follows:

$$Euclidean\ Distance\ between\ A\ \&\ B = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}$$

K-means Clustering

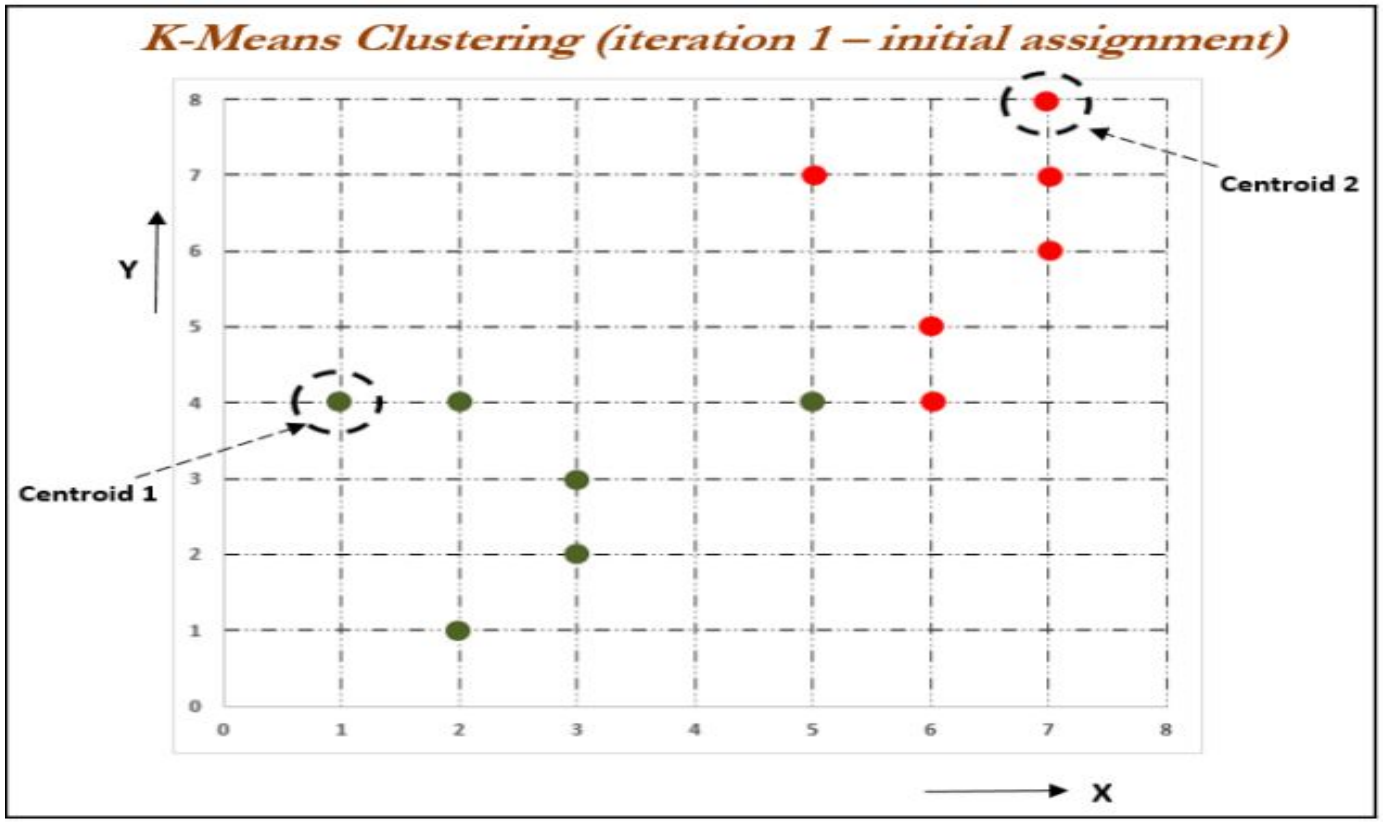
Instance	X	Y	Centroid 1 distance	Centroid 2 distance	Assigned cluster
1	7	8	7.21	0.00	C2
2	2	4	1.00	6.40	C1
3	6	4	5.00	4.12	C2
4	3	2	2.83	7.21	C1
5	6	5	5.10	3.16	C2
6	5	7	5.00	2.24	C2
7	3	3	2.24	6.40	C1
8	1	4	0.00	7.21	C1
9	5	4	4.00	4.47	C1
10	7	7	6.71	1.00	C2
11	7	6	6.32	2.00	C2
12	2	1	3.16	8.60	C1
Centroid 1	1	4			
Centroid 2	7	8			

$$Distance\ w.r.to\ Centroid\ 1\ for\ Instance\ 6 = \sqrt{(5-1)^2 + (7-4)^2} = 5.00$$

$$Distance\ w.r.to\ Centroid\ 2\ for\ Instance\ 6 = \sqrt{(5-7)^2 + (7-8)^2} = 2.24$$

K-means Clustering

Assignment of instances to both centroids



K-means Clustering

Iteration 2: In this iteration, new centroids are calculated from the assigned instances for that cluster or centroid. New centroids are calculated based on the simple average of the assigned points.

Instance	X	Y	Assigned cluster
1	7	8	C2
2	2	4	C1
3	6	4	C2
4	3	2	C1
5	6	5	C2
6	5	7	C2
7	3	3	C1
8	1	4	C1
9	5	4	C1
10	7	7	C2
11	7	6	C2
12	2	1	C1
Centroid 1	2.67	3	
Centroid 2	6.33	6.17	

K-means Clustering

$$Centroid\ 1\ coordinates = Average\ coordinates\ (Instance\ 2,4,7,8,9,\&12)$$

$$Centroid\ 2\ coordinates = Average\ coordinates\ (Instance\ 1,3,5,6,10\ \&11)$$

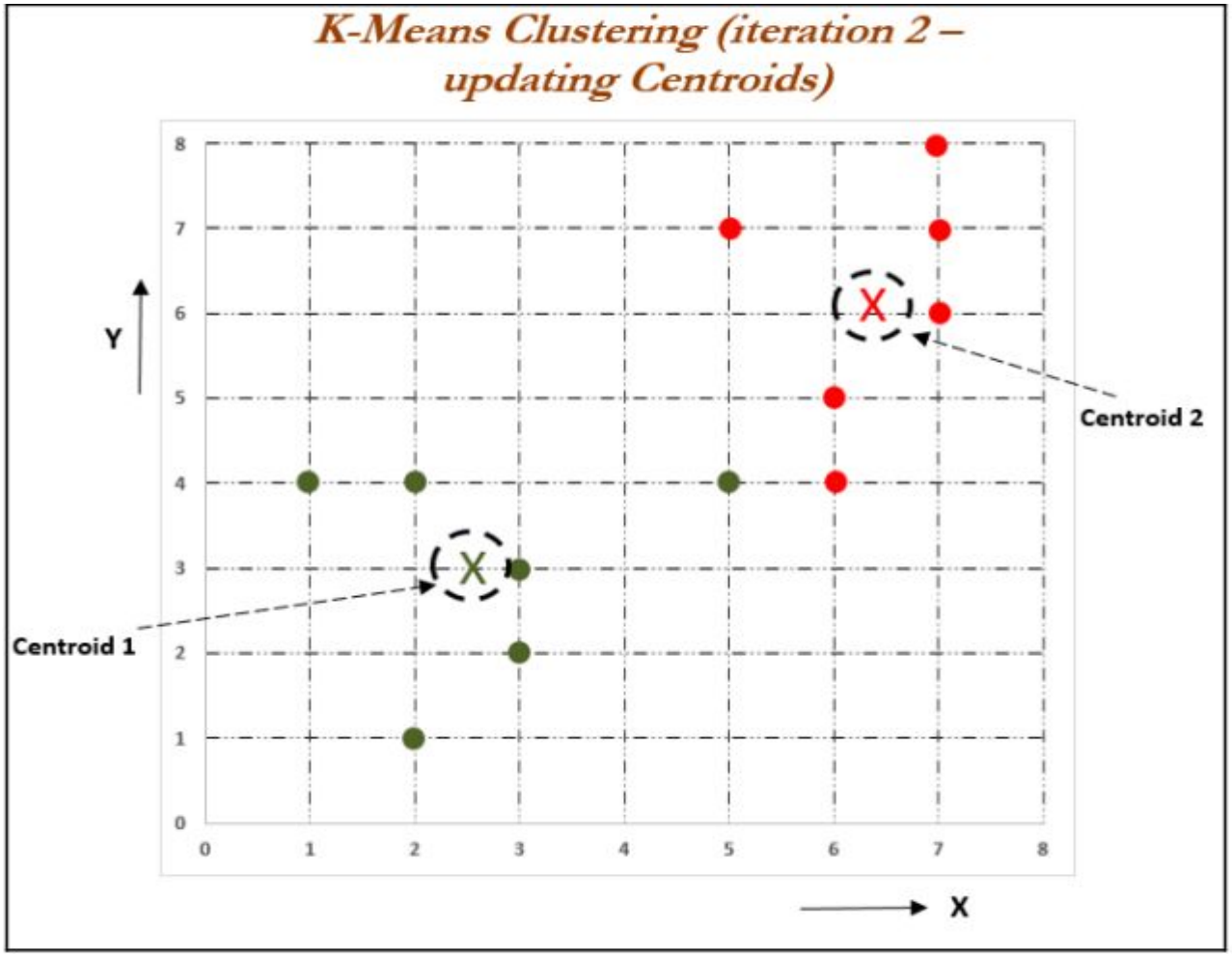
$$Centroid\ 1\ X = \frac{(2+3+3+1+5+2)}{6} = 2.67$$

$$Centroid\ 1\ Y = \frac{(4+2+3+4+4+1)}{6} = 3.0$$

$$Centroid\ 2\ X = \frac{(7+6+6+5+7+7)}{6} = 6.33$$

$$Centroid\ 2\ Y = \frac{(8+4+5+7+7+6)}{6} = 6.17$$

K-means Clustering



K-means Clustering

Iteration 3: In this iteration, new assignments are calculated based on the Euclidean distance between instances and new centroids. In the event of any changes, new centroids will be calculated iteratively until no changes in assignments are possible or the number of iterations is reached.

Instance	X	Y	Centroid 1 distance	Centroid 2 distance	Previously assigned cluster	Newly assigned cluster	Changed?
1	7	8	6.61	1.95	C2	C2	No
2	2	4	1.20	4.84	C1	C1	No
3	6	4	3.48	2.19	C2	C2	No
4	3	2	1.05	5.34	C1	C1	No
5	6	5	3.88	1.22	C2	C2	No
6	5	7	4.63	1.57	C2	C2	No
7	3	3	0.33	4.60	C1	C1	No
8	1	4	1.95	5.75	C1	C1	No
9	5	4	2.54	2.55	C1	C1	No
10	7	7	5.89	1.07	C2	C2	No
11	7	6	5.27	0.69	C2	C2	No
12	2	1	2.11	6.74	C1	C1	No
Centroid 1	2.67	3					
Centroid 2	6.33	6.17					

K-means Clustering

Optimal number of clusters and cluster evaluation

- Though selecting number of clusters is more of an art than science, optimal number of clusters are chosen where, there will not be much marginal increase in explanation ability by increasing number of clusters are possible.
- In practical applications, usually business should be able to provide what would be approximate number of clusters they are looking for.

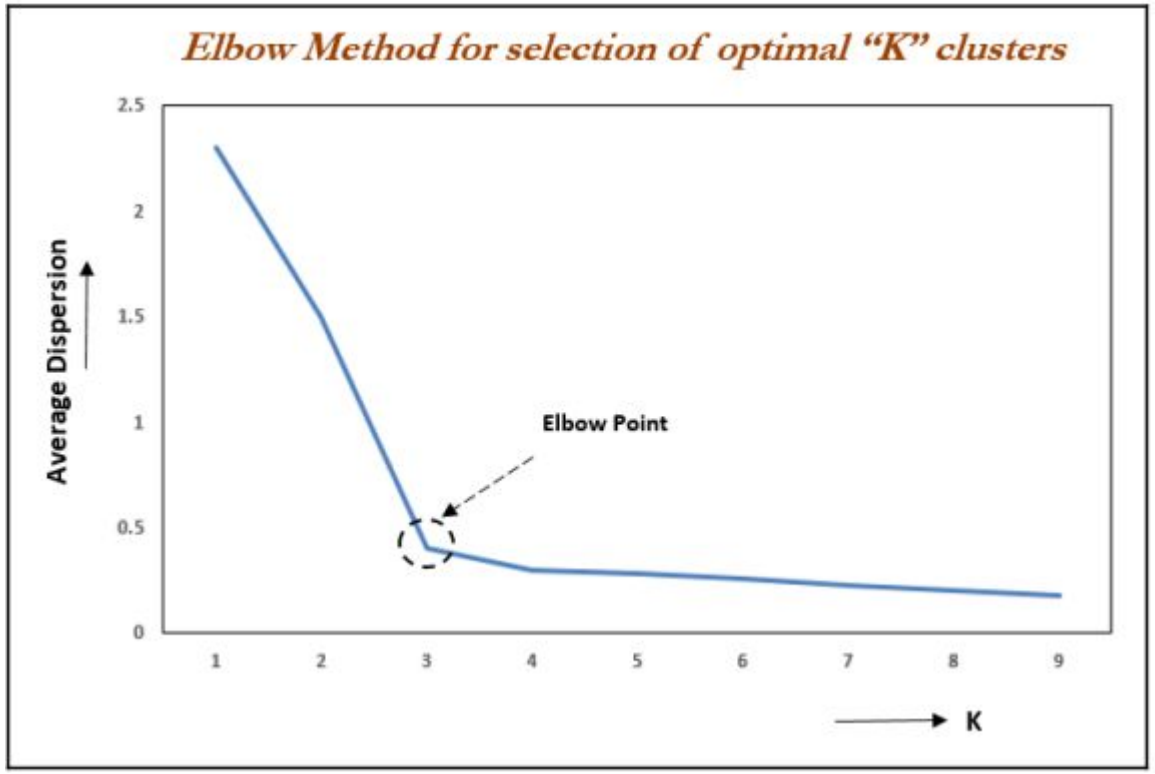
K-means Clustering

The elbow method

- The elbow method is used to determine the optimal number of clusters in k-means clustering.
- The elbow method plots the value of the cost function produced by different
- values of k.
- if k increases, average distortion will decrease, each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids.
- However, the improvements in average distortion will decline as k increases.
- The value of k at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

K-means Clustering

The elbow method



K-means Clustering

Evaluation of clusters with silhouette coefficient

- the silhouette coefficient is a measure of the compactness and separation of the clusters.
- Higher values represent a better quality of cluster.
- The silhouette coefficient is higher for compact clusters that are well separated and lower for overlapping clusters.
- Silhouette coefficient values do change from -1 to +1, and the higher the value is, the better.
- The silhouette coefficient is calculated per instance. For a set of instances, it is calculated as the mean of the individual sample's scores.

$$s = \frac{ba}{\max(a, b)}$$

a is the mean distance between the instances in the cluster,

b is the mean distance between the instance and the instances in the next closest cluster.

K-means clustering with the iris data example

Data set : http://archive.ics.uci.edu/ml/datasets/Iris.

The iris data has three types of flowers: setosa, versicolor, and virginica and their respective measurements of sepal length, sepal width, petal length, and petal width.

The task is to group the flowers based on their measurements.

K-means clustering with the iris data example

K-means algorithm from scikit-learn has been utilized in the following example

 # K-means clustering

```
>>> import numpy as np

>>> import pandas as pd

>>> import matplotlib.pyplot as plt

>>> from scipy.spatial.distance import cdist, pdist

>>> from sklearn.cluster import KMeans
>>> from sklearn.metrics import silhouette_score

>>> iris = pd.read_csv("iris.csv"

>>> print (iris.head())
```

K-means clustering with the iris data example

Following code is used to separate class variable as dependent variable for creating colors in plot and unsupervised learning algorithm applied on given x variables without any target variable does present:

```
>>> x_iris = iris.drop(['class'],axis=1)

>>> y_iris = iris["class"]
```

sepal_length	sepal_width	petal_length	petal_width	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa

K-means clustering with the iris data example

three clusters have been used

The maximum number of iterations chosen here is 300

```
>>> k_means_fit = KMeans(n_clusters=3,max_iter=300)

>>> k_means_fit.fit(x_iris)

>>> print ("\nK-Means Clustering - Confusion

Matrix\n\n",pd.crosstab(y_iris, k_means_fit.labels_,rownames = ["Actuall"],
colnames = ["Predicted"]) )

>>> print ("\nSilhouette-score: %0.3f" % silhouette_score(x_iris, k_means_fit.labels_,
metric='euclidean')
```

K-means clustering with the iris data example

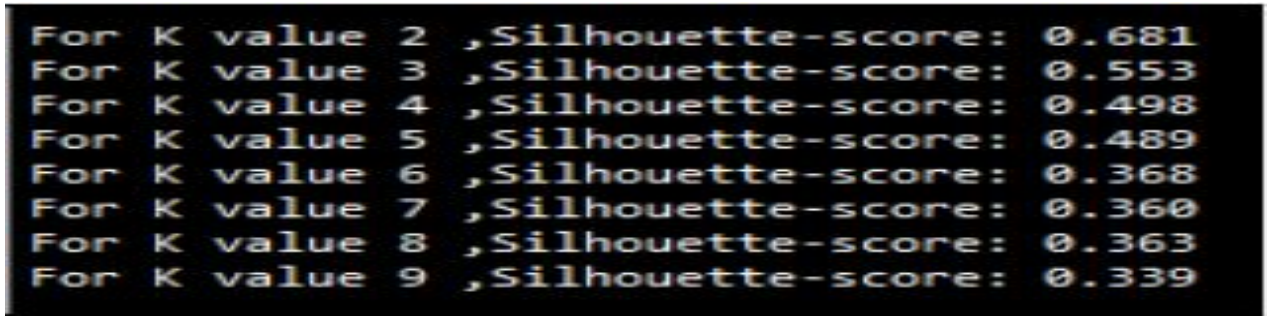
```
K-Means Clustering - Confusion Matrix

 Predicted            0    1    2
Actuall
Iris-setosa         50    0    0
Iris-versicolor      0   48    2
Iris-virginica       0   14   36

Silhouette-score: 0.553
```

From the previous confusion matrix, we can see that all the setosa flowers are clustered correctly, whereas 2 out of 50 versicolor, and 14 out of 50 virginica flowers are incorrectly classified.

K-means clustering with the iris data example

To perform sensitivity analysis to check how many number of clusters does actually provide better explanation of segments:

```
>>> for k in range(2,10):
...     k_means_fitk = KMeans(n_clusters=k,max_iter=300)
...     k_means_fitk.fit(x_iris)
...     print ("For K value",k,",Silhouette-score: %0.3f" %
silhouette_score(x_iris, k_means_fitk.labels_,
metric='euclidean'))
```



The silhouette coefficient values in the preceding results shows that K value 2 and K value 3 have better scores than all the other values.

K-means clustering with the iris data example

we also need to see the average within cluster variation value and elbow plot before concluding the optimal K value.

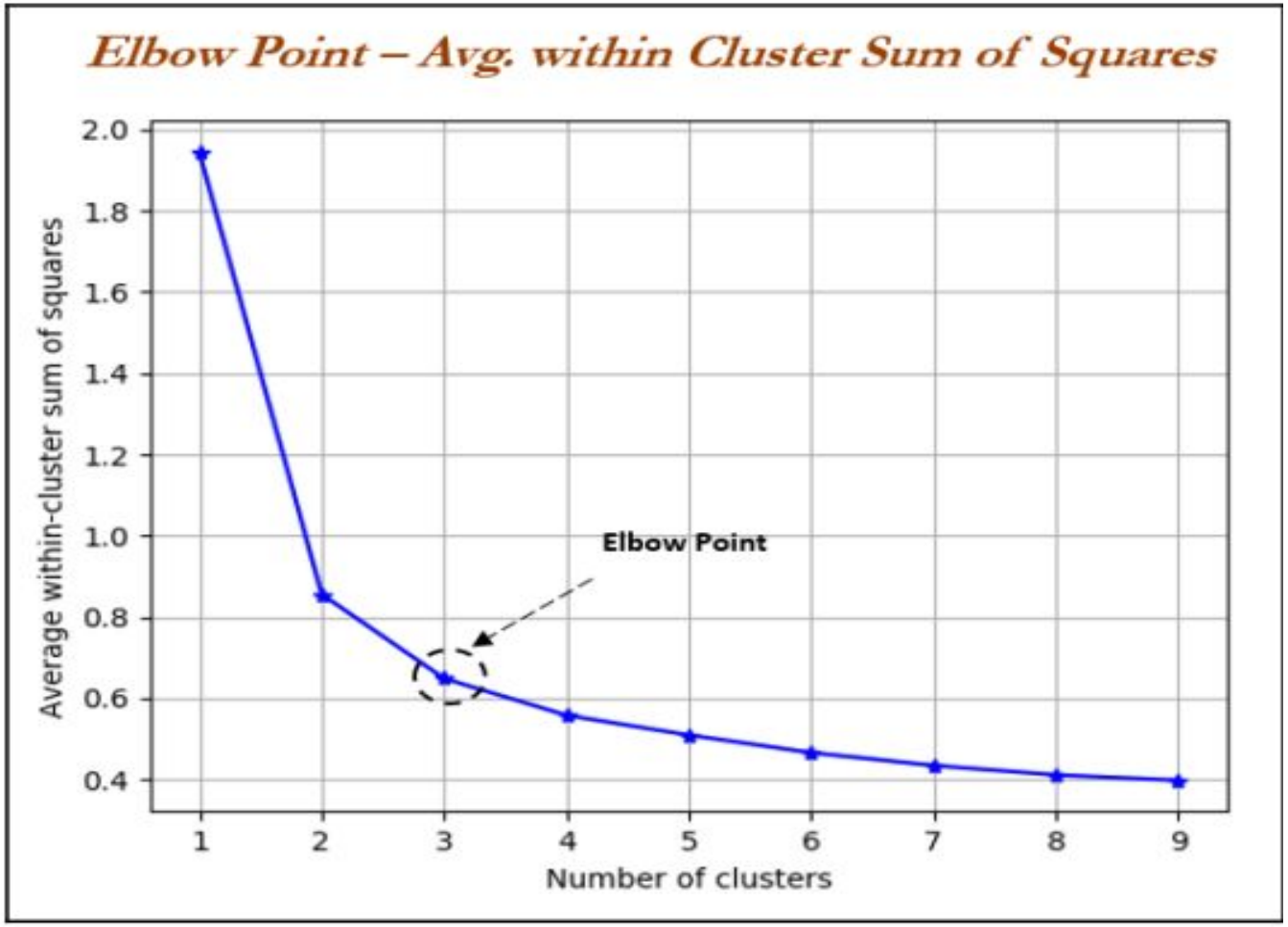
 # Avg. within-cluster sum of squares

```
>>> K = range(1,10)

>>> KM = [KMeans(n_clusters=k).fit(x_iris) for k in K]
>>> centroids = [k.cluster_centers_ for k in KM]

>>> D_k = [cdist(x_iris, centrds, 'euclidean') for centrds in centroids]
>>> cIdx = [np.argmin(D,axis=1) for D in D_k]
>>> dist = [np.min(D,axis=1) for D in D_k]

>>> avgWithinSS = [sum(d)/x_iris.shape[0] for d in dist
```

K-means clustering with the iris data example

```
# Total with-in sum of square
>>> wcss = [sum(d**2) for d in dist]
>>> tss = sum(pdist(x_iris)**2)/x_iris.shape[0]
>>> bss = tss-wcss
# elbow curve - Avg. within-cluster sum of squares
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(K, avgWithinSS, 'b*-')
>>> plt.grid(True)
>>> plt.xlabel('Number of clusters')
>>> plt.ylabel('Average within-cluster sum of squares')
```

K-means clustering with the iris data example

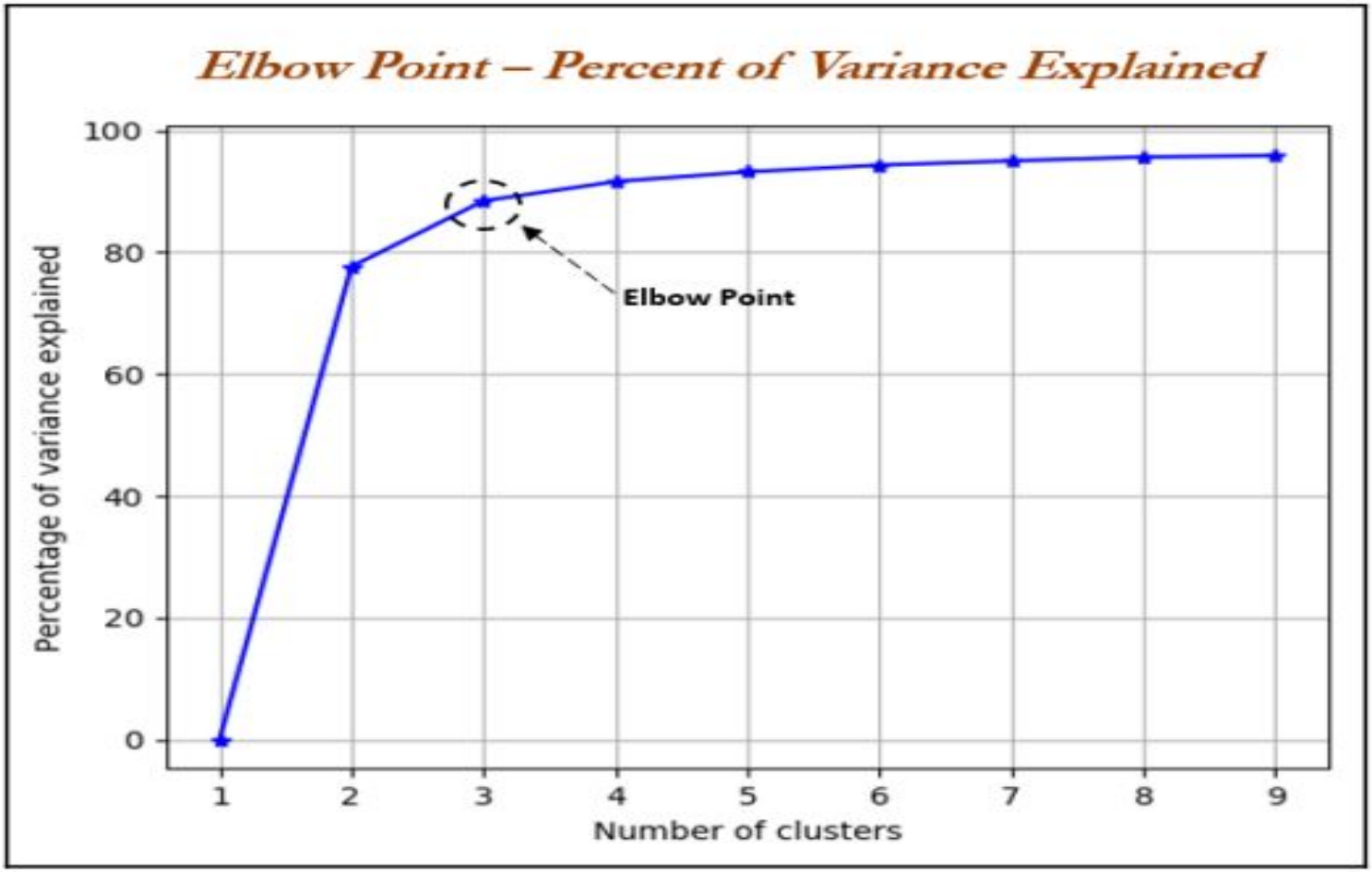


K-means clustering with the iris data example

From the elbow plot, it seems that at the value of three, the slope changes drastically.
Here,we can select the optimal k-value as three.

```
# elbow curve - percentage of variance explained
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(K, bss/tss*100, 'b*-')
>>> plt.grid(True)
>>> plt.xlabel('Number of clusters')
>>> plt.ylabel('Percentage of variance explained')
>>> plt.show()
```

K-means clustering with the iris data example

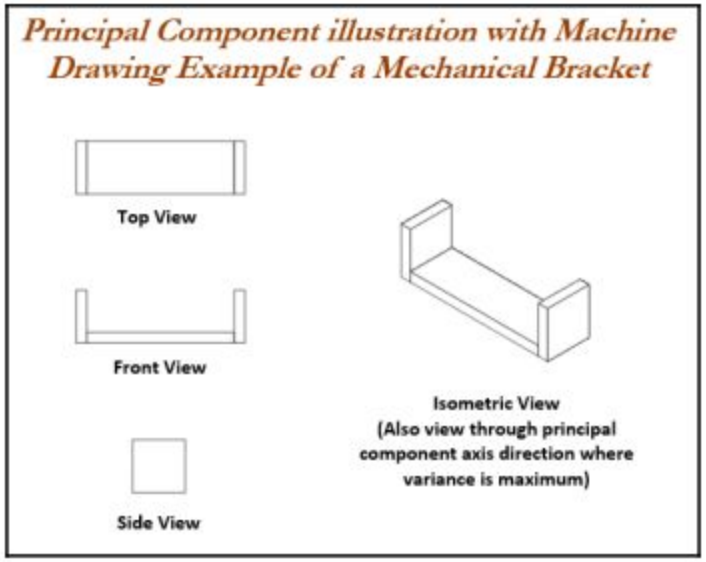


Principal component analysis - PCA

- **Principal component analysis** (**PCA**) is the dimensionality reduction technique which has so many utilities.
- PCA reduces the dimensions of a dataset by projecting the data onto a lower-dimensional subspace.
- For example, a 2D dataset could be reduced by projecting the points onto a line.
- Each instance in the dataset would then be represented by a single value, rather than a pair of values.
- In a similar way, a 3D dataset could be reduced to two dimensions by projecting variables onto a plane.

Principal component analysis - PCA

- PCA can easily be explained with the following diagram of a mechanical bracket which has been drawn in the machine drawing module of a mechanical engineering course.

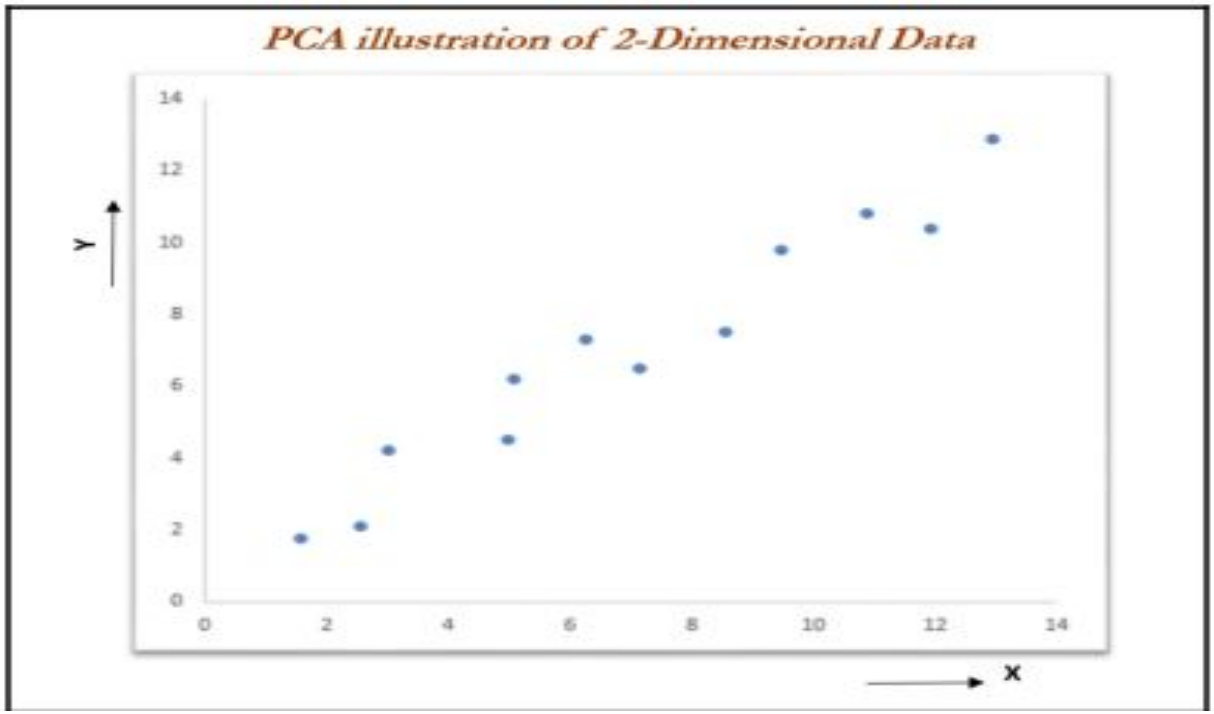


Principal component analysis - PCA

- The left- hand side of the diagram depicts the top view, front view, and side view of the component.
- However, on the right-hand side, an isometric view has been drawn, in which one single image has been used to visualize how the component looks.
- So, one can imagine that the left-hand images are the actual variables and the right-hand side is the first principal component, in which most variance has been captured.
- Finally, three images have been replaced by a single image by rotating the axis of direction. we replicate the same technique in PCA analysis

Principal component analysis - PCA

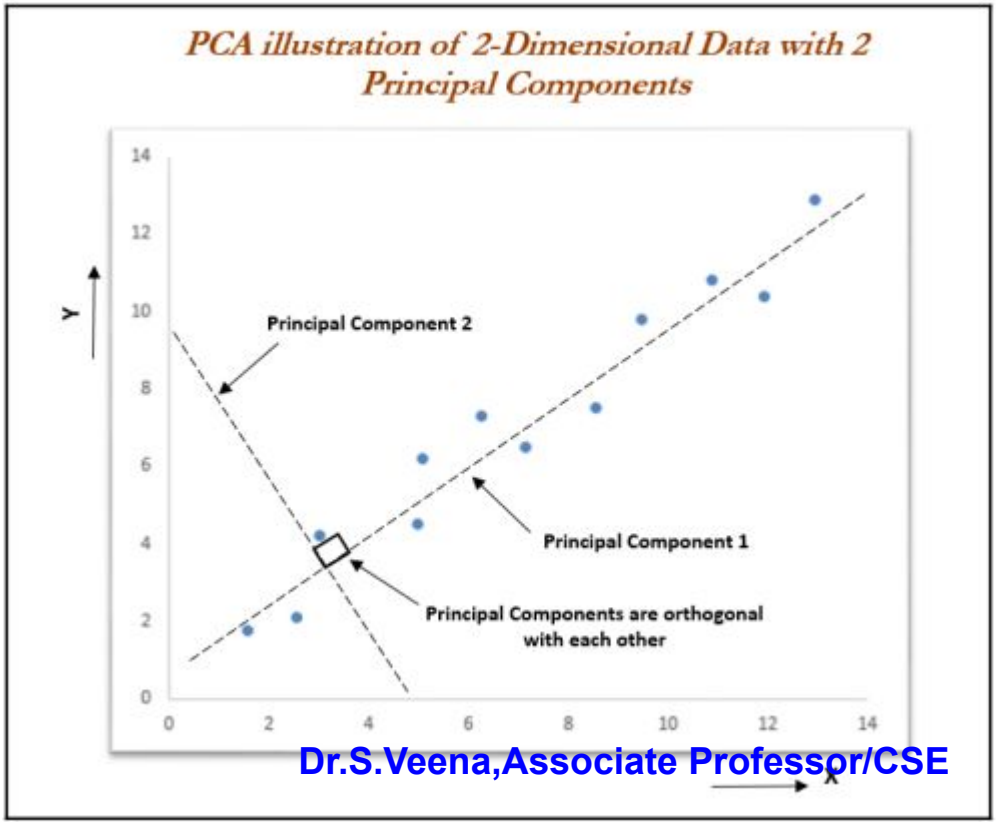
- Actual data has been shown in a 2D space, in which X and Y axis are used to plot the data.
- Principal components are the ones in which maximum variation of the data is captured.



Principal component analysis - PCA

Fitting the principal components.

- The first principal component covers the maximum variance in the data
- second principal component is orthogonal to the first principal component, as we know all principal components are orthogonal to each other.

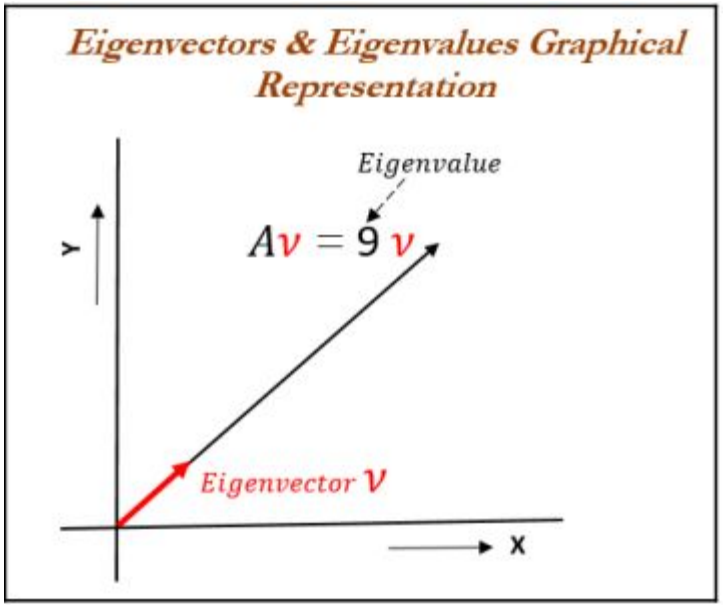


Principal component analysis - PCA

- We can represent the whole data with the first principal component itself.
- It is advantageous to represent the data with fewer dimensions, to save space and also to grab maximum variance in the data,which can be utilized for supervised learning in the next stage.
- This is the core advantage of computing principal components.
- Eigenvectors are the axes (directions) along which a linear transformation acts simply by stretching/compressing and/or flipping
- Eigenvalues give the factors by which the compression occurs. In another way, an eigenvector of a linear transformation is a nonzero vector whose direction does not change when that linear transformation is applied to it.

Principal component analysis - PCA

- More formally, A is a linear transformation from a vector space and is a nonzero vector, then eigenvector of A if is a scalar multiple of .
- The condition can be written as the following equation:



- In the preceding equation, is an eigenvector, A is a square matrix, and λ is a scalar called an eigenvalue.
- The direction of an eigenvector remains the same after it has been transformed by A; only its magnitude has changed, as indicated by the eigenvalue,

Principal component analysis - PCA

- The following example describes how to calculate eigenvectors and eigenvalues from the square matrix and its understanding.
- Note that eigenvectors and eigenvalues can be calculated only for square matrices (those with the same dimensions of rows and columns)

$$A = \begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix}$$

- product of A and any eigenvector of A must be equal to the eigenvector multiplied by the magnitude of eigenvalue:

$$(A - \lambda I)\vec{v} = 0$$

$$|A - \lambda * I| = \left| \begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

Principal component analysis - PCA

- A characteristic equation states that the determinant of the matrix, that is the difference between the data matrix and the product of the identity matrix and an eigenvalue is *0*

$$\left|\begin{bmatrix} 2-\lambda & -4 \\ 4 & -6-\lambda \end{bmatrix}\right| = (\lambda+2)(\lambda+2) = 0$$

- Both eigenvalues for the preceding matrix are equal to -2. We can use eigenvalues to substitute for eigenvectors in an equation:

$$A\vec{v} = \lambda\vec{v}$$

$$(A - \lambda I)\vec{v} = 0$$

$$\left(\begin{bmatrix} 2 & -4 \\ 4 & \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \end{bmatrix}\right)\vec{v} = \begin{bmatrix} 2-\lambda & -4 \\ & \end{bmatrix}\vec{v} =$$

$$|A - \lambda * I| = \left|\begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right| = 0$$

Principal component analysis - PCA

- Substituting the value of eigenvalue in the preceding equation, we will obtain the following formula:

$$\begin{bmatrix} 2-(-2) & -4 \\ 4 & -6-(-2) \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = \begin{bmatrix} 4 & -4 \\ 4 & -4 \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$$

- The preceding equation can be rewritten as a system of equations, as follows

$$4 * v_{1,1} - 4 * v_{1,2} = 0$$
$$4 * v_{1,1} - 4 * v_{1,2} = 0$$

- This equation indicates it can have multiple solutions of eigenvectors we can substitute with any values which hold the preceding equation for verification of equation. Here, we have used the vector *[1 1]* for verification, which seems to be proved

$$\begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Principal component analysis - PCA

- PCA needs unit eigenvectors to be used in calculations, hence we need to divide the same with the norm or we need to normalize the eigenvector. The 2-norm equation is shown as follows:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- The norm of the output vector is calculated as follows

$$\left\| \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\| = \sqrt{1^2 + 1^2} = \sqrt{2}$$

- The unit eigenvector is shown as follows:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} / \sqrt{2} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

Principal component analysis - PCA

PCA working methodology from first principles

- PCA working methodology is described in the following sample data, which has two dimensions for each instance or data point. The objective here is to reduce the 2D data into one dimension (also known as the **principal component**):

Instance	X	Y
1	0.72	0.13
2	0.18	0.23
3	2.50	2.30
4	0.45	0.16
5	0.04	0.44
6	0.13	0.24
7	0.30	0.03
8	2.65	2.10
9	0.91	0.91
10	0.46	0.32
Column mean	0.83	0.69

Principal component analysis - PCA

PCA working methodology from first principles

The first step, prior to proceeding with any analysis, is to subtract the mean from all the observations, which removes the scale factor of variables and makes them more uniform across dimensions.

X	Y
0.72 - 0.83 = -0.12	0.13 - 0.69 = - 0.55
0.18 - 0.83 = -0.65	0.23 - 0.69 = - 0.46
2.50 - 0.83 = 1.67	2.30 - 0.69 = 1.61
0.45 - 0.83 = -0.38	0.16 - 0.69 = - 0.52
0.04 - 0.83 = -0.80	0.44 - 0.69 = - 0.25
0.13 - 0.83 = -0.71	0.24 - 0.69 = - 0.45
0.30 - 0.83 = -0.53	0.03 - 0.69 = - 0.66
2.65 - 0.83 = 1.82	2.10 - 0.69 = 1.41
0.91 - 0.83 = 0.07	0.91 - 0.69 = 0.23
0.46 - 0.83 = -0.37	0.32 - 0.69 = -0.36

Principal component analysis - PCA

PCA working methodology from first principles

Principal components are calculated using two different techniques:

- Covariance matrix of the data
- Singular value decomposition

Covariance is a measure of how much two variables change together and it is a measure of the strength of the correlation between two sets of variables.

If the covariance of two variables is zero, we can conclude that there will not be any correlation between two sets of the variables. The formula for covariance is as follows

$$cov(x,y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

Principal component analysis - PCA

PCA working methodology from first principles

A sample covariance calculation is shown for X and Y variables in the following formulas. However, it is a 2 x 2 matrix of an entire covariance matrix (also, it is a square matrix).

$$cov(Y,Y) = \frac{((-0.55)^2 + (-0.46)^2 + (1.61)^2 + (-0.52)^2 + (-0.25)^2 + (-0.45)^2 + (-0.66)^2 + (1.41)^2 + (0.23)^2 + (-0.36)^2)}{10-1}$$

$$cov(Y,Y) = 0.697029$$

$$Covariance\ matrix = C = \begin{bmatrix} 0.91335 & 0.75969 \\ 0.75969 & 0.69702 \end{bmatrix}$$

Principal component analysis - PCA

PCA working methodology from first principles

we can calculate eigenvectors and eigenvalue

$$|A - \lambda * I| = \left| \begin{bmatrix} 0.91335 & 0.75969 \\ 0.75969 & 0.69702 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

By solving the preceding equation, we can obtain eigenvectors and eigenvalues, as follows:

$$Eigenvalues = \begin{bmatrix} 1.5725 & 0.0378 \end{bmatrix}$$

$$Unit\ Eigenvectors = \begin{bmatrix} 0.7553 & -0.6553 \\ 0.6553 & 0.7553 \end{bmatrix}$$

Principal component analysis - PCA

PCA working methodology from first principles

Python syntax:

```
>>> import numpy as np
>>> w, v = np.linalg.eig(np.array([[ 0.91335 ,0.75969 ],[0.75969,0.69702]]))
\>>> print ("\nEigen Values\n", w)
>>> print ("\nEigen Vectors\n", v)
```

```
Eigen Values
 [ 1.57253666  0.03783334]

Eigen Vectors
 [[ 0.75530088 -0.6553782 ]
 [ 0.6553782   0.75530088]]
```

Principal component analysis - PCA

PCA working methodology from first principles

Once we obtain the eigenvectors and eigenvalues, we can project data into principal components.

The first eigenvector has the greatest eigenvalue and is the first principal component, as we would like to reduce the original 2D data into 1D data.

$$\begin{bmatrix} -0.12 & -0.55 \\ -0.65 & -0.46 \\ 1.67 & 1.61 \\ -0.38 & -0.52 \\ -0.80 & -0.25 \\ -0.71 & -0.45 \\ -0.53 & -0.66 \\ 1.82 & 1.41 \\ 0.07 & 0.23 \\ -0.37 & -0.36 \end{bmatrix} \begin{bmatrix} 0.7553 \\ 0.6553 \end{bmatrix} = \begin{bmatrix} -0.45 \\ -0.79 \\ 2.31 \\ -0.63 \\ -0.76 \\ -0.82 \\ -0.83 \\ 2.29 \\ 0.20 \\ -0.51 \end{bmatrix}$$

From the preceding result, we can see the 1D projection of the first principal component from the original 2D data.

Also, the eigenvalue of 1.5725 explains the fact that the principal component explains variance of 57 percent more than the original variables.

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

- Handwritten digits example from scikit- learn datasets
- Handwritten digits are created from 0-9 and its respective 64 features (8 x 8 matrix) of pixel intensities.
- The idea is to represent the original features of 64 dimensions into as few as possible

```
# PCA - Principal Component Analysis

>>> import matplotlib.pyplot as plt
>>> from sklearn.decomposition import PCA
>>> from sklearn.datasets import load_digits
>>> digits = load_digits()

>>> X = digits.data
>>> y = digits.target

>>> print (digits.data[0].reshape(8,8))
```

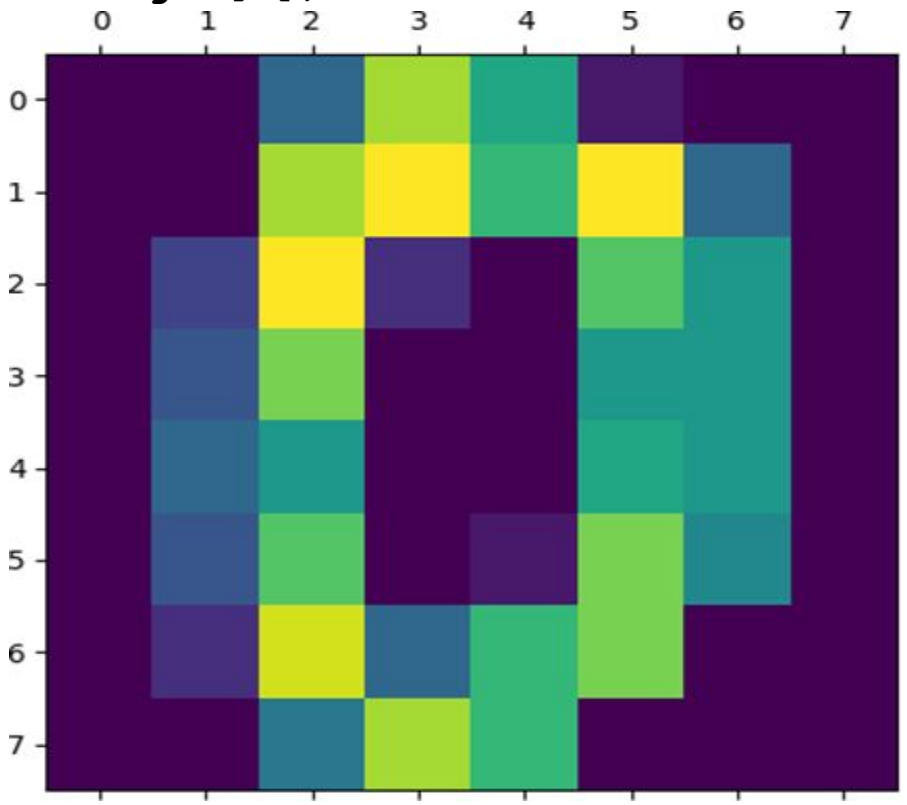
```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

Plot the graph using the plt.show function:

```
>>> plt.matshow(digits.images[0])
>>> plt.show()
```



Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

Before performing PCA, it is advisable to perform scaling of input data to eliminate any issues due to different dimensions of the data.

In the following code, we have applied scaling on all the columns separately:

```
>>> from sklearn.preprocessing import scale

>>> X_scale = scale(X,axis=0)
```

In the following, we have used two principal components, so that we can represent the performance on a 2D graph.

```
>>> pca = PCA(n_components=2)

>>> reduced_X = pca.fit_transform(X_scale)
>>> zero_x, zero_y = [],[] ; one_x, one_y = [],[]

>>> two_x,two_y = [],[]; three_x, three_y = [],[]
>>> four_x,four_y = [],[]; five_x,five_y = [],[]
>>> six_x,six_y = [],[]; seven_x,seven_y = [],[]

>>> eight_x,eight_y = [],[]; nine_x,nine_y = [],[]
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

we are appending the relevant principal components to each digit separately so that we can create a scatter plot of all 10 digits

```
>>> for i in range(len(reduced_X)):
...     if y[i] == 0:
...         zero_x.append(reduced_X[i][0])
...         zero_y.append(reduced_X[i][1])
...     elif y[i] == 1:
...         one_x.append(reduced_X[i][0])
...         one_y.append(reduced_X[i][1])

...     elif y[i] == 2:
...         two_x.append(reduced_X[i][0])
...         two_y.append(reduced_X[i][1])

...     elif y[i] == 3:
...         three_x.append(reduced_X[i][0])
...         three_y.append(reduced_X[i][1])

...     elif y[i] == 4:
...         four_x.append(reduced_X[i][0])
...         four_y.append(reduced_X[i][1])

...     elif y[i] == 5:
...         five_x.append(reduced_X[i][0])
```

Principal component analysis - PCA

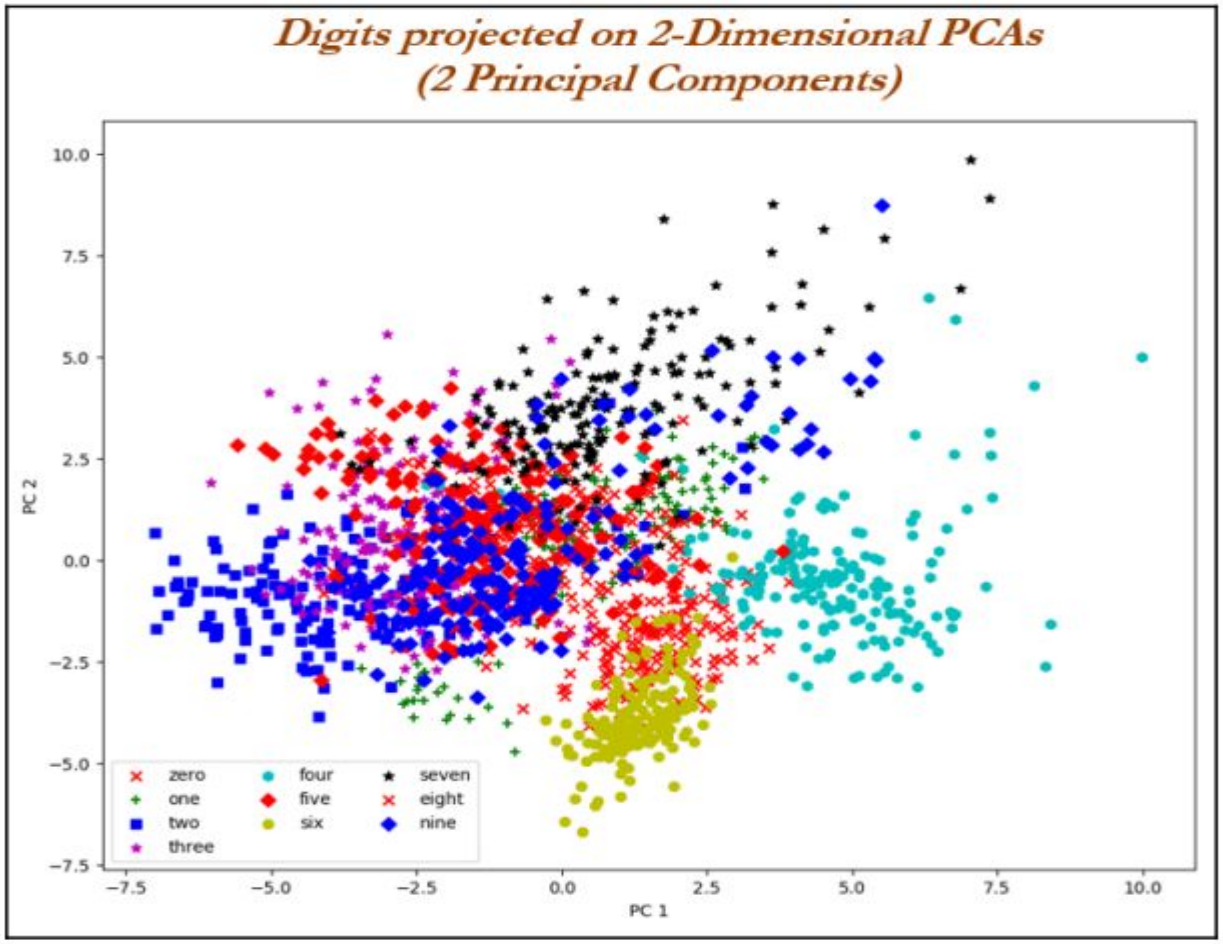
PCA applied on handwritten digits using scikit-learn

.

```
...             five_y.append(reduced_X[i][1])

...         elif y[i] == 6:
...             six_x.append(reduced_X[i][0])
...             six_y.append(reduced_X[i][1])

...         elif y[i] == 7:
...             seven_x.append(reduced_X[i][0])
...             seven_y.append(reduced_X[i][1])

...         elif y[i] == 8:
...             eight_x.append(reduced_X[i][0])
...             eight_y.append(reduced_X[i][1])
...         elif y[i] == 9:
...             nine_x.append(reduced_X[i][0])
...             nine_y.append(reduced_X[i][1])

>>> zero = plt.scatter(zero_x, zero_y, c='r', marker='x',label='zero')
>>> one = plt.scatter(one_x, one_y, c='g', marker='+')
>>> two = plt.scatter(two_x, two_y, c='b', marker='s')

>>> three = plt.scatter(three_x, three_y, c='m', marker='*')
>>> four = plt.scatter(four_x, four_y, c='c', marker='h')
>>> five = plt.scatter(five_x, five_y, c='r', marker='D')

>>> six = plt.scatter(six_x, six_y, c='y', marker='8')
>>> seven = plt.scatter(seven_x, seven_y, c='k', marker='*')
>>> eight = plt.scatter(eight_x, eight_y, c='r', marker='x')

>>> nine = plt.scatter(nine_x, nine_y, c='b', marker='D')


>>> plt.legend((zero,one,two,three,four,five,six,seven,eight,nine),
...             ('zero','one','two','three','four','five','six',
'seven','eight','nine'),
...             scatterpoints=1,
...             loc='lower left',
...             ncol=3,
...             fontsize=10)

>>> plt.xlabel('PC 1')
>>> plt.ylabel('PC 2')

>>> plt.show()
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn



Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

- In the following code, we have applied three PCAs so that we can get a better view of the data in a 3D space.
- The procedure is very much similar as with two PCAs, except for creating one extra dimension for each digit (X, Y, and Z)

```
# 3-Dimensional data
>>> pca_3d = PCA(n_components=3)
>>> reduced_X3D = pca_3d.fit_transform(X_scale)

>>> zero_x, zero_y,zero_z = [],[],[] ; one_x, one_y,one_z = [],[],[]
>>> two_x,two_y,two_z = [],[],[]; three_x, three_y,three_z = [],[],[]
>>> four_x,four_y,four_z = [],[],[]; five_x,five_y,five_z = [],[],[]
>>> six_x,six_y,six_z = [],[],[]; seven_x,seven_y,seven_z = [],[],[]
>>> eight_x,eight_y,eight_z = [],[],[]; nine_x,nine_y,nine_z = [],[],[]
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

```
>>> for i in range(len(reduced_X3D)):
...     if y[i]==10:
...         continue
...     elif y[i] == 0:
...         zero_x.append(reduced_X3D[i][0])
...         zero_y.append(reduced_X3D[i][1])
...         zero_z.append(reduced_X3D[i][2])
...     elif y[i] == 1:
...         one_x.append(reduced_X3D[i][0])
...         one_y.append(reduced_X3D[i][1])
...         one_z.append(reduced_X3D[i][2])

...     elif y[i] == 2:
...         two_x.append(reduced_X3D[i][0])
...         two_y.append(reduced_X3D[i][1])
...         two_z.append(reduced_X3D[i][2])

...     elif y[i] == 3:
...         three_x.append(reduced_X3D[i][0])
...         three_y.append(reduced_X3D[i][1])
...         three_z.append(reduced_X3D[i][2])

...     elif y[i] == 4:
...         four_x.append(reduced_X3D[i][0])
...         four_y.append(reduced_X3D[i][1])
...         four_z.append(reduced_X3D[i][2])

...     elif y[i] == 5:
...         five_x.append(reduced_X3D[i][0])
...         five_y.append(reduced_X3D[i][1])
...         five_z.append(reduced_X3D[i][2])

...     elif y[i] == 6:
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

```
...         six_x.append(reduced_X3D[i][0])
...         six_y.append(reduced_X3D[i][1])
...         six_z.append(reduced_X3D[i][2])

...     elif y[i] == 7:
...         seven_x.append(reduced_X3D[i][0])
...         seven_y.append(reduced_X3D[i][1])
...         seven_z.append(reduced_X3D[i][2])

...     elif y[i] == 8:
...         eight_x.append(reduced_X3D[i][0])
...         eight_y.append(reduced_X3D[i][1])
...         eight_z.append(reduced_X3D[i][2])
...     elif y[i] == 9:
...         nine_x.append(reduced_X3D[i][0])
...         nine_y.append(reduced_X3D[i][1])
...         nine_z.append(reduced_X3D[i][2])
```

Principal component analysis - PCA

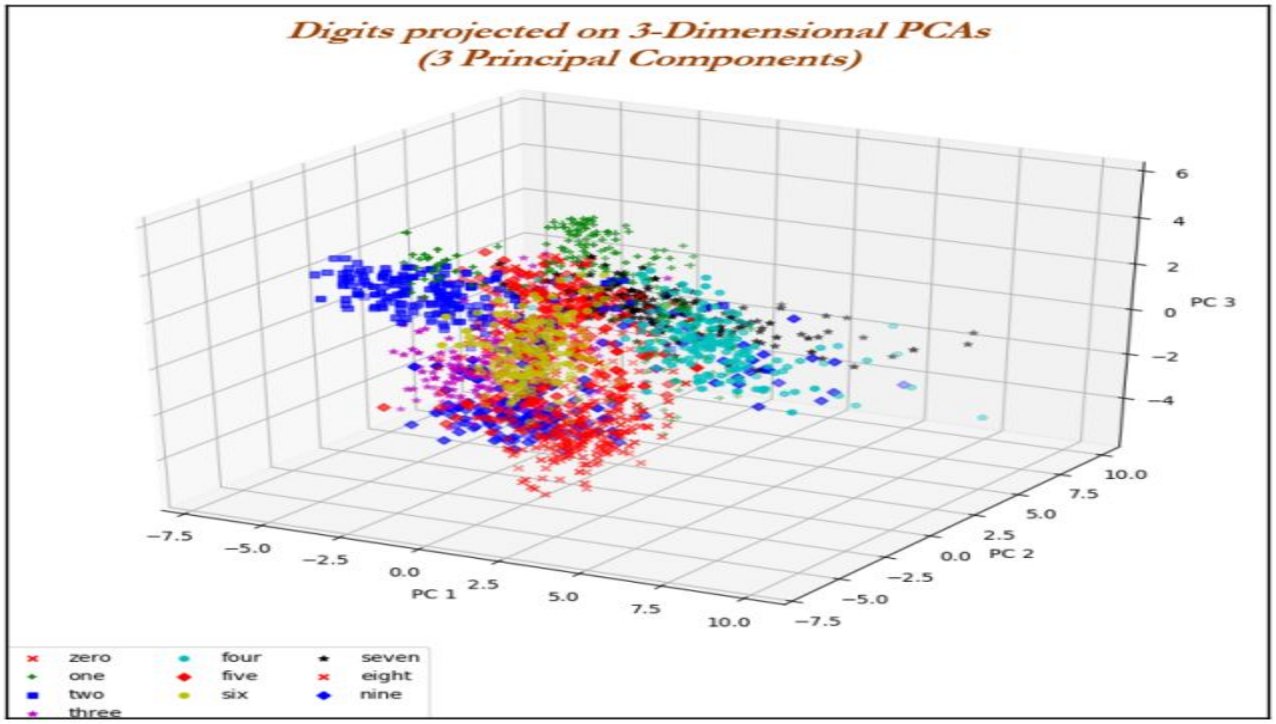
PCA applied on handwritten digits using scikit-learn

```
# 3- Dimensional plot
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')

>>> ax.scatter(zero_x, zero_y,zero_z, c='r', marker='x',label='zero')
>>> ax.scatter(one_x, one_y,one_z, c='g', marker='+',label='one')
>>> ax.scatter(two_x, two_y,two_z, c='b', marker='s',label='two')

>>> ax.scatter(three_x, three_y,three_z, c='m', marker='*',label='three')
>>> ax.scatter(four_x, four_y,four_z, c='c', marker='h',label='four')
>>> ax.scatter(five_x, five_y,five_z, c='r', marker='D',label='five')

>>> ax.scatter(six_x, six_y,six_z, c='y', marker='8',label='six')
>>> ax.scatter(seven_x, seven_y,seven_z, c='k', marker='*',label='seven')
>>> ax.scatter(eight_x, eight_y,eight_z, c='r', marker='x',label='eight')

>>> ax.scatter(nine_x, nine_y,nine_z, c='b', marker='D',label='nine')

>>> ax.set_xlabel('PC 1')
>>> ax.set_ylabel('PC 2')
>>> ax.set_zlabel('PC 3')

>>> plt.legend(loc='upper left', numpoints=1, ncol=3, fontsize=10,
bbox_to_anchor=(0, 0))

>>> plt.show()
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn

In a 3D plot, Digit *2* is at the extreme left and digit *0* is at the lower part of the plot. Whereas, digit *4* is at the top-right end, digit *6* seems to be more towards the *PC 1* axis.



Principal component analysis - PCA

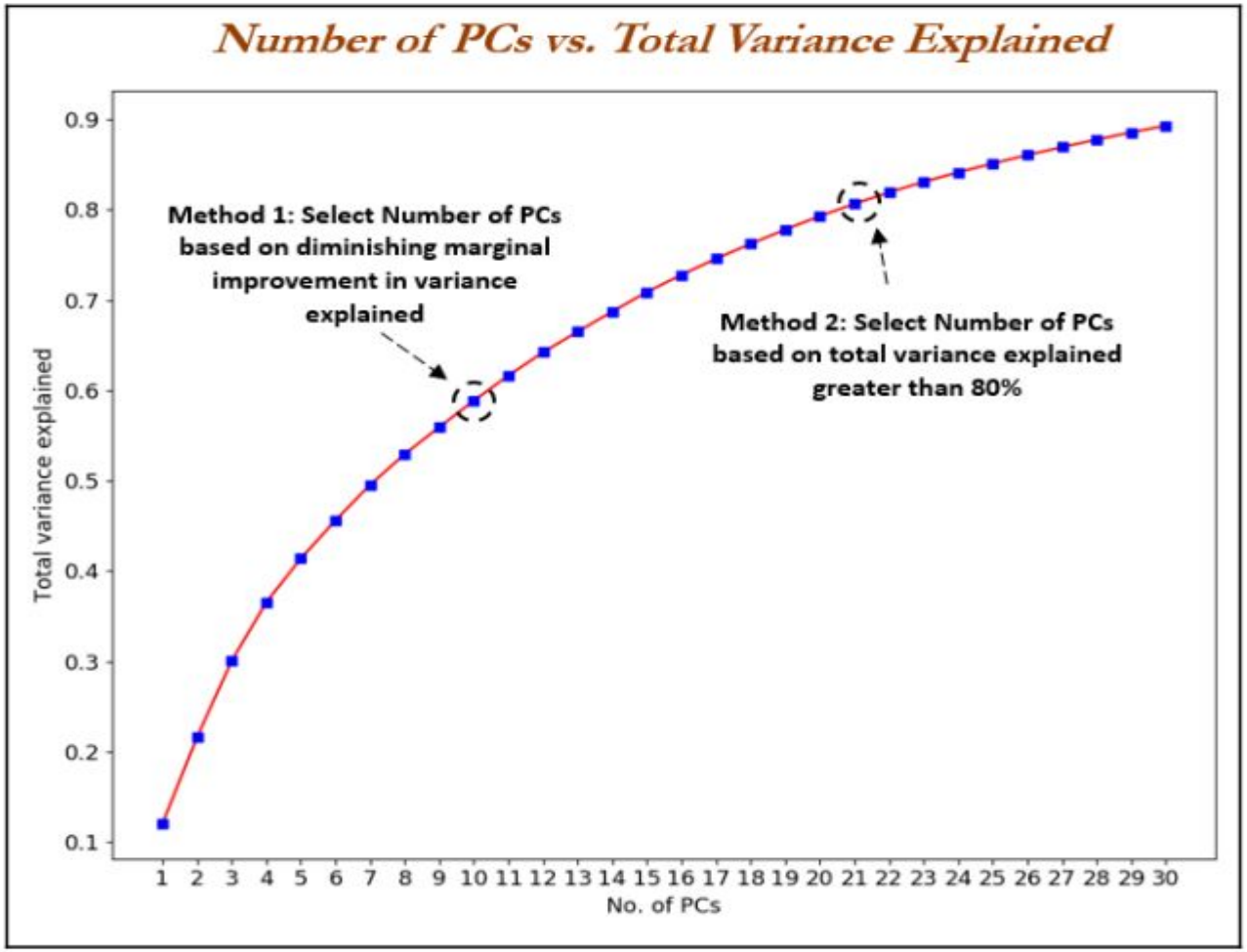
PCA applied on handwritten digits using scikit-learn

- Choosing the number of PCAs to be extracted is an open-ended question in unsupervised learning, but there are some turnarounds to get an approximated view.
- There are two ways we can determine the number of clusters:
  - Check where the total variance explained is diminishing marginally
  - Total variance explained greater than 80 percent
- The following code does provide the total variance explained with the change in number of principal components.

```
# Choosing number of Principal Components
>>> max_pc = 30

>>> pcs = []
>>> totexp_var = []

>>> for i in range(max_pc):
...     pca = PCA(n_components=i+1)
...     reduced_X = pca.fit_transform(X_scale)
...     tot_var = pca.explained_variance_ratio_.sum()
...     pcs.append(i+1)
...     totexp_var.append(tot_var)

>>> plt.plot(pcs,totexp_var,'r')
>>> plt.plot(pcs,totexp_var,'bs')
>>> plt.xlabel('No. of PCs',fontsize = 13)
>>> plt.ylabel('Total variance explained',fontsize = 13)

>>> plt.xticks(pcs,fontsize=13)
>>> plt.yticks(fontsize=13)
>>> plt.show()
```

Principal component analysis - PCA

PCA applied on handwritten digits using scikit-learn



Singular value decomposition - SVD

- Many implementations of PCA use singular value decomposition to calculate eigenvectors and eigenvalues.
- SVD is given by the following equation:

$$X = U \Sigma V^T$$

$$Operations\ performed\ on\ data = (rotate)\ (stretch)\ (rotate)$$

- Columns of U are called left singular vectors of the data matrix, the columns of V are its right singular vectors, and the diagonal entries of are its singular values.
- Left singular vectors are the eigenvectors of the covariance matrix and the diagonal element of are the square roots of the eigenvalues of the covariance matrix

Singular value decomposition - SVD

Advantages of SVD:

- SVD can be applied even on rectangular matrices; whereas, eigenvalues are defined only for square matrices.
- The equivalent of eigenvalues obtained through the SVD method are called singular values, and vectors obtained equivalent to eigenvectors are known as singular vectors.
- However, as they are rectangular in nature, we need to have left singular vectors and right singular vectors respectively for their dimensions.
- If a matrix *A* has a matrix of eigenvectors *P* that is not invertible, then *A* does not have an eigen decomposition.
- However, if *A* is *m* x *n* real matrix with *m* > *n*, then A can be written using a singular value decomposition.
- Both *U* and *V* are orthogonal matrices, which means *UT U* = *I* (*I* with *m* x *m* dimension) or *VT V* = *I* (here *I* with *n* x *n* dimension), where two identity matrices may have different dimensions.
- Σ is a non-negative diagonal matrix with *m* x *n* dimensions.

Singular value decomposition - SVD

- Computation of singular values and singular vectors

$$X^T X = V \Sigma^T \Sigma V^T$$

- First stage - singular values/eigenvalues are calculated with the equation. Once we obtain the singular/eigenvalues, we will substitute to determine the *V* or right singular/eigen vectors:

$$\det(X^T X - \lambda * I) = 0$$

- we will substitute to obtain the left singular vectors *U* using the equation mentioned as follows

$$X V = U \Sigma$$

Singular value decomposition - SVD

SVD applied on handwritten digits using scikit-learn

SVD can be applied on the same handwritten digits data to perform an apple-to-apple comparison of techniques

```
# SVD
>>> import matplotlib.pyplot as plt
>>> from sklearn.datasets import load_digits
>>> digits = load_digits()
>>> X = digits.data
>>> y = digits.target
```

Singular value decomposition - SVD

SVD applied on handwritten digits using scikit-learn

- In the following code, 15 singular vectors with 300 iterations are used,
- Two types of SVD functions are used ,
  - a function randomized_svd provide the decomposition of the original matrix
  - a TruncatedSVD can provide total variance explained ratio.

```
>>> from sklearn.utils.extmath import randomized_svd
>>> U,Sigma,VT =
randomized_svd(X,n_components=15,n_iter=300,random_state=42)
>>> import pandas as pd

>>> VT_df = pd.DataFrame(VT)
>>> print ("\nShape of Original Matrix:",X.shape)

>>> print ("\nShape of Left Singular vector:",U.shape)
>>> print ("Shape of Singular value:",Sigma.shape)
>>> print ("Shape of Right Singular vector",VT.shape)
```

Singular value decomposition - SVD

SVD applied on handwritten digits using scikit-learn

```
Shape of Original Matrix: (1797, 64)

Shape of Left Singular vector: (1797, 15)
Shape of Singular value: (15,)
Shape of Right Singular vector (15, 64)
```

The original matrix of dimension (1797 x 64) has been decomposed into a left singular vector (1797 x 15), singular value (diagonal matrix of 15), and right singular vector (15 x 64).

Singular value decomposition - SVD

SVD applied on handwritten digits using scikit-learn

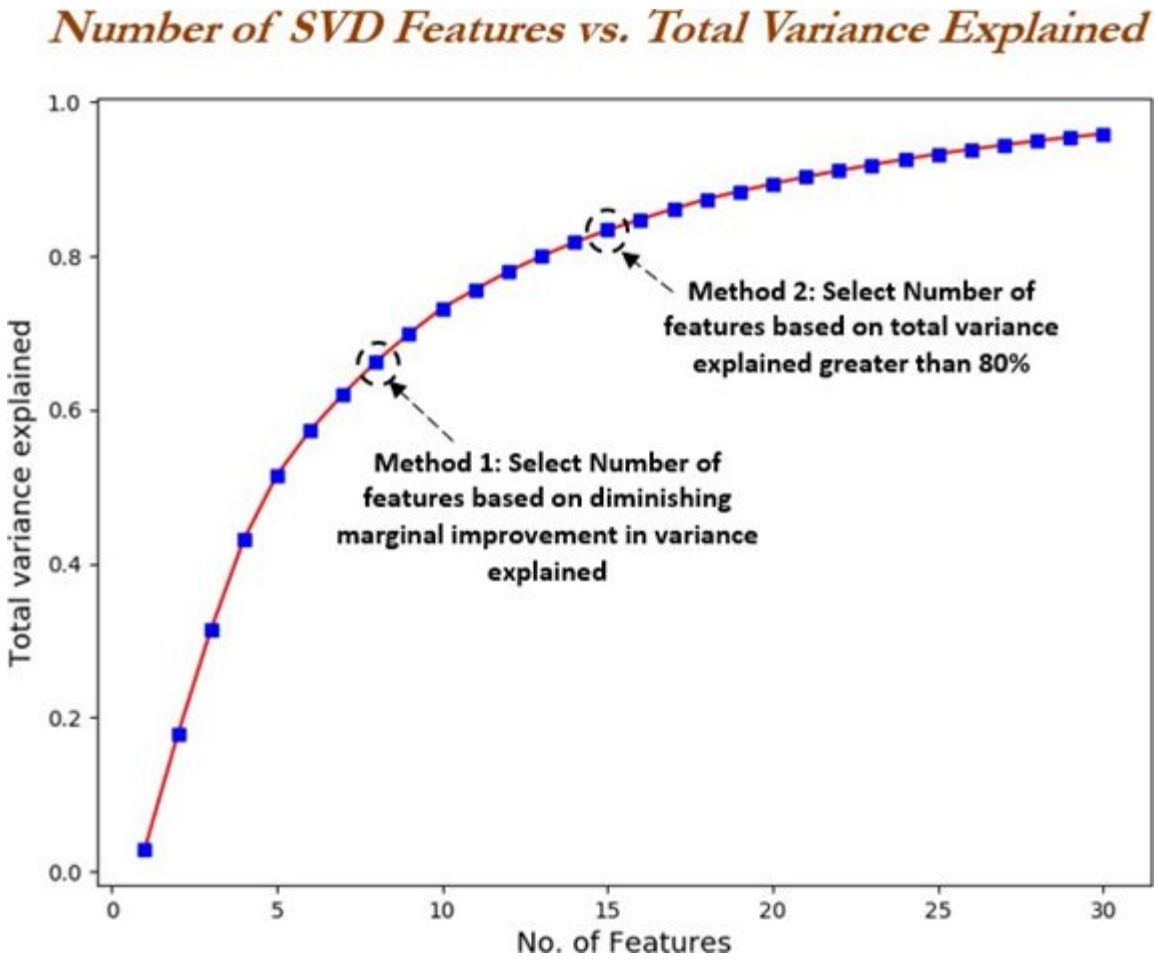
```
>>> n_comps = 15

>>> from sklearn.decomposition import TruncatedSVD
>>> svd = TruncatedSVD(n_components=n_comps, n_iter=300,
random_state=42)
>>> reduced_X = svd.fit_transform(X)
>>> print("\nTotal Variance explained for %d singular
features are %0.3f"%(n_comps,
svd.explained_variance_ratio_.sum()))
```

Singular value decomposition - SVD

The following code illustrates the change in total variance explained with respective to change in number of singular values:

```
# Choosing number of Singular Values
>>> max_singfeat = 30
>>> singfeats = []
>>> totexp_var = []

>>> for i in range(max_singfeat):
...     svd = TruncatedSVD(n_components=i+1, n_iter=300, random_state=42)
...     reduced_X = svd.fit_transform(X)
...     tot_var = svd.explained_variance_ratio_.sum()
...     singfeats.append(i+1)
...     totexp_var.append(tot_var)

>>> plt.plot(singfeats,totexp_var,'r')
>>> plt.plot(singfeats,totexp_var,'bs')
>>> plt.xlabel('No. of Features',fontsize = 13)
>>> plt.ylabel('Total variance explained',fontsize = 13)

>>> plt.xticks(pcs,fontsize=13)
>>> plt.yticks(fontsize=13)
>>> plt.show()
```

Singular value decomposition - SVD



Statistical Machine Learning
Question Bank – all possible questions I all units

UNIT I			
Statistical terminology for model building and validation-Machine Learning, Major differences between statistical modeling and machine learning- Steps in machine learning model development and deployment - Statistical fundamentals and terminology for model building and validation - Bias versus variance trade-off,Train and test data - Linear regression versus gradient descent Machine learning losses- When to stop tuning machine learning models - When to stop tuning machine learning models - Train, validation, and test data Cross-validation – Grid Search – Machine Learning model overview			
PART B (4 Marks)			
1	Analyze the path from statistical to machine learning	1	3
2	Explain Confusion matrix	1	2
3	Illustrate Supervised learning with real time examples	1	2
4	Classify the machine learning loses	1	2
5	When to stop tuning machine learning models	1	3
6	Explain Cross Validation	1	2
7	Illustrate test, validation and test data	1	2
8	Explain Grid search	1	2
PART C (12 Marks)			
1	Compare Statistical and Machine learning	1	2
2	Illustrate the Machine learning terminology for model building and validation	1	2
3	Explain in detail category of machine learning	1	2
4	Explain Quantiles and it types in detail	1	2
5	Compare Linear regression with gradient descent	1	3

UNIT II			
Comparison between regression and machine learning models- Compensating factors in machine learning models- Assumptions of linear regression Steps applied in linear regression modelling - Machine learning models - ridge and lasso regression - Example of ridge regression machine learning, Example of lasso regression machine learning mode - Logistic Regression Versus Random Forest-Maximum likelihood estimation - Terminology involved in logistic regression - Applying steps in logistic regression modelling - Random forest - Example of random forest using German credit data - Grid search on random forest - Variable importance plot - Comparison of logistic regression with random forest			
PART B (4 Marks)			
1	Compensating factors in machine learning models	2	2
2	Explain the steps applied in linear regression modeling	2	2
3	Compare Ridge and Lasso regression modeling	2	2
4	Explain Logistics Regression and its advantages	2	2
5	Explain the steps applied in logistic regression modeling	2	2

6	Write short notes on Random Forest	2	2
7	Explain random forest using German credit data	2	2
8	Explain grid search on random forest	2	2
	PART C (12 Marks)		
1	Compare Linear regression and Machine learning model	2	2
2	Assumption of linear regression in detail	2	3
3	Explain simple linear regression from first principles	2	2
4	Explain Maximum likelihood estimation	2	2
5	Terminology involved in logistic regression	2	1

	UNIT III		
K-nearest neighbors-KNN voter example - Curse of dimensionality-Curse of dimensionality with 1D, 2D, and 3D example. Curse of dimensionality with 3D example. KNN classifier with breast cancer Wisconsin data example- Naive Bayes - Probability fundamentals - Joint probability- Understanding Bayes theorem with conditional probability- Naive Bayes classification- Laplace estimator- Naive Bayes SMS spam classification example.			
	PART B (4 Marks)		
1	Explain short notes on KNN	3	2
2	Explain KNN voter example	3	2
3	Explain Curse of dimensionality	3	2
4	What is tuning of k-value in KNN classifier	3	2
5	What is a Joint probability? Explain it with a simple Venn Diagram.	3	1
6	Explain the use of Conditional probability.	3	1
7	What is the purpose of using Laplace estimator?	3	2
8	Explain TF-IDF with an example.	3	3
	PART C (12 Marks)		
1	What will happen if the Curse of dimensionality with 1D, 2D, and 3D Explain	3	2
2	KNN classifier with breast cancer Wisconsin data example	3	3
3	Describe the various preprocessing stages for Natural Language Processing with a suitable example.	3	1
4	Explain Naïve Bayes Classification with an example	3	2
5	Discuss SMS Spam classification using Naïve Bayes algorithm.	3	3

UNIT IV

Support Vector Machines and Neural Networks-Support vector machines working principles-Maximum margin classifier- Support vector classifier- Support vector machines- Kernel functions- Artificial neural networks – ANN - Forward propagation and back propagation-Optimization of neural networks-Stochastic gradient descent – SGD- Introduction to deep learning-Solving methodology- Deep learning software

PART B (4 Marks)

1	When the Support Vector Machine are used? Give suitable example to illustrate it.	4	3
2	Write short notes on different types of Kernel function.	4	1
3	Explain Super Vector Classifiers with an example.	4	3
4	What are the parameters requiredforchoosingforbuildingneuralnetworks?	4	2
5	What is an Activation function and what are its types?	4	1
6	Write the Thumb rules in designing deep neural networks.	4	1
7	Give the various types of deep learning software used by manypractitionersacross theworld.	4	2
8	Explain the working principle of Artificial Neural Network.	4	2

PART C (12 Marks)

1	Discuss on Super Vector Machine working principles in detail.	4	1
2	Describe in detail on Kernel functions.	4	1
3	Explain in detail on Forward and Backward Propagation in Artificial Neural Network with a neat diagram.	4	2
4	Illustrate Stochastic gradient descent for Optimization of neural networks with a neat diagram.	4	2
5	Explain the Deep architecture of Neural networks. Give the thumb rules for designing the Deep neural Networks. Briefly explain about Deep Learning software	4	2

UNIT V

K-means clustering-K-means working methodology from first principles- Optimal number of clusters and cluster evaluation - The elbow method- K-means clustering with the iris data example- Principal component analysis - PCA-PCA working methodology from first principles- PCA applied on handwritten digits using scikit-learn- Singular value decomposition – SVD- SVD applied on handwritten digits using scikit-learn

PART B (4 Marks)

1	Discuss on evaluating the cost function of K-means clustering.	5	2
2	Explain the elbow method to determine the optimal number of clusters in k-meansclustering.	5	1
3	Give few applications of K-means clustering.	5	2
4	Write notes on utilities of Principal Component Analysis.	6	2
5	Illustrate PCA for two Dimensional data with two principal components.	6	2
6	Illustrate the Graphical representation of Eigen values and Eigen vectors.	6	2
7	Write the advantages of Singular Value Decomposition	6	1
8	Explain the computation of singular values and singular vectors in SVD.	6	2

PART C (12 Marks)			
1	Explain K-means clustering working methodology from first principles with a suitable example.	5	2
2	Discuss in detail on K-means clustering with iris data example.	5	3
3	Describe Principal Component Analysis working methodology from first principles with a suitable example.	6	2
4	Illustratethe application of PCA for handwritten digits using scikit-learn.	6	3
5	Describe SVD applied on handwritten digits using scikit-learn.	6	3