Machine Learning Types

Machine Learning Definition:

Arthur Samuel described it as:

The field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell provides a more modern definition:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience.

Ex: playing cricket.
E = the experience of playing many games of cricket.
T = the task of playing cricket.
P = the probability that the program will win the next game.

Types of Learnings:

1. Supervised Learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforcement learning

1. Supervised Learning:

Supervised learning involves training a model on a **labeled dataset**, which means that each training example is paired with an output label. The goal is for the model to learn a mapping from inputs to outputs, so it can predict the label of new, unseen data.
Ex: Predicting house prices based on features like size, location, and number of bedrooms.

2. Unsupervised Learning

Unsupervised learning involves training a model on unlabeled data. The goal is to identify the natural structure within a set of data points.

Ex: Clustering data-points into different segments without knowing the categories beforehand.

3. Semi-Supervised Learning

Semi-supervised learning is a combination of supervised and unsupervised learning. It uses a small amount of labeled data and a large amount of unlabeled data to improve learning accuracy.

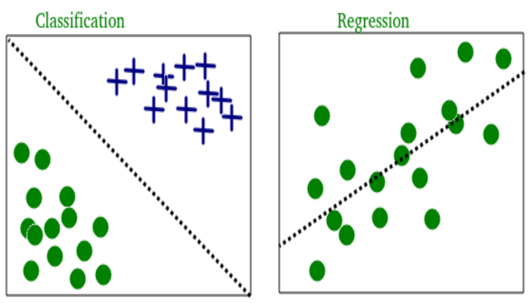
Ex: In a scenario where labeling data is expensive, such as medical image analysis, a few labeled images can be used along with many unlabeled images to train a more accurate model.

4. Reinforcement Learning

Reinforcement learning is about training models to make sequences of decisions, where model learns from real environment to make decisions.

Ex: Training a robot to navigate a maze, where it receives rewards for reaching the end and penalties for hitting walls.

ML types Based on Output:



Classification:

Inputs are divided into two or more classes, and will predict the new value belongs to which class is called classification.

Ex: Identifying email messages are “spam” or “not spam”.

The classification problem solves in two ways.

1. Class labels – Here threshold is fixed as 0.5
2. Probability – Here threshold will take based on probability.

Regression:

It is also a supervised learning problem, but it identifies the next possible values in continues numeric values (target values).

Clustering:

Cluster analysis, or clustering, is an unsupervised machine learning task. It automatically discovering natural grouping in data. Clustering algorithms only interpret the input data and find natural groups or clusters in feature space.

Terminologies involved in ML:

Model:

Model is a mathematical representation or a computational algorithm designed to identify patterns or make predictions based on input data.

Feature:

A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a *feature vector*. Feature vectors are fed as input to the model.

Target:

A target variable or label is the value to be predicted by our model.

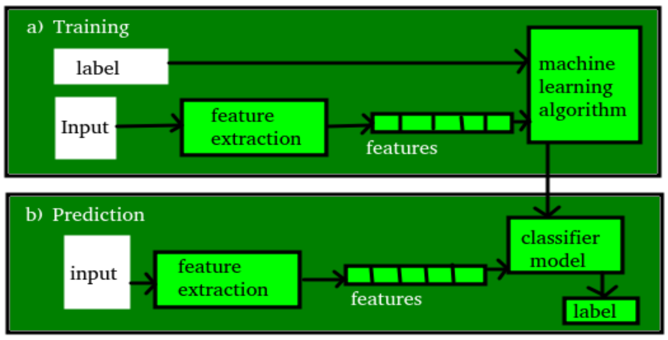
Training:

Giving the data to model for predicting the outputs this process is called training.

Prediction:

The Outputs of model after training is called predicted values, and this process is called prediction.

The figure shown below clears the above concepts:



Linear Regression

Carl Friedrich Gauss
(30 April 1777 – 23 February 1855)

The concept of linear regression can be traced back to the early 19th century when mathematician and scientist **Carl Friedrich Gauss** developed the method of least squares. Gauss used this method to fit a line to a set of data points and determine the coefficients of the line.

However, the first person to formally introduce the concept of linear regression was **Sir Francis Galton**, a British mathematician and scientist, in the late **19th century**. Galton used regression analysis to study the relationship between the height of parents and their offspring. He coined the term "**regression**" as a reference to the tendency of offspring to move towards the average, or mean, of a population.

Later, in the early 20th century, the development of correlation analysis and the formulation of the Pearson correlation coefficient by Karl Pearson paved the way for the widespread use of linear regression in scientific research. Today, linear regression is one of the most widely used statistical techniques and is applied in a wide range of fields, including economics, finance, biology, engineering, and social sciences.

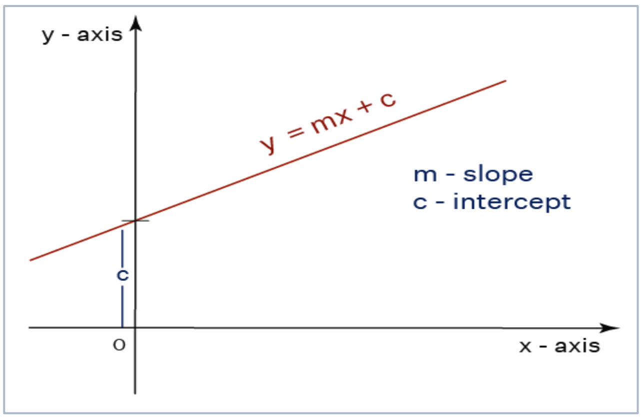
Introduction:

Def: Linear regression is a statistical method used to ***find the relationship between a dependent variable and one or more independent variables***.

The goal of linear regression is to find the best fitting straight line, or linear function, that can predict the value of the dependent variable based on the values of the independent variables.

In other words, linear regression involves finding the equation of a straight line that most closely approximates the relationship between the dependent variable and the independent variables. This equation can then be used to make predictions about the dependent variable given new values of the independent variables.

Linear regression assumes that the relationship between the dependent variable and the independent variables is linear, meaning that as the values of the independent variables increase or decrease, the value of the dependent variable changes in a constant and predictable manner. The technique is widely used in many fields of study, including economics, finance, psychology, and social sciences, among others.



Simple Linear Equation Graph

Simple Working:

$$Y = mX + C$$

1. ***Taking input (X):*** First it takes the input feature X (Independent variable) in the form of matrix.
2. ***Selecting m and C values:*** First it selects the slope *m* and intercept point *C* randomly. But these two values will adjust based on evaluation (testing) result to get the best accuracy.
3. ***Building the trained model Y:*** Next substituting the *m, X* and *C* values in the hypothesis function and we will get the resultant *Y* matrix which is called trained model. We need to build a graph by using *X* and *Y* values to view this model in graphical format.
4. ***Model evaluation:*** Now pass the new unknown *X* values to trained model. which means substitute new matrix *X* in hypothesis function with the same *m, C* values, then it will give predicted values matrix *Y*. Now identify the accuracy of the model by observe the difference between actual and predicted *Y* values.
5. ***Adjust m and C:*** If accuracy of the model is good then the model is ready and no need to adjust the *m, C* values, otherwise we need to take the new *m, C* and repeat all above steps until to get the best accuracy.

Mathematical Intuition:

The linear regression equation is an algebraic linear line equation. The general form of a linear regression equation with one independent variable is:

$$Y = mX + C$$

Where:

- *Y* is the dependent variable (*The variable we want to predict or explain*)
- *X* is the independent variable (*The variable we use to predict Y*)
- *m* is the slope (*The change in Y for a one-unit change in X*)
- *C* is the y-intercept (*The value of Y when X = 0*)

In this equation, the y-intercept (C) is the point where the regression line crosses the Y-axis. It represents C is the expected value of Y when X is zero.

The slope (m) represents the rate of change in Y for each one-unit change in X. It tells us how much Y is expected to increase or decrease for a one-unit increase or decrease in X.

$$m = \frac{\Delta y}{\Delta x}$$

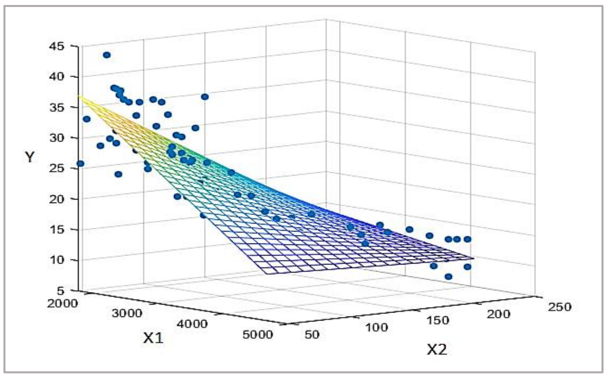
Multiple Linear Regression:

Linear regression can also include multiple independent variables called as multiple linear regression. In that case, the equation form is:

$$Y = C + m_1x_1 + m_2x_2 + m_3x_3 + \cdots \ldots + m_px_p$$

Where:

- *Y* is the dependent variable.
- $x_1, x_2, \ldots, x_p$ are the independent variables.
- *C* is the y-intercept.
- $m_1, m_2, \ldots, m_p$ are the slopes (the change in Y for a one-unit change in each independent variable, holding all other variables constant)



Multiple Linear equation 3D graph

In this case, the slope ($m_1, m_2, \ldots, m_p$) represents the expected change in *Y* for a one-unit change in each independent variable, holding all other variables constant.

However, a multiple linear regression equation cannot be visualized in a picture format beyond three dimensions, as humans can only see or draw up to three dimensions. Therefore, we can observe a multiple linear regression equation in a 3D graph with two independent variables, X_1 and X_2.

In a two-dimensional linear equation, the graph is represented as a line. However, in a multiple linear regression equation with three dimensions, the graph becomes a plane. This indicates that the shape of the graph changes as the number of dimensions in the equation increases.

- Since we understood linear regression equations, now let's understand the linear regression modeled hypothesis function.
- This hypothesis function is the actual linear regression model as shown in below.

$$\boldsymbol{h(x) = m_0 + m_1x_1 + m_2x_2 + m_3x_3 + \cdots \ldots + m_px_p + \varepsilon}$$

- Let's consider C value m_0 as and consider only 3 independent vectors, then equation will be.

$$h(x) = m_0 + m_1x_1 + m_2x_2 + m_3x_3 + \varepsilon$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad m = \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ 1 & x_{31} & x_{32} & x_{33} \end{bmatrix} \qquad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix}$$

$$mx + \varepsilon = \begin{bmatrix} m_0 + m_1x_{11} + m_2x_{12} + m_3x_{13} + \varepsilon_1 \\ m_0 + m_1x_{21} + m_2x_{22} + m_3x_{23} + \varepsilon_2 \\ m_0 + m_1x_{31} + m_2x_{32} + m_3x_{33} + \varepsilon_3 \end{bmatrix}$$

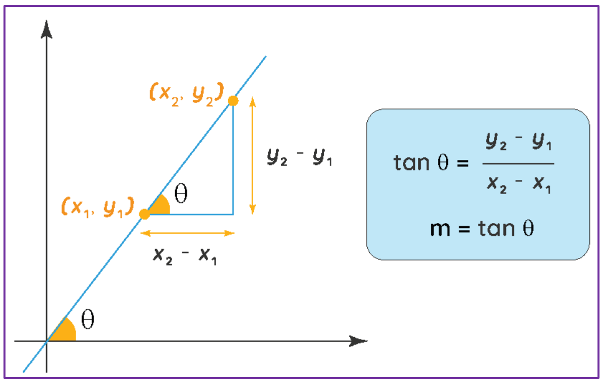
Slop Calculation: We know the slop of line equation is m. let's understand how to calculate the slop.

$$m = \frac{\Delta\, y}{\Delta\, x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Where:

$\Delta\, y$ = small change in y
$\Delta\, x$ = small change in x

The $\boldsymbol{m}$ explains that how much change coming in $\boldsymbol{y}$ when a small change comes in $\boldsymbol{x}$.



Slop calculation

Advantages:

1. ***Simplicity:*** Linear regression is a simple and straightforward statistical method that can be easily understood and implemented.
2. ***Coefficients are Interpretability:*** The coefficients in linear regression have clear and interpretable meanings, allowing researchers to draw insights from the relationships between variables.
3. ***Efficiently handle large datasets:*** Linear regression can handle large datasets efficiently and is computationally inexpensive compared to other more complex methods.

Disadvantages:

1. ***Assumes linear relationship:*** Linear regression assumes that there is a linear relationship between the dependent variable and the independent variables. If this assumption is not met, the model may not accurately represent the data.
2. ***Sensitive to outliers:*** Linear regression is sensitive to outliers in the data. An outlier can have a significant effect on the slope and intercept of the regression line, which can result in a poor fit.
3. ***Sensitive to correlation:*** Linear regression assumes that the observations are independent of each other. If there is a correlation between the observations, the model may not provide accurate predictions.
4. ***Cannot handle categorical variables:*** Linear regression is not well-suited for handling categorical variables. While

there are techniques to handle categorical variables, such as dummy coding, these techniques may result in a large number of predictor variables.

5. ***Assumes homoscedasticity:*** Linear regression assumes that the variance of the errors is constant across all levels of the independent variables. If the variance is not constant, the model may not provide accurate predictions.
6. ***Overfitting:*** Linear regression can suffer from overfitting if the model is too complex. This can occur if there are too many predictor variables or if the model is too flexible.
7. ***Non-linear relationships:*** If there is a non-linear relationship between the independent variables and the dependent variable, linear regression may not provide accurate predictions. In such cases, non-linear regression techniques may be more appropriate.

Code:

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Loading data from csv file
data = pd.read_csv("/path/data.csv")

# Selecting required features
X_y_values = data.iloc[:, 5:7].values
# Fish Height
X = X_y_values[:, 0].reshape(-1, 1)
# Fish width
Y = X_y_values[:, 1].reshape(-1, 1)
print(X.shape, Y.shape)

# splitting the data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=.3, random_state=5)

# Model creation
reg = LinearRegression()
# Model training
reg = reg.fit(x_train, y_train)
# Model testing
Y_pred = reg.predict(x_test

print("Slope of line (weight) : ", reg.coef_)
print("intercepting on Y-Axis : ", reg.intercept_)
print("accuracy of model : ", metrics.explained_variance_score(y_test, Y_pred))

val = float(input('Enter Height of the fish: \n'))
print("Width = ",float(reg.predict([[val]])))
```

Output:

```
(159, 1) (159, 1)
```

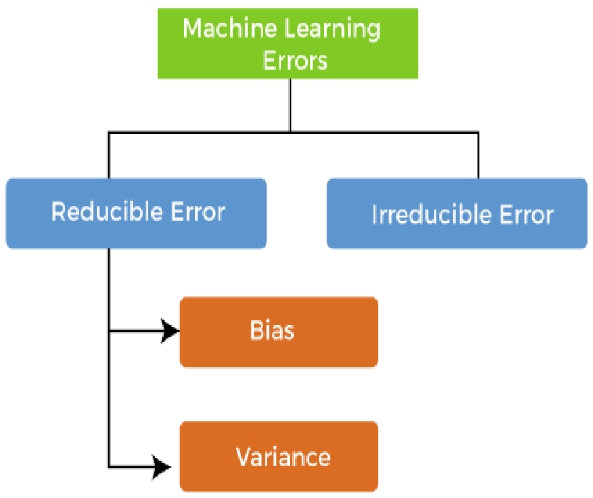
```
Slope of line (weight) :  [[0.31561454]]
intercepting on Y-Axis :  [1.63185062]
accuracy of model :  0.6228021009504172

Enter Height of the fish:
200
Width =  64.75475762935609
```

Bias And Variance

In machine learning, an error refers to the discrepancy or difference between the predicted value and the actual value. It quantifies the model's performance and indicates how well it is able to generalize from the training data to make accurate predictions on new unseen data.

- However, if the machine learning model is not accurate, it can make prediction and training errors are usually known as Bias and Variance.
- The main aim of ML/data science analysts is to reduce these errors in order to get more accurate results.

Types of Errors in ML:



There are mainly two types of errors in machine learning, which are:

Reducible Errors: These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.

Irreducible Errors: These errors will always be present in the model.

Bias:

Def: While **training** the model, the difference occurs between **prediction values** and **actual values**, and this difference is known as **bias** errors.

Low Bias:

- Models with low bias have the ability to capture complex relationships, patterns and can fit the training data closely.
- A low bias model is very flexible can adapt well to a wide range of datasets and can learn from the data without making strong assumptions or simplifications.
- However, it's important to note that while low bias allows the model to fit the training data well, it doesn't necessarily guarantee good performance on unseen data. If the model becomes too complex and tries to capture noise or random fluctuations in the training data, it may suffer from overfitting. This can lead to poor generalization and high error on new, unseen data.

High Bias:

- A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset.
- A high bias model also cannot perform well on new data.
- High bias mainly occurs due to a much simple model.

Ways to Reduce High Bias:

- Increase the input features as the model is underfitted.
- Decrease the regularization term.
- Use more complex models, such as including some polynomial features.

Note: *Some examples of machine learning algorithms with low bias are* ***Decision Trees, k-Nearest Neighbors and Support Vector Machines****. At the same time, an algorithm with high bias is* ***Linear Regression, Linear Discriminant Analysis and Logistic Regression.***

Variance:

Def: While **testing** the model on **new data**, a difference occurs between **prediction values** and **new data values** (random test value) is known as ***variance*** errors.

- In simple words, variance tells that how much a **random variable** is different from its **expected value**.

Low Variance:

- The subject behind low variance is model generalization. Models with low variance have learned the underlying patterns and relationships in the training data without memorizing or overfitting to the specific data points.
- When the variance is low, it means that the model's predictions are **relatively consistent** and stable **across different datasets** or **subsets of the same data**.
- Low variance indicates that the model is not overly sensitive to small changes in the training data.
- It can able to capture the essential features and generalize well to new, unseen data.

High Variance:

- A model that shows high variance learns a lot and perform well with the training dataset, and does not generalize well with the unseen dataset.
- As a result, such a model gives good results with the training dataset but shows high error rates on the test dataset.

Ways to Reduce High Variance:

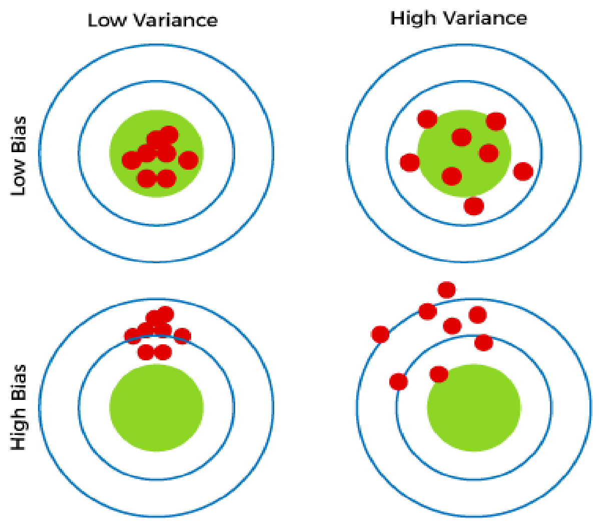
Reduce the input features or number of parameters as a model is overfitted:

- Do not use a much complex model.
- Increase the training data.
- Increase the Regularization term.

Note: *Some examples of machine learning algorithms with low variance are,* ***Linear Regression, Logistic Regression, and Linear discriminant analysis****. At the same time, algorithms with high variance are* ***decision tree, Support Vector Machine, and K-nearest neighbors.***

Different Combinations of Bias-Variance:

There are four possible combinations of bias and variances, which are represented by the below diagram:



Low-Bias, Low-Variance: The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.

Low-Bias, High-Variance: With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting.**

High-Bias, Low-Variance: With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.

High-Bias, High-Variance: With high bias and high variance, predictions are inconsistent and also inaccurate on average.

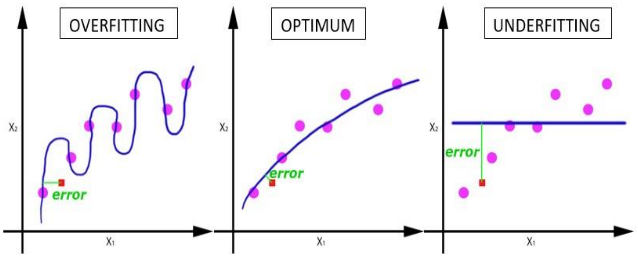
Overfitting and Underfitting

Overfitting and Underfitting are the two main problems that occurs in machine learning model training and these degrade the performance of the machine learning models.

- This overfitting and underfitting concept mainly explain about ***generalization ability*** of the model.
- Here ***generalization*** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input.
- The main goal of each machine learning model is to ***generalize well***.

There are three types of fitting scenarios involved in models:

1. Underfit
2. Overfit
3. Good fit



Underfit:

The condition when a model not fit to the data properly and doesn't identify the hidden actual pattern in the data is called underfit.

Reasons for Underfitting:

- High bias and high variance.
- The size of the training dataset used is not enough.
- The model is too simple.
- Training data is not cleaned and also contains noise in it.

Techniques to Reduce Underfitting:

- Increase model complexity.
- Increase the number of features, and perform the feature engineering.
- Remove noise from the data.
- Increase the number of epochs or increase the duration of training to get better results.

Overfit:

The condition when a model overly fit to the data and trying to satisfy each and every data point from the training data is called overfit.

Reasons for Overfitting:

- High variance and low bias.
- The model is too complex.

Techniques to Reduce Overfitting:

- Increase training data.
- Reduce model complexity.
- Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
- Use Ridge Regularization and Lasso Regularization.
- Use dropout for neural networks to tackle overfitting.

Good Fit:

The condition when a model fit to the data properly and perfectly identify the hidden actual pattern in the data is called good fit.

Gradient Descent

Gradient Descent Algorithm and Its Variants:

Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model. The primary set-up for regression models is to define a cost function (loss function) that measures how well the model predicts outputs on the test set. The goal is to find a set of weights and biases that minimizes the cost. One common function that is often used is the mean square root, which measures the difference between the actual value of y and the estimated value of y (the prediction).

Weight updating formula: $\omega_j := \omega_j - \alpha \frac{\partial}{\partial \theta_j} J(\omega_0, \omega_1)$

Cost function: $J(\omega_0, \omega_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\omega(x_i) - y_i]^2$

$$h_\omega(x_i) = \sum_{i=1}^{m} \omega_i x_i + \omega_0$$

Now we will do partial differentiation for above cost function with respect to θ_j. the main reason is to do partial differentiation is to minimize the cost function.

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} [h_\theta(x_i) - y_i]^2$$

$$= \frac{2}{2m} \sum_{i=1}^{m} [h_\theta(x_i) - y_i] \frac{\partial}{\partial \theta_j} [h_\theta(x_i) - y_i]$$

Now substitute $h_\theta(x_i)$ in above equation.

$$= \frac{1}{m} \sum_{i=1}^{m} [h_\theta(x_i) - y_i] \frac{\partial}{\partial \theta_j} [\theta_j x_{ji} + \theta_0 - y_i]$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y_i) x_{ji}]$$

Therefore θ_j will be

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y_i) x_{ji}]$$

Here j = Feature number
i = Labels in j feature
α = learning rate

Types of Gradient Descent

There are three types of Gradient Descent

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

Batch Gradient Descent:

This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y_i) x_{ji}]$$

Where x_{ji} Represents the j^{th} feature of the i^{th} training example (label of j^{th} feature). So, if m is very large, then the derivative term fails to converge at the global minimum.

Mathematical Derivation:

Gradient descent is an optimization algorithm that works by efficiently searching the parameter space, intercept () and slope() for linear regression. Now we will calculate the gradient decent function θ_j. While training the model, the model calculates the cost function which measures the Root Mean Squared error between the predicted value (predicted) and true value (y). The model targets to minimize the cost function. To minimize the cost function, the model needs to have the best value of θ_0 and θ_1. Initially model selects θ_0 and θ_1 values randomly and then iteratively update these values in order to minimize the cost function until it reaches the local minima. By the time model achieves the minimum cost function, it will have the best θ_0 and θ_1 values. Using these finally updated values of θ_0 and θ_1 in the hypothesis equation of linear equation, model predicts the value of x in the best manner it can.

$$\theta_j \; := \; \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0 , \theta_1)$$

$$h_\theta \,(x_i) = \theta_1 x + \theta_0$$

The above equation is cost function. $h_\theta \,(x_i)$ is hypothesis function of linear form being a predicted value of y. y_i actual value of y and m is number of labels or values in trained x. Differentiation $J(\theta_0 , \theta_1)$ with respect to θ_0.

$$\frac{\partial}{\partial \theta_0} J(\theta_0 , \theta_1)$$

$$= \frac{2}{2m} \sum_{i=1}^{m} [h_\theta \,(x_i)$$

$$- y_i] \; \frac{\partial}{\partial \theta_0} \; [\theta_1 x + \theta_0 - y_i]$$

$$= \frac{1}{m} \sum_{i=1}^{m} [h_\theta \,(x_i) - y_i]$$

$$\theta_0 \; := \theta_0 - \alpha \; \frac{\partial}{\partial \theta_0} \; J(\theta_0 , \theta_1)$$

$$\theta_0 \; := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} [h_\theta \,(x_i) - y_i]$$

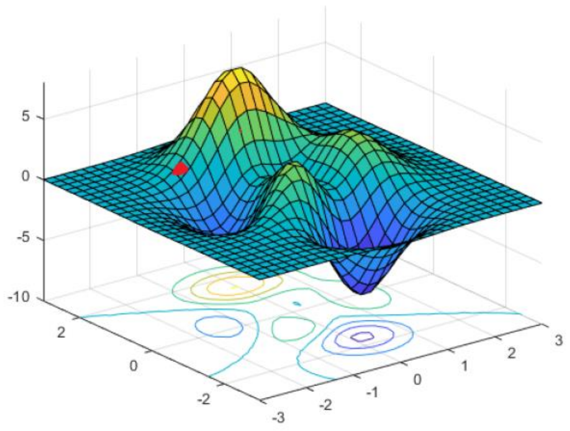


Fig 1: 3D diagram of cost function and weights and intercept point

Differentiation $J(\theta_0 , \theta_1)$ with respect to θ_1.

$$\frac{\partial}{\partial \theta_1} J(\theta_0 , \theta_1)$$

$$= \frac{2}{2m} \sum_{i=1}^{m} \; [h_\theta \,(x_i)$$

$$- y_i] \; \frac{\partial}{\partial \theta_1} \; [\theta_1 x + \theta_0 - y_i]$$

$$= \frac{1}{m} \sum_{i=1}^{m} \; [(h_\theta \,(x_i) - y_i) x_i]$$

$$\theta_1 = \theta_1 - \alpha \; \frac{\partial}{\partial \theta_1} \; J(\theta_0 , \theta_1)$$

$$\theta_1 \; := \; \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta \,(x_i) - y_i) x_i]$$

Like that if data have more than two independent features the θ_2, θ_3 will be.

$$\theta_2 \; := \; \theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta \,(x_{2i}) - y_i) x_{2i}]$$

$$\theta_3 \; := \; \theta_3 - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta \,(x_{3i}) - y_i) x_{3i}]$$

The equation is why partial differentiation applied on cost function with respect to θ_0 and θ_1. Because to reduce the cost function for reach global minima. The partial differentiation is used for minimize the mathematical functions for easy analysis. Initially all weights are taking as zero and find the cost function. At zero values of weight the slope of linear line is zero i.e. at this point cost function value is very high. Now the task is to reduce cost function by taking number of iterations. At each after all iterations the minimum cost function well be archived at that point will take weights of best fit line of that particular model.

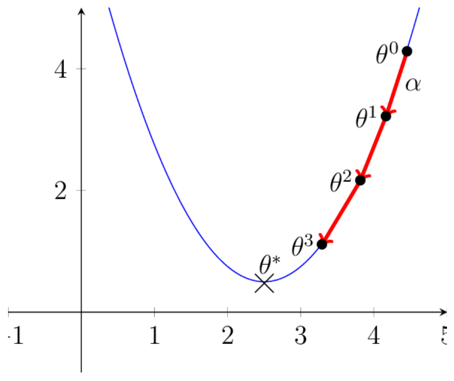


Fig 2: Reducing cost function

Stochastic Gradient Descent (SGD)

The word *stochastic* means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

$$\theta_j \ := \ \theta_j - \alpha\left[\hat{y}_p^i \ - y^i\right] x_j^i$$

So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples. In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that doesn't matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with significantly shorter training time.

One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm. This cycle of taking the values and adjusting them based on different

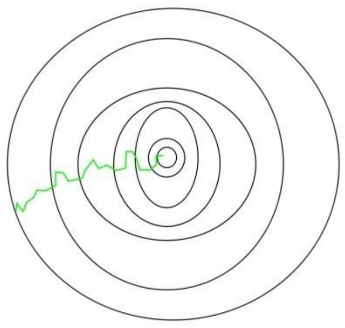
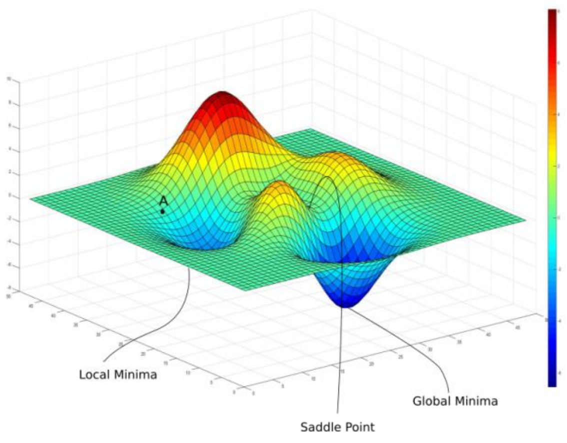


Figure 3: Path taken by SGD

parameters in order to reduce the loss function is called **back-propagation**.

The fig 2 is a 3D graph of cost function, weights, intercepting point. the fig1 is a top view of one global minima depth in fig 2. we can see the path of a point to reach global minima in fig 1. we take a point A on plain from independent variable x instead of taking all points to reach global minima.



Regression Model Validation

The model validation is a very important step after complete the model training. This evaluation matrix provides the statistical information of how much correctly fitted or trained model got generated. The evaluation is mainly depending on errors generated in model. Now let's understand what is the meaning of error and different types of regression evaluation matrices.

Residual or Error:

A residual is a measure of how far away an actual data point is vertically from the regression line. Simply, it is the error between an actual value and the predicted value.

$$\varepsilon = y - \hat{y}$$

ε = error,
y = actual value,
$\hat{y}$ = Predicted value

Types of matrices to evaluate a regression model:

1. Mean absolute error (MAE)
2. Mean square error (MSE)
3. Root mean square error (RMSE)
4. Mean absolute percentage error (MAPE)
5. R-Square (R2) score.
6. Adjusted R-Square score.
7. Residual plots.
8. Cross validation.

Mean Absolute Error:

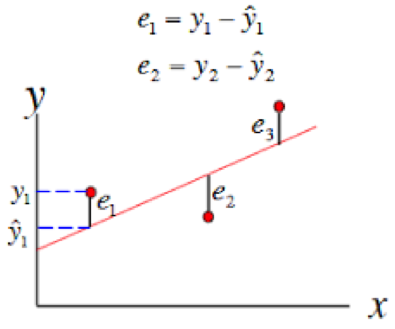
The Mean Absolute Error (MAE) is the average of all absolute errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Where:

n = Number of errors.
$|y_i - \hat{y}_i|$ = Absolute errors.
The steps to calculate MAE:

1. Find all of your absolute errors, $|y_i - \hat{y}_i|$
2. Sum up all.
3. Divide by the number of errors. For example, if you have 10 measurements, divide by 10.



Mean Square Error:

The Mean Square Error (MSE) is the average of all squares of errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where:

n = Number of errors.
$(y_i - \hat{y}_i)^2$ = Squares of errors.

The steps to calculate MSE:

1. Find all of your squares of errors, $(y_i - \hat{y}_i)^2$
2. Add them all up.
3. Divide by the number of errors. For example, if you had 10 measurements, divide by 10.

Root Mean Square Error:

The Root Mean Square Error ($RMSE$) is the root of the average of all squares of errors.

$$MAE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Where:

n = Number of errors.

$(y_i - \hat{y}_i)^2$ = Squares of errors.

The steps to calculate *RMSE*:

1. Squaring the residuals.
2. Finding the average of the residuals.
3. Taking the square root of the result.

Mean Absolute Percentage Error:

The Mean Absolute Percentage Error (*MAPE*) is the average percentage error that tells how much percentage of distance existed between predicted and original values.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Where:

n	= Number of fitted points,
A_t	= Actual value,
F_t	= Forecast / Predicted value.

The steps to calculate *MAPE*:

1. Calculate the percentage of distance between actual and predicted value.
2. Sum up all percentage errors.
3. Now divisible the total percentage aerobic number of data points (n).

R – Squared (R2) Score:

The R-squared error is a model goodness calculation, that explains how much percentage of good fitted the trained model than average model.

$$R^2 = 1 - \frac{sum\ squared\ regression\ (SSR)}{total\ sum\ of\ squares\ (SST)}$$

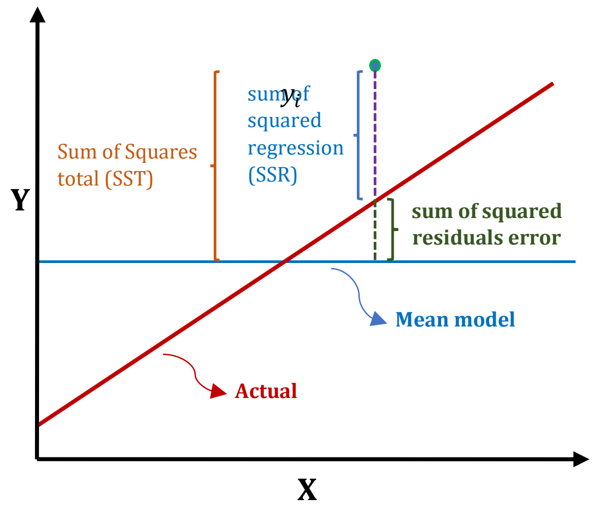
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_{mean})^2}$$

Where:

n	= Number of errors.
$(y_i - \hat{y}_i)^2$	= Squares actual model errors.
$(y_i - \bar{y}_{mean})^2$	= Squares mean model errors.

Here $\bar{y}_{mean}$ is the average or mean of all actual values (y_i). This y mean value will take as mean model. Which *mean model* has same y values as $\bar{y}_{mean}$ for all x values in mean model.

$$\bar{y}_{mean} = \frac{\sum_{i=1}^{n} y_i}{n}$$



Y
Sum of Squares total (SST)
y_i
sum of squared regression (SSR)
sum of squared residuals error
Mean model
Actual
X

Let's understate R2 score vary clearly in below:

- The horizontal blue line represents the avg /mean ($\bar{y}_{mean}$) of all actual values (y_i). It is represented as a mean model in the diagram.
- The sum of squared distance between the actual values and the prediction made by regression line is termed as *sum of squared regression error (SSR)*. Mathematically, SSR is represented as

$$SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ represents the prediction value and the y_i represents the actual value.

- The variation of the actual values from the mean or the horizontal line is called mean model error, this is also called *sum of squares total (SST)*. This variation is calculated as the sum of squared distance of individual points from the mean model.

$$SST = \sum_{i=1}^{n}(y_i - \bar{y}_{mean})^2$$

- The total variation of prediction (represented using regression line) from the mean is represented as sum of squared distance between the prediction and the mean. It is also termed as ***sum of squared error** (SSE)*. Mathematically, SSE is represented as the following where $\hat{y}_i$ represents the prediction and the $\bar{y}_{mean}$ represents the mean value.

$$SSE = \sum_{i=1}^{n} (\hat{y}_i - \bar{y}_{mean})^2$$

The Steps to Calculate R^2 Score:

1. Calculate SSR.
2. Calculate SST.
3. Sum up SSR / SST and subtract that from 1

Limits of R^2 Score:

- The R^2 score lies between -1 to 1.
- If R^2 is near to 1 then that is the good fitted model. Suppose $R^2 = 0.8934\ \%$ means 0.8934 % of the variation in the y values accounted by the x values.
- If R^2 is near to 0 then that is the bad fitted model. Suppose $R^2 = 0.2534\ \%$ means 0.2534 % of the variation in the y values is accounted for by the x values.
- If $R^2 = 1$ means all the variation in the y values accounted by the x values.
- If $R^2 = 0$ means none of variation in the y values accounted by the x values.
- The generated model is worse than average model If R^2 is negative value.

IMP Points:

- R^2 value is lies between 0 to 1 when SSR < SST. In this case $\frac{SSR}{SST} < 1$.
- R^2 value is lies between -1 to 0 when SSR > SST. In this case $\frac{SSR}{SST} > 1$.
- R^2 value is 0 when SSR = SST. Which means both mean model and actual model has equal errors. In this case $\frac{SSR}{SST} = 1$.
- R^2 value is 1 when SSR = 0. Which means actual model has no errors. In this case $\frac{SSR}{SST} = 0$.

Note: Based on above analysis R^2 inversely proportional to $\frac{SSR}{SST}$ ($R^2\ \alpha\ \frac{SST}{SSR}$) .

Over Fit with Useless Features Problem:

- R-square technique has a drawback is not recognize the overfitting problem in model when useless features added.
- Let's take a model with two independent features as showing below. And it gives R2 score as 90%. And assume features x_1 , x_2 are corelated with y.

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2$$

- Now, two more features x_3 , x_4 added to above model to increase the accuracy, then model become $y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4$. But x_3 , x_4 are not corelated to y, means these extra two features not impact anything on y. In this case x_3 , x_4 are useless features so R2 score should decrease, but R2 score still increasing like more than 90%. This is the main drawback in R2 score.
- Model always overfit when we add more useless features.
- There are two reasons of this problem. One is R2 score treated as all independent features correlated with dependent feature. Second is it can't recognize the overfitting of model with useless columns.
- we will use *adjusted R – Square* to avoid this problem

Adjusted R–Squared Score:

- Adjusted R^2 is the updated version of R^2.
- R^2 shows how well terms (data points) fit a curve or line. Adjusted R^2 also indicates how well terms fit a curve or line, but adjusts for the number of independent variables in a model.
- Adjusted R^2 will decrease if add more and more **useless** variables to a model. And it increases when add more **useful** variables.
- Adjusted R^2 always less than or equal to R^2.

Formula:

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(N - 1)}{N - K - 1}\right]$$

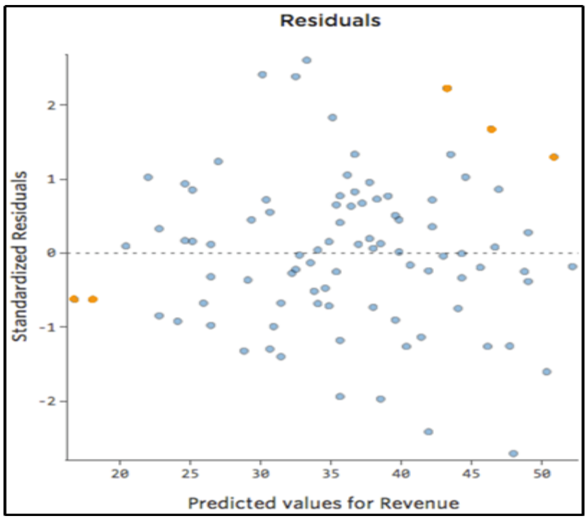
Where:

N = The number of points in your data sample.

K = Number of independent regressors, i.e. the number of variables in your model, excluding the constant.

Residual Plot Analysis:

Typical residual plot has the residual values on the Y-axis and the predicted values ($\hat{Y}$) on the x-axis. The below figure is a good example of how a typical residual plot looks like.

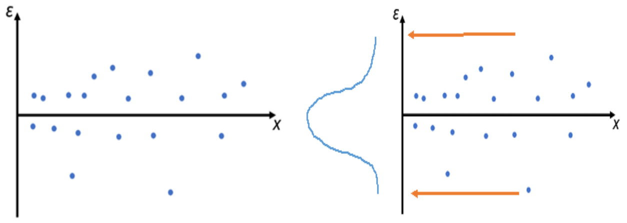


Characteristics of Good Residual Plots:

A few characteristics of a good residual plot are as follows:

1. It has a high density of points close to the origin and a low density of points away from the origin. Means Residuals should be *normally distributed.*
2. It is symmetric about the origin.
3. Residuals should be independent on predicted values. Means error should not form a pattern in the plot.
4. If the residuals exhibit a curved pattern or systematic deviations from the zero line, it indicates that the model may not capture the underlying relationship adequately.

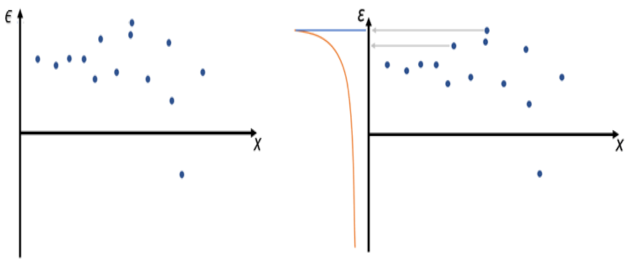
If you see the below figure is a good residual plot based on the above characteristics.



- As seen in right side figure, we end up with a normally distributed curve; satisfying the normality assumption of the residuals.
- It is symmetrically distributed through origin.
- It doesn't have any specific relation between residuals and predicted values that's why it doesn't form any pattern.

The below figure is a bad residual plot because:

- It shows many points far from the origin and few close to it. Additionally, the distribution of the residuals on the y-axis isn't normal.
- It did not symmetrically distribute through the origin.
- We can see a curve pattern in the right-side graph formed by residuals.



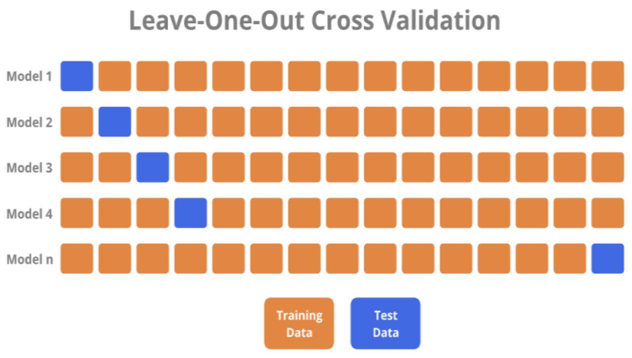
Cross Validation:

- Cross-validation technique used to understand the ***performance and generalization ability*** of a predictive model.
- It dividing the available dataset into multiple subsets to train and test the model on different combinations of data splits.
- Cross-validation allows for better utilization of the available data by using it for both training and evaluation purposes.

Some common cross-validation techniques include:

1. Leave one out cross validation
2. K fold Cross Validation
3. Stratified K fold Cross Validation
4. Time Series Cross Validation

Leave one out cross validation (LOOCV):



- In this technique each observation is considered as the validation/test set and the rest (N-1) observations are considered as the training set.
- Suppose if data has N rows, then first record is the test set and remaining records will be training set. this combination used for first iteration. For second iteration second record is the test set and remaining records will be training set. Like that it will create N number of validation splits.
- Each validation dataset will generate each model and collect all matrix values (accuracy, mean squared error, or area under the curve, etc..).
- Now we can take min, max, and average accuracy values.

Note: Here number of iterations, and number of evaluate models are equal to number of records in data.

Advantages of LOOCV are as follows:

- In this method, each observation is used for both training and validation, so there's no randomness. This means there's less variability in the results, regardless of how many times the method is run.

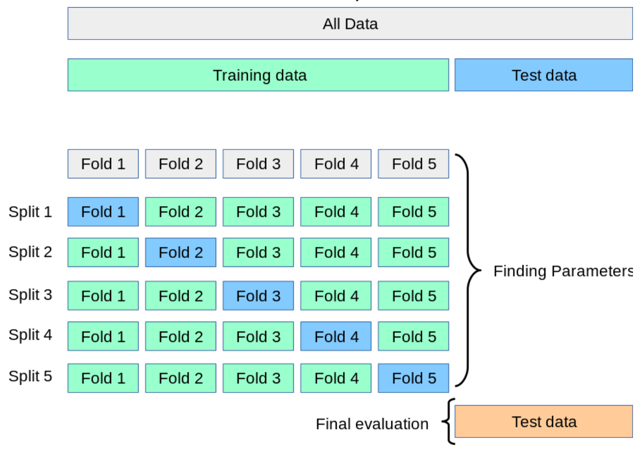
Disadvantage of LOOCV is as follows:

- Training the model N times leads to expensive computation time if the dataset is large.

K fold Cross Validation:

The LOOCV drawback is large number of iterations based on number of records. This drawback overcome in k-fold CV. Here the number of iterations based on selected K value instead of number of records.

- K-fold cross-validation approach divides the input dataset into **N groups (splits)** as shown in fig in figure. Again, each group divided into k parts are called as ***folds***.
- For each learning set (split), the prediction function uses k-1 folds, and the rest of the



folds are used for the test set. It's a very popular CV approach

The steps for k-fold cross-validation:

- Generate N value to divide the data into N groups. Each group will go for each iteration.

For each group:

- Divide each group into N / K folds.
- Take one-fold as the test data set. Use remaining fold as the training dataset
- Fit the model on the training set and evaluate the performance of the model using the test set.
- Repeat above 3-points N times for N groups.

Split the input dataset into K folds:

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1st iteration, the first fold is reserved for test the model, and rest are used to train the model. On 2nd iteration, the second fold is used to test the model, and rest are used to train the model. This process will continue until for all folds.

Advantages:

- Fast computation speed.
- A very effective method to estimate the prediction error and the accuracy of a model.

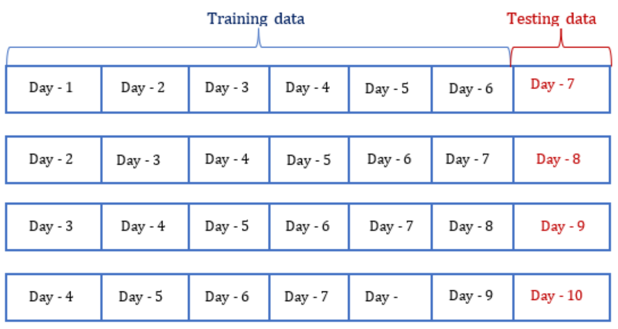
Disadvantages:

- Sometimes this CV technique fails when testing-fold has all recodes belongs to only one class in the classification task.
- A lower value of K leads to a biased model and a higher value of K can lead to variability in the performance metrics of the model. Thus, it is very important to use the correct value of K for the model (generally K = 5 and K = 10 is desirable).

Stratified K fold Cross Validation:

- This technique is similar to k-fold cross-validation with some little changes.
- This approach works on stratification concept, it is a process of re-arranging the data to ensure that each fold is a good representative of the complete dataset
- Which means each fold should have records belongs to all classes. To deal with the bias and variance, it is one of the best approaches.

Time series Cross Validation:



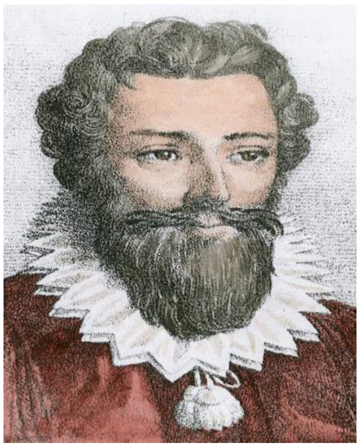
- In time series Cross validation, Training and testing data will divide based on the time or date.
- First it will generate some groups of data for each iteration. In Each group, one day data will be used as testing data set and remining data will be used as a training data set.
- For the second iteration day-1 data will be removed and tenure day data will be added at the new data will be taking as a testing data set.
- Like that these iterations will continue until test number of days.

Learning Curves:

Def: Learning curves is a graphical representations or plots that illustrate the relationship between the performance of a model and the amount of training data.

- Learning curves are used to analyze and evaluate the learning behavior and generalize behavior of a machine learning algorithm.
- Typically, learning curves display the training and validation error (or a performance metric) of a model as a function of the training set size. The x-axis represents the training set size, while the y-axis represents the model's error or performance metric.
- The curves often consist of two lines or curves: one representing the training error and the other representing the validation error.

Polynomial Regression


Francois Viete
(1540 – 23 Feb 1630)

The origins of polynomial equations can be traced back to ancient civilizations such as the Babylonians, who used numerical methods to solve polynomial equations as early as **1800 BC**. However, the modern notation and methods of polynomial equations were developed during the Renaissance by mathematicians such as **François Viète**, who is often credited with being the first to use letters to represent variables in algebraic equations.

Viète was a French mathematician who lived in the 16th century and made significant contributions to the development of algebra. He introduced the idea of using letters to represent variables in equations, which made it easier to generalize results and solve more complex problems. **Viète** also developed a method for solving polynomial equations known as **Viète's** formula, which involves finding the roots of the equation by solving a system of linear equations.

Other mathematicians such as **René Descartes** and **Isaac Newton** also made significant contributions to the development of polynomial equations during the Renaissance and Enlightenment periods. Today, polynomial equations are used extensively in fields such as mathematics, physics, engineering, and economics to model complex relationships between variables.

The first known use of polynomial regression was by the mathematician **Carl Friedrich Gauss**, who used it to analyze astronomical data in the early **1800s**. Gauss developed the method of least squares to fit a polynomial curve to a set of data points, and this method is still widely used today.

In the 19th century, the method of polynomial regression was further developed by other mathematicians, including **Adrien-Marie Legendre** and **Pierre-Simon Laplace**. Legendre developed the method of least squares independently of Gauss and applied it to the analysis of data from the French astronomical survey, which aimed to measure the shape of the Earth. Laplace also used polynomial regression to analyze astronomical data, and his work led to the development of the concept of error analysis.

In the 20th century, polynomial regression became a standard tool in statistics and data analysis. The advent of computers made it easier to perform polynomial regression on large datasets, and the technique was widely used in fields such as economics, engineering, and physics.

Today, polynomial regression remains a popular method for modeling complex relationships between variables in many fields. With the advent of machine learning and deep learning techniques, polynomial regression is often used as a building block for more complex models that combine polynomial terms with other nonlinear functions.

Introduction:

Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an **nth degree polynomial**. In other words, it involves fitting a polynomial function to a set of data points in order to make predictions.

- Polynomial regression can be useful when the relationship between the independent and dependent variables is not linear.
- Polynomial regression helps to capture more complex relationships and make better predictions.
- The simple way perform polynomial regression is add powers to each feature and then train the linear model.

Simple Working:

1. ***Choose degree of polynomial:*** First, we need to choose the degree of the polynomial. This determines the complexity of the model and how well it fits the data. A higher degree polynomial will be more complex and will better fit the data, but may also be more prone to overfitting.
2. ***Generate possible polynomials:*** In this stage, the powers add to the features based on selected degree. And next generate possible combination of polynomial equations.

   For example, if we have two features x_1 and x_2 the combinations will be:

$$x_1 \; x_2^2 \rightarrow h(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2^2$$
$$x_1^2 \; x_2 \rightarrow h(x) = \omega_0 + \omega_1 x_1^2 + \omega_2 x_2$$
$$x_1^2 \; x_2^2 \rightarrow h(x) = \omega_0 + \omega_1 x_1^2 + \omega_2 x_2^2$$

etc ……..

The number of possible combinations of the polynomial equations can be find based on degree and number of the features:

$$\text{number of possible combinations} = \frac{(n-d)!}{d!\,n!}$$

Here: n = Number of features
d = Selected degree

3. ***Training:*** It will select one best suitable polynomial equation and rain with the data.
4. ***Make predictions:*** Finally, polynomial model ready to make predictions about the relationship.

Mathematical Intuition:

- In Polynomial regression, the original features are converted into Polynomial features of required degree (2, 3, … ,n) and then modeled using a linear model.
- It is also called the special case of *Multiple Linear Regression* in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a non-linear dataset, then it will produce a drastic output. Due to the loss function will increase, the error rate will be high, and accuracy will be decreased.
- So, for such cases, where data points are arranged in a non-linear fashion is nothing but Polynomial Regression model.
- Now let's observe the polynomial regression hypothesis equation.

$$h(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2^2 + \omega_3 x_2^3 + \cdots$$

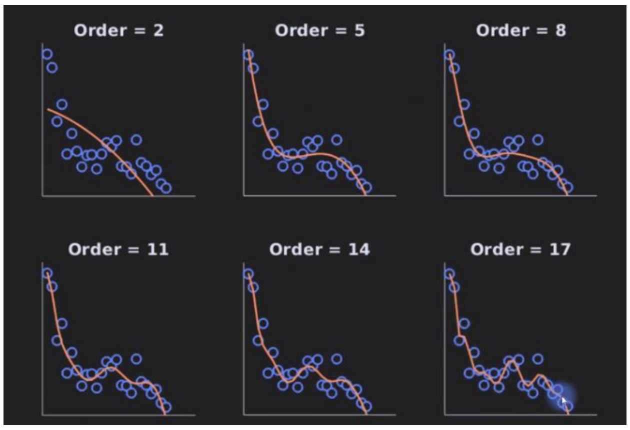


Figure 4 : Polynomial curves changes with order.

- Here the same independent feature can take multiple times as a new feature by squaring or cubing (x_2^2, x_2^3).
- The degree of this polynomial model is 3, because that is the highest **order** of this polynomial equation.
- The polynomial equation line has more curve when the order of the polynomial increases. We can observe these curve changes in figure one.

Advantages:

1. Polynomial regression is independent of the size of the data set.
2. Non-linear problems are solved with good accuracy.
3. It gives the best approximation of the correspondence between the output and explanatory (independent) variables when compared with LR.
4. A large number of functions can be fit under it.

Disadvantages:

1. Polynomial regression can easily overfit the data, especially when the degree of the polynomial is high.
2. Fitting a high-degree polynomial equation can be computationally expensive, especially for large datasets, as the number of parameters to estimate increases rapidly with the degree of the polynomial.
3. The estimates of the coefficients can be unstable, especially for high-degree polynomials, due to multicollinearity or other issues related to numerical precisio

Code:

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Generate some data
np.random.seed(0)
x = np.random.rand(100, 1) * 10  # Random data between 0 and 10
y = 2 - 3 * x + 0.5 * x**2 + np.random.randn(100, 1) * 5  # Quadratic
relationship with noise

# Transform the features into polynomial features
degree = 2
poly_features = PolynomialFeatures(degree=degree)
x_poly = poly_features.fit_transform(x)

# Fit the polynomial regression model
model = LinearRegression()
model.fit(x_poly, y)

# Predict using the model
y_pred = model.predict(x_poly)

# Calculate mean squared error
mse = mean_squared_error(y, y_pred)
print(f'Mean Squared Error: {mse}')
```

Regularized Linear Models

- There is no single method to address the overfitting problem in linear regression.
- In polynomial regression, one common method to mitigate overfitting is by reducing the dimensionality of the model.
- This overfitting problem can be reduced by using regularized models in linear regression.

The regularized linear models are:

1. Ridge regression (L2)
2. Lasso regression (L1)
3. Elastic net regression

The overfitting problem comes in two cases when cost function is equal to zero or similar to zero.

Case -1: When cost function = 0

Case -2: When cost function ≈ 0

Ridge Regression (L2):

- Ridge regression, also known as **Tikhonov regularization**, is a technique used to analyze *multiple regression* that suffer from multicollinearity (when independent variables are highly correlated).
- In such cases, *ordinary least squares (OLS)* regression estimations (slopes/ coefficients) can become unstable and have high variance, leading to unreliable predictions.
- Ridge regression addresses this issue by adding a regularization term to the loss function.

Note: *Ordinary Least Squares* ***(OLS)*** *is a common technique for estimating coefficients of linear regression equations.*

OLS is also called as cost function / loss function.

Mathematical Formulation:

$$cost\ function\ (cf) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Ridge regression modifies the standard OLS loss function by adding a penalty term proportional to the sum of the squares of the coefficients. The loss function becomes:

$$ridge\ cf = cf + \alpha\sum_{j=1}^{p}(slope)^2$$

$$= \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{p}(slope)^2$$

where:

y_i = Actual value of the target variable.
$\hat{y}_i$ = Predicted value of the target variable.
$slope$ = are the coefficients of the model.
α = regularization parameter (also known as the penalty term).

How it overcome overfit problem: The generalized model changes the slope of the line when model got overfit to the training data.

Changing line slope: The slope of the line can be modified by adjusting the cost function. However, directly changing the cost function value is not possible when the model is already overfitted, as the cost function value is zero or minimal.

How it updates the cf: In Ridge Regression, we update the cost function by adding a regularization term. This prevents the cost function from reaching zero, even when the model might be overfitting.

What is the use to update cf: The added regularization term helps the model to continue iterating towards the global minimum, reducing the risk of overfitting by penalizing excessively large coefficients.

Regularization Parameter (α):

- The parameter α controls the strength of the regularization. When $\alpha = 0$, ridge regression reduces to OLS. If α increases, which shrinks the slopes towards zero.
- Choosing an appropriate α value is crucial and can be done using techniques such as cross-validation.

Note: If α = 0 then Ridge Regression is just like a Linear Regression. If α is very large then all weights are equal to zero at this point the linear line is just like a flat line and cost function value is very high. These two conditions are useless in Ridge that means we should maintain α as not too much high and not zero.

An Important note is to scale the data (using StandardScaler) before performing the Ridge Regression Because it's very sensitive to scale of the input features.

Lasso Regression (L1):

- Lasso Regression is similar to Ridge Regression, but key difference is, it handles feature selection also.
- In Lasso Regression, the cost function includes a regularization term that can removes some of the coefficients which are becomes zero.

$$lasso\ cf = cf + \alpha \sum_{j=1}^{p} |slope|$$

- This means if any slope (coefficient) of any feature becomes zero, that feature will remove from the model.
- This feature selection property allows Lasso to train a model with most important features.

Elastic Net:

Elastic net is the combination of both ridge and lasso regression.

It reduces overfitting problem and performs feature selection also.

$$EN\ cf = cf + \alpha \sum_{j=1}^{p} (slope)^2 + \gamma \sum_{j=1}^{p} |slope|$$

Note: Here α and γ are different hyperparameters.

Ridge Regression Code:

```python
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Initialize Ridge regression model with alpha (lambda) = 1.0
ridge = Ridge(alpha=1.0)

# Fit the model to the training data
ridge.fit(X_train, y_train)

# Predict on the test set
y_pred = ridge.predict(X_test)
```

Lasso Regression Code:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
```

```python
# Initialize Lasso regression model with alpha (lambda) = 1.0
lasso = Lasso(alpha=1.0)

# Fit the model to the training data
lasso.fit(X_train, y_train)

# Predict on the test set
y_pred = lasso.predict(X_test)
```

Elastic Net Code:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

# Initialize Lasso regression model with alpha (lambda) = 1.0
lasso = Lasso(alpha=1.0)

# Fit the model to the training data
lasso.fit(X_train, y_train)

# Predict on the test set
y_pred = lasso.predict(X_test)
```

CLASSIFICATION

Logistic Regression

- Logistic Regression *(logit regression)* is a *classification algorithm* working based on Linear Regression.
- Commonly Logistic Regression estimate the probability of current data point belongs to which class at given input (independent variable) value of x.
- If probability is greater than 50% then the data point belongs to a specific class, if it's less than 50% then the data point not belongs to that particular class.
- We use sigmoid function in Logistic Regression. The sigmoid function is represented with $\sigma(y)$. This sigmoid function graph look likes *S – curve.* And it bounds in between 0 to 1 only.

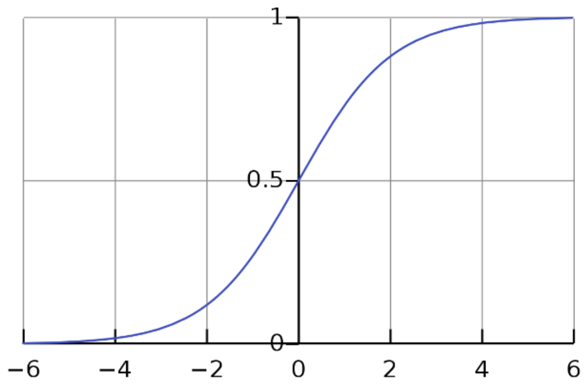


Fig 1: sigmoid function or hypothesis function graph

$$\sigma(y) \text{ is } 1 \text{ when } y \rightarrow \infty$$

$$\sigma(y) \text{ is } 0 \text{ when } y \rightarrow -\infty$$

NOTE: If data classifying as more than two classes then that type of logistic regression is called as Multinomial Logistic Regression.

$$\hat{p} = h_\omega(x) = \sigma(y) = \frac{1}{1 + e^{-y}}$$

Here sigmoid function is hypothesis function, and y is in terms of x. $\hat{p}$ is the probability. Now let's focus on y.

$$Y = \omega^T X$$

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_m \end{bmatrix} \quad X = \begin{bmatrix} 1 & 1 & \dots\dots & 1 \\ x_{11} & x_{12} & \dots\dots & x_{ni} \\ x_{21} & x_{22} & \dots\dots & x_{ni} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots\dots & x_{mi} \end{bmatrix}$$

For example, if calculate a predicted value of i^{th} label then the z value is

$$y_i = \omega_0 + \omega_1 x_{1i} + \omega_1 x_{2i} + \cdots \dots\dots. + \omega_m x_{mi}$$

$$\sigma(\omega^T X) = \frac{1}{1 + e^{-(\omega^T X)}}$$

The predicted value y will be conformed based on probability given by $\hat{p}$. Now see that predicted value $\hat{y}$ based on probability $\hat{p}$.

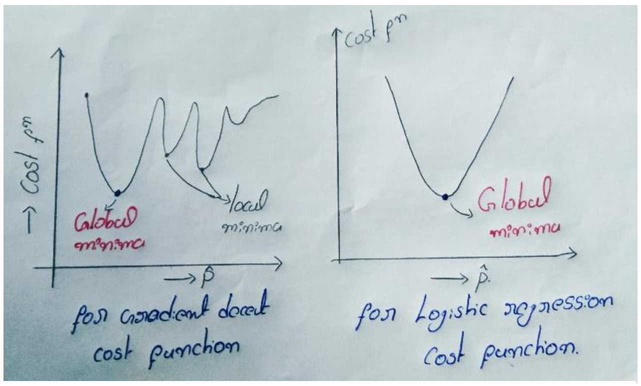
$$\hat{y} = \begin{cases} 0 \ \ if \ \ \hat{p} < 0.5 \\ 1 \ \ if \ \hat{p} \geq 0.5 \end{cases}$$

$$\sigma(\omega^T X) \geq 0.5 \ \text{ when } \ (\omega^T X) \geq 0$$

$$\sigma(\omega^T X) < 0.5 \ \text{ when } \ (\omega^T X) < 0$$

Remember, here $\omega^T X$ is nothing but y. And y is represented as z in fig- 2.

Cost function in Logistic Regression:


→ Cost f^n
Global minima
local minima
→ p̂
for Gradient decent cost function
↑ Cost f^n
Global minima
→ p̂
for Logistic regression Cost function.
Figure 2: Cost function graphs in GD and Logistic regression.

- Every model has some error after training, so we should reduce that error. We can use Gradient decent for reduce the error by updating the weights.

- Here we need new cost function for logistic regression. Because, Linear regression cost function is not useful.
- Local minima is main cause for not using of Linear regression cost function. Let's see in-depth on this point now.
- If you observe the graph-2 has many local minima points. In gradient decent it always looking for local minima point. In this case it may not reach lowest minima point rather it reach any one of the local minima. But actually, we need lowest minima point of cost function.
- This is the main problem in gradient decent, this logistic regression cost function avoids this problem.
- Now see the cost function in logistic regression for finding the loss and for updating the weights ω.

$$J(\omega) = \frac{1}{2m}\sum_{i=1}^{m}[\,\mathrm{cost}(h_\omega\,(x_i, y_i))]$$

$$\mathrm{cost}(h_\omega\,(x_i, y_i)) = \begin{cases} -\log(\hat{p}) & \text{if } \hat{y} = 1 \\ -\log(1-\hat{p}) & \text{if } \hat{y} = 0 \end{cases}$$

- In Logistic Regression log function is used as cost function. The main reason of using log function is, it has very best characteristics in between 1 to 0 values.
- Let's see in-depth on this log function now. Generally, if $\hat{p}$ value is 1 or 0 then we strongly saying that not has any error. So, our cost function should show the error as 0 when $\hat{p}$ is either 1 or 0.
- We can understand the value of $-\log(\hat{p})$ is 0 at $\hat{p}$ is 1 and it is increasing when $\hat{p}$ reaches to 0.5 in positive instance. Function $-\log(1-\hat{p})$ value is 0 at $\hat{p}$ is 0 and it is increasing when $\hat{p}$ reaches to 0.5 in negative instance.
- Here we using two cost functions for positive and negative instance because $-\log(\hat{p})$ function is not suitable for negative instance. If you observe the $-\log(0)$ value is undefinable value.
- Actually, at predicted value 0 the cost function must be zero in negative instance. That's why mathematicians gave another best cost function for negative instance. For this reason, Logistic regression have two cost functions.

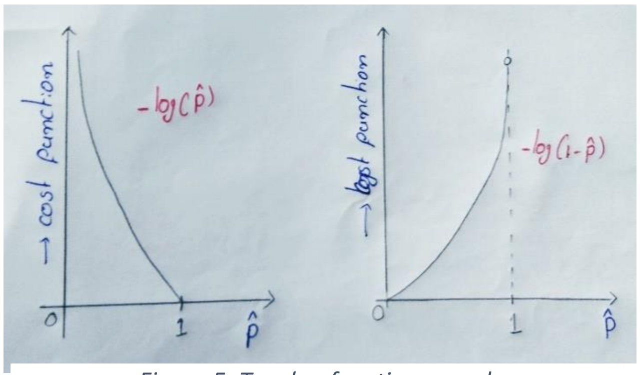


Figure 5: Two log functions graphs.

- Now observe the graph representation of two cost functions. Now the problem is two functions are as separate but we need to combined two cost functions for evaluate the cost function value.
- After combine a new and final cost function will get shown in below. Graph is also changing after combined two functions.

$$J(\omega) = -\frac{1}{m}\sum_{i=1}^{m}[\,y^{(i)}\,\log(\hat{p}^{(i)}) + (1 - y^{(i)})\log(1-\hat{p}^{(i)})\,]$$

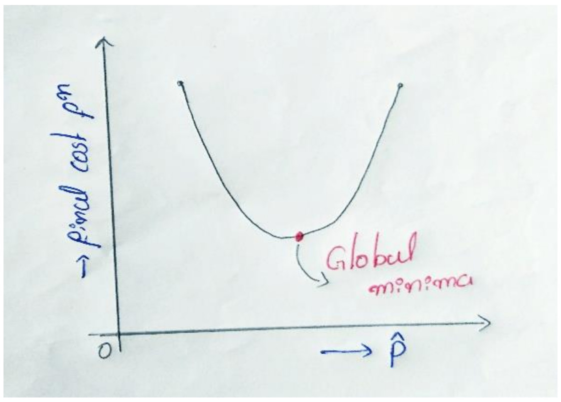


Figure 6: Final cost function graph.

- After combined two cost functions the graph looks like parabola curve. The main good thing is it have only one minima point that considered as global minima. It hasn't any local minima so no problem about getting local minima.
- Now the bad news is this methodology don't have formulization for finding weights (θ). But good news is we can apply gradient for finding weights. The formula as shown in below.

Code:

```python
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print the model's coefficients and intercept
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
```

Support Vector Machine (SVM)

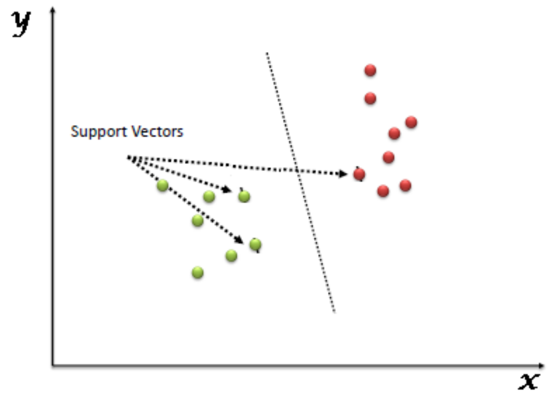
- Support Vector Machine (SVM) is a powerful supervised machine learning algorithm which can perform linear, nonlinear classification, regression. which aims to minimize the number of misclassification errors directly
- It is mostly suited for complex (n-dimensional) classification problems with low or medium size of datasets only.
- SVM automatically detect outliers. No need to remove explicitly removing outliers from data.
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.
- Many terms are involving in SVM model, Now let's see those terms.

Hyper Plane:

Which plane separate the two classes in n-dimensional space that plane is called hyper plane. Sometimes we dealing with 2D space only at that time we call it as a line. Here also we take 2D only because simple understanding concept.

$W^T.X + b = 0$ .... Hyper plane

To understand support vector machines, we must understand hyperplanes. Normally, a hyperplane is an (n – 1) subspace in an n-dimensional space. While that sounds complex, actually it is pretty simple. For example, if we wanted to divide a 2D space, we'd use a one-dimensional hyperplane (i.e., a line). If we wanted to divide a three-dimensional space, we'd use a two-



dimensional hyperplane (i.e., a flat piece of paper or a bed sheet).

Decision Boundaries:

Decision boundary is a boundary line that create a region space to hyperplane on both sides. We can see both positive and negative decision boundaries in fig-3 with dotted line representation. Those are represented as linear equating because this graph is related to linearly separable problem.

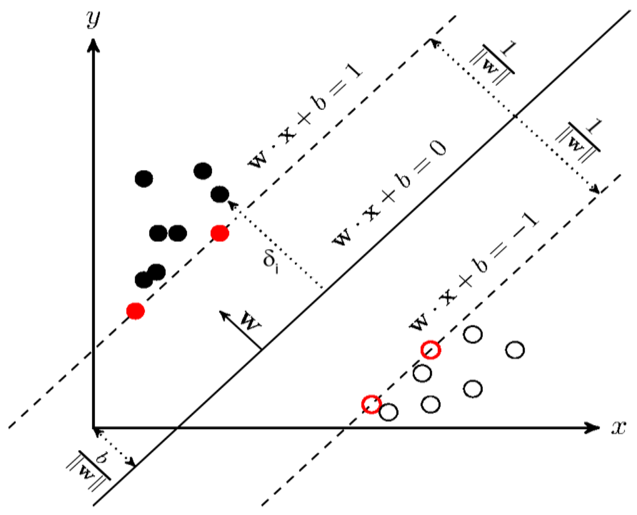


Fig-2: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

$W^T.X + b = 1$ .... +ve decision boundary

$W^T.X + b = -1$ .... -ve decision boundary

Marginal distance (or) margin:

The distance between two decision boundaries is called marginal distance or margin is denoted by M. We should always maintain high margin to get best accuracy.

Support Vectors:

Winch data points are very near to the hyper plane from both classes on both sides is called support vectors as show in fig-1. The support vectors are supporting for drawing decision boundaries of hyper plane. Support vectors may exist many numbers.

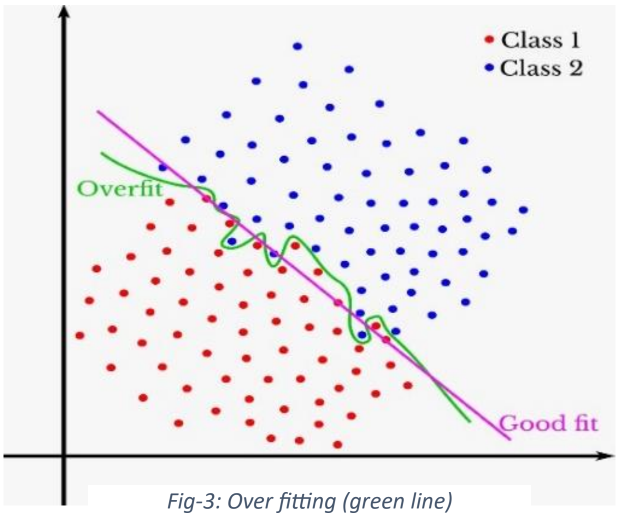


Fig-3: Over fitting (green line)

- The main confusion is why we need SVM? The main expectation from SVM is to reduce over fitting. We can see the over fitting in fig-2 diagrammatically.
- By maintain some distance between line and class over fitting problem arise. That distance will be created by drawing decision boundaries on both sides of hyper-plane we can see in fig-3. Decision boundaries are always parallel to the hyper-plane.
- It is very sensitive to large size of dataset. If number of labels increasing then data points is also increasing in this case data points may exist beside of hyper plane. But our aim is to maintain high margin but here it's not possible that's why SVM not suitable with large data sets.

Note: Please note a point, our aim is not creating a hyper plane rather we creating generalized model with high margin.

Mathematics behind SVM

Now we learn the mathematics behind SVM. A simple 2D problem taking for simple explanation. This mathematical explanation divided into 3 levels for simple understand.

Level -1:

In first level will learn about decision boundaries. We already said as positive and negative decision boundaries. Let's observe mathematically what is this positive and negative meaning.

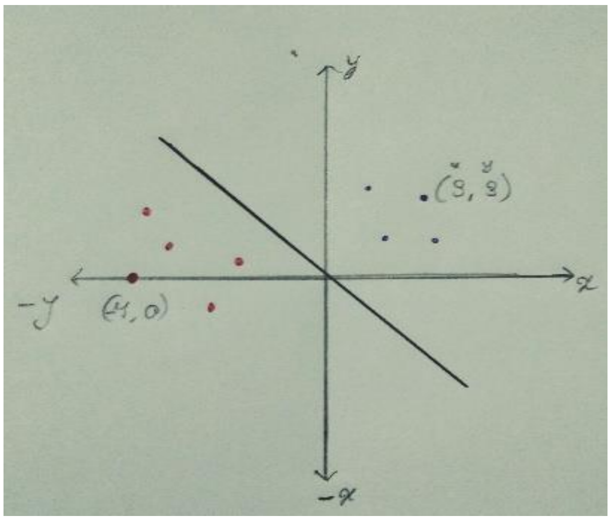


Fig-4: Linearly separable.

Take a linear equation as a hyper plane because we are taking linearly separable classification problem for simple understand (fig-4).

$$W^T.X + b = 0 \text{ ………………. } (1)$$

In fig-4 hyper plane intercept the y-axis at point. And take two points from above and below the hyper plane. Slope of this hyperplane is -1.

$W^T = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$, $X_1 = [-4 \quad 0]$, $X_2 = [3 \quad 3]$, $b = 0$

With $X_1 = [-4 \quad 0]$ *(below the line)*	*With* $X_2 = [3 \quad 3]$ *(above the line)*
$\hat{y} = W^T.X + b$ $\hat{y} = \begin{bmatrix}-1\\0\end{bmatrix}.[-4 \quad 0] + 0$ $\hat{y} = 4$	$\hat{y} = W^T.X + b$ $\hat{y} = \begin{bmatrix}-1\\0\end{bmatrix}.[3 \quad 3] + 0$ $\hat{y} = -3$

- From the above calculation we strongly say that any point above the negative slope plane is negative and below is positive.
- This positive and negative properties are reverse when plane have positive slope (fig-2). That means any point above the positive slope plane is positive and below is negative. You can see the positive slope plane in fig-2.
- The decision boundaries are taking from nearest point of the hyper plane, if that nearest point is above the negative slope plane then that decision boundary is negative. If it is below then decision boundary is negative. Vice versa for positive slope.

$W^T.X_1 + b = -1$ .... -ve decision boundary

$W^T.X_2 + b = 1$ .... +ve decision boundary

Level -2:

In level two the task is to calculating margin. Now subtract two decision boundaries for calculating the distance between them

$$W^T.X_1 + b = -1$$
$$W^T.X_2 + b = 1$$
$$(-) \quad (-) \quad (-)$$

$$W^T(X_1 - X_2) = -2 \text{ .... } (2)$$

Let's take - common from equation (2)

$W^T(X_2 - X_1) = 2$

Now we need $(X_2 - X_1)$ because that is margin distance. We can't directly remove W^T. So that's why we using norm of $\|W\|$.

$$\frac{W^T}{\|W\|}(X_2 - X_1) = \frac{2}{\|W\|}$$

Here $\frac{W^T}{\|W\|} = 1$ now

$$(X_2 - X_1) = \frac{2}{\|W\|}$$

$$\text{Margin} = \frac{2}{\|W\|}$$

Let's generate in verse term for margin

$$L = \frac{1}{Margine} = \frac{\|W\|}{2}$$

Note: We always looking for minimum value of L. If L is minimum then marginal distance will increase. whenever we get max margin at that point hyper plane will be fixed. That is more generalized model.

Level -3:

In this level we learn about how to check our model is perfectly divide classes or not. Remember we taking negative slope hyper plane.

$$\hat{y} = \begin{cases} 1 & if\ W^T.X + b \geq 1 \\ -1 & if\ W^T.X + b \leq -1 \end{cases}$$

Whenever if we do a classification using SVM every point must and should satisfy below equation

$$\hat{y}_i.(W^T.x_i + b) \geq 1$$

If you observe above equation, $W^T.x_i + b$ multiply with $\hat{y}_i$. In first case $\hat{y}_i$ is positive and $W^T.x_i + b$ is also positive finally will get positive value when multiply together. In second case $\hat{y}_i$ is negative and $W^T.x_i + b$ is also negative finally will get positive value when multiply together. That means at the both the cases will get positive value after apply SVM. If any observation is negative then that point have error and we strongly say that point is not related to that presented class.

Note: $\hat{y}_i \cdot W^T \cdot x_i + b$ should always positive if not that model is not 100 % stable. It may contain some error observations.

Hard Margin SVM

- What was learned in level-3 is comes under hard margin. In real scenario hard margin is not preferable because it has mainly two issues.
- First, it only works if the data is linearly separable, it can't work with linearly inseparable data (fig-5).

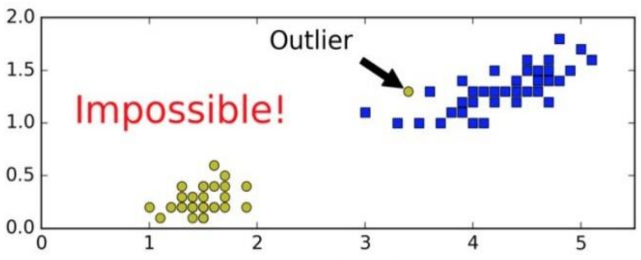


Fig-5: linearly inseparable problem

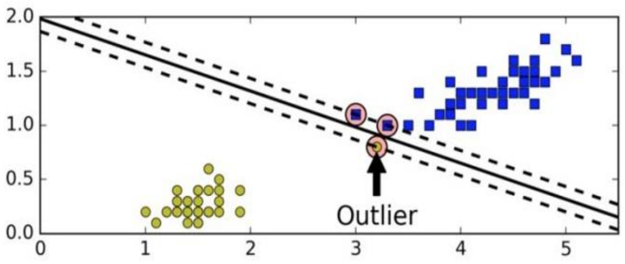


Fig-6: Outlier problem

- Reason for above problem is, hard margin SVM trying to build decision boundaries without allowing a single error. Because it should satisfy $\hat{y}_i \cdot W^T \cdot x_i + b \geq 1$ this equation. If any mismatching occur this condition not satisfy. Fixing decision boundaries without errors is not possible in linearly inseparable data as shown in fig-5.
- Second, it is quite sensitive to outliers. Figure 6 shows with just one additional outlier, it is impossible to find a hard margin, and it will probably not generalize as well.
- Whenever outlier see by SVM in hard margin it not thinking about another point after the outlier for fixing decision boundary. Because if it misses one outlier (again $\hat{y}_i \cdot W^T \cdot x_i + b \geq 1$ is not satisfy) that point is comes under the error in SVM point of view.
- Hard margin SVM thinks whenever it shows the nearest point to the hyperplane in the class it will fix decision boundary at that point. But actually, another decision boundary existed with high margin by allowing some errors.
- In general, always using Soft Margin SVM not hard margin SVM.

Soft Margin SVM

Note: This idea is simple, it allows SVM to make a certain number of mistakes and keep margin as wide as possible so that other points can still be classified correctly. This can be done simply by modifying the objective of SVM.

- To avoid above hard margin problems by soft margin. As mentioned earlier, almost all real-world applications have data that is linearly inseparable.
- In rare cases the data is linearly separable, we might not want to choose a decision boundary that perfectly separates the data to avoid overfitting.
- There are many data points are over lapping and some observations crossed decision boundaries and some observations cross the main hyper plane.
- If some observations are mismatch like this its ok in soft margin SVM don't change hyperplane position.
- In this condition we can't calculate max margin directly. That margin edgiest by below mathematical equation.

$$(w^* ,\quad b^*) = \frac{\|W\|}{2} + C \sum_{e=1}^{C} \xi_i$$

Here C = number of errors (hyper parameter)
ξ_i = error value

By using above equation, we updating weights continually up to we get most perfect max margin. Margin and errors can manage by adjusting hyper parameter C

$$if\ C \xrightarrow{small}\ (margin\ is\ high\ and\ vilations(errors)\ also\ high)$$

$$if\ C \xrightarrow{Large}\ (margin\ is\ small\ and\ vilations(errors) also\ small)$$

Condition term in soft margin is

$$\hat{y}_i\,.\,(W^T\,.\,x_i + b)\ \geq 1 - \xi_i$$

If data point is in correct place $\boldsymbol{\xi_i = 0}$ **and** $[\,\boldsymbol{\hat{y}_i\,.\,(W^T\,.\,x_i + b)}\,] \in \boldsymbol{(+ve)}$ **and** $\boldsymbol{(1 - \xi_i) = 1}$

If data point is on hyperplane $\boldsymbol{\xi_i = 1}$ **and** $[\,\boldsymbol{\hat{y}_i\,.\,(W^T\,.\,x_i + b)}\,] \in \boldsymbol{0}$

If data point not in correct pla $\boldsymbol{\xi_i > 1}$ **and** $[\,\boldsymbol{\hat{y}_i\,.\,(W^T\,.\,x_i + b)}\,] \in \boldsymbol{(-ve)}$ **and** $\boldsymbol{(1 - \xi_i)} \in \boldsymbol{(-ve)}$

SVM Classifier Code:

```python
# Import necessary libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize the SVM model for classification
svm_classifier = SVC(kernel='linear')  # You can also try 'rbf', 'poly', etc.

# Train the model on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = svm_classifier.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Classification Accuracy: {accuracy * 100:.2f}%')
```

SVM Regression Code:

```python
# Import necessary libraries
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Initialize the SVM model for regression
svm_regressor = SVR(kernel='rbf')  # 'linear', 'poly', etc., can also be used

# Train the model on the training data
svm_regressor.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = svm_regressor.predict(X_test)

# Evaluate the model's performance using Mean Squared Error
```

```
mse = mean_squared_error(y_test, y_pred)
print(f'Regression Mean Squared Error: {mse:.2f}')
```

Polynomial Kernel

In machine learning, kernel methods are a class of algorithms for pattern analysis. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

Adding polynomial features is simple to implement and can work great with all sorts of Machine Learning algorithms (not just SVMs), but at a low polynomial degree it cannot deal with very complex datasets, and with a high polynomial degree it creates a huge number of features, making the model too slow.

Code:

```
# Import necessary libraries
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline

# Define a pipeline that standardizes the data and applies a polynomial kernel SVM
# Standardize the features (mean=0, variance=1)
# Polynomial kernel SVM with degree 3 and regularization parameter C=5
poly_kernel = Pipeline([
    ('std_scaler', StandardScaler()),
    ('svm_clf', SVC(kernel='poly', degree=3, C=5))])


# Train the model using the training data
poly_kernel.fit(x_train, y_train)

# Make predictions on the testing data
y_pred = poly_kernel.predict(x_test)

# Calculate and print the accuracy of the model
accuracy = accuracy_score(y_pred, y_test) * 100
print(f'Accuracy: {accuracy:.2f}%')
```

Decision Tree

- **Decision Trees (DT)** are used in statistics, data mining, and machine learning to predict the value of a target.
- It looks like tree diagrams, showing all possible ways to get the target value.
- DTs work by finding the right path at each step (node) and following it to the end of the tree to predict the target based on input.
- DT can be used for both classification and regression problems. In classification it takes decision as True or False (or) Yes or No. In regression it decided the next possible number in a series of data.

Terminology in DT

Root Node
Branch
Node
Leaf Node

The basic pattern of decision tree is shown in fig-1.

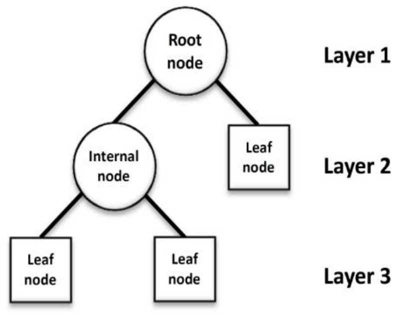


Fig- 8: Decision Tree parts.

Decision Tree types

✓ Classification Tree
✓ Regression Tree
✓ Classification and Regression Tree (CART)
✓ Decision Stream
✓ Boosted Trees
✓ Bootstrap ……

Entropy

Entropy measures imparity disorder or uncertainty in a bunch. We should need to measure the imparity of split to get best DT. The level of entropy is always between 0 to 1

$$0 \leq Entropy \leq 1$$

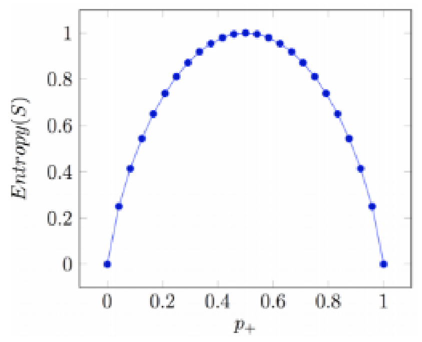


Fig- 7: Entropy level graph.

- If the entropy is equal to 1 then that is worst split.
- The high imparity data will not get best values. And also, the decision going on up to end of the tree but output is not available.
- For example, if we found right way by getting low entropy then we will get perfect output. Incase if chose a wrong way which has high entropy then will get wring output.
- Entropy is very important to choose correct path.
- The formula for calculating entropy at a particular class as shown in below.

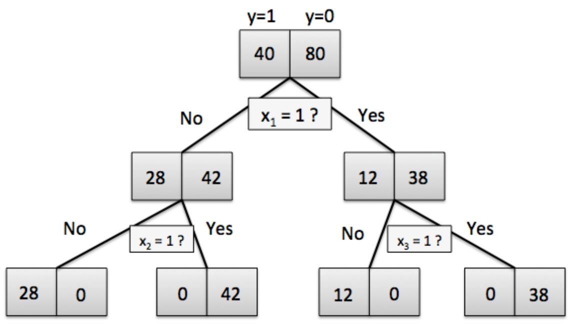


Fig 9: Decision tree for entropy calculation.

$$H(s) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

p_+ = % of positive class.
p_- = % of negative class.
S = subset of training examples.

$$p_+ = \frac{no\ of\ true\ values}{total\ no\ of\ examples}$$
$$p_- = \frac{no\ of\ flase\ values}{total\ no\ of\ examples}$$

Now mathematically solving entropy. In fig-3 every node has two values in box format. The left side value is number of false values and right side is number of true values. Let's take second layer for calculating entropy.

1st node in 2nd layer (x_2) → 28 False and 42 True values. Let's take this subset as x_2

$$H(x_2) = -\left(\frac{42}{70}\right)\log_2\left(\frac{42}{70}\right) - \left(\frac{28}{70}\right)\log_2\left(\frac{28}{70}\right)$$

$$H(x_2) = 0.2928$$

2st node in 2nd layer (x_3) → 28 False and 42 True values. Let's take this subset as f_{22}

$$H(x_3) = -\left(\frac{12}{50}\right)\log_2\left(\frac{12}{50}\right) - \left(\frac{38}{50}\right)\log_2\left(\frac{38}{50}\right)$$

$$H(x_3) = 0.2892$$

Now $H(f_{21}) > H(f_{22})$ so second way is correct root ($H(f_{22})$) for getting output. This is the internal calculation of decision tree for knowing correct root.

Information Gain

The main problem in the decision tree is finding the best pattern. Many patterns can exist for one type of problem prediction. But we don't know which is the best pattern. Now the task is to find the best pattern, from all possible patterns by calculating Information gain.

Information gain is the value that tells the best decision tree pattern. which pattern's information gain is very high, that pattern is the best out of all. So, always chose high gain valued pattern. See the mathematical formula for information gain in below.

$$Gain\ (s) = H(s) - \sum_{v} \frac{|s_v|}{|s|} H(s_v)$$

H(s) = Entropy of root node.

V = Subset (subset number)

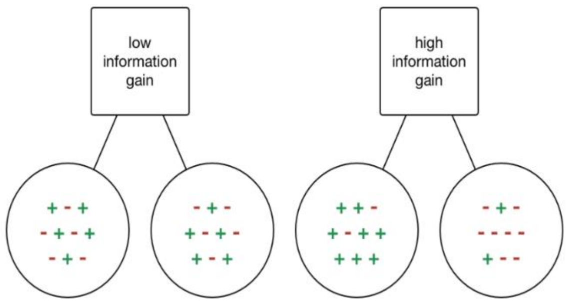


Fig 10: Information gain visualization.

s_v = Subset observations

S = Total observations

$H(s_v)$ = Entropy of subset.

Note: Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

Let's calculate the information gain for fig-3 example. It has total 120 observations.

$$H(s) = 0.2764$$

$$H(s_2) = 0.2928 \qquad s_2 = 70$$

$$H(s_3) = 0.2892 \qquad s_3 = 50$$

$$gain = 0.2764 - \left(\left(\left(\frac{70}{120}\right) \times 0.2928\right) + \left(\left(\frac{50}{120}\right) \times 0.2892\right)\right)$$

$$gain = 0.2764 - (0.1601 + 0.1205)$$

$$gain = -0.0042$$

This process repeats for all possible patterns and at the end will select high gain pattern.

Gini impurity

Gini impurity is an advanced topic for finding impurity level in splitting. Already we learned Entropy topic for finding impurity level but Gini impurity is more accurate method. The max value of Gini impurity is 0.5 and it lies between 0 to 0.5.

The value of Gini impurity is increasing from 0 to 0.5 of p value. And it will be decreased from 0.5 to 1 of p. The formula for GI is

$$GI = 1 - \sum(p_+^2 + p_-^2)$$

Code:

```python
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn import tree
import matplotlib.pyplot as plt

# Initialize the DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)

# Train the model
clf.fit(X, y)

# Display the tree structure
tree_rules = export_text(clf, feature_names=iris.feature_names)
print(tree_rules)

# Visualize the decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.show()
```

Naive Bayes Classifier

Naive Bayes Classifier:

- Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naive Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naive Bayes?

The Naive Bayes algorithm is comprised of two words Naive and Bayes, which can be described as:

- **Naive**: It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$\boldsymbol{P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}}$$

Where:

P(A|B) is Posterior probability*:* Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability*:* Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability*:* Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability*:* Probability of Evidence.

Working:

Working of Naive Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So, using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency Table for Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood Table Weather Condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14 =0.29	10/14 =0.71	

Applying Bayes' Theorem:

$$\boldsymbol{P(Yes|Sunny) = \frac{P(Sunny|Yes) \times P(Yes)}{P(Sunny)}}$$

$P(Sunny|Yes) = 3/10 = 0.3$

$P(Sunny) = 0.35$

$P(Yes) = 0.71$

So $P(Yes|Sunny) = 0.3 * 0.71/0.35 = \mathbf{0.60}$

$$\boldsymbol{P(No|Sunny) = \frac{P(Sunny|No) \times P(No)}{P(Sunny)}}$$

$P(Sunny|NO) = 2/4 = 0.5$

$P(Sunny) = 0.35$

$P(NO) = 0.29$

So $P(Yes|Sunny) = 0.5 * 0.29/0.35 = \mathbf{0.41}$

So, as we can see from the above calculation that $P(Yes|Sunny) > P(No|Sunny)$

Hence on a Sunny day, Player can play the game.

Advantages:

- Naive Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications:

- It is used for **Credit Scoring**.
- It is used in **medical data classification.**
- It can be used in **real-time predictions** because Naive Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naive Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naive Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education,etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Code:

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn import metrics

# Creating a pipeline that combines a CountVectorizer with a MultinomialNB
classifier
model = make_pipeline(CountVectorizer(), MultinomialNB())

# Training the model
model.fit(X_train, y_train)

# Predicting the labels for the test set
predicted_labels = model.predict(X_test)

# Evaluating the model
accuracy = metrics.accuracy_score(y_test, predicted_labels)
print(f"Accuracy: {accuracy:.2f}")
```

Note: **`CountVectorizer()`** is not mandatory, but it is a common method for converting text data into numerical format, which is necessary for machine learning algorithms. It transforms the text into a matrix of token counts, which the classifier can then use.

Ensemble Learning Techniques

Bagging and boosting are ensemble learning techniques used in machine learning to improve the accuracy and robustness of predictive models.

Both methods combine multiple weak learners (weak models) and make it a single model.

Bagging (Bootstrap Aggregating)

- **Parallel Learning:** Bagging trains several models in parallel on subsets of the original training dataset.
- **Bootstrap Sampling:** Each model is trained on a random subset of the training data.
- **Aggregation:** The final prediction will be obtained by combining results from all the models, usually by averaging for regression and by majority voting for classification.

Advantages:

- **Reduction in Variance:** Bagging reduces the variance of the model by averaging the final predictions to get better generalization.
- **Robustness:** It can handle overfitting well because each model is trained on a different subset of data.

Ex Algorithms: Random Forest

Boosting

- **Sequential Learning:** Boosting trains models sequentially, with each new model attempting to correct the errors of the previous ones.
- **Weighted Data:** In boosting, instances that were misclassified by previous models are given higher weights, making subsequent models focus more on these difficult cases.
- **Model Combination:** All models are combined to make the final prediction. However, not all models are considered equally; models that are more accurate (make better predictions) are given more weight in the final decision.

Advantages:

- **Reduction in Bias and Variance:** Boosting can reduce both bias and variance, leading to strong predictive performance.
- **Handling Hard-to-Learn Instances:** By focusing on difficult cases, boosting can improve the model's ability to learn complex patterns in the data.

Ex Algorithms: AdaBoost, Gradient Boosting, XGBoost

Note: **Bagging** focuses on reducing variance by averaging multiple models trained on different subsets of the data.

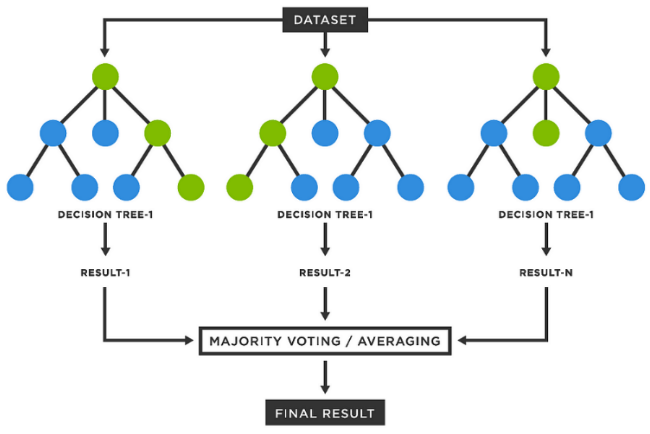
Boosting focuses on reducing both bias and variance by sequentially building models that correct the errors of previous ones.

Random Forest

A Random Forest is an ensemble learning method used for classification, regression.

It constructed by multiple decision trees during training and merges them to get a more accurate and stable prediction.

For classification tasks, the output is determined by majority voting among the trees. For regression tasks, the output is the average of the predictions made by the trees.



Random Forest uses a method called **bootstrap aggregating**, or **bagging**, to select data for training. Here's a step-by-step breakdown of the process:

1. **Random Sampling with Replacement**: The algorithm randomly selects samples from the original dataset with replacement. This means some samples may be chosen multiple times, while others might not be chosen at all. Each of these samples forms a **bootstrap dataset**.
2. **Training Decision Trees**: For each bootstrap dataset, a decision tree is trained. However, instead of considering all features for splitting nodes, only a random subset of features is considered at each split. This introduces additional randomness and helps in creating diverse trees.
3. **Aggregation (Output)**: Once all the trees are trained, the final prediction is made by aggregating the predictions of all individual trees. For classification tasks, this is usually done by majority voting, and for regression tasks, by averaging the predictions

Hyperparameters to Tune:

- **`n_estimators`:** Number of trees in the forest.
- **`max_depth`:** Maximum depth of each tree.
- **`min_samples_split`:** Minimum number of samples required to split an internal node.
- **`min_samples_leaf`:** Minimum number of samples required to be at a leaf node.
- **`max_features`:** Number of features to consider when looking for the best split.

Code:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)
```

```
# Make predictions
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

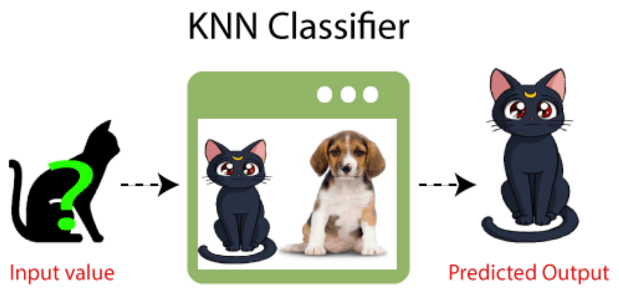
K-Nearest Neighbor (KNN)

- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

Working:

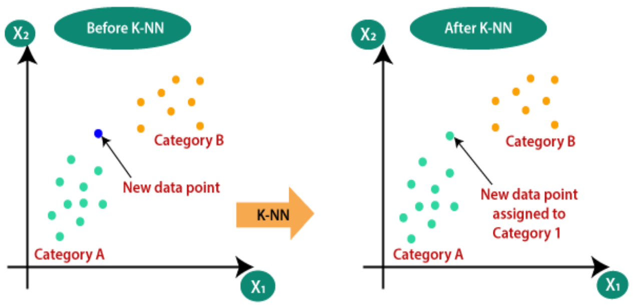
- K-NN algorithm assumes the similarity between the *new data point* and *available data points* and put the *new data point* into most similar class.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

Ex: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs' images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

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



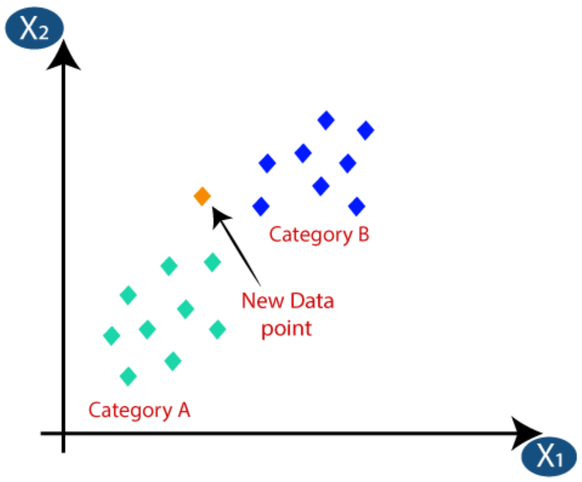
How does K-NN work?

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

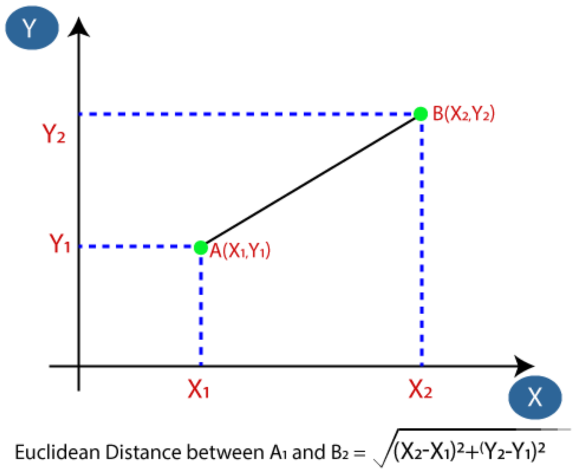
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.

- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

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



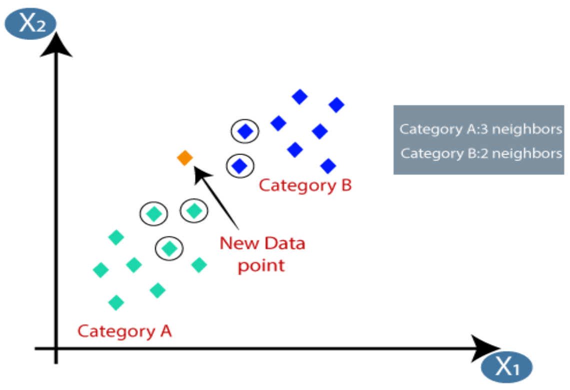
- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



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

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

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



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

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

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

Advantages of KNN Algorithm:

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

Disadvantages of KNN Algorithm:

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

Code:

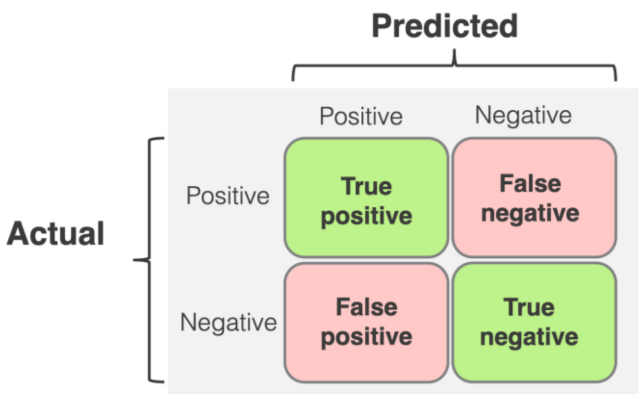
```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Initialize the KNN classifier with k=3 (you can change k as needed)
knn = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the training data
knn.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Metrices in Classification

All Types of Accuracy Measurements:

1. Confusion matrix
2. False positives rate (FPR) - Type I error
3. False negative rate (FNR) - Type 2 error
4. Accuracy
5. Precision
6. Recall (Sensitivity)
7. F beta
8. Cohen Kappa
9. ROC Curve, AVC
10. PR Curve

Confusion Matrix:



A confusion matrix is a table used to evaluate the performance of a classification model by comparing the actual labels with the predicted labels.

Each Term in the Confusion Matrix:

1. **True Positive (TP):** The model correctly predicts the positive class.
2. **False Positive (FP):** The model incorrectly predicts the positive class when the actual class is negative.
3. **True Negative (TN):** The model correctly predicts the negative class.
4. **False Negative (FN):** The model incorrectly predicts the negative class when the actual class is positive.

Example Use Case:

In a binary classification problem, such as detecting spam emails (spam/not spam):

- **True Positives (TP):** Emails correctly identified as spam.
- **False Positives (FP):** Emails incorrectly identified as spam (actually not spam).
- **True Negatives (TN):** Emails correctly identified as not spam.
- **False Negatives (FN):** Emails incorrectly identified as not spam (actually spam).

False Positives Rate (FPR):

The False Positive Rate is the proportion of negative instances that are incorrectly classified as positive.

$$FPR = \frac{FP}{FP + TN}$$

False Negative Rate (FNR):

The False Negative Rate is the proportion of positive instances that are incorrectly classified as negative.

$$FPR = \frac{FN}{FN + TP}$$

Accuracy:

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

Precision:

Correctly predicted 1's values out of all predicted 1's

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

$$Precision = \frac{correctly\ pred\ 1's}{all\ predicted\ 1's}$$

Recall:

Correctly predicted 1's values out of all actual 1's

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

$$Recall = \frac{correctly\ predicted\ 1's}{all\ \ actual\ 1's}$$

F Beta Score:

$$F - beta = (1 + \beta^2)\ \frac{Precision\ \times\ Recall}{\beta^2\ \times Precision +\ Recall}$$

- If $\beta\ = 1$ the it is called as F1 score
- If $\beta\ = 2$ the it is called as F2 score
- If $\beta\ = 0.5$ the it is called as F0.5 score

Important Points About β:

- If both precision and recall are important then set $\beta\ = 1$.
- If precision (FP) is more important than decrees β value to 0.5.
- If precision (FN) is more important than set $\beta = 2$.
- Mostly the β value selecting between 0 to 10 $0\ \le\ \beta\ \le 10$

Cohen Kappa (K):

Def: Cohen's kappa is a statistical measure used to evaluate the ***reliability of agreement*** between *two observations* (actual and predicted sets).

Reliability Of Agreement Meaning: ***Reliability of agreement*** means how two observations same each other's. which means How much predicted values are similar to actual values.

Working:

Create a Confusion Matrix: Create a matrix where rows represent actual values and columns represent predicted values.

Suppose you have a binary classification problem, and the confusion matrix for the predictions is:

	Predicted Yes	*Predicted No*
Actual Yes	50	10
Actual No	5	35

Calculate Observed Agreement (P_0): This is the proportion of instances where the prediction matches the actual value, summed across all categories.

$$P_0\ = \frac{50 +\ 35}{100} = 0.85$$

Calculate Expected Agreement (P_e): This involves calculating the expected frequency of each category being predicted, based on the marginal frequencies of the actual and predicted values.

- Actual Yes: 0.60 (60/100)
- Actual No: 0.40 (40/100)
- Predicted Yes: 0.55 (55/100)
- Predicted No: 0.45 (45/100)

$$P_e\ = (0.60\ \ \times 0.55) + (0.40 + 0.45)$$

$$= 0.33 + 0.18 = 0.51$$

Compute Cohen's Kappa:

$$K\ = \frac{P_0\ -\ P_e}{1 -\ P_e}$$

$$K\ = \frac{0.85 -\ \ 0.51}{1 -\ \ 0.51}\ = 0.69$$

Value Rage of Cohen's Kappa:

The value of kappa can range from -1 to 1:

- 1: Perfect agreement
- 0: Agreement equivalent to chance
- -1: Perfect disagreement (inverse agreement)

Common Interpretation Guide:

- < 0.0: Poor agreement
- 0.01–0.20: Slight agreement
- 0.21–0.40: Fair agreement
- 0.41–0.60: Moderate agreement
- 0.61–0.80: Substantial agreement

- 0.81–1.00: Almost perfect agreement

Code:

```python
from sklearn.metrics import
cohen_kappa_score

# Sample data
actual = [0, 1, 0, 1, 0, 1, 0, 1, 1]
predicted = [0, 1, 0, 0, 0, 1, 1, 1, 1]

# Calculate Cohen's kappa
kappa = cohen_kappa_score(actual,
predicted)

print(f"Cohen's Kappa: {kappa}")
```

ROC Curves:

ROC - Receiver Operating Characteristic

A ROC curve is a graphical representation used to identify the performance of a binary classification model. It plots the ***True Positive Rate (TPR)*** against the ***False Positive Rate (FPR)*** at ***various threshold*** settings.

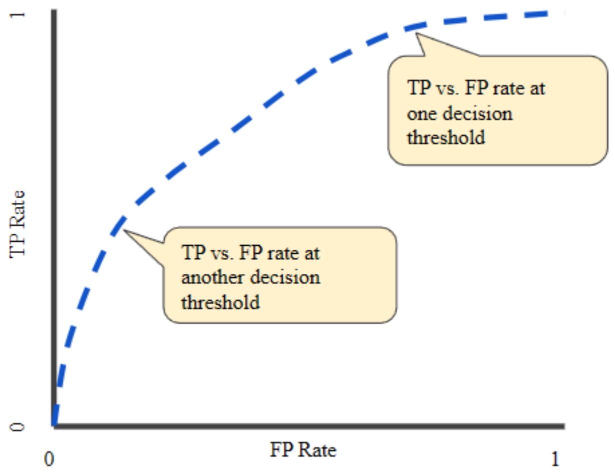
Key Metrics:

1. **True Positive Rate (TPR) or Sensitivity**: This measures the proportion of actual positives that are correctly identified by the model.
2. **False Positive Rate (FPR)**: This measures the proportion of actual negatives that are incorrectly identified as positives.
3. **Threshold**: threshold is a cutoff value between two classes. If the model's prediction is above this threshold, it predicts as positive class; if it's below, it predicts as negative class.

   By varying this threshold at different TPR and FPR values are obtained, leading to the ROC curve.

Plotting the ROC Curve:

- The **x-axis** represents the False Positive Rate (FPR).
- The **y-axis** represents the True Positive Rate (TPR).



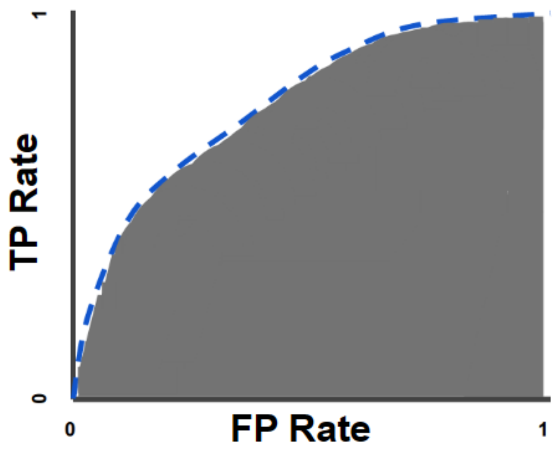
A point on the ROC curve represents a specific decision threshold, and the curve is obtained by varying this threshold from 0 to 1.

Diagonal Line: A model with no discrimination ability (like random guessing) will have a ROC curve that lies on the diagonal line (FPR = TPR).

Above the Diagonal: The better the model, the more the curve bows towards the top-left corner (higher TPR for a given FPR).

Area Under the Curve (AUC): The area under the ROC curve (AUC) is a single scalar value that summarizes the overall performance of the model. An AUC of:

- 0.5 indicates no discrimination (random guessing).
- indicates perfect classification.
- Between 0.5 and 1.0 indicates the degree of usefulness, with higher values being better.



Use of ROC:

- Compare the performance of different classifiers.
- Choose a suitable threshold based on the trade-offs between TPR and FPR.
- Understand the model's ability to discriminate between classes, especially in cases with imbalanced datasets.

Code:

```python
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Sample data
# true_labels: 1 for positive class, 0 for negative class
actual_values = np.array([0, 0, 1, 1])
# predicted_probabilities: probability scores predicted by the model
predicted_values = np.array([0.1, 0.4, 0.35, 0.8])

# Compute ROC curve and ROC area
fpr, tpr, thresholds = roc_curve(actual_values, predicted_values)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = 
{roc_auc:0.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Handling Imbalanced Data

Synthetic sampling in data science refers to techniques used to generate new, artificial data points based on the characteristics of an existing dataset. This approach is commonly employed to address issues of class imbalance in classification problems, where certain classes have significantly fewer instances than others. Synthetic sampling aims to create a more balanced dataset, which can lead to improved performance of machine learning models.

Synthetic Sampling Techniques:

1. SMOTE (Synthetic Minority Over-sampling Technique)
2. ADASYN (Adaptive Synthetic Sampling)
3. Borderline-SMOTE
4. Random Over-Sampling

SMOTE (Synthetic Minority Over-sampling Technique):

- SMOTE generates synthetic examples in the feature space by interpolating between existing minority class examples. It creates new instances that lie between two minority class instances.
- **How it works**: For each minority class sample, SMOTE selects one or more of its nearest neighbors and creates new synthetic samples along the line segments connecting the sample and its neighbors.

ADASYN (Adaptive Synthetic Sampling):

- ADASYN is an extension of SMOTE that focuses on generating more synthetic data for minority class samples that are harder to learn (i.e., those that are misclassified by a learning algorithm).
- **How it works**: ADASYN adaptively generates more synthetic samples in the regions where the density of minority class examples is low, aiming to reduce the bias towards the majority class.

Borderline-SMOTE:

- This technique focuses on generating synthetic samples near the decision boundary where the instances are more likely to be misclassified.
- **How it works**: It identifies minority class samples near the borderline (close to the majority class samples) and generates synthetic samples only in these regions.

Random Over-Sampling:

- This is a simpler approach where existing minority class samples are randomly duplicated until the class distribution is balanced.
- **How it works**: It involves randomly replicating minority class samples without introducing any new variability or synthetic data.

Advantages of Synthetic Sampling

- **Improved Model Performance**: Balancing the class distribution can help machine learning models perform better, especially in terms of metrics like precision, recall, and F1-score.
- **Data Augmentation**: It effectively augments the existing dataset, providing more training examples for the minority class.
- **Versatility**: Techniques like SMOTE and ADASYN can be applied to various types of data, including text, images, and structured data.

Disadvantages and Considerations

- **Risk of Overfitting**: Synthetic samples can sometimes lead to overfitting, especially if they do not adequately capture the underlying data distribution.

- **Computational Cost**: Some synthetic sampling methods can be computationally intensive, especially for large datasets.
- **Quality of Synthetic Samples**: The quality of the synthetic samples depends on the method used and the nature of the data. Poor-quality synthetic samples can negatively impact model performance.

SMOTE Code:

```python
from imblearn.over_sampling import SMOTE
from collections import Counter
import pandas as pd

# Sample dataset
data = {'Feature1': [1, 2, 3, 4, 5, 6],
        'Feature2': [10, 20, 30, 40, 50, 60],
        'Class': [0, 0, 0, 1, 1, 1]}
df = pd.DataFrame(data)

# Features and labels
X = df[['Feature1', 'Feature2']]
y = df['Class']

# Apply SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)

print(Counter(y_resampled))
```

ADASYN Code:

```python
from imblearn.over_sampling import ADASYN

# Apply ADASYN
adasyn = ADASYN()
X_resampled, y_resampled = adasyn.fit_resample(X, y)

print(Counter(y_resampled))
```

Random Over Sampling Code:

```python
from imblearn.over_sampling import RandomOverSampler

# Apply Random Over-Sampling
ros = RandomOverSampler()
X_resampled, y_resampled = ros.fit_resample(X, y)

print(Counter(y_resampled))
```