# ML Modelling

Bhadri Vaidhyanathan – Personal Notes

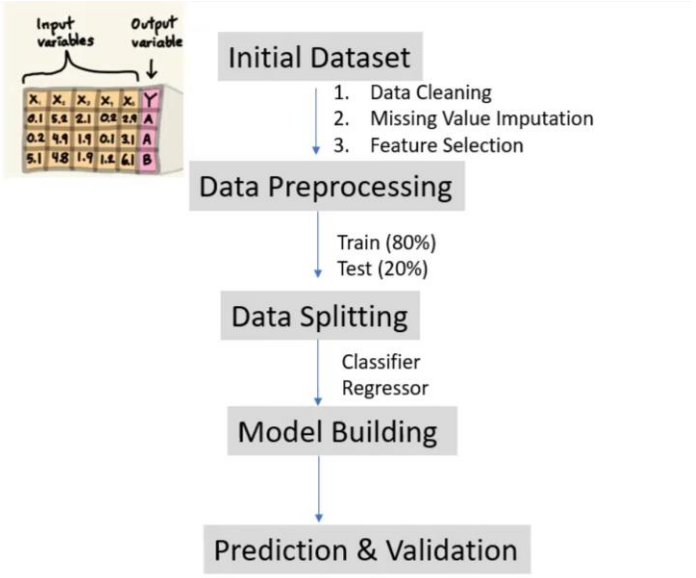
## Contents

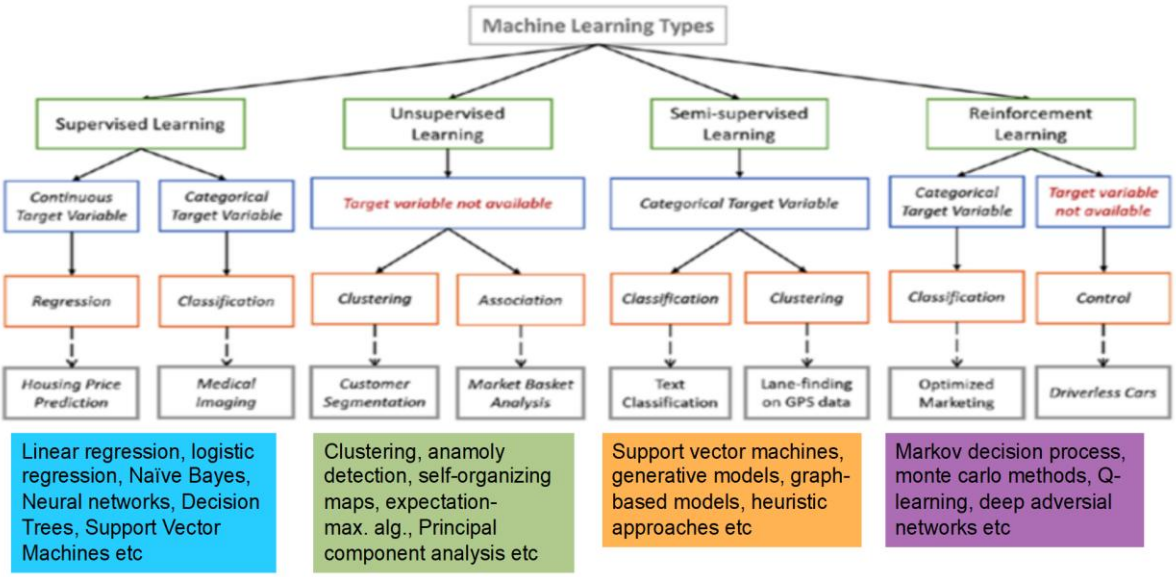
# Information Theory

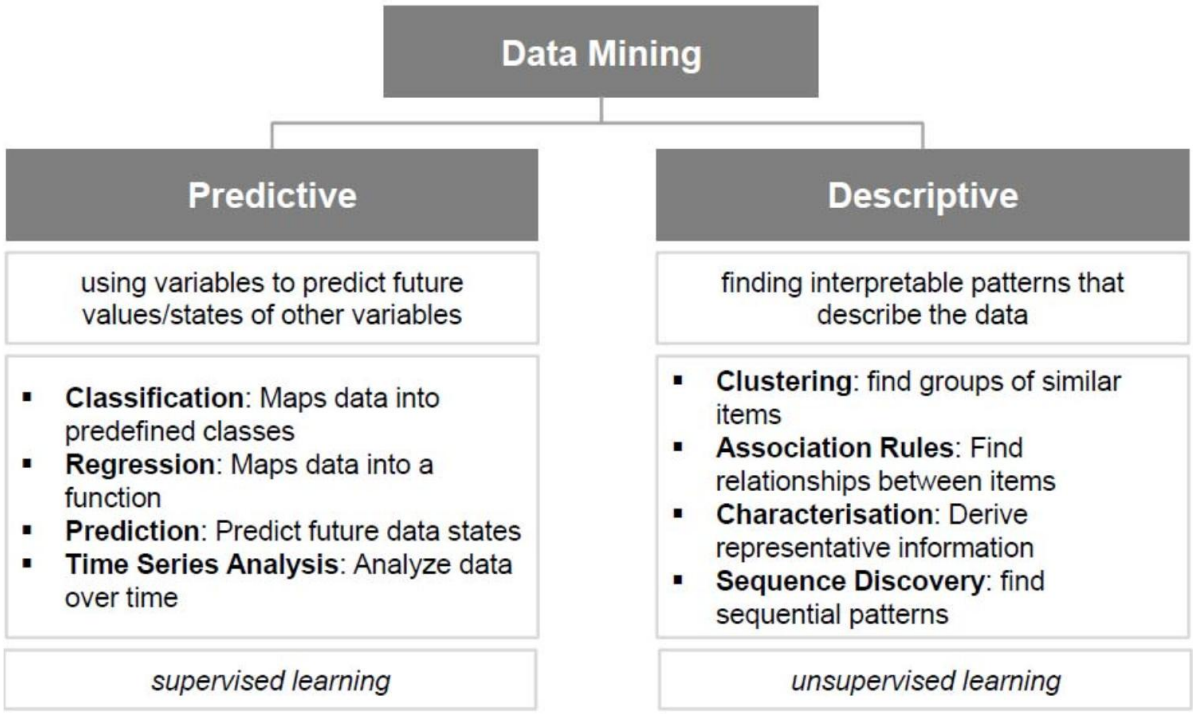
- Models estimate probability distributions of the data
- Parameters, variables, weights are all information in and about the data
- Models capture the information we need from the data in our parametersS

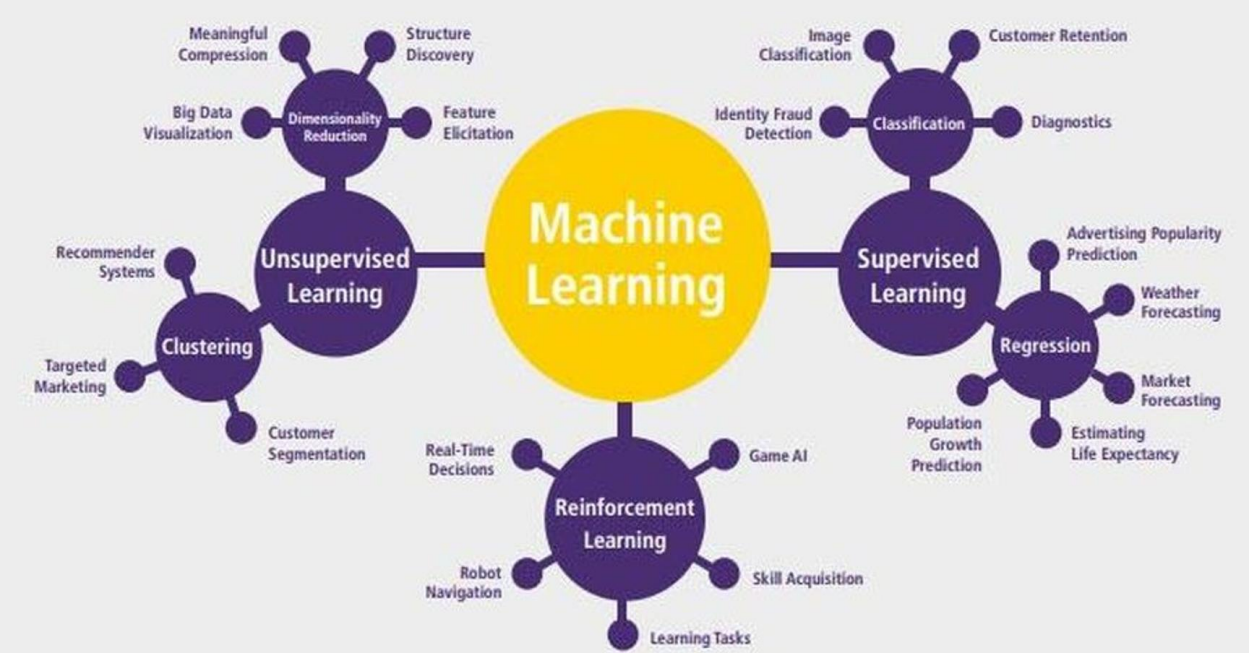
## Intro

Basic process:






**ML Applications:**

- **Regression** (or prediction) — a task of predicting the next value based on the previous values.

- **Classification (supervised)** — a task of separating things into different categories where we have few samples/test cases for each category.
- **Clustering** (unsupervised) — similar to classification but the classes are unknown, grouping things by their similarity.
- Association rule learning (or **recommendation**) — a task of recommending something based on the previous experience.
- Dimensionality reduction — or generalization, a task of searching common and most important features in multiple examples.
- Generative models — a task of creating something based on the previous knowledge of the distribution.

## Definitions/Terminology:

**Artificial intelligence (AI):** A broad discipline with the goal of creating intelligent machines, as opposed to the natural intelligence that is demonstrated by humans and animals. It has become a somewhat catch all term that nonetheless captures the long term ambition of the field to build machines that emulate and then exceed the full range of human cognition.

**Machine learning (ML):** A subset of AI that often uses statistical techniques to give machines the ability to "learn" from data without being explicitly given the instructions for how to do so. This process is known as “training” a “model” using a learning “algorithm” that progressively improves model performance on a specific task.

**Reinforcement learning (RL):** An area of ML concerned with developing software agents that learn goal-oriented behavior by trial and error in an environment that provides rewards or penalties in response to the agent’s ctions (called a “policy”) towards achieving that goal.

**Deep learning (DL):** An area of ML that attempts to mimic the activity in layers of neurons in the brain to learn how to recognise complex patterns in data. The “deep” in deep learning refers to the large number of layers of neurons in contemporary ML models that help to learn rich representations of data to achieve better performance gains.

**Algorithm:** An unambiguous specification of how to solve a particular problem.

**Model:** Once a ML algorithm has been trained on data, the output of the process is known as the model. This can then be used to make predictions.

**Supervised learning:** A model attempts to learn to transform one kind of data into another kind of data using labelled examples. This is the most common kind of ML algorithm today.

Supervised Learning: When you have input and output(target/response) variable data.

**Unsupervised learning:** A model attempts to learn a dataset's structure, often seeking to identify latent groupings in the data without any explicit labels. The output of unsupervised learning often makes for inputs to a supervised learning algorithm at a later point.

**Transfer learning:** An approach to modelling that uses knowledge gained in one problem to bootstrap a different or related problem, thereby reducing the need for significant additional training data and/or boosting performance.

**Natural language processing (NLP):** Enabling machines to analyse, understand and manipulate human language.

**Computer vision:** Enabling machines to analyse, understand and manipulate images and video.

**Classification Algorithm**: When response variable is categorical

Notes:

**Dummy Coding**

Traditional ML like regression, random forest cannot work on categorical data directly and must be represented with “dummy coding” (add a column for each category and fill with 0/1).

- Dummy coding should be done for all categorical data for Traditional ML methods.
- Deep Learning / Neural Network can handle categorical data directly.
- It is always recommended to have only N-1 columns of data where N is the number of categories. This is because, it is not necessary to have two columns to represent 2 categories like gender. Gender can be represented in just one column where 1 is female and 0 is male.

**Imputation:**
Replacing wrong values with mean of the rest of the data. Should mean be calculated with the outliers/(assumed) wrong values or without? Both can be done on the other hand instead of mean, median can be used since median is not affected by the outliers.

**Feature Engineering / Development:**
Process of creating a new predictor/column by combining two or three columns by a mathematical operations. Eg: New column D is created from (A+B)/C

**Conditioning**
Conditioning is deleting anomaly data by deleting the entire row

**Fitting** – Fitting a model is taking data and creating a fitted/model as per the algorithm which is later used to predict. Eg: Fitting a model in Linear regression is abt getting that linear regression line which can be removed from data and used later to predict Y. Fitting a model is in decision tree is creating that flowchart of decisions which makes the model and later using it to predict values.

**Centering** – subtracting mean from each value so in the end we have values that are around a 0 scale. A step done in PCA.

# Important Sub-Topics in ML Modelling:

## Parametric and Nonparametric Algorithms:

**Parametric**: Methods that make assumptions about the underlying data distribution (e.g., normal distribution) or the POPULATION and **Training involves estimating parameters of the POPULATION** (e.g., mean, variance).
Eg: Linear Regression, Logistic Regression, Naive Bayes, ARIMA (for time series), Generalized Linear Models (GLMs)

**Characteristics**:

- Assumes a specific functional form for the relationship between input and output.
- Training involves estimating a finite set of parameters.
- Generally faster and requires less data.
- May perform poorly if assumptions are violated.

**Nonparametric:** Methods that do not assume a fixed form for the data distribution and can model complex relationships without a predefined equation.
**Examples**: Decision Trees, Random Forests, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Neural Networks

**Characteristics**:

- Flexible and can capture complex patterns.
- Require more data to generalize well.
- Can be computationally intensive.
- Less interpretable (Black Box feeling) than parametric methods.

Above Topic has nothing to do with HYPERPARAMETER TUNING Topic. The "Hyper" add-on is expected to separate them. HyperParameters or experiment setup variables like learning rate in NN or depth of trees in Decision Trees.

## Model Assumptions of the Data:

**Linear Regression**

1. **Linearity**: The relationship between independent and dependent variables is linear.
2. **Homoscedasticity**: Errors have constant variance.
    - Homo – same; scedacity – Variance → "same variance"
    - Variation in the data is consistent throughout; essentially, the spread of the data around the regression line should be roughly the same at all points

3. **Correlations:**
    - **No Multicollinearity**: Independent variables are not highly correlated.
    - **No Endogeneity**: Independent variables are uncorrelated with the error term.
4. **Normality of Errors**: Errors follow a normal distribution.
5. **Independence of Errors**: Residuals are independent of each other.
6. **Independence of Observations**: Observations are independent.

**Logistic Regression**

1. **Linearity (of log-odds)**: The log-odds of the dependent variable have a linear relationship with the independent variables.
2. **Correlations**
    - **No Multicollinearity**: Independent variables are not highly correlated.
    - **No Endogeneity**: Independent variables are uncorrelated with the error term.
3. **Independence of Observations**: Observations are independent.
4. **Independence of Errors**: Residuals are independent of each other.

**Naive Bayes**

1. **Feature Independence**: Assumes all features are conditionally independent given the target variable.

**Decision Trees / Random Forests / Gradient Boosting Models (e.g., XGBoost, LightGBM)**

1. **Sufficient Data**: Enough data to avoid overfitting and ensure accurate splits.
2. **Feature Independence (Optional)**: Correlated features may reduce interpretability but don't affect performance.

**Neural Networks**

1. **Sufficient Data**: Requires large datasets, especially for deep learning.
2. **Independence of Observations**: Observations should not be dependent (e.g., i.i.d.).
3. **No Multicollinearity (Optional)**: Helps with faster convergence during training but not mandatory.

## Model Assumptions of the Data: IID

**IID stands for Independent and Identically Distributed:**

1. **Independent**: Each data point is independent of the others. The occurrence of one does not influence the others. **Independence** ensures that the relationship between observations is not confounded by any dependent relations
2. **Identically Distributed**: All data points come from the same probability distribution. **Identical distribution** ensures consistency, meaning the model learns patterns from a homogenous data-generating process.

**IID is a fundamental assumption in many statistical and machine learning models, as it simplifies analysis and ensures that mathematical properties (like convergence) hold.** If the data points are not IID, the theoretical guarantees of many models (e.g., consistency of estimators, normality of errors) may not hold.

- **Applies strongly to parametric models** because these models rely on fixed assumptions about the data distribution (e.g., normality, linearity). For these models, IID is often required to make inferences about parameters (e.g., coefficients) or to calculate p-values, confidence intervals, and predictions.

- **Applies, but less strictly to non-parametric models**: Non-parametric models make fewer assumptions about the underlying data distribution, so they are more robust to violations of IID. These models can handle certain forms of dependency or heterogeneity better (e.g., clustering, time series) but may still degrade in performance if IID is **grossly violated.**

1. **When IID Assumptions are usually good**:
    - Data collected via random sampling from a population (e.g., a survey of voters' opinions).
    - Controlled experiments where conditions are uniform.
2. **When IID Assumptions Break**:
    - **Time Series Data**: Observations are often dependent (e.g., stock prices today depend on yesterday's prices).
    - **Spatial Data**: Nearby locations often have correlated data (e.g., temperature measurements).
    - **Clustered Data**: Observations within clusters (e.g., students in the same classroom) are often correlated.

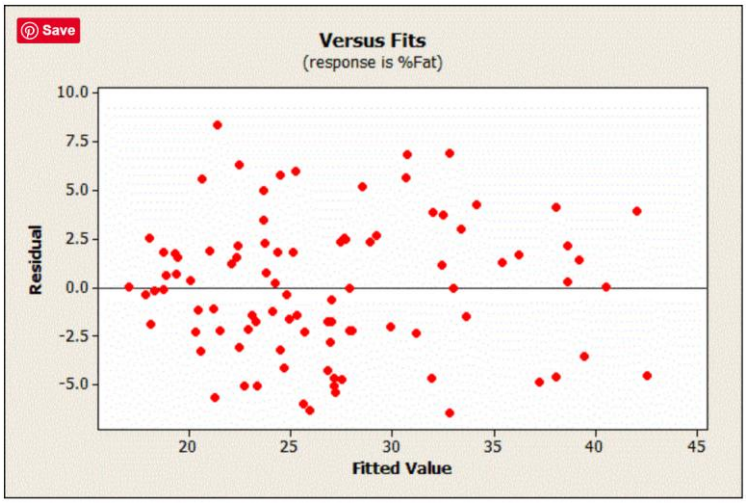
In such cases, specialized methods (e.g., time series models, mixed-effects models) or adjustments (e.g., resampling techniques) are required.

## Model Selection:

- If you have 2000 columns and rows, which so much dimensions, linear model should be good.
  Lasso Linear Reg is best method to shrink the beta of many unuseful predictors to zero
    - feature_importance_ in random forest are also another way to eliminate predictors

## Residual Analysis

Residuals must be normally distributed and so must be clustered around zero but below chart shows that they are not hence not normally distributed
Residual plot is the chart between residuals values vs fitted values



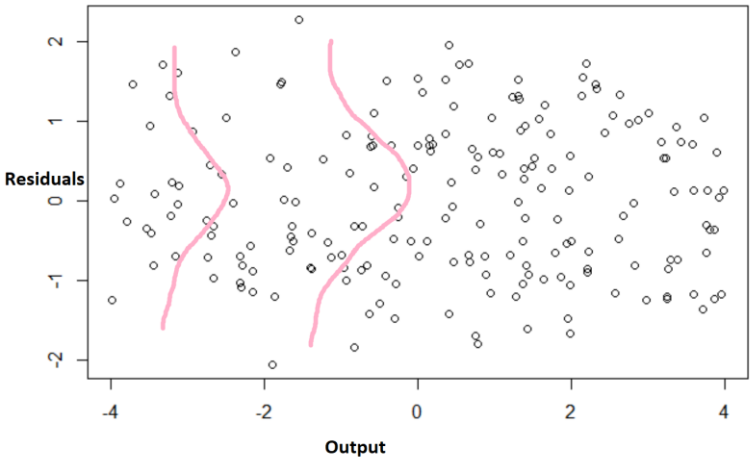

After fitting a regression model, check the residual plots first to be sure that you have unbiased estimates.

Why?

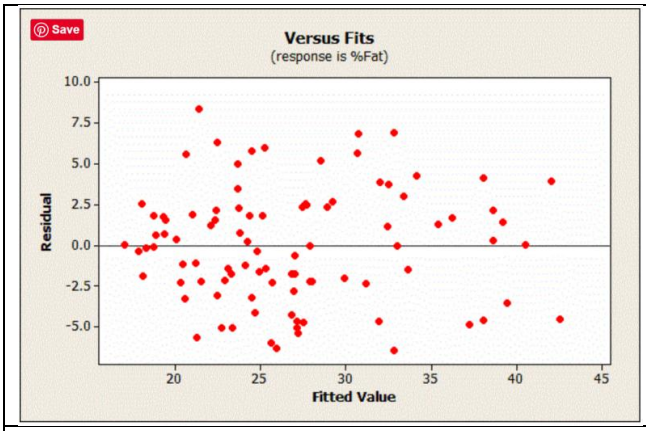
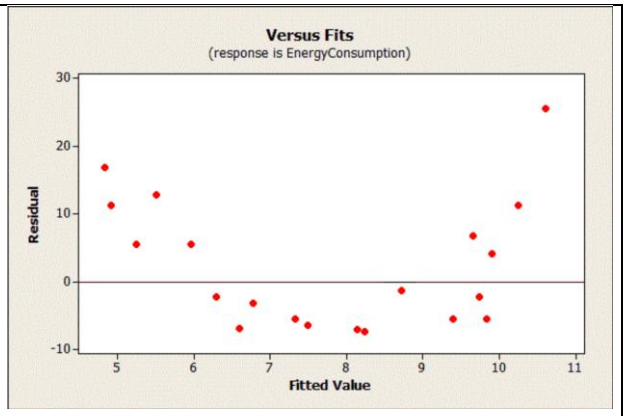
Like when you throw a die and if the die is always landing on 4, you know there is a pattern and that die might be biased. In same fashion, when you look at the errors/residuals, it should be random but if a pattern is visible you say residuals are biased or estimates are biased just like you wld say the rolling die is biased.

- just check that they are randomly scattered around zero for the entire range of fitted values.
- the residuals follow a normal distribution and that the degree of scattering is the same for all fitted values.
  The normal distribution should be observed side ways like in the chart below.

  In below chart the residuals are **NOT** normally distributed as in nor clustered around Residual = 0 Line.




| Good one: randomly distributed residuals | Bad one: pattern observed |
|---|---|

| | |
|---|---|
|  |  |
| | Consequence: If you know the fitted value, you can use it to predict the residual.<br>For instance, fitted values near 5 and 10 tend to have positive residuals while fitted values near 7 tend to have negative values. If they were truly random, you wouldn't be able to make these predictions.<br>Like in time series, if the detrended series is showing a pattern, here also, it means that some pattern is still left unrepresented in the errors.<br>The regression line dint do a good job of capturing all capturable pattern |

**If the residuals (errors) are not normally distributed, then what is implied and what are the consequences?**
It can indicate problems with your model or data:

1. **Violation of Model Assumptions**: normally distributed residuals are required for valid hypothesis testing (e.g., t-tests and F-tests) and accurate confidence intervals.
2. **Something missing**: Non-normal residuals suggest that the model has not generalized well and seems to be inclining towards few end solutions which are called as "biased predictions". Basically some thing is missing is the conclusion.
3. **Misspecification of Model**: Non-normal residuals can imply:
    - The functional form of the model is incorrect (e.g., a linear model used for non-linear data).
    - Important variables are missing.
    - Outliers or influential points distort the model.

In practice, if residuals are not normal, consider transformations, adding interaction terms, or switching to a more robust modeling technique (e.g., generalized linear models).

## What to do if residual plot looks bad?

https://statisticsbyjim.com/regression/check-residual-plots-regression-analysis/

There are a variety of reasons why a model can have this problem. The possibilities include a missing:

- Independent variable.
- Polynomial term to model a curve.
- Interaction term.

To fix the problem, you need to identify the missing information, variable, or higher-order term and include it in the model. It might require subject-area knowledge and research to do this. The solution is very particular to your research.

### Bad Residual plot: Other Potential Problems

There are several other ways that explanatory information might make its way into your residuals:

- **Another variable must not be correlated with the residuals.** If a variable is related to the residuals, that variable can predict the residuals, which is a no-no. Try including this variable in the model. To identify this correlation, graph the residuals by other variables. This problem relates to confounding variables and causes omitted variable bias.
- **Neighboring residuals must not be correlated.** If adjacent residuals are correlated, one residual can predict the next residual. In statistics, this is known as autocorrelation. This correlation represents explanatory information that the independent variables do not describe. Models that use time-series data are susceptible to this problem. To resolve this issue, try adding an independent variable that contains the pertinent time information. Use the Durbin-Watson test to assess autocorrelation.
- **Residuals must have a constant variance.** Heteroscedasticity refers to cases where the residuals have a non-constant variance. Read my post about how to identify and correct heteroscedasticity.

## Data Leakage:

Data leakage happens when merging data from different sources. Some of this data could possibly not be obtained and thus we are adding a predictor that helps in creating the model but be absent in real world data.

Data Leakage is very important to “predictive” models.

Ways to mitigate:

- Perform data preparation within your cross validation folds.
- Hold back a validation dataset for final sanity check of your developed models.
- https://machinelearningmastery.com/data-leakage-machine-learning/

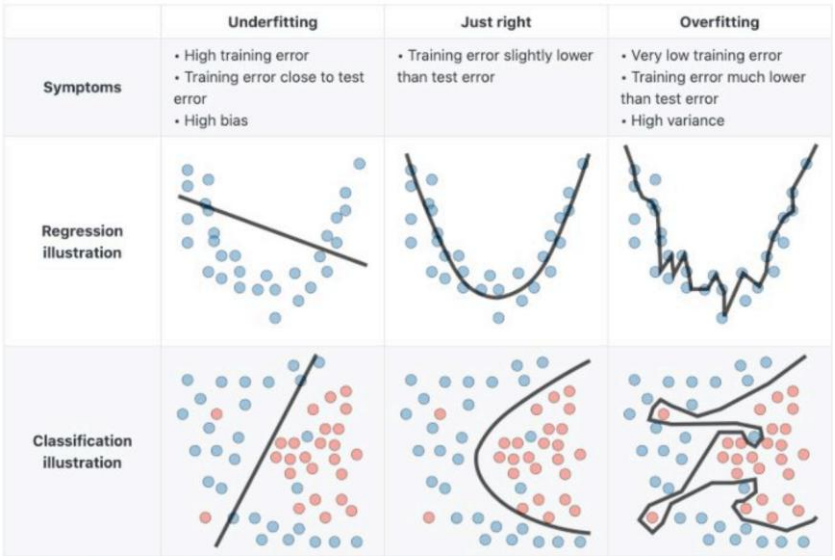
## Bias – Variance Trade Off ; UnderFitting & OverFitting:

**Bias-Variance terms are a property of the solution** (the line that is estimated and the solution)

- **Bias ~ Underfitting** = High Bias as in my way or highway(too straight)
- **Variance ~ Overfitting** = High Variance is too bendy and pleasing every point. Too much variation

Bias is when the model thinks a relationship between variables is too simple and is unwilling to bend to incorporate more or be more inclusive.

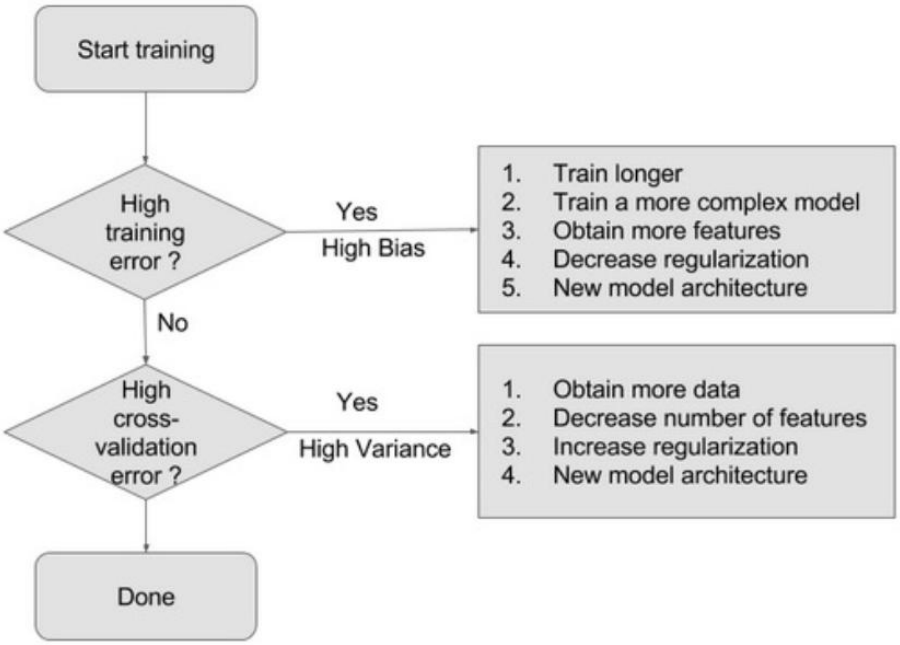
Variance – Is the opp of bias and being too malleable and trying to fit all points into the model. By being too bendy, when this model is applied to the real world data – huge errors are seen thus a high variance.




Expectation is a model with low bias and low variability but however, trying to improve the bias might also lead to increasing the variability hence the **"tradeoff"**.

Regularization is a method to improve both overfitting and indirectly underfitting.

Troubleshooting High Bias or High Variance :




# Outlier Treatment

Outliers are values 3SDs from the mean (use Median for skewed distributions).

Outliers can break the model or skew it and so if they can be removed or treated, they should. Caveat: Do not remove if they are very relevant to the business case.

**Why treat?**

- To stabilize variance and reduce skewness.

Also see the table below on sensitivity of different Algos to outliers

**Dont treat them and let them be** if you are using models that are not susceptible to Outliers as mentioned below.

## Treatment strategies:

### 1) Remove them

Remove them only if they are irrelevant or misleading your ML task.

### 2) Transform them But

By Transform we mean,

## 1. Log

- Right Skewed distributions are quite common and because they are not normal, Log transformations make them normal.

**Example:**
If a column represents "income," and you take the log, you're **no longer working with raw income values but their relative magnitudes.**

| Original Income | Log Transformation (log(x)) |
|---|---|
| 10,000 | 4 |
| 50,000 | 4.7 |
| 1,000,000 | 6 |

In the transformed data, the differences between large values are compressed, making extreme outliers less influential.

## 2. Sq Rt

## 3. Box-Cox

**Box-Cox searches for the best exponent λ thro Max. Likelihood method** that best gets the data to a normal distribution. Unlike log or square root transformations (which use fixed exponents) and so rigid which makes them sometimes work and sometimes don't.

4. Winsorization : replace outlier with the highest non-outlier value

**Disadvantages of Transformations:**

1. Loses interpretability particulary the non-linear transforms like log, sqrt,box-cox
    - A log-transformed "income" is harder to explain (log(5000) ≠ real-world money).
- Log, SqRt and Box-Cox don't work on zero/negative values
- Non-Linear transforms (log, sqrt,box-cox) **can't** be used for Parametric models which assume linear relationships

### 3) Test them

Validate that Outliers are the model performance bad by re-running the training with and without them and checking the test performance.

### 4) Make ML more robust

Choose models that are not sensitive to Outliers as below.

### 5) Dont treat if relevant/valid

If they are rare but valid events then let them be and not treat them.

**ML Algorithms Sensitive to Outliers**

Outliers **significantly impact** models that rely on **distance, mean, or variance calculations**. Below are the algorithms most affected by outliers:

| Algorithm Type | Examples | Why Affected? | Level of Impact |
|---|---|---|---|
| **Linear Models** | Linear Regression, Logistic Regression, Lasso, Ridge | Outliers shift the regression line, leading to poor predictions. | High |
| **Distance-Based Models** | KNN, K-Means Clustering | Distance calculations are distorted by extreme values. | High |
| **Variance-Based Models** | PCA (Principal Component Analysis) | Outliers dominate variance, distorting principal components. | High |
| **Gradient-Based Models** | Neural Networks | Extreme values lead to unstable training and weight updates. | Medium |
| **SVM (with Linear Kernel)** | Support Vector Machines | Outliers influence the decision boundary heavily. | Medium |

---

### ◇ ML Algorithms Not Compromised by Outliers

These models are **robust** and either ignore or handle outliers well:

| Algorithm Type | Examples | Why Robust? |
|---|---|---|
| **Tree-Based Models** | Decision Trees, Random Forest, | Trees split data into bins, so extreme values don't distort |

| Algorithm Type | Examples | Why Robust? |
|---|---|---|
| | XGBoost | splits. |
| **Robust Regression** | Huber Regression, RANSAC Regression | Use robust loss functions that reduce outlier impact. |
| **SVM (with RBF Kernel)** | Support Vector Machines | The RBF kernel maps data to a higher dimension, reducing outlier influence. |

---

### When safe to remove, treat/keep or investigate?

First an outlier here is 3 SD s away.

**Final Decision Framework**

| % Outliers | Action | Justification |
|---|---|---|
| **≤ 1%** | **Delete outliers** | Safe to remove without affecting data integrity |
| **1 - 5%** | **Cap outliers** (Winsorization) | Prevents excessive data loss while reducing extreme values |
| **> 5%** | **Investigate further** | Outliers may be meaningful or indicate issues with the data |

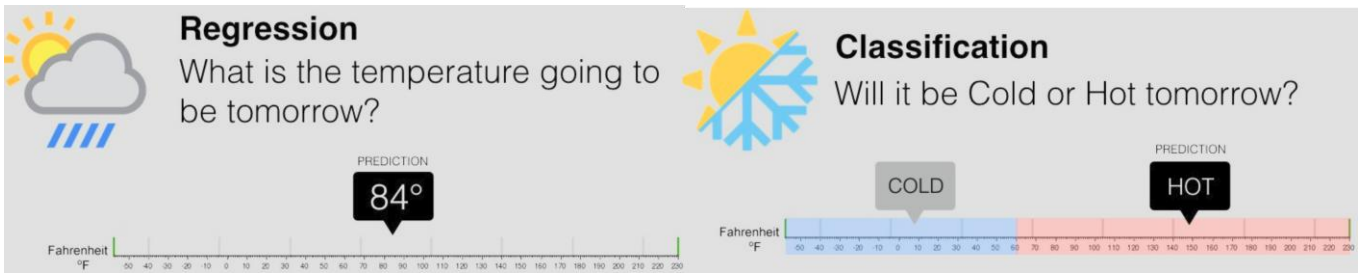
---

---

# Regression

In ML, “Regression” refers to guessing a number in a continuous scale (often not discrete scale) while “Classification” refers to guessing a categorical type.

**If you hear Regression, Think its about guessing a continuous number.**



Regression can be done with many diff. algos as below but this gets side-lined given the amount of info each new topic like Decision Tree or Gradient Boosting involves so putting an advance topic image below just so its not disregarded. **Just Observe and move along!!!**


Regression - regression toward the mean (aka reversion to the mediocrity) is simplifying a possibly complex relation into a simple line or closer to the mean of the set hence the term regression.

Regression applies to any method that is trying to find this cookie cutter or template including Neural Networks or Decision trees. These advanced methods can also be manipulated to be a simple "Regression". But of course Linear Regression is the most popular one among them.

# Model: Linear Regression

- Linear Regression: The process of finding a straight line (as by least squares) that best approximates a set of points on a graph.
- LR is all about finding a line and its formula that will help define relationship between two sets of variables – Dependent and Independents.
- Eqn of line is $y = mx + intercept$
  Hence its about finding m (slope of line) and intercept (higher or lower line on graph)

- SSR or sum of sq. error is metric **(Used by Algo internally)** that tells how badly a line fits the data so its abt finding that line that has lowest SSR and/or similar error.
- Linear – refers to relationship between predictors and target variable is linear and not nonlinear like sq. or sqrt or exponentials. if say relationship between x and y is $x^2$ then this a parabolic curve, non-linear
- Regression - regression refers toward the mean (aka reversion to the mediocrity) is simplifying a possibly complex relation into a simple line or closer to the mean of the set hence the term regression.

Linear Regression can be done in many ways, Least Squares and Gradient Descent.

- Least Squares is a math formula just applied so the coefficients are discovered instantly. This is an analytical approach.
- Gradient Descent is an iterative approach where values are changed a little, solution is calculated, values again changed a little, solution found. The solutions of cos are compared to make sure it is improving or the error is actually reducing

## Least Squares Regression Method

aka OLS – Ordinary Least square, Normal Equation method

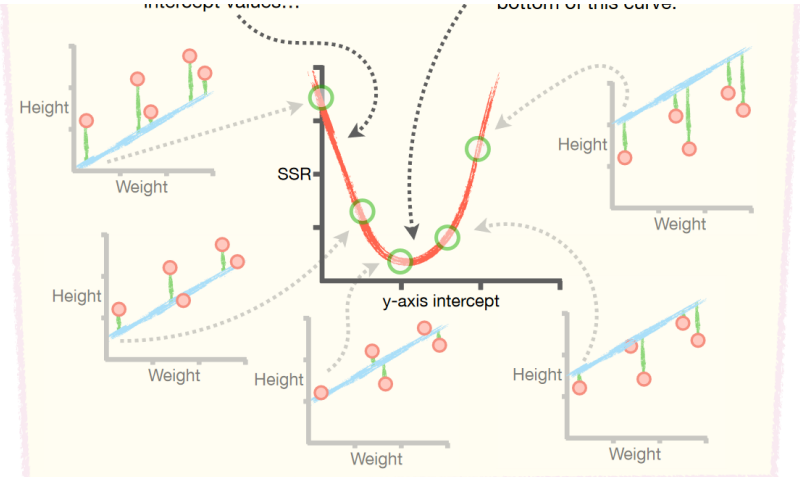
No analytical solution for **Log.** Regression so this method has limited use.

Least squares is a method and the equation used to find the values is the 'Normal Equation' so this technique is referred to by 'Least squares or Normal Equation. Both refers to this same technique against Gradient Descent which is different.

"Least Squares Regression" is the most widely used method to draw best fit line. Least Squares Regression basically minimizes the Sum of Squared Residual (SSR) thus also the name least "square".

Least Squared method is the Analytical approach where we have a mathematical formula that directly gives us the slope of least squares line.

How is the optimal value of m (slope of regression line) obtained?




Using Calculus, the partial derivative for SSR eqn in y= mx + b form is solved and bam you get the m and intercept of the best regression line.

## Deriving Slope and intercept of the Best fit line:

Differentiation is applied to the SSR equation to find the normal equations, which provide the analytical solution for the coefficients β0 and β1. This ensures that the regression line minimizes the total squared differences between the observed values and the predicted values.

In **linear regression**, when using the **least squares method** to derive the best-fit line, the goal is to minimize the **sum of squared residuals (SSR)**, which quantifies the difference between the observed values and the values predicted by the regression model.

The Equation for Sum of Squared Residuals (SSR):

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

Where:

- y : Actual observed value.
- y^: To be Predicted value best fit line coordinates.
- n: Number of data points.

For a simple linear regression, $\hat{y}_i = \beta_0 + \beta_1 x_i$ which put in above formula:

$$SSR = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

## Differentiation to Minimize SSR:

To find the coefficients β0(intercept) and β1(slope) that minimize the SSR, we take partial derivatives of SSR with respect to β0 and β1 and set them to zero.

1. **Partial derivative with respect to β0:**

$$\frac{\partial SSR}{\partial \beta_0} = \sum_{i=1}^{n} -2\left(y_i - (\beta_0 + \beta_1 x_i)\right)$$

2. **Partial derivative with respect to β1:**

$$\frac{\partial SSR}{\partial \beta_1} = \sum_{i=1}^{n} -2x_i\left(y_i - (\beta_0 + \beta_1 x_i)\right)$$

   Setting these derivatives equal to zero gives the normal equations, which can be solved simultaneously to find β0 and β1.

3. **Normal Equations:** After solving the derivatives, the normal equations for a simple linear regression are:

1. For β0(intercept):

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

2. For β1(slope):

$$\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

Differentiation is applied to the SSR equation to find the normal equations, which provide the analytical solution for the coefficients β0 and β1. This ensures that the regression line minimizes the total squared differences between the observed values and the predicted values.

## P value of a regression:

Regression runs in R will print results which will include a p value.

What is this pvalue and what the related null hypothesis?

p value is abt the predictor/independent variable and whether it is correlated with target variable.

Null hypothesis: No correlation between predictor and target (which is bad in sense, we sld remove independent var.)

Low pvalue> reject null hypothesis. -> there is some correlation -> keep the independent variable in the equation since changes to it leads to change in target variable

Coefficients

| Term | Coef | SE Coef | T | P |
|---|---|---|---|---|
| Constant | 389.166 | 66.0937 | 5.8881 | 0.000 |
| East | 2.125 | 1.2145 | 1.7495 | 0.092 |
| South | 5.318 | 0.9629 | 5.5232 | 0.000 |
| North | -24.132 | 1.8685 | -12.9153 | 0.000 |

## Gradient Descent Linear Regression:

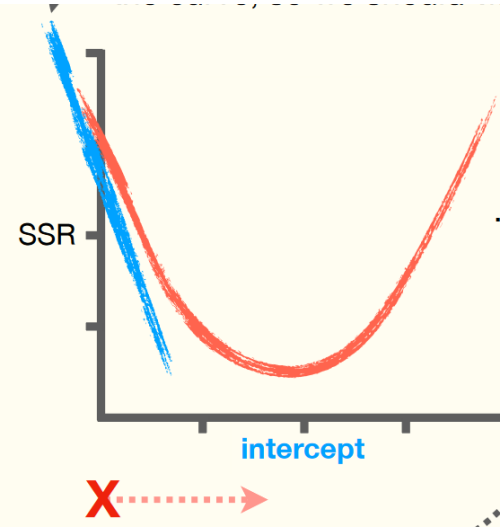
**PreReq - Loss Function and Derivatives:**

Logistic regression's optimal value or coeff cannot be obtained with just a mathematical formula as done in OLS above. That's is where Gradient Descent comes in. GD is also key for Neural networks.

With regard to Linear regression, we evaluate a line based on its SSR. Lower SSR means good line.

So its all about turning the dial on variables,…etc etc until SSR is low. MSE and other types of loss variable can also be used and thus this loss variable that we are trying to get control of is the Loss Function or Cost function.

GD is all about getting to the lowest point of this line or curve or whatever where the SSR or MSE or error is lowest. Think Elbow plot in Kmeans Classification.




- red curve is the loss function which is unknown in the beginning
- So one picks few random input variable

Derivatives is the slope, in a way, to a nonlinear equation like $x^2$. Partial Derivatives is the slope for multiple variables / in a 2D environment where there is a x and y.

Partial derivatives (are estimated similar to derivatives) here the other variables say y are 0 or kept constant when partial derivative of x is being found )in math lang.

The gradient/partial derivatives found gives the slope value which gives the most increase in loss for a function at that specific point/equation. We want to go the exact other way to reduce the loss and this gradient is reduced from coefficients/parameters of the equation in question. Changed values are applied to the equation and new solution is found which represents a new point in the 2Dplane of the equation.

# GD Types: Batch and Stochastic

## Batch GD

- In standard (batch) gradient descent, the gradient is calculated using **all the training data** in one go. This is in cases when all data is actually small but enough to train the model. Some models will need a lot of data to train and processing all data points will be too much for the architecture so smaller bites maybe work
- The batch size is a hyperparameter that controls the number of training samples to work through before the model's internal parameters are updated. Popular are 32, 64 & 128
- One Epoch is one complete passes with all the data in the training set. Multiple epochs are done to ensure the initialization related errors are averaged out.

**What Is a Batch?**

Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

A training dataset can be divided into one or more batches.

- **Batch Gradient Descent**. Batch Size = Size of Training Set
    - **Mini-Batch Gradient Descent**. 1 < Batch Size < Size of Training Set

## Stochastic GD (SGD) – Randomness involved

Stochastic came about when data sets started to be unmanageably big and requiring too much compute to train on the entire dataset. It comes with its own pros and cons but preferred in large-scale projects.

When the batch size is not the whole training set but a "random" (Stochastic means random) subset either just one **random** sample or a **random** mini-batch then it is called it is called stochastic gradient descent.

- **Stochastic Gradient Descent**. Batch Size = 1, random
    - **Mini-Batch Stochastic Gradient Descent.** Random mini batch
- **Stochastic** means randomness is introduced by computing the gradient using **only a single data point (or a small random subset of data called a "mini-batch")** at each step instead of the entire dataset.
- This randomness makes SGD faster and more scalable for large datasets.
- The random approach introduces noise in the gradient calculation, which at times is preferred like in the case of non-convex optimization problems
- **Advantages of SGD:**
    1. **Large Scale Datasets/projects: Works well with big data** (1 million data points etc)**.**
    2. **Efficiency:** Faster updates compared to full-batch gradient descent for large datasets.
    3. **Exploration:** Noise in updates helps escape local minima or saddle points.
- **Challenges of SGD:**
    1. **Convergence:** May oscillate around the minimum due to noisy updates.
    2. **Instability/Tuning:** Can diverge if the learning rate is too large. Requires careful tuning.
- **Key Variants of SGD:**
    1. **Mini-Batch SGD:** Uses a small, random subset of data points (mini-batch)
    2. **Momentum:** Adds a momentum term to stabilize updates and accelerate convergence.
    3. **Adam:** Combines ideas from Momentum and RMSprop for adaptive learning rates.

# Linear Reg. Model Evaluation Tools:

**Popular metrics for regression:** "$r^2$" (default), "mae", "rmse", "mse"

## Metrics for Regression:

There are 3 **main** metrics for model evaluation in regression:

*1. R Square/Adjusted R Square*

*2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)*

*3. Mean Absolute Error(MAE)*

### R Squared $R^2$: (best only for Linear Reg. alone)

$R^2$ by formula is the % of variance of the variance from the mean line's fit and thus how much better the Linear Reg. Line versus the mean line.
$R^2$ mathematically, the way it is calculated, tells how much better new line fits data versus a simple mean of

values line. Fitted line is usually better fit and thus variance of mean line higher than variance of Fiiter Linear Reg. Line so $R^2$ ,tells the % of variance from the Mean line fit that is explained by the Linear Reg line. Thus showing how $R^2$ is a measure of how good line fits data than the mean line. It is a value between 0 to 1 and also described as a %.

**Variance** is the mean squared distance between each data point and the mean point. It is measure of spread of the data. Squared to prevent negative coordinates cancelling out. SD is also a measure of variance and it sq.rt of variance and the same unit as data points.

$$\textbf{R2 Squared} = 1 - \frac{\textbf{SSr}}{\textbf{SSm}}$$

**SSr = Squared sum error of regression line**

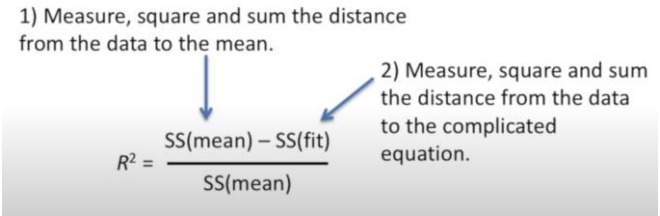
**SSm = Squared sum error of mean line**

$$R^2 = \frac{\text{Var(mean)} - \text{Var(fit)}}{\text{Var(mean)}}$$

Var(mean) is the variance(residuals) of data points around the mean line

Var(fit) is the variance (residuals) of the data points with fitted/reg. line

$R^2$ can be applied for any model.




Values of $R^2$ **outside** the range 0 to 1 occur when the model fits the data worse than the worst possible least-squares predictor (equivalent to a horizontal hyperplane at a height equal to the mean of the observed data). This occurs when a wrong model was chosen, or nonsensical constraints were applied by mistake.

| Calculated measures | What it is? | what else does it indicate or used for? |
|---|---|---|
| Sum of Sq. Residuals | the sq. distance between fitted line and each point | measure of how well the line fits the data. compared with other lines's SSR |

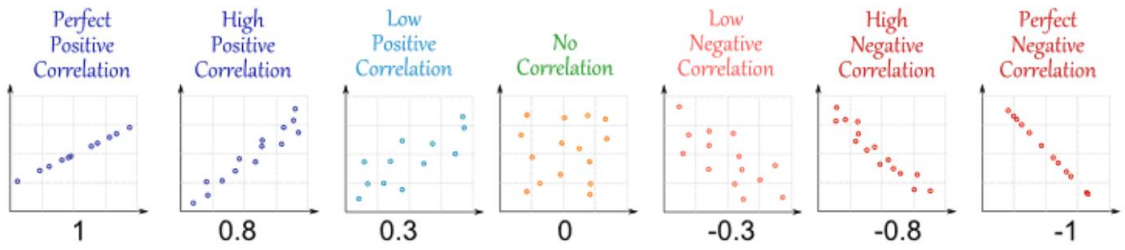
- **$R^2$ is scale Invariant while other measures below varies with scale**

## *Problems with R-squared statistic*

1) The R-squared metric in its original intended form is only valid for ordinary least squares (OLS) regression methods, with linear regression being the most common/popular of these. For logistic/maximum likelihood regression models, different goodness-of-fit metrics exist that sometimes use R-squared in their name, but are mathematically quite different and often very unreliable as measures of fit.

2) The R2 suffers a major flaw. Its value never decreases no matter the number of variables we add to our regression model. That is, even if we are adding redundant variables to the data, the value of R-squared does not decrease.

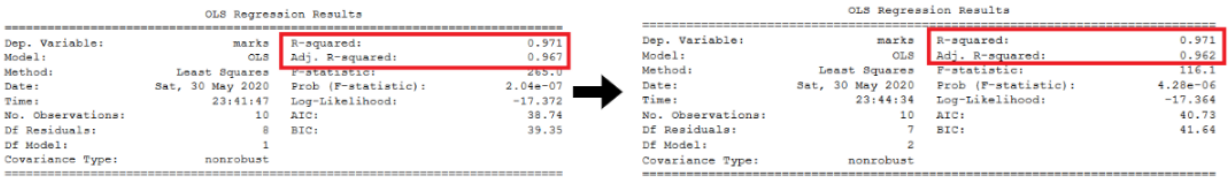
On the other hand it does increases with the addition of certain new independent variables (in a multivariate problem) which add value. This clearly does not make sense because some of the independent variables might not be useful in determining the target variable. Adjusted R-squared deals with this issue.




# Adjusted R square

is the only metric here that considers the overfitting problem.

R2 remains same when a variable (in a multivariate problem) with no value is added to the equation giving false idea that it is helpful. Adjusted R2 formula is such that it decreases when a use**less** variable is added and increases when a useful variable is added.



OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | marks | R-squared: | 0.971 |
| Model: | OLS | Adj. R-squared: | 0.967 |
| Method: | Least Squares | F-statistic: | 265.0 |
| Date: | Sat, 30 May 2020 | Prob (F-statistic): | 2.04e-07 |
| Time: | 23:41:47 | Log-Likelihood: | -17.372 |
| No. Observations: | 10 | AIC: | 38.74 |
| Df Residuals: | 8 | BIC: | 39.35 |
| Df Model: | 1 | | |
| Covariance Type: | nonrobust | | |

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | marks | R-squared: | 0.971 |
| Model: | OLS | Adj. R-squared: | 0.962 |
| Method: | Least Squares | F-statistic: | 116.1 |
| Date: | Sat, 30 May 2020 | Prob (F-statistic): | 4.28e-06 |
| Time: | 23:44:34 | Log-Likelihood: | -17.364 |
| No. Observations: | 10 | AIC: | 40.73 |
| Df Residuals: | 7 | BIC: | 41.64 |
| Df Model: | 2 | | |
| Covariance Type: | nonrobust | | |

As you can see, adding a random independent variable did not help in explaining the variation in the target variable. Our R-squared value remains the same. Thus, giving us a false indication that this variable might be helpful in predicting the output. However, the Adjusted R-squared value decreased which indicated that this new variable is actually not capturing the trend in the target variable.

Hence,

To control this situation Adjusted R Squared came into existence.

$$R_a^2 = 1 - \left[\left(\frac{n-1}{n-k-1}\right) \times (1 - R^2)\right]$$

where:

$n$ = number of observations

$k$ = number of independent variables

$R_a^2$ = adjusted $R^2$

R Square has a direct library in Python to calculate but I did not find a direct library to calculate Adjusted R square except using the statsmodel results. If you really want to calculate Adjusted R Square, you can use statsmodel or use its mathematic formula directly.

## Mean Square Error(MSE)/Root Mean Square Error(RMSE)

While R Square is a relative measure of how well the model fits, Mean Square Error is an absolute measure of the goodness for the fit. It gives you an absolute number on how much your predicted results deviate from the actual number. You cannot interpret many insights from one single result but it gives you a real number to compare against other model results and help you select the best regression model.

**Scale Variant**:

RMSE is scale variant in the sense if we change the scale, the RMSE changes.

Ex.: If we used dollar values as is (5000) versus the same values in 1000s (5) and so be aware when comparing the values that the scales of two models being compared have same scale.

Root Mean Square Error(**RMSE**) is the square root of MSE. It is used more commonly than MSE because firstly sometimes MSE value can be too big to compare easily. Secondly, RMSE is same unit as error while MSE is squared error and thus RMSE makes it easier for interpretation.

```
1. from sklearn.metrics import mean_squared_error
2. import math
3. print(mean_squared_error(Y_test, Y_predicted))
4. print(math.sqrt(mean_squared_error(Y_test, Y_predicted)))
```

## R2 vs RMSE

- R2 compares new regression line with the line made using mean.
- RMSE is a measure of regression line and residual with actual values.

## Mean Absolute Error(MAE)

Mean Absolute Error(MAE) is similar to Mean Square Error(MSE). However, instead of the sum of square of error in MSE, MAE is taking the sum of the absolute value of error.

Compare to MSE / RMSE, MAE is a more direct representation of sum of error terms. MSE gives larger penalization to big prediction error by square it while MAE treats all errors the same.

```
1. from sklearn.metrics import mean_absolute_error
2. print(mean_absolute_error(Y_test, Y_predicted))
```

**Conclusion: RMSE vs MSE vs MAE**

- R Square/Adjusted R Square is better used to **explain the model to other people** because you can explain the number as a percentage of the output variability.
- MSE, RMSE, or MAE are better be used to **compare performance between different regression models**. Personally, I would prefer using RMSE and I think Kaggle also uses it to assess the submission.
- However, it makes total sense to use **MSE if the value is not too big and MAE if you do not want to penalize large prediction errors**.

## Generalized Linear Models:

Linear Models expect a linear relationship between predictors and target variable. It is very narrow in usage and while little more wiggle room allows for few more data models to be fit under the Regression methodology. GLM is a framework which allows few more distributions like Binomial, Logistic and Poisson to also be able to modelled.

Traditional Linear Regression had strict conditions and only appropriate when:

- The response variable is continuous.
- The residuals are normally distributed.
- The variance of residuals/errors are constant (homoscedasticity).

When these conditions are violated, GLMs step in with greater flexibility:

- **Logistic Regression**: For binary outcomes (e.g., yes/no).
- **Poisson Regression**: For count data (e.g., number of visits).
- **Gamma Regression**: For skewed continuous data (e.g., expenses).

GLM came up with a framework of $g(\mu)=\mu$ and components such that if a data model could be fit into it it is a GLM.

LinkFunc(data) ~ Target variable with known distribution like Poisson or Gamma

The predictors are connected to the target via transformation of the data to a format or distribution that is standard, defined and known enough to be included in the Regression family of ML methods/algos.

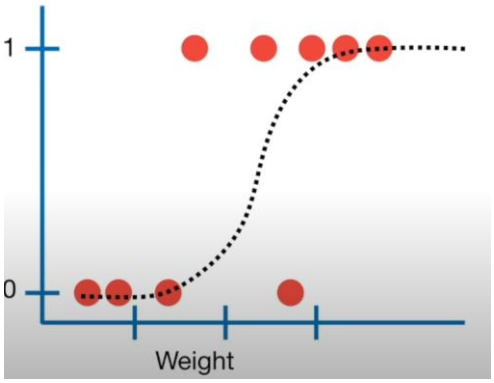
Linear Regression also fits into above framework where the Link function is unit and it is also included ad GLM as a special case.

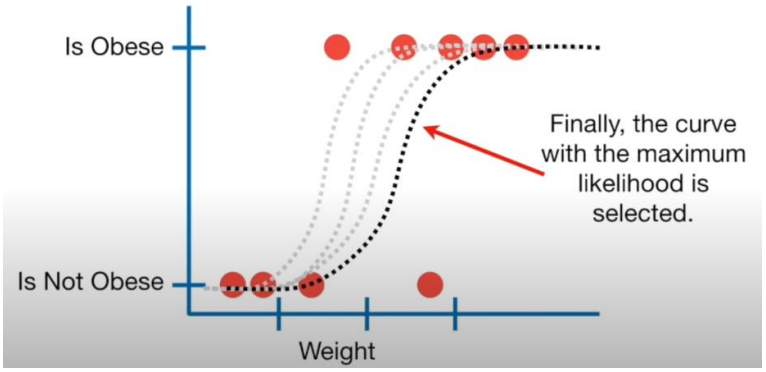
# Model: Logistic Regression

**Log Reg is a simple binary classification algorithm.**

**Premise: log-odds of the binary dependent variable have a linear relationship with the independent variables**

**Log Reg provides probabilities between 0 to 1 and classifies samples to one side (x<0.5) or the other(x>0.5)** is used for classification problems (Log Reg still falls under Regression family of methods) since it can separate and classify a record between two values be it True/False, Yes/No etc



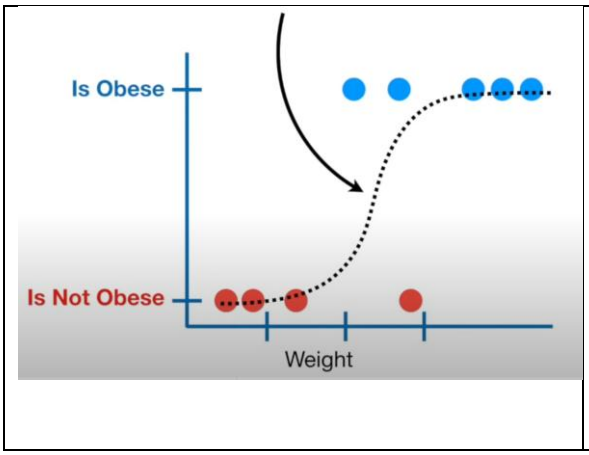

For any linear issues, monovariate (above) or multi-variate, Log Reg. Fits an S curve or sigmoid curve between 2 classes. Say Y=1 and Y=0.

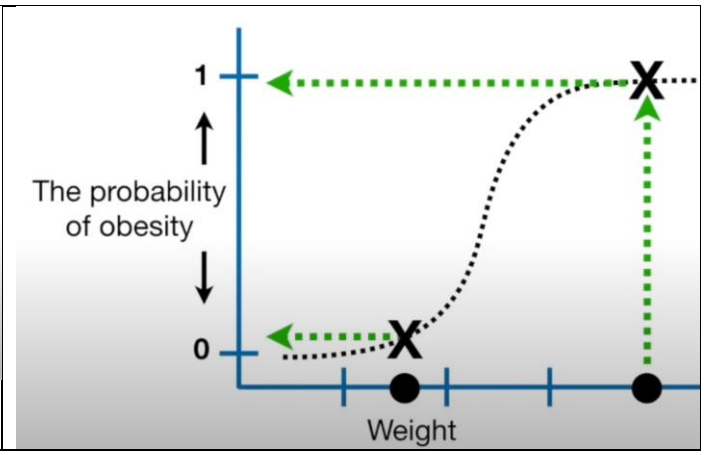
Which curve fits the best is found through iterative Gradient Descent. It is also said maximum likelihood. The GD algo tries to maximize this log odds and thus maximum likelihood.

Log Reg fits an s shaped logistic curve on the data where the curve goes from 0 to 1.

LogReg is all about deciding between two classes based on a factor or bunch of them.

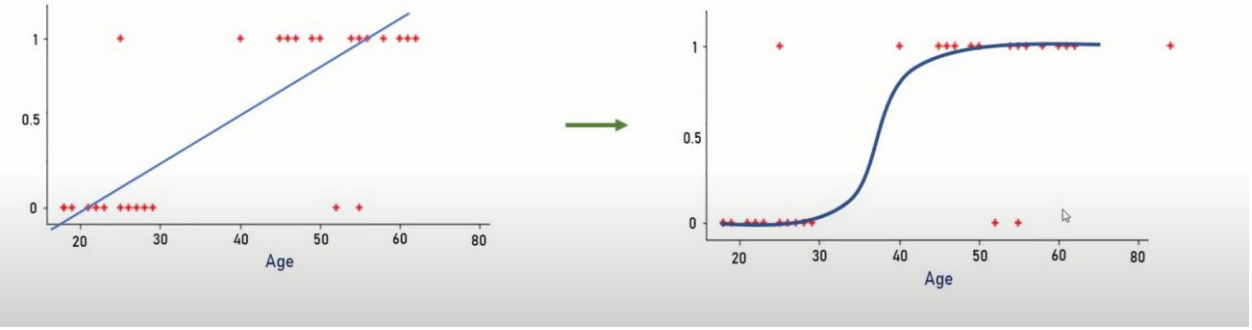
LogReg line defines this relation with the steep s curve. Where it splits is where LogReg s curve helps and besides, later, when you plug in factor, using the s cruve, you can immediately tell if it falls on top class or bottom class.



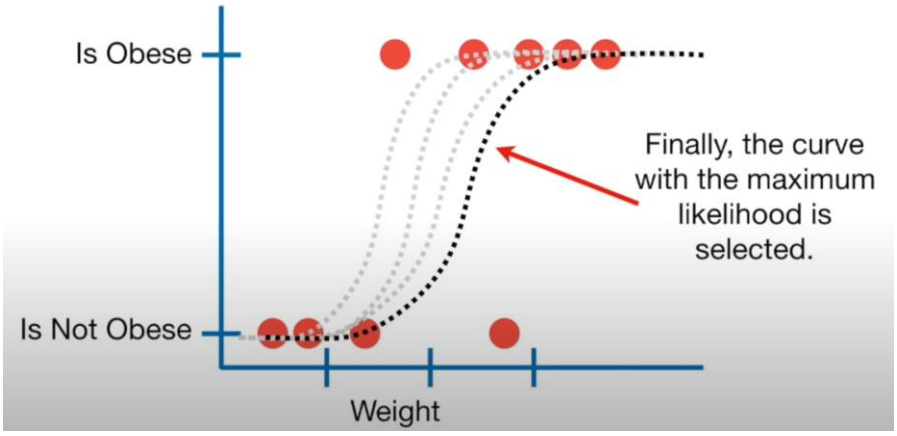
$$y = m * x + b$$

$$y = \frac{1}{1 + e^{-(m*x+b)}}$$




Log Reg can work on multiple factors/predictors. The way it is done to test if each factor's effect is stat significant from 0 effect.

In linear reg, the line is found by least squares method. In Log Reg, the curve is found with "maximum likelihood", an iterative approach.




Maximum likelihood is basically fitting multiple s curves on the data and pick the one with the max likelihood and best classification.

## How to interpret LogReg Coefficients?

https://towardsdatascience.com/a-simple-interpretation-of-logistic-regression-coefficients-e3a40a62e8cf

A simple Linear Regression's line eqn is

Y = mx +b

In LogReg it is like

log ( P{Y=1] / P{Y=0} ) = mx + b

- where Y=1 and Y=0 are the binomial target variables (could be dummied like male=0,female=1 or Heads=1 and Tails=0)
- P{Y=1] / P{Y=0} is called odds ratio =>

| | | |
|---|---|---|
| probability of (heads) / (tails) | no. of heads / no. of tails | |

**Odds-Ratio:**

*The **probability** of getting a 4 when throwing a fair 6-sided dice is 1/6 or ~16.7%. On the other hand, the **odds** of getting a 4 are 1:5, or 20%. This is equal to p/(1-p) = (1/6)/(5/6) = 20%.*

Now when value of a predictor (x) goes up by 1,

- there will be a m increase in [log(odds ratio)] => [log(Heads/Tails)]
- if m increases then the log-odds-ratio increases such that Odds of numerator increases at the cost of Denominator
- If m goes negative then numerator decreases while denominator increases which increases the chance of Tails overall

*Imagine the final LogReg eqn to be :*

*Log (heads/tails) = log(p/1-p) = 0.5 + 0.13 * factor1 + 0.97 * factor2*

In the model above, *b = 0.13, c = 0.97,* and *p = P{Y=1}* is the probability of getting heads.

Let's pick *f1* and see how it impacts the chances of heads. Increasing f1 by 1 unit will result in a 0.13 increase in *logit(p)* or *log(p/1-p)*. Now, if *log(p/1–p)* increases by 0.13, that means that *p/(1 — p)* will increase by *exp(0.13) = 1.14*. This is a 14% increase in the odds of getting Heads (*assuming that the f2 remains fixed*).

Let's also interpret the impact of f2 on getting heads. We know that *exp(0.97) = 2.64*. That being said, the odds for heads are 164% higher when f2 goes up.

*Since both f1 and f2 are addition +, any increase in them is an increase in odds of Heads. If F2 was -*0.97f2 *then increase in f2 will be increase in odds of Tails.*

## Why is Logarithm needed in first place?

logarithm is introduced is simply because the logarithmic function will yield a normal distribution with consideration to odds-ratio distributions while shrinking extremely large values

## How to Check Linearity in Logistic Regression?

Instead of checking feature vs. target, we should check feature vs. log-odds (transformed target).

1) **Box-Tidwell Test (for Continuous Variables)**

The Box-Tidwell test is used to check for linearity between the predictors and the logit. This is done by adding log-transformed interaction terms between the continuous independent variables and their corresponding natural log into the model.

**How Does the Test Work?**

1. **Create an interaction term** between the feature X and its log transformation: $X' = X.\log(X)$
2. **Fit a logistic regression model** with both X and X' as predictors: $\log(p/1-p) = \beta 0 + \beta 1X + \beta 2*(X.\log X)$
3. **Check the significance of β2 (p-value of X*logX)**:
   - **If p<0.05** → The relationship is **non-linear**, so you need to transform XXX.
   - **If p≥0.05** → The relationship is **approximately linear**, and no transformation is needed.

2) **Log-Odds vs. Feature Plot**

- Fit a **logistic regression model** and compute the **log-odds** from predicted probabilities: $\log(p1-p)=\beta 0+\beta 1X$
- Plot **log-odds vs. feature** → Should be approximately linear.
- If curved, try transformations (e.g., **log, square, interaction terms**).

3) **Generalized Additive Models (GAM)**

- Fit a **non-parametric smooth function** to see if the feature effects are linear.
- **Deviation from a straight line suggests a need for transformation.**

Given that we have better frameworks like Decision Trees and NN, when should one us LogReg:

1. The dataset is small or well-structured
2. relationship between features and Log of Odds-Ratio is roughly linear
3. Need a simple model and simpler system requirement to process

## Logistic Regression Algorithm Step-By-Step

Let us assume a dataset:

| Hours Studied (**X1**) | Practice Tests (**X2**) | Outcome / Target |
|---|---|---|

| | | (Y) |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |
| 4 | 2 | 1 |
| 5 | 2 | 1 |

**Step-by-Step Process**

1. **Initialize Coefficients:** Assume initial guesses for the coefficients: β0=0,β1=0,β2=0

This means the model starts with no effect from the predictors.

2. **Compute Predicted Probabilities:** For each row in the dataset, calculate the log-odds and probability ppp:

$$\text{log-odds} = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

$$p = \frac{1}{1 + e^{-\text{log-odds}}}$$

For the first iteration, since all coefficients are 0:

$$\text{log-odds} = 0$$

$$p = \frac{1}{1 + e^{0}} = 0.5$$

- Predicted p for all rows: p1=0.5, p2=0.5, p3=0.5, p4=0.5, p5=0.5
- **Calculate the Error for Each Row:** The error is the difference between the actual outcome Y and the predicted probability p:

$$\text{Error}_i = Y_i - p_i$$

For our dataset:

Error1=0−0.5= **−0.5**, Error2=> **−0.5**,

Error3=1−0.5=0.5, Error4=>0.5, Error5=>0.5

**4. Update Coefficients (each Predictor) Using Gradient Descent using all the values of that predictor:** For each coefficient βj compute the gradient:

$$\text{Gradient } \beta_j = \sum \text{Error}_i * \text{PredictorVariable}X_{ij}$$

where Xij is the value of predictor j for observation i.

**4a.** Example for β1(Hours Studied):
Gradient $\beta 1 = (-0.5 \cdot 1) + (-0.5 \cdot 2) + (0.5 \cdot 3) + (0.5 \cdot 4) + (0.5 \cdot 5) = -0.5 - 1 + 1.5 + 2 + 2.5 = 4.5$

Update β1 using a learning rate, α=0.1:

$$\beta 1_{new} = \beta 1_{old} + \alpha * Gradient\beta 1$$
$$\beta 1_{new} = 0 + 0.1 * 4.5 = 0.45$$

**4b.** Repeat and find β2(Practice Tests)

**4c.** Finally, find β0 (Intercept)

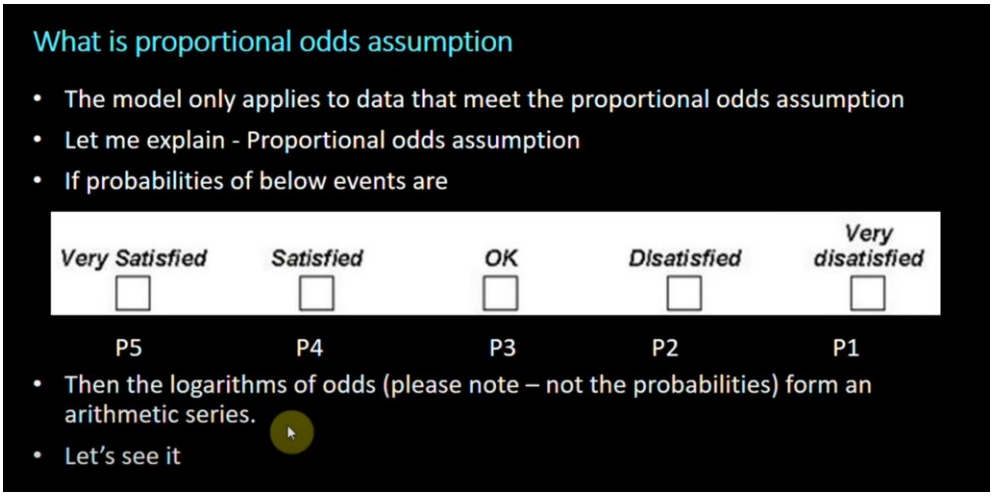
### *Q: What is the difference between linear and logistic regression?*

A: Linear predicts continuous variables; logistic predicts probabilities.

## Logistic Reg and Ordinal data :

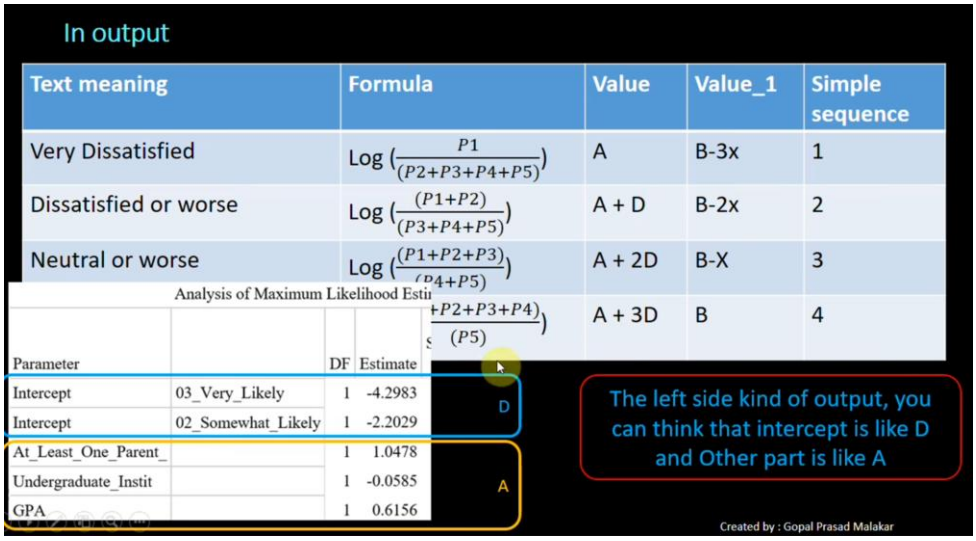
(what is ordinal? very good – good)

Ref video in MOOC\Modelling\Logistic\Ordinal Logistic Regression.mp4



In output

| Text meaning | Formula | Value | Value_1 | Simple sequence |
|---|---|---|---|---|
| Very Dissatisfied | $\text{Log}\left(\frac{P1}{(P2+P3+P4+P5)}\right)$ | A | B-3x | 1 |
| Dissatisfied or worse | $\text{Log}\left(\frac{(P1+P2)}{(P3+P4+P5)}\right)$ | A + D | B-2x | 2 |
| Neutral or worse | $\text{Log}\left(\frac{(P1+P2+P3)}{(P4+P5)}\right)$ | A + 2D | B-X | 3 |
| Satisfied, Neutral or worse | $\text{Log}\left(\frac{(P1+P2+P3+P4)}{(P5)}\right)$ | A + 3D | B | 4 |




## Poisson Regression:

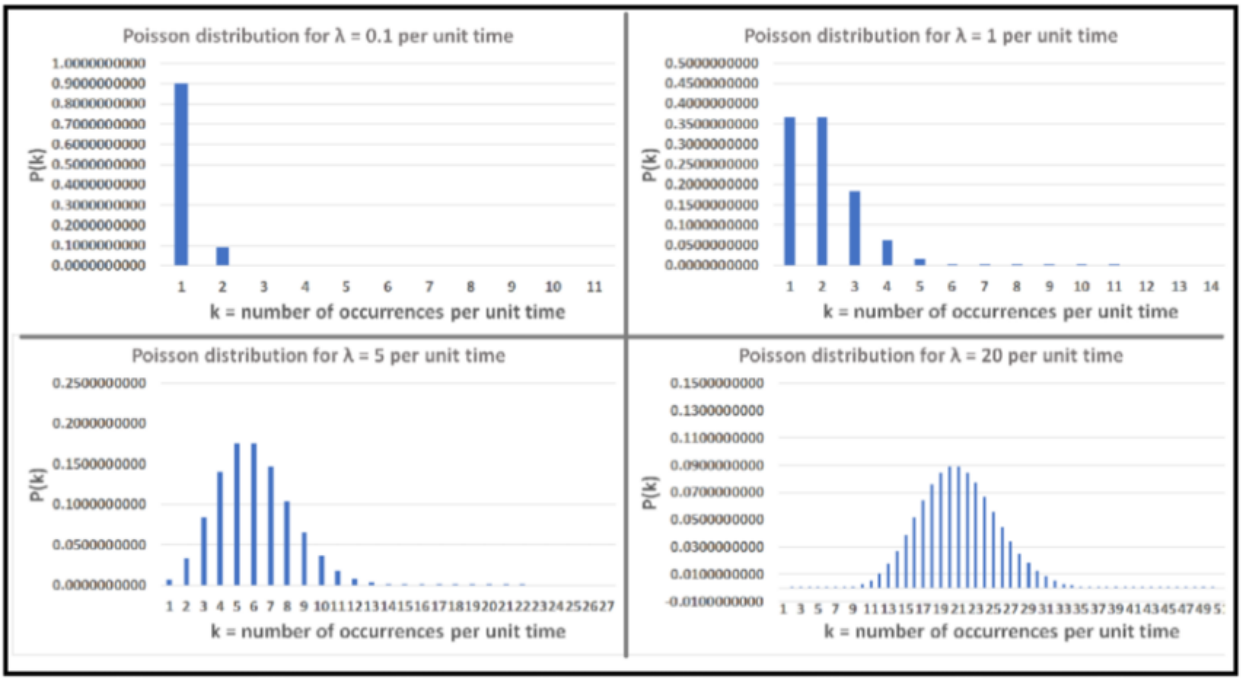
$$P_X(k) = \frac{e^{-\lambda} * \lambda^k}{k!}$$

Given we know that there are on average 5 customers/day at a restaurant, thus lambda = 5 and the time unit is day

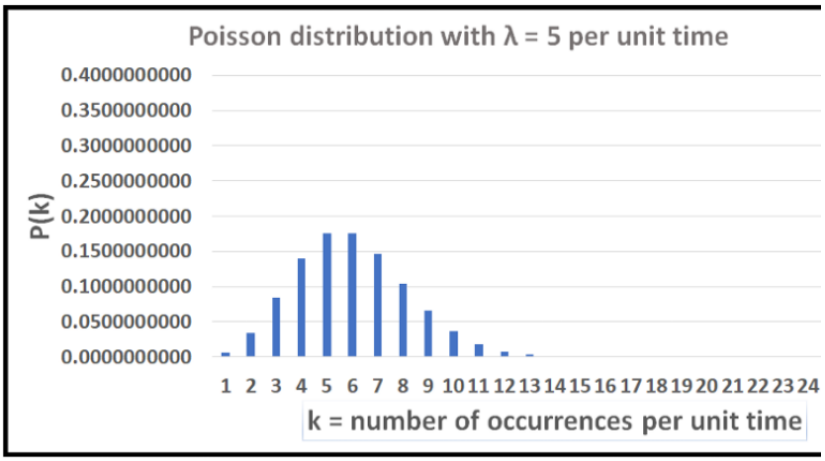
Now, We can find the probability of k customers per unit of time or (per day) using above formula.

Using lambda the PMF or PDF can be drawn like:

Below lambda is changed 0.1, 1, 5 and 20. note: graph peaks for lambda value




Take one lambda = 5, say here lambda is the number of custmers at restaurant, unit of time = day




Here each of the bars gives the probability of k customers per day like P(1customer), P(2custs)…etc

Using this formulae, to model the likelihood of k patients in t days we draw out the PMF with lambda * t like,

$$P_X(k) = \frac{(\lambda t)^k * e^{-(\lambda t)}}{k!}$$

# Classification Modeling:

**Popular metrics for classification:** “f1” (default), “log_loss”, “precision”, “accuracy”, “roc_auc”, “PR-AUC”

## Classification – Model Evaluation Metrics:

1. Confusion matrix

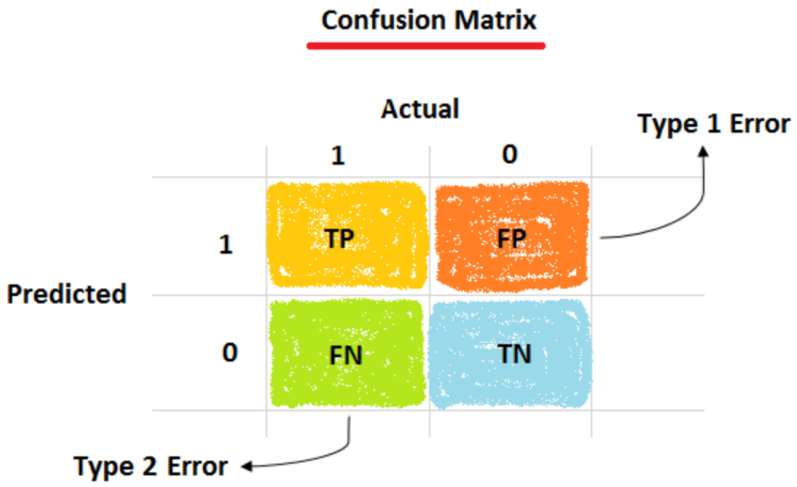
2. Metrics: Precision, Recall, Accuracy, Specificity

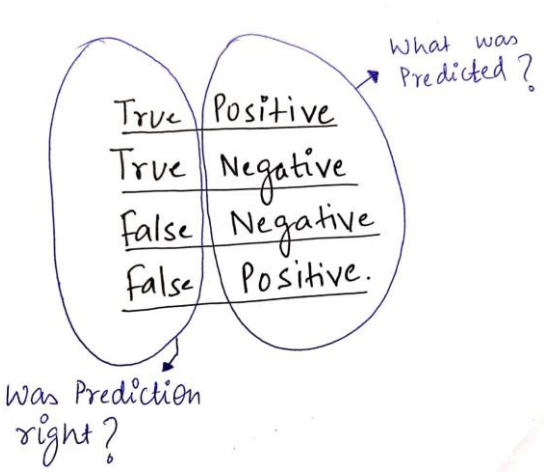
3. AUC-ROC curve

4. PR-AUC: PrecisionRecall AUC helps when the data is imbalanced in deciding the best model

# Confusion Matrix

Classification metric in Supervised Learning






**Actual Positive = 1**

**Actual Negative = 0**

**True = Right:** **(you read as ->)** Right positive, Right negative

Correctly +VE, Correctly -VE

**False = Wrong:** **(you read as ->)** Wrong positive, wrong negative

Should have been -VE, should have been +VE

| Actual | Predicted | | |
|---|---|---|---|
| 1 | 1 | True Positive | |
| 0 | 0 | True negative | |
| 0 | 1 | False Positive | Type I err |
| 1 | 0 | False Negative | Type II err |

TP

| | |
|---|---|
| TP+FP | Precision |
| TP+FN | Recall |
| F1 score | Harmonic mean of P and R |

**Accuracy:**

How much of the total was it correct?

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

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

**False Positives and False Negatives are the most unwanted elements and contentious aspects in any experiment result hence two important metrics exist on tracking each, Precision and Recall or both (F1 score).**

## Precision "within positives"

**aka Positive prediction rate / lower false positive :**

- **Expectation is lower False positives which happens when Precision increases in numeric value.**
- **all about how bad are the false positives or reducing FP**

Use Case: **Spam Detection or Credit Card Fraud Activity Detection**: FP is when a normal email is incorrectly classified as Spam.
We definitely want a spam classification with highest precision aka lowest False Positives.

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

Other Analogies:

- **Target Practice in Gun range**: Precision tells how many hit target vs how many missed.

## Recall "Recall/Frisk many(False positives) just so we dont let even one wrong(False negative) in"

aka Sensitivity / Probability of detection / True Positive Rate (TPR):

- **Expectation is lower False Negative which happens when Recall increases in numeric value.**
- **all about how bad are the false negatives or reducing FN**

Use Case: **TSA Bomb Detection**: FN is when a dangerous material or bomb is wrongly allowed in.
It is totally ok to mistake a innocent item as dangerous and investigate it but not ok to let even one wrong item through. Lowest False negative. Dont let the wrong one in

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

Other analogies:

| Metric | Formula | Priority When | Use Case Example |
|---|---|---|---|
| **Precision** | TP / (TP + **FP**) | Least False Positives (FP) | Spam detection |
| **Recall** | TP / (TP + **FN**) | Least False Negatives (FN) | TSA Bomb Detection |
| F1 Score | $2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$ | When Precision and Recall are equally important | Balanced systems |

## F1 Score:

- **Harmonic mean of Precision and Recall.**
- **When Precision and Recall are equally important**

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

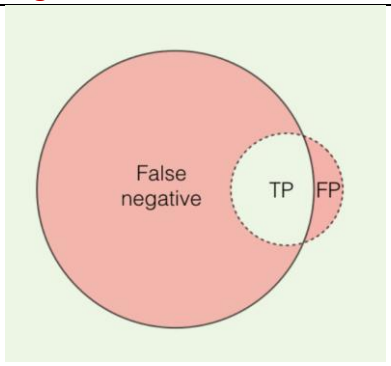
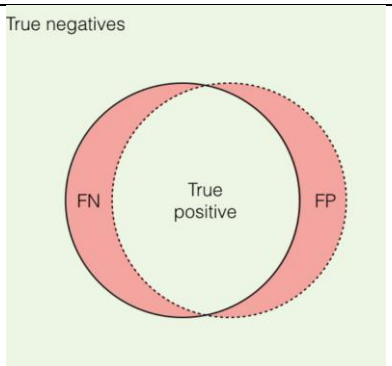
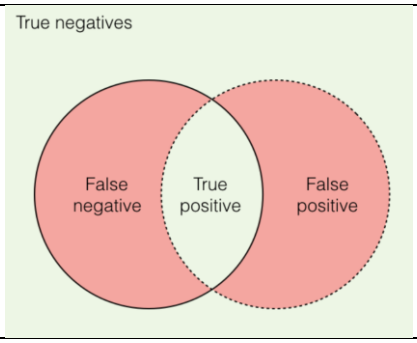
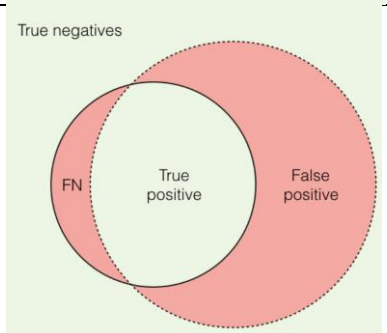
**Specificity:**

How much of all the correctly negative cases (incl FP, ones that were missed/miss categorized) were caught.

S = $\frac{TN}{(TN + FP)}$

## Precision and Recall:

https://medium.com/@klintcho/explaining-precision-and-recall-c770eb9c69e9

| | | |
|---|---|---|
| | Low Recall/High Precision -> High FN/Low FP | High Precision or High Recall -> Low FP and Low FN |
| -----Precision -----> |  |  |
| |  |  |
| | Low Precision/low Recall -> High FP/High FN | Low Precision/High Recall -> Low FN/High FP |
| | -----Recall ----> | |

## AUC – ROC for Binary Classification:

(comparison metric for classification models like the Elbow graph for deciding K in Kmeans)
(when multiple confusion matrices from multiple config experiments are involved)

**ROC** helps in visualizing the trade-off between True Positive Rate (TPR) and False Positive Rate (FPR) at different classification thresholds.

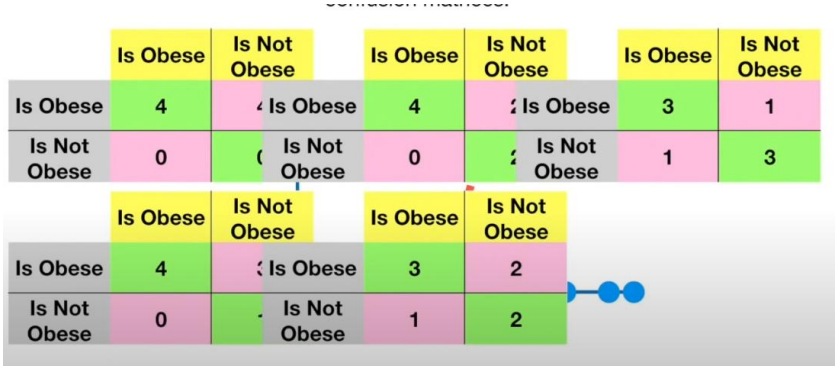
The **Area Under the** ROC**Curve (AUC)** helps in quantifying the performance of a classification model.

AUC – ROC together is a measure of the ability of a classifier to distinguish between classes.

https://www.youtube.com/watch?v=4jRBRDbJemM

- AUC-ROC is a quick way to compare different configs of same model (ROC Only needed) or between different ML Algos like LogReg or RandomForest

- In classification, the confusion matrix tells the full picture of how **<u>one</u>** model-config did. So when many are involved, it is not easy comparing each of the CM. AUC-ROC exists and so it is popular!




This is where AUC-ROC comes to help.

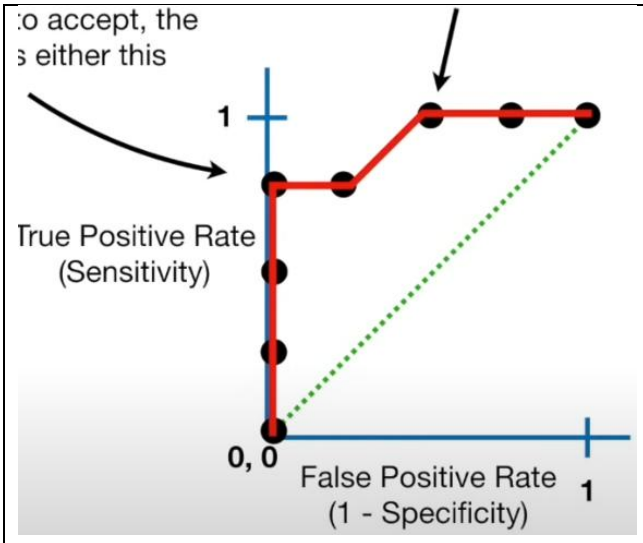
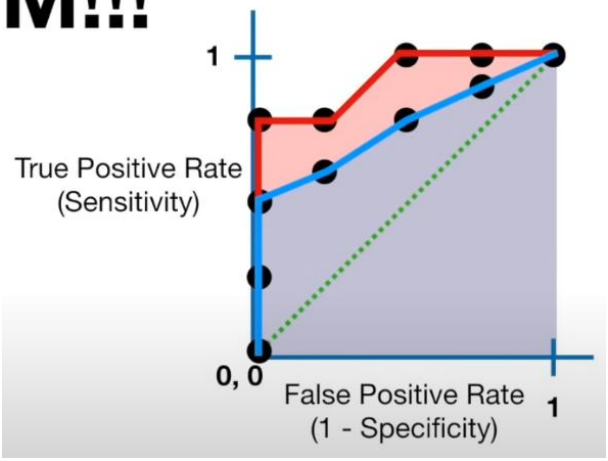
## Receiver Operator Characteristic (ROC)

ROC metrics measure **the ability to distinguish between the classes.**

**ROC Curve is plotted on a chart with TPR on Yaxis and FPR on Xaxis.**

| $Sensitivity = \frac{TP}{TP + FN}$ | $FPR = \frac{FP}{TN + FP} = 1 - Specificity$ |
|---|---|
| What fraction of the real Positive Class did we catch (mark True)?<br><br>TPR is a mark of a model's ability to distinguish between positive and negative classes. | False Positive Rate |

Example: Lets do LogReg. In Logreg, we can change the threshold point at which the classification changes between 1 and 0. It is usually 0.5 but we can change it according to each case. In such a case, we change the thresholds and run the classification model and each black dot below is one of the runs

<table>
<tr>
<td> 
</td>
<td>M!!!<br> 
<br>Higher area Algo is better. Higher area indicates higher TPR</td>
</tr>
<tr>
<td>Compare different configs of same model using ROC curve.<br>• Red curve is the ROC Curve.<br>• Each dot is one run with a certain config.<br>• The two dots pointed above are the best two points and based on need one should pick between them<br>• The point leftmost point has 0 FPR and a high TPR while the other one has highest TPR while least FPR relative to worser ones. If you want lowest FPR, then leftmost point.</td>
<td>Compare different models using AUC<br>• Now if you are trying different models like LogReg and RandomForrest, then AUC helps in deciding instead of doping zillion confusion matrices<br>• Red area is LogReg and BlueArea is RF. More Area means more TPR and so Red is better<br>• higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.<br>• When AUC = 1; identify all positive and negative 100%<br>When AUC = 0; all positive as negative and negative as positive 100%<br>When AUC = 0.5, worst, cant distinguish positive from negative and results seem random or constant/fixed result<br>When 0.5>AUC>1, then AUC is better at identifying <u>true positives and true negatives</u> then 0>AUC>0.5 which is better at false <u>positives</u> and false <u>negatives</u></td>
</tr>
</table>

Thus without having to bunch of confusion matrices we can quickly decide which config ot model to go with.

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification (yes/no problem) problems. It is a probability curve that plots the **TPR (True Positive Rate)/Recall/Sensitivity** against **FPR(False Positive Rate) in various configs** and essentially **separates the 'signal' from the 'noise'**.

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

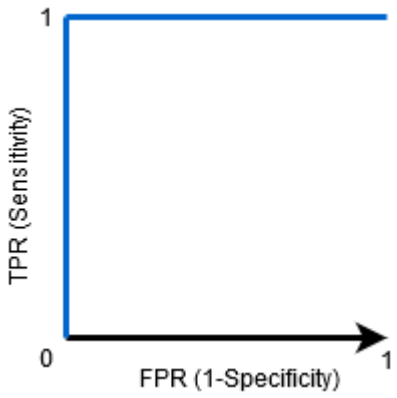
$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

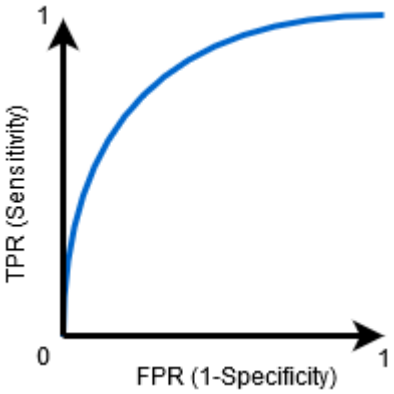
**AUC (Area Under the Curve)**:

- Helps in **quantifying** the performance of a classification model.
- AUC ranges from **0 to 1**:
  - **0.5** → No discrimination (random guessing).
  - **> 0.7** → Acceptable.
  - **> 0.8** → Good.
  - **> 0.9** → Excellent.
- A higher AUC means the model is **better at distinguishing between classes**.

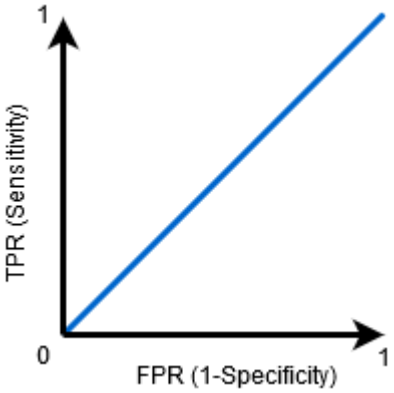
The higher the AUC, the better the performance of the model at distinguishing between classes.




When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.




When 1<AUC<0.5, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.




When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

## Precision or Recall or AUC-ROC

Precision is more focused in the positive class than in the negative class, it actually measures **the probability of correct detection of positive values**,

while FPR and TPR (ROC metrics) measure **the ability to distinguish between the classes**.

### PR-AUC:

PrecisionRecall AUC helps when the data is imbalanced in deciding the best model

## Final intuition to metric selection

https://towardsdatascience.com/what-metrics-should-we-use-on-imbalanced-data-set-precision-recall-roc-e2e79252aeba

1. **Use precision and recall to focus on small positive class —** When the positive class is minority (imbalanced-data) and the ability to detect correctly positive samples is our main focus (correct detection of negatives examples is less important to the problem) we should use precision and recall.
2. **Use ROC when both classes detection is equally important —** When we want to give equal weight to both classes prediction ability we should look at the ROC curve.
3. **Use ROC when the positives are the majority or switch the labels and use precision and recall —** When the positive class is larger we should probably use the ROC metrics because the precision and recall would reflect mostly the ability of prediction of the positive class and not the negative class which will naturally be harder to detect due to the smaller number of samples. If the negative class (the minority in this case) is more important, we can switch the labels and use precision and recall (As we saw in the examples above — switching the labels can change everything).

---

# Model: Naive Bayes Classifier

**https://www.youtube.com/watch?v=Q8l0Vip5YUw**

Naive Bayes Classifier is a simple yet powerful probabilistic and **supervised** ML algorithm used for classification tasks. It is based on Bayes' Theorem, with the key assumption that all features (or predictors) are conditionally independent.

**Naïve Bayes Classifier:**

1. **Separates training data** into "Yes" and "No" groups.
2. **Calculates probabilities** of each feature within these groups.
3. **Applies Bayes' Theorem** to compute P(Yes | features) and P(No | features) for the two groups separately.
4. **Chooses the higher probability** as the prediction.

This keeps it simple while emphasizing the Bayes' Theorem step.

Why Naive?
NB Classifier makes an assumption that the factors tracked or involved are independent (have no dependency) of

each other. This assumption is big and in many cases might not be blatantly wrong, but NB does a decent job even if the factors don't seem independent.

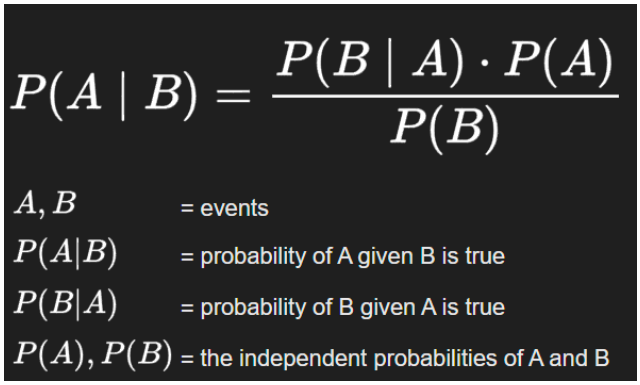
**Use Cases:**

- The predicted class is Binary/ usually Yes or No. in the sense is the email a spam or not given it contain the words "nigerian prince".
- Will a Tennis match happen or not given, weather, rain, temperature
- Will person default loan given demographic details of Loanees like age, gender, education, address etc.

### *How It Works:*

**NB uses training data on tracked past conditions and results(predicted class). NB calculates the different probabilities it needs to apply in the modified Bayes theorem formula. It then calculates the probability for each target class (Yes and No) and picks the class with highest probability as answer.**

1. **Bayes' Theorem:** The classifier uses Bayes' Theorem to calculate the probability of each class.




**<u>All said, the usage is a simpler formula than Baye's theorem or Conditional Probability formula above.</u>**

2. **Estimation:** For each class, the probability of a data point belonging to that class is calculated as the product of the probabilities of each feature given that class.
3. **Prediction:** The class with the highest probability is assigned as the predicted class for the given data point.

**Formula Used:**

The formula used in NB is simpler than Bayes Thm or Conditional Probability.

$$P(\text{Yes}|A,B,C) = \frac{P(A,B,C|\text{Yes}) \times P(\text{Yes})}{P(A,B,C)}$$

A,B,C are factors like weather, rain, temperature

Although Baye's thm is used and above is the formula wch is bit complex to fill, NB classifier uses a simpler formal

$$P(\text{Yes}|A,B,C) = P(\text{Yes}) \times P(A|\text{Yes}) \times P(B|\text{Yes}) \times P(C|\text{Yes})$$

Where :

**Numerator:**

Naive Bayes Classifier makes a naive independence assumption, which simplifies the Bayes formula. This assumption states that the features (factors) A,B,C are conditionally independent and thus numerator component just becomes:

$$P(A,B,C|\text{Yes}) = P(A|\text{Yes}) \times P(B|\text{Yes}) \times P(C|\text{Yes})$$

**Denominator: P(A,B,C)**

In NB, the term P(A,B,C) need not be calculated since in the comparison to see which is greater, the denominators cancel each other. We just need to know which of the classes has higher probabilities and the comparison form is as below where the denominators cancel each other.

So as per Bayes thm, the prob is :

$$\frac{P(A,B,C|\text{Yes}) \times P(\text{Yes})}{P(A,B,C)} \text{ vs } \frac{P(A,B,C|\text{No}) \times P(\text{No})}{P(A,B,C)}$$

Since both the denominator is the same term, we can simply calc numerator and find which is bigger term.

So both probabilities for Yes:

$$P(\text{Yes}|A,B,C) = P(\text{Yes}) \times P(A|\text{Yes}) \times P(B|\text{Yes}) \times P(C|\text{Yes})$$

And No: similar to above

Are calculated every time and the biggest probability is the solution.

## Implementation in Scikit-learn:

- **Scikit-learn's Naive Bayes implementations** (like `GaussianNB`, `MultinomialNB`, and `BernoulliNB`) follow this principle. They compute the numerator directly without calculating the full Bayes' Theorem denominator.
- These models internally calculate the product of the prior probability P(class) and the likelihood terms P(features|class), which is sufficient to determine the most likely class.

**Way it works:**

Naïve Bayes does not "learn" weights and saves them to be used for prediction like neural networks or linear regression.

- NB **stores probabilities in a table** from the training data (like P(feature | Yes) and P(feature | No)).
- When a **new data point arrives**, it **retrieves** these precomputed probabilities and applies Bayes' Theorem to classify the point.

## When Should It Be Used?

**Naive Bayes Classifier** is particularly well-suited for the following situations:

1. **Text Classification:** It is widely used in text classification tasks like spam detection, sentiment analysis, and document categorization. The independence assumption works well with text data, where the presence of certain words can be considered independent given the class.
2. **High-Dimensional Data:** It performs well on high-dimensional data, such as text data, where the number of features (e.g., words) is large compared to the number of data points.
3. **Real-Time Predictions:** Due to its simplicity and fast computation, Naive Bayes is effective for making real-time predictions.
4. **When Features are Independent:** The Naive Bayes classifier works best when the features are truly independent of each other given the class label. While this is often not the case in real-world data, the classifier can still perform well even with correlated features.

## Are There Better Classifiers Today?

Yes, there are more advanced classifiers that often outperform Naive Bayes, especially when the independence assumption does not hold. Some of these include:

1. **Logistic Regression:** A linear model that works well for binary classification tasks and can handle correlated features better than Naive Bayes.

2. **Support Vector Machines (SVM):** SVMs are powerful classifiers that work well in high-dimensional spaces and are effective when the relationship between features is complex.
3. **Decision Trees and Random Forests:** These are non-linear models that can capture complex interactions between features and are often more accurate than Naive Bayes.
4. **Gradient Boosting Machines (e.g., XGBoost, LightGBM):** These ensemble methods are among the most powerful classifiers available and often achieve state-of-the-art performance on structured data.
5. **Neural Networks:** For tasks with a large amount of data and non-linear relationships, neural networks can outperform Naive Bayes, especially in complex problems like image recognition or natural language processing.

## Simple Use case and workflow:

Predict whether a tennis match with happen or not given past results.

| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* | *PlayTennis* |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

1) Training data: We need training data as above that has tracked the different factors and the Target class which is Yes or No.

2) The prior and likelihood terms are calculated based on the training data

3) After Training, during application, these pre-calc terms are simply plugged in and formula calculated for both the classes. The class with the highest term is returned

---

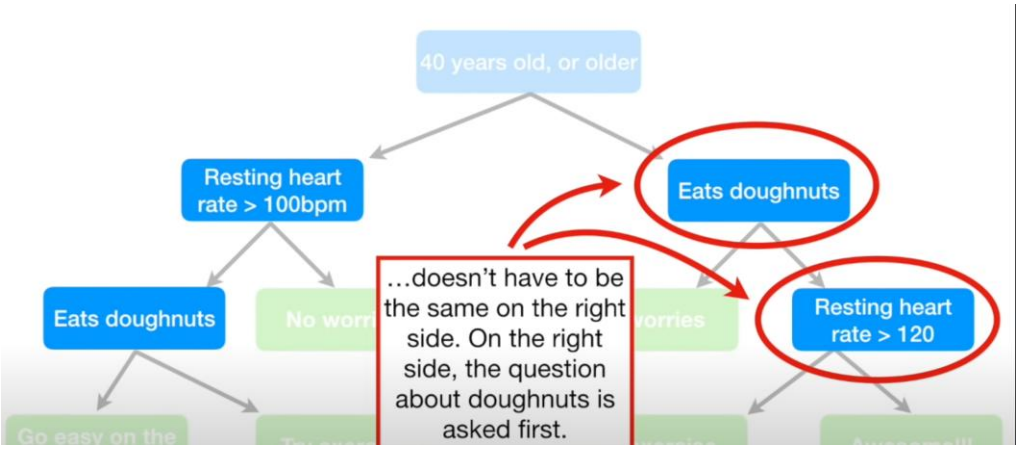
---

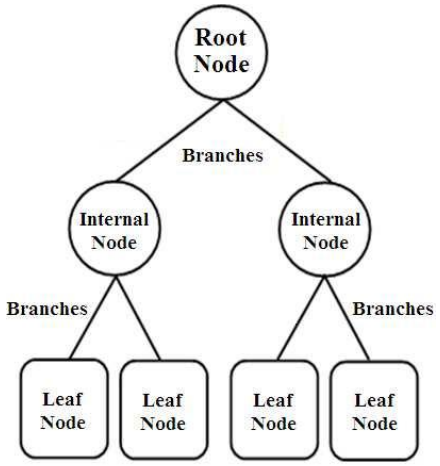
## Model: Decision Tree Classifier

Breaks items into two and classifies based on some estimation and repeats until no more split.

It is like a flowchart where each decision box is called a node

- The question asked to split a node can be repeated in a different branch
- Splits can be made based on numeric data or categorical data (yes/no)






(a)

## Key DT mechanism:

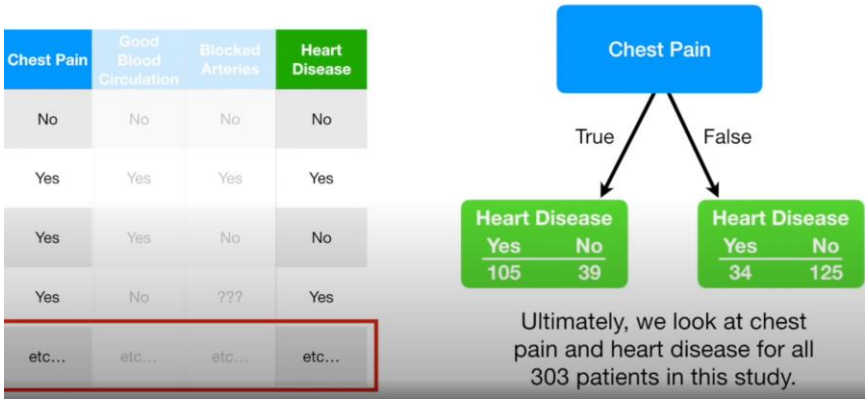
- DT is all about finding the best Threshold for each feature that will give the best split. Then the best Feature-Threshold pair is selected from all the calculated thresholds for each feature.
    - Thus the best Feature-Threshold pair at each node of the Tree that allows to split between the classes is found and used. And repeating the same at each node until the Leaf node which is the end node.
- How the Feature-Threshold pair is calculated is by using the Total Gini Impurity Coeff
    - the technique differs for the datatype of the feature be it Numeric, Binomial (Categorical Binary like Yes/No, 1/0) or Categorical-Ordinal (1,2,3) or Categorical-Nominal (Blue, Green, Red)

**First the mechanism for Categorical Binary (aka Binomial) yes/no variables:**

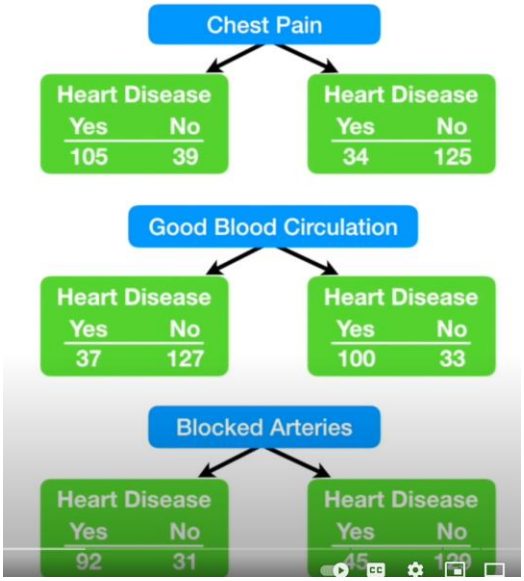
Eg: Heart disease with few yes/no variables. Heart disease is response while other factors are predictors.

| Chest Pain | Good Blood Circulation | Blocked Arteries | Heart Disease |
|---|---|---|---|
| No | No | No | No |
| Yes | Yes | Yes | Yes |
| Yes | Yes | No | No |
| Yes | No | ??? | Yes |
| etc… | etc… | etc… | etc… |

- Algo looks at each of the features “Chest pain”, “Blocked..” etc. and splits them into YES/NO according to Target Var - Heart Disease YES/NO s, all YES Heart Disease will be in one grp and NO Heart Diseases data points in other grp.
- Then, within the group, Each Features own Yes/No counts are found
  Eg: For Chest pain – YES Heart Disease grp what is the count of YES and No and same for ChestPain -NO Heart Disease grp where the count of Yes & No
  - Note: the split did not results in pure case as in all yes s did not lead to heart disease and this is workflow on how each node in DT ends up with impurity.

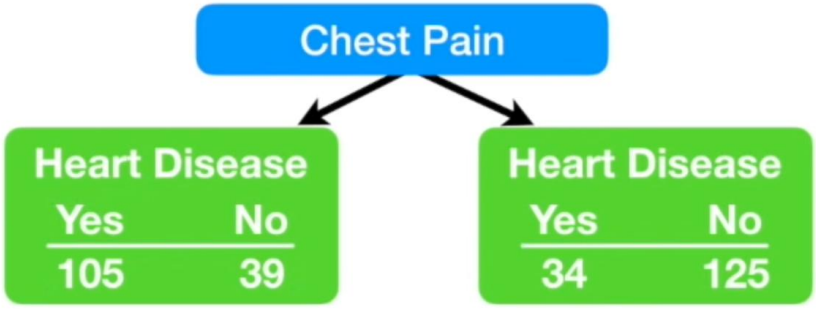



- Same is does for every feature:




- Next, Feature Gini Impurity Coeff is found for each feature and each Group (Yes and No)
  - Feature Gini Impurity coeff is the inverse probability of the split

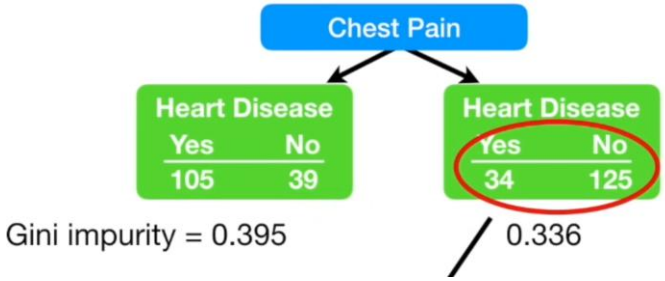
$$\text{Gini impurity} = 1 - (\text{the probability of “yes”})^2 - (\text{the probability of “no”})^2$$




| | |
|---|---|
| $= 1 - (\frac{105}{105 + 39})^2 - (\frac{39}{105 + 39})^2$<br>$= 0.395$ | $= 1 - (\frac{34}{34 + 125})^2 - (\frac{125}{34 + 125})^2$<br>$= 0.336$ |
| | |

- Same is done for the other node as above

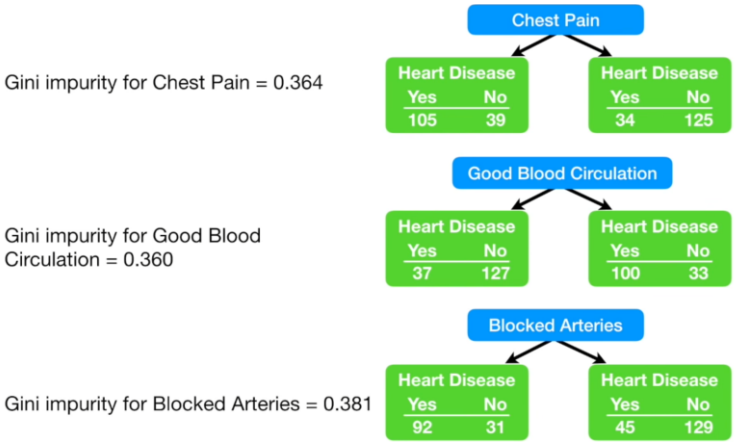
- Now answers of the two nodes are “weighted averaged” to get the Total Gini Impurity Coeff for ChestPain




Calc the Weighted Average to get the Total Feature Gini Impurity Coeff

$$= (\frac{144}{144 + 159})\ 0.395 + (\frac{159}{144 + 159})\ 0.336$$

- Thus Total Feature Gini for each feature is found

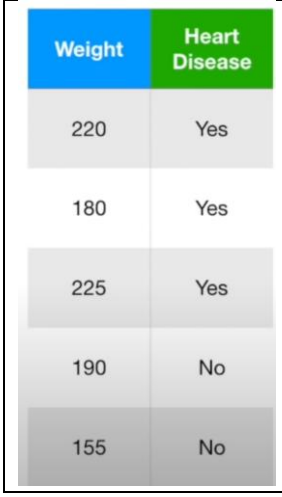
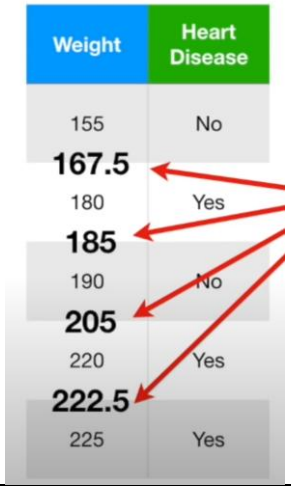


- The factor with lowest Gini impurity (0.36) implies highest purity and so that factor is selected as the threshold value/decision maker and used for splitting that node in the DT.
- The same is repeated to do further splits, the algo stops when the lowest Gini impurity is higher than that used for previous split. Eg: say the lowest for previous split is 0.36 and all possible splits in the next level's best Gini impurity is 0.4, then algo stops and simply makes the node an end node aka leaf node.

**How do we do this for numeric variables since they are not simple yes no?**

- First the numeric values are sorted small to big and starting from the smallest, the coeff is calculated for this one value versus all higher values, then higher value selected **which includes all lower values in the bucket** vs rest of the higher values, Gini Coeff calculated and then next higher value till the end. In the end, from the sequence of Gini's calculates, lowest one is picked as the decision maker.

- With more details,
  First the feature is organized in increasing order:

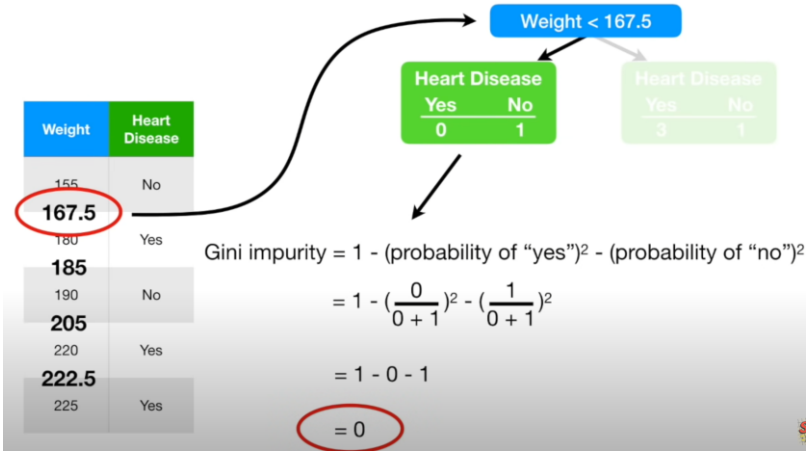
| As is | First, sort | Second, calculate average between two cell numeric values |
|---|---|---|
|  |  |  |



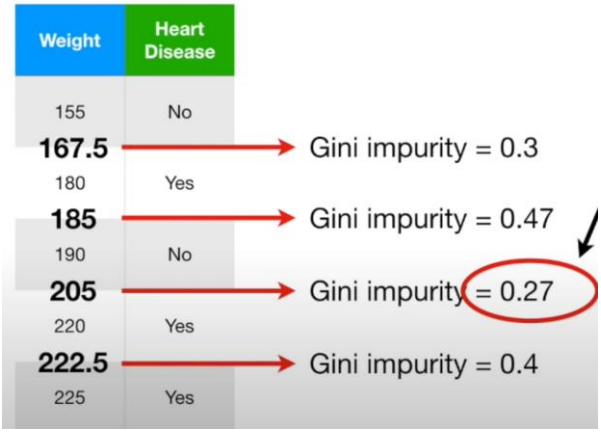




- Then calc, Gini impurity coeff with the first mid/avg weight (wt) in one bucket and higher wts in other bucket, as in Weights < 167.5 is one bucket and rest other: the two gini coeffs are found and then weighted avg calculated

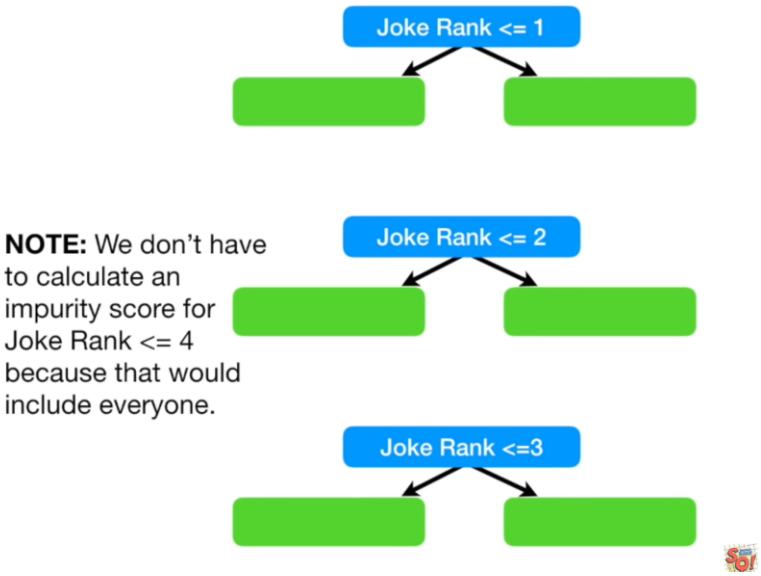



- The same is done for each avg-mid cell wt. and finally, the lowest weighted avg. coeff is selected as the decision maker.




## How do we do for ordinal / ranked variables?

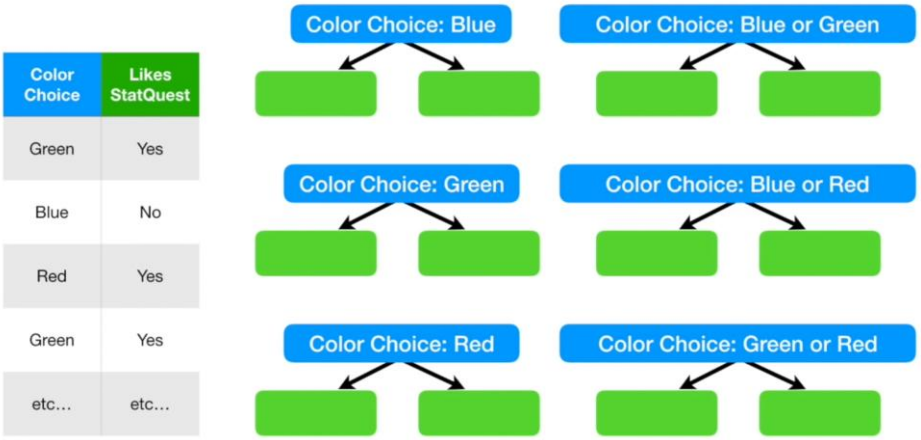
- Ranked/Ordinal like young/adult/old or rank1/rank2/rank3. It is done similar to numeric above, in that <=rank1 alone is used to split, gini coeffs and the weighted avg. coeff is found, then <=rank2 (which include rank1) is used and then <=rank3 (includes rank1 and 2)




- The gini impurity coeff are compared and lowest is selected to make split

## How do we do Categorical-Nominal of groups 3 or more?

If only two categs, then it becomes binomial



| Color Choice | Likes StatQuest |
|---|---|
| Green | Yes |
| Blue | No |
| Red | Yes |
| Green | Yes |
| etc... | etc... |



- The gini impurity coeff is found for:
  - Individual Categ. vs Rest splits : (Blue) vs (Green + Red)
  - combined combinations like Red+Blue vs Green+Pink
- lowest gini impurity coeff which implies highest purity combination is selected for splitting

Additional info on mechanism:

https://towardsdatascience.com/growing-a-random-forest-using-sklearns-decisiontreeclassifier-24e048e8bd84

The algorithm finds the feature-threshold pair (feature-threshold pair is basically “chest pain & 0.360” like in above example) that maximizes the **gain information** and makes a split by evaluating if the value of the selected feature is less than / equal to or greater than the threshold. The threshold is the selected value in the feature that **maximises** the information gain or **minimising** the entropy.

**So what do we mean by information gain? Selecting the best of the worst options available.** First, we need to define **entropy**. Entropy is a measure of the uncertainty about a random variable. If all values of a random variable have the same probability, then we have the maximum entropy. If the random variable can take only one value, then the entropy takes its minimum value.

For example, in a **pure node**, which is leaf node where all the instances were predicted the same class, the entropy would be 0.

Information gain is a measure based on **entropy.** It measures the reduction in entropy when we split a subset according to the selected feature-threshold pair. When the algorithm chooses a feature-threshold pair that maximises the information gain, it is minimising the entropy.

### Key Limitations of Decision Trees:

1. **Overfitting and High Variance**: Decision Trees are highly sensitive to small changes in the data, leading to overfitting.
2. **Low Accuracy**: A single tree might not generalize well to unseen data

## Bagging :

Bagging and Boosting are techniques to improve Tree algos.

Bagging is actually short for Bootstrap Aggregating + choosing best. Bootstrapping is a technique in Stats where samples are taken with replacement. The results and learnings from the different Bags are combined to give superior performance.  Random Forest is basically this. Bagging if leads to multiple models/runs and the best result of which is chosen then that is an Ensemble model.

## Model: Random Forest

Random Forest is an ensemble learning method that combines the predictions of multiple Decision Trees to improve accuracy, robustness, and generalization. It can be used for both **classification** and **regression** tasks. The key idea is to build a "forest" of Decision Trees, each trained on a random subset of the data and features, and aggregate their results.

DT s have limitations and is not robust for large datasets. Which is where RF comes into place.

Random Forest Mechanism:

1. **Bootstrap Sampling**: For each tree, a random subset of the training data (with replacement) is sampled. This is known as **bagging (Bootstrap Aggregating)**.
2. **Feature Subset Selection**: **At each split in a tree**, only a random subset of features is considered for determining the best split. This introduces randomness and reduces overfitting. It also ensures that one tree does **not correlate** with another tree in the RF.
3. **Building the Forest**: Multiple Decision Trees are trained independently using the steps above.
4. **Aggregation**:
    - For classification: Combine predictions using majority voting.
    - For regression: Combine predictions using averaging.

1. Create a new dataset of same features from the original dataset by bootstrapping "with replacement".
2. From the "Random" dataset, pick a random number of features and build a decision tree
3. Step1 and 2 above are repeated many many times such that we get many completed DTs with randomly picked dataset and random set of features.
4. Now when a new data comes for predition, it is run through each of the DTs and count of the classification is obtained.
5. The classification with majority gets selected
6. Further, some data rows gets missed due to bootstrapping picking method. These missed rows for each tree is run through all the trees and the proportion of correctly/incorrectly guessd ratio is found
7. Further optimization: then the DT created with 2 variables is compared with 3 variables and another RF is built.
8. Best is selected in the end

**Random Forest** is an Ensemble type model that uses Bagging since it combines multiple decision trees to make predictions.

Missing data and clustering in RF

- Missing data or predictions on new data are guessed by RF and the guesses are optimized using proximity matrix
    - A row is said to be similar to another if both end of up in the same leaf node.
    - **Proximity Matrix** is an item-item matrix where points awarded when they each end up in same leaf node in all the trees created.
    - The points are then scaled by dividing by total number of trees found. This proximity is used to predict a value after RF is created

## Why Random Forest ?

1. **Ensemble Learning**: Random Forest combines multiple Decision Trees (each trained on a random subset of the data and features) and aggregates their outputs, typically by majority voting (for classification) or averaging (for regression). This reduces overfitting.

2. **Random Feature Selection**: During training, each **SPLIT IN the TREE** considers only a random subset of features for splits. This ensures that one tree does not correlate with other trees in RF and improves robustness.

3. **Better Generalization**: By averaging predictions across multiple trees, Random Forest achieves higher accuracy and stability, especially on unseen data.

## Advantages of Random Forest

1. **Robust to Overfitting**: By averaging multiple trees, Random Forest reduces the risk of overfitting compared to a single Decision Tree.
2. **Handles Non-Linearity**: Captures complex relationships without requiring feature transformations.
3. **Feature Importance**: Provides insights into which features are most important in predictions.
4. **Works with Missing Data**: Can handle missing values in the dataset without much preprocessing.

## Random Forest and Missing Data:

Scikit-learn's Random Forest doesn't handle missing data natively and in fact will crash with ValueError if there is missing data.

That said, other library RandomForest do have some neat features:

**Surrogate Splits**: Surrogate splits are used to handle missing values during tree building by finding alternative predictors (features) that closely mimic the split made by the primary feature. Libraries like **H2O.ai** support surrogate splits natively.

**Weighted Splits**: Adjusts splitting criteria to account for missing data by weighting instances with missing values instead of discarding them. XGBoost has options**.**

**Proximity-Based Imputation :** Uses the proximity matrix (a similarity measure between observations generated by the Random Forest) to estimate missing values based on the most similar observations. missingpy.missForest can do it.

**Iterative Imputation:** Missing values are imputed iteratively using predictions from other features in the dataset, often leveraging models like Random Forest for imputation. Scikit's sklearn.impute.IterativeImputer can do it explicitly before fitting the RF.

## DT vs RF

While a Decision Tree excels in interpretability and small datasets, Random Forest shines in situations where predictive accuracy is critical, and the dataset is large or complex. DT's feature importance are not that worthy since it is single tree as those obtained from RF.

## Popular Usage of Random Forest

1. **Classification** : Spam detection, Credit risk modeling, Medical diagnostics.

2. **Regression** : Predicting house prices, Estimating customer lifetime value, Weather prediction.

3. **Feature Selection**: The importance scores derived from Random Forest can guide the selection of relevant features in a dataset.
4. **Anomaly Detection**: Used in detecting outliers in high-dimensional data.

## Considerations When Using Random Forest

1. **Hyperparameter Tuning**: Key parameters include `n_estimators` (number of trees), `max_depth` (depth of trees), and `min_samples_split` (minimum samples to split a node).
2. **Computational Cost**: Random Forest can be resource-intensive for large datasets since it involves training multiple trees.
3. **Not Ideal for Very High Dimensional Data**: In cases like text or image data or many columns in tabular dataset, methods like Gradient Boosting or Neural Networks might perform better. If text and image, RF struggles with sparse or the just the thousands/millions of data points making it even more compute/memory heavy. Even in tabular data, the high number of columns don't workout during the random subsetting of features. In High dimension case, it is advised to first do feature selection or Dimensionality Reduction.

## Proximity Matrix

A simple n X n matrix of each record/row with points. Points are granted when two records ends up in same leaf node.

The proximity matrix in Random Forest (RF) is a tool that measures the similarity or "closeness" between data points based on how often they fall into the same leaf node across all trees in the forest. It is primarily used for clustering and anomaly detection tasks rather than classification or regression.

### *How the Proximity Matrix is Built:*

1. **Initialization:** Start with an n×n matrix (where n is the number of data points), initialized with zeros.
2. **Tree Traversal:** For each tree in the forest:
    - Traverse the tree and record which data points land in the same leaf node.
    - Increment the corresponding cell in the matrix for pairs of points that share a leaf.
3. **Normalization:** After processing all trees, normalize the matrix by dividing each cell by the total number of trees.

**Proximity Matrix Interpretation:**

- **Values in the Matrix:**A high value (≈1) between two data points indicates that two points frequently fall into the same leaf node and are thus considered similar. While a low value (≈0) means the points rarely share a leaf node.

**Limitations:**

- **Computational Cost:** can be computationally expensive, especially for large datasets.
- **Memory Intensive:** For large datasets, the n×n matrix can require significant memory.

**Proximity Matrix Usage:**

1. **Clustering:** The matrix can be treated as a similarity matrix for clustering algorithms like k-means or hierarchical clustering. Helps group similar data points based on their behavior in the Random Forest.
2. **Anomaly Detection:** Outliers tend to have low proximity to most other points. Anomalies can be identified by their lower average proximity to other data points.
3. **Visualization:** The proximity matrix can be used with dimensionality reduction techniques like **t-SNE** or **MDS (Multidimensional Scaling)** to visualize data structure in a lower-dimensional space.
4. Missing Value imputation: If datapoints are missing RF can use the proximity to take data from similar

While **NOT** typically used in standard RF applications, it provides additional insights.

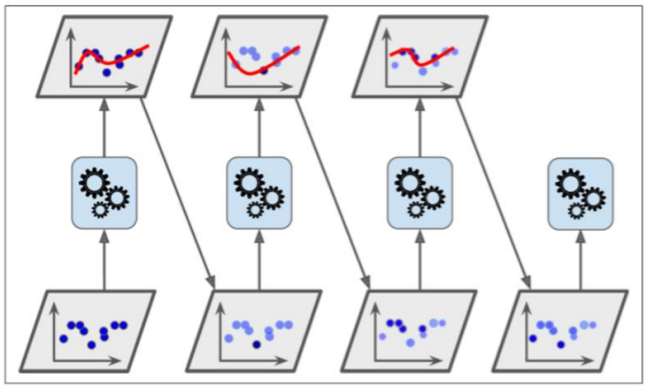
# Boosting and GBMs:

**Boosting** is an ensemble technique where each new model corrects errors made by the previous trees. Boosting is when the errors or residuals are focused on to improve the previous Tree thus always improving the Tree.

## Gradient Boosting Machine (GBM):

- **Gradient Boosting** is a specific type of boosting technique where trees are trained using **gradient descent** to minimize the loss function.
- The main idea is to fit each new tree to the residual errors (differences between predicted and actual values) of the previous tree.
- **Gradient Boosting Machine (GBM) are fundamentally different from RF in that in RF, multiple trees are parallelly built (Bagging technique) while in Gradient Boosting, Trees are done one after another, sequentially, where learning from one are carried forward to next.**

| | |
|---|---|
| **Rnadom Forest** | Multiple trees built parallely under Bagging technique |
| **Gradient Boosting Tree** | Trees built Sequentially |



**SEQUENTIAL GBM**

- Popular GBM frameworks:
    - **XGBoost**: An optimized and scalable implementation of gradient boosting.
    - **LightGBM**: A faster and more efficient implementation that uses histogram-based algorithms.
    - **CatBoost**: An implementation that works well with categorical features without needing pre-processing.

# XGBoost

**XGBoost (eXtreme Gradient Boosting)** is an advanced implementation of gradient boosting. It is widely used in machine learning competitions and real-world applications due to its performance, efficiency, and flexibility. **XGBoost** is a pre-set highly optimized GBM Decision Tree algorithm that is known to have high performance.

Why Extreme?

**1. In Built Regularization**: built in L1 (Lasso) and L2 (Ridge) regularization.

**2. Handling Missing Data:** Automatically learns the best direction for missing data splits during training.

**3. Cross-Validation**: Built-in cross-validation methods for fine-tuning hyperparameters.

**4. Parallel Processing: O**ptimized for parallel computation, **Traditional GBM is often sequential**, making it slower.

**5. Memory Usage:** uses a specialized data structure called the "block structure" to manage sparse datasets **efficiently**. This helps it handle large-scale datasets that wouldn't fit into memory.

**6. Tree Pruning:**

- Uses "max_depth" instead of "depth-first" growth to control tree size.
- Employs a "minimum loss reduction" criterion to prune trees early.

**7. Custom Cost Functions:** allows users to define custom loss functions, adaptable and flexible.

**8. Scalable and Distributed:** scalable across distributed systems, making it suitable for big data applications.

## Popular Use Cases for XGBoost

1. **Classification Tasks**:
    - Predicting customer churn.
    - Fraud detection in banking and e-commerce.
    - Disease diagnosis in healthcare.
2. **Regression Tasks**:
    - Demand forecasting.
    - Predicting financial market trends.

3. **Ranking Tasks**:
    - Search engine ranking.
    - Recommendation systems.

### Important XGB Parameters

1. **Tree Parameters**:
    - `max_depth`: Maximum depth of a tree (controls model complexity).
    - `min_child_weight`: Minimum sum of instance weights needed in a child (prevents overfitting).
    - `gamma`: Minimum loss reduction required to make a further split (prunes trees).
2. **Boosting Parameters**:
    - `learning_rate` (or `eta`): Controls the contribution of each tree.
    - `n_estimators`: Number of trees to grow.
    - `subsample`: Fraction of samples used to grow each tree.
3. **Regularization Parameters**: `lambda`: L2 term & `alpha`: L1 term.
4. **Objective Parameter**:
    - `objective`: Defines the type of task (e.g., `binary:logistic` for binary classification, `reg:squarederror` for regression).

## LightGBM (Light Gradient Boosting Machine)

**Uses Histogram based approach (memory usage &speed better); Leaf wise growth vs Depth wise in XGBoost; Better for large datasets with many columns**

**LightGBM** is an implementation of gradient boosting that is designed to be more efficient and scalable than traditional gradient boosting algorithms like **XGBoost**. It uses **histogram-based algorithms** to speed up the training process and reduce memory consumption, making it particularly useful for large datasets.

*Key Features:*

1. **Speedy Histogram-based approach**:
    - LightGBM uses a **histogram-based** approach for decision tree learning, which allows it to bucket continuous feature values into discrete bins. This significantly reduces memory usage and speeds up the learning process.
2. **Leaf-wise Growth**: LightGBM grows trees **leaf-wise** (selecting the leaf with the maximum delta in loss to grow), which can lead to deeper and more complex trees. This is in contrast to the **depth-wise** growth used by XGBoost. Leaf-wise growth tends to result in lower loss but can lead to overfitting in small datasets.
3. **Categorical Feature Support**: LightGBM natively supports **categorical features** without the need for one-hot encoding.

4. **Speed and Scalability**: LightGBM is highly optimized for speed and can handle **large-scale datasets** efficiently (better than XgBoost) , especially on systems with limited memory.
5. **Regularization**: It includes options for **L1 and L2 regularization** (just like XGBoost).

*Strengths:*

- **Speed**: LightGBM is **faster** than XGBoost on larger datasets because of its efficient data handling and use of histogram-based decision trees.
- **Scalability**: Can handle very **large datasets** (millions of rows and features) with relatively lower memory usage compared to other boosting algorithms.
- **Categorical Data**: Automatically handles **categorical features** without preprocessing
- **Model Performance**: Often provides **better performance** than XGBoost in terms of both speed and accuracy on large datasets, especially when the number of features is very large.

*Weaknesses:*

- **Overfitting in Small Datasets**: Because of its leaf-wise tree growth, LightGBM may suffer from **overfitting** on small datasets.
- **Sensitive to Hyperparameters**: It may require more **tuning** than XGBoost, especially with respect to regularization and tree depth parameters.

---

## CatBoost (Categorical Boosting)

**Uses Ordered Target Encoding which makes it better with too many categs; Symmetric Tree growth against Depthwise(XGBoost) or Leafwise(LightGBM); Easier to use**

**CatBoost** is another gradient boosting algorithm, designed to handle **categorical features** more effectively than XGBoost or LightGBM. It is known for its ability to naively handle categorical data without the need for additional preprocessing (like one-hot encoding or label encoding).

*Key Features:*

1. **Handling Categorical Features**: **CatBoost** directly processes **categorical features** using **ordered target statistics**. This reduces the risk of overfitting and avoids the need for manual encoding.
2. **Symmetric Tree Construction**: Unlike LightGBM's leaf-wise growth, **CatBoost** builds trees in a **symmetric** way (similar to traditional decision trees), where each split on a level of the tree affects all branches, ensuring a more **stable** model.
3. **Boosting on Ordered Leaves**:
    - CatBoost uses a technique known as **ordered boosting**, which helps reduce the **bias** introduced by ordering data (a common problem with traditional gradient boosting methods).
4. **NLP Champ**: CatBoost has specific enhancements for working with **textual data** and **complex feature types** (like embeddings), which makes it a strong candidate for NLP tasks or datasets that contain a lot of complex features.

5. **Speed and Regularization**: Like LightGBM, CatBoost is optimized for speed and supports **regularization** to avoid overfitting.

*Strengths:*

- **Categorical Data Handling**: CatBoost has **native support** for categorical variables
- **Less Tuning Required**: It is designed to be relatively **user-friendly** and often requires **less hyperparameter tuning** compared to XGBoost and LightGBM.
- **High Accuracy**: CatBoost has shown **competitive accuracy** with other gradient boosting models, especially on tabular data.
- **Robustness**: Due to its **symmetric tree construction** and **ordered boosting**, CatBoost is typically more **robust** to overfitting compared to other boosting methods.

*Weaknesses:*

- **Slower on Small Datasets**: While CatBoost is very powerful for large datasets, it may not be the fastest on smaller datasets.
- **Training Time**: Training may still be **slower** compared to LightGBM, especially for large-scale datasets.

## XGBoost vs LightGBM vs CatBoost

| Feature | XGBoost | LightGBM | CatBoost |
|---|---|---|---|
| When to Use Relatively | Small Datasets and when ready to hypertune | For | |
| Handling Categorical Data | Requires encoding (one-hot or label) | Native support. Converts to numbers and then split searches using histograms. No need for one-hot encoding | Native support. uses Ordered Target Encoding which also prevents data leakage between training and validation sets. Minimal processing for large number of categs. |
| Tree Growth | Depth-wise (traditional) | Leaf-wise (faster, but may overfit) | Symmetric tree (more stable) |
| Speed | Moderate, optimized for speed | Very fast, especially for large datasets | Moderate to fast, but slower than LightGBM |
| Performance | Excellent for many tasks | Often better for large datasets | Often competitive, especially on tabular data with categorical features |
| Ease of Use (Tuning | Requires careful tuning | Requires tuning, especially | Relatively easier to use with less |

| | | | |
|---|---|---|---|
| Required) | | for small datasets | tuning needed |
| Overfitting Risk | Can overfit with deeper trees | Can overfit on small datasets | Less prone to overfitting due to symmetric trees |
| Popular Use Cases | General-purpose classification/regression | Large-scale classification/regression | Tabular data with categorical features, robust classification |

## When to use what?

- LightGBM is better, faster and more efficient than XGBoost and great with large datasets. XGBoost is better if dataset is small/medium when LightGBM might overfit.
- XGBoost provides more control levers and so an expert would prefer it over LightGBM which is easier for amateurs
- **LightGBM** might outperform **XGBoost** in terms of speed and memory efficiency for large datasets, the **maturity**, **reliability**, and **established ecosystem** of **XGBoost** contribute to its continued dominance in certain domains

### *Categorical Dataset: LightGBM or CatBoost?*

Considering that both CatBoost and LightGBM are good with categorical data, then which one to choose if dataset has many categoricals.

| Feature | LightGBM | CatBoost |
|---|---|---|
| **Categorical Data** | Good, requires integer encoding | Excellent, no need for preprocessing |
| **Handling of High Cardinality** | Good (uses integer encoding) | Great (ordered target encoding) |
| **Speed and Memory** | Fast, memory efficient | Slightly slower but highly accurate |
| **Training on Large Datasets** | Excellent for large datasets | Can handle large datasets but not as fast |
| **Complex Interactions** | Efficient for simpler categorical interactions | Good at handling complex categorical data |
| **Scalability** | Scalable to very large datasets | Works well on large but not massive datasets |
| **Ease of Use** | Requires manual encoding but efficient | Automatic handling of categorical features |

# Clustering

How to select a clustering method? How to validate a cluster solution (to warrant the method choice)?

Choosing the right linkage method for hierarchical clustering

Hierarchical clustering with mixed type data - what distance/similarity to use?

### Top Clustering Algorithms and Their Mechanisms

| Algorithm | Mechanism | Popular Real-World Applications |
|---|---|---|
| **K-Means** | Iteratively assigns points to the nearest centroid, recalculates centroids, and repeats until convergence. | Customer segmentation, market segmentation, document clustering. |
| **Hierarchical Clustering** | Builds a tree-like structure (dendrogram) by successively merging or splitting clusters. | Social network analysis, gene expression analysis, hierarchical taxonomy. |
| **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** | Groups points based on density; detects outliers as noise. | Fraud detection, anomaly detection, spatial data clustering (e.g., earthquakes). |
| **Mean Shift** | Moves centroids to denser regions iteratively until convergence. | Image segmentation, object tracking. |
| **Gaussian Mixture Model (GMM)** | Assumes data comes from multiple Gaussian distributions; estimates cluster probabilities. | Speaker recognition, financial fraud detection, customer profiling. |
| **Spectral Clustering** | Uses graph theory and eigenvalues to find clusters based on similarity. | Community detection in social networks, image segmentation. |

Let me know if you need deeper insights into any specific one!

## Clustering Model Evaluation:

Confusion Matrix, ROC-AUC, F1 Score, Log Loss

## Distance Calculation:

The method by which distance is measured in a Clustering Algo is controllable factor and this major impact on the results. Choose wisely.

**Euclidean**: Scale of variables affect values hence normalize the variables before calculating distance

**Matching coefficient**: use when absence of a values in multiple columns in a record does not matter

**Jaccard's coefficient**: use when absence of a values in multiple columns in a record matters

# Model: K Means Clustering

- K is first decided (need domain knowledge to decide this)
- Algo picks tree random points (centroids) and calculates distance with every point in the space.
- Based on the closeness, the points are classified into k groups.
- The mean of each group is found and the mean becomes the new centroid
- The distance is calc again and points are classified between the groups

TIPS:

- odd number of k is good since it helps in breaking ties more than even number of clusters

## What k to use? Finding K:

### Elbow Chart

SSE is calculated

### Silhouette coefficient

The silhouette coefficient is a measure of cluster cohesion and separation. It quantifies how well a data point fits into its assigned cluster based on two factors:

> How close the data point is to other points in the cluster

> How far away the data point is from points in other clusters

Silhouette coefficient values range between -1 and 1. Larger numbers indicate that samples are closer to their clusters than they are to other clusters.

In the scikit-learn implementation of the silhouette coefficient, the average silhouette coefficient of all the samples is summarized into one score. The `silhouette_score()` function needs a minimum of two clusters, or it will raise an exception.

### Cluster Separation numbers

Looks at the direct differences between centroids and decide

### Limitations:

- does not work well with categorical data
- distance calculations of categorical(binary) is possible but not great even if you can do it
    - eg. zipcodes – to omany zipcodes and so one-hot encoding will make the table high dimensionality problem

### K-Modes (for categorical data)

- initially, k records are chosen in random as the cluster centroid
- compares each cell within column with other records, if there is a match

### K-prototype (for both numeric and categorical data)

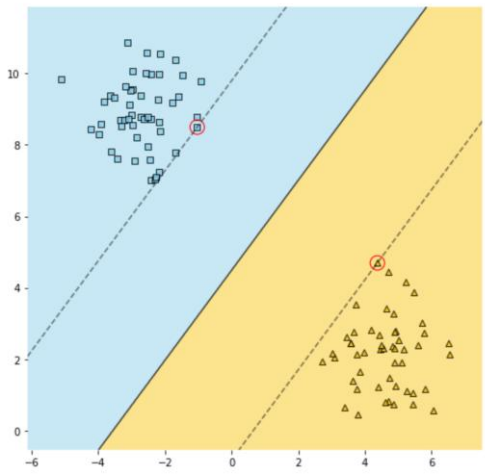
- works well with both numeric and categorical

### DBSCAN

- clustering when there is noise in data

## Support Vector Machines:

SVM simply helps in putting the widest possible hyperplane between set of points separating/classifying them in return. The 'widest' is defined by few datapoints that are close to the separation. These separation vectors are the 'Support' Vectors. (red circled ones below)



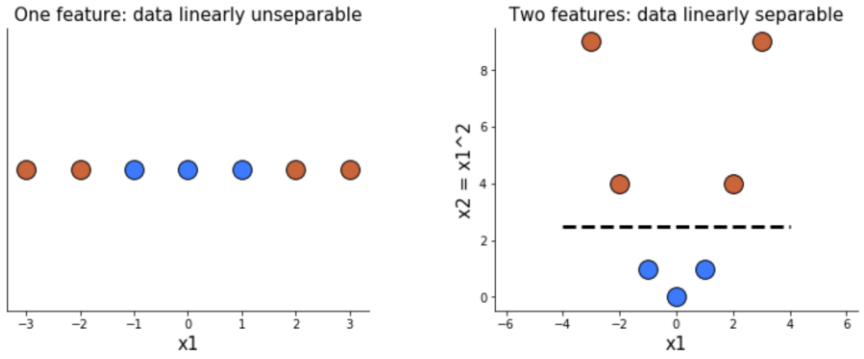

Linear SVM with linearly separable data works pretty well.

As seen above no point is allowed within the band. You can increase the tolerance and allow one or two points within the band such that a even better classification is possible by doing so. This is done by controlling C hyperparameter in Scikit lean SVC.

**SVM for NonLinear Data?**

SVM is thus really good for Linear problems since a straight line (thus linear) is able to separate the classes. Not good for Non-linear decision boundaries **as is**.

A trick against this problem is that some of these non-linear data are linearly separable in higher dimension (as in adding another axis and going from 1d to 2d or 3d etc.). like how x2 dimension is added below and suddenly reds and blues are linearly separable.




Adding another feature makes data linearly separable.

Kernel is the tool that separates these non-linear data in higher dimension and brings it back to lower dimension.

https://towardsdatascience.com/svm-kernels-what-do-they-actually-do-56ce36f4f7b8

Kernels are simply a smart way of adding more features to the data in the hope of making it linearly separable. Smart, because instead of actually adding them, which would make the model slow, they make use of some magical mathematical properties (which are out of scope of this article) that allow us to obtain **exactly the same results as if we had actually added these features**, without slowing the model down.

**SVM Hyperparameter:**

C: allows few points to be tolerated in band/hyperplane area for the benefit of better classification. 1 which is default means no tolerance and none allowed while 0.1 means highest tolerance and this many points or widest band.

kernel:

- Linear – default,
- Poly – adds polynomial features

SVMs outdated:
SVMs are compute heavy and best only for small datasets so read more?

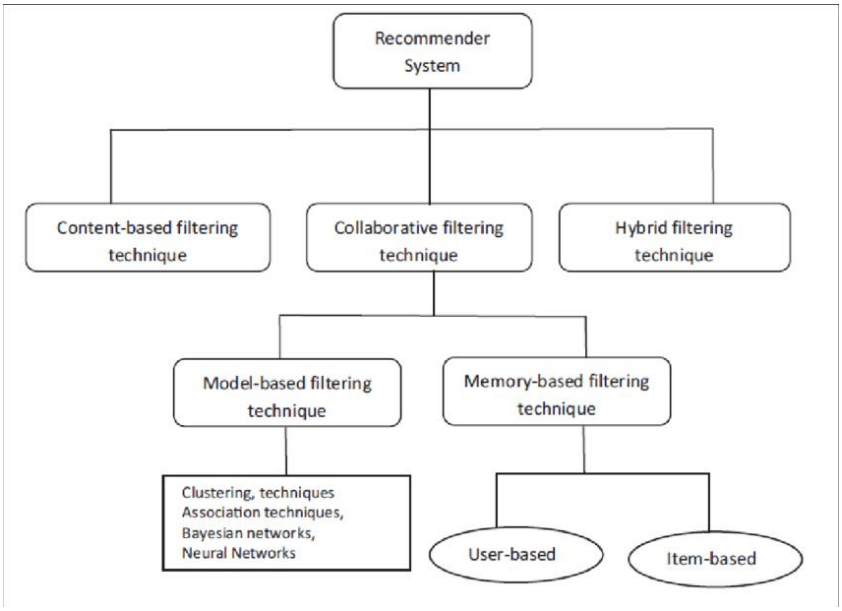
SVMs are still relevant for smaller datasets, where deep learning is overkill, and in scenarios requiring **interpretability**, especially with high-dimensional data. However, as computational power increases and modern ML methods evolve, SVMs are being replaced in many real-world applications.

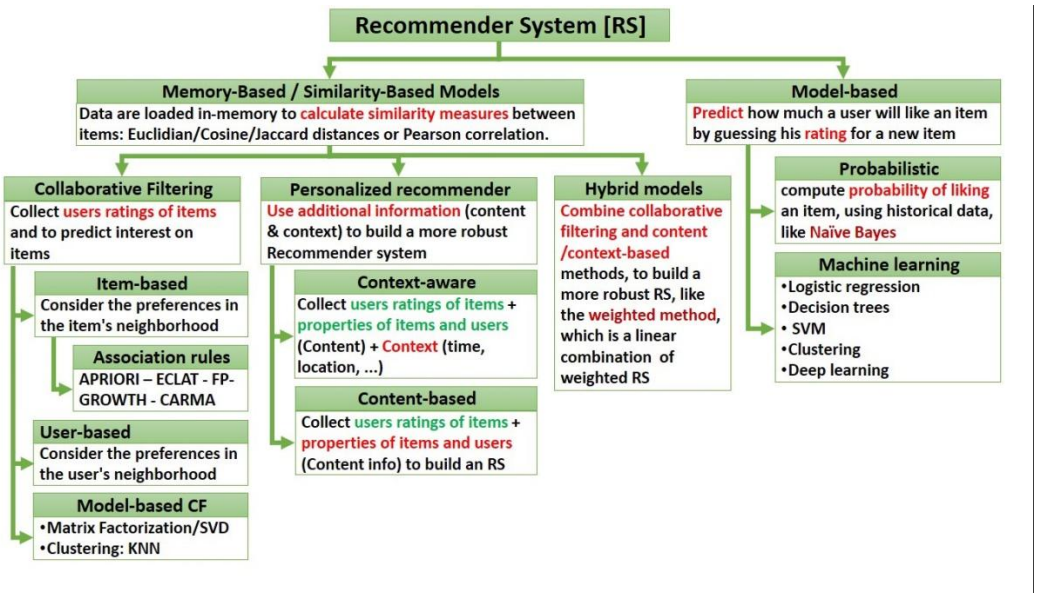
# Recommender Systems:

https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

https://realpython.com/build-recommendation-engine-collaborative-filtering

## Types:






## Definitions:

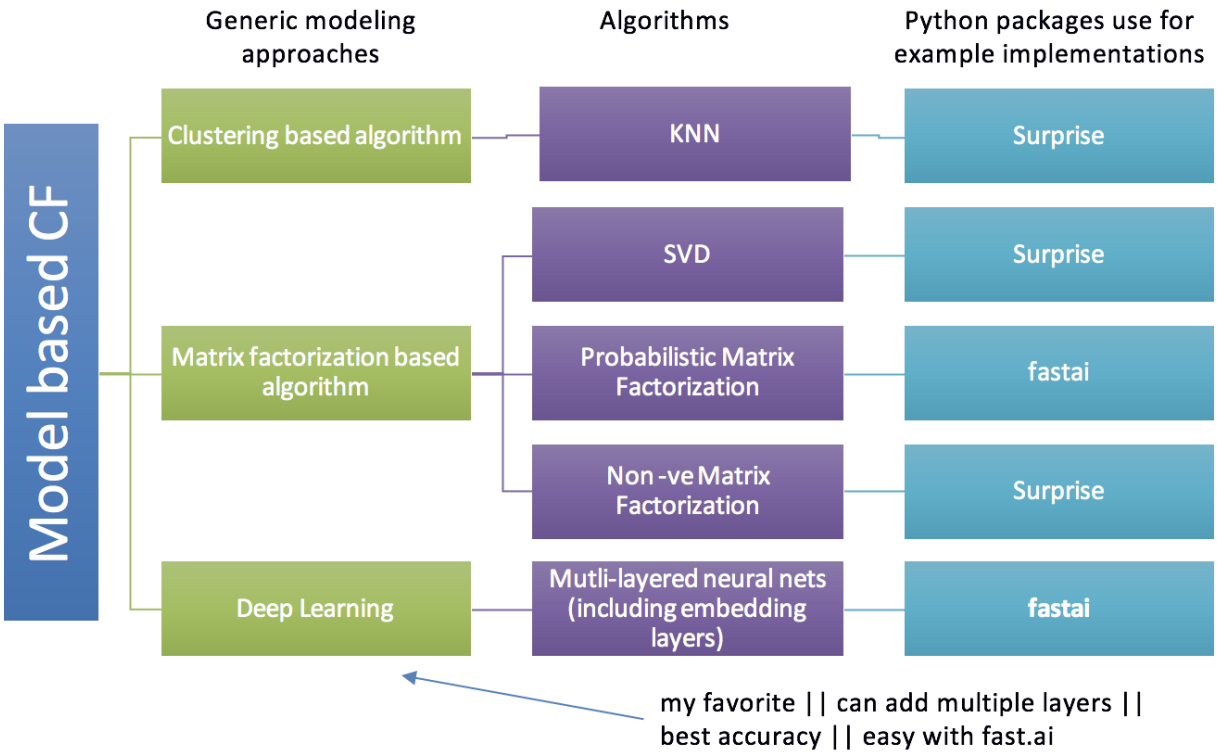
**Content Based**: Find items similar to items liked by user based on the similarities between items.

content-based recommender system the algorithms used are such that it recommends users similar items that the user has liked in the past or is examining currently.

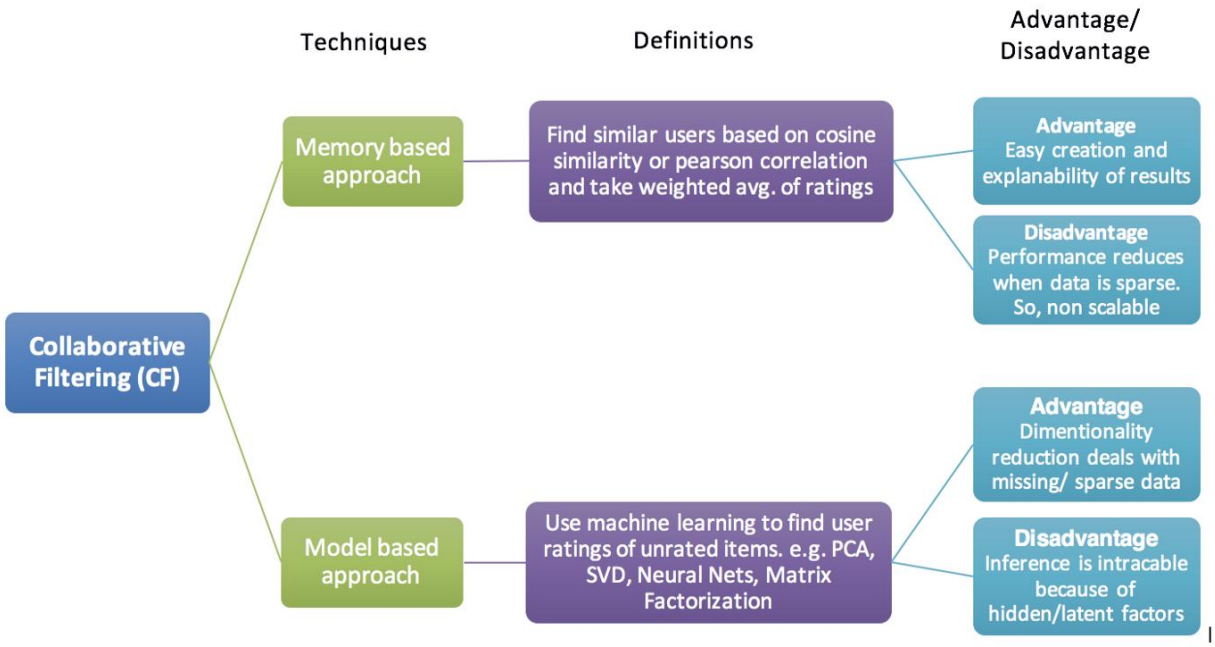
**Collaborative Filtering CF**: Find similar users based on likings and recommend items from each others list

Collaborative recommender systems aggregate ratings or recommendations of objects, recognize commonalities between the users on the basis of their ratings, and generate new recommendations based on inter-user comparisons. **The greatest strength** of collaborative techniques is that they are completely independent of any machine-readable representation of the objects being recommended and **work well for complex objects where variations in taste are responsible for much of the variation in preferences.** Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future and that they will like similar kind of objects as they liked in the past.

**CF - Model based:** predict ratings using models and find product that has a high rating



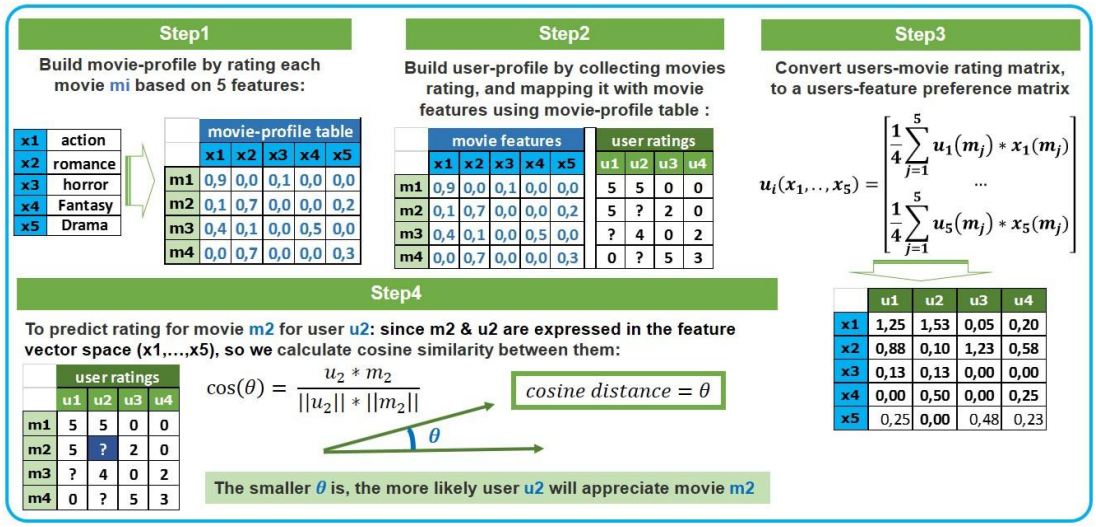

**CF - Memory based:** Find relations with data in memory



CF - Mem - User based (user-item filtering):

“Users who are similar to you also liked …”

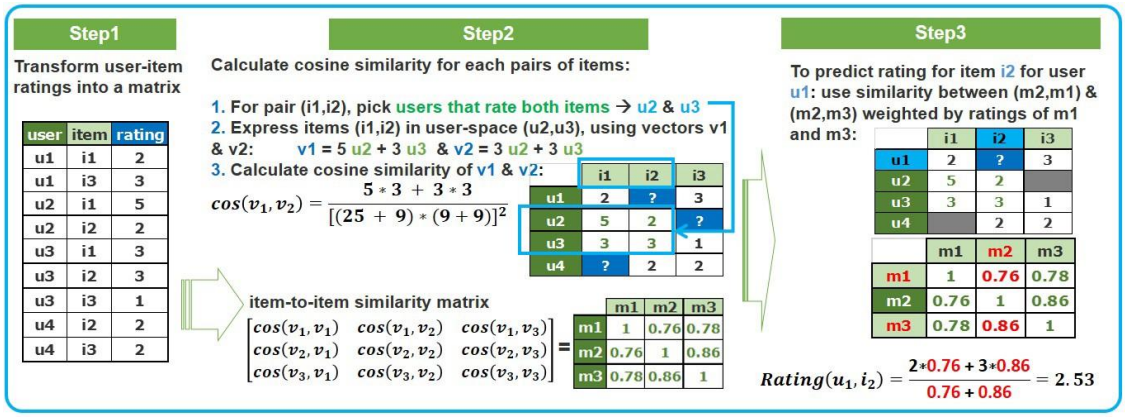
Base recommendation on relationship between users (people who have the same preferences will buy the same items)




CF - Mem - Item based (item-item filtering):

“Users who liked this item also liked …”

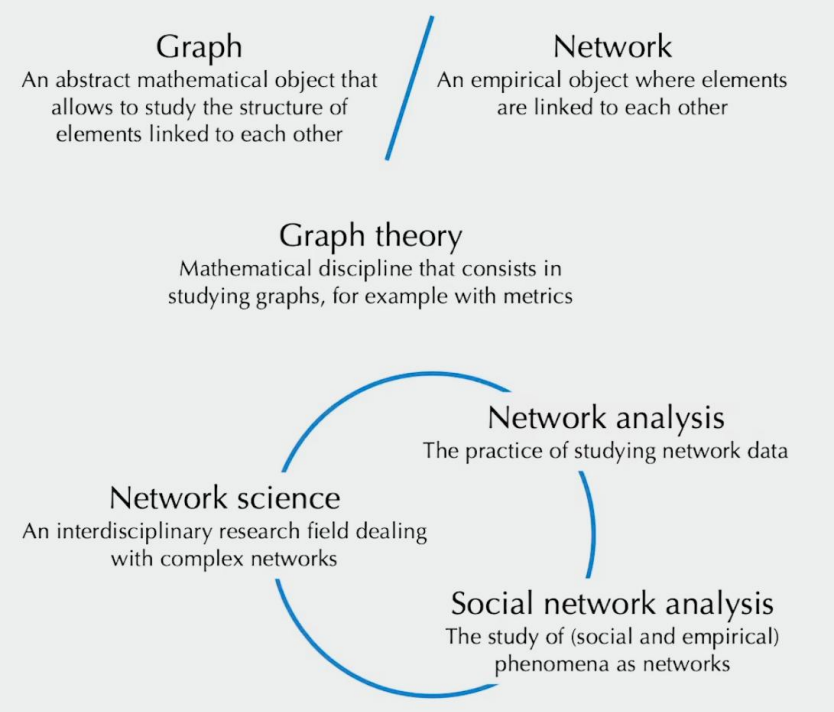
Base recommendation on relationship between items to identify those who are bought together




Normalizing data in CF

In a user-item matrix, certain users are typical like ‘tough rater’ who is possibly a movie critic who rates all moves as average. These tough raters pair as similar with a normal average rater. To take into consideration such differences, we can normalize the data by subtracting the avg. rating of the users all ratings with each of his rating. This brings the average to zero. (standard scaling).

# Network Analysis




- Graph is a mathematical object which is analysed and relationship (numbers) defined using graph theory
- Network is its concrete counterpart. The analysed graph is made into an object called network.

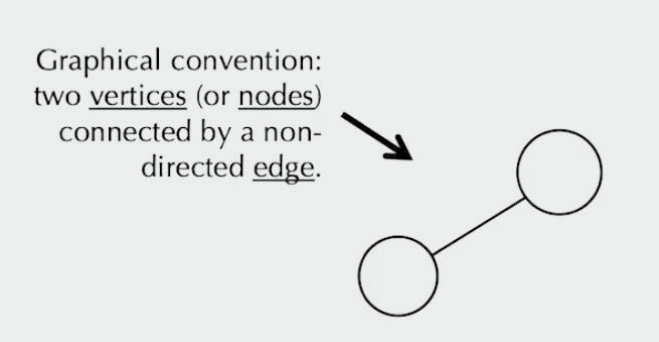
Good Tutorials:

Reducible: https://www.youtube.com/watch?v=LFKZLXVO-Dg

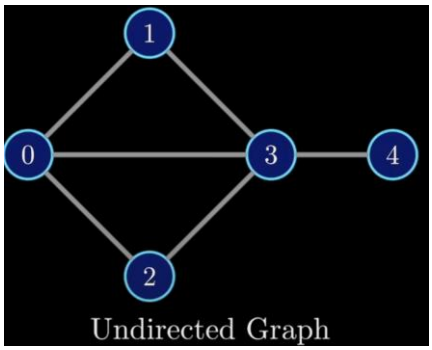
Main Concepts:

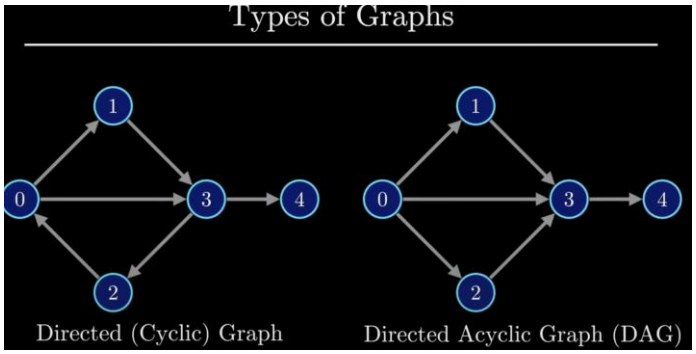
Nodes and Edges

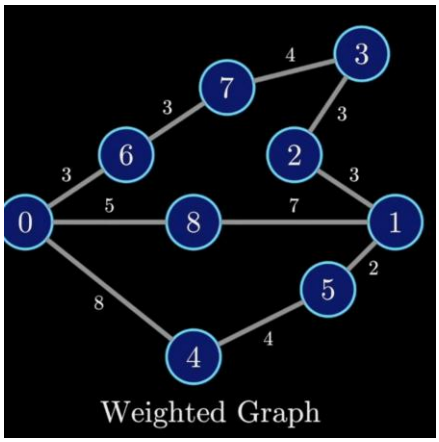
Nodes represent objects in the network and the edges represent the strength of the relationship between them.

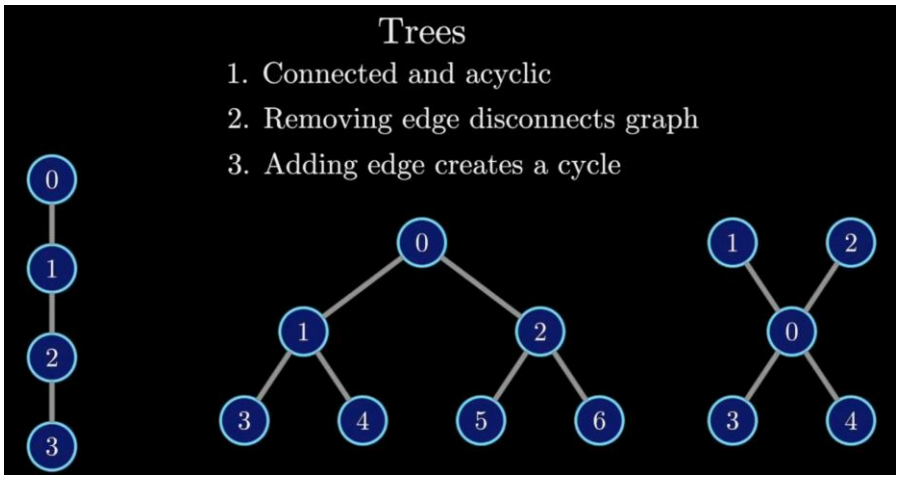



**Types of Graphs**:




**Degree of a Vertex :** Number of edges connected to a vertex.

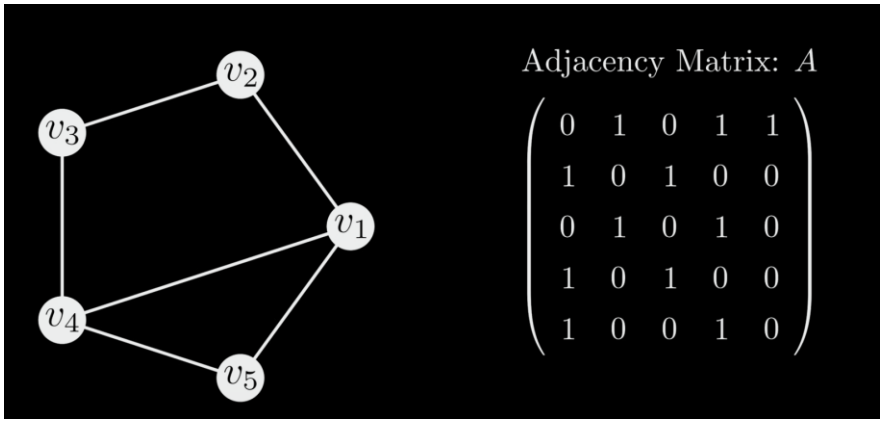
**Neighbours** : Two vertices connected directly by one edge are neighbours

**Path** : Sequence of vertices indicating a specific path in the network.

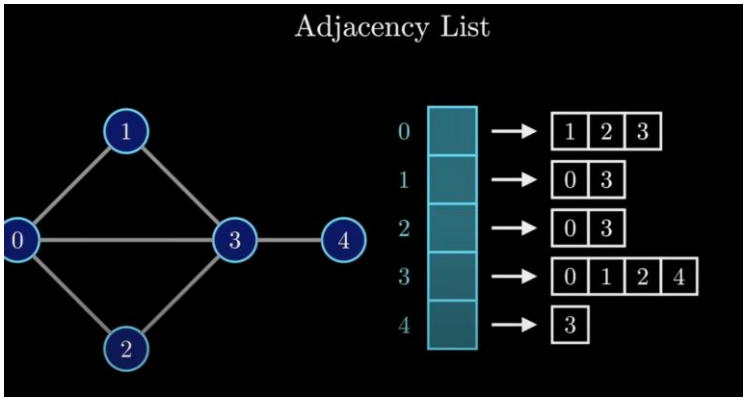
## Matrices / Graph Representations:

**Adjacency matrix**

Captures the connections between vertices




**Adjacency List**:




**Degree Matrix**

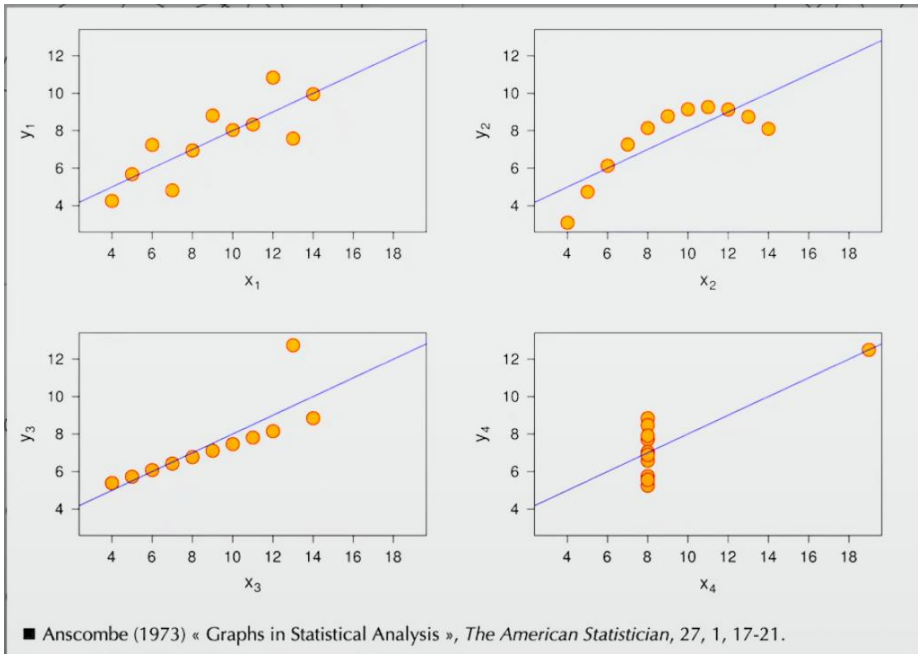
Sum of each row of adjacency matrix which also gives the degree for each node and thus a good quick ref.

Degree Matrix: $D$

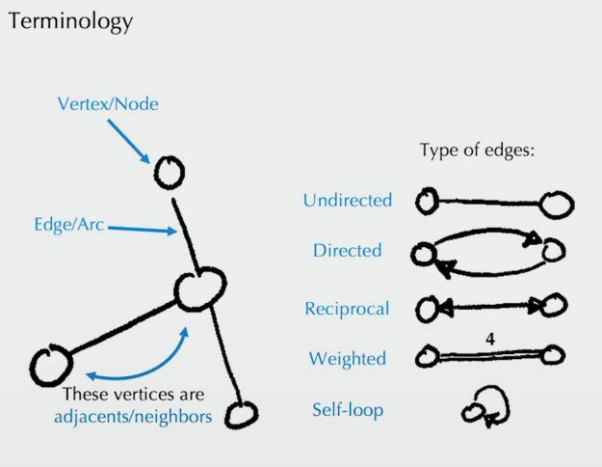
$$\begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

**Why do you need Network / Graph Analysis?**

Below are different datasets with same distribution – same mean, variance and regression but as you can see they seem to be telling entire different stories. Thus if you rely only on stats, you can miss other informatio nbeing presented. This is why context is important and why we need network analysis.




Terminologies:




Spectral Theory

Important for Org. Network analysis

A theory of Eigenvalues and Eigenvectors of graph matrices