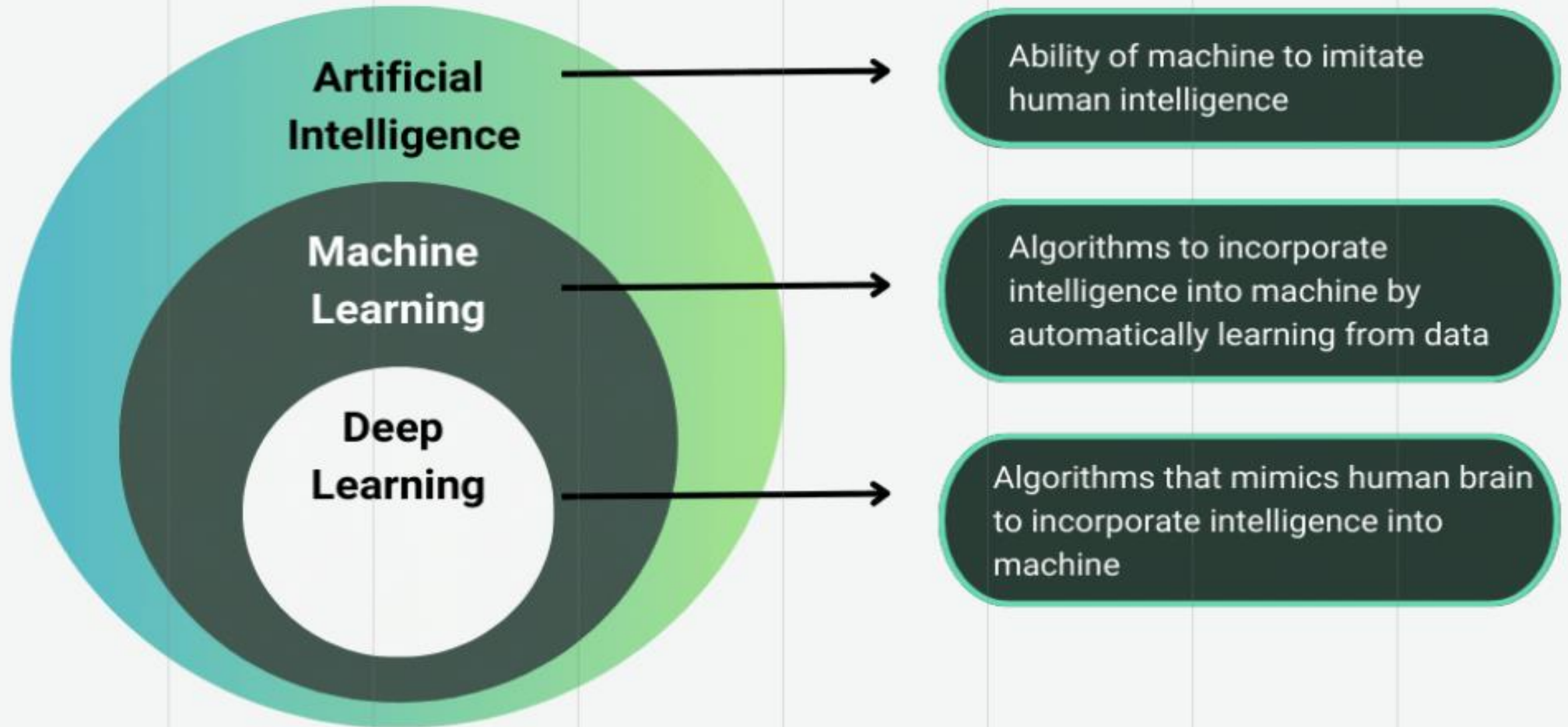


Machine Learning

Prepared By
Vijay Kumar



Artificial Intelligence

- ▶ **Artificial intelligence**, on the other hand, is a technique that enables machines to **imitate human intelligence and behaviour**.
- ▶ It involves developing systems capable of performing tasks that would typically require human intelligence, such as **visual perception, speech recognition, problem-solving, and decision-making**.

Machine Learning

- ▶ ML is a subset of AI, focuses on algorithms and statistical models that allow machines to learn and improve from data (experience) without being explicitly programmed.
- ▶ A machine learning model is trained on a dataset and based on that training, it can make observations or predictions about new, unseen data.
- ▶ Machine learning is an essential component in developing AI systems.

Deep Learning

- ▶ **Deep learning (DL)**, which is now a subset of machine learning, utilizes artificial neural networks with multiple layers to process and learn from vast amounts of data.
- ▶ It aims to mimic the structure and functionality of the human brain.

What is Machine Learning?

- ▶ ML is the subset of computer science that gives “computer the ability to learn without being explicitly programmed”
- ▶ INPUT DATA
- ▶ Feature Extraction
- ▶ A Set of Rules - ML Model
- ▶ OUTPUT

ML - Applications

- ▶ Amazon / Flipkart - recommendation System
- ▶ Loan - finds possible defaulters
- ▶ Telecom - demographic data and plans
- ▶ Chatbots
- ▶ Face recognition

ML Techniques - I

- ▶ Regression / Estimation
 - ▶ Predicting continuous values
- ▶ Classification
 - ▶ Predicting the item class/category of a case
- ▶ Clustering
 - ▶ Finding the structure of data; summarization
- ▶ Associations
 - ▶ Associating frequent co-occurring items/events

ML Techniques - II

- ▶ Anomaly Detection

- ▶ Discovering abnormal and unusual cases

- ▶ Sequence Mining

- ▶ Predicting next events; click stream (Markov model, HMM)

- ▶ Dimension Reduction

- ▶ Reducing the size of data (PCA)

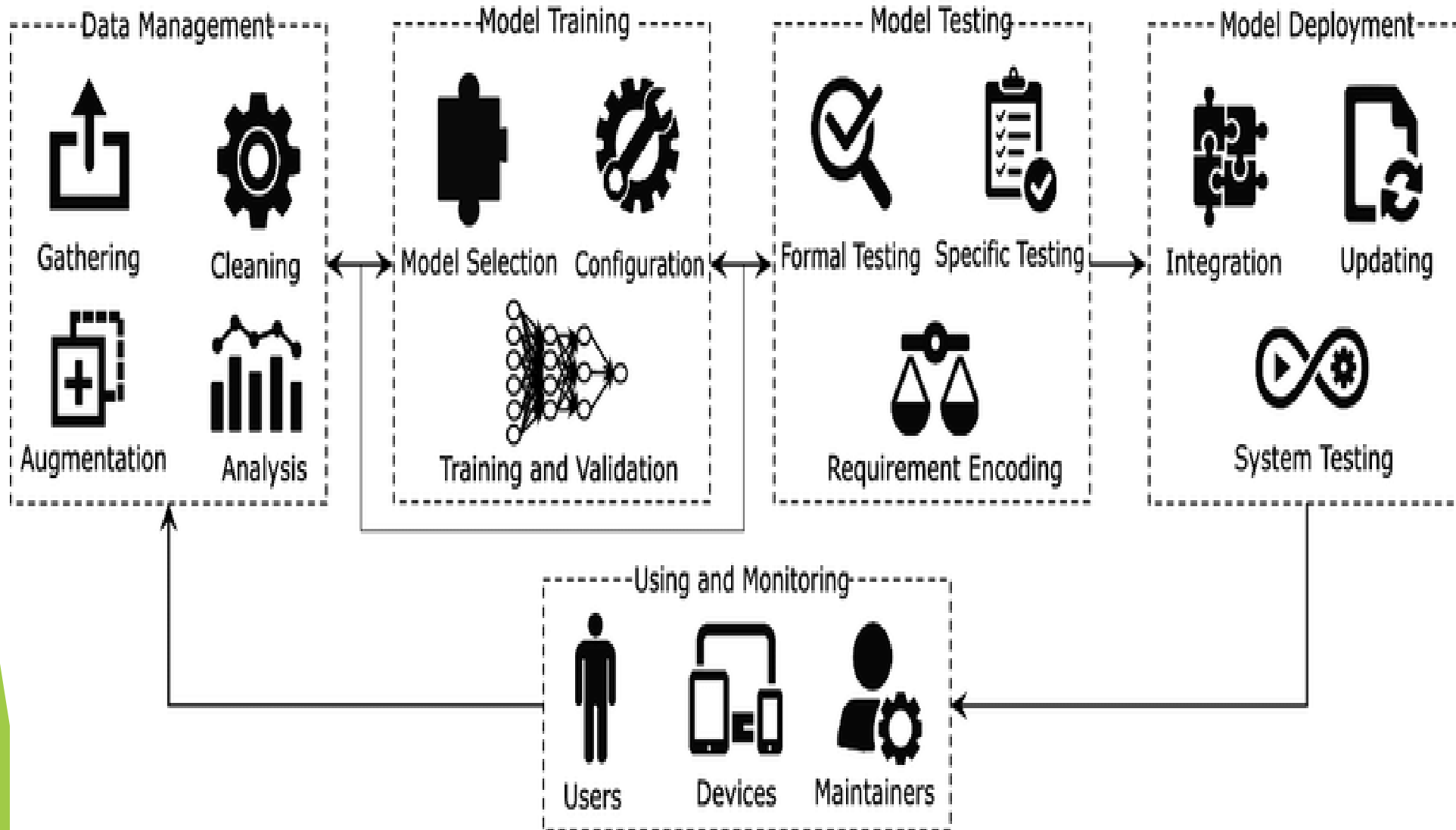
- ▶ Recommendation systems

- ▶ Recommending items

AI, ML, DL

- ▶ AI Components (Mimic the cognitive functions of Human)
 - ▶ Computer Vision
 - ▶ Language processing
 - ▶ Creativity & so on
- ▶ Machine Learning (Statistical part of AI)
 - ▶ Classification
 - ▶ Clustering
- ▶ Revolution in ML (intelligent decisions)
 - ▶ Deep Learning - Neural Network & so on

Life Cycle



ML Platform Capabilities

Build ML Foundations

Operating Model

Multi-Account Foundations

Data Foundations

Central Feature Store

Scale ML Operations

Self-Service Onboarding

ML Development Workflow

CI/CD Build & Deployment

Central Model Registry

Observable ML

Experiment Tracking & Lineage

Centralized Logging & Monitoring

Model Governance

Cost Control & Reporting

Secure ML

Security Foundations

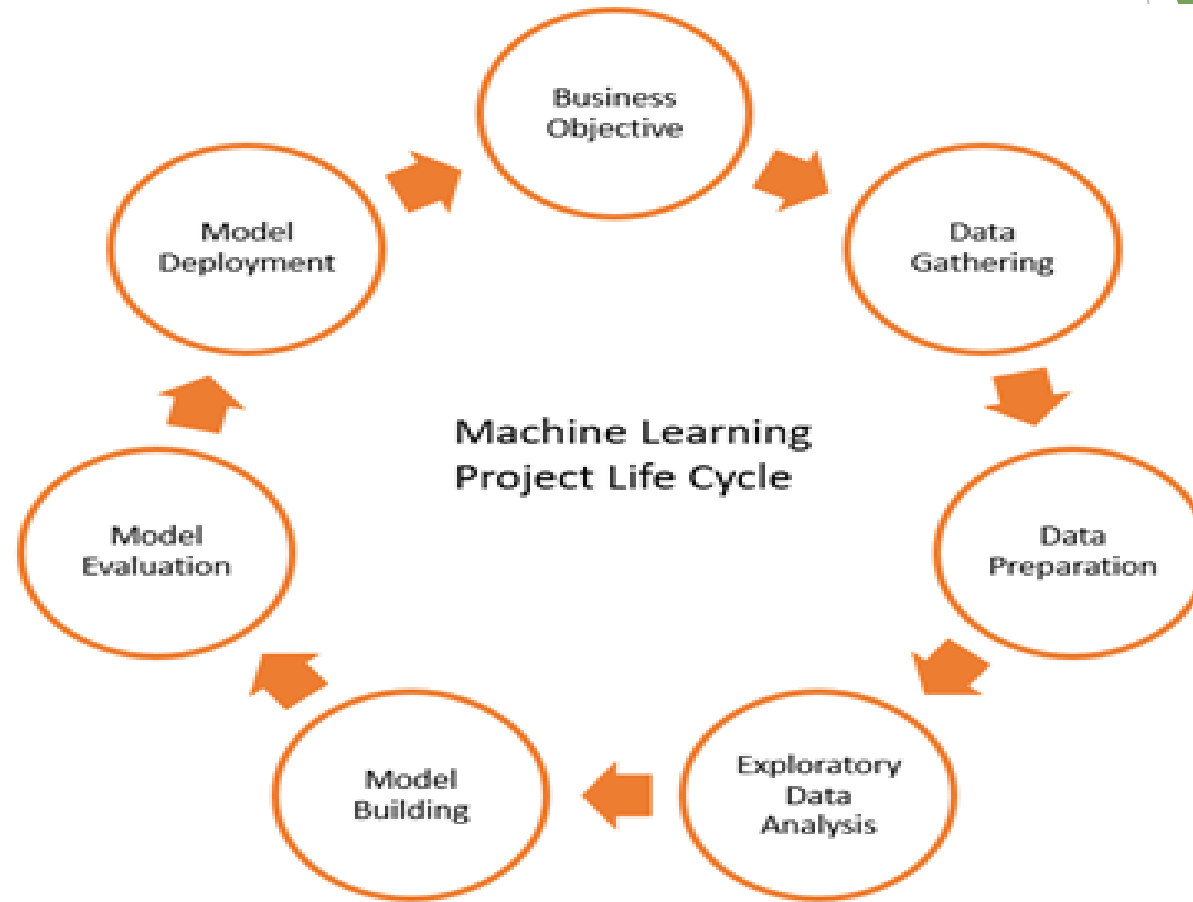
Authentication & Authorization

Data & Model Security

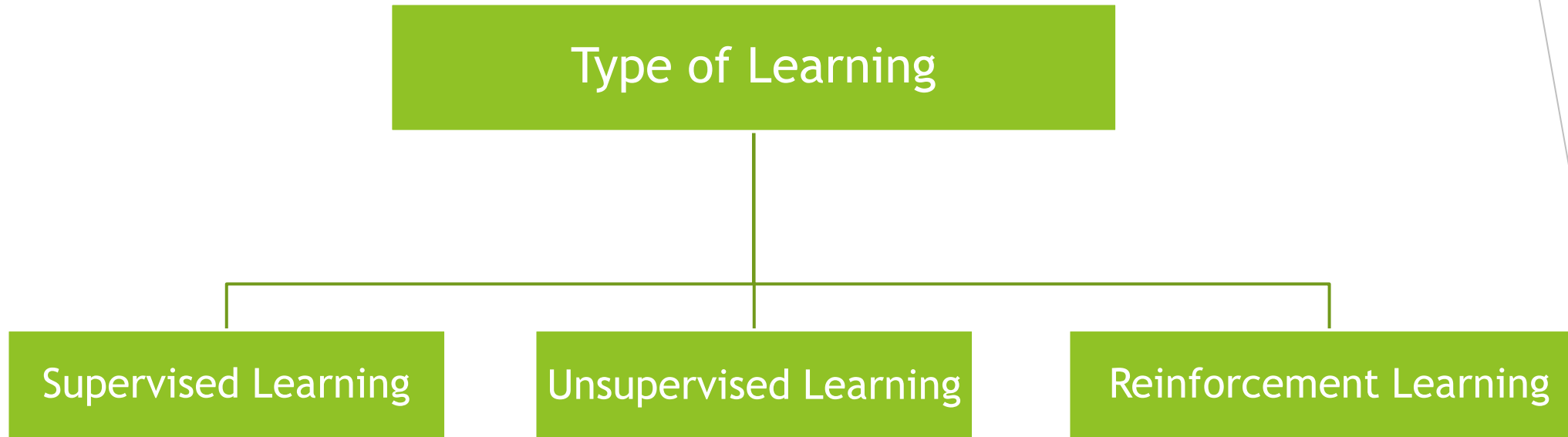
Differential Privacy

Life Cycle

- ▶ Collecting data
- ▶ Data Wrangling
- ▶ Analyse data
- ▶ Train Algorithm
- ▶ Test Algorithm
- ▶ Deployment



Types of Machine Learning



ML Algorithms in brief

- ▶ **Supervised learning** involves training a model on labelled data, where the correct output is already known.
- ▶ **Unsupervised learning** involves training a model on un-labelled data, where the correct output is not known.
- ▶ **Reinforcement learning** involves training a model through trial and error, where the model receives feedback in the form of rewards or punishments.

Regression - Supervised Learning

- ▶ Linear
- ▶ Ridge & Lasso
- ▶ Polynomial
- ▶ Multilinear
- ▶ Non-Linear
- ▶ Ordinary Least Square(OLS)
- ▶ Time Series Forecasting

Classification

- ▶ Logistic
- ▶ KNN
- ▶ Decision Tree
- ▶ Naïve Bayes
- ▶ Random Forest (RF)
- ▶ XGBoost
- ▶ GBM
- ▶ SVM
- ▶ ANN
- ▶ Catboost, adaboost

Clustering

- ▶ Clustering
 - ▶ K-Mean, DBScan, Hierarchical
- ▶ Dimensionality Reduction
 - ▶ PCA, tSNE, ICA, SVD, Auto Encoding
- ▶ Apriori
- ▶ Recommendation
 - ▶ Centred
 - ▶ Memory
 - ▶ User based
 - ▶ Item based

Supervised Learning

- ▶ Whenever we want to predict a certain outcome from a given input, and we have examples of input/output pairs.
- ▶ Observe & Direct
- ▶ Training the Model to predict from labelled data
- ▶ Deals with Historical data
- ▶ Has more evaluation models, controlled env
- ▶ **Regression** - process of predicting continuous values
- ▶ **Classification / Segmentation** - is the process of predicting discrete class labels or categories
 - ▶ Linear Regression
 - ▶ Logistic Regression
 - ▶ DecisionTreeClassifier

Unsupervised Learning

- ▶ Let it model on itself
- ▶ Un labelled data
- ▶ Model works on its own to discover information
- ▶ Has fewer evaluation model than SL
- ▶ Less controlled env
- ▶ Is a type of ML-algorithm used to draw inferences from datasets consisting of input data “WITHOUT LABELLED RESPONSES”
- ▶ **CLUSTERING - Customer Segmentation**
- ▶ Finds patterns & groups them from unlabelled data

Reinforcement Learning

- ▶ Is a type of ML where an agent learns to behave in an environment by performing actions and seeing the results
- ▶ No Training set, RL Algorithm that learns from Trial & Error
- ▶ No Expected Output

Supervised Learning Algorithms

- ▶ Linear Regression
- ▶ Logistic Regression - is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome, outcome is a binary class type (Sigmoid Curve) , polynomial regression
- ▶ Decision Tree - is an classifier
- ▶ Random Forest - is an ensemble classifier made using many Dtree models
- ▶ Naïve Bayes Classifier - is a simple but surprisingly powerful algorithm for predictive modelling i.e naïve and bayes.
Usecases - news categorization, Weather predictions, Spam Filtering

Supervised Learning - challenges

- ▶ Best Fit / Generalization
 - ▶ Over Fit
 - ▶ Under Fit
-
- ▶ we want to build a model on the training data and then be able to make accurate predictions on new, unseen data that has the same characteristics as the training set that we used.
 - ▶ If a model is able to make accurate predictions on unseen data, we say it is able to *generalize* from the training set to the test set.

Regression

- ▶ Regression tasks, the goal is to predict a continuous number, or a *floating-point number* in programming terms (or *real number* in mathematical terms).
- ▶ Predicting a person's annual income from their education, their age, and where they live is an example of a regression task. - **predicted value - Income**
- ▶ Predicting the yield of a farm given attributes such as previous yields, weather, rainfall, resources and number of workers working on the farm.- **predicted value - Yield**
- ▶ Ask whether there is some kind of continuity in the output. If there is continuity between possible outcomes, then the problem is a regression problem.

Linear Regression

- ▶ If there is single variable - prediction can be mean of that column
- ▶ Goodness of fit

Assumptions

- 1) Linear Relationship between input and output - scatter plot
- 2) No multicollinearity - VIF
- 3) Normality of Residual
- 4) Homoscedasticity
- 5) No autocorrelation of error

Linear Regression

► Simple Linear Regression - SLR Equation

- $Y = B_0 + B_1X$ - coefficients are the weights assigned to the features, based on their importance
- Model with one independent variable “x” and “y” dependent variable
- B_0 is an Intercept and B_1 is a Slope
- RSS - residual sum of squares
- TSS - Total sum of squares
- Better R^2 score - Better the Model

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

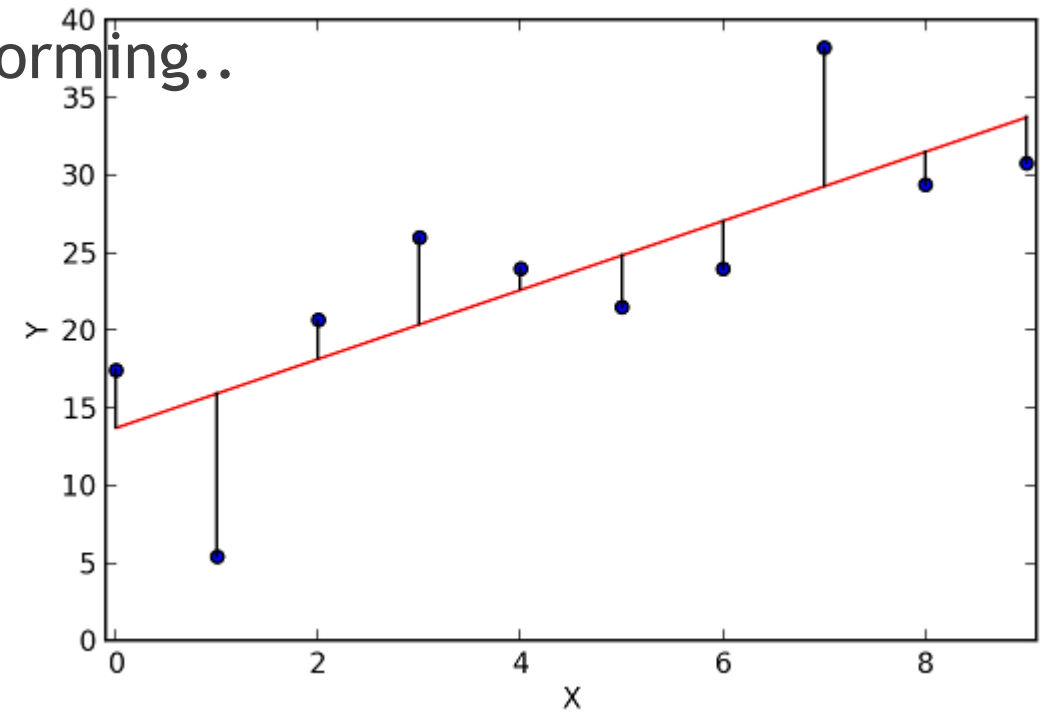
Residuals or Errors

- ▶ Residual sum of squares (RSS) is a statistical method that calculates the variance between two variables that a regression model doesn't explain.
- ▶ It measures the distance between a regression model's predictions and ground truth variables.
- ▶ Higher the RSS, the worse a model is performing..

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

$$TSS = \sum_i^N (y_i - y_{mean})^2$$

$$\begin{aligned} \text{But, } R^2 &= 1 - \frac{RSS}{TSS} \\ \therefore R^2 &= \frac{ESS}{TSS} \end{aligned}$$




Goodness-of-Fit

R^2 statistic: The **proportion** of variance explained

TSS: Total Sum of Squares

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2 \quad \text{This is total variation in } y.$$

How much variation is removed by the regression


$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

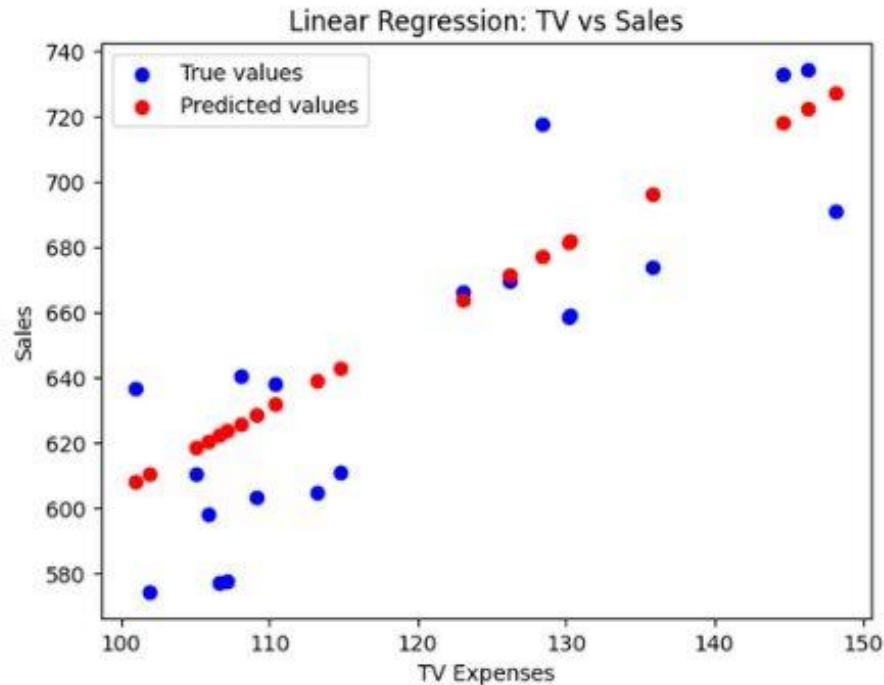
RSS: Total Sum of Residuals

$$RSS = \sum_{i=1}^n \hat{\epsilon}_i^2 \quad \text{This is our estimation of variation in } \epsilon.$$

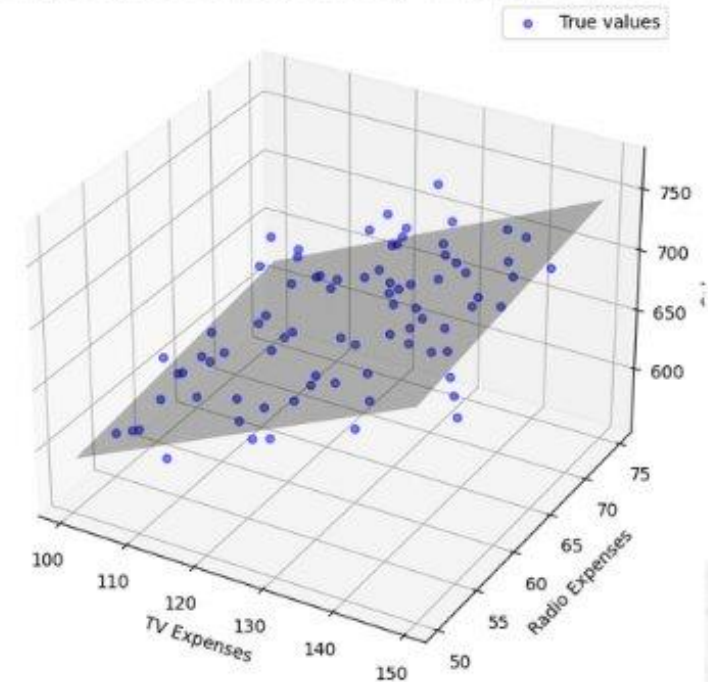
LINEAR REGRESSION



MULTIPLE REGRESSION



Multiple Regression: Sales predicted by TV and Radio Expenses



MLR

- ▶ Multiple Linear Regression - MLR
 - ▶ Model with more than 1 independent variables
 - ▶ $Y = B_0 + B_1X_1 + B_2X_2 + B_3X_3$

Linear Regression

```
from sklearn.linear_model import LinearRegression  
from sklearn.model_selection import train_test_split
```

```
x = df[“column”]  
y = df[“column”]  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)  
x_train.reshape(-1,1)  
y_train.reshape(-1,1)  
lm = LinearRegression()  
lm.fit(x_train, y_train)  
print(lm.intercept_)  
print(lm.coef_)  
ypredict = lm.predict([[x_test]])
```

Linear Regression

```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split

#add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

#fit the regression line using OLS - Ordinary Least Square
lr = sm.OLS(y_train, X_train_sm).fit()
print(lr.params)
print(lr.summary())
```


OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.816			
Model:	OLS	Adj. R-squared:	0.814			
Method:	Least Squares	F-statistic:	611.2			
Date:	Tue, 09 Oct 2018	Prob (F-statistic):	1.52e-52			
Time:	11:38:37	Log-Likelihood:	-321.12			
No. Observations:	140	AIC:	646.2			
Df Residuals:	138	BIC:	652.1			
Df Model:	1					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

const	6.9487	0.385	18.068	0.000	6.188	7.709
TV	0.0545	0.002	24.722	0.000	0.050	0.059
=====						
Omnibus:	0.027	Durbin-Watson:	2.196			
Prob(Omnibus):	0.987	Jarque-Bera (JB):	0.150			
Skew:	-0.006	Prob(JB):	0.928			
Kurtosis:	2.840	Cond. No.	328.			
=====						

- ▶ Prob(F-stat) < 0.05 - overall model fit is SIGNIFICANT
- ▶ R-Squared - between 0 to 1 - higher better the model
- ▶ Coef/p-value = 0.0545 & error is 0.002 is Significant (lesser is best)

Eval Metrics

- ▶ Simplest way of calculating error will be, to calculate the difference in the predicted and actual values.
- ▶ Add them, they might cancel out, so we square these errors before adding.
- ▶ Divide them by the number of data points to calculate a mean error since it should not be dependent on number of data points.

- ▶ `import pandas as pd`
- ▶ `from sklearn.metrics import mean_squared_error`

- ▶ `train_y = train_data["Sales"]`
- ▶ `test_y = test_data["Sales"]`

- ▶ `print("Mean of Target Variable :",train_y.mean())`

Evaluation Metrics

- ▶ Mean Absolute Error (MAE)
- ▶ Mean Squared Error (MSE)
- ▶ Root Mean Squared Error (RMSE)
- ▶ Mean Absolute Percentage Error (MAPE)

Actual Value	Predicted Value	Error(Actual-Predicted)	Absolute Error	Squared Error
10	14	-4	4	16
21	19	2	2	4
34	43	-9	9	81
108	97	11	11	121
10000	10400	-400	500	2500
	Sum values:	-400	526	2722

Divide by total Number of Data Points

Actual Output

Predicted Output

$$MAE = \frac{1}{N} \sum |Y - \hat{Y}|$$

Sum Of

Absolute Value of residual

$$R^2 \text{ Squared} = 1 - \frac{SSr}{SSm}$$

SSr = Squared sum error of regression line

SSm = Squared sum error of mean line

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

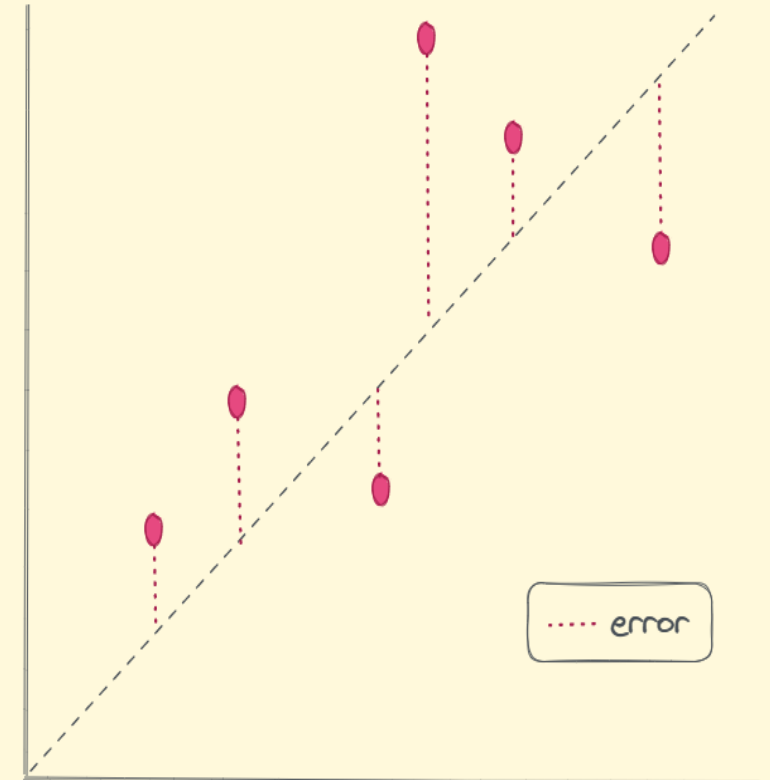
$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

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

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

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

Predicted Value



Actual Value

Multi Linear Regression

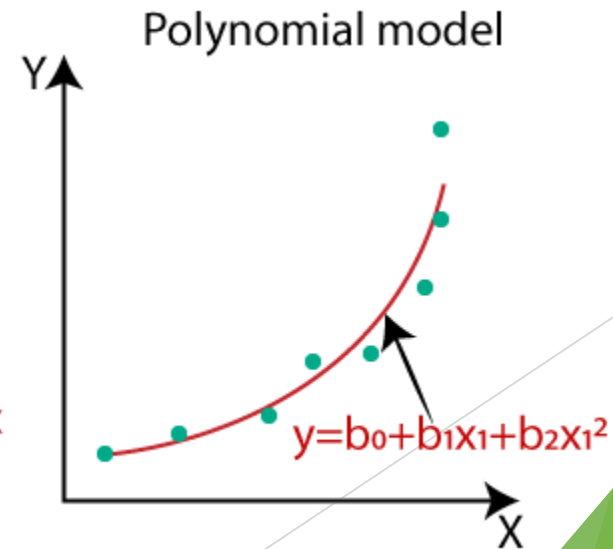
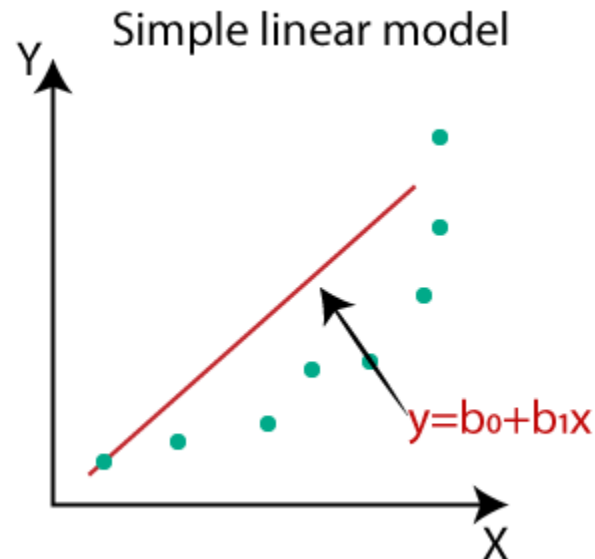
- ▶ Variance Inflation Factor (VIF)
- ▶ $VIF > 10$ - High VIF & variable should be eliminated
- ▶ $VIF > 5$ - can be okay, but it is worth inspecting
- ▶ $VIF < 5$ - GOOD VIF , no need to eliminate the variable
- ▶ Feature Scaling - Standardization & MinMax Scaling

Polynomial Regression

- ▶ Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as n th degree polynomial.
- ▶ A nonlinear connection between the value of x and the associated dependent mean of y , denoted $E(y | x)$, can be estimated via poly model.

Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial.
- $y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$



SRC Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as mtp
import seaborn as sns

a=["BA","JC","SC","Mgr","CM","RM","P","SP","CFO","CEO"]
b=[1,2,3,4,5,6,7,8,9,10]
c=[45000,50000,60000,80000,110000,150000,200000,300000,500000,1000000]
df = pd.DataFrame({"position":a,"level":b,"salary":c})
print(df)
x = df.iloc[:,1:2].values
y = df.iloc[:,2].values

#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

SRC Code

```
in_regs= LinearRegression()
```

```
lin_regs.fit(x,y)
```

```
poly_regs= PolynomialFeatures(degree= 4)
```

```
x_poly= poly_regs.fit_transform(x)
```

```
lin_reg_2 =LinearRegression()
```

```
lin_reg_2.fit(x_poly, y)
```

```
mtp.scatter(x,y,color="blue")
```

```
mtp.plot(x,lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
```

```
mtp.title("Bluff detection model(Linear Regression)")
```

```
mtp.xlabel("Position Levels")
```

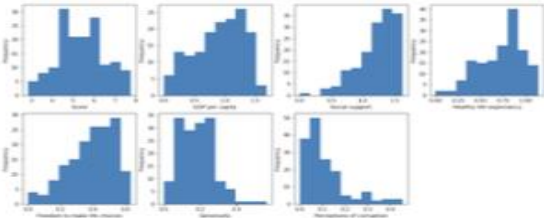
```
mtp.ylabel("Salary")
```

```
mtp.show()
```

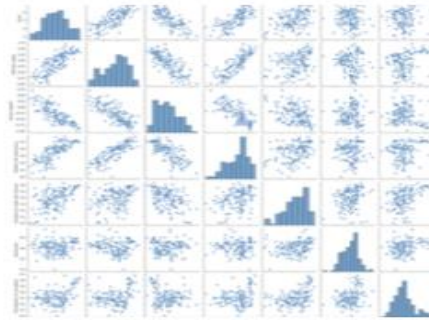
Machine Learning Algorithms - Regression

Exploratory Data Analysis (EDA)

Histogram: `df.plot(kind = 'hist')`

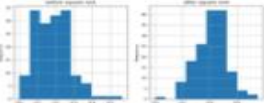


Pairplot: `sns.pairplot()`

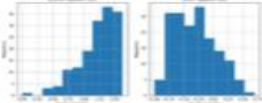


Feature Engineering

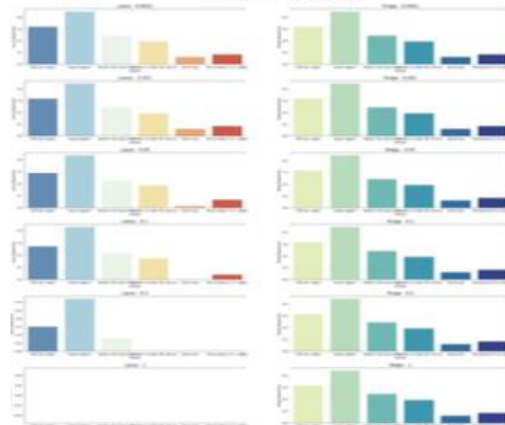
Log Transform
`np.log()`



Square Root Transform
`np.sqrt()`

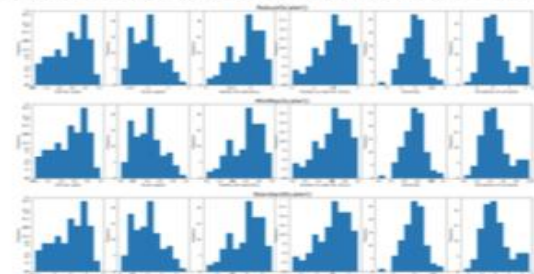


Feature Importance
`coef_.ravel()`

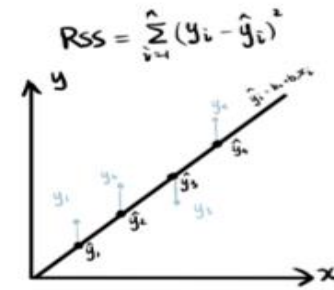


Feature Scaling

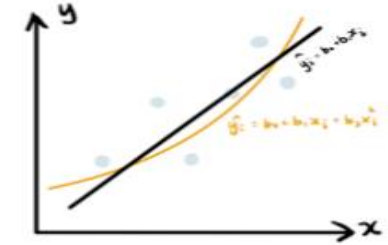
`StandardScaler()`, `RobustScaler()`, `MinMaxScaler()`



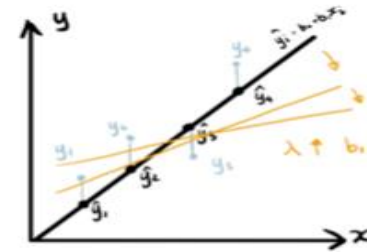
Linear Regression



Polynomial Regression



Regression with Regularization Techniques



Lasso Regression

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda |b|$$

"L1 regularization term"

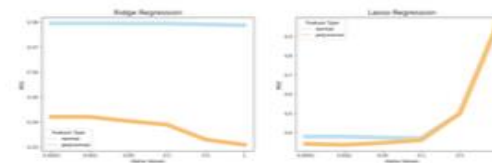
Ridge Regression

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

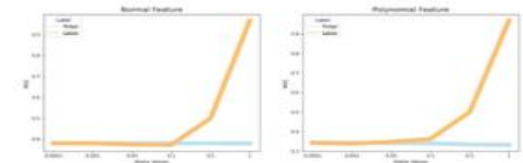
"L2 regularization term"

Model Evaluation

Ridge vs Lasso



Normal vs. Polynomial



Why Regularization ?

- ▶ Some coefficients of LR is much higher as compared to rest of the coefficients.
- ▶ Therefore the PREDICTED VARIABLE would be more driven by these HIGHER features
- ▶ How can we reduce the magnitude of coefficients in our model?
- ▶ For this purpose, we have different types of regression techniques which uses regularization to overcome this problem.

What is Ridge Regression?

- ▶ Model Tuning Method
- ▶ Used to analyse any data that suffers from multicollinearity
- ▶ Multicollinearity occurs, least square are unbiased and variance are large
- ▶ This results in predicated values being far away from actual values
- ▶ REGULARIZATION used to calibrate ML model to minimize adjusted loss function and avoid overfitting and underfitting
- ▶ Three types of Regularization Techniques
 - ▶ RIDGE REGRESSION (L2 Regularization)
 - ▶ LASSO REGRESSION (L1 Regularization)
 - ▶ ELASTIC NET (Combo of Ridge and Lasso)

Source code

- ▶ `from sklearn.linear_model import Ridge`
- ▶ `ridge = Ridge(alpha=0.5).fit(X_train, y_train)`
- ▶ `print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))`
- ▶ `print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))`
- ▶ Alpha is a HYPERPARAMETER set manually by the user
- ▶ If we increase the alpha value magnitude of the coefficient decreases

$$\min \left(\|Y - X(\theta)\|_2^2 + \lambda \|\theta\|_2^2 \right)$$

Lasso Regression

- ▶ LASSO (Least Absolute Shrinkage Selector Operator), is quite similar to ridge.
- ▶ Automatic feature selection – some of the coefficients are set to zero

Source code

- ▶ `from sklearn.linear_model import Lasso`
- ▶ `lasso = Lasso().fit(X_train, y_train)`
- ▶ `print("Training set score: {:.2f}".format(lasso.score(X_train, y_train)))`
- ▶ `print("Test set score: {:.2f}".format(lasso.score(X_test, y_test)))`
- ▶ `print("Number of features used: {}".format(np.sum(lasso.coef_ != 0)))`

Classification Algorithms

- ▶ Logistic Regression (LR)
- ▶ Support Vector Machine (SVM)
- ▶ KNN (K-Nearest Neighbour)
- ▶ Decision Tree (DT)
- ▶ Random Forest (RF)

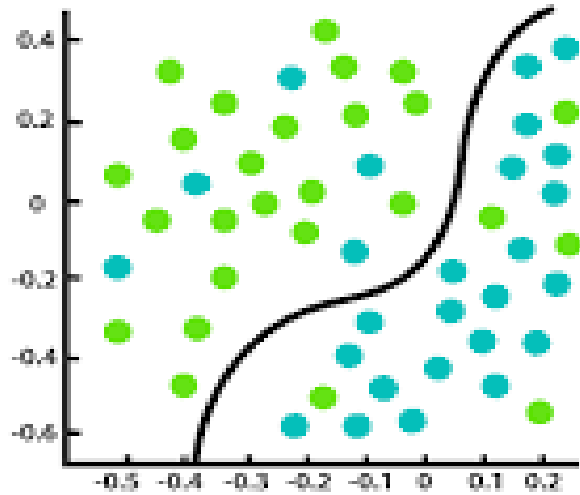
Logistic Regression

- ▶ Is a supervised classification model
- ▶ Binary Classification
- ▶ Target is categorical variable

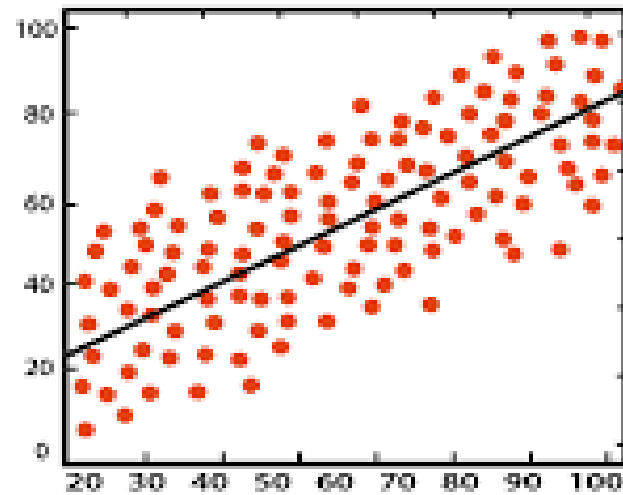
- ▶ Categorical Variables
- ▶ Solves classification problems
- ▶ S-Cure

LR-classifier

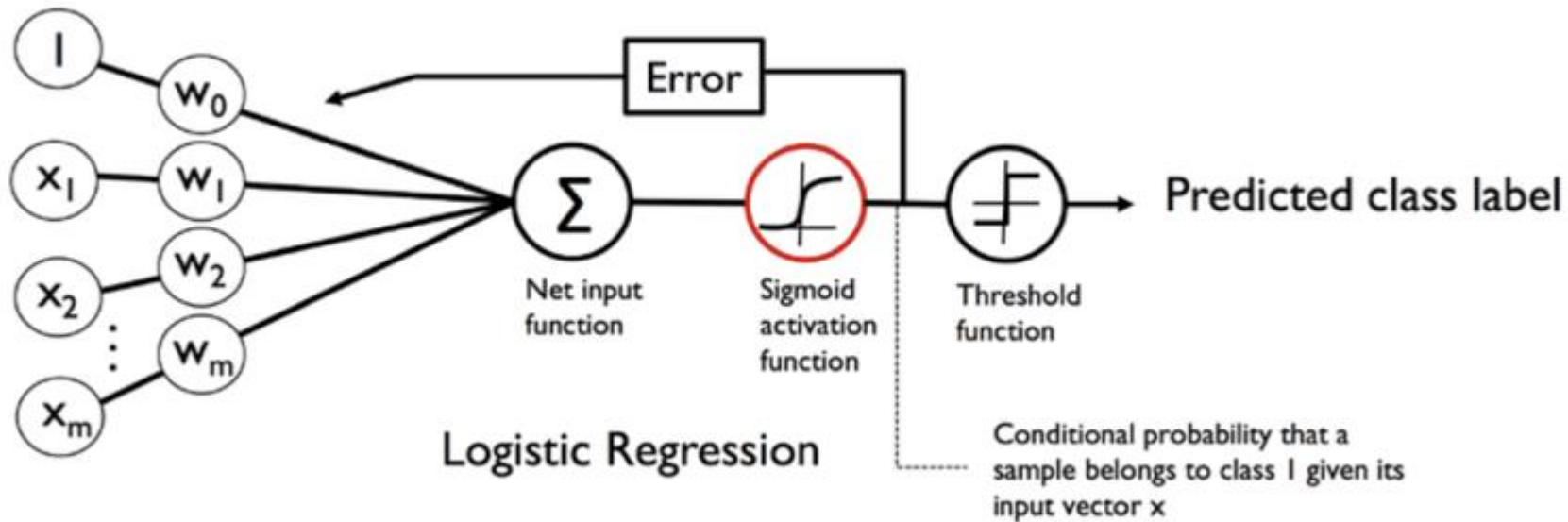
- ▶ Bank to predict loan defaulter
 - ▶ Factory to predict load of a machine - down/up
 - ▶ Email - Spam or Ham detector
 - ▶ Diseases classifier
-
- ▶ Use Cases
 - ▶ Weather predictions, Determine Illness problem
 - ▶ Purchase propensity Vs advertisement spend analysis
 - ▶ Customer churn predictions, Employee attrition modelling
 - ▶ Hazardous event prediction,

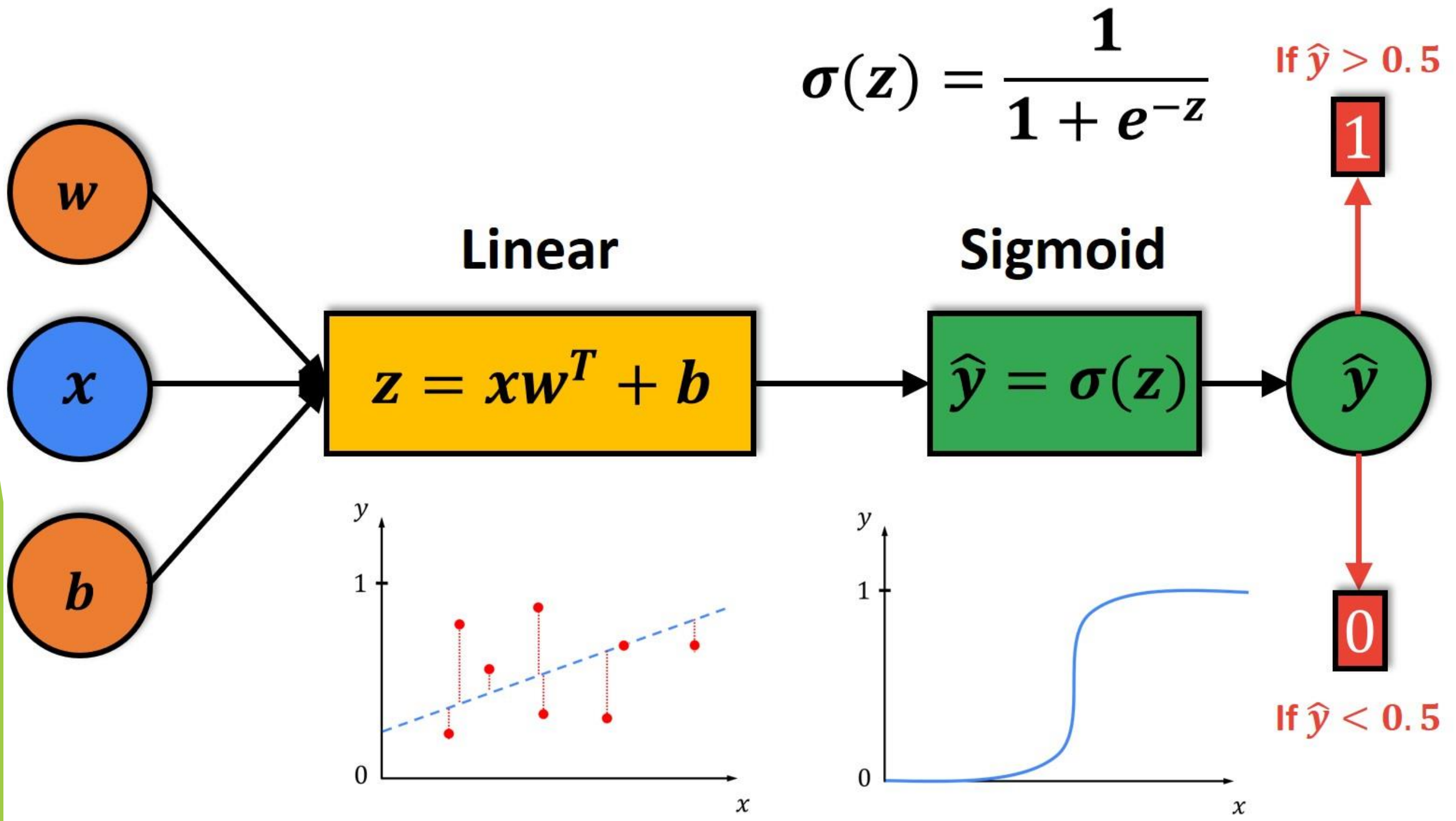


Classification



Regression





Math behind Logistic Regression

X	-9	-8	0	8	9
Y	0	0	1	1	1
Assume	$z = 5x + 10$		$y = 1 / 1 + e^{-z}$		
$x = -9$	$x = -8$	$x = 0$	$x = 8$	$x = 9$	
$z = 5(-9) + 10$	$z = 5(-8) + 10$	$z = 5(0) + 10$	$z = 5(8) + 10$	$z = 5(9) + 10$	
-35	-30	10	50	55	
$y = 1 / 1 + e^{35}$	$y = 1 / 1 + e^{30}$	$y = 1 / 1 + e^{10}$	$y = 1 / 1 + e^{50}$	$y = 1 / 1 + e^{55}$	
y=0	y=0	y=1	y=1	y=1	

Pros & Cons

- ▶ Easy to implement
- ▶ Performs well on data with linear relationship
- ▶ Less prone to overfitting for low dimensional dataset
- ▶ High dimensional datasets causes OVERFITTING
- ▶ Difficult to computer complex relationship in dataset
- ▶ Sensitive to OUTLIERS
- ▶ Needs LARGE DATASET
- ▶ CLASS IM-BALANCING

Confusion Matrix

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Predicted Class

Actual Class

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

The you saw that the different elements in the confusion matrix can be labelled as follows –

Actual/Predicted	Not Churn	Churn
Not Churn	True Negatives	False Positives
Churn	False Negatives	True Positives

Hence, you rewrote the sensitivity and specificity formulas as –

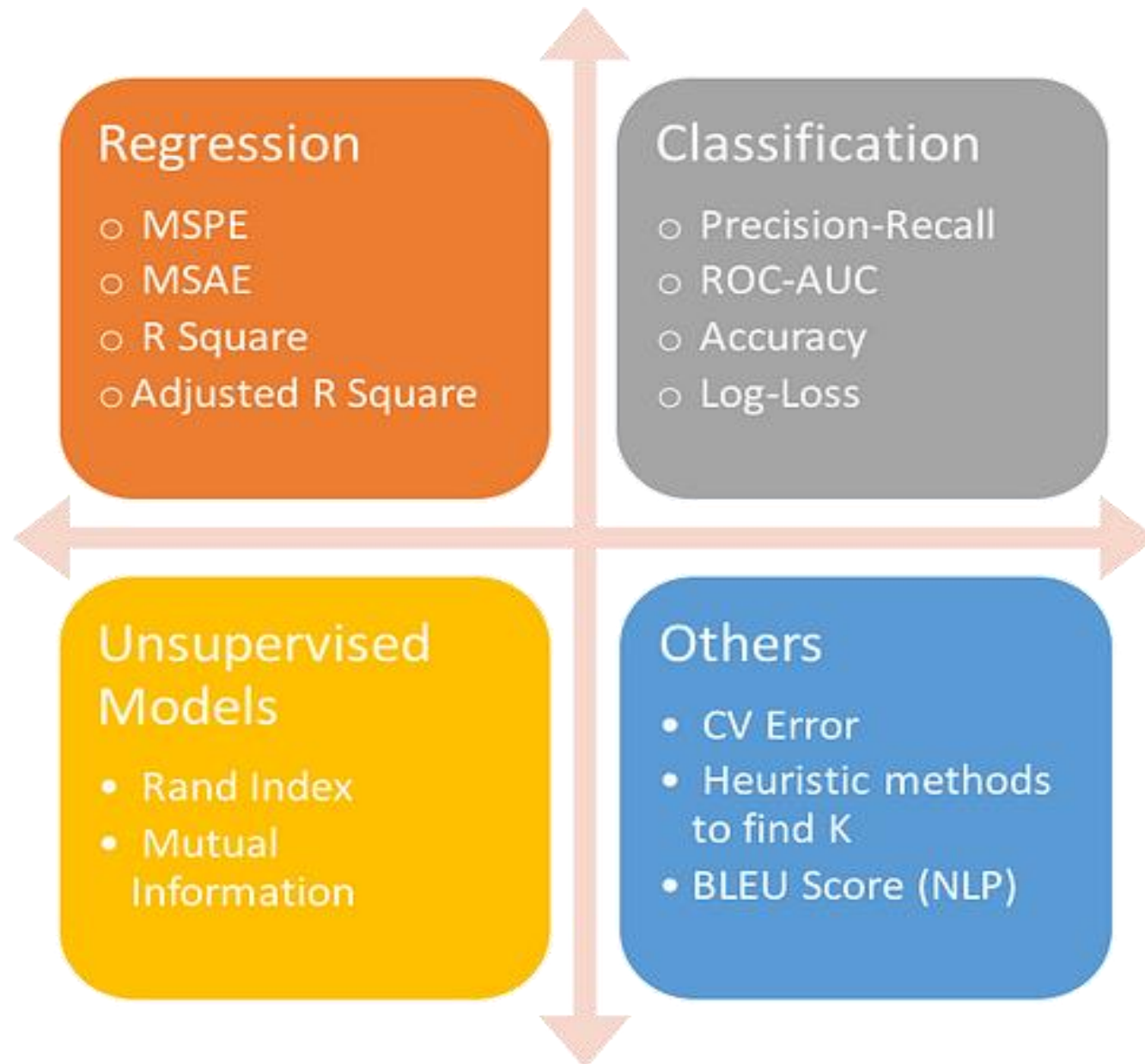
$$Sensitivity = \frac{TP}{TP+FN}$$

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

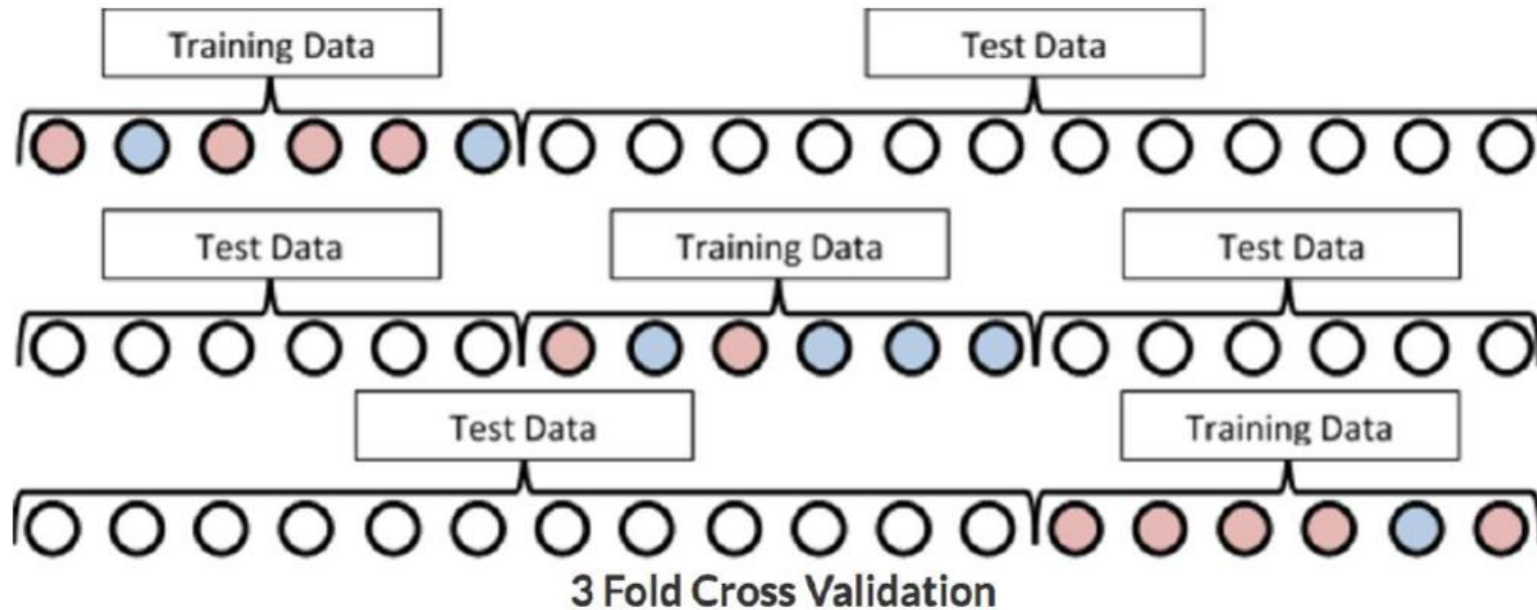
LR - Eval Metrics

- ▶ Accuracy = Correctly Predicted Labels / Total no of labels
- ▶ Sensitivity = No of actual Yeses correctly predicted / total no of actual yeses
- ▶ Specificity = No of actual Nos correctly predicted / total no of actual nos
- ▶ Precision = $TP / TP + FP$
- ▶ Recall(True Positive Rate) = $TP / TP + FN$

		PREDICTED classification				
		Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN	
	b	FN	TP	FN	FN	
	c	TN	FP	TN	TN	
	d	TN	FP	TN	TN	



K-Fold Cross Validation



Basically, there are 3 iterations in which evaluation is done. In the first iteration, $\frac{1}{3}$ rd of the data is selected as training data and the remaining $\frac{2}{3}$ rd of it is selected as testing data. In the next iteration, a different $\frac{1}{3}$ rd of the data is selected as the training data set and then the model is built and evaluated. Similarly, the third iteration is completed.

Such an approach is necessary if the data you have for model building is very small, i.e., has very few data points.

Support Vector Machine

- ▶ SVMs belongs to the class of Linear ML models
- ▶ SVM offers high accuracy compared to other classifiers - logistic regression and decision trees
- ▶ SVM can handle multiple continuous and categorical variables
- ▶ Can be used for Image recognition, face detection, voice detection and so
- ▶ Hyper parameter will be choosing HYPERPLANE
- ▶ Can handle outliers & work with linear & non-linear data (Kernels)

SVM

- ▶ Support Vectors are the data points which are nearest to HYPERPLANE
- ▶ If these datapoints changes position of hyperplane also changes

Maximal Margin Classifier

There could be multiple lines(Hyperplanes) possible which perfectly separate the two classes as shown in the figure below. But the best line, is the one which maintains the largest possible equal distance from the nearest points of both the classes so for the separator to be optimal, the margin or the distance of the nearest point to the separator should be maximum. This is called **Maximal Margin classifier**.

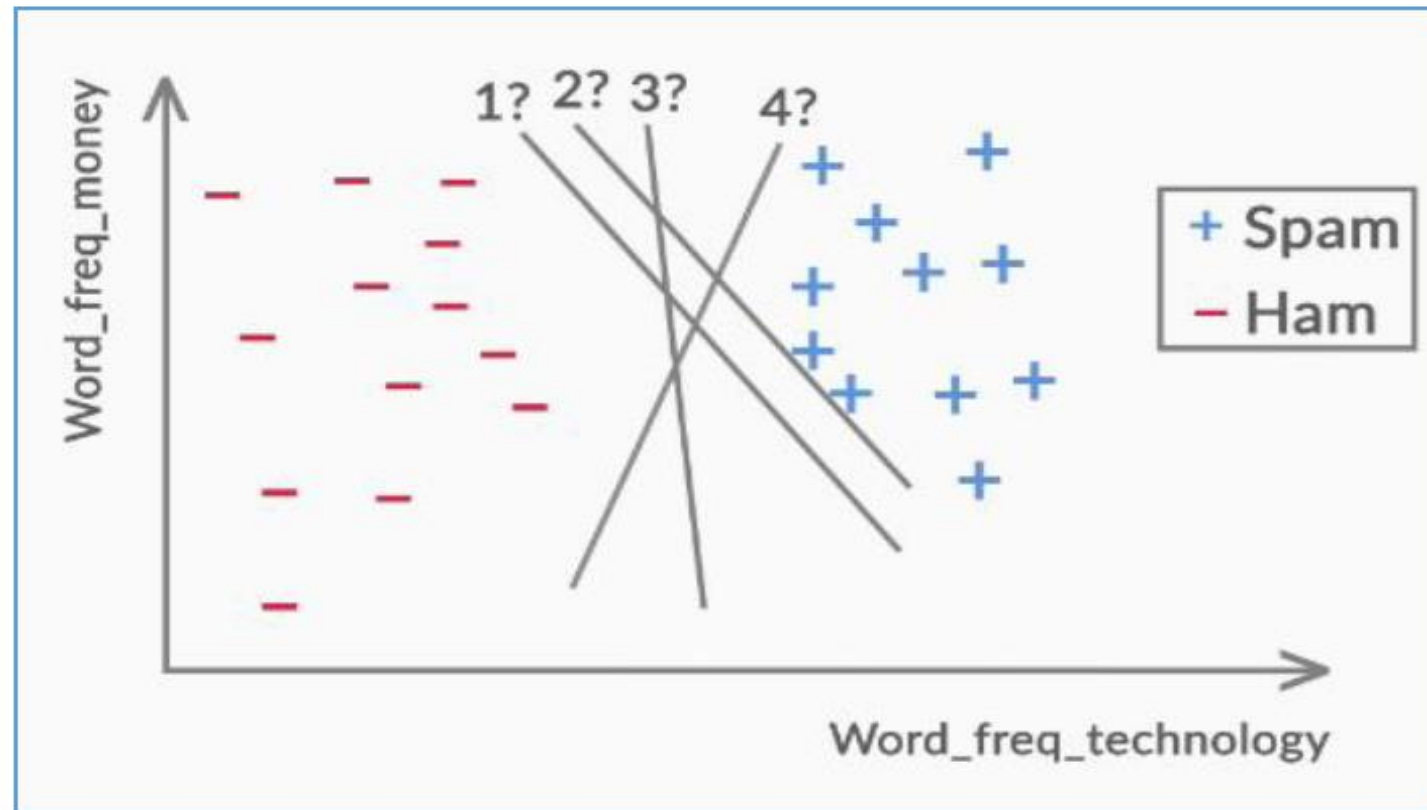
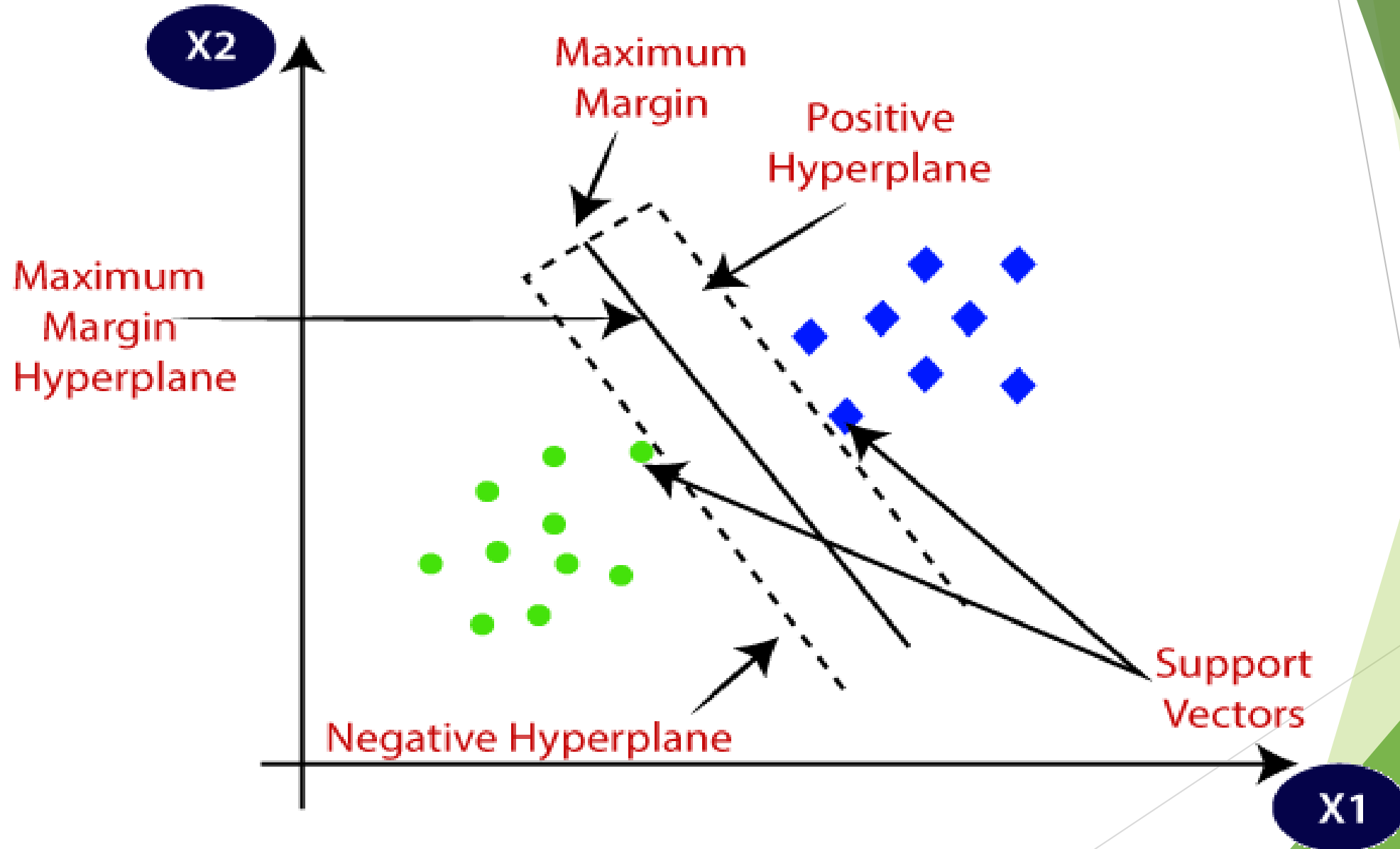
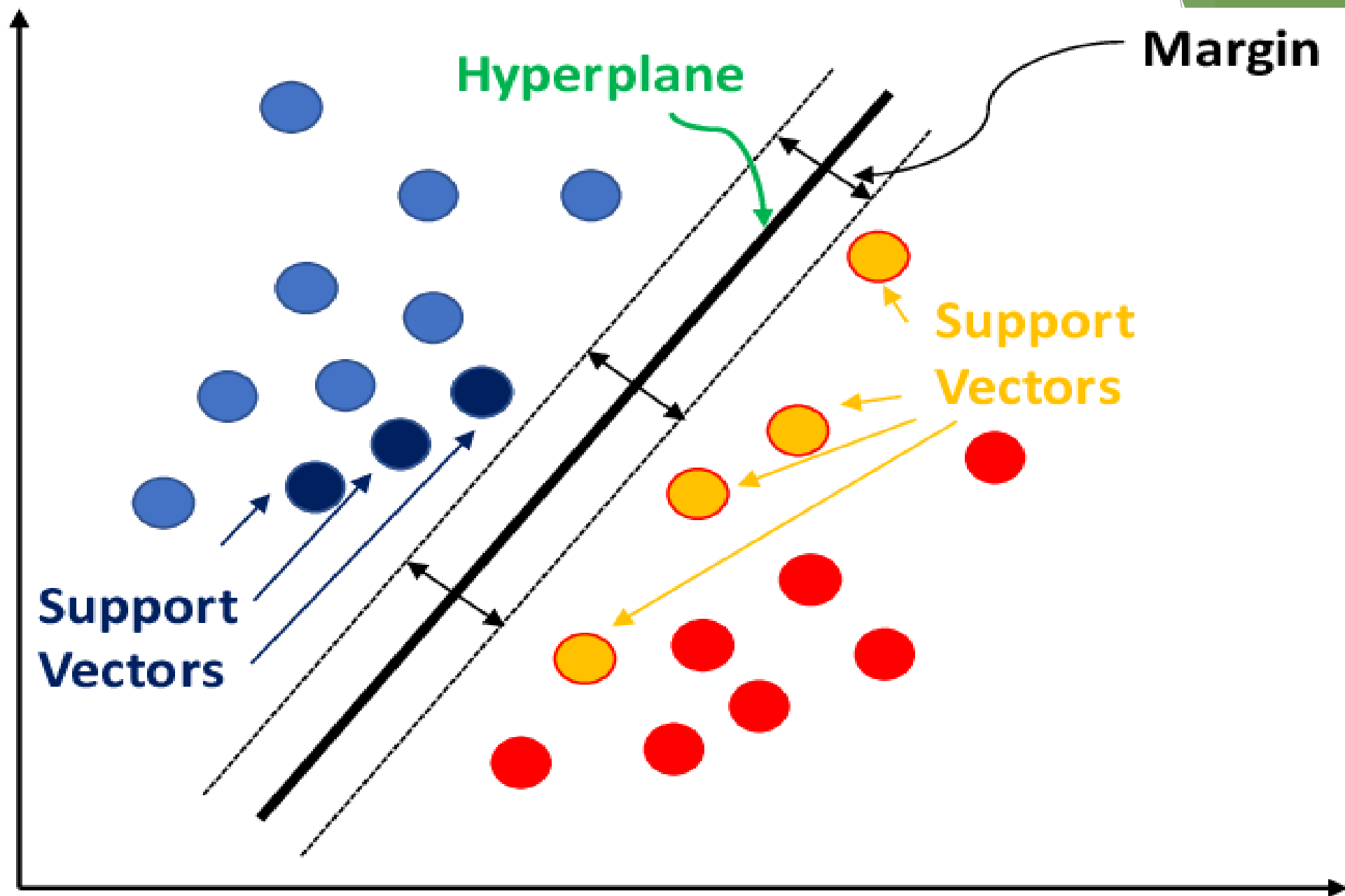
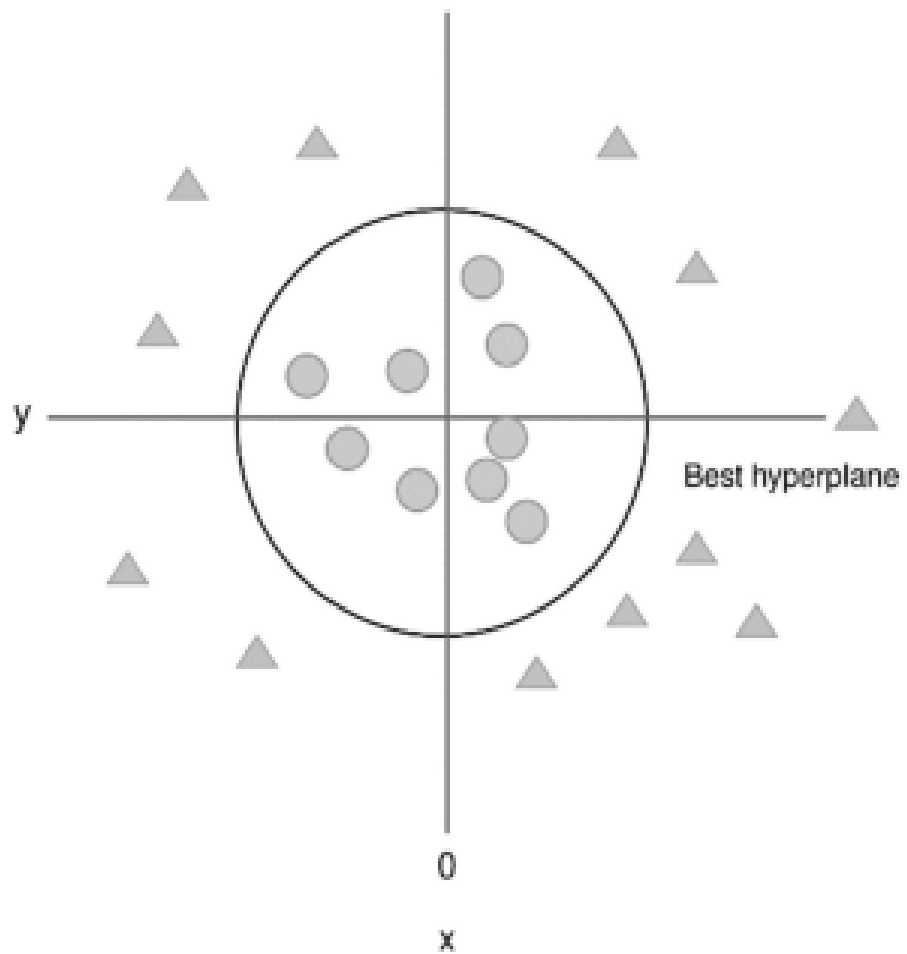


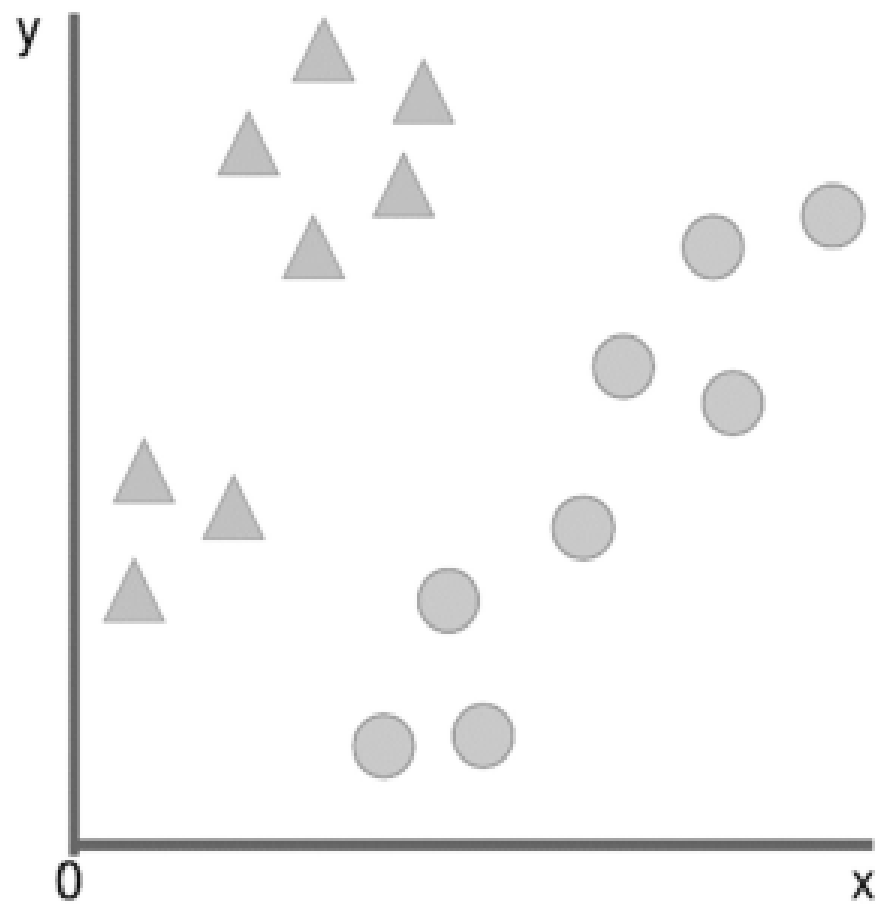
Figure 5: Maximal Margin Classifier







a) Nonlinear



b) Linear

Kernel	Formula
Linear	$K(x, y) = x \cdot y$
Polynomial	$K(x, y) = (ax \cdot y + b)^d$
RBF	$K(x, y) = \exp(-\ x - y\ ^2 / \sigma^2)$
KMOD	$K(x, y) = a \left(\exp\left(\frac{\gamma^2}{\ x - y\ ^2 + \sigma^2}\right) - 1 \right)$

Equation of a Hyperplane

- ▶ $Y = mx + c$
- ▶ $ax + by + c = 0$ **# General FORM of Line Equation**
- ▶ $ax_1 + bx_2 + c = 0$ **# rename “X-axis” as X_1 & “Y-axis” as X_2**
- ▶ $w_1x_1 + w_2x_2 + w_0 = 0$ **# General Equation**
- ▶ $w_1x_1 + w_2x_2 + w_3x_3 + w_0 = 0$ **# 3D Plane - X_1, X_2, X_3**
- ▶ $w_1x_1 + w_2x_2 + w_3x_3 + \dots\dots\dots w_nx_n + w_0 = 0$ **# “n”D Plane**
- ▶ **$w \cdot x + w_0 = 0$ # dot product form of the above equation**

Equation of the Hyperplane

$w = [w_1, w_2, w_3 \dots w_n]$ # in vector format

$x = [x_1, x_2, x_3 \dots x_n]$ # in vector format

$[w_1, w_2, w_3 \dots w_n]$ # in a matrix format - row vector

$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix}$ # in a matrix format - column vector

Can simplified as $w^T x + w_0 = 0$ (T is transpose)

$w^T x = 0$ (if the line passes thru the origin w_0 will be ZERO)

Equation of a Hyperplane

- ▶ $w^T x = w \cdot x = ||w|| ||x|| \cos \theta = 0$
- ▶ # $||w||$ - absolute value
- ▶ “w” will perpendicular LINE to the HYPERPLANE

Maximal Margin classifier

- ▶ There could be multiple lines(Hyperplanes) possible which perfectly separate the two classes.
- ▶ Best line, is the one which maintains the largest possible equal distance from the nearest points of both the classes so for the separator to be optimal, the margin or the distance of the nearest point to the separator should be maximum.

Math behind SVM

Let slope and intercept of hyperplane is

$m = -1$ and $c = 0$ (assume it passes thru the origin)

Let parameters of hyperplane in “w” which is nothing but weight

$w \rightarrow (m, c) = (-1, 0)$

Lets multiply “x” or P1 by transpose of “w” - $p1 = (-3, 0)$

$$w^T x = \begin{bmatrix} -1 & 0 \end{bmatrix} \begin{bmatrix} -3 \\ 0 \end{bmatrix} = 3 \quad (\text{POSITIVE})$$

For P2 (3,3)

$$w^T x = \begin{bmatrix} -1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = -3 \quad (\text{NEGATIVE})$$

Support Vector Machine - SVM

- ▶ Hyper Plane - to classify
 - ▶ $ax + by + c > 0$ - class-A
 - ▶ $ax + by + c < 0$ - class-B
 - ▶ $ax + by + c = 0$ - on the Hyper Plane
- ▶ Mis-classification is controlled by slack variable
 - ▶ Each data point has a slack variable
 - ▶ Value of slack lies b/w 0 to infinity
 - ▶ Slack = 0 - correct classification
 - ▶ Slack > 1 - in-correct
 - ▶ Slack $> 0 < 1$ - classifies correctly but violates the margin

SVM

► On 3D- Hyper plane

- $ax + by + cz + d > 0$ - class-A
- $ax + by + cz + d < 0$ - class-B
- $ax + by + cz + d = 0$ - on the Hyper Plane

► Kernels - Non-Linear Data

- Linear Kernel
- Polynomial Kernel
- Radial Basis Function - (RBF) Kernel
- Sigmoid

SVM Pros & Cons

PROS

- ▶ Works with smaller data set
- ▶ Works efficiently where there is clear margin of separation
- ▶ Works well with high dimensional data

CONS

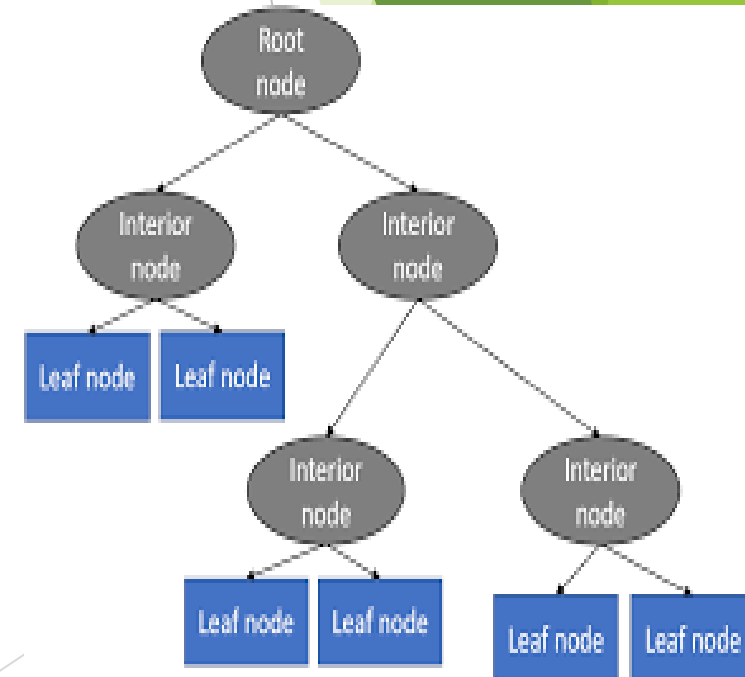
- ▶ Not suitable for large dataset as training time would be more
- ▶ Not suitable for noisier (outlier) datasets with overlapping classes

Self Learning

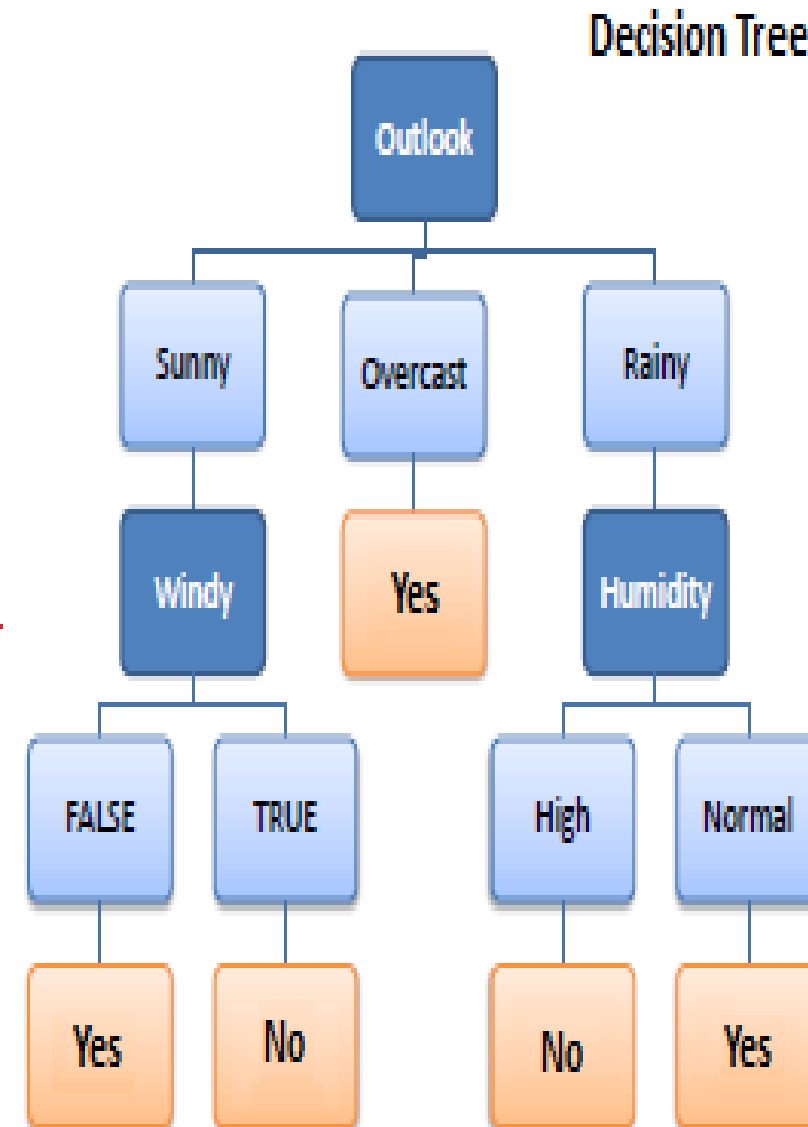
- ▶ What is SOFT HYPER PLANE ?
- ▶ Slack variable ?
- ▶ Log Likelihood ?
- ▶ ROC & AUC ?
- ▶ XGBOOST ?

Decision Tree

- ▶ Supervised ML Algorithm - used for regression and classifier
- ▶ CART - Classification and Regression Tree
- ▶ Hierarchical tree structure with root node, branches, internal nodes and leaf nodes.
- ▶ Terms
 - ▶ Root node, Decision node, Leaf, Sub-Tree, Pruning
- ▶ Binary Decision Tree & Multiway Decision Tree
 - ▶ Decision Tree - Binary
 - ▶ Nodes - Testing
 - ▶ Leaf - Decision / Model



Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

$$\begin{aligned}
 \text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\
 &= \text{Entropy}(0.36, 0.64) \\
 &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\
 &= 0.94
 \end{aligned}$$

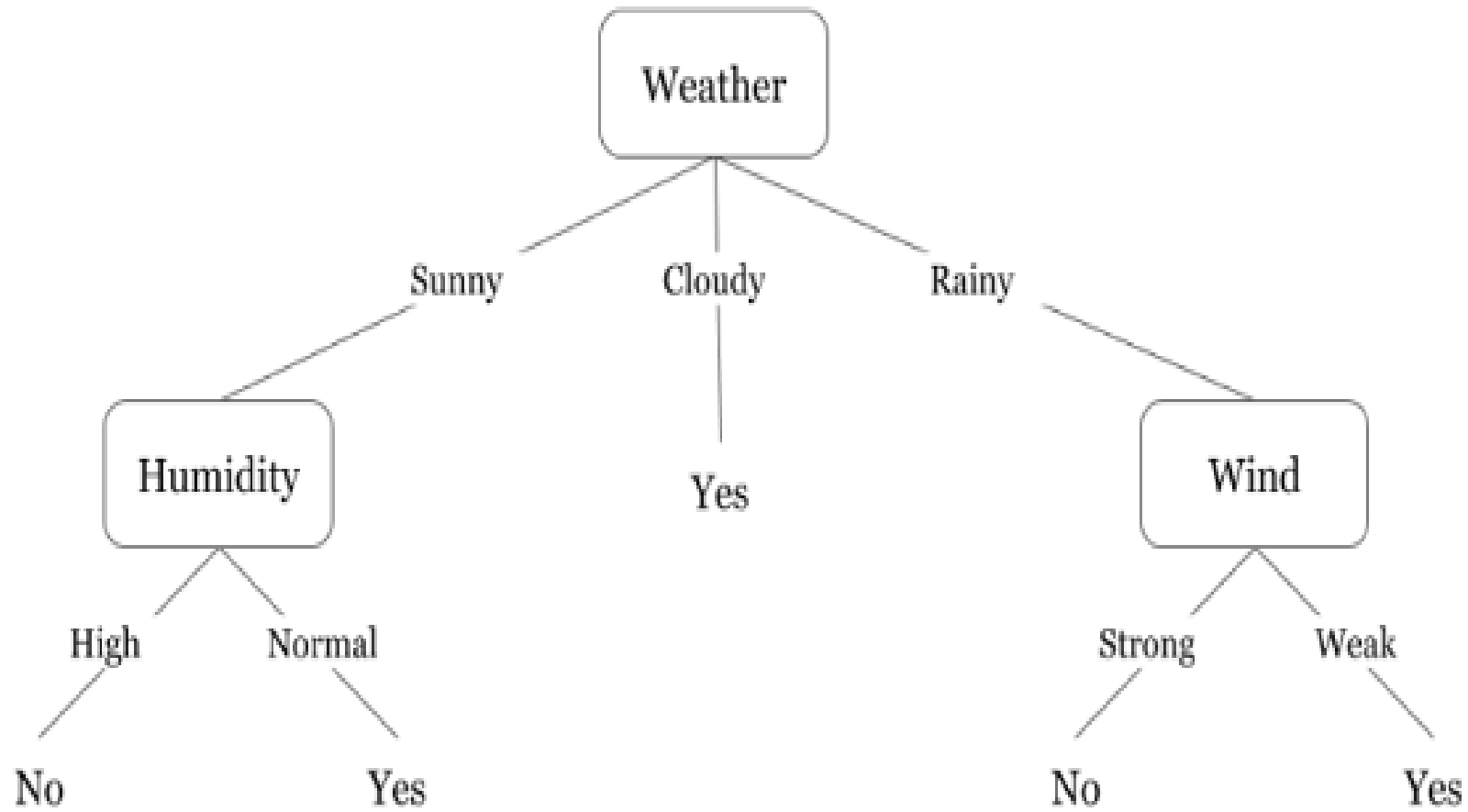
$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

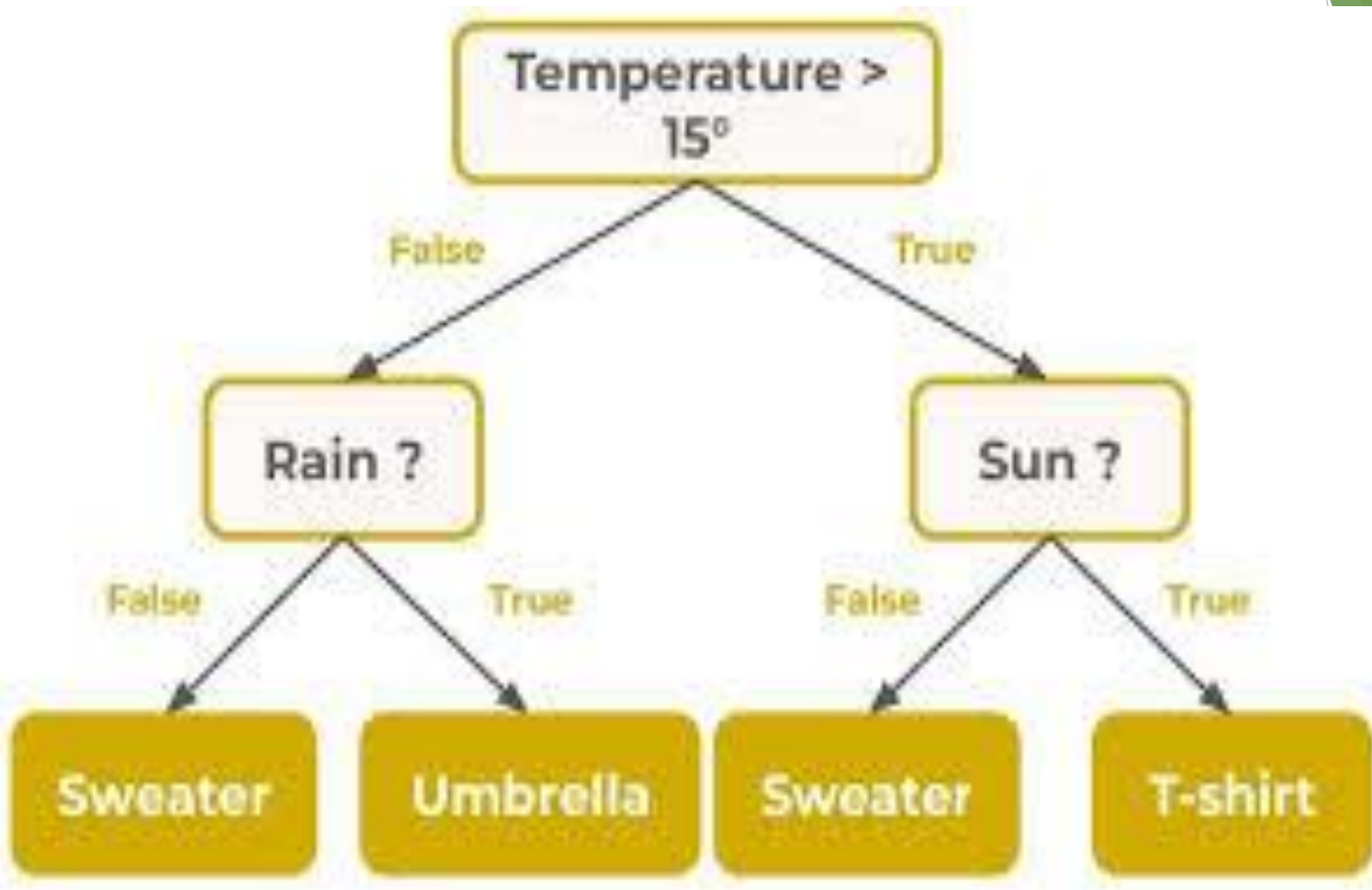
		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned}
 E(\text{PlayGolf, Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\
 &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\
 &= 0.693
 \end{aligned}$$

Decision Tree





Decision Tree

► Assumptions

- Binary Splits - two splits
- Recursive Partitioning - until stop criteria is met
- Feature Independence - not correlated
- Homogeneity - as similar as possible
- Top-Down Greedy Approach -
- Categorical and Number Features
- Overfitting
- No Missing values and no Outliers

DT

- ▶ **Step-1:** Begin the tree with the root node, says S , which contains the complete dataset.
- ▶ **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- ▶ **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- ▶ **Step-4:** Generate the decision tree node, which contains the best attribute.
- ▶ **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

DT - Attribute Selection Measure

- ▶ While Designing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes.
- ▶ There is a technique which is called as **Attribute selection measure**
- ▶ By this measurement, we can easily select the best attribute for the nodes of the tree.
- ▶ There are two popular techniques for ASM, which are:
 - **Information Gain**
 - **Gini Index**

DT

► Entropy

- Amount of uncertainty in the data set
- Measure of dis-order
- Quantative measure of randomness
- With each split, the data becomes more homogenous which will decrease the entropy
- Low entropy - less randomness & vice-versa

$$E(S) = -p_{(+)} \log p_{(+)} - p_{(-)} \log p_{(-)}$$

► Information Gain

- Difference between entropy before split
- Measures the reduction of uncertainty
- Deciding factor to select the root node/decision node

$$\text{IG} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$$

► Gini Index

- Measure of purity or impurity at node level
- Measure of homogeneity

DT-Example

- ▶ Example - Employees of an organization
- ▶ We want to build a model for who among them plays football
- ▶ Each employee has two explanatory attributes — Gender and Age.
- ▶ The target attribute is whether they play football.
- ▶ The numbers against P and N indicate the numbers of employees who play football and those who don't respectively, for each combination of gender and age.

		AGE	
		< 50	> 50
GENDER	F	P - 10 N - 390	P - 0 N - 100
	M	P - 250 N - 50	P - 50 N - 150

$$\begin{aligned} \text{Gini Impurity} &= 1 - \sum_{i=1}^K p_i^2 \\ &= 1 - \text{Gini Index} \end{aligned}$$

where K is the number of class labels,

p_i is the proportion of i^{th} class label

DT-Example

- ▶ Gini Index uses the probability of finding a data point with one label as an indicator for homogeneity
- ▶ if the dataset is completely homogeneous, then the probability of finding a datapoint with one of the labels is 1 and the probability of finding a data point with the other label is zero.
- ▶ An empirical estimate of the probability p_i of finding a data point with label i (assuming the target attribute can take say k distinct values) is just the ratio of the number of data points with label i to the total number of data points.
- ▶ It must be that $\sum_{i=1}^K p_i = 1$. For binary classification problems the probabilities for the two classes become p and $(1 - p)$.

DT-Example

- ▶ Gini index is maximum when $P_i = 1$ for exactly one of the classes and all others are zero.
- ▶ Higher the Gini index higher the homogeneity.
- ▶ In a Gini based decision tree algorithm, we therefore find the split that maximizes the weighted sum (weighted by the size of the partition) of the Gini indices of the two partitions created by the split.

DT-Example

- ▶ Split on gender: the two partitions will have 10/500 and 300/500 as the probabilities of finding a football player respectively. Each partition is half the total population.
- ▶ $Gini = 1/2((1/50)^2 + (49/50)^2) + 1/2((3/5)^2 + (2/5)^2) = 0.7404$
- ▶ Split on Age: the two partitions will have 260/700 and 50/250 as the probabilities, and 700 and 300 as the sizes respectively.
- ▶ $Gini = 0.7((26/70)^2 + (44/70)^2) + 0.3((1/5)^2 + (4/5)^2) = 0.5771$
- ▶ Therefore we would first need to split on the gender — this split gives a higher GINI index for the partitions. Gini index can only be used on classification problems where the target attribute is categorical.

Source code

```
# Importing decision tree classifier from sklearn library
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
# Fitting the decision tree with default hyperparameters
dt_default = DecisionTreeClassifier()
dt_default.fit(X_train, y_train)
# Let's check the evaluation metrics of our default model
# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report, confusion_matrix
# Making predictions
y_pred_default = dt_default.predict(X_test)
tree.plot_tree(dt_default, feature_names=features)
# Printing classification report
print(classification_report(y_test, y_pred_default))
# Printing confusion matrix
print(confusion_matrix(y_test, y_pred_default))
```

DT Hyper parameters

- ▶ Criterion (Gini/IG or entropy)
- ▶ max_features
- ▶ max_depth
- ▶ min_samples_split
- ▶ min_samples_leaf

Tuning for max_depth

```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'max_depth': range(1, 40)}
# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini", random_state =100)
# fit tree on training data
tree = GridSearchCV(dtree, parameters, cv=n_folds,scoring="accuracy")
tree.fit(X_train, y_train)
```

Tuning for min_samples_leaf

```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
# specify number of folds for k-fold CV
n_folds = 5
# parameters to build the model on
parameters = {'min_samples_leaf': range(5, 200, 20)}
# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini", random_state = 100)
# fit tree on training data
tree = GridSearchCV(dtree, parameters, cv=n_folds, scoring="accuracy")
tree.fit(X_train, y_train)
```

Tuning for min_samples_split

```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
# specify number of folds for k-fold CV
n_folds = 5
# parameters to build the model on
parameters = {'min_samples_split': range(5, 200, 20)}
# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini", random_state =100)
# fit tree on training data
tree = GridSearchCV(dtree, parameters, cv=n_folds,scoring="accuracy")
tree.fit(X_train, y_train)
```


Final optimal parameters

Create the parameter grid

```
param_grid = {  
    'max_depth': range(5, 15, 5),  
    'min_samples_leaf': range(50, 150, 50),  
    'min_samples_split': range(50, 150, 50),  
    'criterion': ["entropy", "gini"]  
}
```

n_folds = 5

Instantiate the grid search model

```
dtree = DecisionTreeClassifier()  
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,  
    cv = n_folds, verbose = 1)
```

Fit the grid search to the data

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

Build the model

```
# model with optimal hyperparameters
clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100,
max_depth=10,
min_samples_leaf=50,
min_samples_split=50)
clf_gini.fit(X_train, y_train)
# accuracy score
clf_gini.score(X_test, y_test)
# plotting the tree
dot_data = StringIO()
export_graphviz(clf_gini,
out_file=dot_data, feature_names=features, filled=True, rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

Random Forest

- ▶ Random Forest - Collection of Decision Trees
- ▶ **Bagging** (bootstrap **aggregating**) means training each decision tree on a random subset of the examples in the training set.
- ▶ Each decision tree in the random forest is trained on a *different subset* of examples.
- ▶ Ensembles, a collection of models is used to make predictions, rather than individual models.
- ▶ Arguably, the most popular in the family of ensemble models is the random forest: an ensemble made by the combination of a large number of decision trees.

RF Source Code

```
# Importing random forest classifier from sklearn library
from sklearn.ensemble import RandomForestClassifier
# Running the random forest with default parameters.
rfc = RandomForestClassifier()
# fit
rfc.fit(X_train,y_train)
# Making predictions
predictions = rfc.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
# Let's check the report of our default model
print(classification_report(y_test,predictions))
print(accuracy_score(y_test,predictions))
# Printing confusion matrix
print(confusion_matrix(y_test,predictions))
```

K-Nearest Neighbour -Classifier

- ▶ Supervised ML Algorithm - used for regression and classifier
- ▶ Non-parametric algorithm, since there are no assumptions on data distribution
- ▶ In other models if the data grows, parameters also grow, but this not the same with KNN.
- ▶ The distance calculation step requires quadratic time complexity, and the sorting of the calculated distances requires an $O(N \log N)$ time. Together, we can say that the process is an $O(N^3 \log N)$ process, which is a monstrously long process.
- ▶ **Space complexity:**
- ▶ Since it stores all the pairwise distances and is sorted in memory on a machine, memory is also the problem. Usually, local machines will crash, if we have very large datasets.

Naïve Bayes

- ▶ Naïve Bayes is a probabilistic classifier which returns the probability of a test point belonging to a class rather than the label of the test point.
- ▶ The algorithm assumes that the features are conditionally independent, meaning that the presence of one feature in a class has no effect on the presence of any other feature.
- ▶ The algorithm then uses Bayes' Theorem to calculate the probability of each class, and the class with the highest probability is the predicted class

NB

- ▶ Bayes Theorem

- ▶ Probability -

- ▶ The **probability** of an event is the measure of the chance that the event will occur as a result of an experiment. The **probability** of an event A is the number of ways event A can occur divided by the total number of possible outcomes.

- ▶ $P(E)$ - probability of an event “E”

- ▶ Joint probability $P(A|B)=P(A \cap B)/P(B)$

- ▶ $P(A|B)$ is the probability of event A occurring, given that event B occurs. Example: given that you drew a red card, what's the probability that it's a four ($p(\text{four}|\text{red})=2/26=1/13$). So out of the 26 red cards (given a red card), there are two fours so $2/26=1/13$.

NB

- ▶ Conditional probability - $P(A \cap B) = P(A|B) * P(B)$
- ▶ $P(A \text{ and } B)$.
- ▶ The probability of event A **and** event B occurring.
- ▶ It is the probability of the intersection of two or more events.
- ▶ The probability of the intersection of A and B may be written $p(A \cap B)$.
- ▶ Example: the probability that a card is a
four and red = $p(\text{four and red}) = 2/52 = 1/26$.

(There are two red fours in a deck of 52, the 4 of hearts and the 4 of diamonds).

Bayes Theorem

- **Bayes Theorem** : Given Probability of an event B occurring given event A has already occurred and individual Probabilities of A and B we can find the reverse conditional probability $P(A|B)$ by using what is called Bayes Theorem

$$\begin{array}{l} P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \\ P(A|B) = \frac{P(A \cap B)}{P(B)} \end{array} \quad \left. \vphantom{\begin{array}{l} P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \\ P(A|B) = \frac{P(A \cap B)}{P(B)} \end{array}} \right\} \text{Bayes Theorem}$$

Types of NB Classifiers

- ▶ **Gaussian Naïve Bayes (GaussianNB):** This is a variant of the NB classifier, which is used with Gaussian distributions—i.e. normal distributions—and continuous variables. This model is fitted by finding the mean and standard deviation of each class.
- ▶ **Multinomial Naïve Bayes (MultinomialNB):** This type of NB classifier assumes that the features are from multinomial distributions. This variant is useful when using discrete data, such as frequency counts, and it is typically applied within natural language processing use cases, like spam classification.
- ▶ **Bernoulli Naïve Bayes (BernoulliNB):** This is another variant of the NB classifier, which is used with Boolean variables—that is, variables with two values, such as True and False or 1 and 0.

NB

- ▶ The Naive Bayes algorithm is well-suited for large datasets and is effective for both binary and multi-class classification.
- ▶ It's often used for
 - ▶ text classification
 - ▶ spam filtering
 - ▶ recommendation systems
 - ▶ Document classification
 - ▶ Sentiment Analysis
 - ▶ Mental state predications

NB

- ▶ The textblob library also supports a random forest classifier,
- ▶ Which works best on text written in proper English such as a formal letter might be.
- ▶ For text that is not usually written with proper grammar, such as customer feedback, Naive Bayes works better.
- ▶ Naive Bayes has another advantage in real-time analytics. You can update the model without losing the previous training.

Model Fitting

- ▶ Fitting a model means that you're making your algorithm learn the relationship between predictors and outcome so that you can predict the future values of the outcome.
- ▶ So the best fitted model has a specific set of parameters which best defines the problem at hand

Types of Fitting

- ▶ Under Fit
- ▶ Appropriate
- ▶ Over Fit

What is Clustering ?

- ▶ “Clustering” is the process of dividing the datasets into groups, consisting of similar data-points
- ▶ It means grouping of objects based on the information found in the data, describing the objects or their relationship

Clustering used ?

- ▶ The goal of clustering is to determine the intrinsic grouping in a set of Unlabelled data
- ▶ Sometimes, partitioning is the goal
- ▶ Retail store, Banking, Insurance,
- ▶ NetFlix, Flickr (recommending)
- ▶ Spam filtering

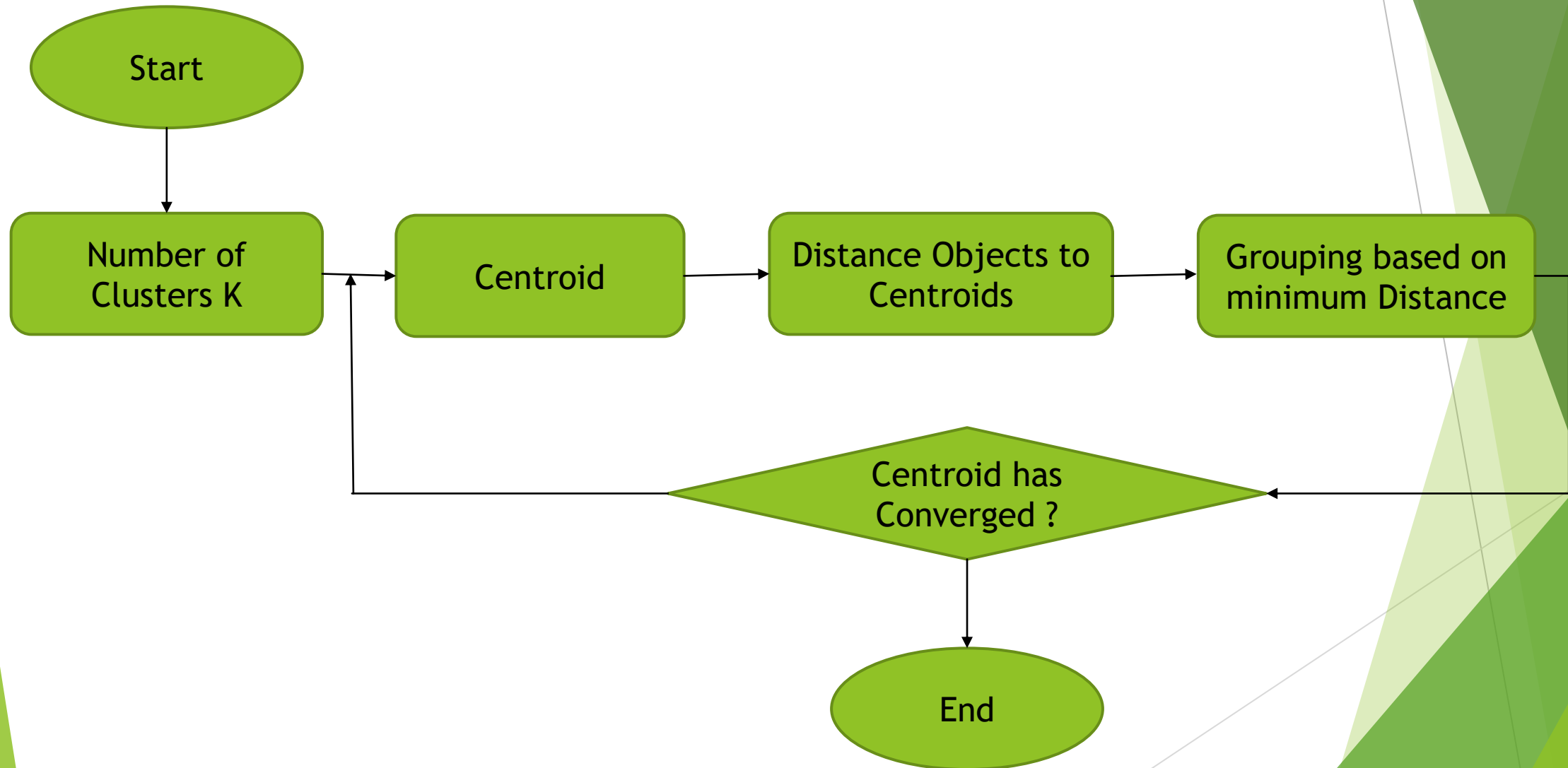
Types of Clustering

- ▶ Exclusive Clustering - Kmeans
 - ▶ Overlapping Clustering - CMeans
 - ▶ Hierarchical Clustering
-
- ▶ Centroid-based clustering algorithms(K-Means)
 - ▶ Connectivity-based clustering algorithms (Hierarchical clustering),
 - ▶ Distribution-based clustering algorithms
 - ▶ Density-based clustering algorithms.

K-Means Clustering

- The process by which objects are classified into a predefined number of groups so that they are as much dissimilar as possible from one group to another group, but as much similar as possible within each group.

K-Means Algorithm Working



Kmeans Clustering Steps

- ▶ First we need to decide the no of clusters to be made. (Guessing)
- ▶ Then we provide centroids of all the clusters. (Guessing)
- ▶ The Algorithm calculates Euclidian distance of the points from each centroid and assigns the point to the closer cluster.
- ▶ Next the centroids are calculated again, when we have our new cluster.
- ▶ The distance of the points from the centre of clusters are calculated again and points are assigned to the closest cluster.
- ▶ And then again the new centroid for the cluster is calculated
- ▶ These steps are repeated until we have a repetition in centroids or new centroids are very close to the previous ones.

How to Decide the no of Clusters

- ▶ The Elbow method:
- ▶ First of all, compute the sum of squared error (SSE) for some values of “K” (for example 2,4,6,8, etc.). The SSE is defined as the sum of squared distance between each member of the cluster and its centroid.
- ▶
$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist(x, c_i)^2$$

Pro & Cons

- ▶ Simple, understandable
- ▶ Items automatically assigned to clusters
- ▶ Must define number of clusters
- ▶ All items forced into clusters
- ▶ Unable to handle noisy data & Outliers

Fuzzy C - Means

- ▶ Is an extension of K-Means, the popular simple clustering technique
- ▶ Fuzzy clustering (also referred to as soft clustering) is a form of Clustering in which each data point can belong to more than one cluster.

Pros & Cons

- ▶ Allow a data point to be in multiple clusters
- ▶ A more natural representation of the behaviours of genes
- ▶ Genes usually are involved in multiple functions

- ▶ Need to define “C”, the no of clusters
- ▶ Need to determine membership cut-off value
- ▶ Clusters are sensitive to initial assignment of centroids
- ▶ Fuzzy c-means is not a deterministic algorithm

Hierarchical Clustering

- ▶ Hierarchical clustering is an alternative approach which builds a hierarchy from the bottom-up, and doesn't require us to specify the number of clusters beforehand.
- ▶ Hierarchical clustering analysis, is a cluster analysis technique that creates a hierarchy of clusters from points in a dataset.
- ▶ With clustering, data points are put into groups – known as clusters – based on similarities like colour, shape or other features.
- ▶ Each cluster is placed within a nested tree-like hierarchy, where clusters are grouped and break down further into smaller clusters depending on similarities.
- ▶ Closer clusters are together in the hierarchy, the more similar they are to each other.

Hierarchical Clustering

- ▶ K-means can visualize data points as distinct and linear groups, hierarchical clustering visualizes data groups in relation to one another with multiple levels of similarity.
- ▶ Hierarchical clustering is used to help find patterns and related occurrences within datasets, especially those that are complex or multifaceted.
- ▶ Hierarchical clustering process involves finding the two data points closest to each other and combining the two most similar ones.
- ▶ After repeating this process until all data points are grouped into clusters, the end result is a hierarchical tree of related groups known as a dendrogram.

Hierarchical Clustering

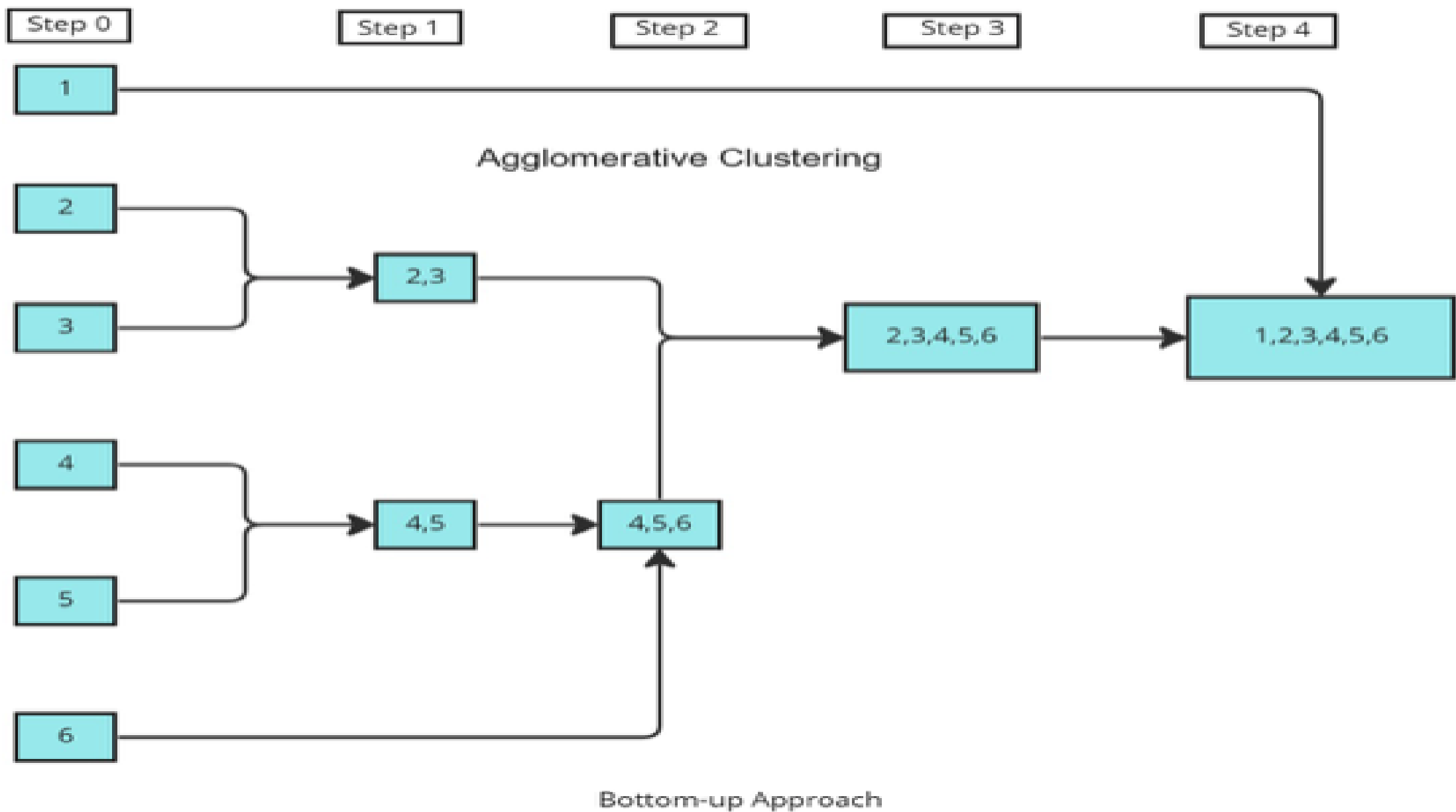
- ▶ Dendogram
- ▶ **Agglomerative (HAC) - bottom-up**
 - ▶ Divide the data points into different clusters and then aggregate them as the distance decreases.
- ▶ **Divisive H Clustering - top-down**
 - ▶ Combine all the data points as a single cluster and divide them as the distance between them increases.

Agglomerative Clustering

- ▶ Bottom-up approach.
- ▶ It starts clustering by treating the individual data points as a single cluster, then it is merged continuously based on similarity until it forms one big cluster containing all objects.
- ▶ It is good at identifying small clusters.
- ▶ The steps for agglomerative clustering are:
 1. Compute the proximity matrix using a distance metric.
 2. Use a linkage function to group objects into a hierarchical cluster tree based on the computed distance matrix from the above step.
 3. Data points with close proximity are merged together to form a cluster.
 4. Repeat steps 2 and 3 until a single cluster remains.

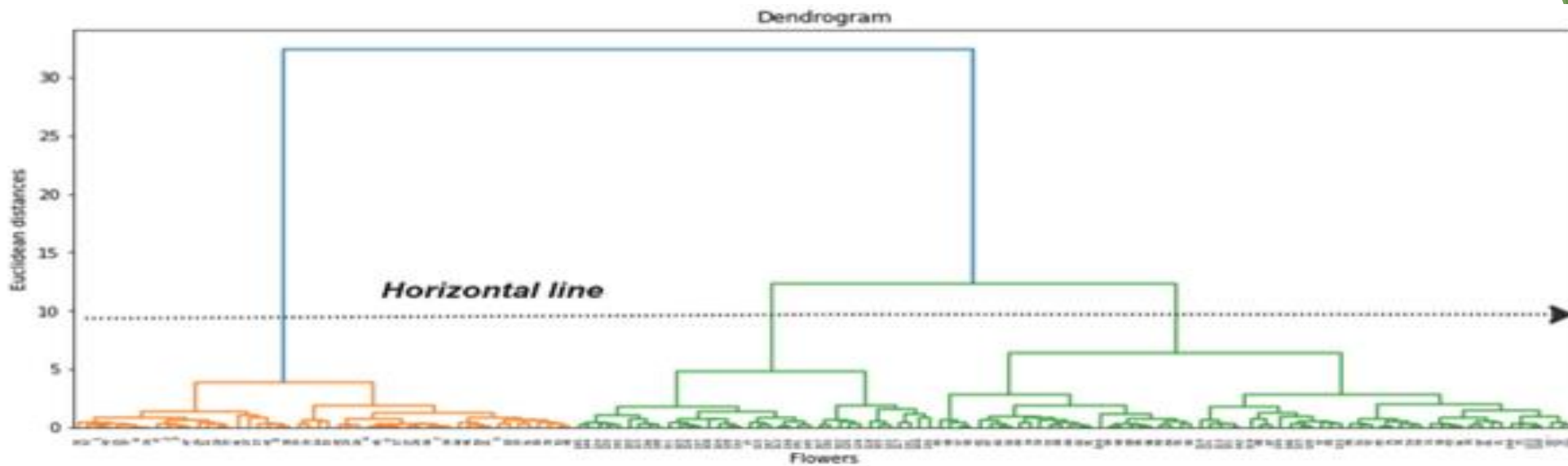
Proximity Matrix

	x_1	x_2	x_3	...	x_n
x_1	$d(x_1, x_1)$	$d(x_1, x_2)$	$d(x_1, x_3)$...	$d(x_1, x_n)$
x_2	$d(x_2, x_1)$	$d(x_2, x_2)$	$d(x_2, x_3)$...	$d(x_2, x_n)$
x_3	$d(x_3, x_1)$	$d(x_3, x_2)$	$d(x_3, x_3)$...	$d(x_3, x_n)$
...
x_n	$d(x_n, x_1)$	$d(x_n, x_2)$	$d(x_n, x_3)$		$d(x_n, x_n)$



Types Linkage

- **Complete linkage:** The maximum of all pairwise distance between elements in each pair of clusters is used to measure the distance between two clusters.
- **Single linkage:** The minimum of all pairwise distance between elements in each pair of clusters is used to measure the distance between two clusters.
- **Average linkage:** The average of all pairwise distances between elements in each pair of clusters is used to measure the distance between two clusters.
- **Centroid linkage:** Before merging, the distance between the two clusters' centroids are considered.
- **Ward's Method:** It uses squared error to compute the similarity of the two clusters for merging.



- ▶ how to find the optimal number of clusters from the above chart?
- ▶ To find it, draw a horizontal line where there is no overlap in the vertical lines of the bars.
- ▶ The number of bars without the overlap below the line is the optimal number of the clusters.
- ▶ From the above figure, we have three bars below the horizontal line, so the optimal number of clusters is three.

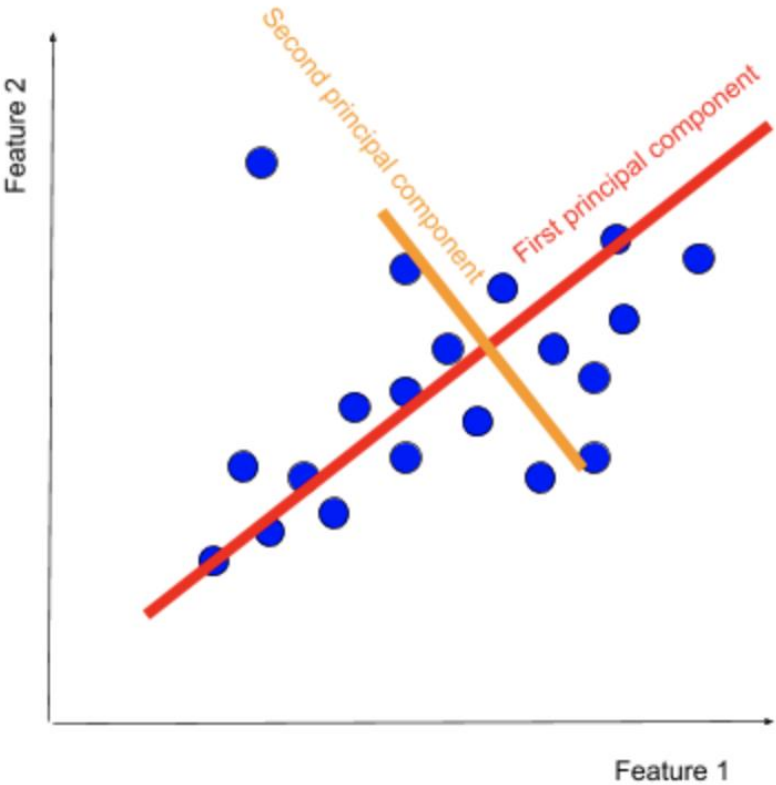
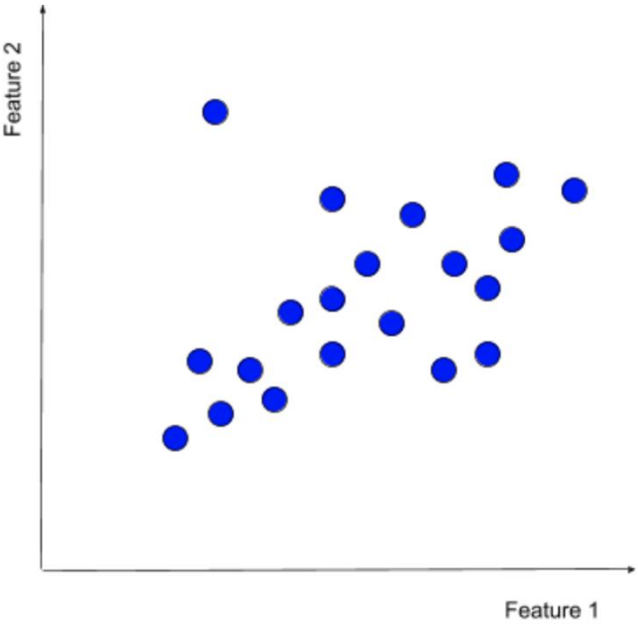
Pro & Cons

- ▶ No assumptions of a particular number of clusters
- ▶ May corresponds to meaningful taxonomies
- ▶ Once a decision is made to combine two clusters, it can't be undone
- ▶ Too slow for large datasets

Principle Component Analysis - PCA

- ▶ PCA is one of the most commonly used unsupervised ML algorithms across a variety of applications:
 - ▶ Exploratory Data Analysis
 - ▶ Dimensionality Reduction
 - ▶ Information Compression
 - ▶ Data de-noising

Feature 1	Feature 2
4	2
6	3
13	6
...	...



PCA

- ▶ PCA are vectors, they are not chosen randomly
- ▶ The first principal component is computed so that it explains the greatest amount of variance in the original features.
- ▶ The second component is orthogonal to the first, and it explains the greatest amount of variance left *after* the first principal component.
- ▶ The original data can be represented as feature vectors.
- ▶ PCA allows us to go a step further and represent the data as linear combinations of principal components.
- ▶ Getting principal components is equivalent to a linear transformation of data from the feature1 x feature2 axis to a PCA1 x PCA2 axis.

PCA

- ▶ Large datasets with the number of dimensions can surpass 100 different variables, principal components remove noise by reducing a large number of features to just a couple of principal components.
- ▶ Principal components are orthogonal projections of data onto lower-dimensional space.
- ▶ In theory, PCA produces the same number of principal components as there are features in the training dataset.
- ▶ In practice, though, we do not keep all of the principal components.
- ▶ Each successive principal component explains the variance that is left after its preceding component, so picking just a few of the first components sufficiently approximates the original dataset *without* the need for additional features.

Math behind PCA

- ❑ Eigen decomposition of the covariance matrix
- ❑ Singular value decomposition of the data matrix
- ❑ Eigenvalue approximation via power iterative computation
- ❑ Non-linear iterative partial least squares (NIPALS) computation

PCA use cases

- ▶ Visualize multidimensional data
- ▶ Compress information
- ▶ Simplify complex business decisions.
- ▶ Clarify convoluted scientific processes
- ▶ PCA is used as part of preprocessing
 - ▶ Reduce the number of dimensions in the training dataset.
 - ▶ De-noise the data.

PCA Assumptions and Limitations

- ▶ Cor-relation between features
- ▶ Sensitive to the scale of the features
- ▶ Sensitive against outliers
- ▶ Assumes a linear relationship between features
- ▶ Cannot handle missing values

Pros & Cons

Pros

- ▶ Easy to compute
- ▶ Speeds up other ML algorithms
- ▶ Counteracts the issues of high-dimensional data

Cons

- ▶ Low interpretability of PCA
- ▶ The trade-off b/w information loss & dimensionality reduction

Ensemble

- ▶ Ensemble technique is a machine learning technique that combine several base models in order to produce one optimum predictive model
- ▶ Collectively these models are called as ENSEMBLE model
- ▶ Provided model should be different from each other
- ▶ For making the model different we can adopt 2 ways
- ▶ 1) Algorithm chosen for the model should be different
 - ▶ LR, SVM, NB, KNN & so on
- ▶ 2) If we select same algorithm for all the model, then choose different data for all the model by same sampling method

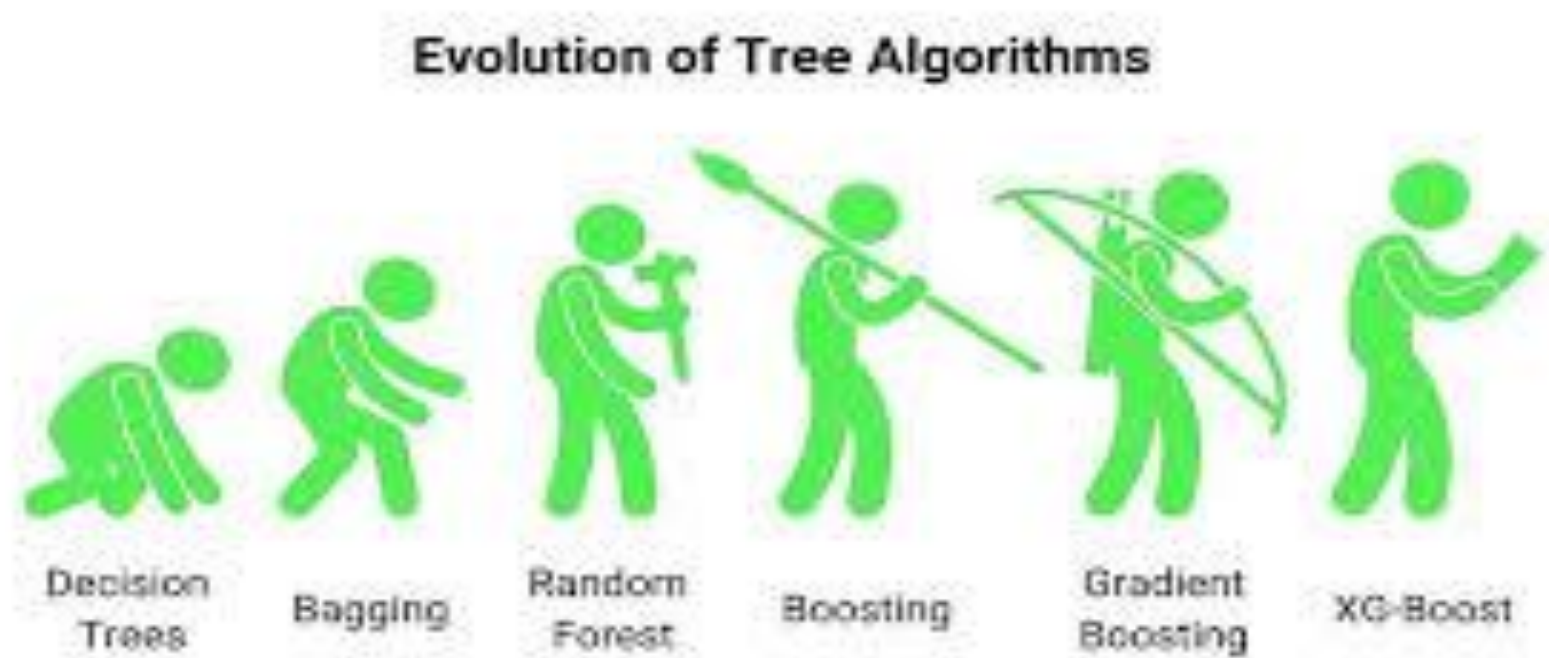
Ensemble

- ▶ In Classification problem output is decided based on majority count for the particular class predicted by all the model
- ▶ In Regression mean or average of all the output predicted by model

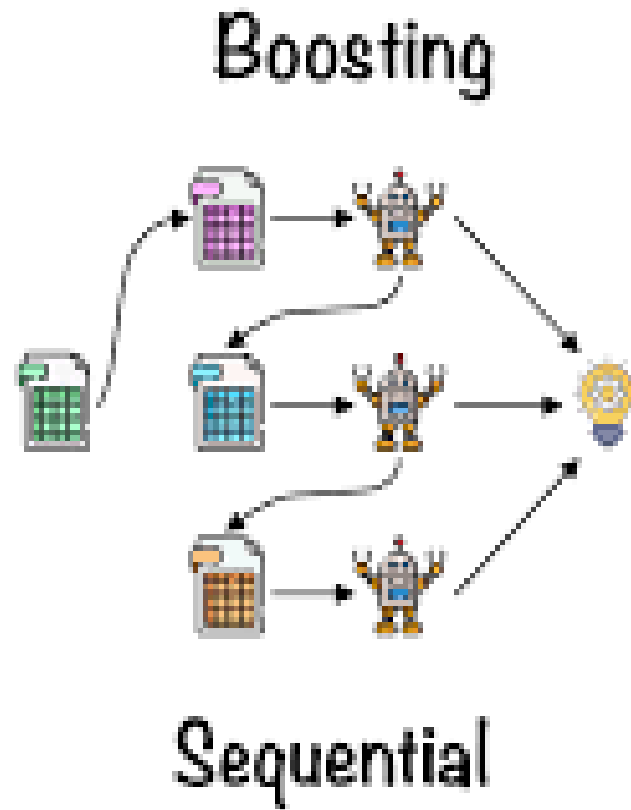
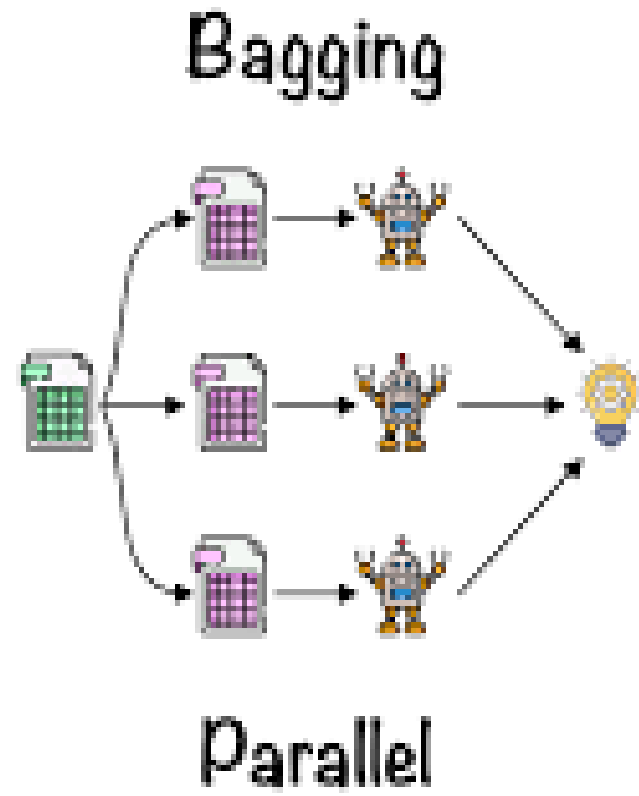
Types of Ensemble

- ▶ Voting Ensemble
 - ▶ model should be different or different algorithm
- ▶ Bagging - RF
- ▶ Boosting - AdaBoost, GBM, XGBoost.
- ▶ Stacking

Evolution of Tree Algorithms

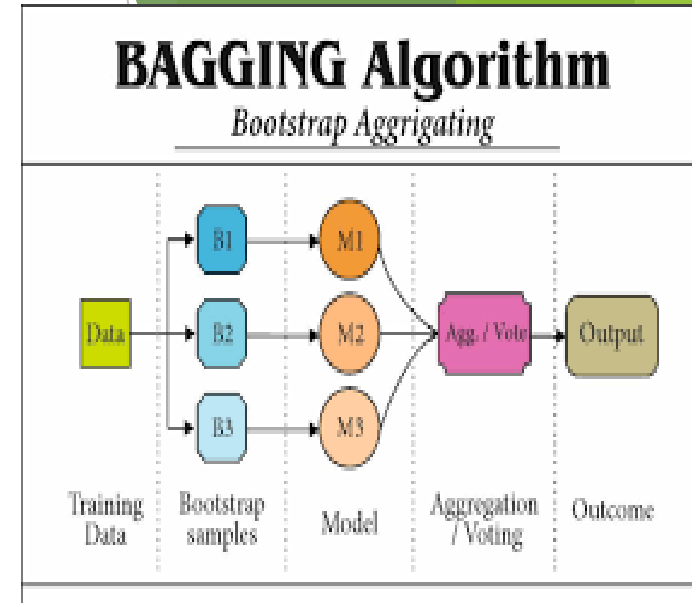


Bagging & Boosting

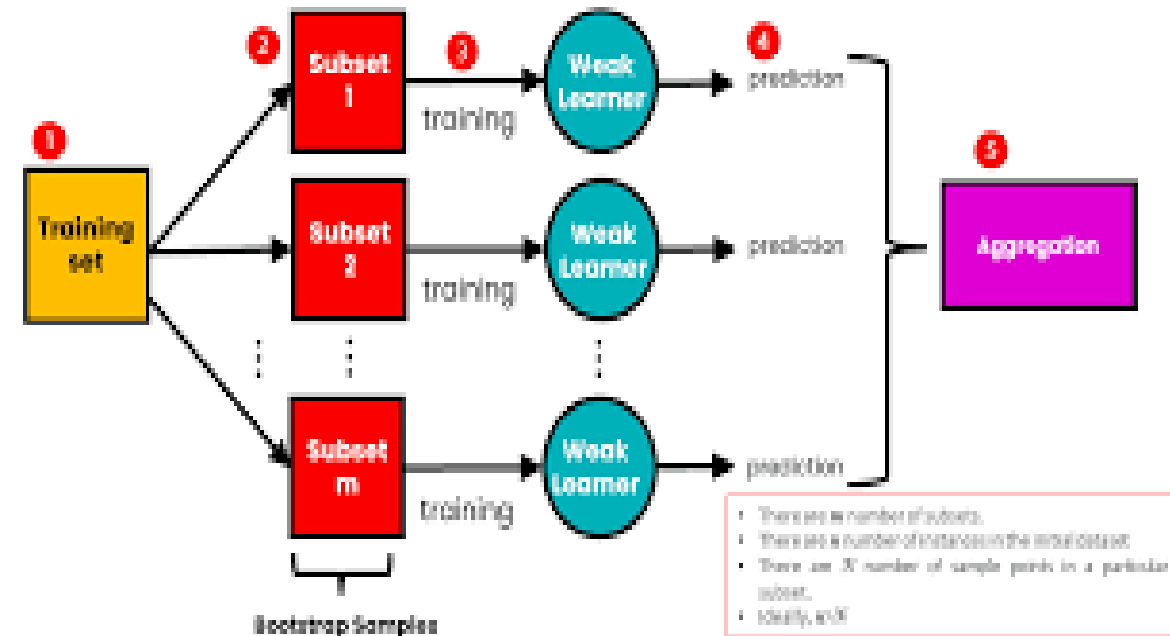


Bagging

- ▶ Divide the actual data set into multiple sub data sets & apply Ensemble ML model
- ▶ Boot strapping the data sets & Aggregate the results
- ▶ Parallel
- ▶ Subset of the dataset
- ▶ Unweighted subset
- ▶ Equally weighted predictions
- ▶ No Bias Reduction, but reduces variance
- ▶ Less prone to overfitting

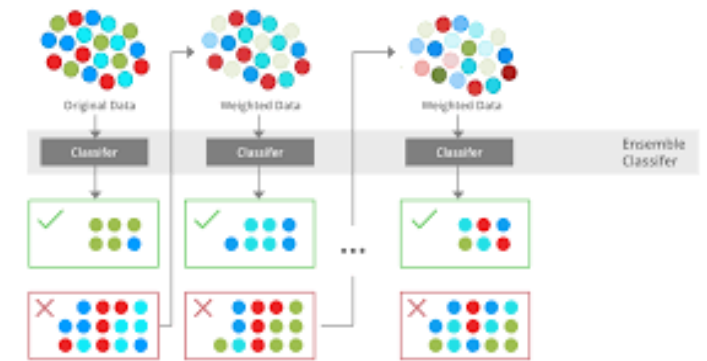


The Process of Bagging (Bootstrap Aggregation)

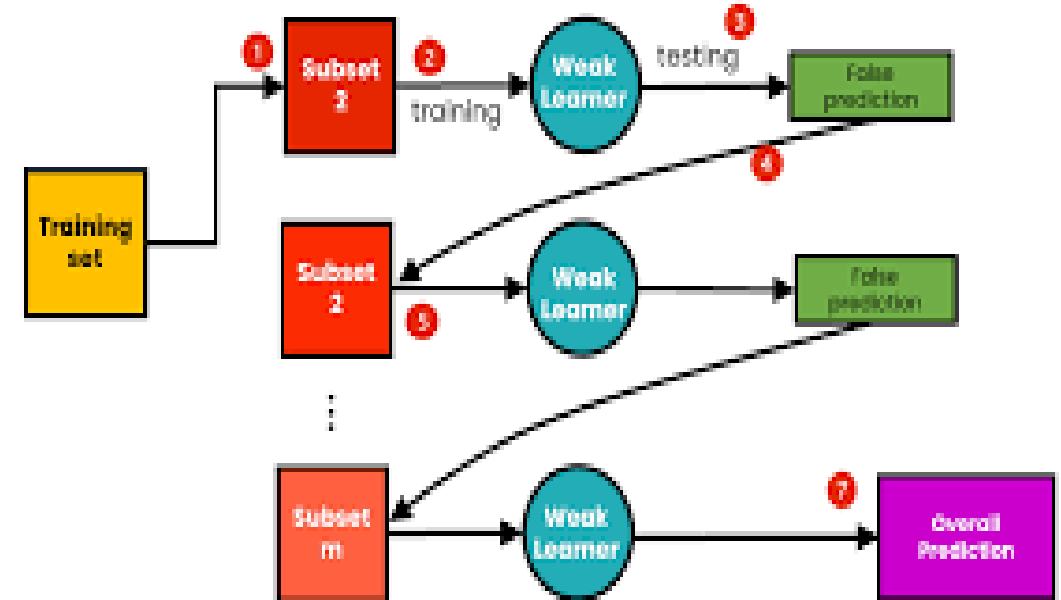


Boosting

- ▶ Apply the ML model on the dataset & choose the misclassified values & build another model
- ▶ Weight up the incorrectly classified samples (up sampling - ADABOOST)
- ▶ Sequential
- ▶ Same data set for all the stages
- ▶ Weighted Dataset
- ▶ Weighted Average for predictions
- ▶ Bias Reduction and reduces variance to
- ▶ More prone to overfitting



The Process of Boosting



Gradient Boosting Model (GBM)

- ▶ GBM is one of the most popular forward learning ensemble methods in machine learning.
- ▶ It is a powerful technique for building predictive models for regression and classification tasks.
- ▶ GBM helps us to get a predictive model in form of an ensemble of weak prediction models such as decision trees.
- ▶ Whenever a decision tree performs as a weak learner then the resulting algorithm is called gradient-boosted trees.
- ▶ It enables us to combine the predictions from various learner models and build a final predictive model having the correct prediction.

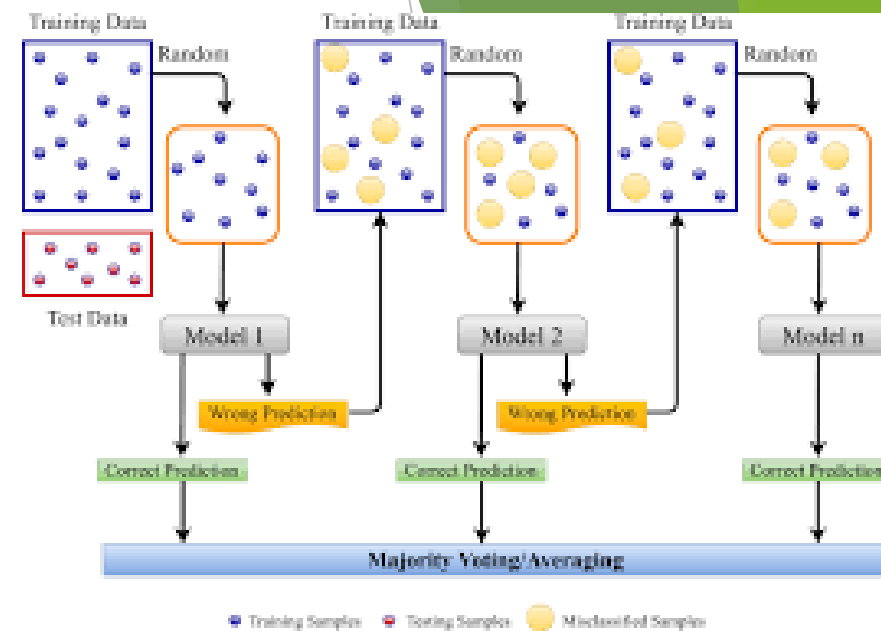
GBM

► Gradient boosting machines consist 3 elements as follows:

- Loss function
- Weak learners
- Additive model

XGBOOST

- ▶ Linear Regression works better with LINEAR DATA
- ▶ Naïve Bayes works better with TEXT DATA
- ▶ Random Forest - Generic Small Data
- ▶ SVM - Generic Small Data
- ▶ Gradient Boosting - Generic Small Data
- ▶ Common problems include - Overfitting(performance) & Scalability (speed)
- ▶ To overcome these problems we have XGBOOST
- ▶ 2014 - major Kaggle competition winners were of XGBOOST
- ▶ 2016 - open source project



XGBOOST - www.xgboost.ai

- ▶ Cross Platform
- ▶ Multi language support
 - ▶ python, R, Java, Ruby, Swift, Julia, C, C++
- ▶ Integration with other libraries and tools
- ▶ Support all kinds of ML problems
- ▶ Optimized
- ▶ Used by Multiple projects
- ▶ Good Documentation
- ▶ Many competitions popularized
- ▶ Large user base

XGBOOST major feature

- ▶ Major ML Models are python based, where as Enterprise applications are JAVA based.
- ▶ Earlier, build the model in python & expose the model in the form of API service, later can be consumed by JAVA or others languages
- ▶ XGBOOST eased this by multilanguage support
- ▶ Build the model in python & consume it in java

Integration with other libs & tools

- ▶ Model Building
 - ▶ Numpy, pandas, matplotlib, scikit
- ▶ Distributed Systems
 - ▶ Spark, PySpark, Dask
- ▶ Model Integration
 - ▶ SHAP, LIME
- ▶ Model Deployment
 - ▶ Docker, Kubernetes
- ▶ MLOps & workflow management
 - ▶ AirFlow, MLFlow

ML Problems - XGBoost supports

- ▶ Regression problems
- ▶ Classification problems - Binary-class & Multi-class
- ▶ Time Series forecasting
- ▶ Ranking -
- ▶ Anomaly Detection

XGBOOST - Speed

- ▶ Training time is less
- ▶ Parallel Processing - `njobs=-1` feature
- ▶ Optimized Data Structures - column block
- ▶ Cache Awareness - reason for training time is less
- ▶ Out of Core computing - chunking - hyper para - `tree_method=hist`
- ▶ Distributed Computing - training the model on distributed nodes & aggregate them (external libs Dask or Kubernetes)
- ▶ GPU Support - `tree_method="gpu_hist"`

XGBOOST - Performance

- ▶ Regularized Learning Objective -
- ▶ Handling Missing Values
- ▶ Sparsity Aware Split Finding -
- ▶ Efficient Split Finding (Weighted Quantile Sketch + Approximate Tree Learning)
converts continuous values to discrete values
- ▶ Tree Pruning
 - ▶ trim the tree
 - ▶ reduces the complexity
 - ▶ Pre, post and gamma value

XGBOOST - others

- ▶ LIGHT GBM
- ▶ CAT BOOST

Source Code

XGBoost model for Pima Indians dataset

```
from numpy import loadtxt
```

```
from xgboost import XGBClassifier
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
# load data
```

```
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
```

```
# split data into X and y
```

```
X = dataset[:,0:8]
```

```
Y = dataset[:,8]
```

Source code-2

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
```

```
# fit model training data  
model = XGBClassifier()  
model.fit(X_train, y_train)
```

```
# make predictions for test data  
y_pred = model.predict(X_test)  
predictions = [round(value) for value in y_pred]
```

```
# evaluate predictions  
accuracy = accuracy_score(y_test, predictions)  
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

XGBoost Regressor

evaluate an xgboost regression model on the housing dataset

```
from numpy import absolute
```

```
from pandas import read_csv
```

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import RepeatedKFold
```

```
from xgboost import XGBRegressor
```

load the dataset

```
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
```

```
dataframe = read_csv(url, header=None)
```

```
data = dataframe.values
```

XGBoost regressor

split data into input and output columns

```
X, y = data[:, :-1], data[:, -1]
```

define model

```
model = XGBRegressor()
```

define model evaluation method

```
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
```

evaluate model

```
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
```

force scores to be positive

```
scores = absolute(scores)
```

```
print('Mean MAE: %.3f (%.3f)' % (scores.mean(), scores.std())) # Mean Square Error
```

CAT Boost

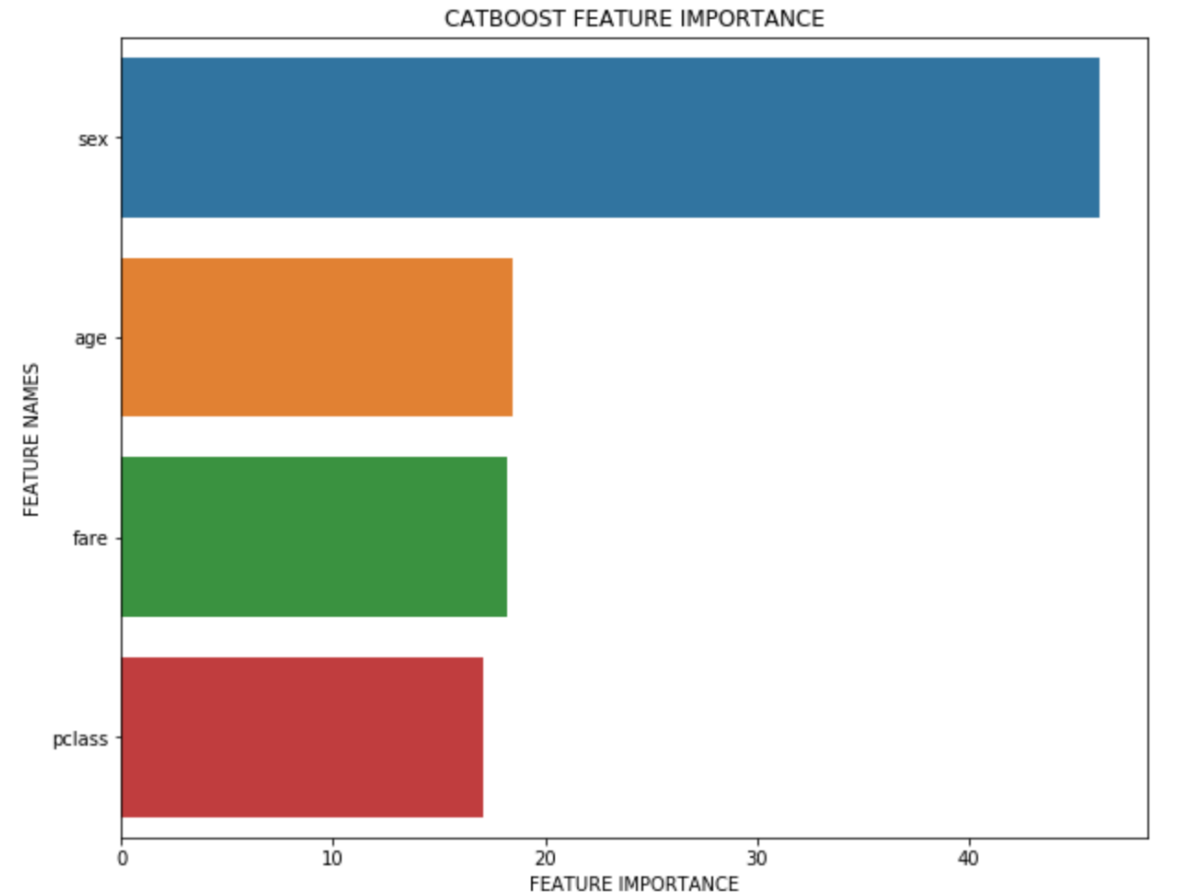
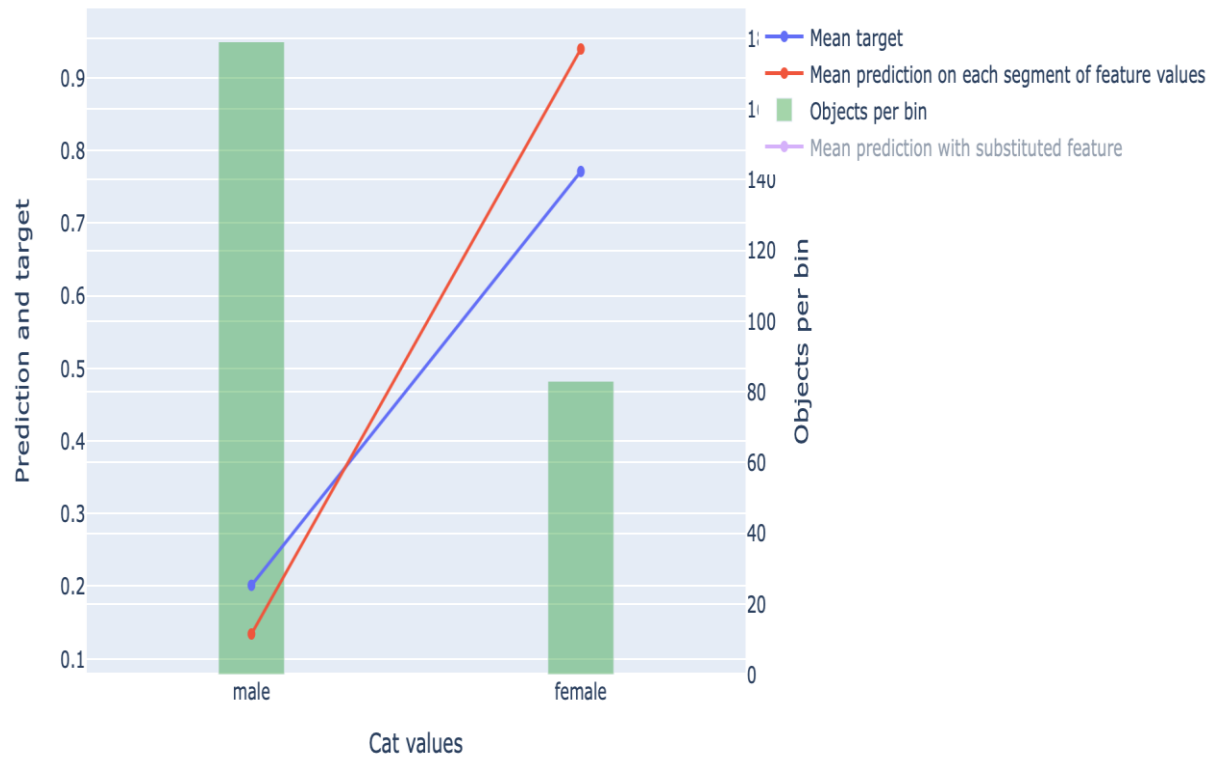
- ▶ Catboost is a boosted decision tree machine learning algorithm
- ▶ It works in the same way as other gradient boosted algorithms such as XGBoost but provides support out of the box for categorical variables, has a higher level of accuracy without tuning parameters and also offers GPU support to speed up training.
- ▶ CatBoostClassifier
- ▶ CatBoostRegressor

CATBoost

```
from pandas.api.types import is_numeric_dtype
import catboost as cb
```

```
df = pd.read_csv('titanic.csv')
df.dropna(subset=['survived'], inplace=True)
X = df[['pclass', 'sex', 'age', 'fare']]
y = df['survived']
X['pclass'] = X['pclass'].astype('str')
X['fare'].fillna(0, inplace=True)
X['age'].fillna(0, inplace=True)
```


	precision	recall	f1-score	support
0.0	0.84	0.83	0.83	162
1.0	0.73	0.74	0.73	100
accuracy			0.79	262
macro avg	0.78	0.78	0.78	262
weighted avg	0.79	0.79	0.79	262



Bayes Theorem

- ▶ Good For Large Data
- ▶ Given a Hypothesis H and evidence E , Bayes theorem states that the relationship between the probability of the hypothesis before getting the evidence $P(H)$ and the probability of the hypothesis after getting the evidence $P(H|E)$ is
- ▶ $P(H|E) = P(E|H) \times P(H) / P(E)$
- ▶ News Categorization
- ▶ Object & Face recognition
- ▶ Medical Diagnosis

Discriminant Analysis

- ▶ The algorithm involves developing a probabilistic model per class based on the specific distribution of observations for each input variable.
- ▶ A new example is then classified by calculating the conditional probability of it belonging to each class and selecting the class with the highest probability.
- ▶ Linear - LDA
- ▶ Quadratic - QDA

Ex 1) A survey of middle school students asked their grade and their favorite type of music. The results are shown in the table.

a) What is the probability that a 6th grader likes hip-hop?

$$\frac{10}{36} = 0.278 \quad \boxed{27.8\%}$$

b) Given that a student likes rock music, what is the probability that they are in 7th grade?

$$\frac{5}{15} = 0.333 \quad \boxed{33.3\%}$$

	6th	7th	8th	Total
Rock	8	5	2	15
Country	3	4	1	8
Hip-hop	10	18	15	43
Pop	15	12	7	34
Total	36	39	25	100

Linear Discriminant Analysis

- ▶ Classification ML algorithm using flavour of bayes theorem
- ▶ Fisher Linear Discriminant - statistical approach (dim reduction)
- ▶ It works by calculating summary statistics for the input features by class label, such as the mean and standard deviation.
- ▶ It is probabilistic classification model that makes strong assumptions about the distribution of each input variable, although it can make effective predictions even when these expectations are violated (e.g. it fails gracefully).
- ▶ linear algebra operations are used to calculate the required quantities efficiently via matrix decomposition.
- ▶ LDA can be applied to enhance the operation of classification algorithms such as a decision tree or random forest.

LDA

- ▶ The primary function of LDA is to project high-dimensional data on to a lower-dimensional space while retaining the data's inherent class separability.
- ▶ LDA can be used for binary and multi-class classification problems

LDA Assumptions

- ▶ Input variables are numeric
- ▶ Normally distributed
- ▶ Data has same variance
- ▶ If this is not, it may be desirable to transform the data to have a Gaussian distribution and standardize or normalize the data prior to modelling.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis  
model = LinearDiscriminantAnalysis()
```

Tuning the Model

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['solver'] = ['svd', 'lsqr', 'eigen']
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

Quadratic Discriminant Analysis (QDA)

- ▶ QDA is a method you can use when you have a set of predictor variables and you'd like to classify a response variable into two or more classes.
- ▶ It is a non-linear equivalent to linear discriminant analysis.
- ▶ `from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis`

Time Series Forecasting

- ▶ Time-series data is a sequence of data-points, typically ordered in time
- ▶ Data which is INDEX BY TIME
- ▶ Sequence
- ▶ Capture the past data and based on that predict for future
- ▶ Forecasting models make predictions at regular intervals, such as Hourly, Daily or Weekly.
- ▶ Plotting time series
- ▶ Components of time series
- ▶ Forecasting time series

Plotting

- ▶ Line plot gives overall view, not so interpretable
- ▶ Line plot gives having understanding about long term, what happened to data
- ▶ Line plots are difficult to understand
- ▶ To overcome this we have a technique - SMOOTHING

Smoothing

- ▶ Moving Average - Ex: Rainfall data

Months - M1, M2, M3 M200

Data - Y1, Y2, Y3 Y200

- ▶ Where M1 - January, M2 - February, M3 - March, for window size = 3
- ▶ $Y2 = Y1+Y2+Y3/3$ $Y3 = Y2+Y3+Y4/3$ 4 and so on
- ▶ First & Last data points remains the same
- ▶ For even no of window size = 5
- ▶ $Y3 = y2+y3+y4+y5/4$ - giving high priority to FUTURE data
- ▶ $Y3 = y1+y2+y3+y4/4$ - giving priority to PAST data
- ▶ Solution = $\frac{1}{2}[y1+y2+y3+y4/4] + \frac{1}{2}[y2+y3+y4+y5/4]$

Components of Time Series

- ▶ Trend - holistic view - moving average
- ▶ Seasonality - periodic change
- ▶ Residual - noise or error - lesser the better

Trend, Seasonality, Noise

- ▶ To find seasonality and Noise, subtract trend from original data
- ▶ Lets say we have data of year 2001 to 2010

Month - M1, M2, M3 M120

Data - Y1, Y2, Y3 Y120

Trend = T1 = M1

T2 = M1+M2+M3/3 T3=M2+M3+M4/3

Seasonality = S1 = Jan = (T1 +T13 + T25+ T109)/10

S2 = Feb = (T2 +T14 + T26+ T110)/10

Noise = N1 = Y1-T1-S1

= N2 = Y2-T2-S2

= N120 = Y120 - T120 - S120

Forecasting in Time Series

- ▶ Model Based forecasting
- ▶ Data Based Forecasting
 - ▶ **SIMPLE EXPONENTIAL** - there is no TREND & SEASONALITY
 - ▶ **DOUBLE EXPONENTIAL**- there is TREND & no SEASONALITY
 - ▶ **HOLT's WINTER MODEL** - (Triple Exponential) there is TREND & SEASONALITY
 - ▶ When we are not sure about data has TREND & SEASONALITY, better to use HOLT's winter model

Simple Exponential

- ▶ Day - D1,D2,D3,D4,D5,D6,D7.....D200
- ▶ Temp -t1,t2,t3,t4,t5,t6,t7.....t200
- ▶ What will be easiest way to predict the temperature for tomorrow, when we have data up to today ?
- ▶ One way - take the average all the temperatures
 - ▶ This is not a good method since temp varies a lot throughout the year summer to winter
- ▶ Another way - whatever the temp is today - it will be same tomorrow
 - ▶ Today its it 28 so tomorrow will be 28

Simple Exponential

- ▶ Instead of taking all 200 data into consideration, we will take recent data for prediction.
- ▶ SE tries to calculate the component in local average
- ▶ Local Average

Double Exponential

- ▶ We cannot rely on local average because, some values change each day
- ▶ Local average + TREND = prediction

Holts winter model

- ▶ Here along with local average we will find out trend as well as seasonality

Model Based Fore Casting

- ▶ AR (Auto Regressive)
 - ▶ MA (Moving Average)
 - ▶ ARMA (Autoregressive moving average)
 - ▶ ARIMA (Autoregressive integrated moving average)
-
- ▶ Data should be stationary
 - ▶ Distribution of data should be same across the time
 - ▶ If data is stationary then go for Model based, otherwise Data-based Model since there is no any condition

Auto Regressive - AR

- ▶ AR is completely depend on past data and nothing else from future prediction
- ▶ Similar to Regression Model
- ▶ In Regression $Y_Pred/Y_Hat = \text{Alpha } X + \text{Beta} + \text{Error}$
- ▶ Future $Y\text{-Hat} = \text{Alpha } Y\text{-hat-1} + \text{Beta} + \text{Error}$

- ▶ Here future data is only depend on past data
- ▶ If it would depend on past two data
- ▶ Python Module
- ▶ **`from statsmodels.tsa.ar_model import AutoReg`**

AR

- ▶ $Y_t = \text{Alpha1 } Y_{t-1} + \text{Alpha2 } Y_{t-2} + \text{Beta} + \text{Error}$ AR(2)
- ▶ $Y_t = \text{Alpha1 } Y_{t-1} + \text{Alpha2 } Y_{t-2} + \text{Alpha3 } Y_{t-3} + \text{Beta} + \text{Error}$ AR(3)
- ▶ Y_t - Tomorrow
- ▶ $\text{Alpha1 } Y_{t-1}$ - Today
- ▶ $\text{Alpha2 } Y_{t-2}$ - Yesterday
- ▶ $\text{Alpha3 } Y_{t-3}$ - Day before yesterday
- ▶ AR1, AR2, AR3 - choose the best model based on the accuracy
- ▶ AR1 - similar to SLR
- ▶ AR2 - similar to MLR with 2 features

Moving Average - MA

- ▶ MA - future data is depend on external factors
- ▶ Ex: rate of some production which depends on season (grain/fruit)
- ▶ Data = $Y_1 \ Y_2 \ Y_3 \ \dots \ Y_t$
- ▶ External factors = $E_1 \ E_2 \ E_3 \ \dots \ E_t$

- ▶ $Y_t = \text{Alpha } E_{t-1} + \text{Beta} + E_t \quad \text{MA(1)}$
- ▶ E_{t-1} depends on External Error factors
- ▶ Beta - constant

- ▶ $Y_t = \text{Alpha1 } E_{t-1} + \text{Alpha2 } E_{t-2} + \text{Beta} + E_t \quad \text{MA(2)}$
- ▶ $Y_t = \text{Alpha1 } E_{t-1} + \text{Alpha2 } E_{t-2} + \text{Alpha3 } E_{t-3} + \text{Beta} + E_t \quad \text{MA(3)}$

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

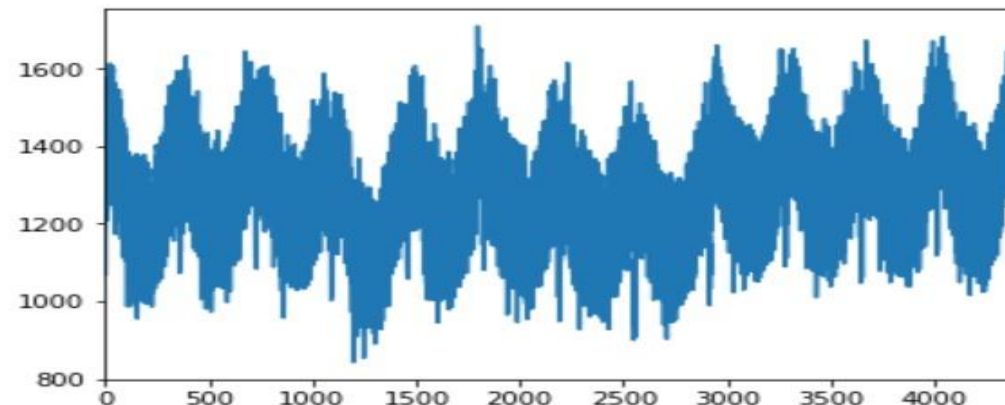
```
# Load AutoReg class from statsmodels.tsa.ar_model module
```

```
from statsmodels.tsa.ar_model import AutoReg
# Load and plot the time-series data
```

```
url='https://raw.githubusercontent.com/jenfly/opsd/master/opsd_germany_daily.csv'
df = pd.read_csv(url,sep=",")
df['Consumption'].plot()
```

```
In [43]: df['Consumption'].plot()
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x2b9a63591d0>
```



```
# Check for stationarity of the time-series data
# We will look for p-value. In case, p-value is less than 0.05, the time series
# data can said to have stationarity
#
from statsmodels.tsa.stattools import adfuller
#
# Run the test
#
df_stationarityTest = adfuller(df['Consumption'], autolag='AIC')
#
# Check the value of p-value
#
print("P-value: ", df_stationarityTest[1])
#
# Next step is to find the order of AR model to be trained
# for this, we will plot partial autocorrelation plot to assess
# the direct effect of past data on future data
#
from statsmodels.graphics.tsaplots import plot_pacf
pacf = plot_pacf(df['Consumption'], lags=25)
```



```
#  
# Create training and test data  
#  
train_data = df['Consumption'][:len(df)-100]  
test_data = df['Consumption'][len(df)-100:]  
#  
# Instantiate and fit the AR model with training data  
#  
ar_model = AutoReg(train_data, lags=8).fit()  
#  
# Print Summary  
#  
print(ar_model.summary())
```

```
#  
# Make the predictions  
#  
pred = ar_model.predict(start=len(train_data), end=(len(df)-1), dynamic=False)  
#  
# Plot the prediction vs test data  
#  
from matplotlib import pyplot  
pyplot.plot(pred)  
pyplot.plot(test_data, color='red')
```

SRC code

```
# Import the ARIMA and plot_predict from statsmodels
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_predict
```

```
# Forecast the first MA(1) model
mod = ARIMA(COLUMNTOPREDICT, order=(1,1,1))
res = mod.fit()
print(res.summary())
```

```
# Plot the data and the forecast
fig, ax = plt.subplots()
simulated_data_1.loc[950:].plot(ax=ax)
plot_predict(res, start=____, end=____, ax=ax)
plt.show()
```

ARMA

- ▶ Future data only going to deal with both PAST data and EXTERNAL factors
- ▶ $ARMA(p,q) = AR(p) + MA(q)$
- ▶ $Y_t = \alpha_1 Y_{t-1} + \beta_1 \epsilon_{t-1} + \beta_0 + \epsilon_t$
- ▶ If $\beta_1 \epsilon_{t-1} = 0$ then it will be $AR(1)$
- ▶ If $\alpha_1 Y_{t-1} = 0$ then it will be $MA(1)$

ARIMA

- ▶ Auto Regressive Integrated Moving Average
- ▶ ARIMA doesn't care about data is stationary or not
- ▶ $Y_1 \ Y_2 \ Y_3 \ Y_4 \ \dots \ Y_n$
- ▶ Transformed data
- ▶ $Z_1 \ Z_2 \ Z_3 \ Z_4 \ \dots \ Z_n$

Evaluation of Time Series Model

- ▶ Mean Square Error (MSE)
- ▶ MAPE Score - Mean Absolute Percentage Error

Market Basket Analysis -UNSPL

- ▶ Market basket analysis explains the combinations of products that frequently co-occur in transactions
- ▶ Ex: while buying a “bread” he may buy “butter” - provide them offer
- ▶ Market Basket Analysis Algorithm
 - ▶ 1. Association Rule Mining
 - ▶ 2. Apriori

Association Rule Mining

- ▶ Association Rule Mining is a technique that shows how items are associated to each other.
- ▶ Example:
 - ▶ Customer who purchase bread have a 60% likelihood of also purchasing JAM
 - ▶ Customers to purchase laptops are more likely to purchase laptop bags
- ▶ It means that if a persons buys item “A” then will also buy item “B”
- ▶ Three common way to measure association
- ▶ Support, Confidence, Lift

Support, conf, lift

- ▶ Support gives the fraction of transactions which contains the item A and B
- ▶ **Support = $\text{freq}(A,B) / N$**
- ▶ Confidence gives how often the items A & B occur together, given no of times A occur
- ▶ **Confidence = $\text{freq}(A,B) / \text{freq}(A)$**
- ▶ Lift indicates the strength of a rule over the random co-occurrence of A & B
- ▶ **Lift = $\text{support} / \text{supp}(A) \times \text{supp}(B)$**

Recommendation System

- ▶ Subclass of Information Filtering System that seeks to predict the rating or the preferences a user might give to an item
- ▶ Algorithm that suggests relevant item to users
- ▶ NetFlix, Amazon
- ▶ Three main types
 - ▶ Content based RS
 - ▶ Popularity based RS
 - ▶ Collaborative RS - User-based Filtering & Item-based Filtering

Content Based RS

- ▶ Rs that suggest item to users based on characteristics or features or features of item itself.
- ▶ it analyses the content or attribute of item, such as text, genre, actors or other metadata to identify to make recommendation
- ▶ It assumes that user will be interested in item, that are similar to those they have liked or engaged in the past

Popularity Based RS

- ▶ RS that suggest item to user based on the popularity or overall popularity among all the users.
- ▶ It relies on the assumption that popular item are more likely to be preferred by users and therefore recommends item that already or have high number of ratings or engagement metrics
- ▶ It typically rank the items based on their popularity or sales and recommends top rank items to user, without considering user preference and behaviours
- ▶ Pros - simplicity and ease of implementation & not needed complex algorithms or user specific data
- ▶ Cons - it does not take into account individual user preference interest or specific item characteristics

Collaborative RS

- ▶ RS that suggest item to users based on some pattern or user interaction or behaviours.
- ▶ It uses past behaviour of users such as rating, reviews or purchase history to identify similarities or pattern among users and item and make recommendation based on these similarities
- ▶ User Based Filtering - in this approach similarities among the users are used to make recommendation
- ▶ Users who have similar behaviour or preference in the past are considered to have similar taste and item liked or rated highly by users with similar behaviour are recommended to target user

Item Based Filtering

- ▶ In this approach similarities among the items are used to make recommendation
- ▶ Item that often liked or rated highly by same user in the past are considered to be similar and item similar to those liked or rated highly by the target user are recommended

Apriori Algorithm

- ▶ Uses frequent item sets to generate association rules. It is based on the concept that a subset of a frequent itemset must also be frequent itemset.
- ▶ Frequent item set is an itemset whose support value is greater than a threshold value
- ▶ If $\{A,B\}$ is a frequent item set, then $\{A\}$ and $\{B\}$ should be a frequent item sets
- ▶ Python library
- ▶ `from apyori import apriori`

Src Code

```
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3,
min_length=2) association_results = list(association_rules)

print(len(association_rules))

print(association_rules[0])

for item in association_rules:
    # first index of the inner list
    # Contains base item and add item
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])
    #second index of the inner list
    print("Support: " + str(item[1]))
    #third index of the list located at 0th
    #of the third index of the inner list
    print("Confidence: " + str(item[2][0][2])) print("Lift: " + str(item[2][0][3]))
    print("=====")
```


Reinforcement Learning

- ▶ Reinforcement Learning is a type of ML where an agent learns to behave in an environment by performing actions and seeing the results
- ▶ Markov Decision Process = Shortest path Process
- ▶ Q-Learning

Deep Learning

- ▶ Deep neural networks, deep belief networks and recurrent neural networks have been applied to fields such as computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, and bioinformatics where they produced results comparable to and in some cases better than human experts have.
- ▶ Deep Learning Algorithms and Networks –
- ▶ are based on the unsupervised learning of multiple levels of features or representations of the data. Higher-level features are derived from lower level features to form a hierarchical representation.
- ▶ use some form of gradient descent for training.

Deep Learning

- ▶ The act of sending data straight through a neural network is called a **feed forward neural network**.
- ▶ Our data goes from input, to the layers, in order, then to the output.
- ▶ When we go backwards and begin adjusting weights to minimize loss/cost, this is called **back propagation**.
- ▶ This is an **optimization problem**. With the neural network, in real practice, we have to deal with hundreds of thousands of variables, or millions, or more.
- ▶ The first solution was to use **stochastic gradient descent as optimization** method. Now, there are options like AdaGrad, Adam Optimizer and so on.

Deep learning models/algorithms

- ▶ Some of the popular models within deep learning are as follows –
- ▶ **Convolutional Neural Networks (CNN)**
- ▶ **Recurrent Neural Networks (RNN)**
- ▶ **Deep Belief Networks (DBN)**
- ▶ **Generative Adversarial Networks (GAN)**
- ▶ **Auto-Encoders and so on**

Deep Learning

- ▶ The circles are neurons or nodes, with their functions on the data and the lines/edges connecting them are the weights/information being passed along.
- ▶ Each column is a layer. The first layer of your data is the input layer. Then, all the layers between the input layer and the output layer are the hidden layers.
- ▶ If you have one or a few hidden layers, then you have a **shallow neural network**. If you have many hidden layers, then you have a **deep neural network**.
- ▶ Deep structured learning or hierarchical learning or deep learning in short is part of the family of machine learning methods which are themselves a subset of the broader field of Artificial Intelligence.
- ▶ Deep learning is a class of machine learning algorithms that use several layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.

TensorFlow

- ▶ TensorFlow provides a collection of workflows to develop and train models using Python, JavaScript, or Swift, and to easily deploy in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Keras

- ▶ Keras is a high-level neural networks API, written in Python and capable of running on top of [TensorFlow](#), [CNTK](#), or [Theano](#). It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*
- ▶ Use Keras if you need a deep learning library that:
- ▶ Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- ▶ Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- ▶ Runs seamlessly on CPU and GPU.

Theano

- ▶ **Theano is a Python library** that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
- ▶ Theano, computations are expressed using a NumPy-esque syntax and compiled to run efficiently on either CPU or GPU architectures

PyTorch

- ▶ **PyTorch** is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing.
- ▶ It is primarily developed by Facebook's AI Research lab (FAIR).
- ▶ It is free and open-source software released under the Modified BSD license.
- ▶ Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.
- ▶ A number of Deep Learning software are built on top of PyTorch, including Uber's Pyro and HuggingFace's Transformers
- ▶ PyTorch provides two high-level features:
- ▶ Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- ▶ Deep neural networks built on a tape-based autodiff system

CNN



RNN



AutoEncoders

- ▶ Autoencoders are special neural networks that learn how to recreate the information given.
- ▶ They are useful for many tasks, like reducing the number of features in a dataset, extracting meaningful features from data, detecting anomalies, and generating new data.
- ▶ Encoders
 - ▶ compresses the input data into reduced data - latent code
- ▶ Bottleneck layer
 - ▶ Holds the latent code - essential features of input data
- ▶ Decoders
 - ▶ Reproduces the actual input data from compressed latent code

Single Layer NN



Neural Networks

- ▶ A set of inputs are passed to the first hidden layer, the activations from that layer is passed to the next layer and so on, until it reach the output layer
- ▶ Where the results of the classification are determined by the scores at each node. This happens for each node.
- ▶ This series of events starting from the input where each activation is sent to the next layer - known as FORWARD PROPAGATION / forward PROP
- ▶ Each node has the classifier and none of them fire randomly, so if you repeat an input, you get the same output

Why ?

- ▶ To Analyze Simple pattern - SVM
- ▶ Moderate patterns - Neural Networks
- ▶ Complex patterns - Deep Networks

Unlabelled data

- ▶ Unsupervised learning
- ▶ Feature Extraction
- ▶ Pattern recognition

- ▶ Restricted Boltzmann (RBM)
- ▶ Auto Encoder

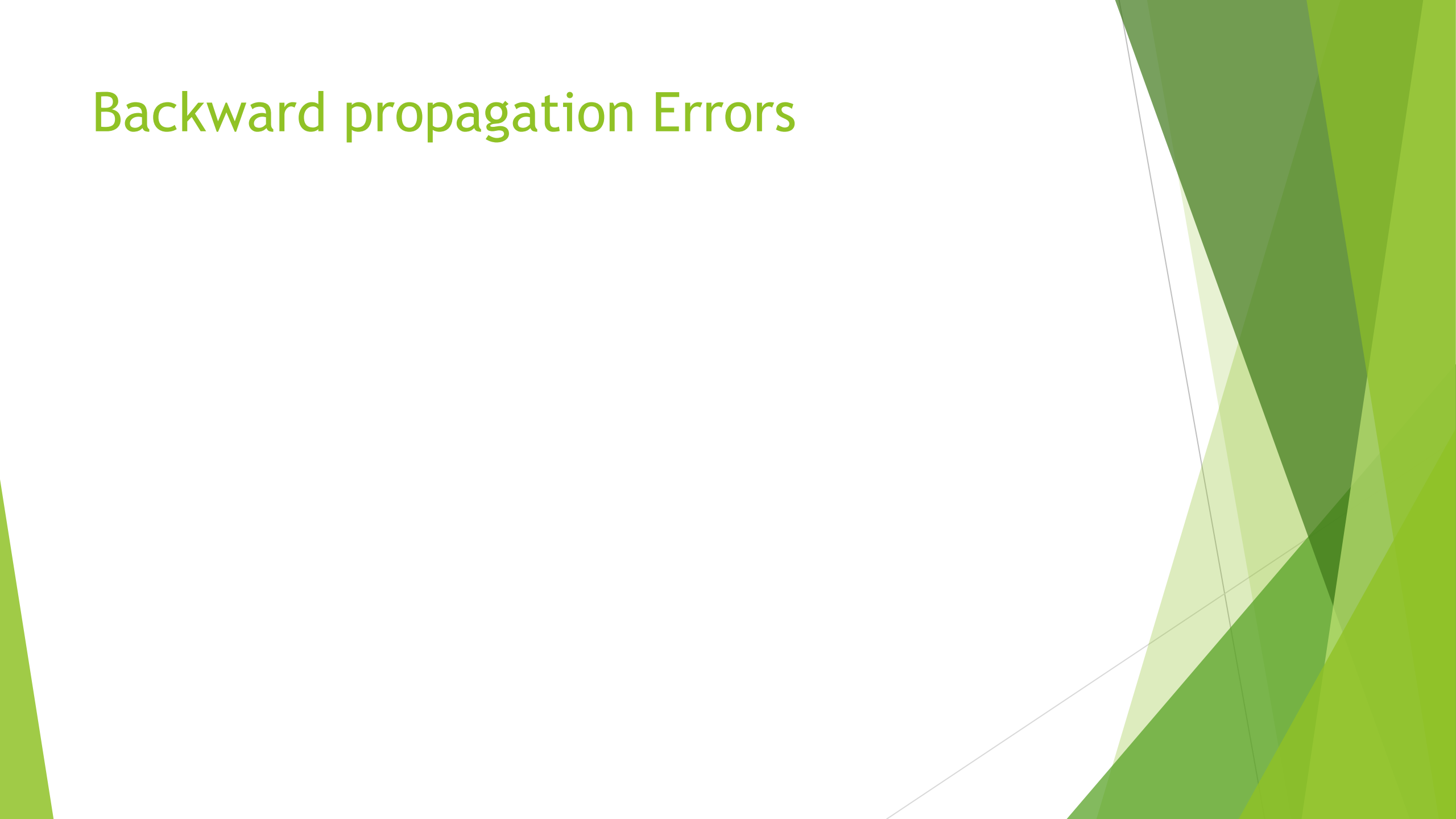
Labelled Data - classifier

- ▶ Supervised learning
 - ▶ Text processing (sentiment analysis, parsing, named entity recognition)
 - ▶ Recursive Neural Tensor Network(RNTN), for char recognition - RNN
 - ▶ Image recognition - Deep Belief Network (DBN), CNN
 - ▶ Object recognition - RNTN, CNN
 - ▶ Speech recognition - RNN
-
- ▶ In General DBN, Multi Layered Perceptrons (MLP) with Rectified Linear Units (RELU) - are good choices for CLASSIFICATION problems
 - ▶ For time series analysis - RNN

Activation Functions

- ▶ Sigmoid
- ▶ Hyperbolic
- ▶ Tangent
- ▶ ReLU

Backward propagation Errors





CNN

- ▶ $B \times A \times 3$
- ▶ B - ROWS
- ▶ A - COLUMNS
- ▶ 3 - channels i.e RGB

- ▶ Ex:
- ▶ $28 \times 28 \times 3$ (coloured channels)
- ▶ $28 \times 28 \times 2$ (B & W channels)
- ▶ Fully Connected Networks
 - ▶ $28 \times 28 \times 3 =$ total no weights required in the hidden layer 2,352
 - ▶ $200 \times 200 \times 3 = 1,20,000$ - requires more neurons & leads to overfitting

CNN

- ▶ The neuron in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a FULLY Connected Manner
- ▶ Convolution Layer
- ▶ RELU Layer
- ▶ Pooling layer
- ▶ Fully connected

CNN

- ▶ Store Image white pixels into -1 & black pixels to 1
- ▶ CNN Compares the image piece by piece. The pieces that it looks for are called features.
- ▶ By finding rough feature matches, in roughly the same position in two images, CNN gets a lot better at seeing similarity than whole image matching schemes
- ▶ Collect FEATURES or FILTERS
- ▶ These are small pieces of the bigger image. We choose a feature and put it on the input image, if it matches then the image is classified correctly

Convolution Layer

- ▶ Here we will move the filter to every possible position on the image
- ▶ Line up the feature and the image
- ▶ Multiply each image pixel by the corresponding feature pixel
- ▶

ReLU Layer

- ▶ In this layer we remove every negative values from the filtered images and replaces it with ZERO's
- ▶ This is done to avoid the values from summing up to ZERO
- ▶ Rectified Linear Unit - transform function only activates a node if the input is above a certain quantity, while the input is below ZERO, the output is ZERO, but when the input rises a above certain threshold. It has a linear relationship with the dependent variable
- ▶ $F(x) \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

Pooling layer

- ▶ Pick a window size (usually 2 or 3)
 - ▶ Pick a stride (usually 2)
 - ▶ Walk your window across your filtered images
 - ▶ From each window, take the min value
-
- ▶ Choose window size = 2
 - ▶ Stride= 2

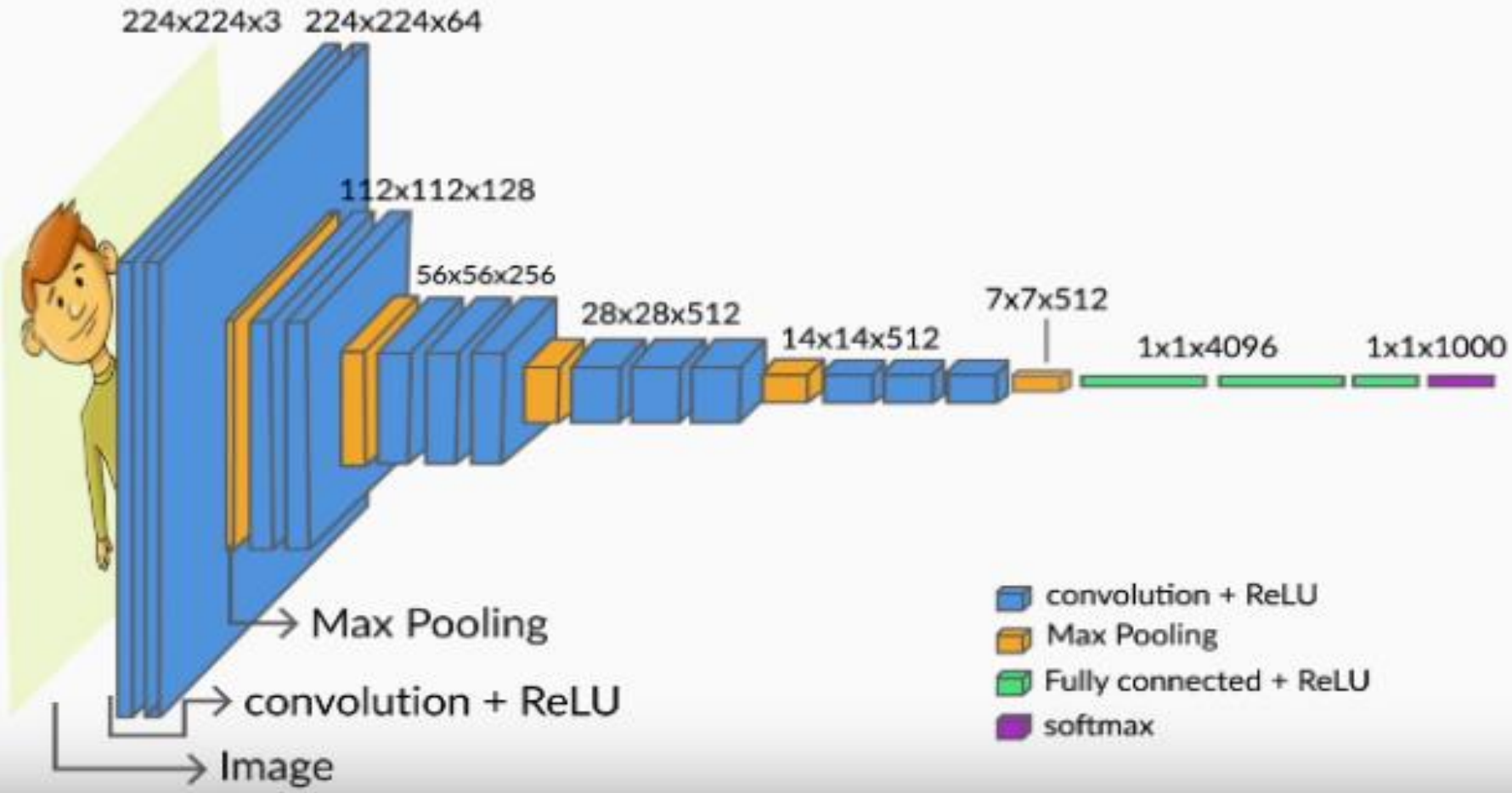
How to build CNN

- ▶ Download the DATA SET
- ▶ ENCODE the LABELS
- ▶ Resize image to 50 x 50 pixel & read as GREY SCALE
- ▶ Split the data, 24500 images for training and 500 for testing
- ▶ Reshape the data appropriately for TF
- ▶ Build the MODEL
- ▶ Calculate LOSS function, it is categorical cross entropy
- ▶ ADAM as optimizer with learning rate set to 0.0001
- ▶ Train the DNN for 10 epochs
- ▶ Make predictions

CNN

- ▶ AlexNet
- ▶ GoogleNet
- ▶ VGGNEt
- ▶ ResNet
- ▶ SeNet

VGGNet Architecture



RNN

- ▶ **Recurrent Neural Networks** - neural networks in which data can flow in any direction. These networks are used for applications such as language modelling or Natural Language Processing (NLP).
- ▶ Recurrent neural networks are a type of neural network where outputs from previous time steps are taken as inputs for the current time step.
- ▶ The basic concept underlying RNNs is to utilize sequential information.
- ▶ If we want to predict the next word in a sentence we have to know which words came before it.
- ▶ Long short-term memory networks (LSTMs) are most commonly used RNNs

GPipe

- ▶ GPipe is a scalable pipeline parallelism library that enables learning of giant deep neural networks.
- ▶ It partitions network layers across accelerators and pipelines execution to achieve high hardware utilization.
- ▶ It leverages re-computation to minimize activation memory usage.
- ▶ For example, using partitions over 8 accelerators, it is able to train networks that are 25× larger, demonstrating its scalability. It also guarantees that the computed gradients remain consistent regardless of the number of partitions. It achieves an almost linear speed up without any changes in the model parameters: when using 4× more accelerators, training the same model is up to 3.5× faster. We train a 557 million parameters AmoebaNet model and achieve a new state-of-the-art 84.3% top-1 / 97.0% top-5 accuracy on ImageNet.

References

- ▶ <https://365datascience.com/linear-regression/>