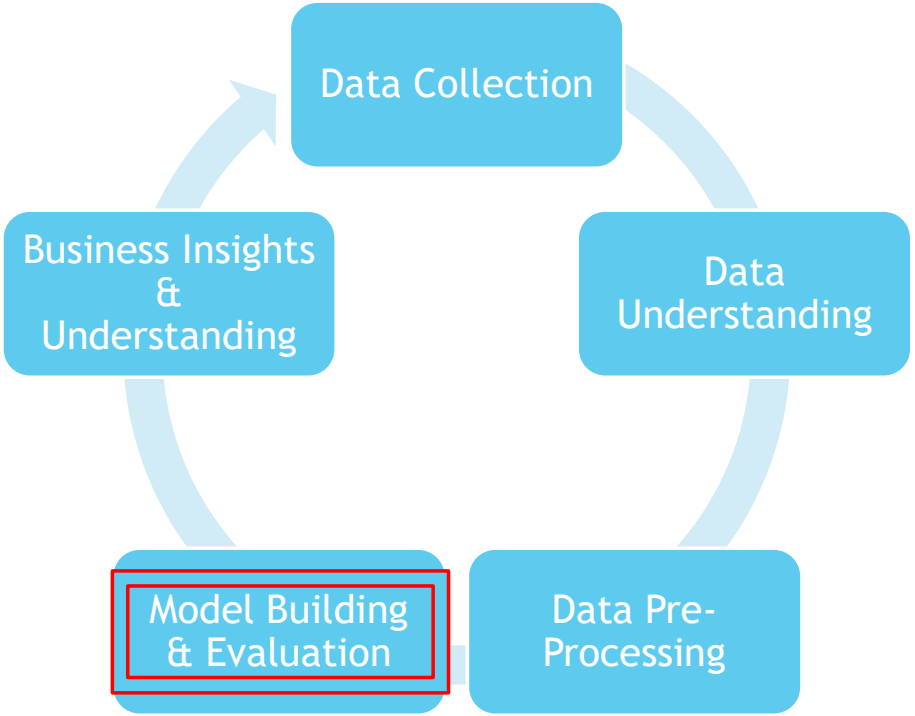
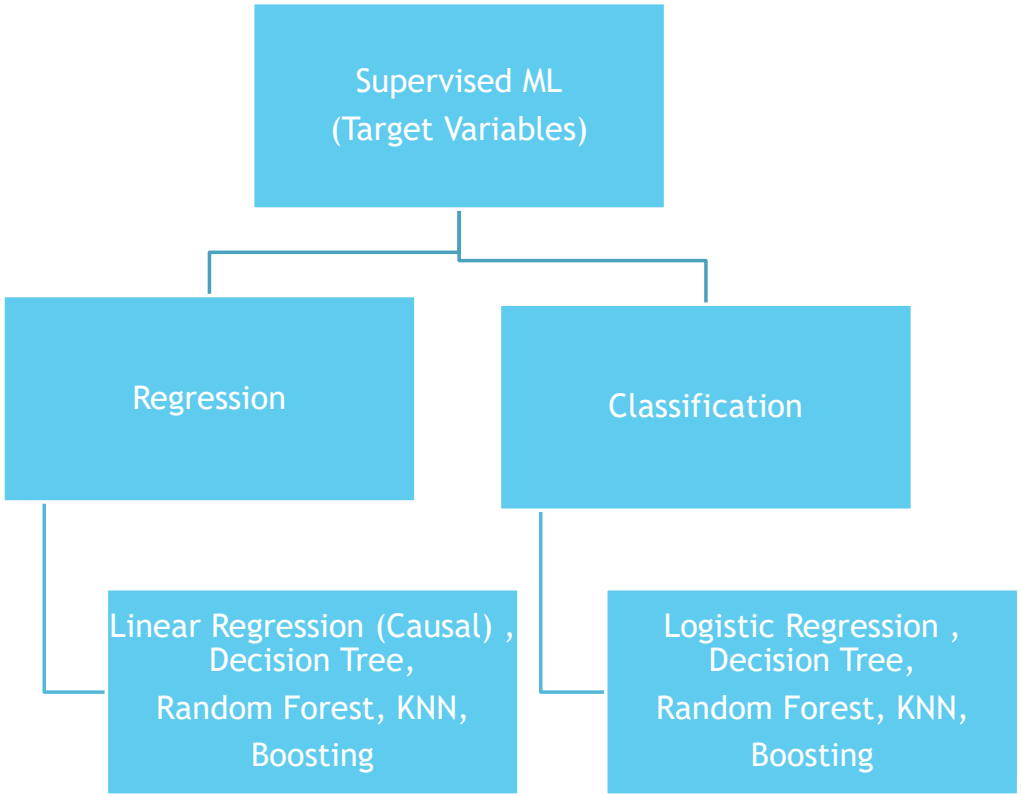
Boosting

DataDNA

Typical Data Science Cycle

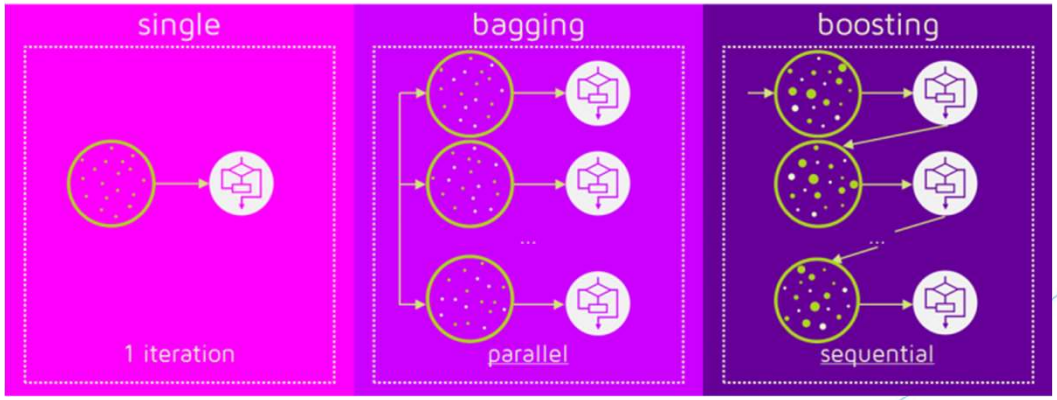


Machine Learning



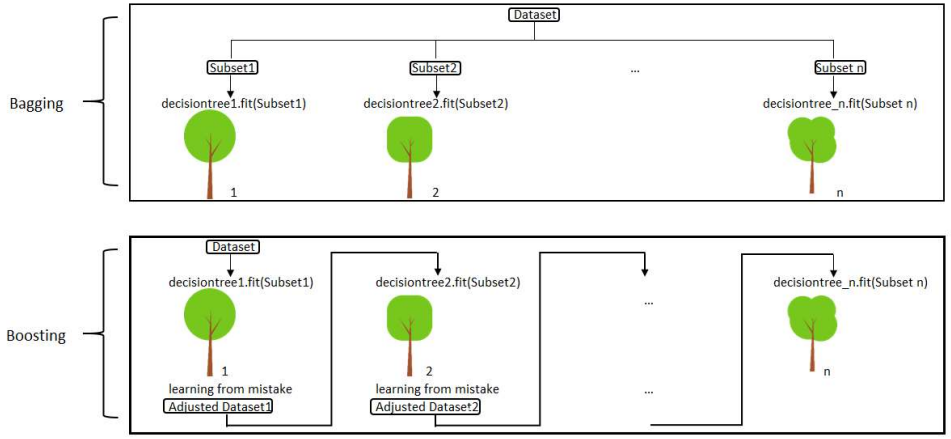
Ensemble Methods

- Ensemble methods are machine learning techniques that combine several base models in order to produce one optimal predictive model.
- They are more useful and proficient when the model is unstable(flexible) resulting in high variance between different sample sets taken out of the population. The basic intuition behind this is to expose the model to maximum variance within the given dataset itself
- Two most common methods Are bagging and boosting.



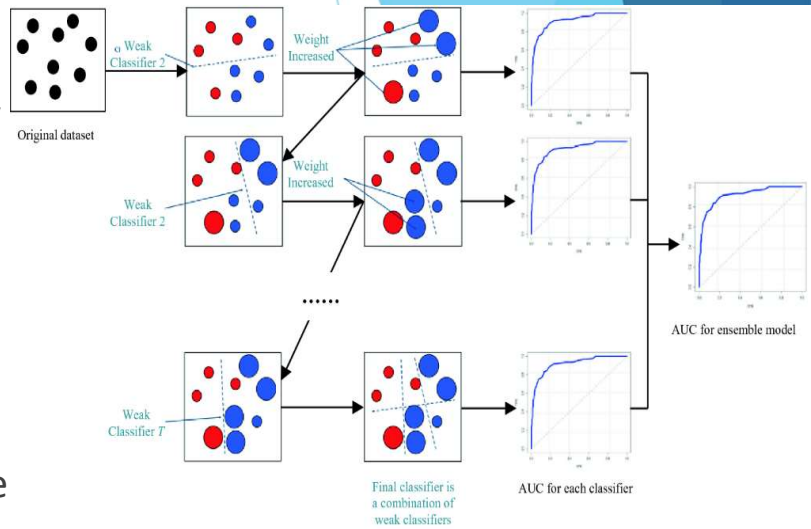
Introduction

- The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.
- Boosting is an ensemble method for improving the model predictions of any given learning algorithm.
- The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor.



How Boosting Works

1. The base learner applies equal weights for all the observations.
2. If there is prediction errors, those observations are given higher priority & then the next base learning algorithm is applied upon
3. Run Step 2 until the highest accuracy is reached upon
4. Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model.
5. Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.
6. This algorithm is additive and sequential (since they add more and more learner sequentially)



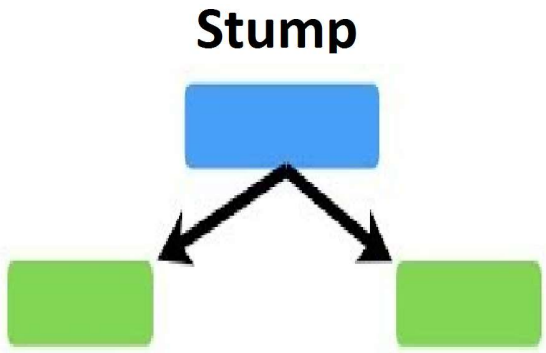
Boosting Type

Mostly 3 boosting algorithms are available :

- AdaBoost : Adaptive Boosting
- Gradient Boosting
- XGBoosting : eXtreme Gradient Boosting

ADABOOST

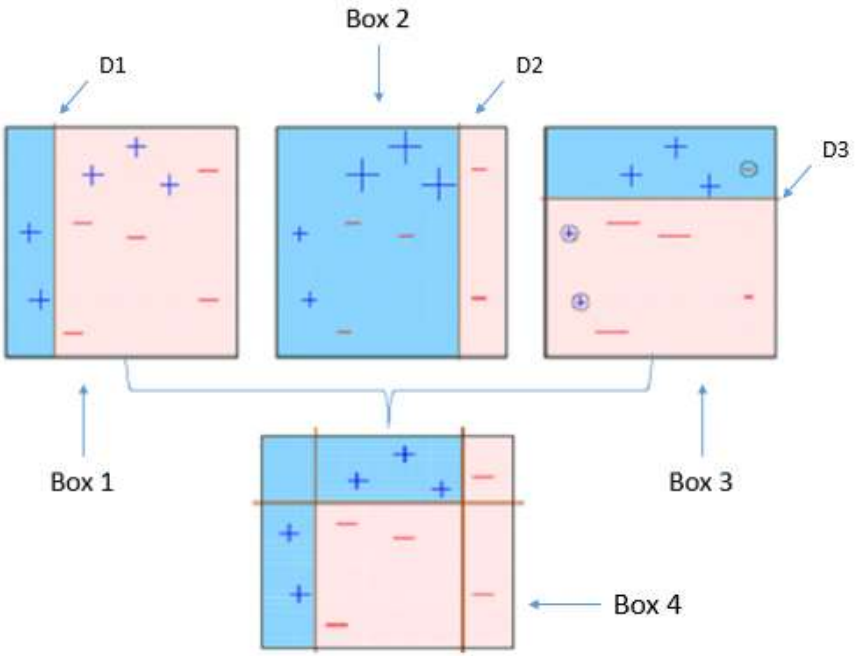
- This method focuses on training upon misclassified observations
- The weak learners incase of adaptive boosting are a very basic form of decision tree known as stumps.
- A decision stump is a machine learning model consisting of a one-level decision tree.
- It is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes (its leaves).
- A decision stump makes a prediction based on the value of just a single input feature.



DataDNA

ADABOOST

- D1 & D2 & D3 form a SEQUENCE of Weak Learners
- The weightage of the observations will be altered according to the ‘Accuracy’ of predictions using the learners.
- Each learner also has different weights assigned to it based on the classifier’s performance
- It continues to add learner until a limit is reached in the number of models or accuracy.
- The final prediction is based on a majority vote of the weak learners’ predictions weighted by their individual accuracy.
- Adaboost increases the predictive accuracy by assigning weights to both observations at end of every tree and weights(scores) to every classifier



DataDNA

How AdaBoost Works

The importance of classifier is given by-

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$$

Where α_i ranges from (-∞) if ϵ_i is 1 and (+∞) if ϵ_i is 0.

the error rate of a classifier is given by

$$\epsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j\, I(C_i(x_j) \neq y_j)$$

Where wj is weight assigned to each observation of training set Di [Initially, for the first classifier, it is equally distributed]. I is Identity function to filter out wrong predictions made by the classifier.

The mechanism of updating weights to observation is-

$$w_i^{j+1} = \frac{w_i^j}{z_j} * \begin{cases} exp^{-\alpha_i}\ if\ C_i(x_j) = y_j \\ exp^{\alpha_i}\ if\ C_i(x_j) \neq y_j \end{cases}$$

zj is the normalization factor that is used to reassign weights of observation such that all weights sum up to 1.

How AdaBoost Works

PREDICTED	1	1	-1	-1
ACTUAL	-1	1	-1	1
WEIGHTS	0.5	0.2	0.1	0.04
CORRECT/INCORRECT	1	0	0	1
MISCLASSIFICATION RATE	1*0.5	0*0.2	0*0.1	1*0.04

Error Rate of classifier = Sum of Misclassification Rates/(Sum of weights)
= 0.54/0.84 = 0.642

$\alpha = 1/2 * \ln(1- \text{error} / \text{error}) = -0.22$

INCORRECT

Exp(-0.22 * -1)
Exp(0.22)
=1.659

CORRECT

Exp(-0.22 * 1)
Exp(-0.22)
=0.6025

DataDNA

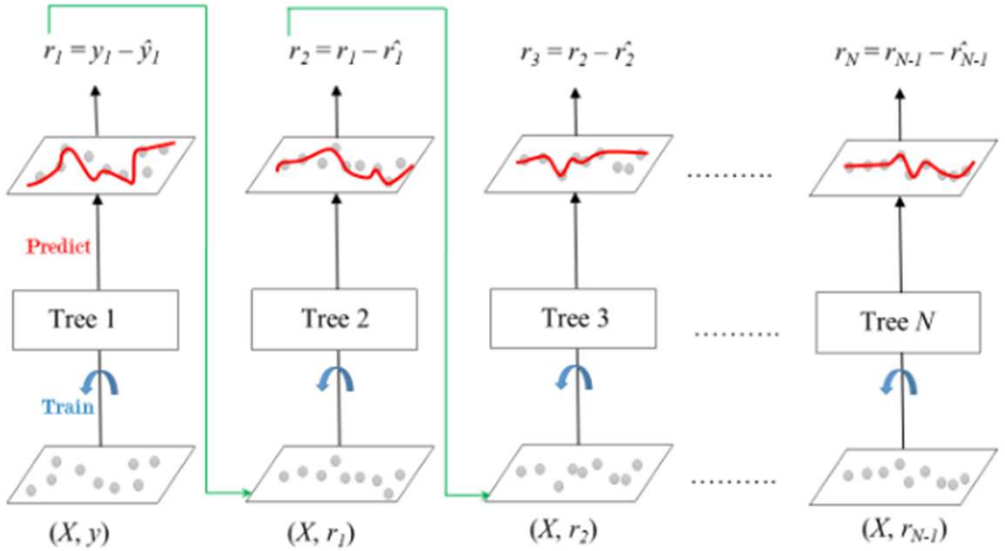
Adaboost Hyperparameters

- **base_estimators**: specifies the base type estimator, i.e. the algorithm to be used as base learner.
- **n_estimators**: It defines the number of base estimators, where the default is 10 but you can increase it in order to obtain a better performance.
- **learning_rate** : same impact as in gradient descent algorithm
- **random_state** : makes the model's output replicable. It will always produce the same results when you give it a fixed value as well as the same parameters and training data.

Gradient Descent Boosting

- Also called gradient boosting machine (GBM)
- Compared to AdaBoost (focuses on training upon misclassified observations), GBM method focuses minimising the loss function by adding more and more learners sequentially.
- There are different loss functions depends upon problems such as for classification we prefer negative log likelihood and for regression we use absolute loss etc
- All the learners have equal weights in the case of gradient boosting.
- However misclassified examples are given importance by new learners

Gradient Descent Boosting



y(pred) = y_pred(1)+ (eta * r2) + (eta * r3) + ... + (eta * rN)

learning rate (eta)

Gradient Descent Boosting

- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- A new model is created using the errors calculated as target variable. Our objective is to find the best split to minimise the error.
- By training on the residuals of the model, this is an alternative means to give more importance to misclassified observations.

Gradient Descent Boosting

- The predictions made by this new model are combined with the predictions of the previous.
- New errors are calculated using this predicted value and actual value.
- This process is repeated until the error function does not change. This is done by identifying negative gradient and moving in the opposite direction to reduce the loss. hence it is called Gradient Boosting in line with Gradient Descent where similar logic is employed
- Residuals gets reduced as we move to next learner.

$$y(pred) = y\_pred(1) + (eta * r2) + (eta * r3) + ... + (eta * rN)$$

learning rate (eta)

DataDNA

Gradient Boosting Hyper parameters

- **min_samples_split:** Minimum number of observation which is required in a node to be considered for splitting. It is used to control overfitting.
- **min_samples_leaf :** Minimum samples required in a terminal or leaf node. Lower values should be chosen for imbalanced class problems since the regions in which the minority class will be in the majority will be very small.
- **learning_rate:** Learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **n_estimators :**The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- **max_depth :** maximum depth of a tree. Used to control overfitting.
- **max_leaf_nodes :** maximum number of terminal leaves in a tree. If this is defined max_depth is ignored.
- **max_features :** number of features it should consider while searching for the best split.

Extreme Gradient Boosting

- XGBoost aka extreme gradient boosting aims to focus on computation speed and model efficiency. XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

- It is a perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.

- The speed of XGBoost is both in adding parallelism in the construction of individual trees, and in the efficient preparation of the input data (e.g. splitting based upon Xth percentile) to aid in the speed up in the construction of trees.

- **Parallelization**: XGBoost approaches the process of sequential tree building using parallelized implementation. It uses mechanisms such as Column Block for Parallel Learning . A block is basically a small subset of data distributed among different machines (cores)

- **Tree Pruning** : XGBoost uses ‘max_depth’ parameter as specified instead of criterion first, and starts pruning trees backward

- **Hardware Optimization:** This algorithm has been designed to make efficient use of hardware resources. It uses mechanisms such as CPU caches for speed.

Performance Validation

Performance validation of Boosting

Customer	City	NoOfGamesPlayed	Channel	FavoriteGame	Actual	
1059	1	36	Favorite	Uniform	1	Training Data
1060	1	298	Favorite	Uniform	0	
1061	1	27	Uniform	Uniform	1	
1062	1	156	Favorite	Uniform	0	
1063	1	217	Favorite	Uniform	0	
1064	1	51	Favorite	Uniform	0	
1065	1	30	Favorite	Uniform	1	
1066	2	74	Favorite	Uniform	0	
1067	2	37	Favorite	Uniform	0	
1068	2	110	Favorite	Uniform	0	
1069	2	140	Favorite	Uniform	0	
1070	2	60	Favorite	Uniform	1	
1071	2	75	Favorite	Uniform	0	Test Data
1072	2	116	Favorite	Uniform	0	
1073	2	171	Favorite	Uniform	1	
1074	2	67	Favorite	Uniform	0	
1075	2	16	Favorite	Uniform	1	
1076	2	121	Uniform	Uniform	0	
1077	2	78	Favorite	Uniform	1	

Modelling on Training Data

ML Model

Apply on Test Data

Prediction

Customer	City	NoOfGamesPlayed	Channel	FavoriteGame	Actual	Prediction	
1071	2	75	Favorite	Uniform	0	1	Test Data
1072	2	116	Favorite	Uniform	0	1	
1073	2	171	Favorite	Uniform	1	0	
1074	2	67	Favorite	Uniform	0	1	
1075	2	16	Favorite	Uniform	1	1	
1076	2	121	Uniform	Uniform	0	0	
1077	2	78	Favorite	Uniform	1	1	

Model Evaluation Metrics

The various metrics used to evaluate the results of the prediction are :

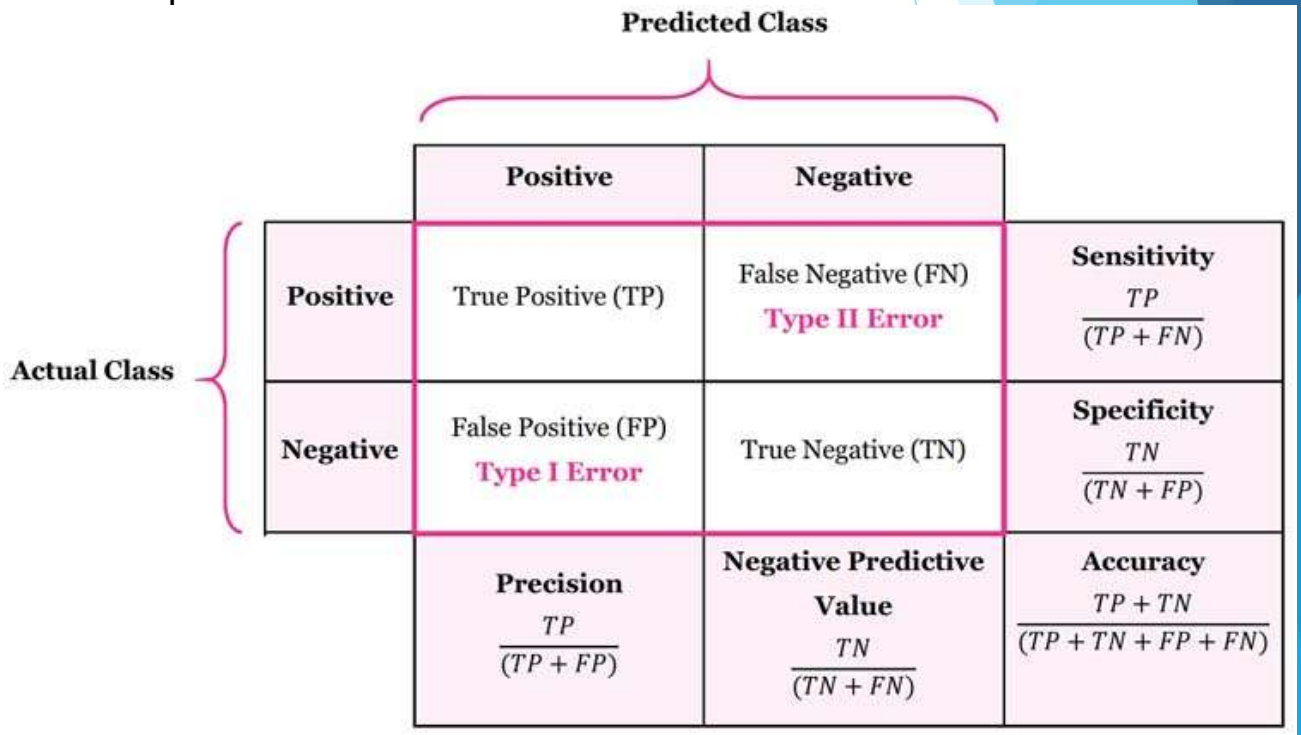
1.Accuracy

2.Precision

3.Recall

4.Area Under ROC Curve

5.MAPE/ MAD/ RMSE (For Regressors)



		Predicted Class: Positive	Predicted Class: 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** $\frac{TP+TN}{(TP+TN+FP+FN)}$