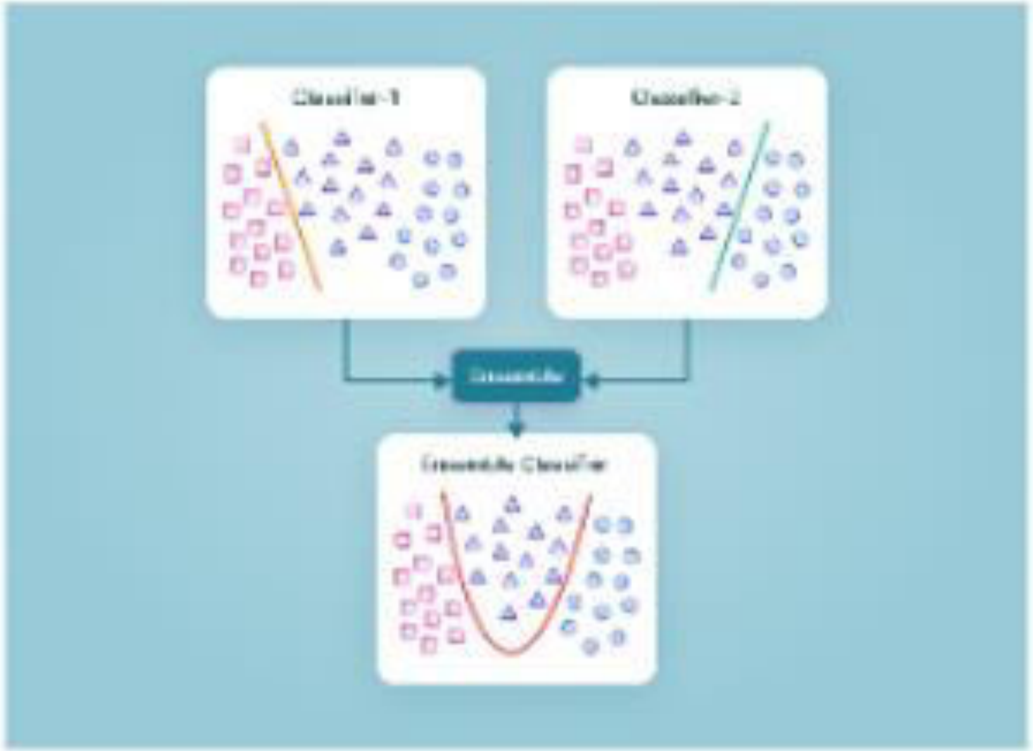
Ensemble learning:-

Ensemble learning is a machine learning technique that combines multiple individual models (learners) to create a more powerful and accurate model.

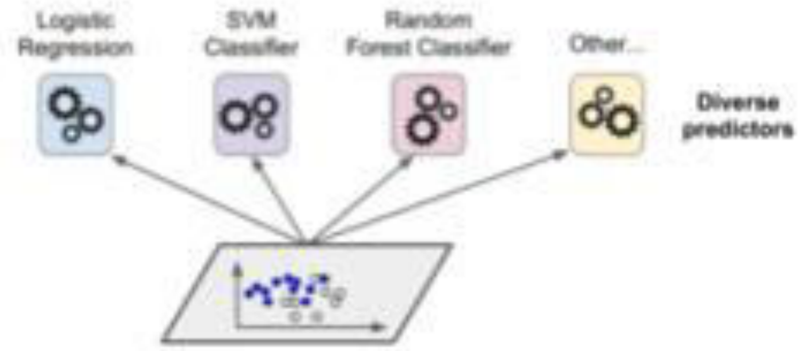


Methods for Ensemble Learning:-

1. Voting
2. Bagging (Bootstrap Aggregating)
3. Boosting
4. Stacking

Voting ensemble learning

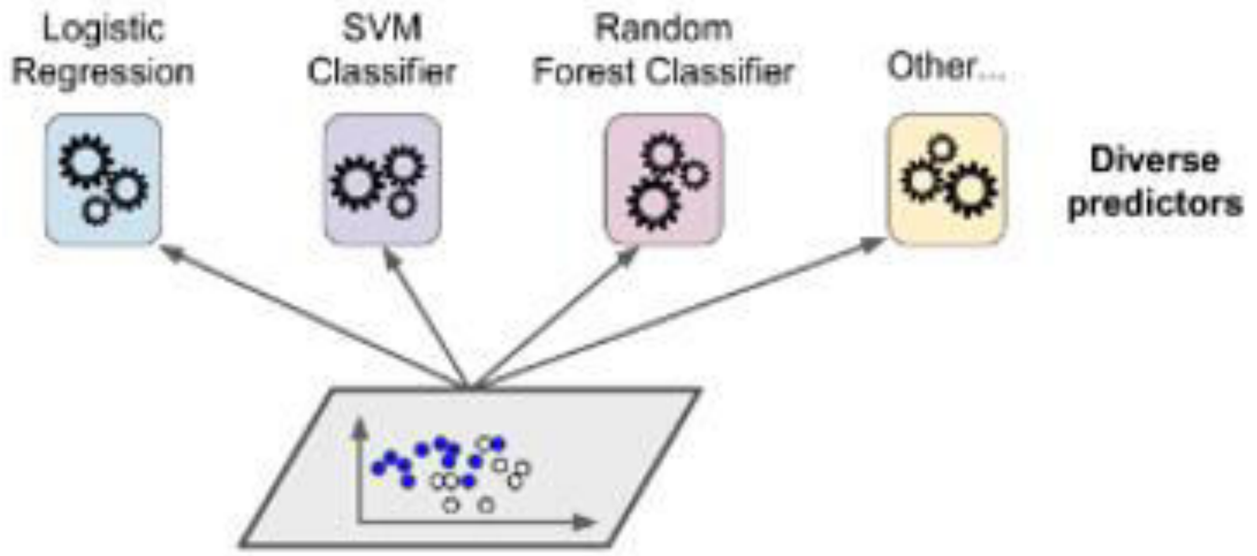
- **The Voting Ensemble method** - combines the predictions of multiple individual models to make a final prediction.
- The Voting Ensemble method involves aggregating the predictions through a simple voting mechanism.



Voting Ensemble Learning

Types of Voting:-

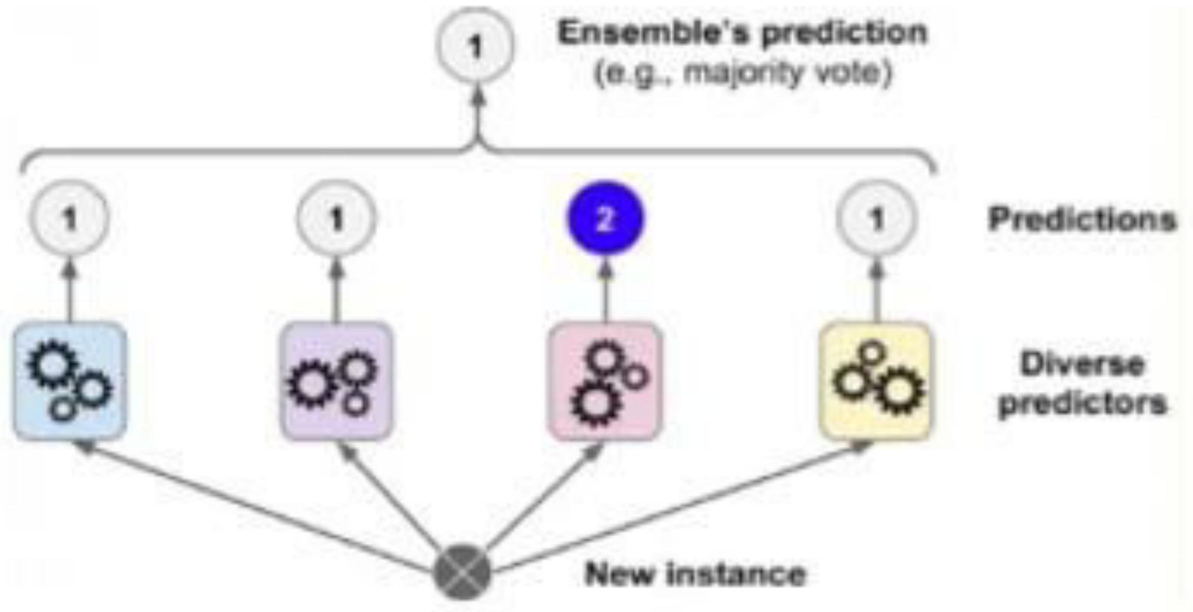
Hard Voting:



- For classification, the class label with the most votes is chosen, and for regression, the average or median of the predicted values is taken.

Hard Voting:

- In hard voting, each base model in the ensemble predicts the class label (for classification) or the value (for regression).
- The final prediction is determined by majority voting.



Hard Voting:

Types of Voting:-

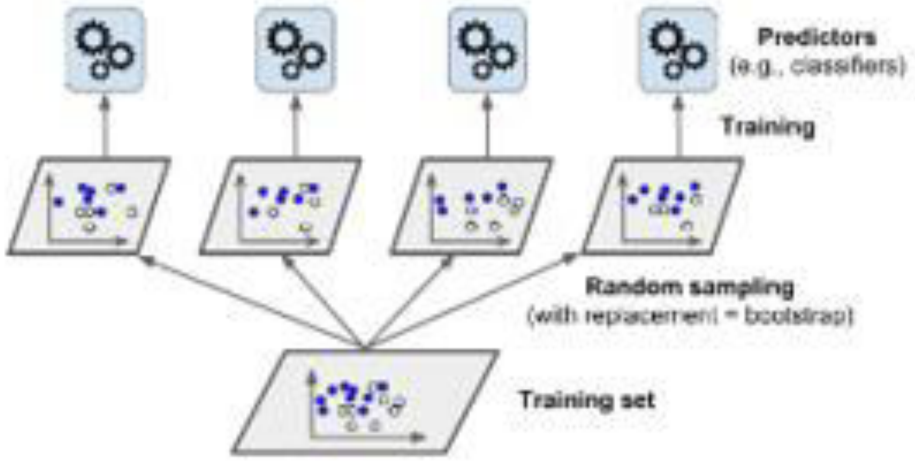
Soft Voting:-

- If all classifiers are able to estimate class probabilities (i.e., they have a **predict_proba() method**), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual classifiers.
- It often achieves higher performance than hard voting because it gives more weight to highly confident votes.

- *Ensemble methods work best when the predictors are as independent from one another as possible.*

Bagging (Bootstrap Aggregating):-

- In **bagging,** multiple copies of the **same base model** are trained on **different subsets** of the training data.
- **Each model** is trained **independently,** and the **final prediction** (averaging (for regression) or voting (for classification))
- **Eg: Random Forest and k-NN**



. When sampling is perfomed with replacement, this method is called bagging.

Bias:-

Bias is the **difference** between the **average prediction** of our model and the **correct value** which we are trying to predict.

Being high in biasing gives a large error in training as well as testing data. It recommended that an algorithm should always be low-biased to avoid the problem of underfitting.

Variance:-

Variance is the **variability** of **model prediction** for a **given data point** or a value which tells us **spread** of our data.

More specifically, variance is the variability of the model that how much it is sensitive to another subset of the training dataset.

Mathematical relation between Bias and Variance:-

Let the variable we are trying to predict as Y and other covariates as X. We assume there is a relationship between the two such that

$$Y=f(X) + e$$

Where e is the error term and it's normally distributed with a mean of 0.

We will make a model $\hat{f}(x)$ of f(X) using linear regression or any other modeling technique.

So the expected squared error at a point x is

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

The Err(x) can be further decomposed as

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$\boxed{Err(x) = \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}}$$

The proof the above equation can be seen in next slide.
Irreducible error is the error that can't be reduced by creating good models. It is a measure of the amount of noise in our data. Here it is important to understand that no matter how good we make our model, our data will have certain amount of noise or irreducible error that can not be removed.

$$\underset{\text{output/Label}}{y} = f(\underset{\text{Input}}{x}) + \underset{\text{error}}{e} \rightarrow e \sim N(0,1), \quad \hat{f}(x) = \hat{y}$$

$$E(y-\hat{f}(x))^2 = E((\underbrace{y-f(x)}_{a} + \underbrace{f(x)-\hat{f}(x)}_{b})^2$$

$$= E(a^2) + E(b^2) + 2E(ab)$$

$$= \underbrace{E(e^2)}_{\sigma_e^2} + \quad E(e(f(x)-\hat{f}(x)) \rightarrow E(e f(x)) - E(e\hat{f}(x))$$

$$\underbrace{E(e f(x))}_{f(x)E(e)} \quad \text{Orthogonality} \rightarrow 0 \quad \text{princip}$$

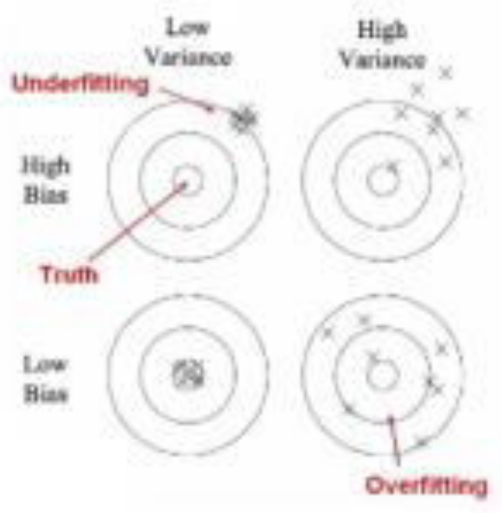
$$E((\overbrace{f(x)-\hat{f}(x)}^{b^2})^2) = E((\overbrace{f(x)-E\hat{f}}^{\text{bias}})^2)$$

$$+ \underbrace{E(E\hat{f}-\hat{f}(x))^2}_{} \leftarrow \text{Vari} \ \text{nc}$$

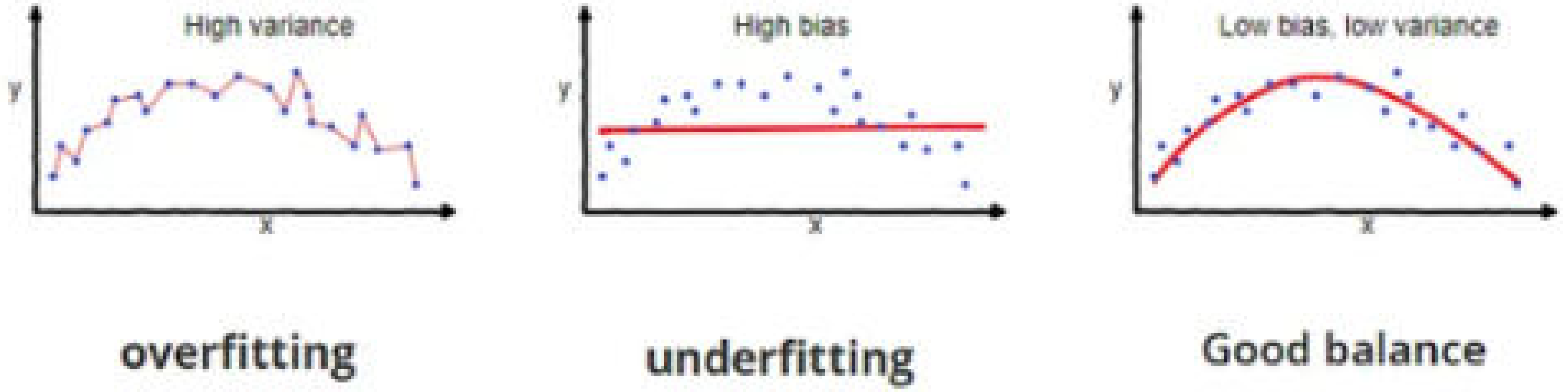
$$+2\underbrace{E((d)}_{=0}$$

$$(\overbrace{f(x)-E\hat{f}(x)}^{c} + \underbrace{E\hat{f}(x)-\hat{f}(x)}_{d})$$

Bias and variance using bulls-eye diagram

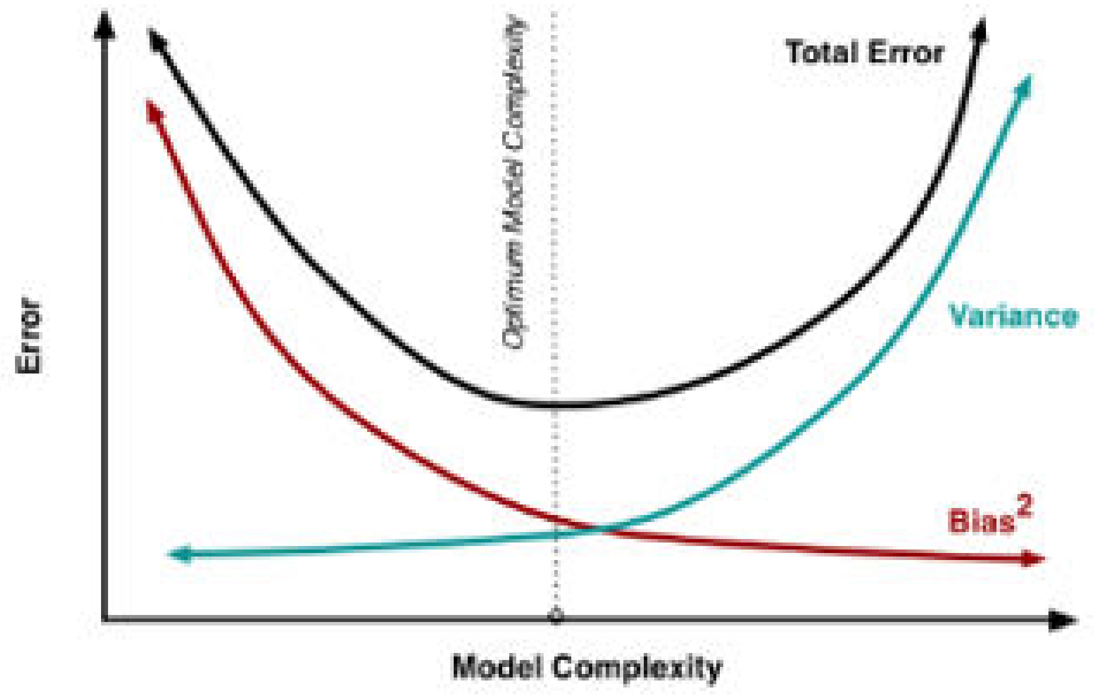


center of the target is a model that perfectly predicts correct values. As we move away from the bullseye our predictions become get worse and worse. We can repeat our process of model building to get separate hits on the target.

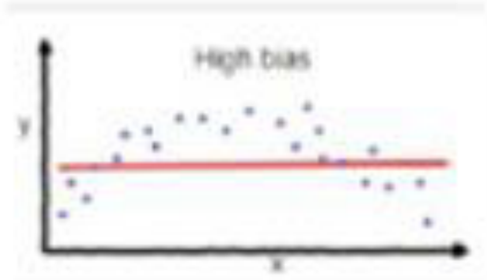
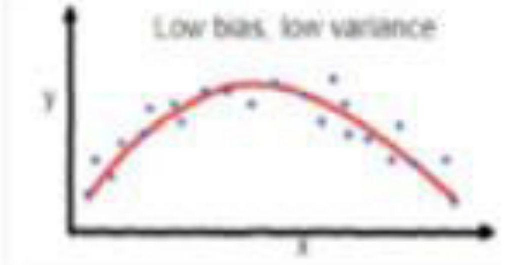
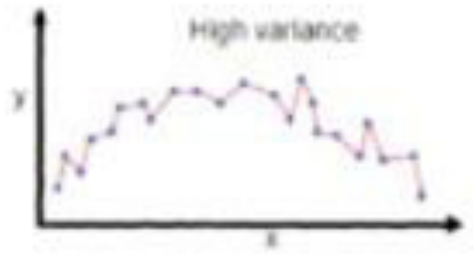
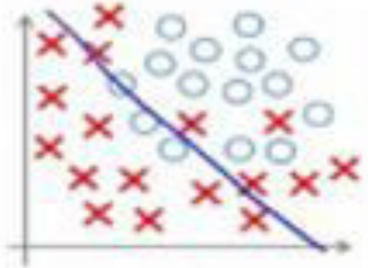
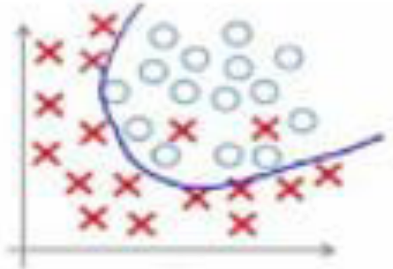
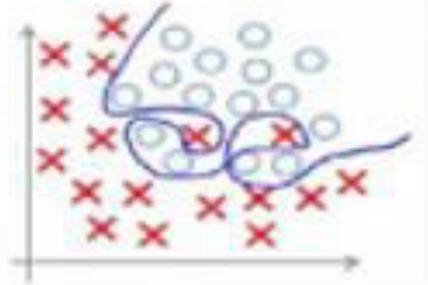


1. In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.
2. In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.

Bias Variance Tradeoff:-



If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

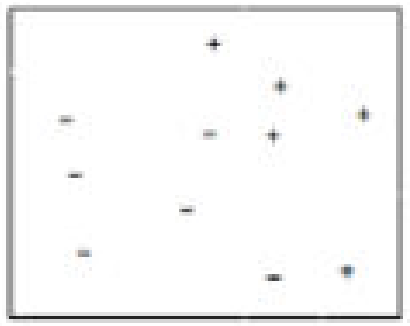
	Underfitting	Optimal-fitting	Overfitting
Symptoms	1. High bias 2. High training error 3. Training error close to testing error 4. Unable to capture the relationship between training data and labels 5. inaccurate(not valid)	1. Low-bias, low variance 2. Optimize training error 3. Training error slightly lower than test error 4. Capture well the relationship between training data and labels 5. accurate(valid) and consistency	1. High variance 2. Very low training error 3. Training error much lower than test error 4. Customize too much the relationship between training data and labels 5. Inconsistency (not reliable)
Regression			
Classification			

	Underfitting	Optimal-fitting	Overfitting
Remedies	1. Increase training data 2. Reduce noise 3. Perform feature scaling (normalization or standardization) 4. Train longer 5. Add more features 6. Increase complexity 7. Combine models using boosting ensemble 8. Increase capacity 9. Change network structure or parameters 10. Add inputs, nodes, layers		1. Use data augmentation 2. Reduce noise 3. Perform feature scaling 4. Perform regularization 5. Use cross-validation 6. Early stopping 7. Reduce features 8. Simplify complexity 9. Combine models using bagging ensemble 10. Pruning number of weights (structure) and values (parameters) 11. Dropout inputs or units

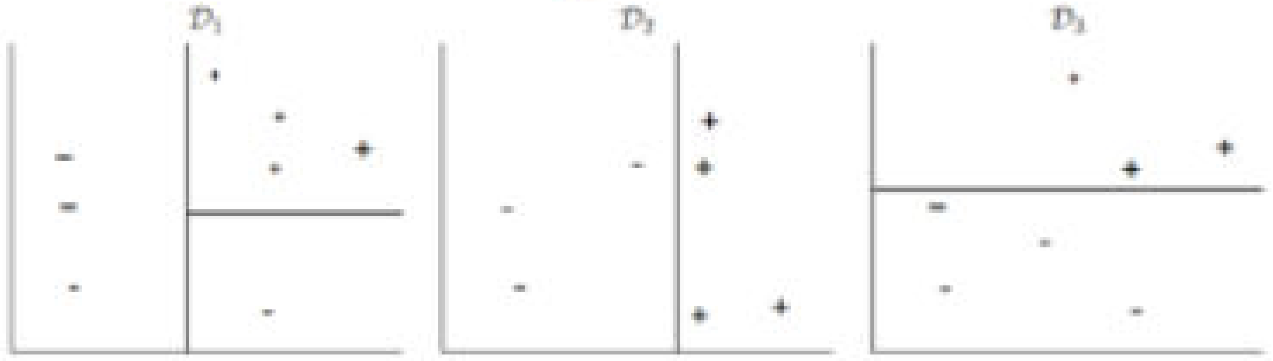
Remember that the bias-variance trade-off is not always straightforward and can depend on factors such as data complexity, the chosen hyperparameters, the quality and quantity of training data, and the presence of noise.

Machine Learning Models	**Bias**	**Variance**
Linear Regression	Low	Low
Logistic Regression	Low	Low
Decision Tree	Medium	Medium
Random Forest	**Low**	**Low**
Gradient Boosting	Low	Low-Med
Support Vector Machines	Low-High	Low-Med
K-Nearest Neighbors	Low-Med	High
Naive Bayes	Medium	Low

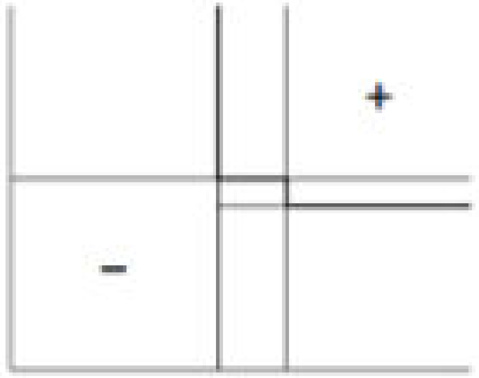
Bagging (Bootstrap Aggregating):-



Original data



3 models (from some model class) learned using three data sets chosen via bootstrapping



= Averaged model

Pasting :-

- When use the same training algorithm for every predictor, but to train them on different random subsets of the training set.
- When **sampling is performed without replacement,** it is called pasting.

Out-of-Bag Evaluation

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
- The training instances that are not sampled are called **out-of-bag (oob) instances**.

By default a BaggingClassifier samples m training instances with replacement (bootstrap=True), where m is the size of the training set. This means that only about 63% of the training instances are sampled on average for each predictor.6 The remaining 37% of the training instances that are not sampled are called out-of-bag (oob) instances.

Random Forests:-

A Random Forest is an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting), typically with **max_samples** set to the size of the training set.

The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node

This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance, generally yielding an overall better model.

Extra-Trees:-

- A forest of extremely random trees is simply called an Extremely Randomized Trees ensemble (or Extra-Trees for short).
- Once again, this trades more bias for a lower variance.

When you are growing a tree in a Random Forest, at each node only a random subset of the features is considered for splitting (as discussed earlier). It is possible to make trees even more random by also using random thresholds for each feature rather than searching for the best possible thresholds (like regular Decision Trees do)

It is hard to tell in advance whether a RandomForestClassifier will perform better or worse than an ExtraTreesClassifier. Generally, the only way to know is to try both and compare them using cross-validation (and tuning the hyperparameters using grid search).

Feature Importance:-

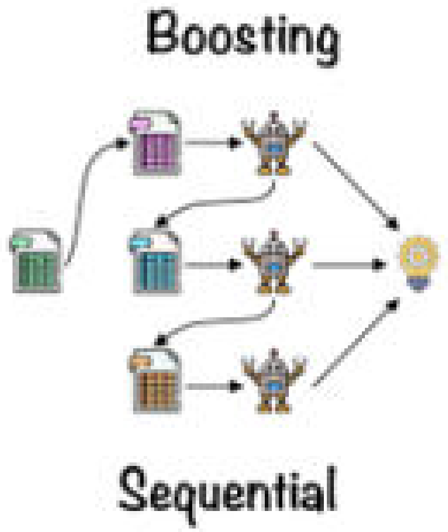
Scikit-Learn measures a feature importance

it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it

feature_importances_ variable.

Boosting:-

- In boosting, **multiple models** are trained **sequentially**, and **each model** tries to **correct the errors** of **its predecessor.**
- The **models** are **combined** with **different weights** based on **their performance.**
- **AdaBoost, Gradient Boosting Machines (GBM), and XGBoost.**



Boosting:-

IDEA:-

- The basic idea is as follows:
- Take a weak learning algorithm
    1. Only requirement: Should be only slightly better than random.
    2. Turn it into an awesome one by making it focus on difficult cases.

Boosting:-

Most boosting algorithms follow these steps:

1. Train a weak model on some training data
2. Compute the error of the model on each training example
3. Give higher importance to examples on which the model made mistakes
4. Re-train the model using "importance weighted" training examples
5. Go back to step 2

Note: Unlike bagging, boosting is a sequential algorithm (models learned in a sequence)

The idea of boosting came out of the idea of whether a weak learner can be modified to become better.

A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance.
These ideas built upon Leslie Valiant's work on distribution free or Probably Approximately Correct (PAC) learning, a framework for investigating the complexity of machine learning problems.

The idea is to use the weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified. … Note, however, it is not obvious at all how this can be done

The first realization of boosting that saw great success in application was Adaptive Boosting or AdaBoost for short.

The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns.

This means that samples that are difficult to classify receive increasing larger weights until the algorithm identifies a model that correctly classifies these samples

— Applied Predictive Modeling, 2013

Predictions are made by majority vote of the weak learners' predictions, weighted by their individual accuracy. The most successful form of the AdaBoost algorithm was for binary classification problems and was called AdaBoost.M1.

Modern boosting methods build on AdaBoost, most notably stochastic gradient boosting machines.

error = (correct – N) / N

error = sum(w(i) * terror(i)) / sum(w)

Each instance in the training dataset is weighted. The initial weight is set to:

weight(xi) = 1/n

For example, if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the instances were -1, 1 and -1, then the terrors would be 0, 1, and 0. The misclassification rate would be calculated as:

error = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2)

or

error = 0.704

A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

stage = ln((1-error) / error)

the weight of one training instance (w) is updated using:

w = w * exp(stage * terror)

AdaBoost Ensemble

Weak models are added sequentially, trained using the weighted training data.

The process continues until a pre-set number of weak learners have been created (a user parameter) or no further improvement can be made on the training dataset.

Once completed, you are left with a pool of weak learners each with a stage value.

Making Predictions with AdaBoost

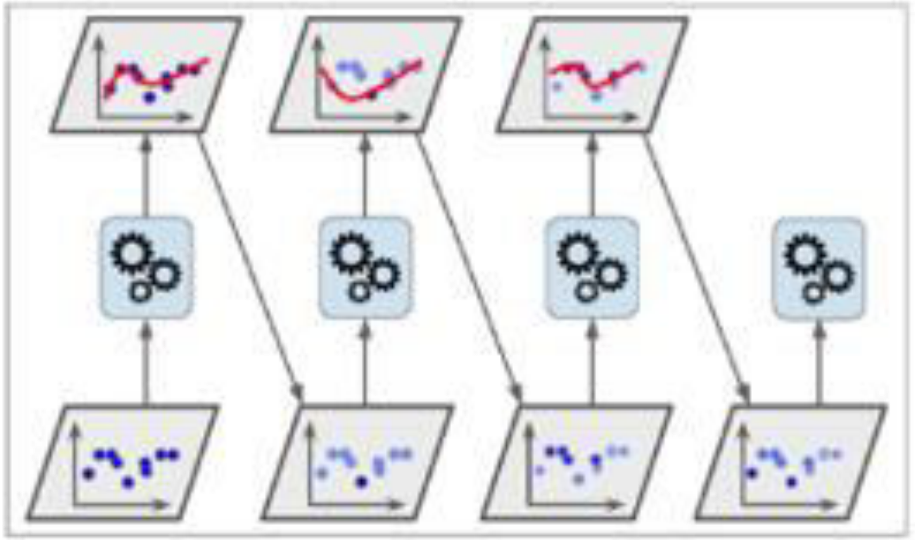
Predictions are made by calculating the weighted average of the weak classifiers.

For a new input instance, each weak learner calculates a predicted value as either +1.0 or -1.0. The predicted values are weighted by each weak learners stage value. The prediction for the ensemble model is taken as a the sum of the weighted predictions. If the sum is positive, then the first class is predicted, if negative the second class is predicted.

For example, 5 weak classifiers may predict the values 1.0, 1.0, -1.0, 1.0, -1.0. From a majority vote, it looks like the model will predict a value of 1.0 or the first class. These same 5 weak classifiers may have the stage values 0.2, 0.5, 0.8, 0.2 and 0.9 respectively. Calculating the weighted sum of these predictions results in an output of -0.8, which would be an ensemble prediction of -1.0 or the second class.

AdaBoost :-

One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor under fitted.
This results in new predictors focusing more and more on the hard cases.
This is the technique used by **AdaBoost.**



AdaBoost sequential training with instance weight updates

There is one important drawback to this sequential learning technique: it cannot be parallelized (or only partially), since each predictor can only be trained after the previous predictor has been trained and evaluated. As a result, it does not scale as well as bagging or pasting.

Each instance weight $w^{(i)}$ is initially set to $\frac{1}{m}$. A first predictor is trained and its weighted error rate r_1 is computed on the training set;

Weighted error rate of the j^{th} predictor

$$r_j = \frac{\displaystyle\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\displaystyle\sum_{i=1}^{m} w^{(i)}}$$

where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.

The predictor's weight α_j

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$ where η is the learning rate hyperparameter

Next the instance weights are updated. The misclassified instances are boosted.

Weight update rule

$$\text{for } i = 1, 2, \cdots, m$$
$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$

Then all the instance weights are normalized (i.e., divided by $\sum_{i=1}^{m} w^{(i)}$).

The more accurate the predictor is, the higher its weight will be. If it is just guessing randomly, then its weight will be close to zero. Finally, a new predictor is trained using the updated weights, and the whole process is repeated (the new predictor's weight is computed, the instance weights are updated, then another predictor is trained, and so on). The algorithm stops when the desired number of predictors is reached, or when a perfect predictor is found.

To make predictions,

AdaBoost predictions

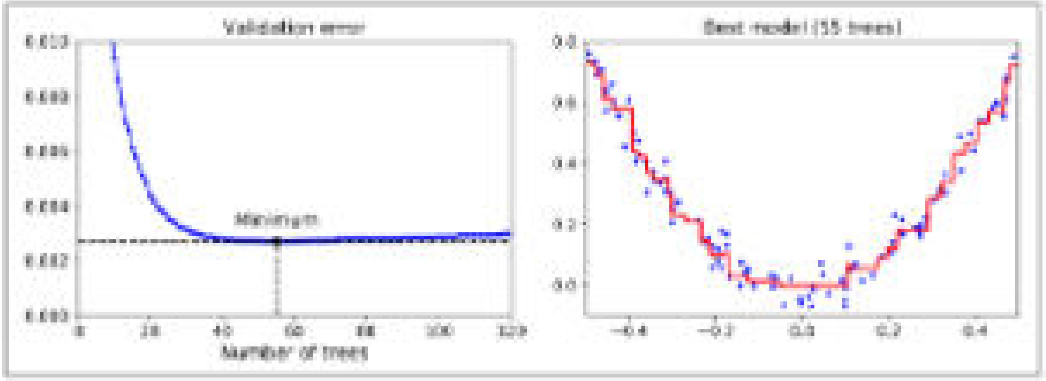
$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x}) = k}}^{N} \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$

Scikit-Learn actually uses a multiclass version of AdaBoost called SAMME16 (which stands for Stagewise Additive Modeling using a Multiclass Exponential loss function).

If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator

Early stopping :-

- To find the optimal number of trees.
- **Staged_predict()** method: it returns an iterator over the predictions made by the ensemble at each stage of training (with one tree, two trees, etc.).

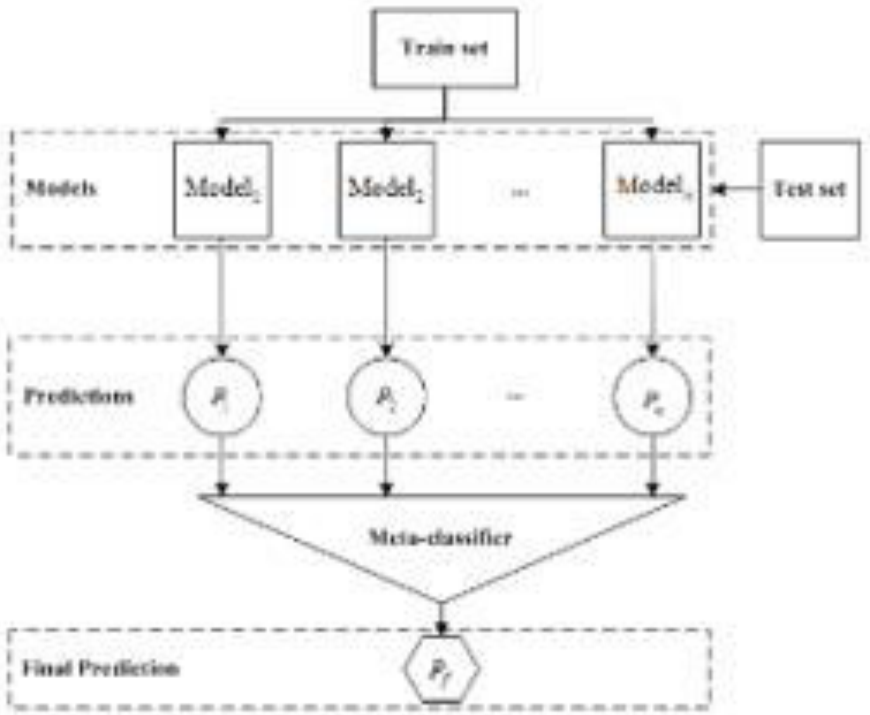


Tuning the number of trees using early stopping

It is also possible to implement early stopping by actually stopping training early (instead of training a large number of trees first and then looking back to find the optimal number).

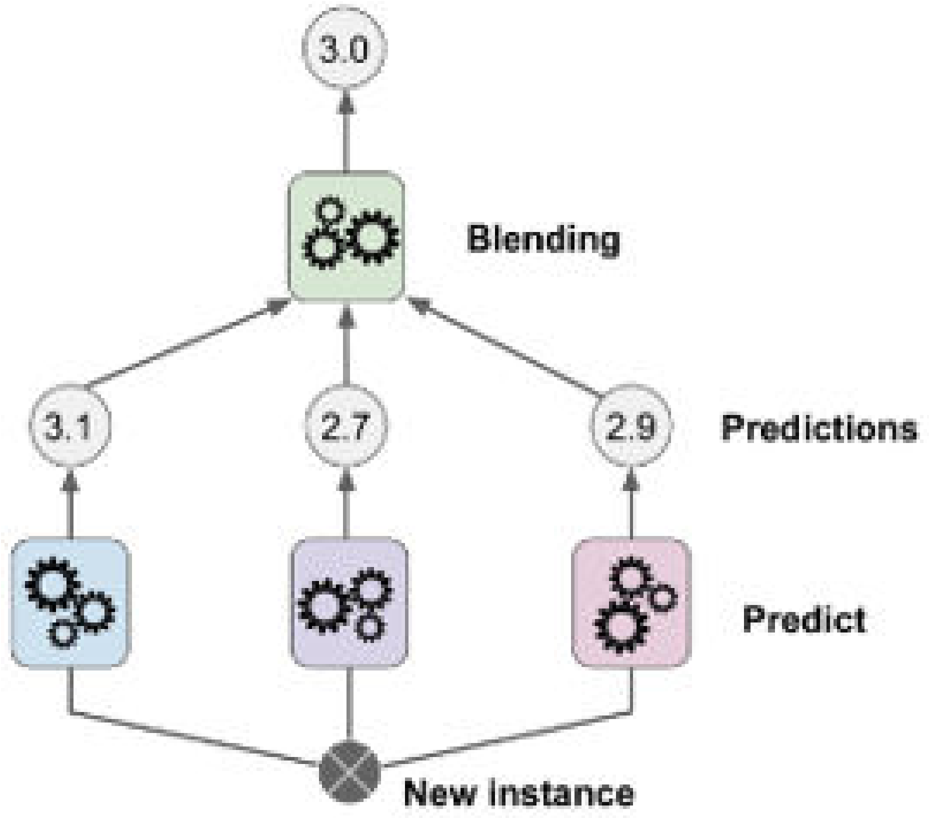
Stacking:-

- **Stacking** in **machine learning** is also known as **Stacking Generalisation**, which is a technique where **all models aggregated are utilized according to their weights for producing an output which is a new model**. As a result, this model has better accuracy and is stacked with other models to be used.
- It as a **two-layer model** where the first layer incorporates all the models, and the second one is the prediction layer which renders output.



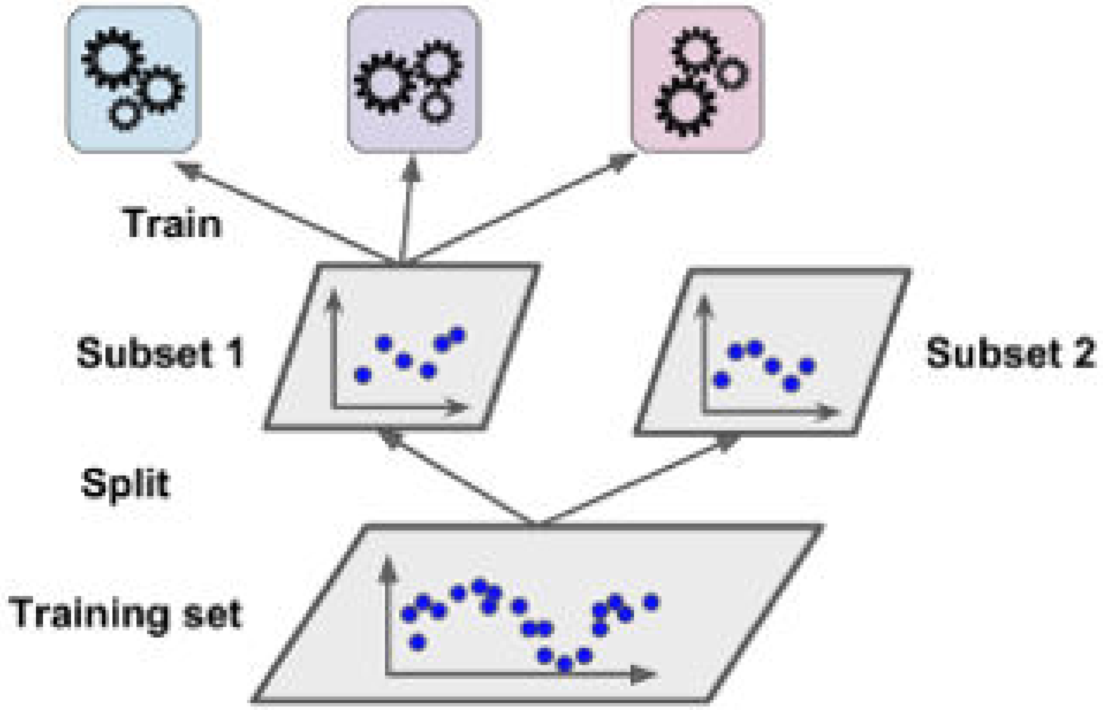
Stacking Method

The principle is that you can always tackle a learning issue with various models that can learn a subset of the problem but not the whole problem space. This is where Stacking is used.



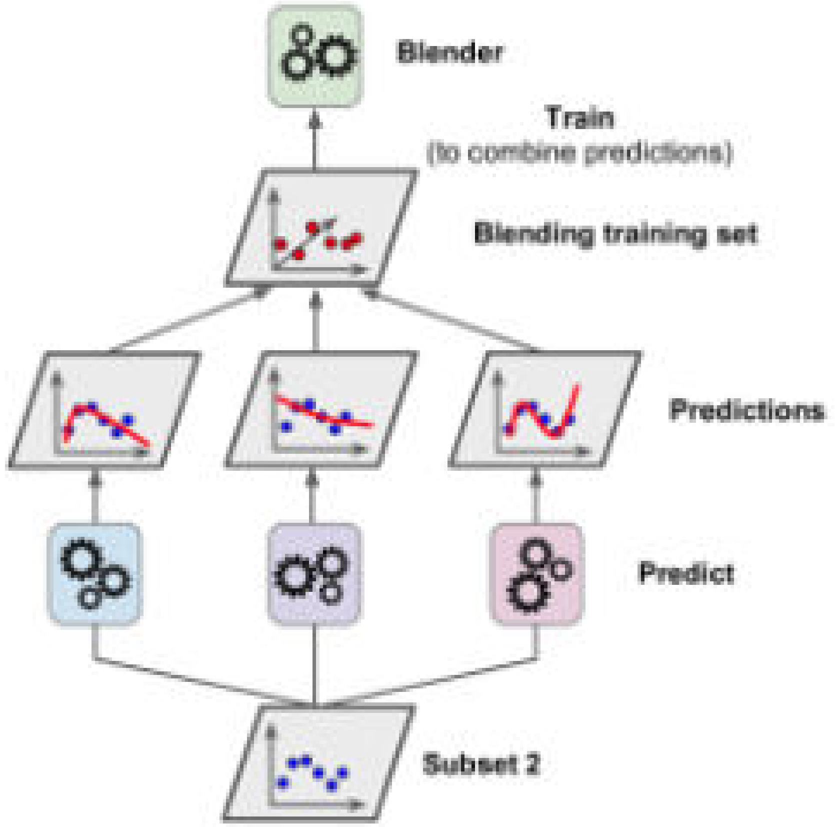
Aggregating predictions using a blending predictor

ensemble performing a regression task on a new instance. Each of the bottom three predictors predicts a different value (3.1, 2.7, and 2.9), and then the final predictor (called a blender, or a meta learner) takes these predictions as inputs and makes the final prediction (3.0).



Training the first layer

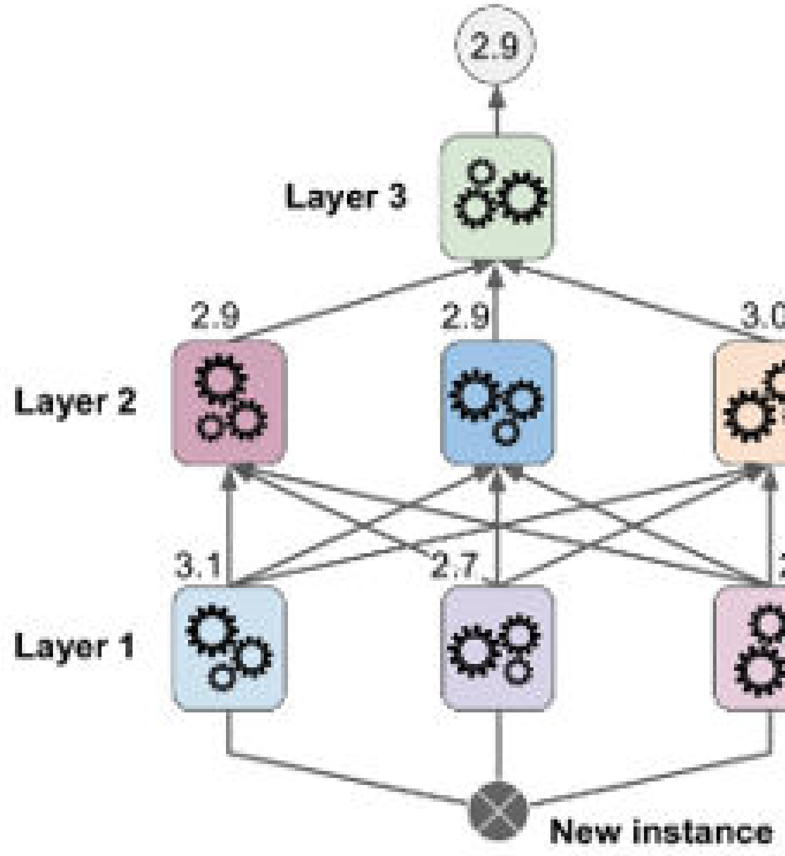
To train the blender, a common approach is to use a hold-out set.First, the training set is split in two subsets. The first subset is used to train the predictors in the first layer.Next, the first layer predictors are used to make predictions on the second (held-out) set. This ensures that the predictions are "clean," since the predictors never saw these instances during training. Now for each instance in the hold-out set.there are three predicted values. We can create a new training set using these predic- ted values as input features (which makes this new training set three-dimensional), and keeping the target values.



Training the blender

The blender is trained on this new training set, so it learns to predict the target value given the first layer's predictions.It is actually possible to train several different blenders this way (e.g., one using Linear Regression, another using Random Forest Regression, and so on): we get a whole layer of blenders.

Predictions in a multilayer stacking ensemble:-



The trick is to split the training set into three subsets: the first one is used to train the first layer, the second one is used to create the training set used to train the second layer (using predictions made by the predictors of the first layer), and the third one is used to create the training set to train the third layer (using pre- dictions made by the predictors of the second layer). Once this is done, we can make a prediction for a new instance by going through each layer sequentially.

Isolation Forest :-

Isolation Forest is an algorithm for data anomaly detection initially developed by Fei Tony Liu in 2008.

Isolation Forest detects anomalies using binary trees.

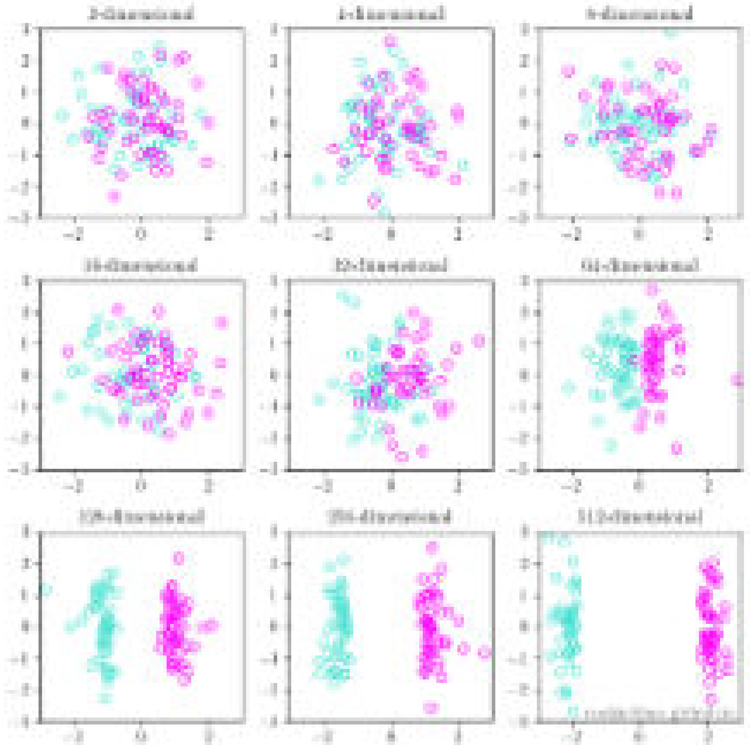
The algorithm is different from decision tree algorithms in that only the path-length measure or approximation is being used to generate the anomaly score, no leaf node statistics on class distribution or target value is needed.

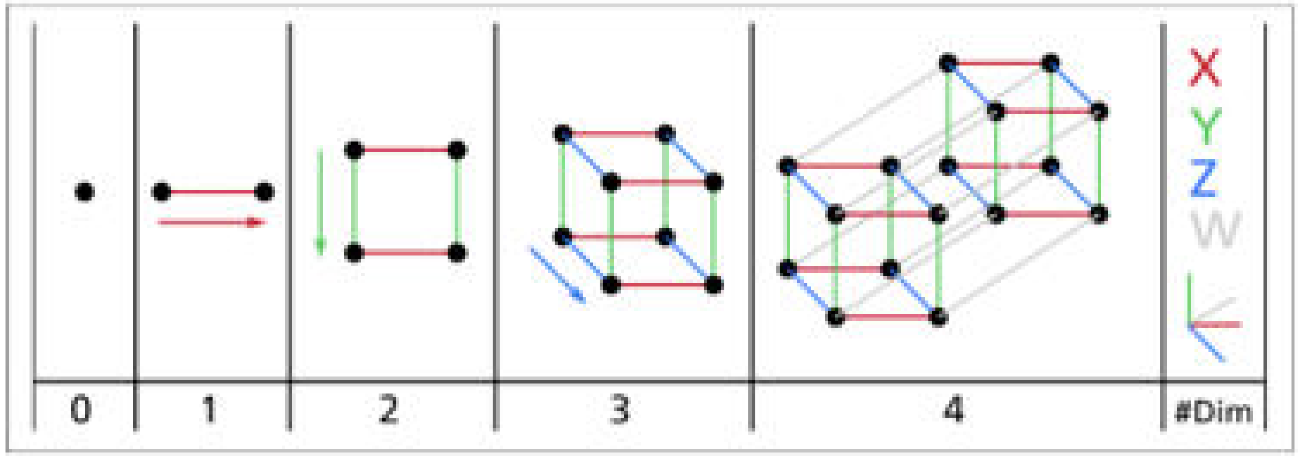
The algorithm has a linear time complexity and a low memory requirement, which works well with high-volume data.

Isolation Forest is fast because it splits the data space randomly, using randomly selected attribute and randomly selected split point. The anomaly score is invertedly associated with the path-length as anomalies need fewer splits to be isolated, due to the fact that they are few and different.

Dimensionality Reduction

Curse of Dimensionality-



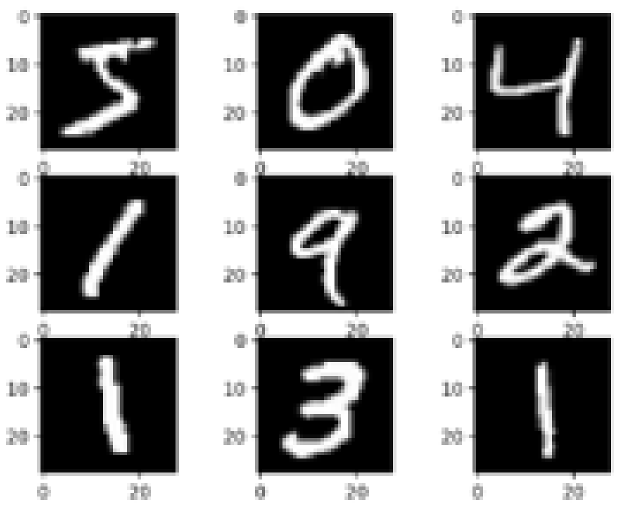


Point, segment, square, cube, and tesseract (0D to 4D hypercubes)

Many Machine Learning problems involve thousands or even millions of features for each training instance. Not only does this make training extremely slow, it can also make it much harder to find a good solution.

In real-world problems of dimensionality:-

The MNIST images:- the pixels on the image borders are almost always white.



you could completely drop these pixels from the training set without losing much information. these pixels are utterly unimportant for the classification task. Moreover, two neighboring pixels are pixels are highly correlated: if you merge them into a single pixel (e.g., by taking the mean of the two pixel intensities), you will not lose much information.

Effect of Reducing dimensionality :-

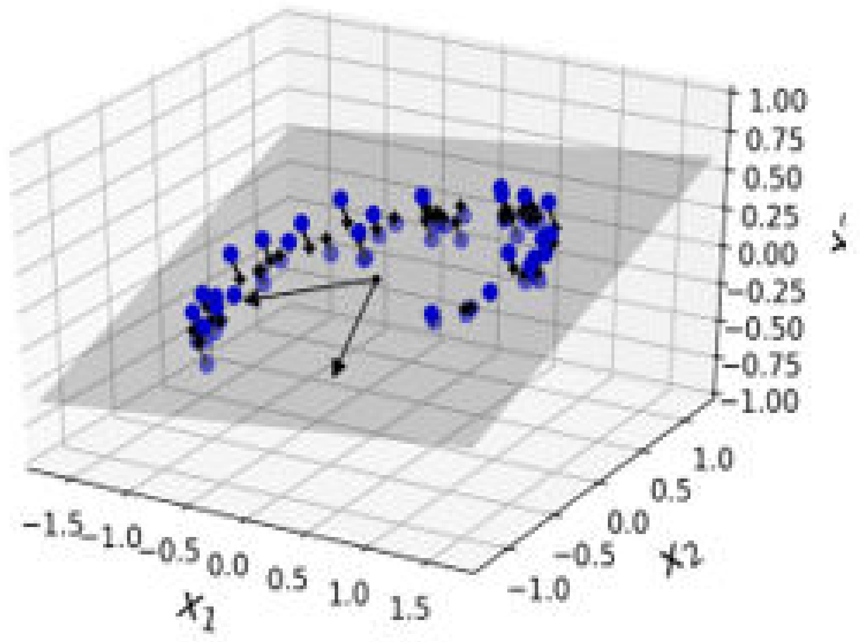
1. Lose some information
2. Speed up training
3. Makes your pipelines a bit more complex and thus harder to maintain.
4. Filter out some noise and unnecessary details
5. Useful for data visualization

<u>Main Approaches for Dimensionality Reduction:-</u>

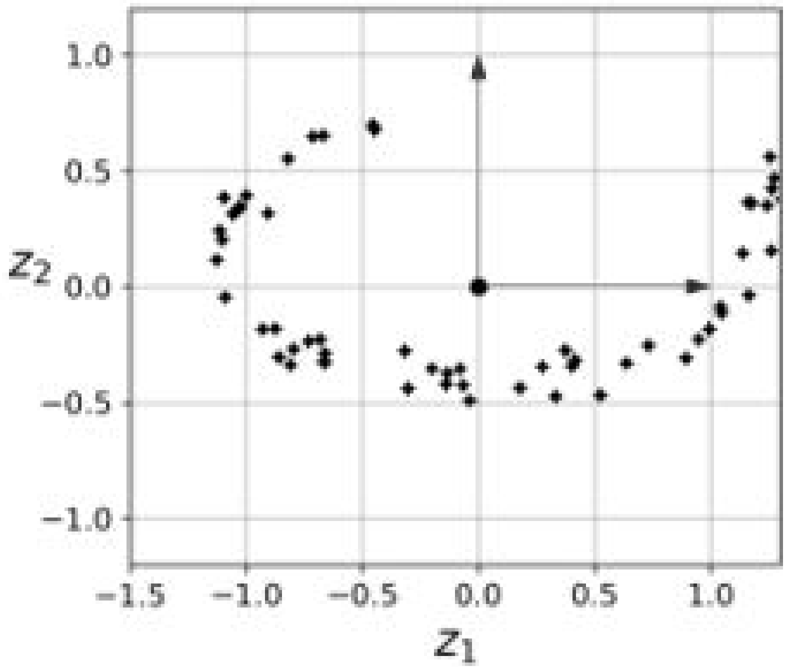
Projection

Manifold Learning.

Projection:-



A 3D dataset lying close to a 2D subspace

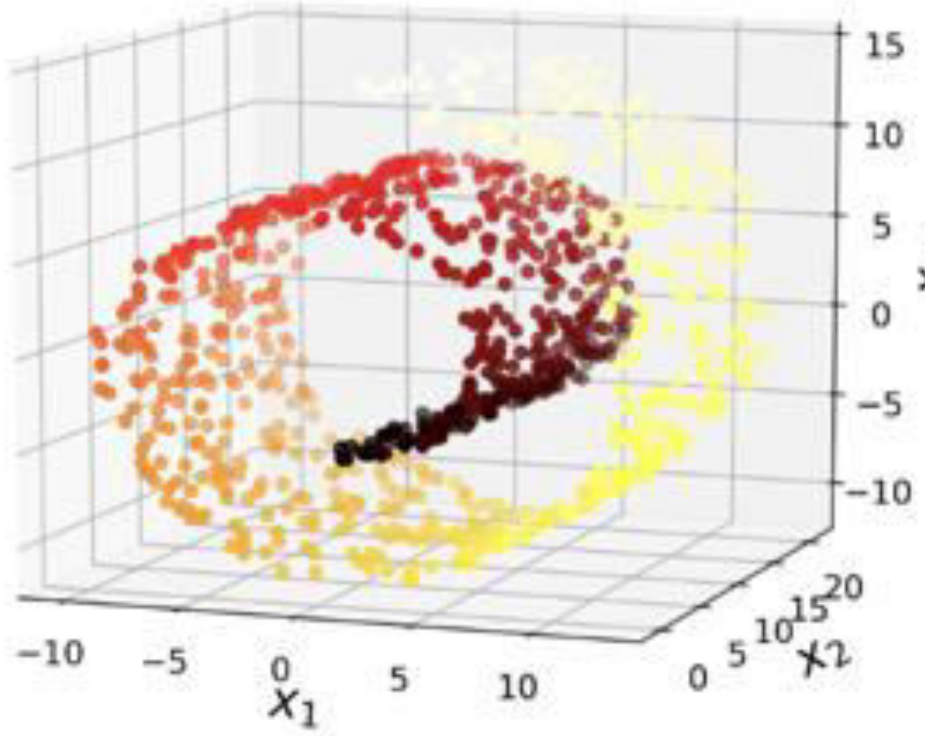


The new 2D dataset after projection

In most real-world problems, training instances are not spread out uniformly across all dimensions. Many features are almost constant, while others are highly correlated (as discussed earlier for MNIST). As a result, all training instances actually lie within (or close to) a much lower-dimensional subspace of the high-dimensional space.

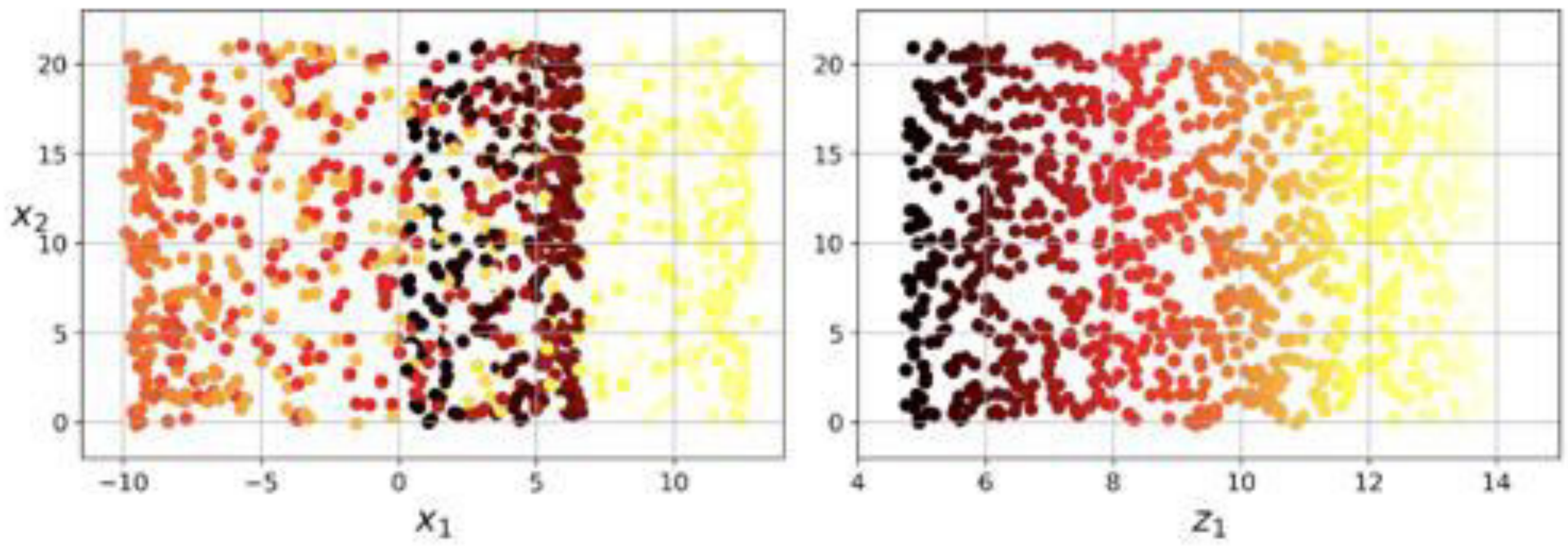
All training instances lie close to a plane: this is a lower-dimensional (2D) subspace of the high-dimensional (3D) space. Now if we project every training instance perpendicularly onto this subspace (as represented by the short lines connecting the instances to the plane). We have just reduced the dataset's dimensionality from 3D to 2D. Note that the axes correspond to new features z1 and z2 (the coordinates of the projections on the plane).

Exception case of Projection:-



Swiss roll dataset

In many cases the subspace may twist and turn, such as in the famous Swiss roll toy dataset.



Squashing by projecting onto a plane (le) versus unrolling the Swiss roll (right)

Simply projecting onto a plane (e.g., by dropping x3) would squash different layers of the Swiss roll together, as shown on the left.-However, what you really want is to unroll the Swiss roll to obtain the 2D dataset on the right

Principal component Analysis:-

1. Principal Component Analysis (PCA) is by far the most popular dimensionality reduction algorithm.
2. First it identifies the hyperplane that lies closest to the data, and then it projects the data onto it.

Principal component Analysis:-

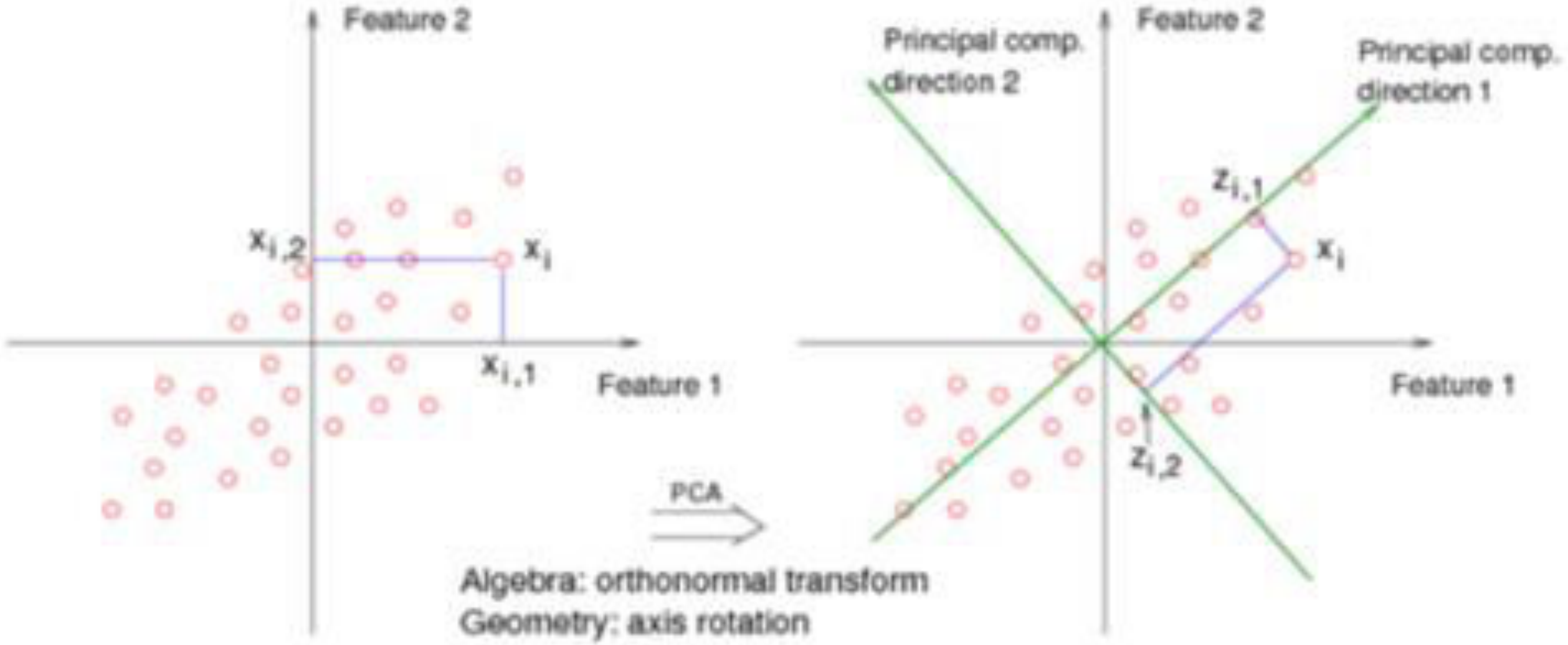
- PCA was first introduced as a linear regression method in 1901 by Karl Pearson
- Hotelling in 1933 formally introduced the name principal components analysis
- C.R. Rao in a classic paper in 1964 describes many uses and interpretation of PCA
- Currently PCA is mainly well known and used as a compression technique to deal with large dimensional data

PCA : Overview

- PCA is a linear transformation of variables
- PCs are linear (weighted) combinations of original variables
- PCs are directions of maximum variability
- PCs are orthogonal to each other (uncorrelated)

Usually beneficial for dense data (few zero values in features)

PCA – Geometry of Linear Transformation



PCA - Procedure

- Data matrix: Columns are variables (p), rows are samples (N) : p < N

$$\mathbf{X}_{N\times p} = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{p1} \\ x_{12} & x_{22} & \cdots & x_{p2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1N} & x_{2N} & \cdots & x_{pN} \end{bmatrix}$$

- Mean (row) vector: $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_{i.}$ *where* $\mathbf{X}_{i.}$ *is row i of* $\mathbf{X}$

- Covariance matrix (p x p): $S_X = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{X}_{i.} - \bar{\mathbf{x}})^T(\mathbf{X}_{i.} - \bar{\mathbf{x}}) = \frac{1}{N}\mathbf{X}_s^T\mathbf{X}_s$

- $\mathbf{X}_s$ is data matrix after mean (row) is subtracted from each row

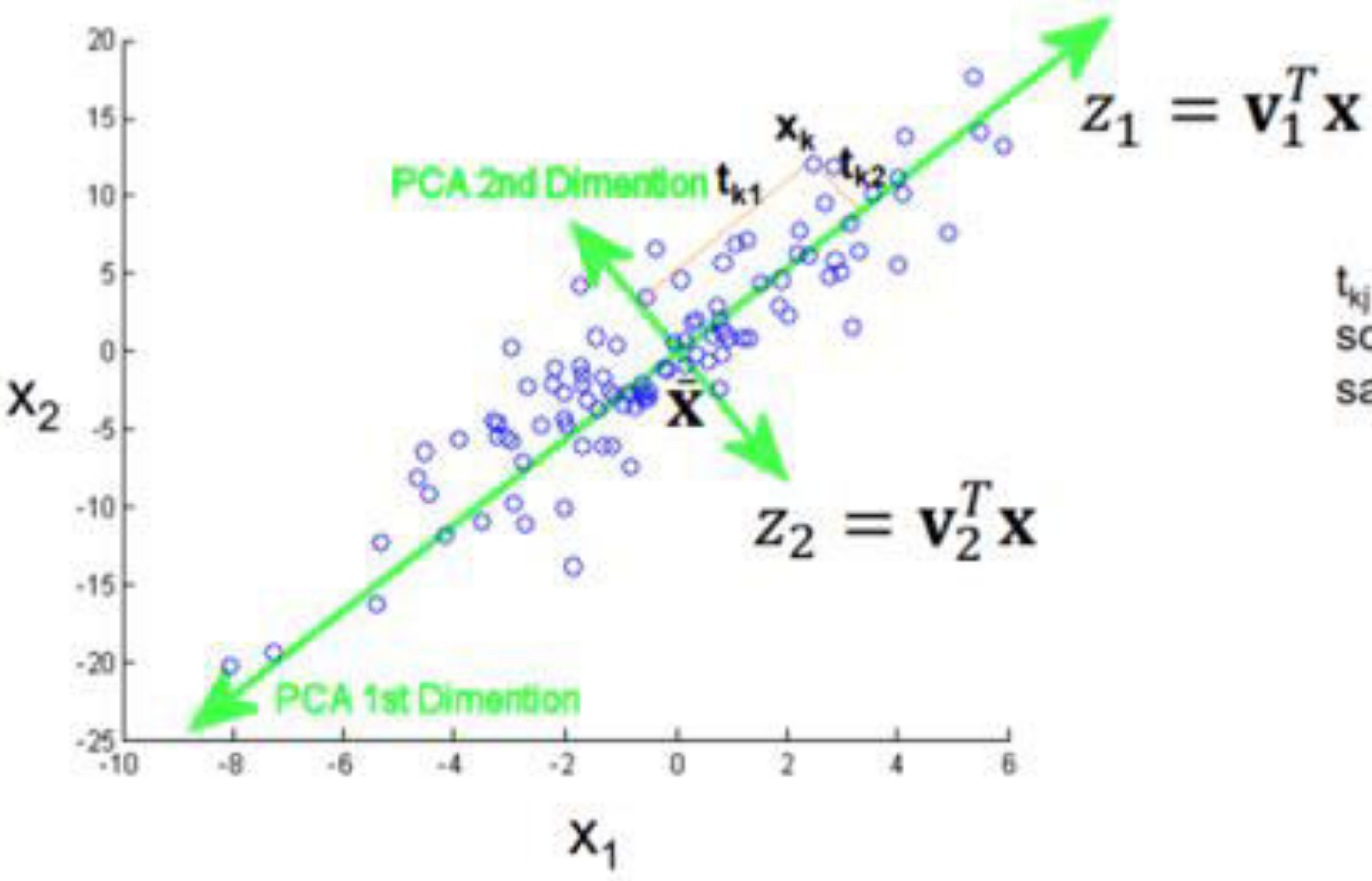
PCA – Method…(2)

- Find eigenvalues and eigenvectors of $\mathbf{S_x}$

$$\mathbf{S_x}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

- Number of non-zero eigenvalues = rank of covariance matrix = p
- Eigenvalues and corresponding eigenvectors are ordered from highest to lowest
- Eigenvectors are orthonormal
- $z_i = \mathbf{v}_i^T \mathbf{x}$ are the new transformed variables or PCs

PCA -Geometry



t_{kj} are also known as scores corresponding to sample k

Choosing number of PCs

Percentage variance

- Total variance in data = $Tr(\mathbf{S_X}) = \sum_{i=1}^{p} \mathbf{S}_{\mathbf{x,ii}} = \sum_{i=1}^{p} \lambda_i$

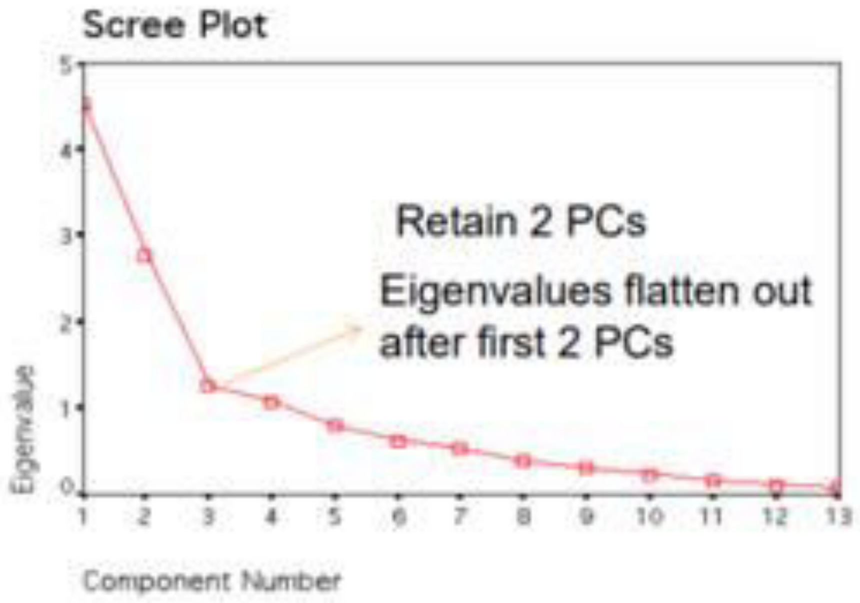
- Variance of *k*'th PC = λ_k
- % Variance explained by first *k* PCs

$$100 * \sum_{i=1}^{k} \lambda_i \,/ \sum_{i=1}^{p} \lambda_i$$

  - Choose first *k* PCs such that 95% or 99% of variance is explained by them

Scree Plot

- Scree Plot – Plot $\log(\lambda_i)$ vs index i



- Look for 'kinks' in Scree plot and 'flattening' of the eigenvalues

Singular Value Decomposition (SVD)

- SVD (economical) of the data matrix

$$svd(\boldsymbol{X}_s) = \boldsymbol{U}_{N\times p}\boldsymbol{S}_{p\times p}\boldsymbol{V}^T_{p\times p}$$

- ***U*** and ***V*** are orthonormal matrices

$$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}_{p\times p} \qquad \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_{p\times p}$$

- ***S*** is a diagonal matrix whose diagonal entries (ordered from largest to smallest) are called singular values

- Columns of ***U*** are left singular vector and columns of ***V*** are the right singular vectors of the data matrix

Usually beneficial for sparse data (in features)

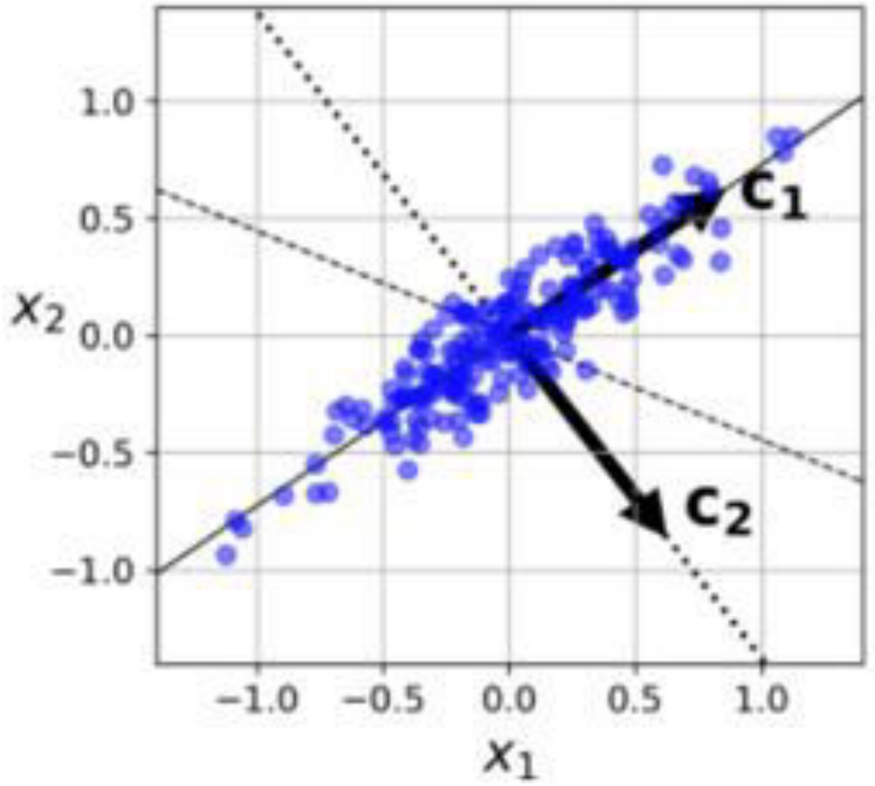
SVD and Eigenvectors

- SVD of data matrix are related to eigenvectors of covariance matrix

- Columns of $\mathbf{V}$ are eigenvectors of $\mathbf{S}_x$

- Singular values are square root of eigenvalues

Preserving the Variance:-

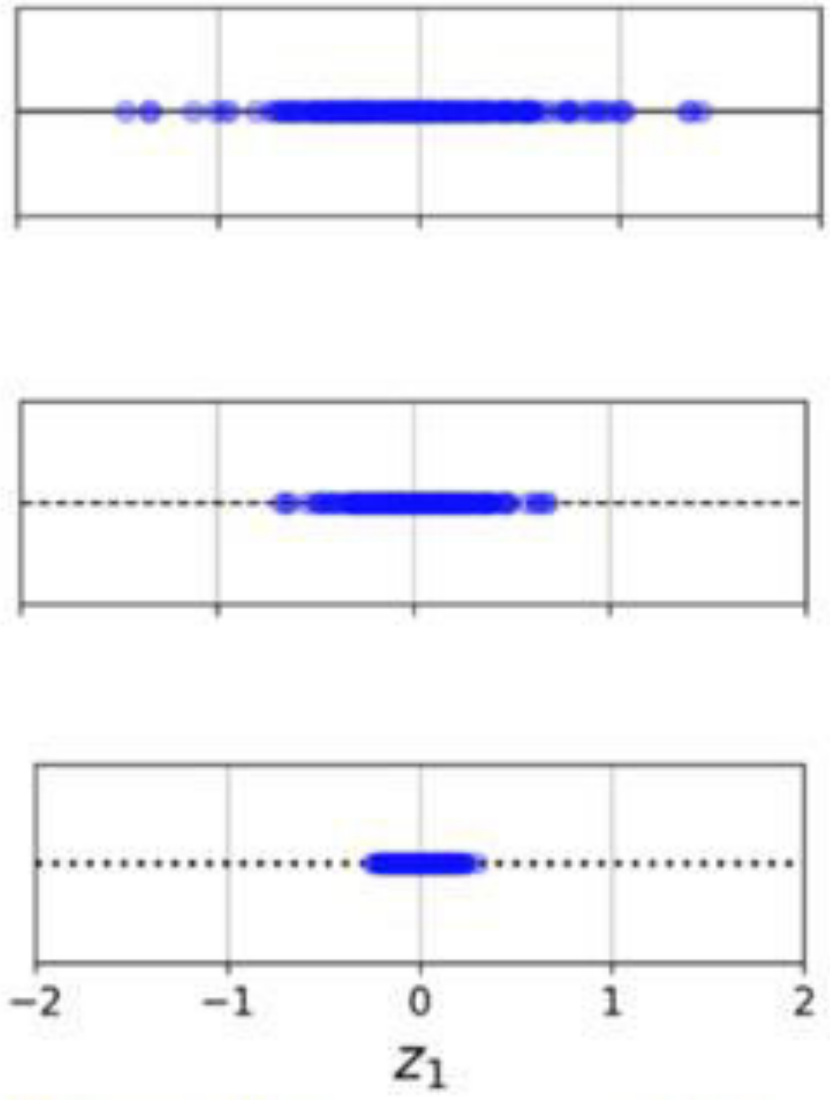


A simple 2D dataset along with three different axes

Maximum variance

Very little variance

Intermediate amount of variance



The projection of the dataset onto each of these axes.

PCA identifies the axis that accounts for the largest amount of variance in the training set.

If it were a higher-dimensional dataset, PCA would also find a third axis, orthogonal to both previous axes, and a fourth, a fifth, and so on as many axes as the number of dimensions in the dataset.

PCA used for:-

- Data Compression (reducing dimension of high dimensional multivariate data)
  - Reduced storage
  - Ease of visualization
- Building linear models
  - Regression model (using a total least squares approach)
- Denoising and reconciling erroneous data

PCA for decompression:-

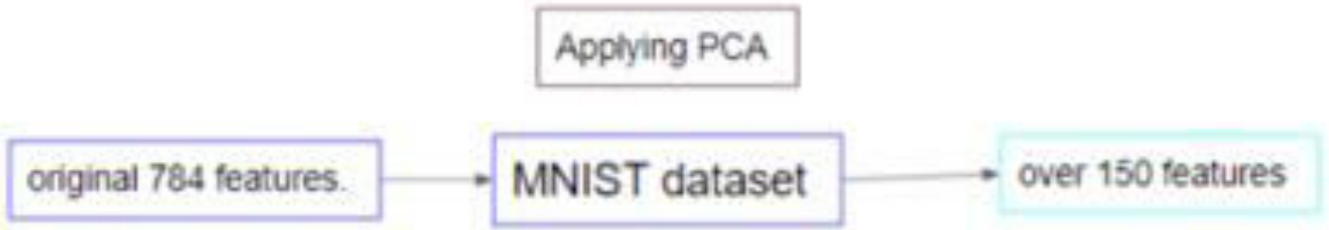
Use the inverse transformation of the PCA

this won't give you back the original data, since the projection lost a bit of information (within the 5% variance that was dropped), but it will likely be quite close to the original data.

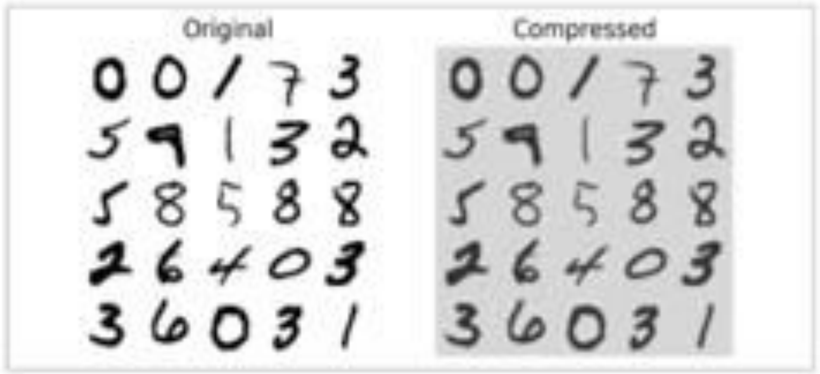
The Reconstruction Error:-

The mean squared distance between the original data and the reconstructed data (compressed and then decompressed) is called the **reconstruction error**.

PCA for Compression:-



MNIST compression preserving 95% of the variance



Manifold Learning:-

Swiss roll is an example of a 2D manifold.

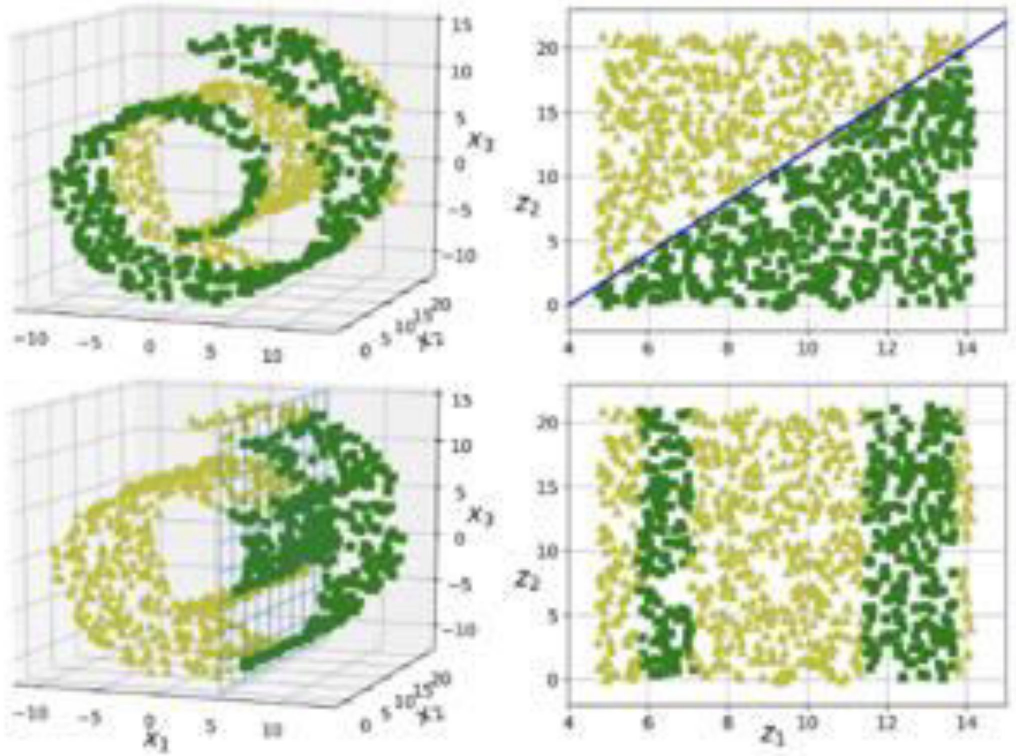
A 2D manifold is a 2D shape that can be bent and twisted in a higher-dimensional space.

d-dimensional manifold is a part of an n-dimensional space (where $d < n$) that locally resembles a d-dimensional hyperplane.

Manifold Learning can be thought of as an attempt to generalize linear frameworks like PCA to be sensitive to non-linear structure in data. Though supervised variants exist, the typical manifold learning problem is unsupervised: it learns the high-dimensional structure of the data from the data itself, without the use of predetermined classifications.

In the case of the Swiss roll, $d = 2$ and $n = 3$: it locally resembles a 2D plane, but it is rolled in the third dimension.

Manifold Learning:-



The decision boundary may not always be simpler with lower dimensions

the decision boundary is located at x1

LLE (Locally Linear Embedding):-

1. It is another very powerful nonlinear dimensionality reduction (NLDR) technique.
2. LLE works by first measuring how each training instance linearly relates to its closest neighbors (c.n.), and then looking for a low-dimensional representation of the training set where these local relationships are best preserved.
3. Locally Linear Embedding, or LLE, creates an embedding of the dataset and attempts to preserve the relationships between neighborhoods in the dataset.

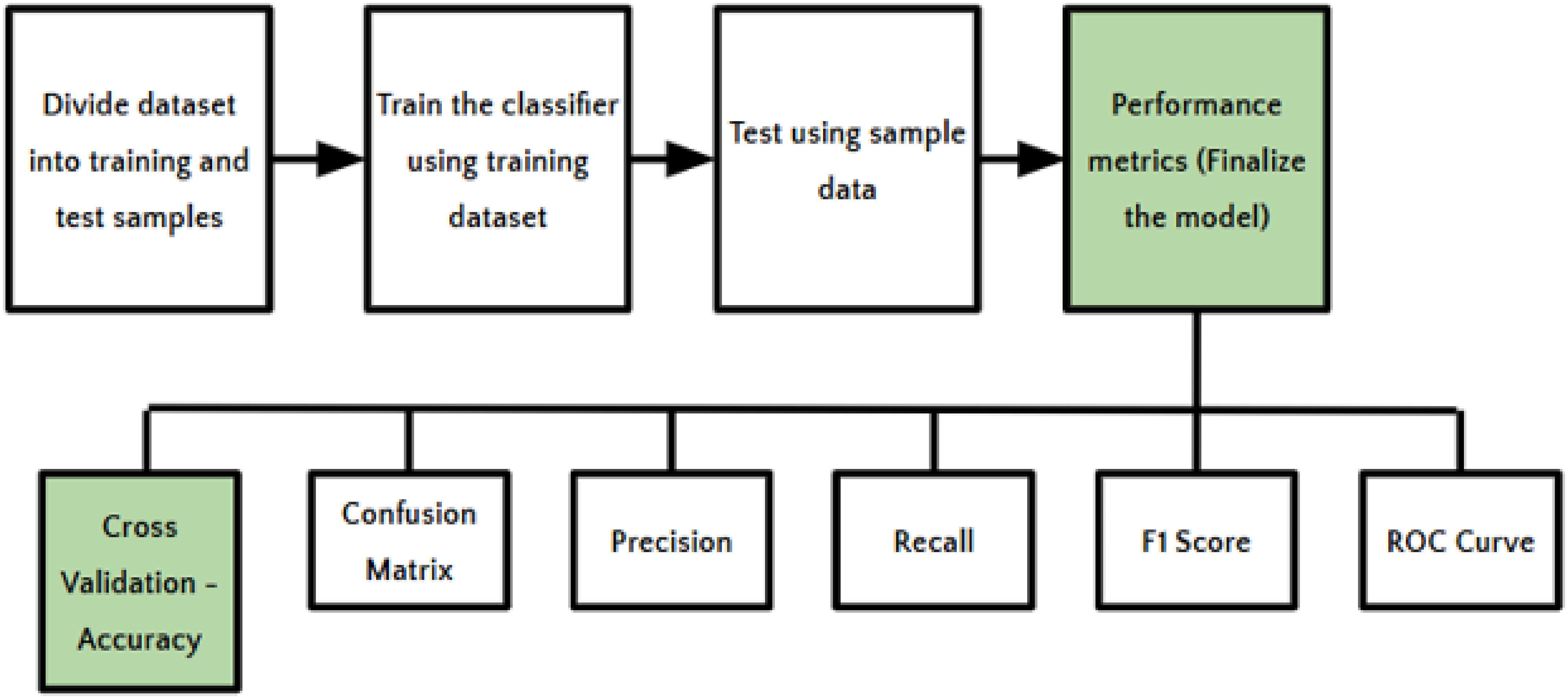
t-SNE, ISOMAP etc

It is a Manifold Learning technique that does not rely on projections like the previous algorithms. This makes it particularly good at unrolling twisted manifolds, especially when there is not too much noise.

Measuring Performance of Classifier

- Cross Validation – Accuracy
- Confusion Matrix
    - Precision
    - Recall
    - F1 score
- ROC Curve

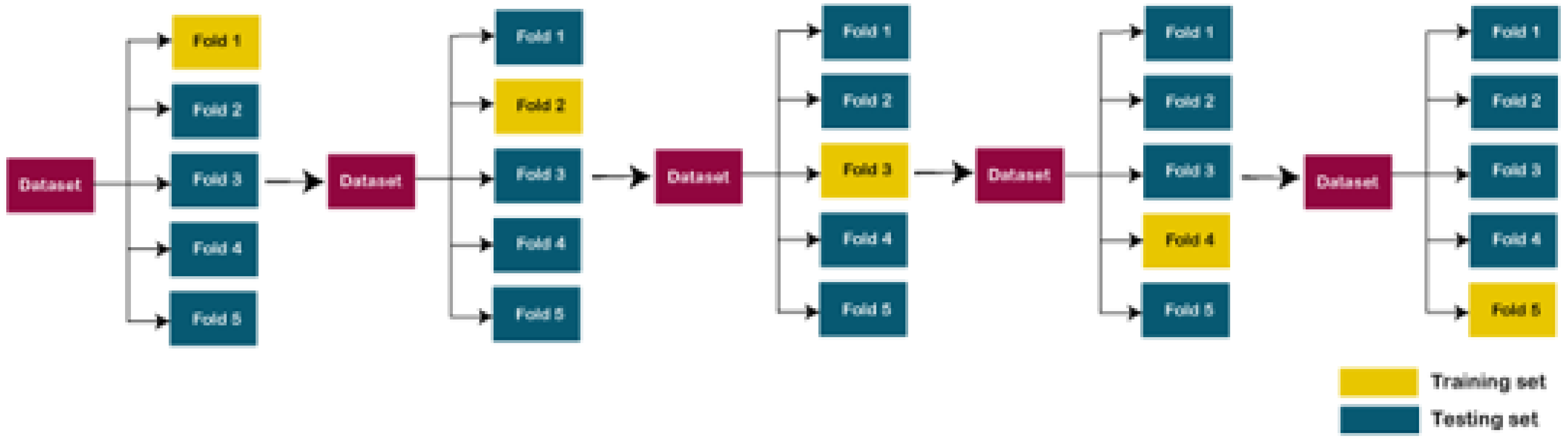
Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data
Performance metrics (Finalize the model)
Cross Validation – Accuracy
Confusion Matrix
Precision
Recall
F1 Score
ROC Curve

Cross-Validation:-

- K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called **folds**.
- For each learning set, the prediction function uses **k-1** folds, and the rest of the folds are used for the test set.
- Example of 5-folds cross-validation:-

1. The dataset is grouped into 5 folds.
2. On 1st iteration, the first fold is reserved for test the model, and rest are used to train the model.
3. On 2nd iteration, the second fold is used to test the model, and rest are used to train the model.
4. This process will continue until each fold is not used for the test fold.



Performance Measures - Cross Validation:-

cross_val_score

cross_val_score() function in scikit-learn can be used to perform cross validation.

- Value of the number of folds (k) should be chosen based on the dimension of the dataset. If dimension of the data is less go for more cross-validation steps. If dimension of the data is large go for the less cross-validation steps.

Performance Measures - Cross Validation accuracy

```
>>> array([ 0.938, 0.93766667, 0.94666667])
```

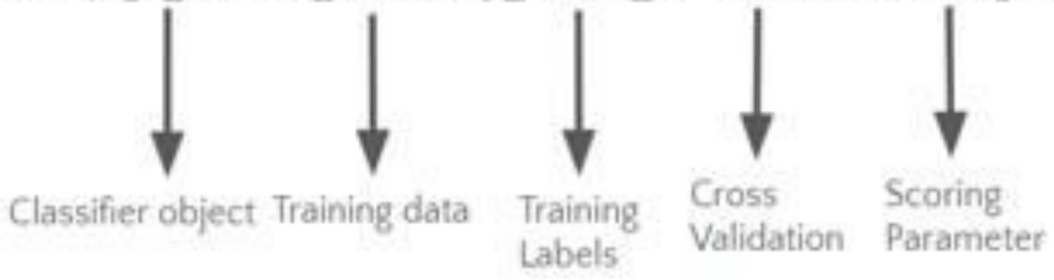
- The resulting accuracy is above 92% for each of the folds

Performance Measures - Cross Validation

Initiating cross validation method

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

>>> cross_val_score(sgd_clf, X_train, y_train_8, cv=10, scoring="accuracy")
```



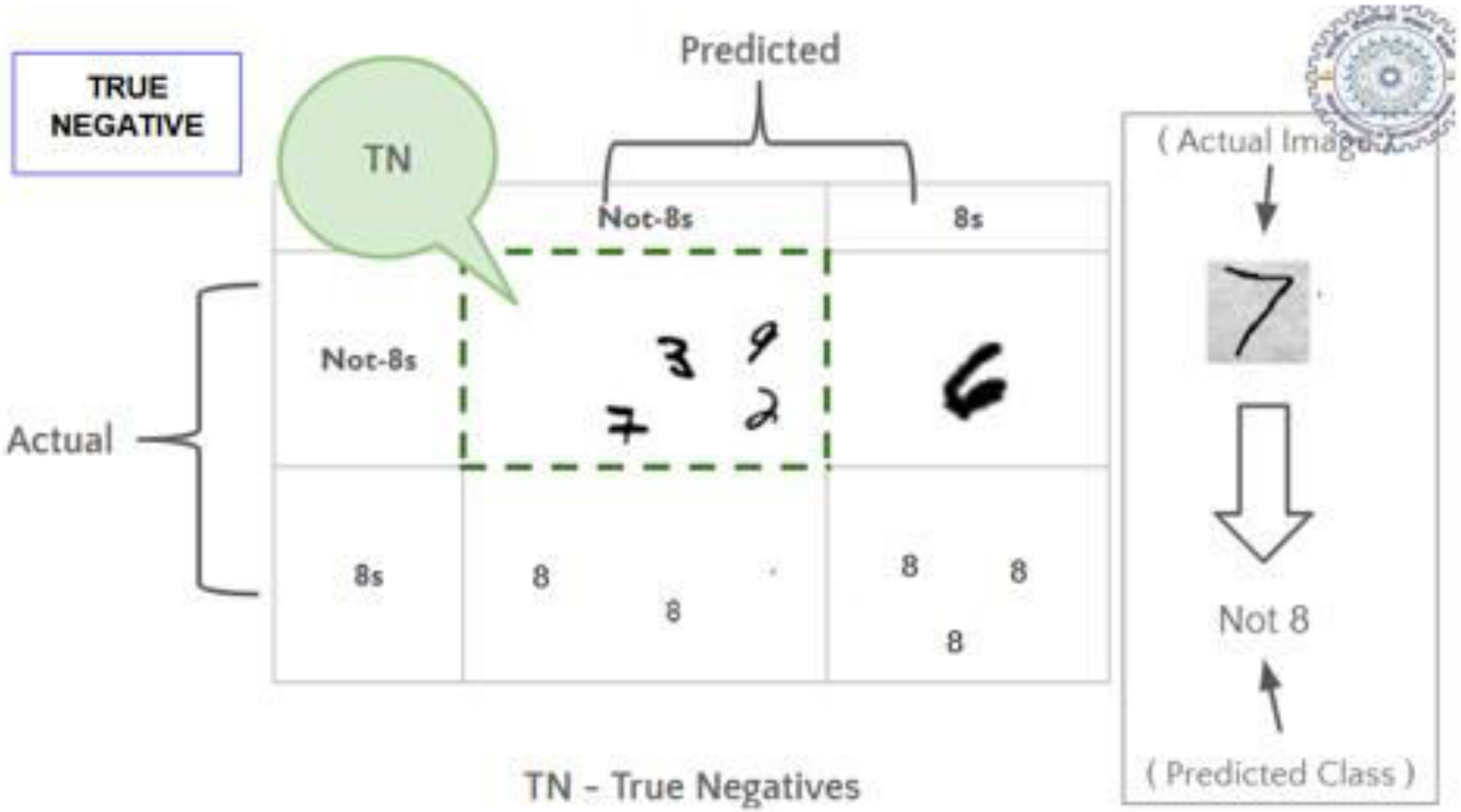
Performance Measures - Confusion Matrix

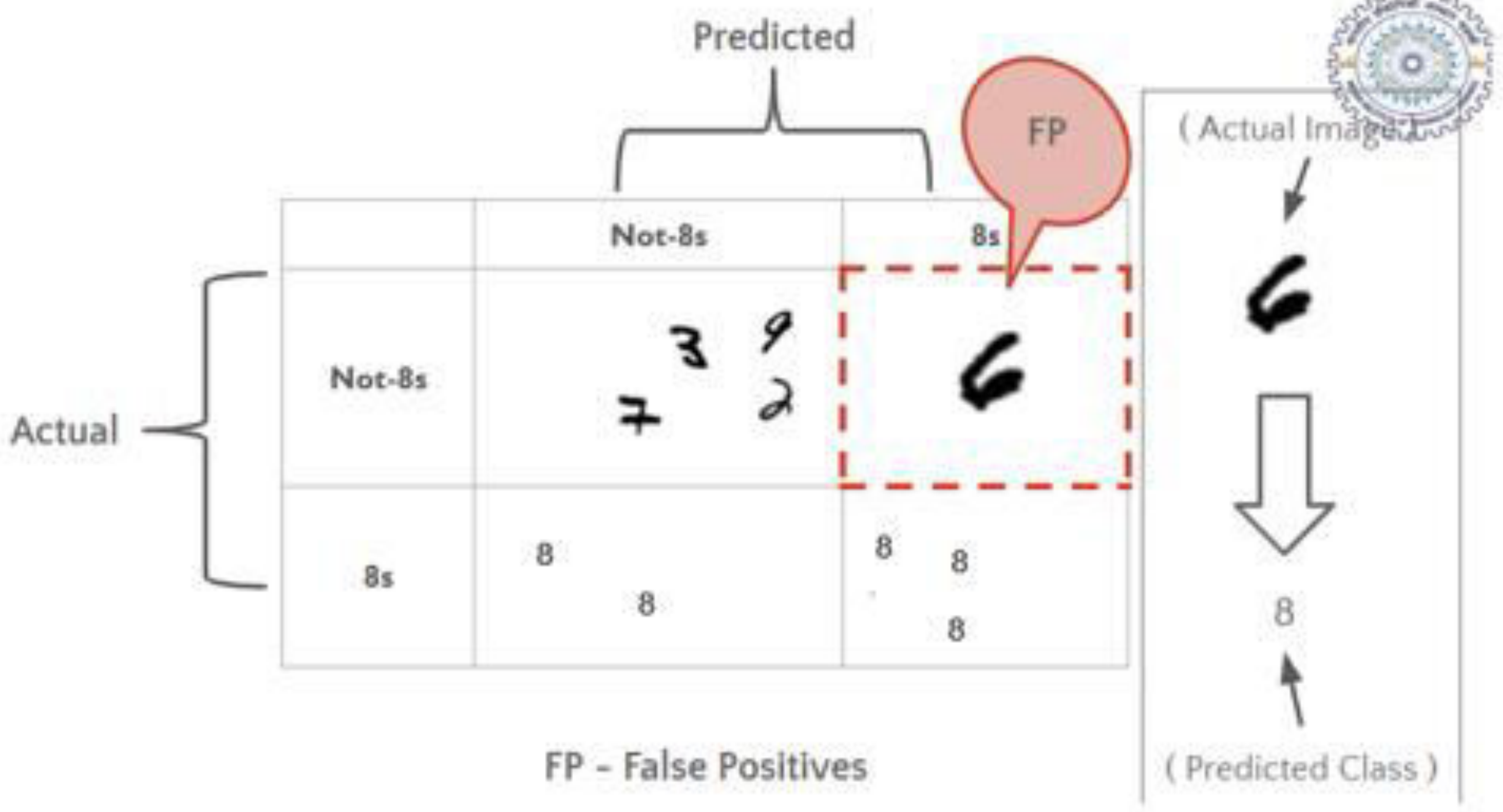
- A confusion matrix visualizes and summarizes the performance of a classification algorithm.
- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.

Performance Measures – Confusion Matrix

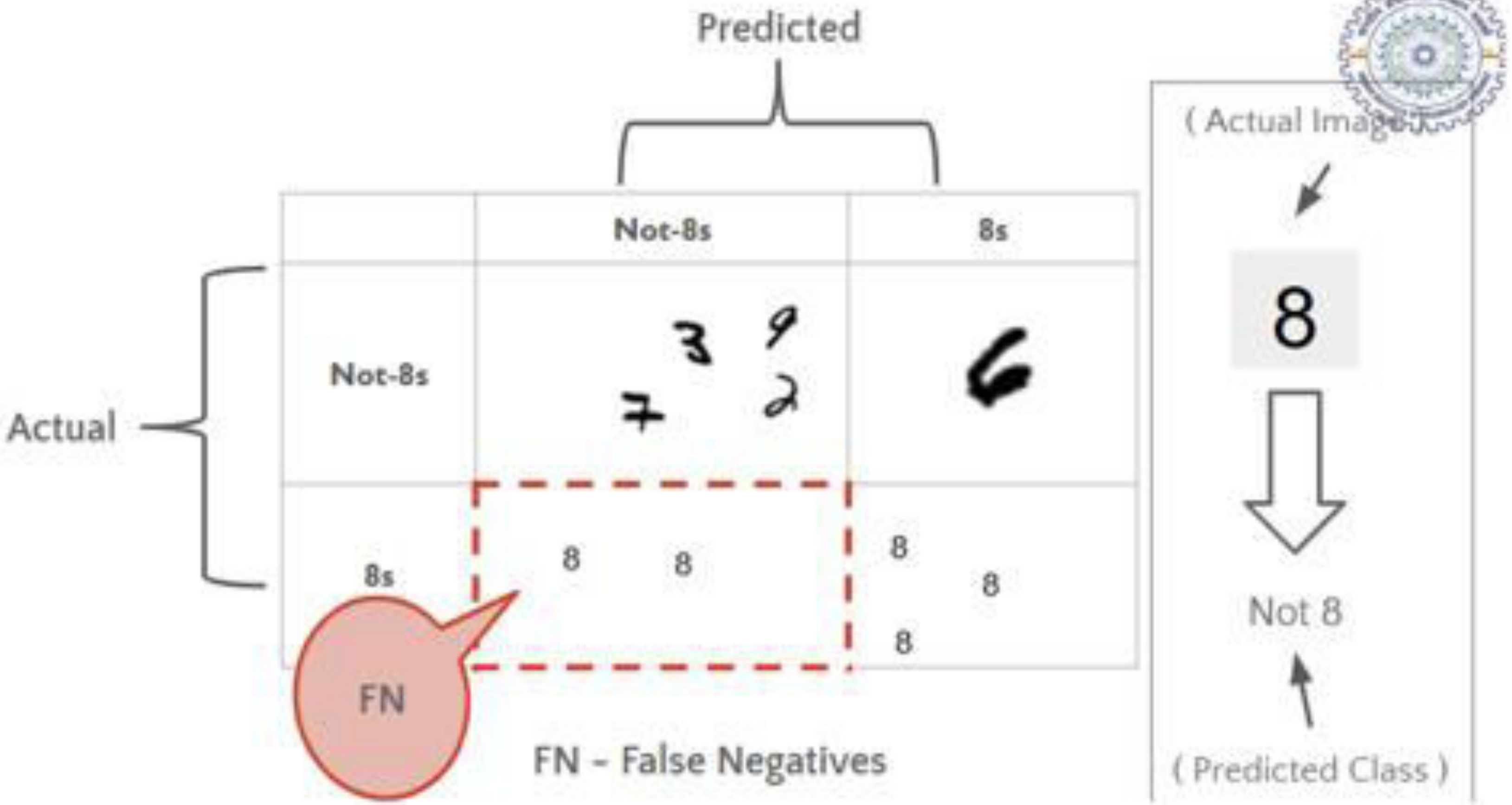
	Predicted YES	Predicted NO
Actual YES	620	380
Actual NO	180	8820

TRUE NEGATIVE

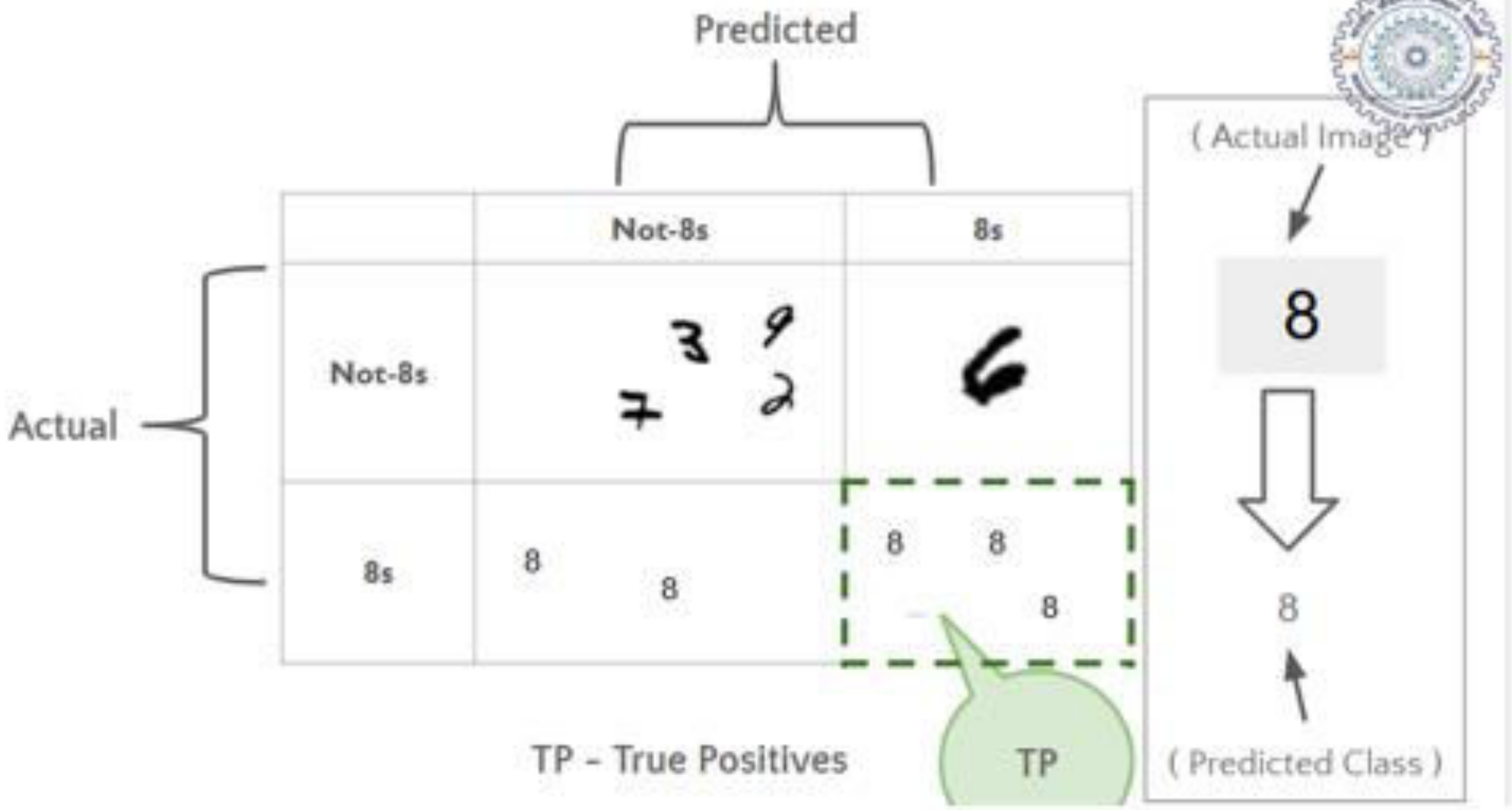




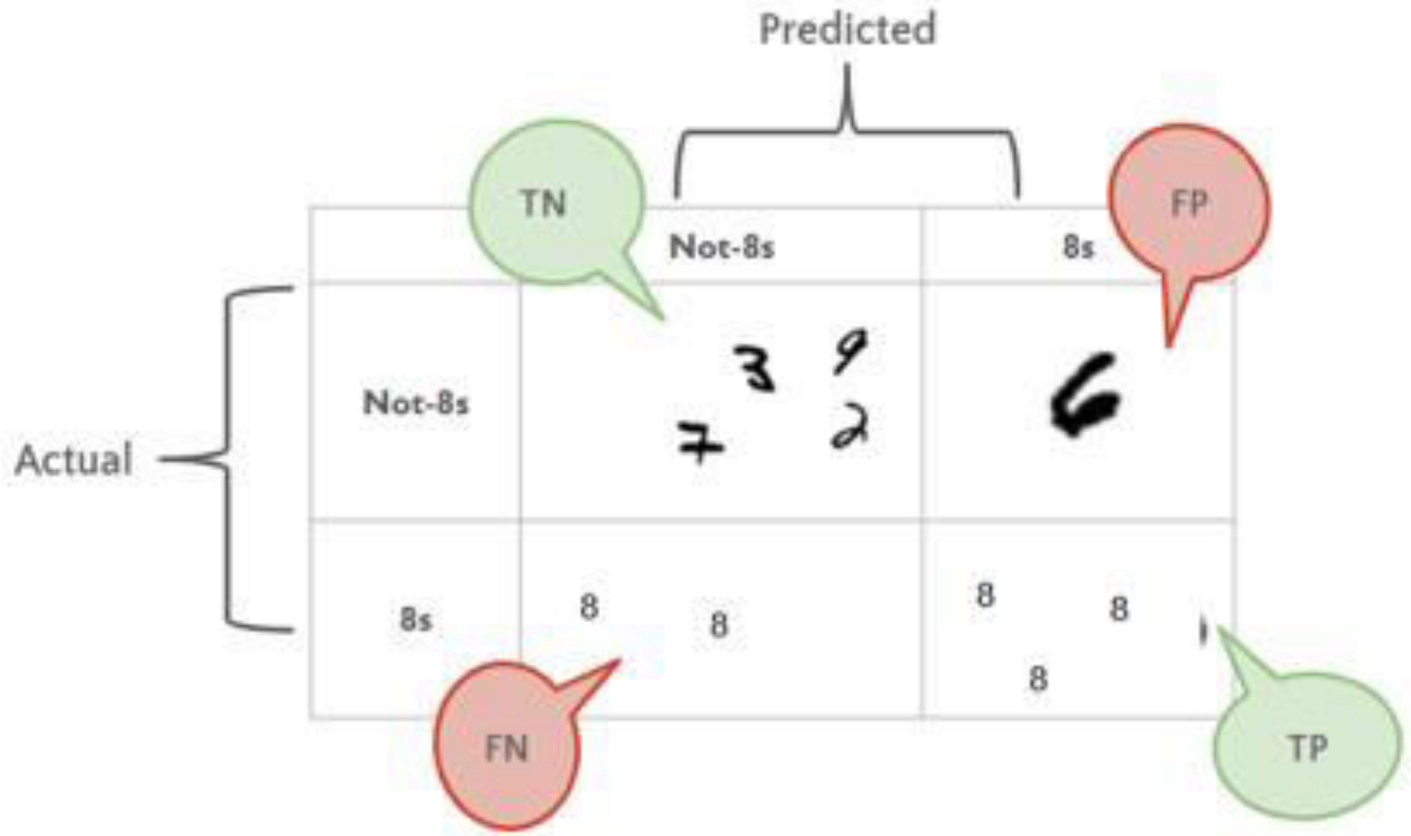
FP - False Positives



FN – False Negatives



TP – True Positives


Predicted
Actual
Not-8s
8s
Not-8s
8s
TN
FP
FN
TP
3
9
7
2
6
8
8
8
8
8

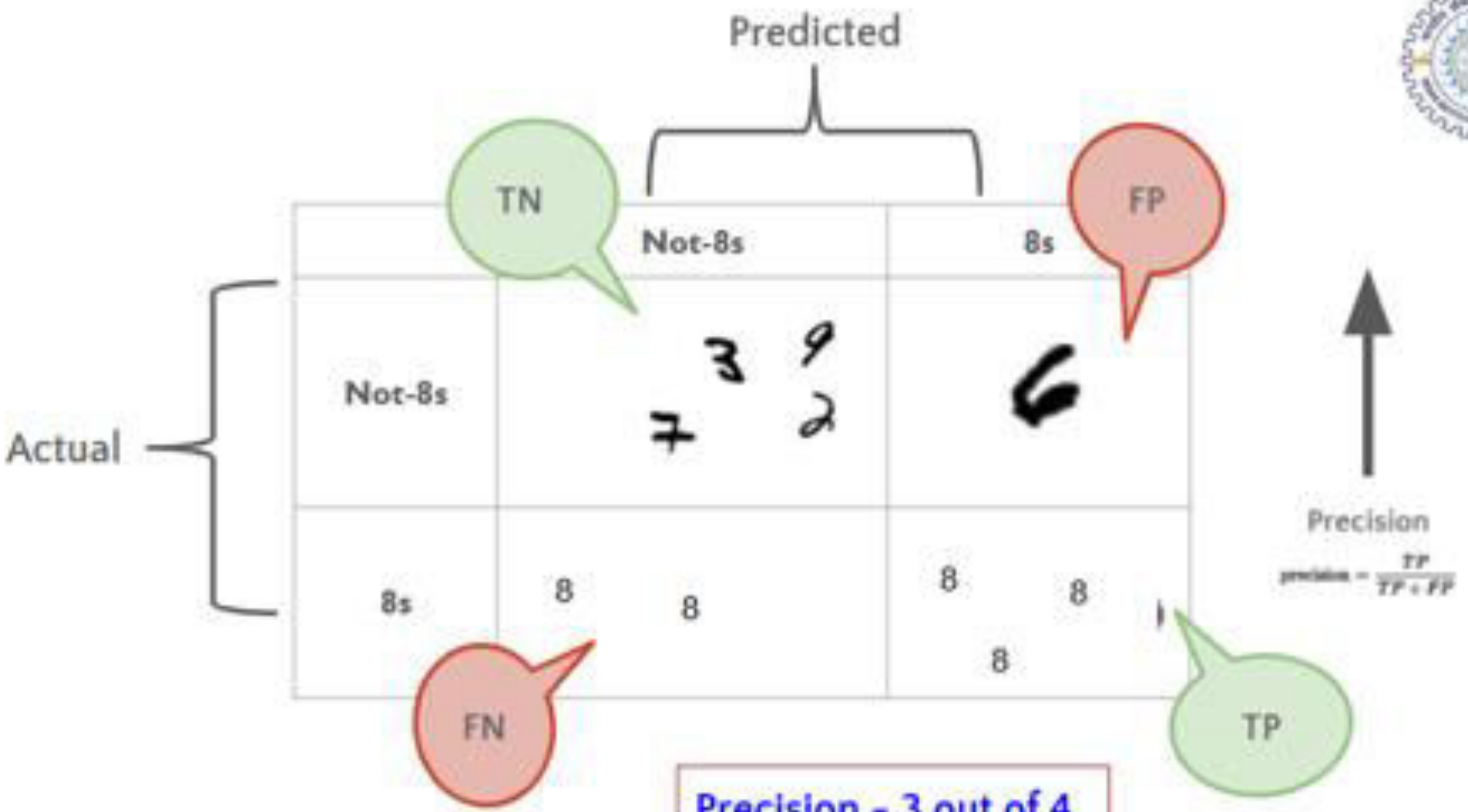
Precision and Recall

Performance Measures - Precision

- what proportion of images that were classified correctly as 8s were

  Precision is how many times an accurate prediction of a particular class
- occurs per a false prediction of that class.

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



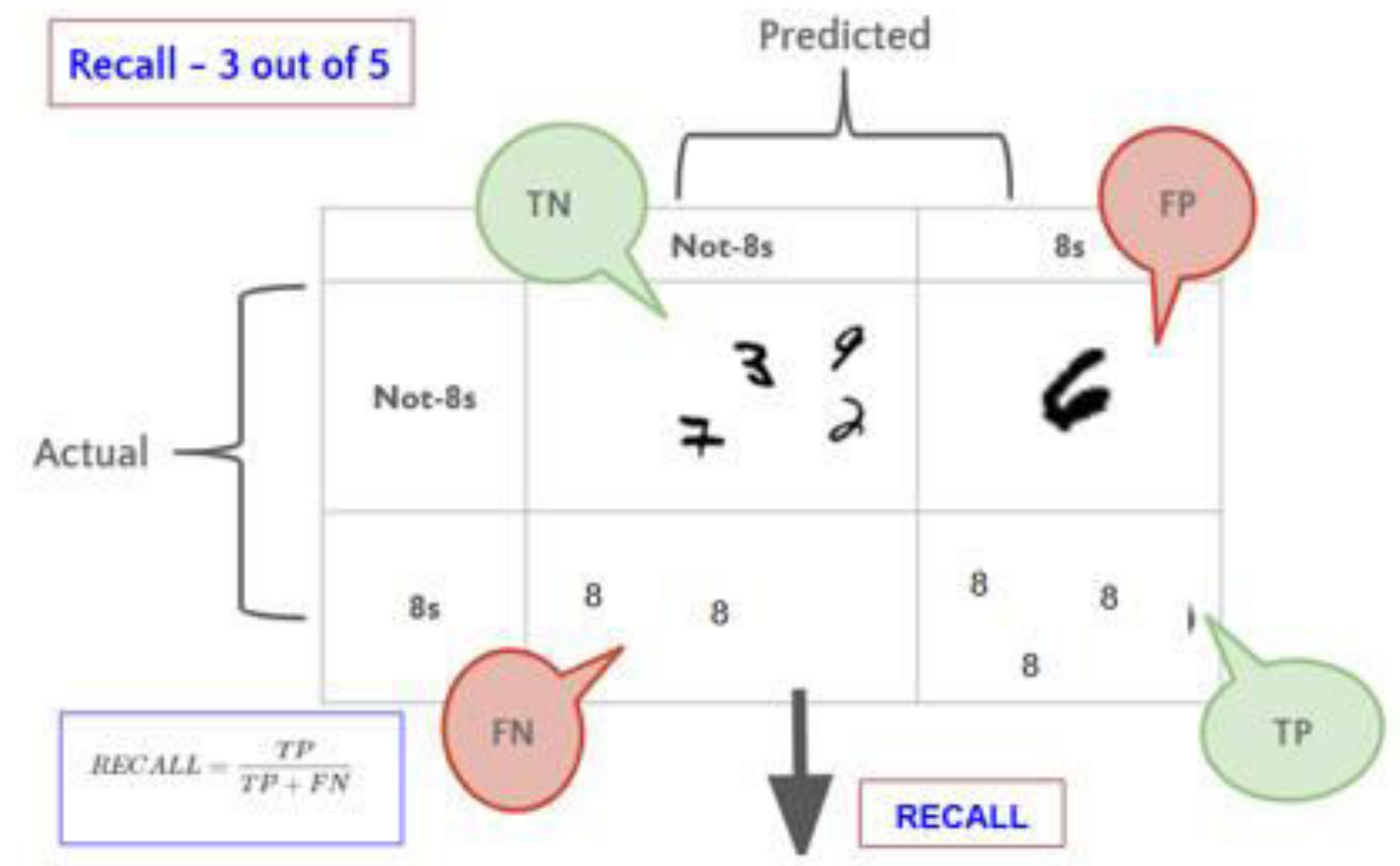
Precision - 3 out of 4

Performance Measures - Recall

Recall means remember something learnt in past

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

- **Performance Measures - Recall**
what proportion of images that were actually 8 were predicted as class 8, to measure the completeness of positive predictions.


Recall - 3 out of 5
Predicted
TN
Not-8s
8s
FP
Actual
Not-8s
8s
8
8
8
8
8
FN
TP
$RECALL = \frac{TP}{TP + FN}$
RECALL

Performance Measures – Confusion Matrix

Precision = 58 %

Correct only
58% of time

Recall = 50 %

Detects only
50% of 8s

Precision / Recall Score of SDG Classifier

Performance Measures – F1 Score

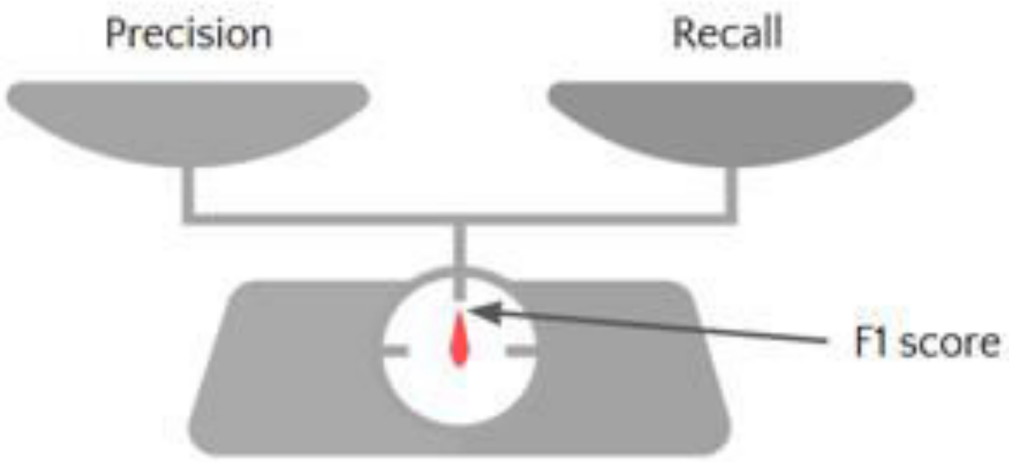
- Instead of computing precision and recall every time
- We prefer a single metric which combines both precision and recall
- This single metric is F1 score

- **Performance Measures - F1 Score**

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

- F1 score is harmonic mean of precision and recall

Performance Measures - F1 Score



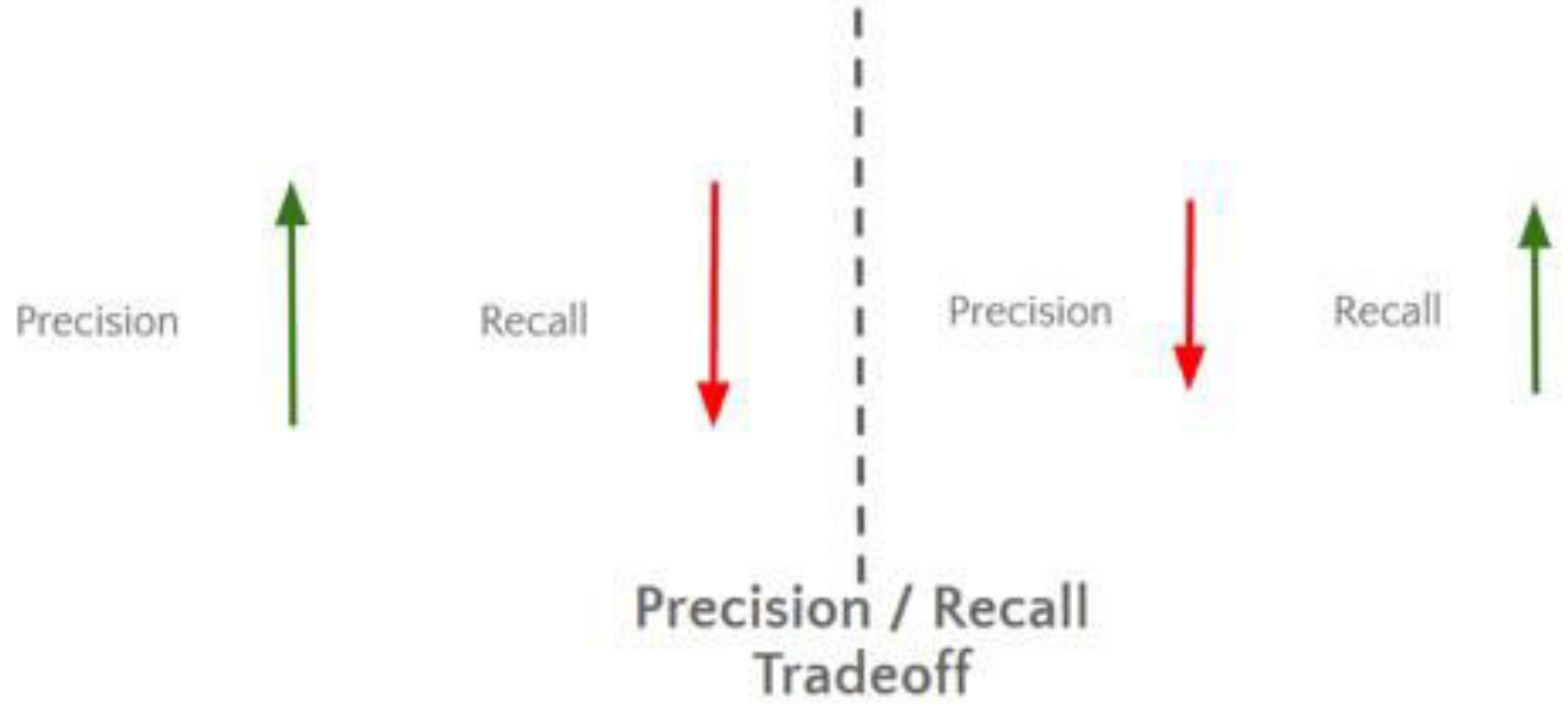
F1 score is high when both **Precision** and **Recall** are almost similar

Performance Measures - Precision / Recall Tradeoff

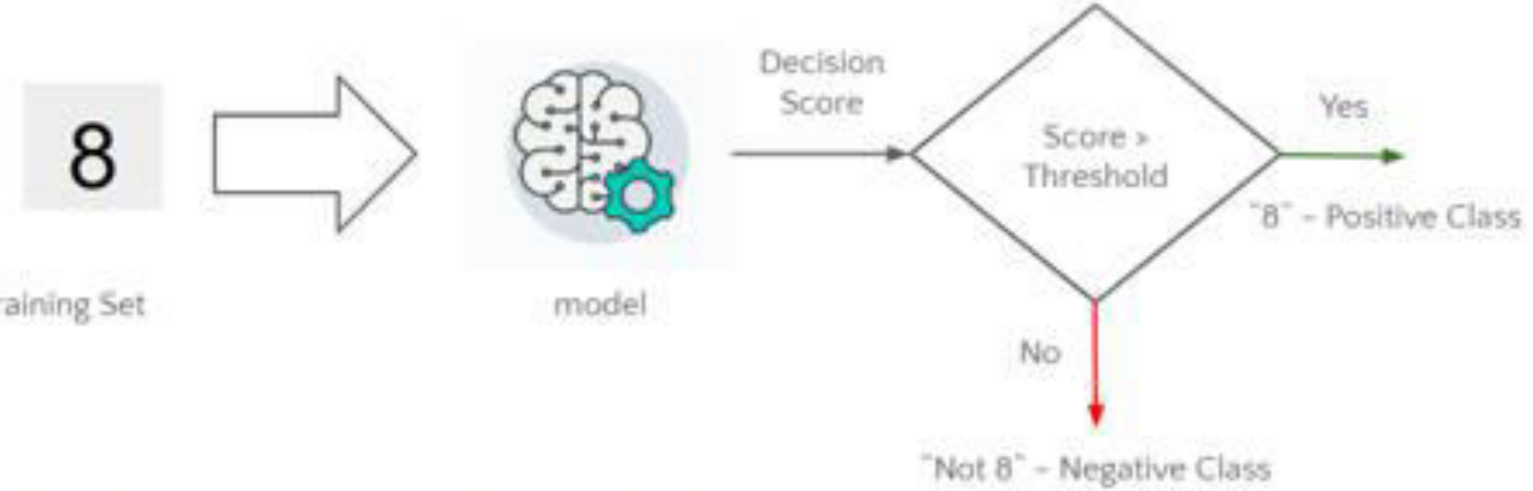
Now you may think that we can have both high precision and high recall in a model. But unfortunately, we can't have both high precision and high recall at the same time.

Precision ↑ **Recall** ↑

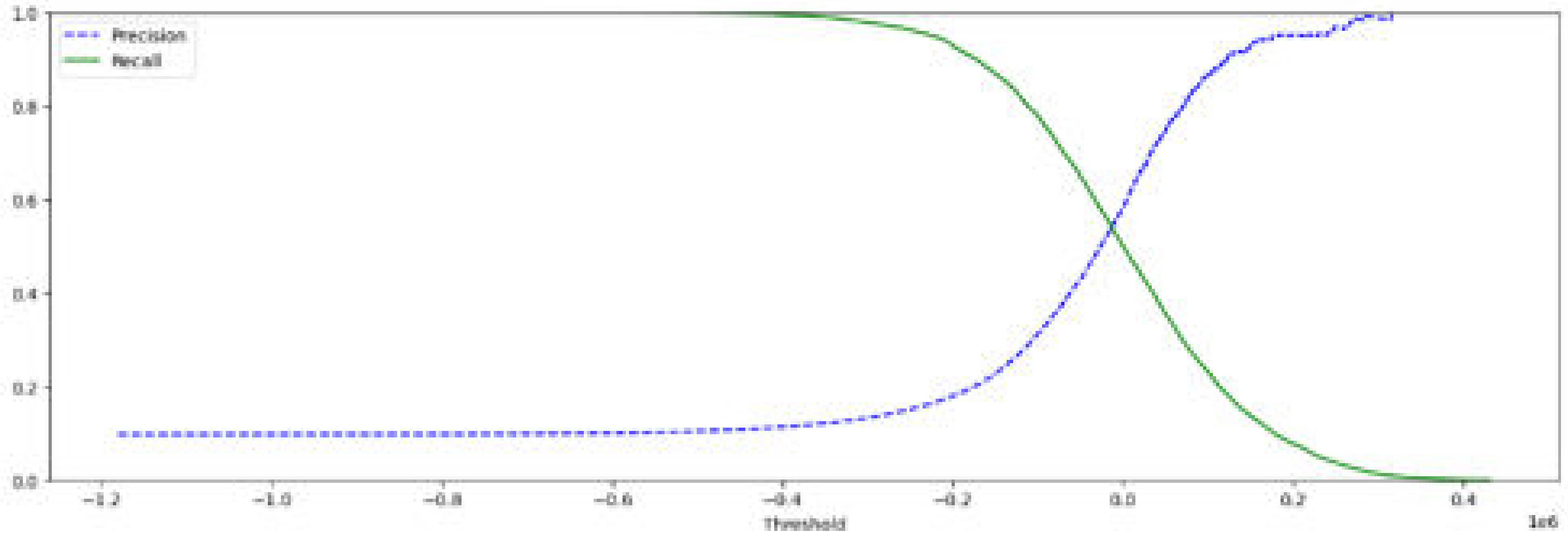
Performance Measures – Precision / Recall Tradeoff



Performance Measures - Precision / Recall Tradeoff



Threshold - decided by the classification algorithm to differentiate +ve & -ve classes. Decision score- distance from the decision boundary (if it +ve, sample belongs to positive class, Otherwise negative class.)



Plot to show the inverse relation of Precision and Recall

The ROC (Receiver Operating Characteristic) Curve

Credit: ROC Curves for Continuous Data, Author(s): Wojtek J. Krzanowski, David J. Hand, 2009 CRC

Performance Measures - ROC Curve

- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.
- This curve plots two parameters:
- True Positive Rate: It is a synonym for recall.

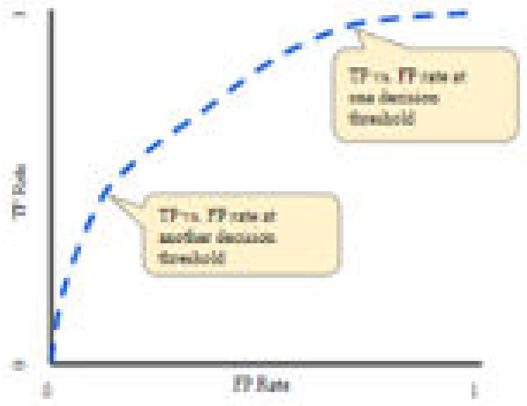
$$TPR = \frac{TP}{TP + FN}$$

- False Positive Rate: it is defined as follows: (the proportion falsely classified among all actual negatives)

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

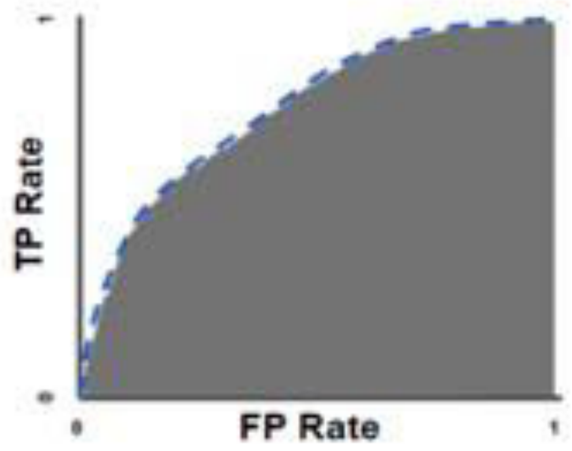
Performance Measures – ROC Curve

- An ROC curve plots TPR vs. FPR at different classification thresholds.
- Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).



Model Comparisons:-

Confusion Matrix

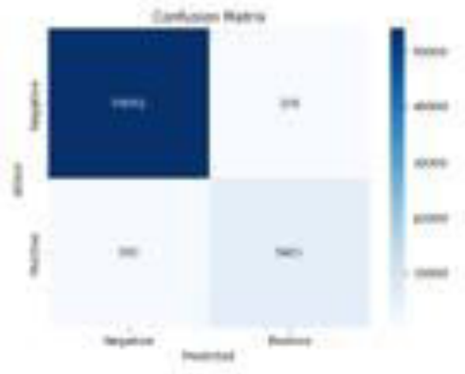
SGD

Confusion Matrix
Negative
Positive
Predicted
Actual

RF

Confusion Matrix
Negative
Positive
Predicted
Actual

SVM

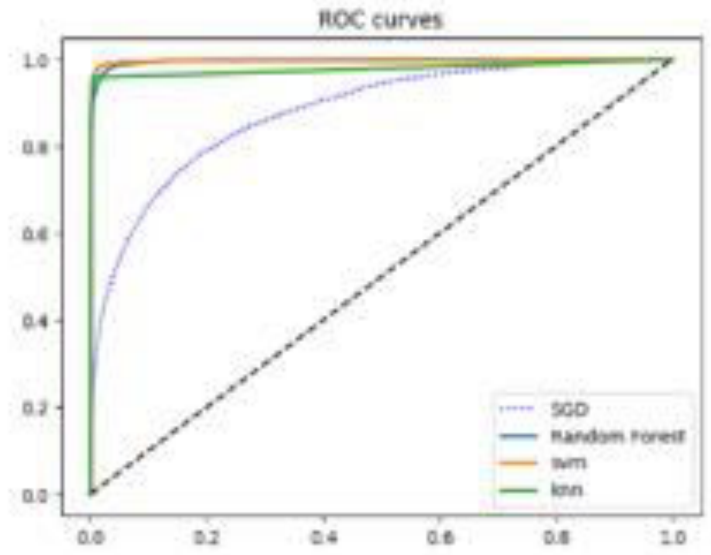


KNN

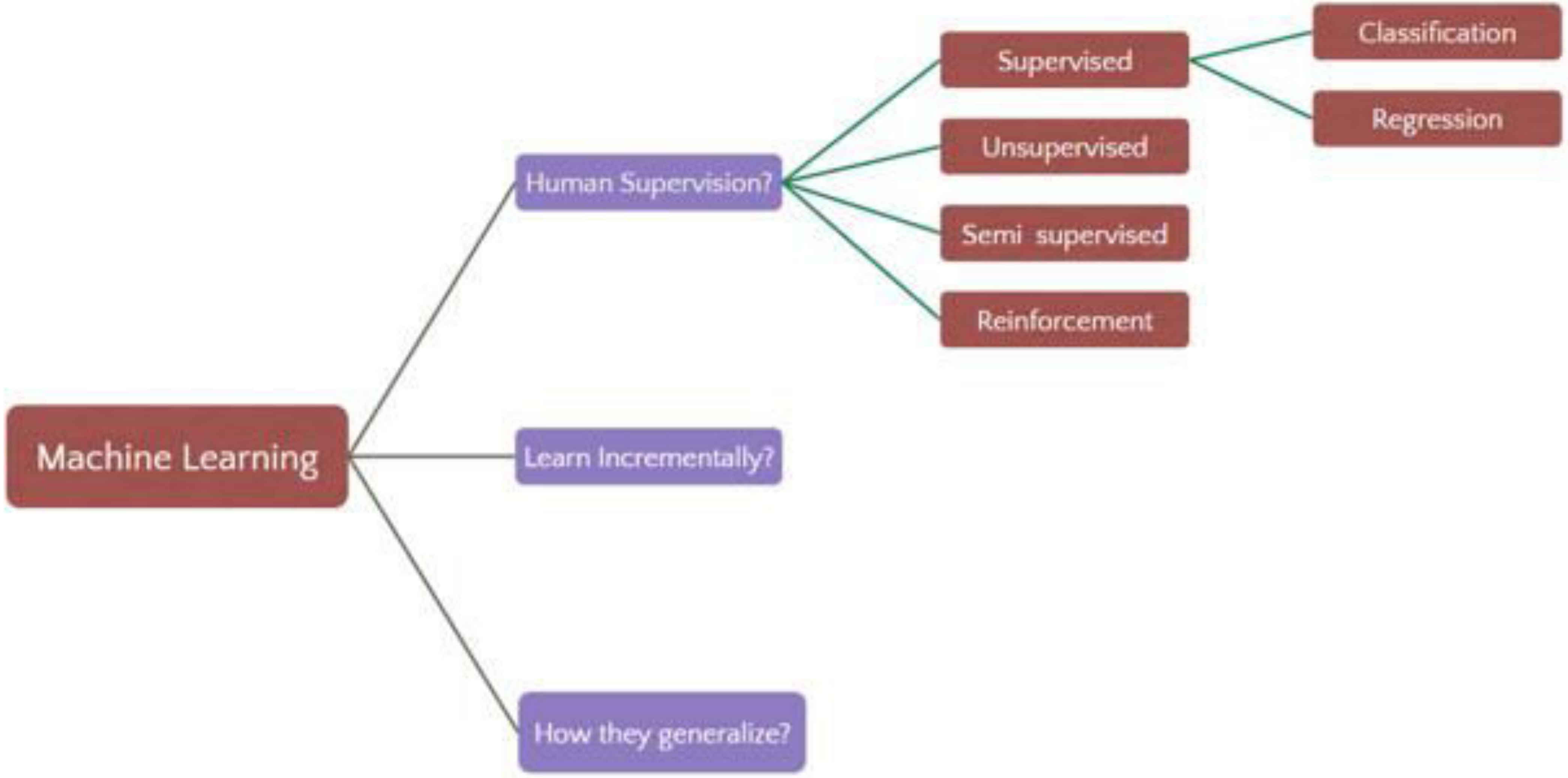
Confusion Matrix
Negative
Positive
Predicted
Actual

Comparison with other metrics:-

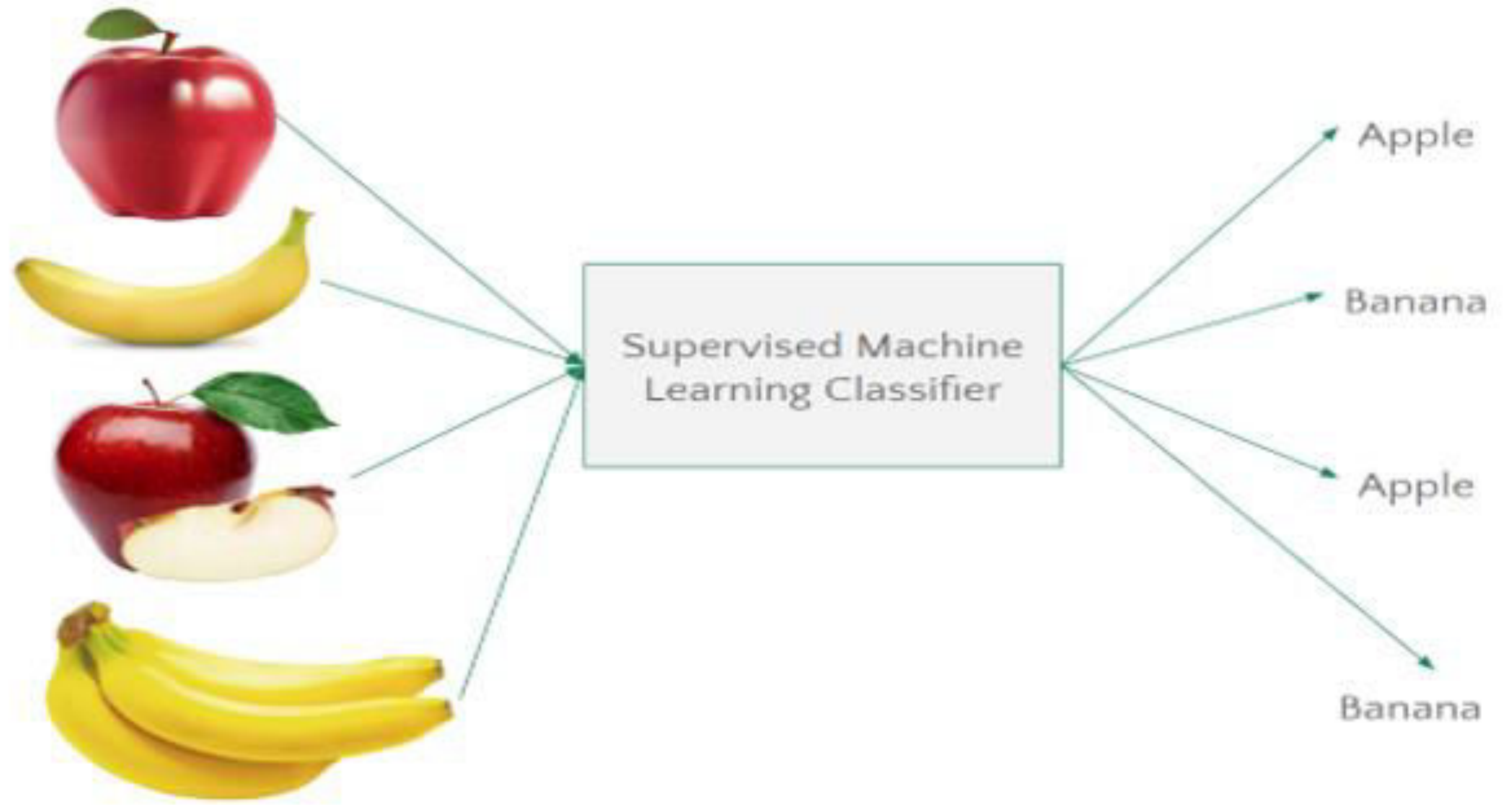
Classifier	Accuracy	Precision	Recall	f1-score	AUC
SGD	0.94	0.58	0.5	0.53	0.87
Random Forest	0.984	0.99	0.82	0.89	0.996
SVM	0.9914	0.98	0.93	0.956	0.96
KNN	0.9916	0.98	0.93	0.956	0.96



ROC Curve comparison between different models



The machine learning problems can be divided into four major categories - supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning.



So, far we learnt that if we have labels associated with the samples, we can learn to predict the labels for unseen data. For example, if we are given the photos of fruits labeled as apple or banana, using machine learning we can create a model that can predict the fruit name. That

What are unsupervised Algorithms?

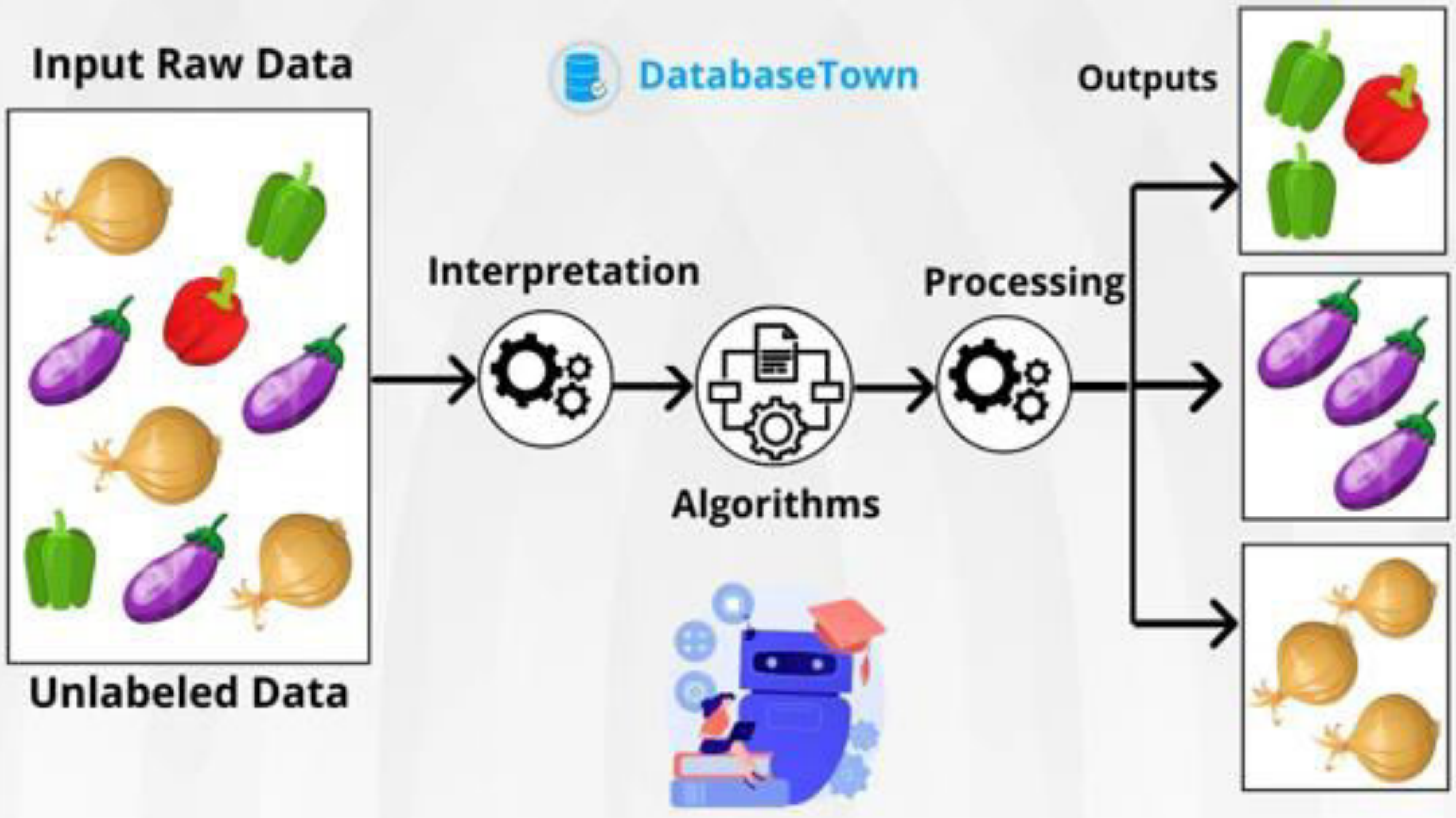
What if we don't have labels?

What if we have just the samples and no label. And using machine learning, we need to train a model that can come up with patterns?

For example, given the photos of fruits, cluster them together based on similarity. Such machine learning problems are called Unsupervised.

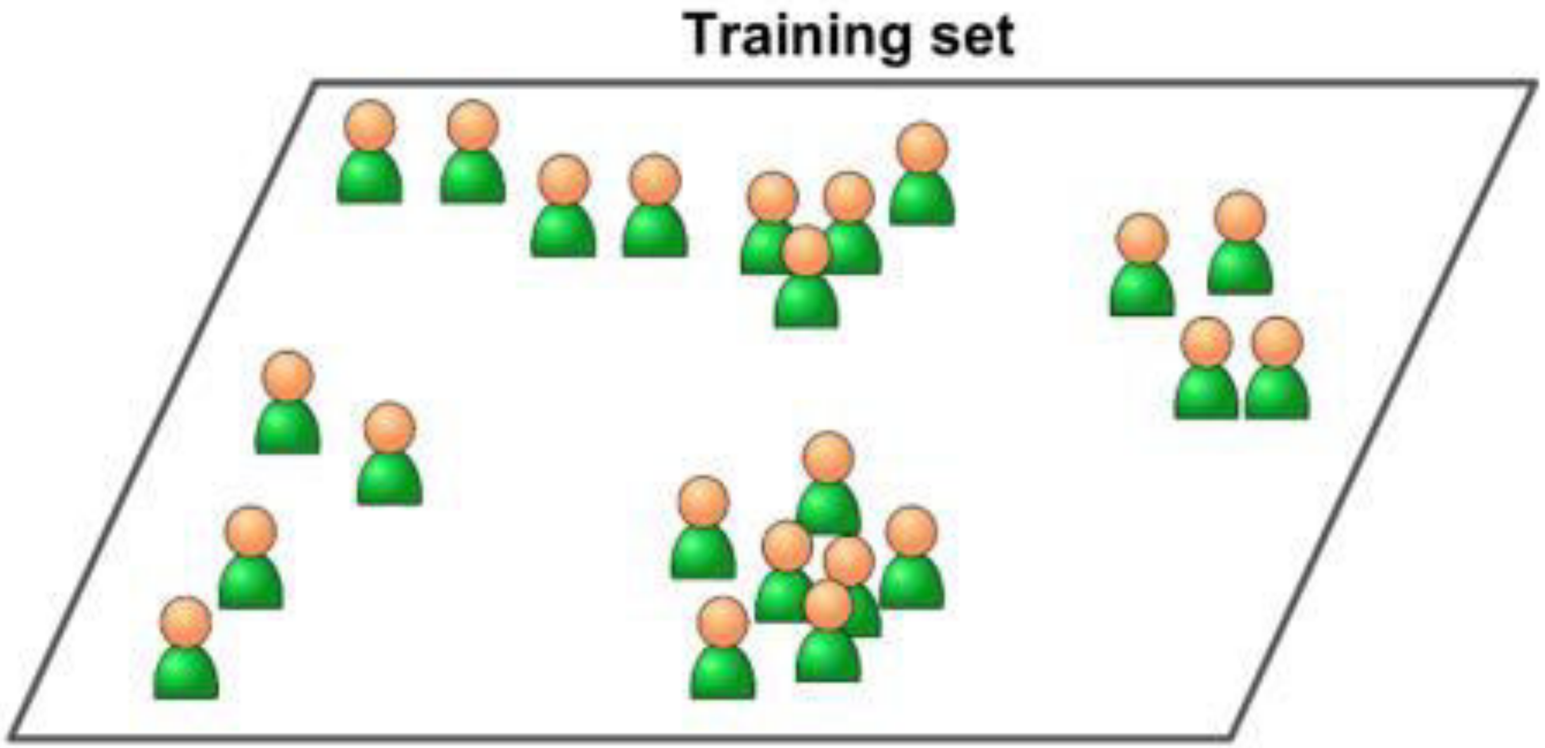
UNSUPERVISED LEARNING

Unsupervised learning is a type of machine learning where the algorithm learns from unlabeled data without any predefined outputs or target variables.



Where are unsupervised Algorithms used ?

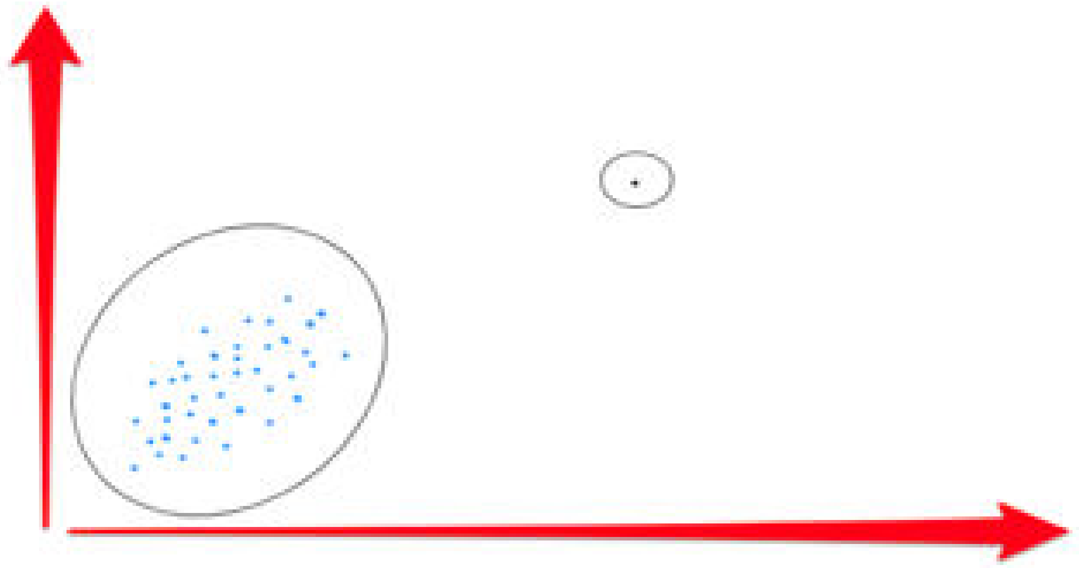
1. Finding Patterns in User behaviour:- **Customer Segmentation** means the segmentation of customers on the basis of their similar characteristics, behavior, and needs.



Unsupervised machine learning could be used in customer segmentation meaning based on the past purchase history, we can come up with various groups of the users such that in each group one user's behavior is similar to other user while its behavior is different from the user from other group.

Where are unsupervised Algorithms used ?

2. Anomaly Detection:- **Anomaly detection is a process of finding those rare items, data points, events, or observations that make suspicions by being different from the rest data points or observations.** Anomaly detection is also known as **outlier detection**.



We can also say in another words that the other use case of unsupervised learning algorithm is Anomaly detection. A real application is fraud detection. If you have lots of transactions with the various details like amount, location, timezone, frequency of purchase, type of purchase and we want to find out which of the transactions are fraud, this is called Anomaly detection which is an example of unsupervised learning. Note that we don't have transactions labels as fraud and genuine.

Where are unsupervised Algorithms used ?

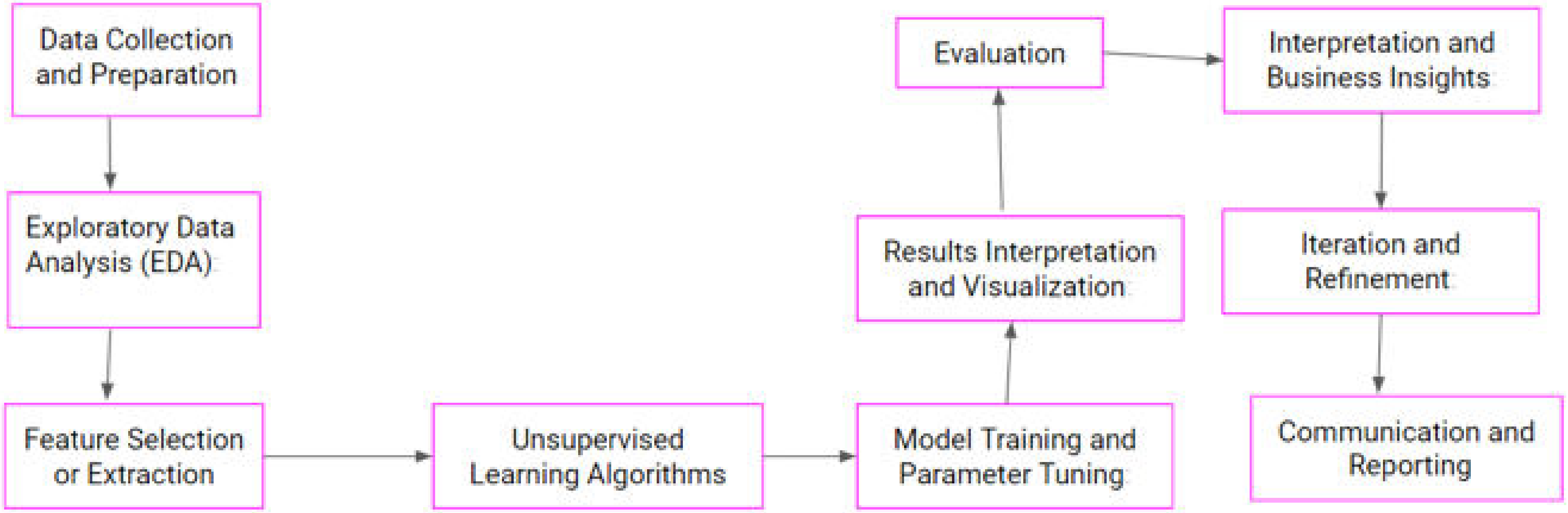
3. Reducing Data - dimensionality reduction.

One very less popular use case of **unsupervised learning algorithms** is **dimensionality reduction**.

Given a lot of attributes of data, if we want drop some attributes or compress some attributes, we can use dimensionality reduction. There are quite a few algorithms which do the dimensionality reduction such as PCA or principle component analysis.

The way these algorithms works is by observing which of the attributes are less important and coming up with the new smaller set of attributes which are more significant. When we are taking a picture, we are reducing data meaning we are converting 3d object in 2d object - what we generally do is we take the picture from the side where we get maximum information. Similarly, these algorithms reduce the dimension such that new dimensions or features or attributes .

Unsupervised Learning Analysis Process


Data Collection and Preparation
Exploratory Data Analysis (EDA)
Feature Selection or Extraction
Unsupervised Learning Algorithms
Model Training and Parameter Tuning
Results Interpretation and Visualization
Evaluation
Interpretation and Business Insights
Iteration and Refinement
Communication and Reporting

Main types of unsupervised learning problems:

- Clustering
- Dimensionality Reduction

Clustering

1. The goal is to group similar instances together into clusters.
2. This is a great tool for data analysis,
3. Customer segmentation, recommender systems, search engines, image segmentation, semi-supervised learning, dimensionality reduction, and more.

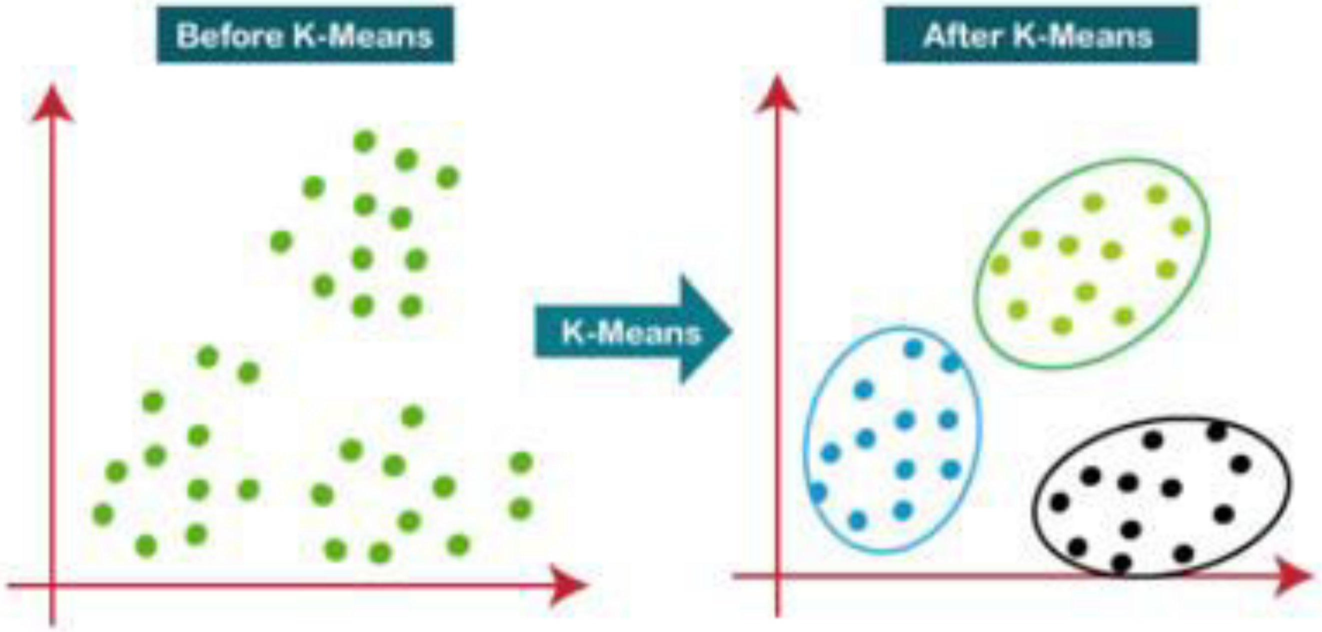
How to find patterns in user behaviour?

Use clustering algorithm such as KMeans

The question is how to find the patterns in user behavior? The main models that are used in unsupervised learning for clustering are Kmeans.

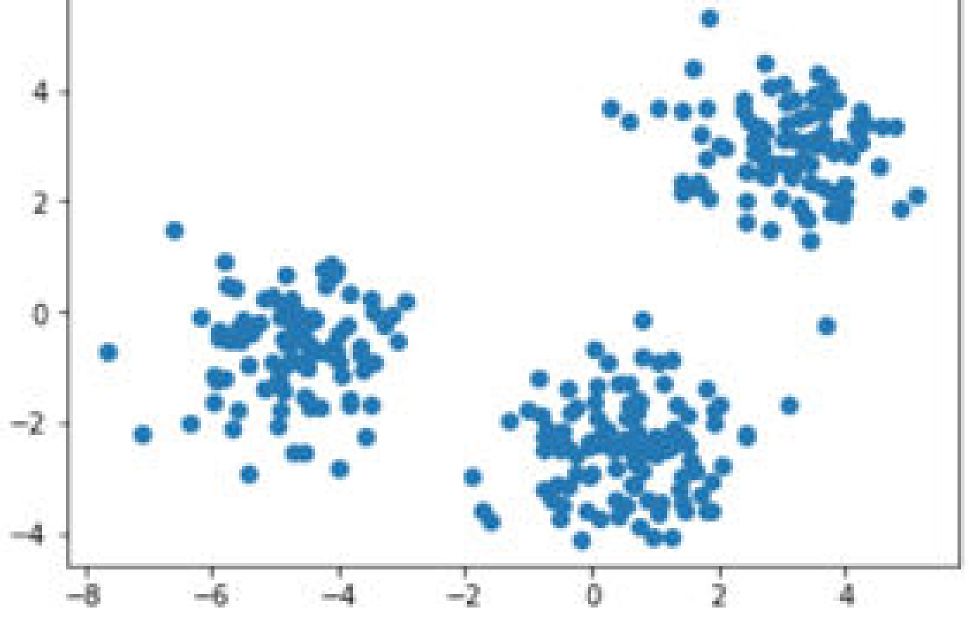
K-Means Algorithm:-

K-Means Clustering may be the most widely known clustering algorithm and involves assigning examples to clusters in an effort to minimize the variance within each cluster.



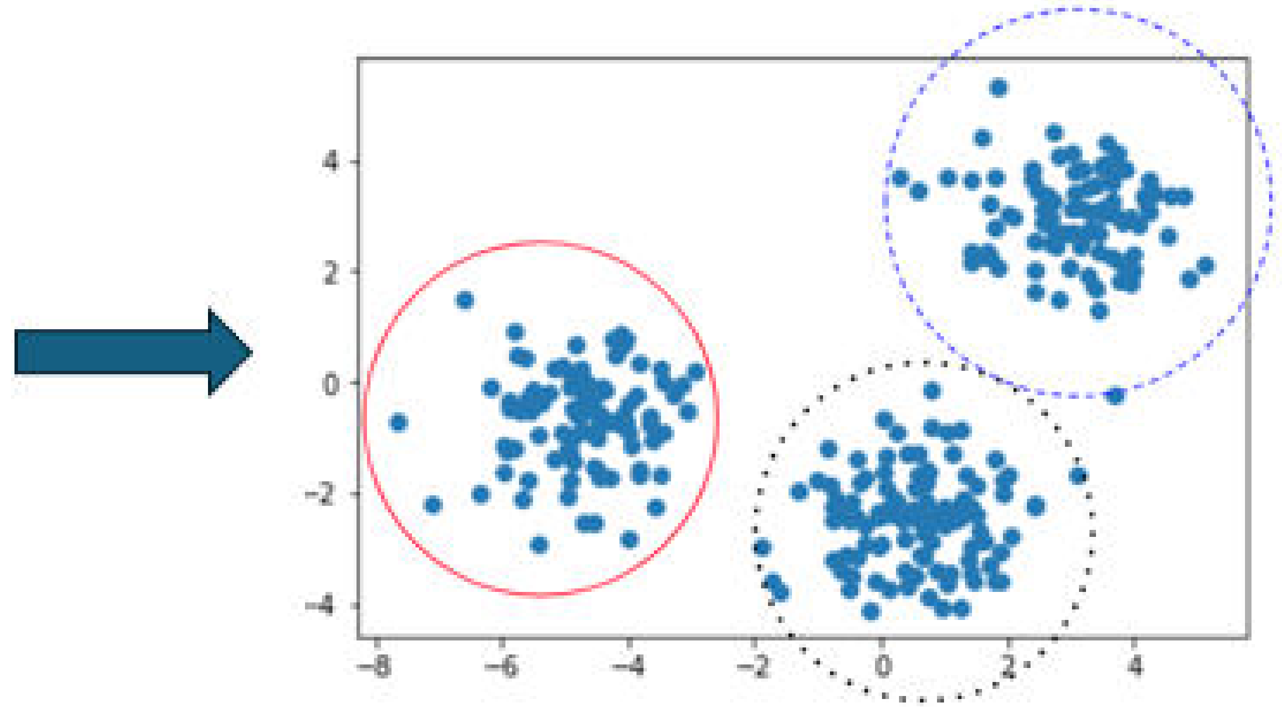
Clustering – KMeans

Let see clusters



That's right we have three clusters.

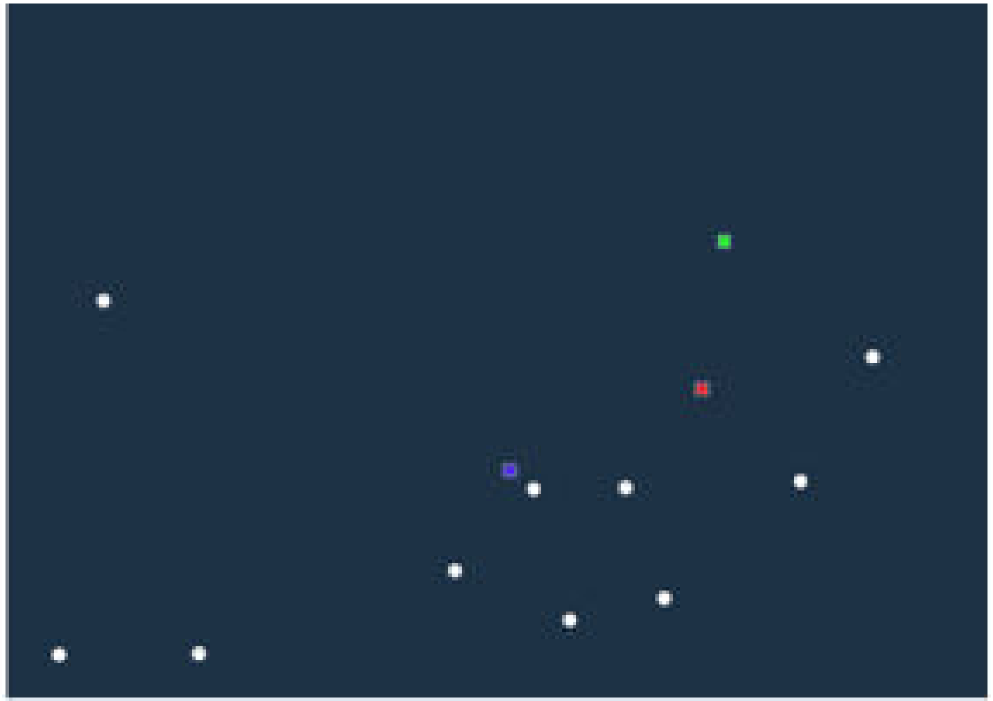
Let see clusters



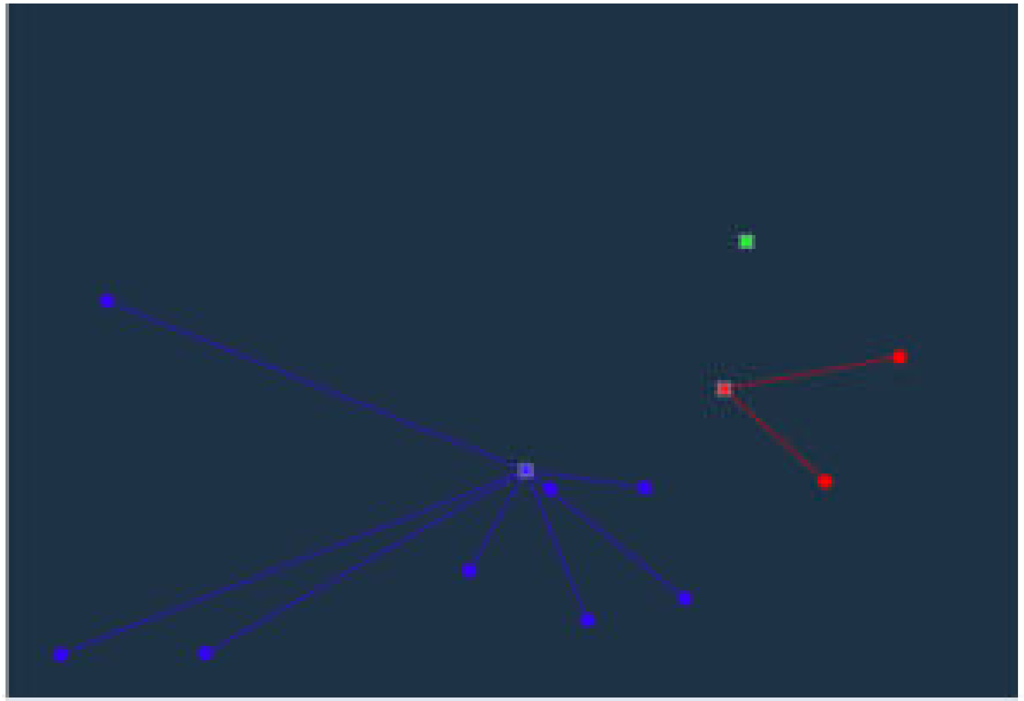
Using KMeans algorithm we can group the data points in the given buckets. So, if we want to group this data we can call KMeans library with number of group. We can group these data points into 3 or 2 or 4 and so on.

Clustering – KMeans

Initialize – 10 samples and 3 centroids



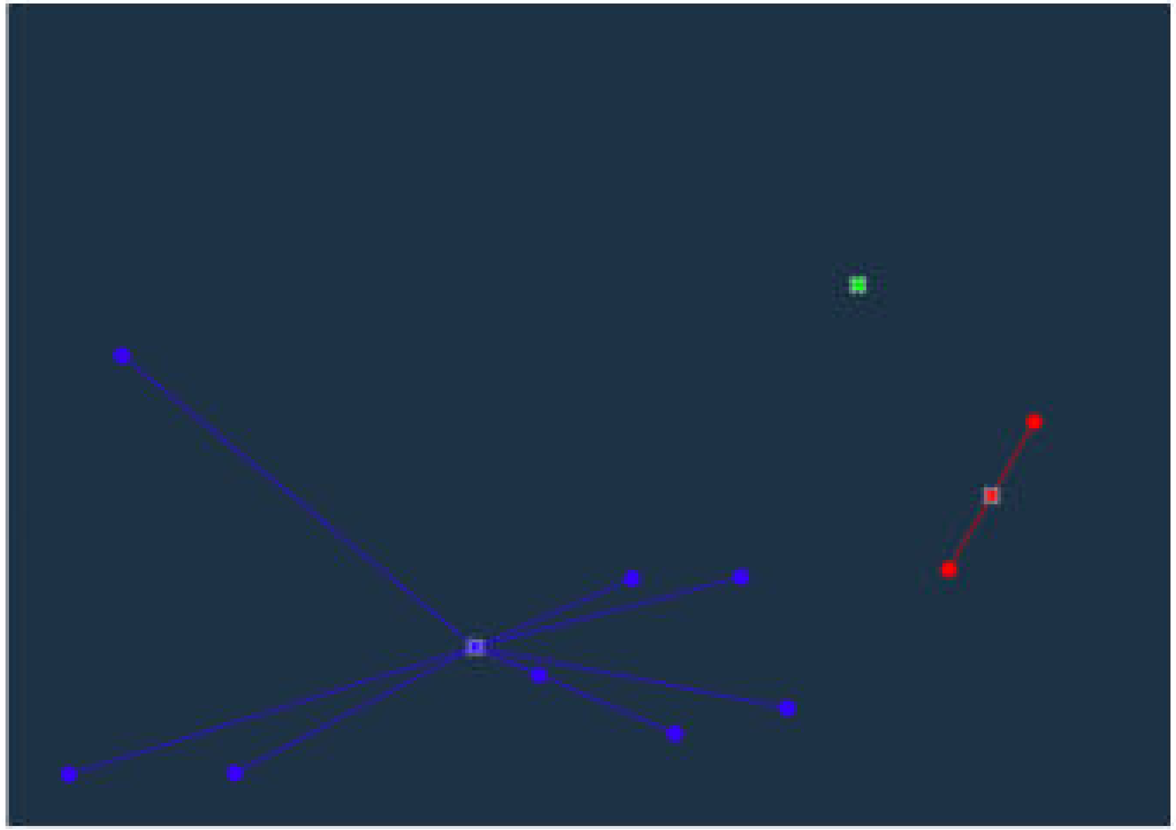
we are trying to group these 10 data points or observations into 3 groups.



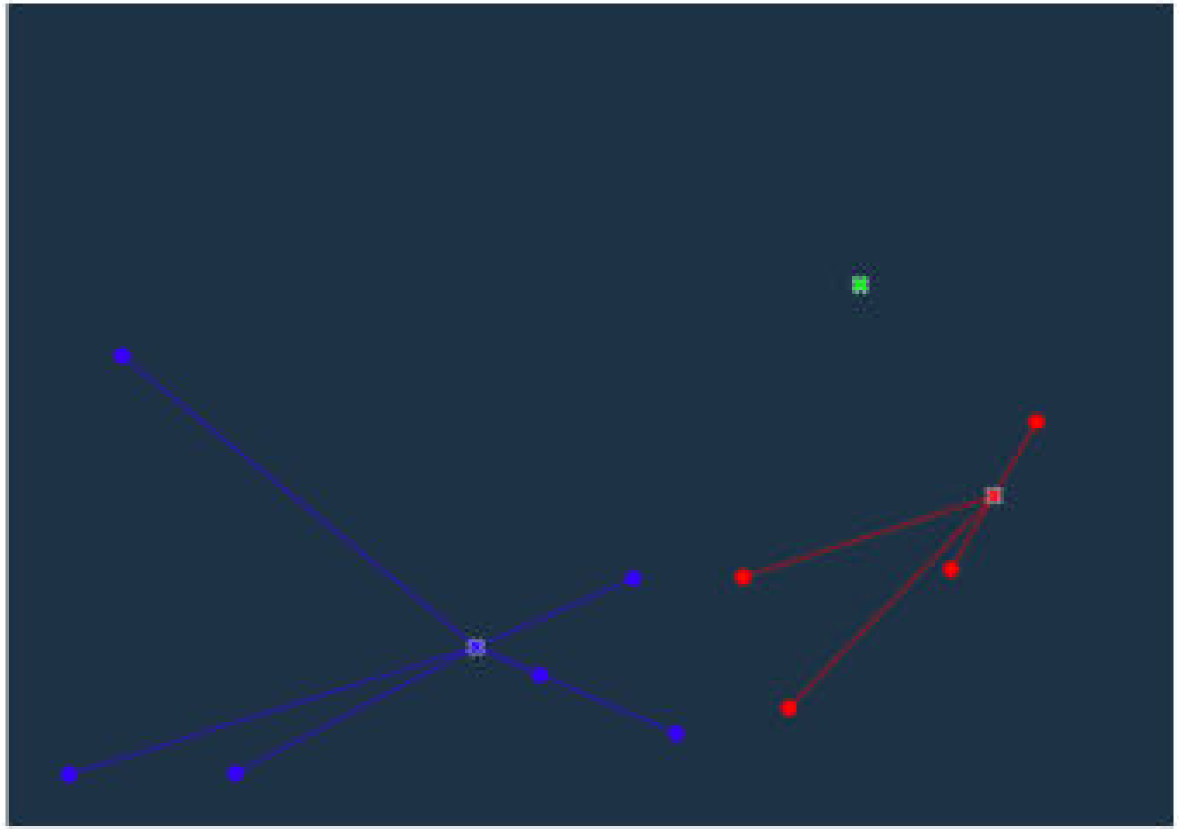
Now we will go over each of the 10 instances or you can say samples and for each these instances we calculate the distance from the centroids and we assign the instance to the corresponding group.

We have right now 8 instances in blue group, two instances in red and no instance in green group.

Clustering – KMeans

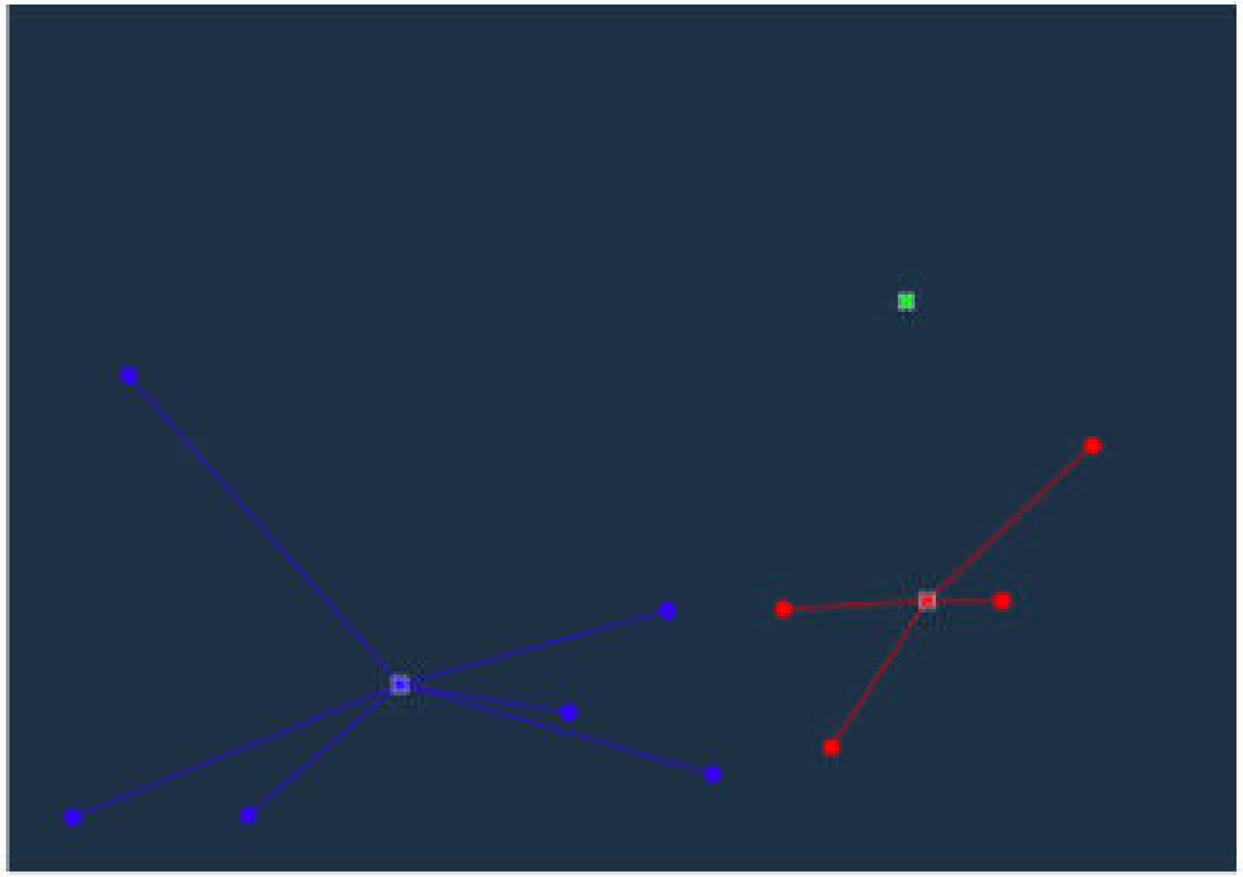


And once we are done assigning the groups, we recalculate the centers or centroids based on the average value of x and y coordinates of the points in the group. You can see that the new points marked with cross are the new centers.

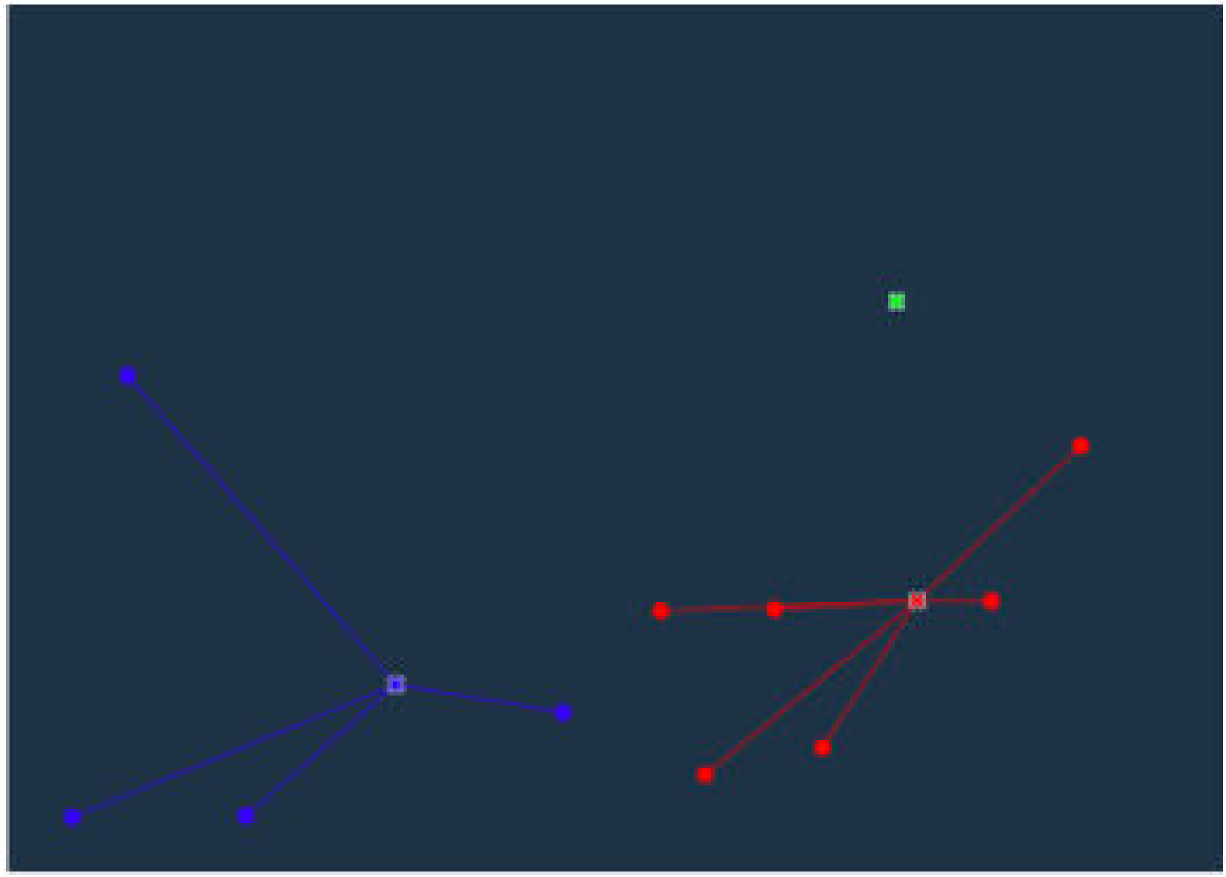


Afterwards we restart the process of grouping the instances based on the distances from various centers. You can see that two point have moved from blue group to red group. As of now, there are 6 instances in blue group, 4 instances in red group and no instance in green.

Clustering – KMeans

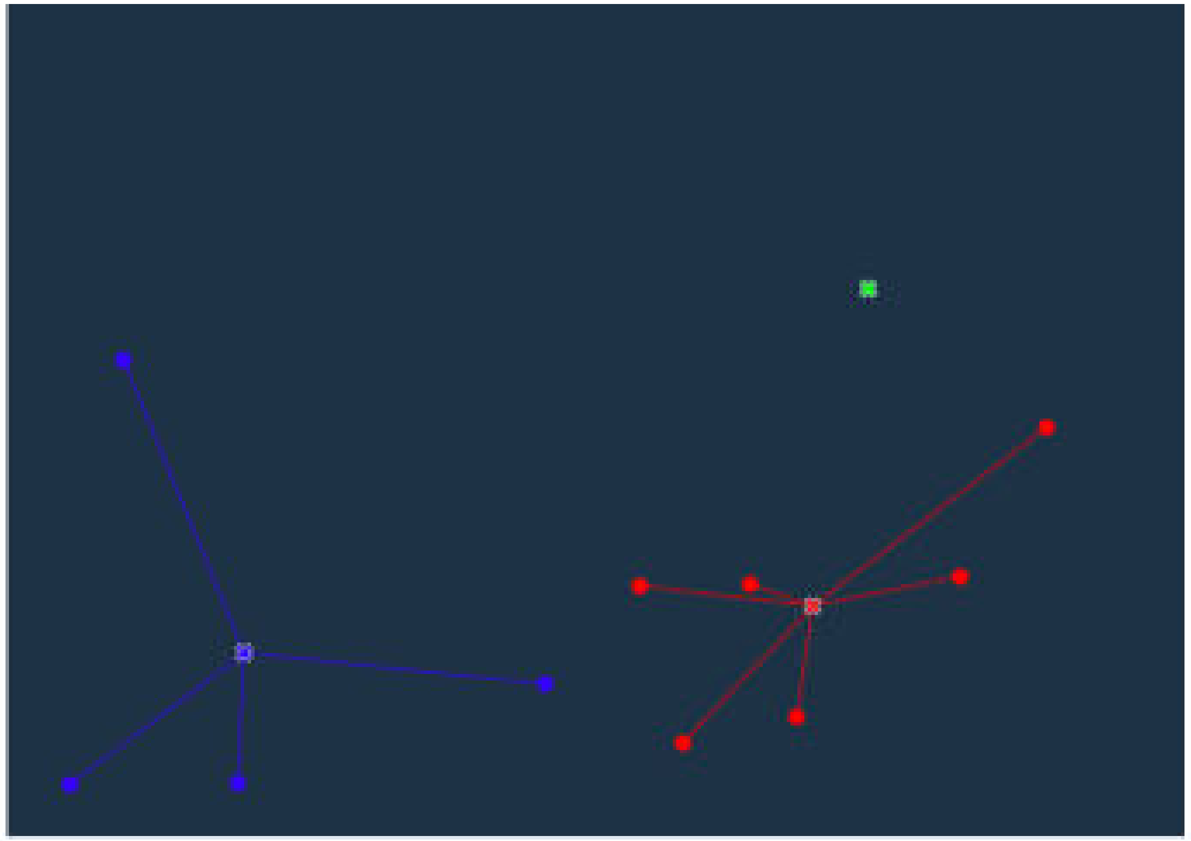


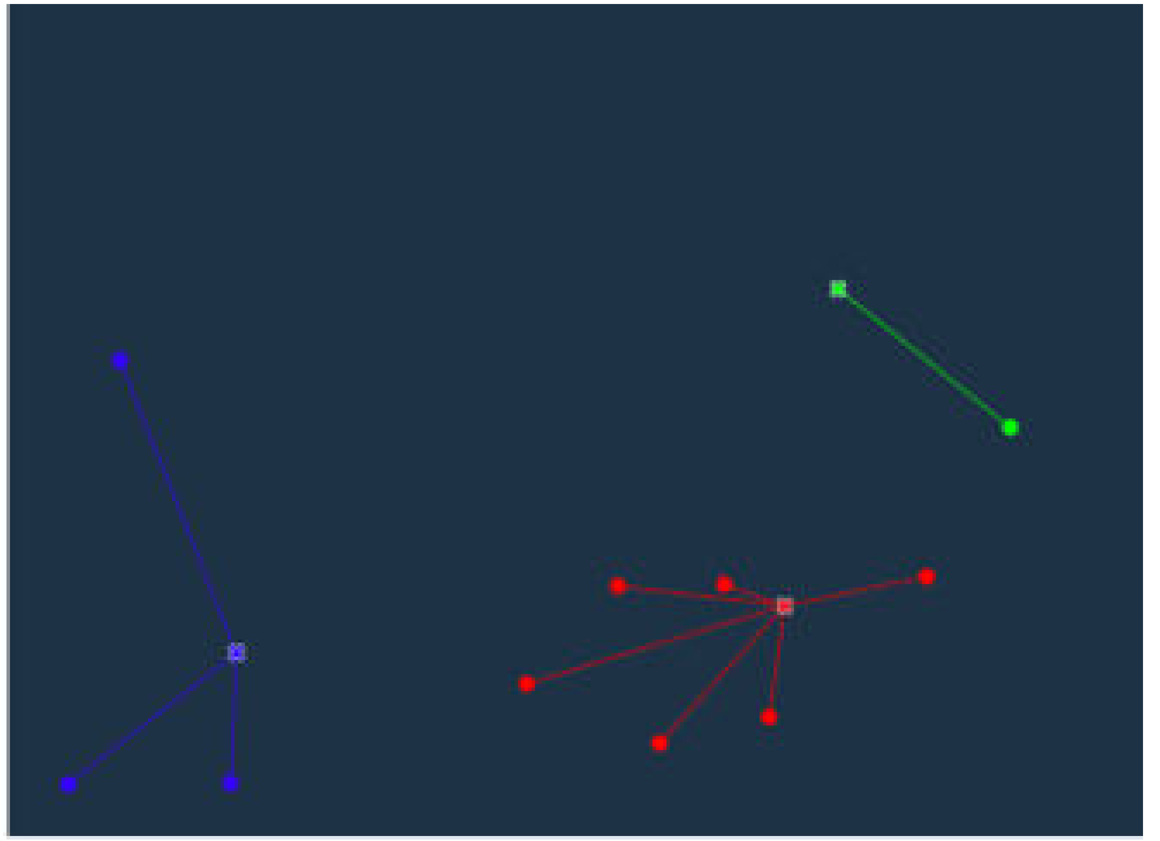
And once we are done we again recalculate the centers based on the instances in each group.



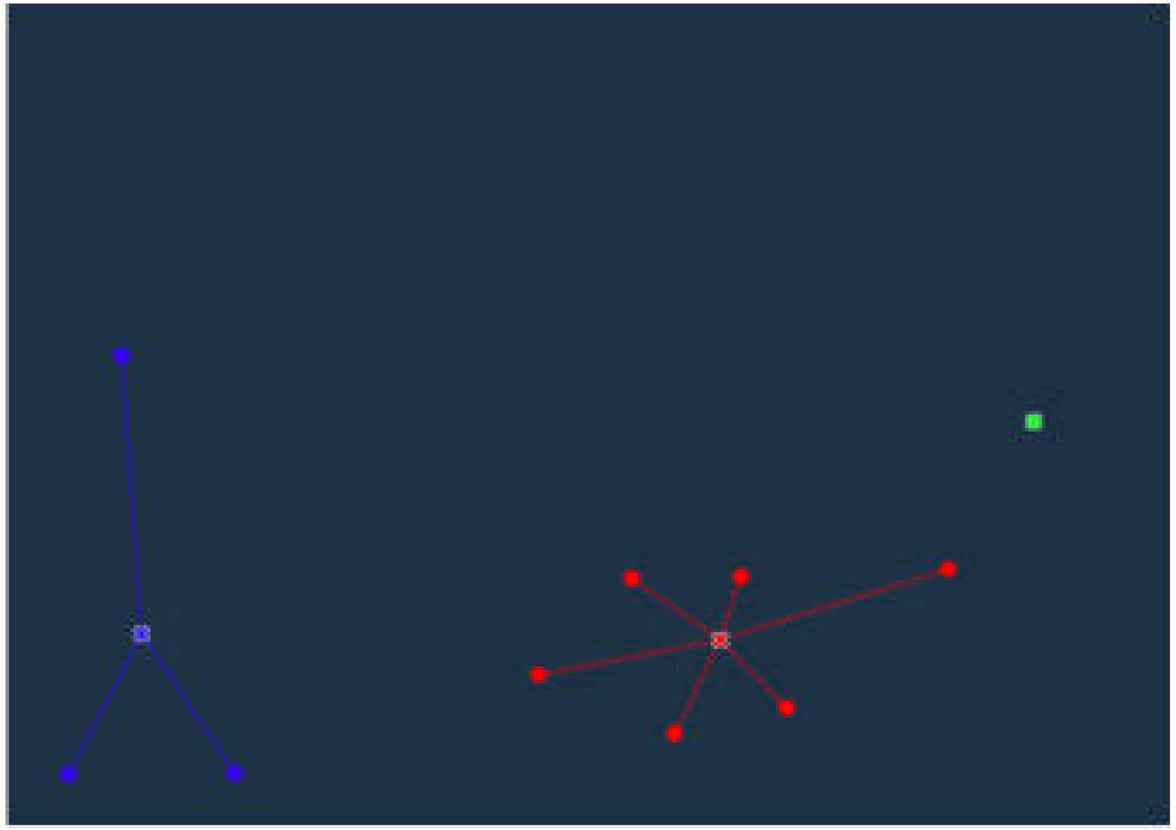
And again in the next iteration we form groups based on the distances from the new centers.

Clustering – KMeans

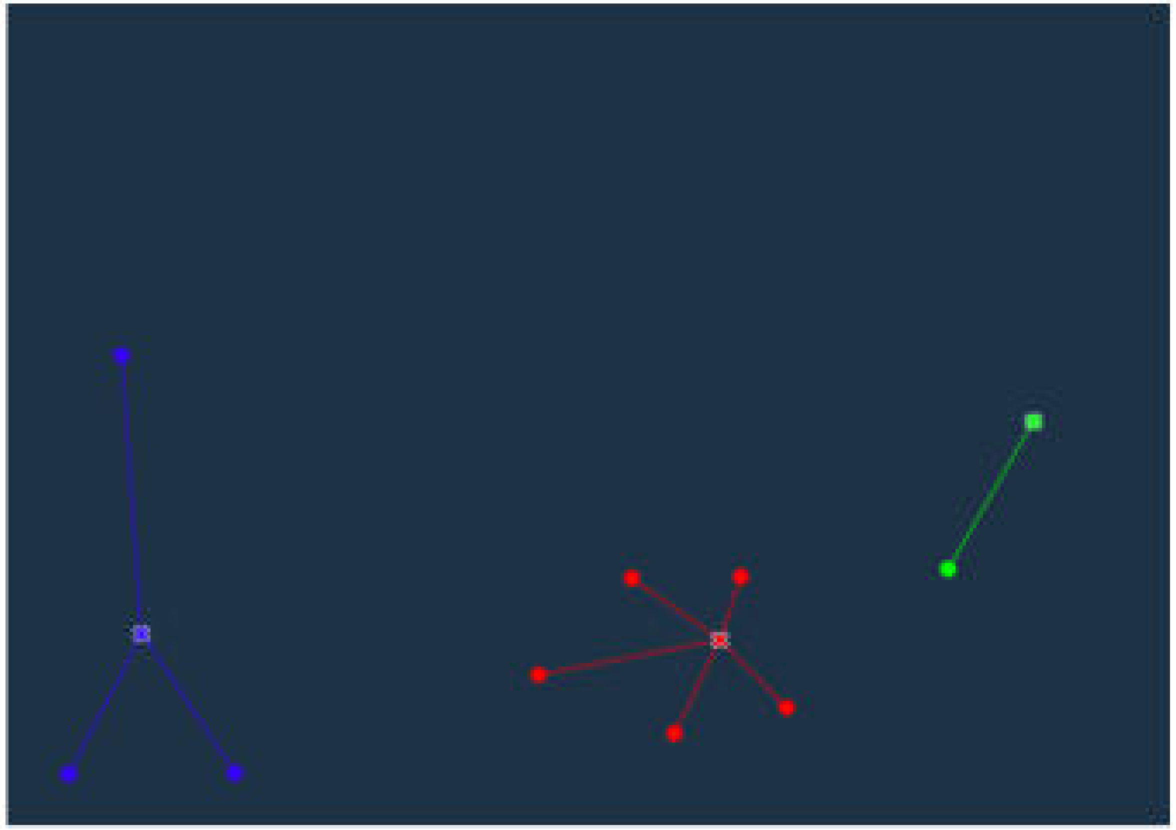

and again re-compute the centers.


Again we calculate the distances from the new centers. And this time one of points have moved from red group to green group.

Clustering – KMeans

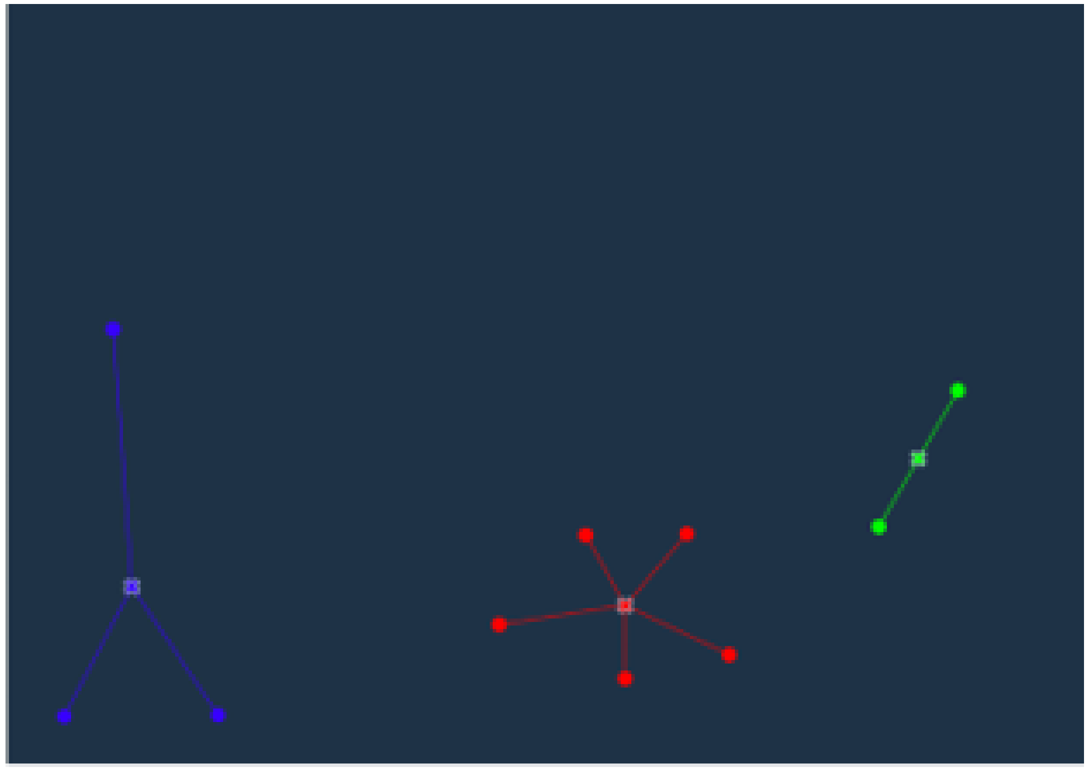


Recalculate the centers. Notice that in case of green centroid, there is only a single point therefore the point is coinciding with the centroid.

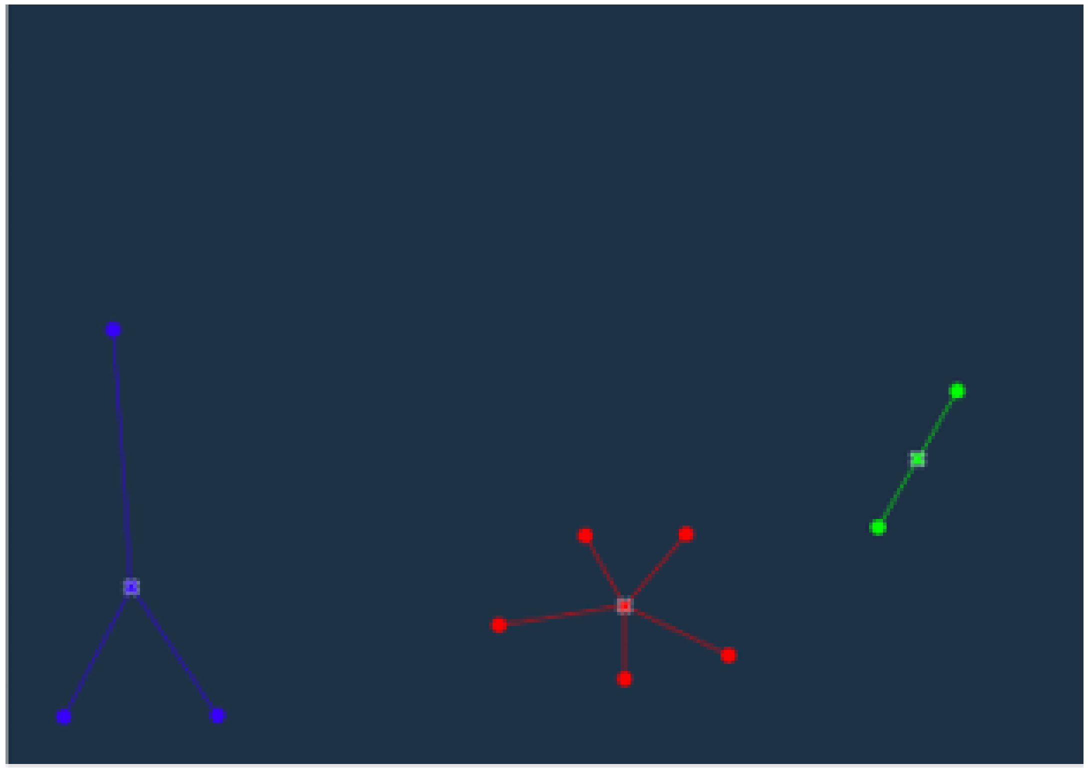


And again we regroup the points based on the distances. This time, one more point moved from the red to green.

Clustering – KMeans



We recalculate the center or centroids.



We repeat this process over and over untill a certain number of iterations or the re-grouping is no longer happening.

KMeans
How many groups should we have?

- Field knowledge
- Business decision
- Elbow Method

There are three way. First, we know from the knowledge of domain that this problem has these many clusters. The second way is that we actually just want only certain number of the clusters as part of our business objectives.

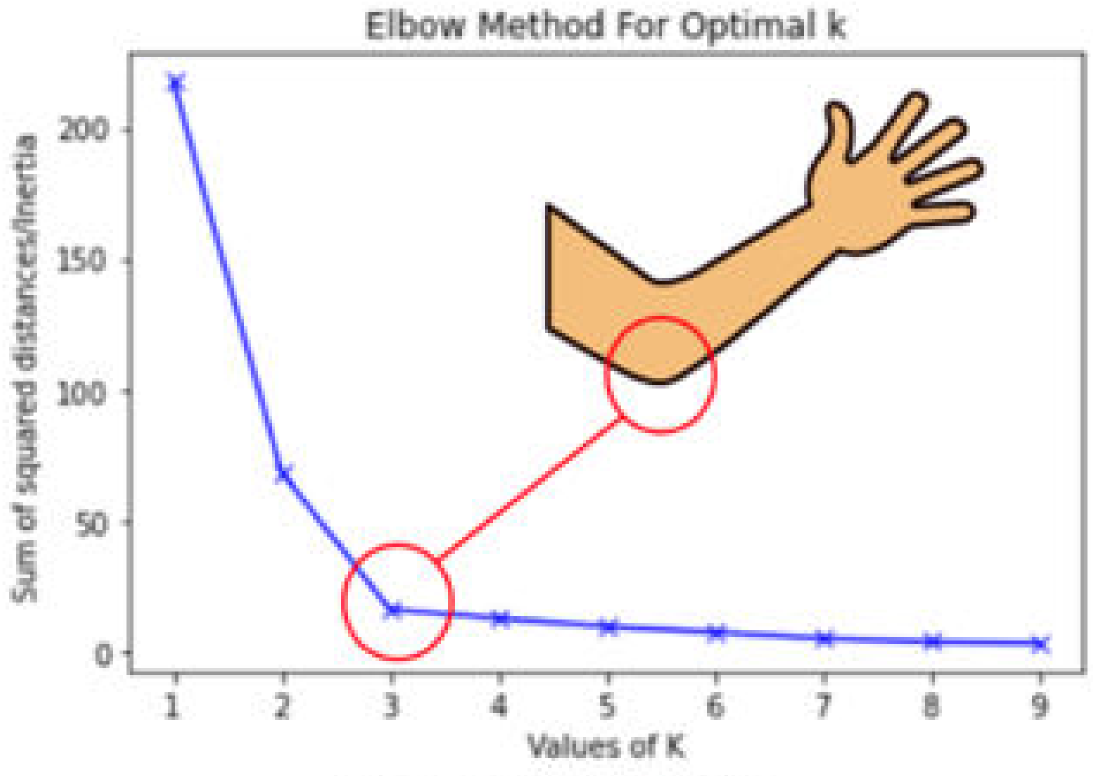
KMeans
Percentage of variance explained?

$$\% \text{ Variance} = \frac{\text{Variance between groups}}{\text{Total variance}}$$

Before proceeding with the elbow method, we need to understand the % variance explained by the number of groups. We basically calculate the variance between the groups and overall variance in the data and divide the two. This gives the % variance.

Elbow Method:-

1. The elbow method is a popular technique used in clustering analysis to determine the optimal number of clusters for a dataset.
2. It helps you find the point at which increasing the number of clusters no longer significantly improves the quality of clustering.



Line plot between K and inertia

How it works:

1. Select a range of candidate cluster numbers

2. Apply the clustering algorithm

3. Calculate the Within-Cluster Sum of Squares (WCSS)

4. Plot the WCSS values

5. Identify the "elbow" point

6. Select the optimal number of clusters

Select a range of candidate cluster numbers: To begin, you need to decide on a range of possible cluster numbers (e.g., from 1 to some upper limit).