BCAC 0017
FUNDAMENTALS OF MACHINE LEARNING

Section 5 : Validation

Faculty Name
Mr. Sachin Sharma, Dr. Vinod Jain, Dr. Manu Banga,
Ms.Paromita Goswami, Ms.Chestha Bhardwaj

Section 5 : Validation

- **Validation:** True and sample error, over-fitting, role of cross validation, regularization, bias-variance analysis.

Section 5 : Validation

- **Validation:**
- True and sample error
- over-fitting
- role of cross validation
- regularization
- bias-variance analysis

Validation:

- Model validation is the process that is carried out after Model Training where the trained model is evaluated with a testing data set.

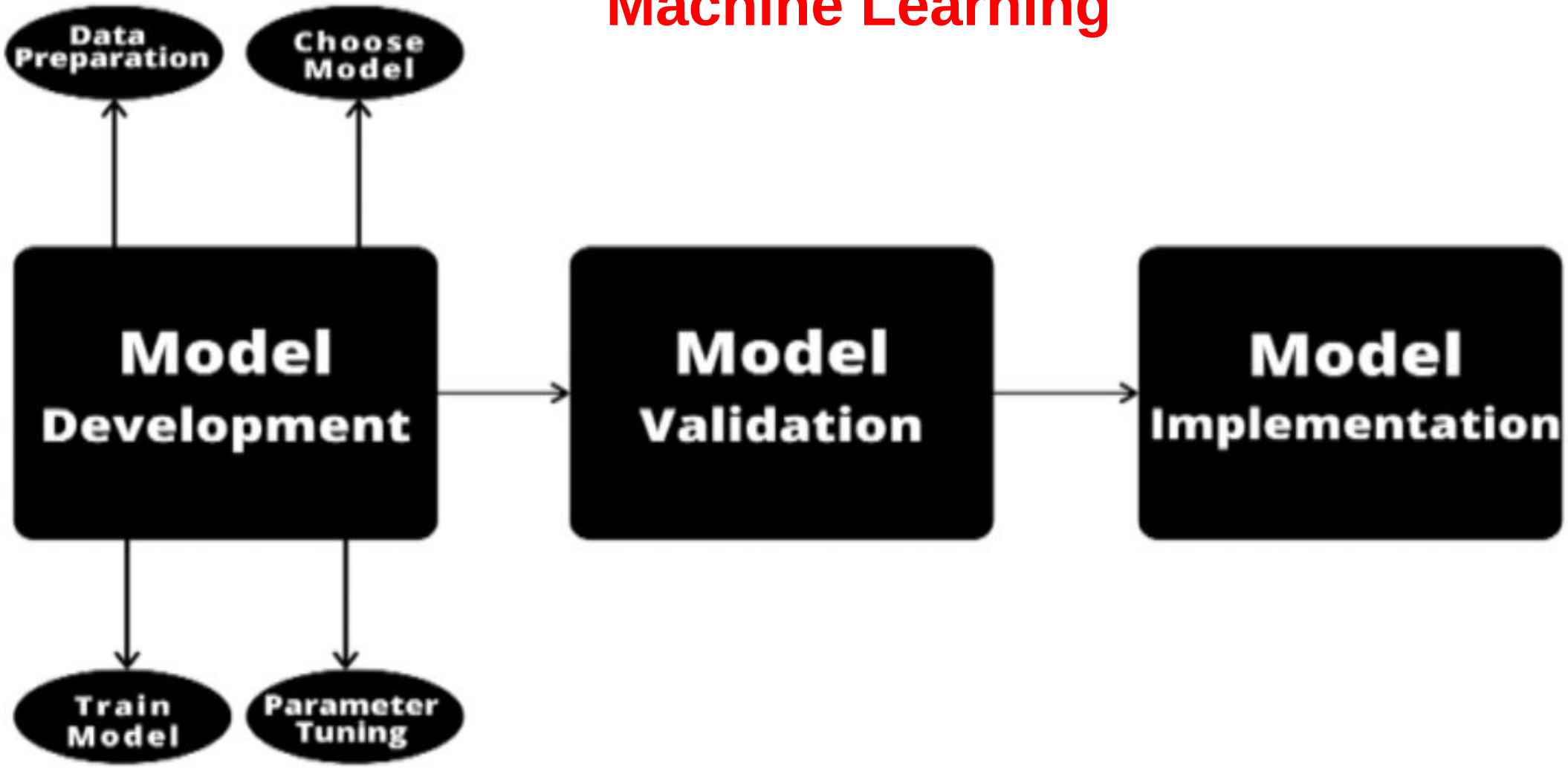
Validation:

- *'Machine Learning is about making predictions.'*

- These predictions come up after assorted processes like

1. Data Preparation,
2. Choosing a Model,
3. Training the Model,
4. Parameter Tuning,
5. Model Validation, etc.

Machine Learning



Validation:

- ***Model validation is the process that is carried out after Model Training where the trained model is evaluated with a testing data set.***
- The testing data may or may not be a chunk of the same data set from which the training set is procured.

Validation:

- The two types of Model Validation techniques are namely,
- **In-sample validation** – *testing data from the same dataset that is used to build the model*.
- **Out-of-sample validation** – *testing data from a new dataset that isn't used to build the model*

Validation:

- Model validation is the process that is carried out after Model Training where the trained model is evaluated with a testing data set.

- Model validation refers to the process of confirming that the model achieves its intended purpose i.e., how effective our model is.

Model Validation Techniques

- Train/test split
- k-Fold Cross-Validation
- Leave-one-out Cross-Validation
- Leave-one-group-out Cross-Validation
- Nested Cross-Validation
- Time-series Cross-Validation
- Wilcoxon signed-rank test
- McNemar's test
- 5x2CV paired t-test
- 5x2CV combined F test

Validation:

- There are various ways of validating a model among which the two most famous methods are Cross Validation and Bootstrapping.
- But there is no single validation method that works in all scenarios.
- Therefore, it is important to understand the type of data we are working with.

References

- https://datatron.com/what-is-model-validation-and-why-is-it-important/

True and sample error

- The true error represents the probability that a randomly drawn instance from the entire distribution is misclassified while the sample error is the fraction of sample which is misclassified.

True error

- The true error represents the probability that a randomly drawn instance from the entire distribution is misclassified .

- As true error represents entire population it becomes difficult to calculate hence we use sample to check our hypothesis.

True error

- The true error can be said as the probability that the hypothesis will misclassify a single randomly drawn sample from the population.
- Here the population represents all the data in the world.
- Let's consider a hypothesis h(x) and the true/target function is f(x) of population P.
- The probability that h will misclassify an instance drawn at random i.e. true error is:

$$T.E. = Prob[f(x) \neq h(x)]$$

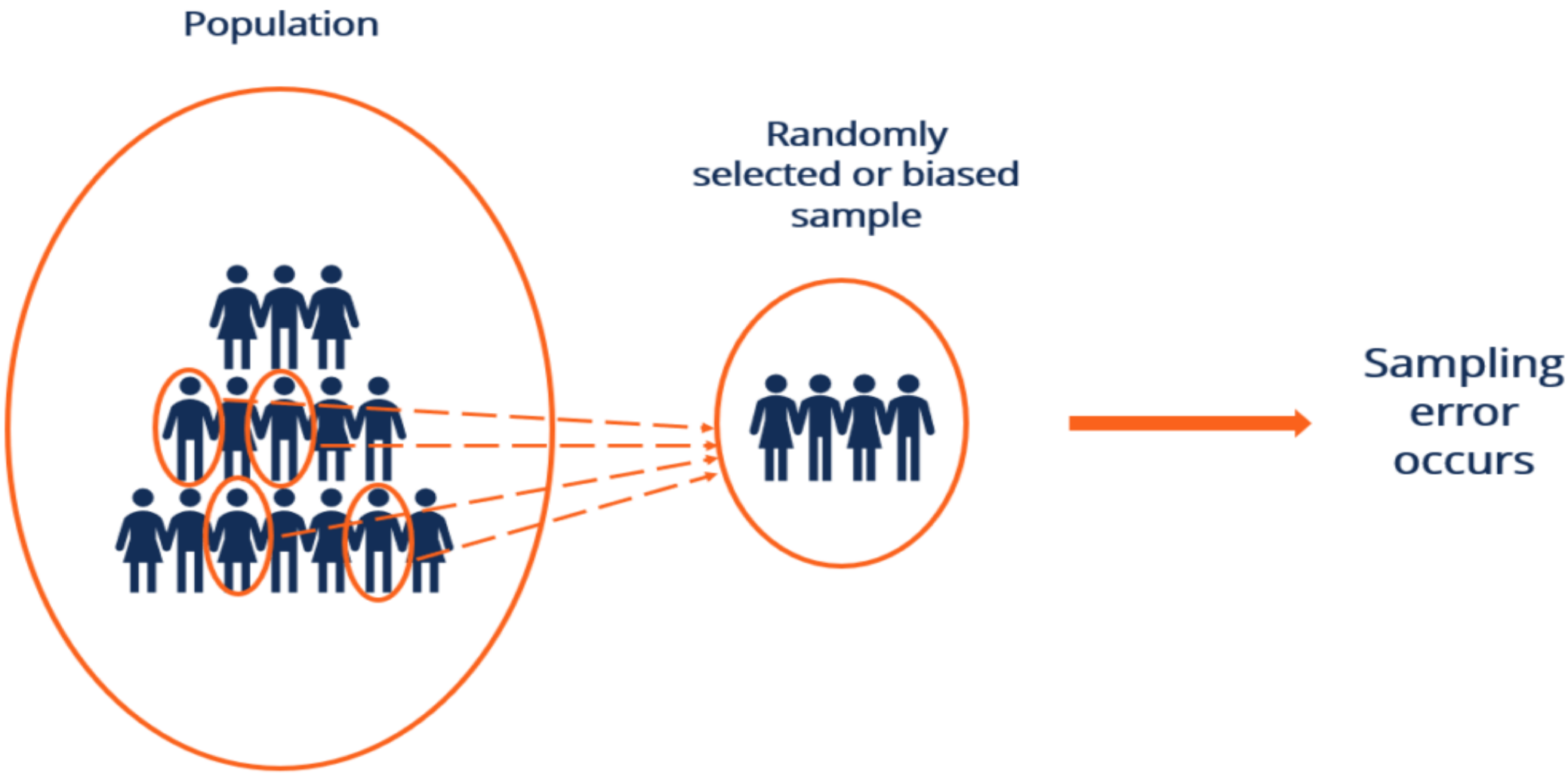
Sampling Error

['sam-pliŋ 'er-ər]

A statistical error that occurs when the analyst selects a sample that is not representative of the population being studied.

sample error

- **What are Sampling Errors?**
- Sampling errors are statistical errors that arise when a sample does not represent the whole population.
- They are the difference between the real values of the population and the values derived by using samples from the population.
- Since there is a fault in the data collection, the results obtained from sampling become invalid.


Population
Randomly selected or biased sample
Sampling error occurs

Sample error

- The sample error of S with respect to target function f and data sample S is the proportion of examples S misclassifies.

$$S.E. = \frac{1}{n} \sum_{x \epsilon S} \delta(f(x) \neq h(x))$$

$$Sample\ Error = \frac{Number\ of\ missclassified\ instances}{Total\ Number\ of\ Instance}$$

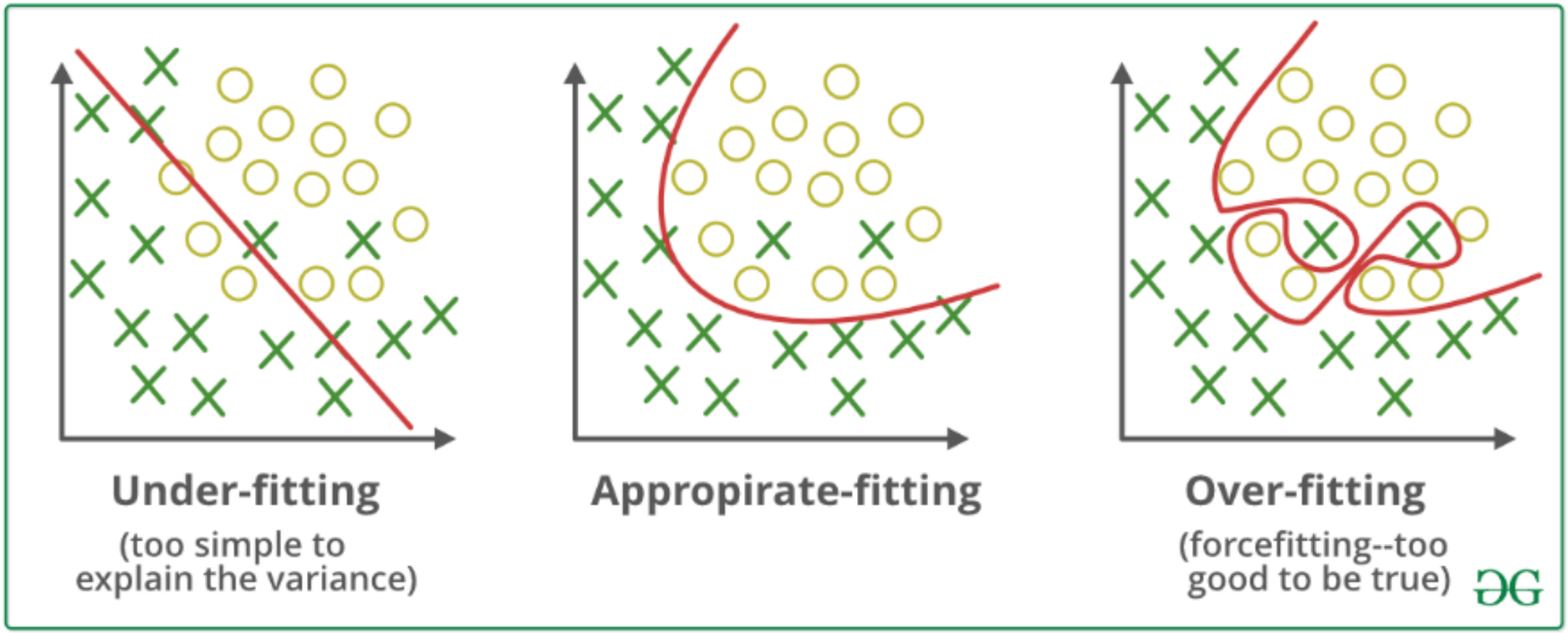
True Error	Sample Error
True error is used to estimate the error of the population.	Sample Error is used to estimate the errors of the sample.
The true error represents the probability that a random sample from the population is misclassified.	Sample Error represents the fraction of the sample which is misclassified.
True error is difficult to calculate. It is estimated by the confidence interval range on the basis of Sample error.	Sample Error is easy to calculate. You just have to calculate the fraction of the sample that is misclassified.
The true error can be caused by poor data collection methods, selection bias, or non-response bias.	Sampling error can be of type population-specific error (wrong people to survey), selection error, sample-frame error (wrong frame window selected for sample), and non-response error (when respondent failed to respond).

References

- https://www.geeksforgeeks.org/true-error-vs-sample-error/
- https://corporatefinanceinstitute.com/resources/data-science/sampling-errors/

Under-fitting

- It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data.
- In simple terms, an underfit model's are inaccurate.
- It mainly happens when we uses very simple model with overly simplified assumptions.



Underfitting and Overfitting

Under-fitting

- A statistical model or a machine learning algorithm is said to have underfitting when a model is too simple to capture data complexities.

- To address underfitting problem of the model, we need to use more complex models, with enhanced feature representation.

- **Note: The underfitting model has High bias and low variance.**

Under-fitting

- **Reasons for Underfitting**

1. The model is too simple, So it may be not capable to represent the complexities in the data.
2. The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
3. The size of the training dataset used is not enough.
4. Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.
5. Features are not scaled.

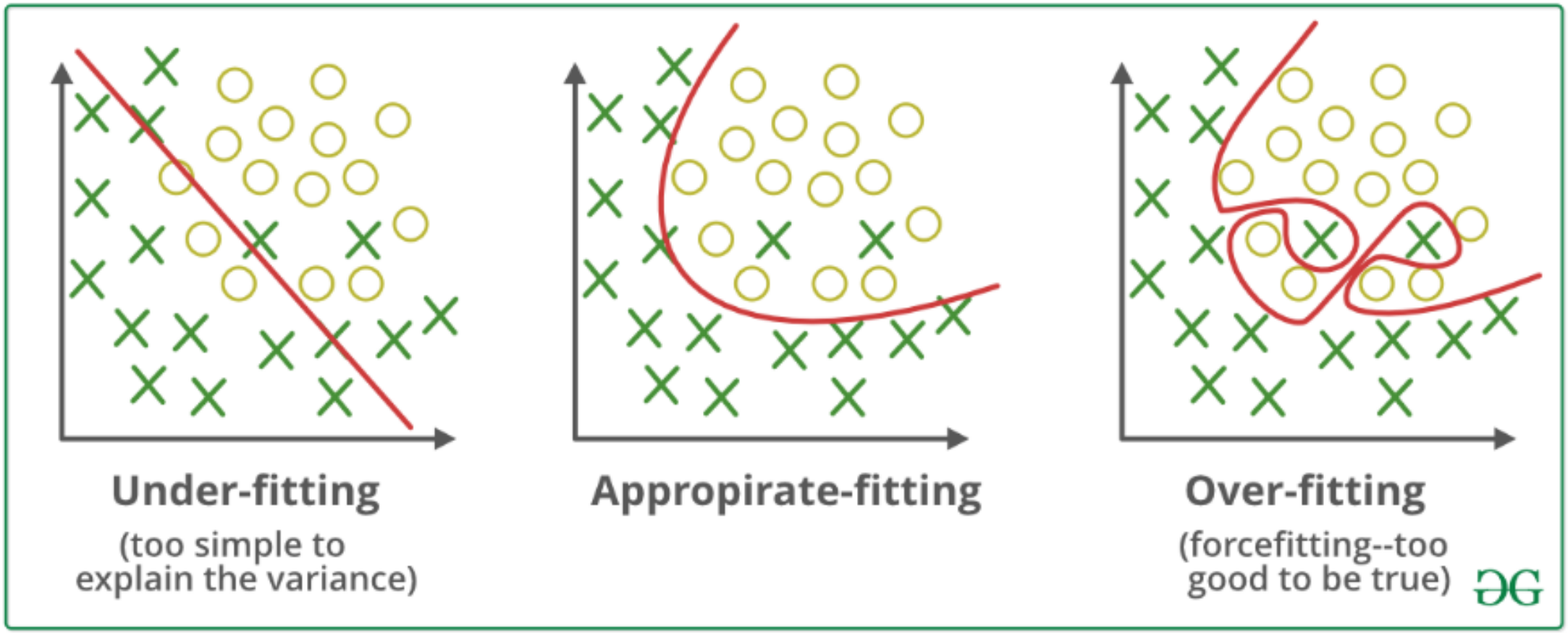
Under-fitting

- **Techniques to Reduce Underfitting**

1. Increase model complexity.
2. Increase the number of features, performing feature engineering.
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

Over-fitting

- A statistical model is said to be overfitted when the model does not make accurate predictions on testing data.
-
- When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set.



Underfitting and Overfitting

Over-fitting

- Overfitting is a problem where the evaluation of machine learning algorithms on training data is different from unseen data.
- The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really **build unrealistic models.**
-

Over-fitting

- **Reasons for Overfitting:**
- High variance and low bias.
- The model is too complex.
- The size of the training data.

Over-fitting

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

References

- https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Cross Validation

- In practice, we use the following process to calculate the MSE of a given model:
- **1.** Split a dataset into a training set and a testing set.
- **2.** Build the model using only data from the training set.
- **3.** Use the model to make predictions on the testing set

Cross Validation

- However,
- the drawback of using only one testing set is that the error can vary greatly depending on which observations were used in the training and testing sets.

Role Of Cross Validation

- One way to avoid this problem is to fit a model several times using a different training and testing set each time.

- This general method is known as cross-validation and a specific form of it is known as k-fold cross-validation.

Role Of Cross Validation

- Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data.
- ***We can also say that it is a technique to check how a statistical model generalizes to an independent dataset***.

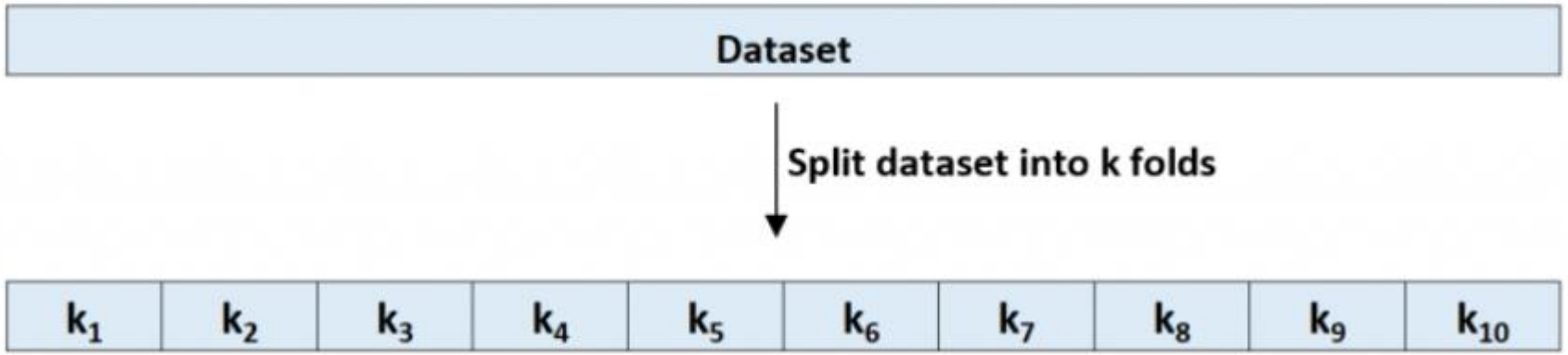
Role Of Cross Validation

- Hence the basic steps of cross-validations are:

1. Step 1: Randomly divide a dataset into *k* groups, or “folds”, of roughly equal size.
2. Step 2: Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.
3. Step 3: Repeat this process *k* times, using a different set each time as the holdout set.
4. Step 4: Calculate the overall test MSE to be the average of the *k* test MSE’s.

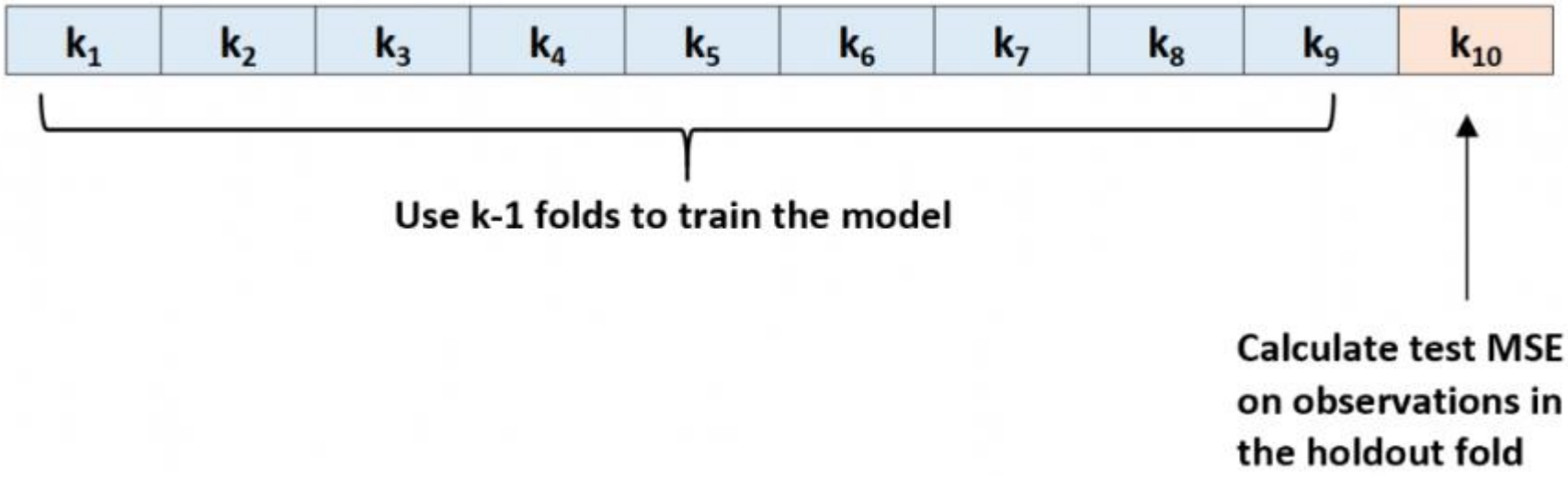
Role Of Cross Validation

Step 1: Randomly divide a dataset into *k* groups, or "folds", of roughly equal size.



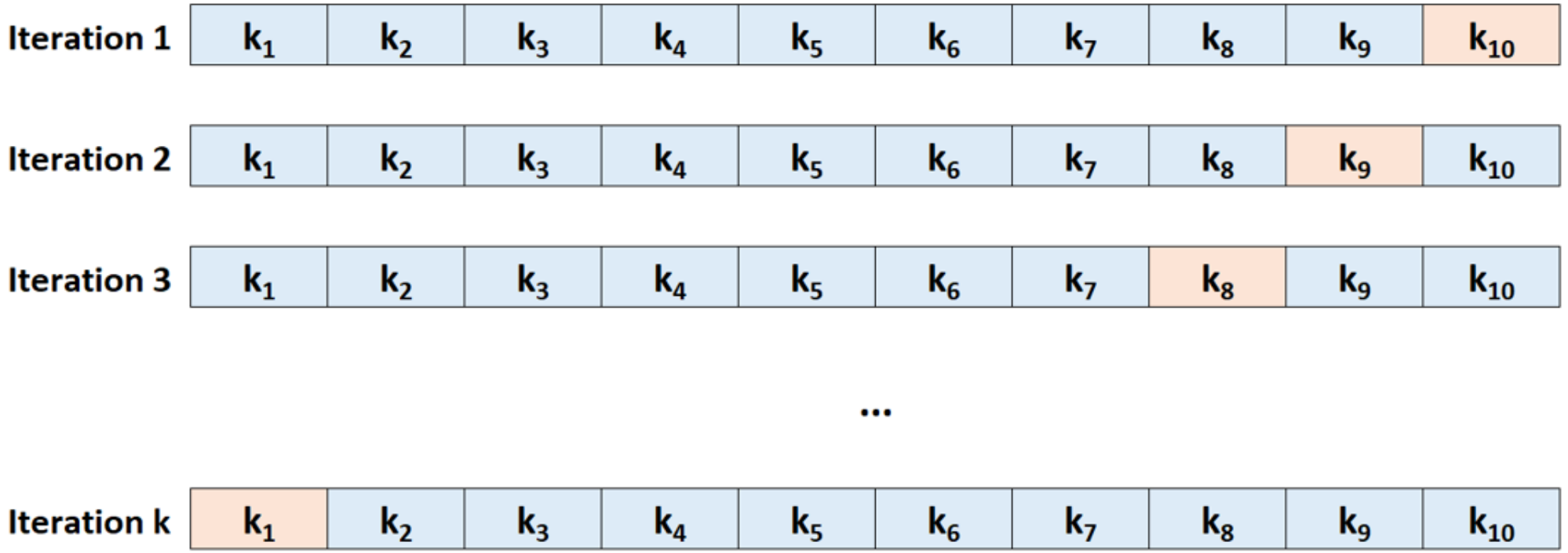
Role Of Cross Validation

- **Step 2: Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.**



Role Of Cross Validation

Step 3: Repeat this process *k* times, using a different set each time as the holdout set.



Role Of Cross Validation

Step 4: Calculate the overall test MSE to be the average of the *k* test MSE's.

Test MSE = $(1/k)*\Sigma MSE_i$

where:

- **k:** Number of folds
- **MSE_i**: Test MSE on the i^{th} iteration

Role Of Cross Validation

- How to Choose K
- In general, the **more folds** we use in k-fold cross-validation the lower the bias of the test MSE but the higher the variance.
- Conversely, the **fewer folds** we use the higher the bias but the lower the variance.
- This is a classic example of the bias-variance tradeoff in machine learning.

Applications of Cross-Validation

1. This technique can be used to compare the performance of different predictive modeling methods.
2. It has great scope in the medical research field.
3. It can also be used for the meta-analysis, as it is already being used by the data scientists in the field of **medical statistics**.

Role Of Cross Validation

- Extensions of K-Fold Cross-Validation
- There are several extensions of k-fold cross-validation, including:

1. Repeated K-fold Cross-Validation:
2. Leave-One-Out Cross-Validation:
3. Stratified K-Fold Cross-Validation:
4. Nested Cross-Validation:

Code

- # Applying K fold cross validation
- # https://www.statology.org/k-fold-cross-validation-in-python/
- from sklearn.model_selection import train_test_split
- from sklearn.model_selection import KFold
- from sklearn.model_selection import cross_val_score
- from sklearn.linear_model import LinearRegression
- from numpy import mean
- from numpy import absolute
- from numpy import sqrt
- import pandas as pd

```
df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})

#define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']
```

```
#define cross-validation method to use
# cv = KFold(n_splits=10, random_state=1, shuffle=True)
cv = KFold(n_splits=10)

#build multiple linear regression model
model = LinearRegression()

#use k-fold CV to evaluate model
scores = cross_val_score(model, X, y,
scoring='neg_mean_absolute_error',cv=cv, n_jobs=-1)

#view mean absolute error
mae= mean(absolute(scores))
print("MSE = ", mae)
```

Output

MSE = 3.1461548083469744

Code

```
# Applying K fold cross validation
# https://www.statology.org/k-fold-cross-validation-in-python/
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd

df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})

print(df)

#define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']

#define cross-validation method to use
# cv = KFold(n_splits=10, random_state=1, shuffle=True)
cv = KFold(n_splits=10)

#build multiple linear regression model
model = LinearRegression()
#use k-fold CV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',cv=cv, n_jobs=-1)
#view mean absolute error
mae= mean(absolute(scores))
print("MSE = ", mae)

print("Scores = ", scores)
```

Kfold() Function

- **K-Folds cross-validator**
- Provides train/test indices to split data in train/test sets.
- Split dataset into k consecutive folds (without shuffling by default).
- Each fold is then used once as a validation while the k - 1
  remaining folds form the training set.
- Example
- **cv = KFold(n_splits=10, random_state=1, shuffle=True)**

Kfold() Function : Parameters

- cv = KFold(n_splits=10, random_state=1, shuffle=True)
- **Number of folds**. Must be at least 2.
  random_state : int, RandomState instance or None,
- default=None
- When shuffle is True, random_state affects the ordering of the indices, which controls the randomness of each fold.
- **Shuffle** : bool, default=False
- shuffle the data before splitting into batches.

from sklearn.model_selection import Kfold

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

sklearn.model_selection.KFold

class sklearn.model_selection.**KFold**(*n_splits=5, *, shuffle=False, random_state=None*) [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

For visualisation of cross-validation behaviour and comparison between common scikit-learn split methods refer to Visualizing cross-validation behavior in scikit-learn

Parameters:

n_splits : *int, default=5*

Number of folds. Must be at least 2.

from sklearn.model_selection import Kfold

Official Documentation

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

cross_val_score ()

- Evaluate a score by cross-validation.
- General Form
- sklearn.model_selection.**cross_val_score**(*estimator*, *X*, *y=None*, ***, *groups= None*, *scoring=None*, *cv=None*, *n_jobs=None*, *verbose=0*, *fit_params=None*, *pre_dispatch='2*n_jobs'*, *error_score=nan*)

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

cross_val_score ()

- Evaluate a score by cross-validation.
- General Form
- sklearn.model_selection.**cross_val_score**(*estimator*, *X*, *y=None*, ***, *groups= None*, *scoring=None*, *cv=None*, *n_jobs=None*, *verbose=0*, *fit_params=None*, *pre_dispatch='2*n_jobs'*, *error_score=nan*)

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

cross_val_score ()

- **estimator** : estimator object
- The object to use to fit the data.
- **Xarray-** The data to fit.
- yarray- The target variable to try to predict in the case of supervised learning.
- **scoring** : str , A str or a scorer callable object
- **cv** : int, cross-validation generator or an iterable
- Determines the cross-validation splitting strategy.

cross_val_score ()

- **n_jobs :**
- int, default=None
- Number of jobs to run in parallel.
- -1 means using all processors.

from sklearn.model_selection import cross_val_score

Official Documentation

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

from sklearn.model_selection import cross_val_score

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

sklearn.model_selection.cross_val_score

`sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)` [source]

Evaluate a score by cross-validation.

Read more in the User Guide.

Parameters:

estimator : *estimator object implementing 'fit'*
The object to use to fit the data.

X : *array-like of shape (n_samples, n_features)*
The data to fit. Can be for example a list, or an array.

y : *array-like of shape (n_samples,) or (n_samples, n_outputs), default=None*
The target variable to try to predict in the case of supervised learning.

References

- https://www.statology.org/k-fold-cross-validation/
- https://www.javatpoint.com/cross-validation-in-machine-learning

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

Regularization

- Text

Regularization

- **Overfitting** is a phenomenon that occurs when a Machine Learning model is constrained to the training set and not able to perform well on unseen data.
- That is when our model learns the noise in the training data as well.
- This is the case when our model memorizes the training data instead of learning the patterns in it.

Regularization

- Regularization is a technique used to reduce errors by fitting the function appropriately on the given training set and avoiding **overfitting**.

Regularization

- The commonly used regularization techniques are :
- Lasso Regularization – L1 Regularization
- Ridge Regularization – L2 Regularization
- Elastic Net Regularization – L1 and L2 Regularization

Lasso Regression

- **LASSO (Least Absolute Shrinkage and Selection Operator)**
- A regression model which uses the L1 Regularization technique is called LASSO regression.
- **Lasso Regression** adds the *"absolute value of magnitude"* of the coefficient as a penalty term to the loss function(L).
- Lasso regression also helps us achieve feature selection by penalizing the weights to approximately equal to zero if that feature does not serve any purpose in the model.

Lasso Regression

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |w_i|$$

- *where,*
- ***m** – Number of Features*
- ***n** – Number of Examples*
- ***y_i** – Actual Target Value*
- ***y_i(hat)** – Predicted Target Value*

Ridge Regression

- A regression model that uses the **L2 regularization** technique is called **Ridge regression**.
- **Ridge regression** adds the "*squared magnitude*" of the coefficient as a penalty term to the loss function(L).

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} w_i^2$$

Elastic Net Regression

- This model is a combination of L1 as well as L2 regularization.
- That implies that we add the absolute norm of the weights as well as the squared measure of the weights.
- With the help of an extra hyperparameter (Alpha) that controls the ratio of the L1 and L2 regularization.

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\left((1-\alpha)\sum_{i=1}^{m}|w_i| + \alpha\sum_{i=1}^{m}w_i^2\right)$$

Lasso Regression

- Key Difference between Ridge Regression and Lasso Regression
- **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model.
- **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

Regularization : Program

- In this guide, you will learn how to implement the following linear regression models using scikit-learn:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Elastic Net Regression

- https://www.pluralsight.com/guides/linear-lasso-ridge-regression-scikit-learn

Regularization : Program

- The data used in this project was produced from US economic time series data.
- available from
- [https://www.kaggle.com/datasets/yeonseokcho/economicsdataset]
- The data contains 574 rows and 5 variables, as described below:
- **psavert** - personal savings rate.
- **pce** - personal consumption expenditures, in billions of dollars.
- **uempmed** - median duration of unemployment, in weeks.
- **pop** - total population, in thousands.
- **unemploy-** number of unemployed in thousands (dependent variable).

Regularization : Program

- **Evaluation Metrics**
- We will evaluate the performance of the model using two metrics - R-squared value and Root Mean Squared Error (RMSE)

Regularization : Program

- **Steps**
- In this guide, we will follow the following steps:
- *Step 1 - Loading the required libraries and modules.*
- *Step 2 - Loading the data and performing basic data checks.*
- *Step 3 - Creating arrays for the features and the response variable.*
- *Step 4 - Creating the training and test datasets.*
- *Step 5 - Build, Predict and Evaluate the regression model.*
- We will be repeating Step 5 for the various regression models.
- The following sections will cover these steps.

Regularization : Program

- # Note download the data set from kaggle.
- # rename it as regressionexample2.csv
- # delete the first (date) column from data set
- # run the following code

Regularization : Program : Font Size 1

```
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

df = pd.read_csv('regressionexample2.csv')
print("Orriginala data set ")
print(df.head())
```

Orriginala data set

	pce	pop	psavert	uempmed	unemploy
0	506.7	198712.0	12.6	4.5	2944
1	509.8	198911.0	12.6	4.7	2945
2	515.6	199113.0	11.9	4.6	2958
3	512.2	199311.0	12.9	4.9	3143
4	517.4	199498.0	12.8	4.7	3066

```
# Normalization of data
print("Normalized train (divide by max value) data set ")
target_column = ['unemploy']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe()
print(df.head())

X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=40)
print(X_train.shape);
print(X_test.shape)
```

```
Normalized train (divide by max value) data set
        pce       pop    psavert    uempmed  unemploy
0  0.041554  0.620195  0.728324  0.178571        2944
1  0.041808  0.620816  0.728324  0.186508        2945
2  0.042284  0.621447  0.687861  0.182540        2958
3  0.042005  0.622065  0.745665  0.194444        3143
4  0.042431  0.622648  0.739884  0.186508        3066
(401, 4)
(173, 4)
```

```
# Standard LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)

pred_train_lr= lr.predict(X_train)
pred_test_lr= lr.predict(X_test)
print("Result for Standard LinearRegression")
print("On Testing Data Score = ", r2_score(y_test, pred_test_lr))
```

Output

```
Result for Standard LinearRegression
On Testing Data Score =  0.8316156967087425
```

```
# Ridge Regression
rr = Ridge(alpha=0.01)
rr.fit(X_train, y_train)
pred_test_rr= rr.predict(X_test)
print("Result for Ridge Regression")
print("On Testing Data Score = ", r2_score(y_test, pred_test_rr))
```

Output

```
Result for Ridge Regression
On Testing Data Score =  0.8330386695177572
```

```
# Lasso Regression
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(X_train, y_train)
pred_test_lasso= model_lasso.predict(X_test)
print("result for Lasso Regression")
print("On Testing Data Score = ", r2_score(y_test, pred_test_lasso))
```

Output

```
result for Lasso Regression
On Testing Data Score =  0.8316452818380049
```

```
# ElasticNet Regression
model_enet = ElasticNet(alpha = 0.01)
model_enet.fit(X_train, y_train)
pred_test_enet= model_enet.predict(X_test)
print("Result for ElasticNet Regression")
print("On Testing Data Score = ", r2_score(y_test, pred_test_enet))
```

Output

```
Result for ElasticNet Regression
On Testing Data Score =  0.7059436829753674
```

Result for Standard LinearRegression
On Testing Data Score = 0.8316156967087425
Result for Ridge Regression
On Testing Data Score = 0.8330386695177572
result for Lasso Regression
On Testing Data Score = 0.8316452818380049
Result for ElasticNet Regression
On Testing Data Score = 0.7059436829753674

Conclusion

Linear Regression Model: R-square of 83.1615 percent.
Ridge Regression Model: R-square of 83.30 percent. Best
Lasso Regression Model: R-square of 83.1645 percent.
ElasticNet Regression Model: R-square of 70.59 percent.

References

- # https://www.pluralsight.com/guides/linear-lasso-ridge-regression-scikit-learn
- # https://www.kaggle.com/datasets/yeonseokcho/economicsdataset

Bias-variance Analysis

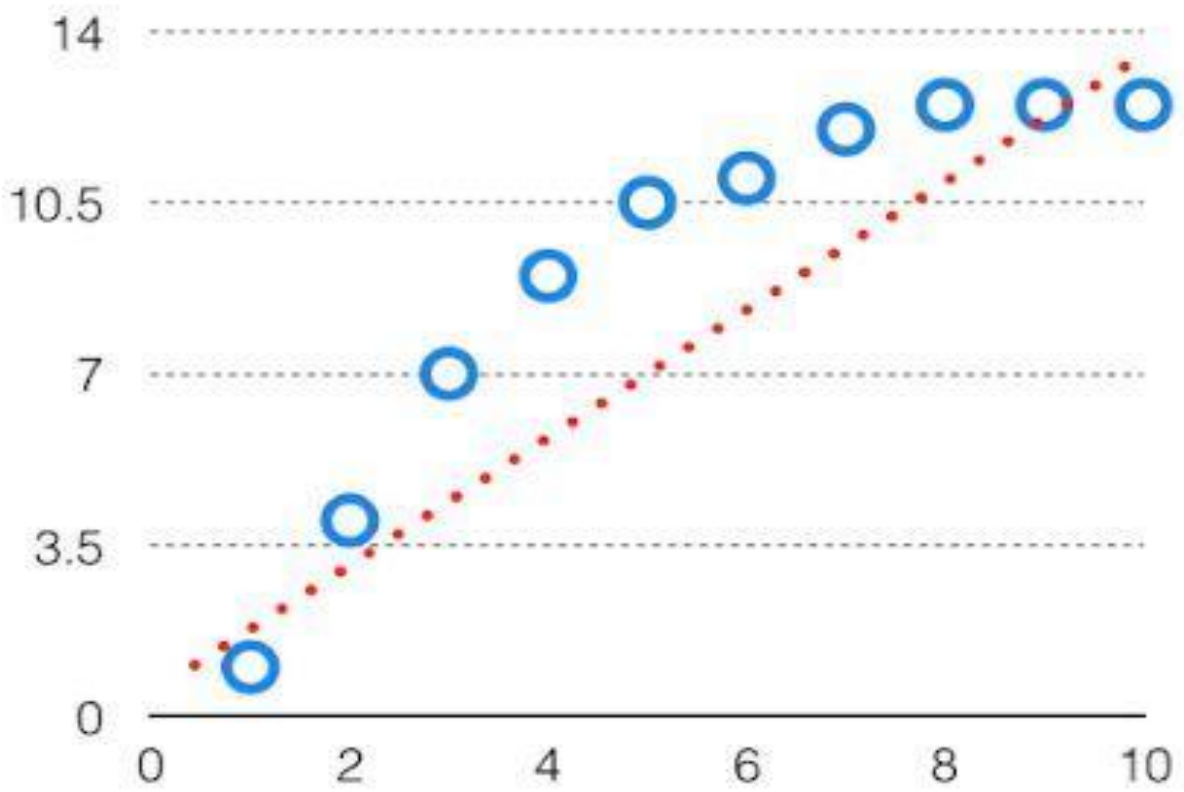
- **What is Bias?**
- The bias is known as the difference between the prediction of the values by the Machine Learning model and the correct value.
- Being high in biasing gives a large error in training as well as testing data.
- It recommended that an algorithm should always be low-biased to avoid the problem of underfitting.

Bias-variance Analysis

- **What is Bias?**
- By **high bias**, the data predicted is in a straight line format, thus not fitting accurately in the data in the data set.
- Such fitting is known as the **Under fitting** of Data.
- This happens when the hypothesis is too simple or linear in nature.

Bias-variance Analysis

- Refer to the graph given below for an example of such a situation.



Bias-variance Analysis

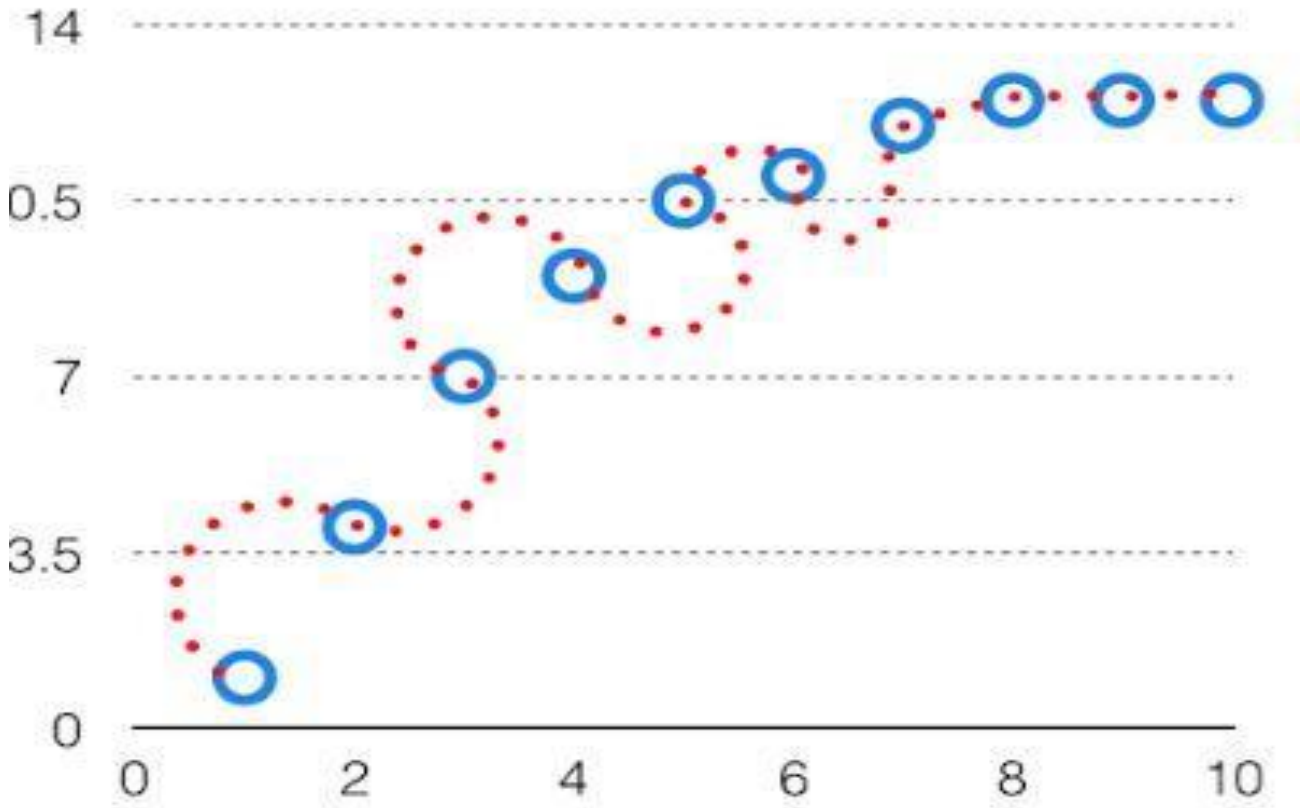
- **What is Variance?**
- The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn’t seen before.
- As a result, such models perform very well on training data but have high error rates on test data.
- When a model is high on variance, it is then said to as **Overfitting of Data**.

Bias-variance Analysis

- **What is Variance?**
- Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high.

Bias-variance Analysis

- The high variance data looks as follows.



Bias Variance Tradeoff

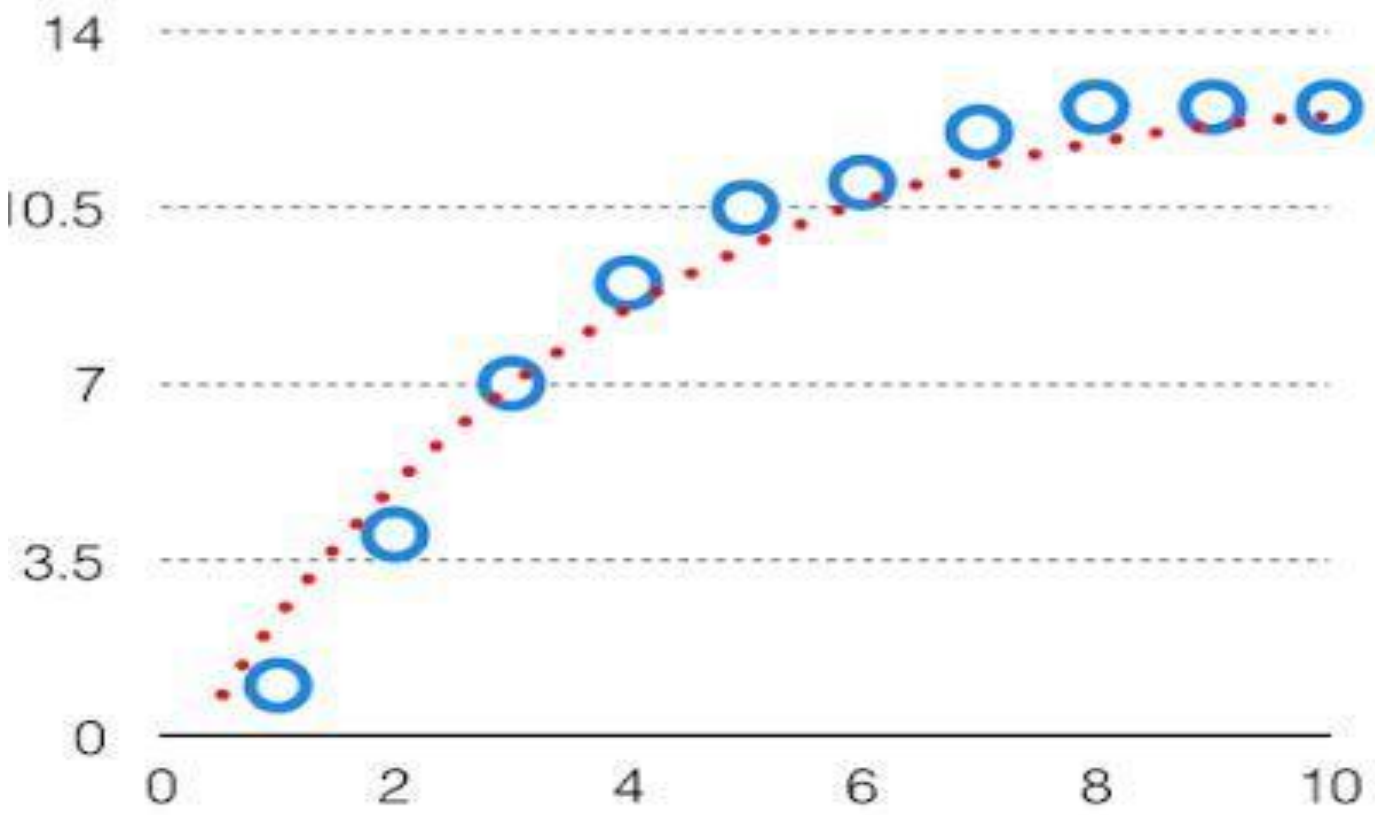
- **Bias Variance Tradeoff**
- If the algorithm is too simple (hypothesis with linear equation) then it may be on high bias and low variance condition and thus is error-prone.
- If algorithms fit too complex (hypothesis with high degree equation) then it may be on high variance and low bias.
- In the latter condition, the new entries will not perform well.

Bias Variance Tradeoff

- Well, there is something between both of these conditions, known as a Trade-off or Bias Variance Trade-off.
- This tradeoff in complexity is why there is a tradeoff between bias and variance.
- An algorithm can’t be more complex and less complex at the same time.

Bias Variance Tradeoff

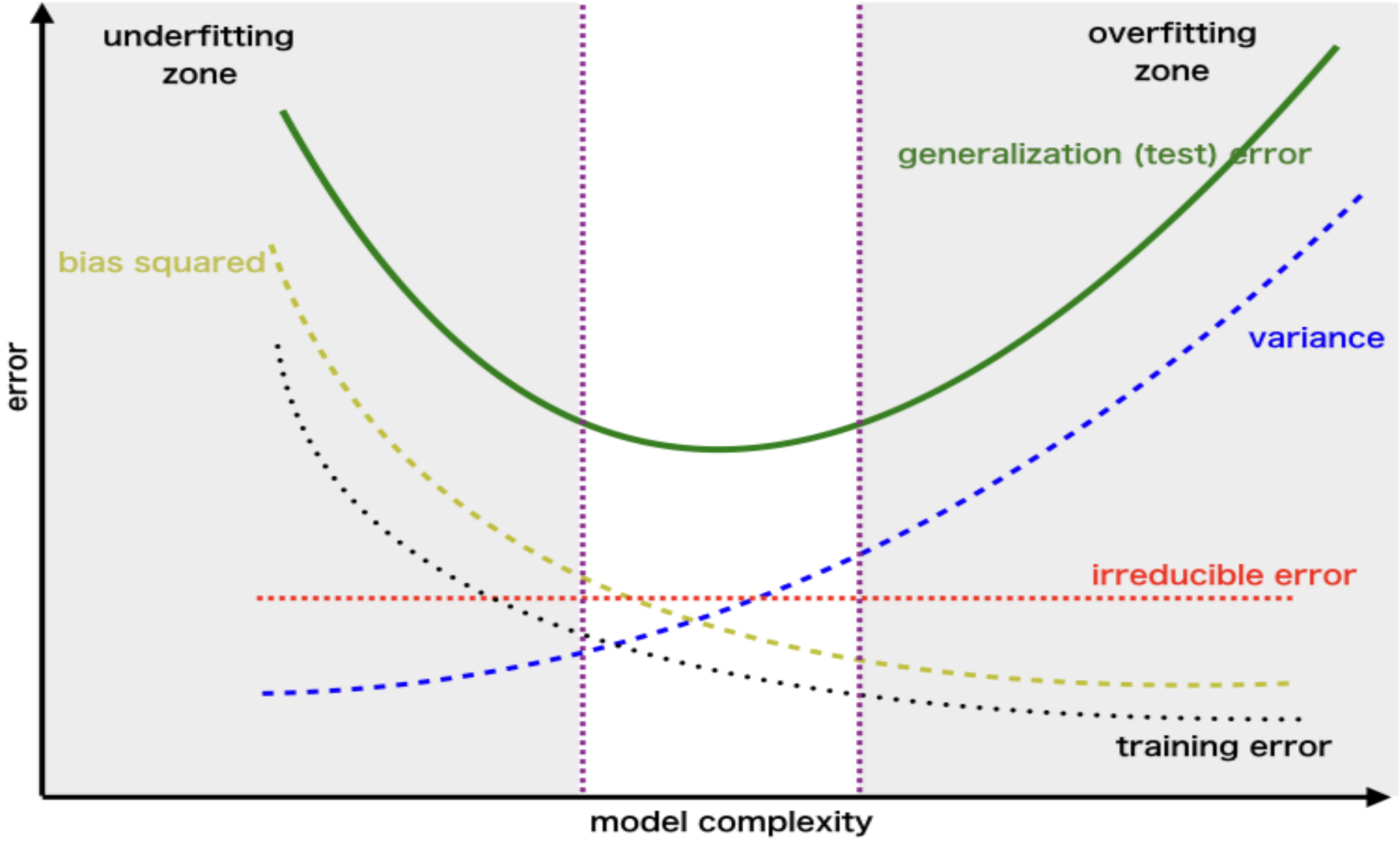
- For the graph, the perfect tradeoff will be like this.



Bias Variance Tradeoff

- We try to optimize the value of the total error for the model by using the Bias-Variance Tradeoff.
- The best fit will be given by the hypothesis on the tradeoff point.
- The error to complexity graph to show trade-off is given as –

The error to complexity graph to show trade-off is given as –



Bias Variance Tradeoff

- This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

References

- https://www.geeksforgeeks.org/ml-bias-variance-trade-off/

END

BCAC 0017
FUNDAMENTALS OF MACHINE LEARNING

Section 6: Performance-Measures:

Faculty Name
Mr. Sachin Sharma, Dr. Vinod Jain, Dr. Manu Banga,
Ms.Paromita Goswami, Ms.Chestha Bhardwaj

Section 6 : Performance-Measures:

- **Performance-Measures:** Types-of-errors, accuracy, confusion-matrix, precision-recall.

Section 6 : Performance-Measures:

- **Performance-Measures:**
- Types-of-errors
- accuracy
- confusion-matrix
- precision-recall

Performance-Measures

- Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model.
- ***To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics.***
- These performance metrics help us understand how well our model has performed for the given data.

Performance-Measures

- In this way, we can improve the model's performance by tuning the hyper-parameters.
- Each ML model aims to generalize well on unseen/new data, and performance metrics help determine how well the model generalizes on the new dataset.

Performance-Measures

- In machine learning, each task or problem is divided into **classification** and **Regression**.
- Not all metrics can be used for all types of problems; hence, it is important to know and understand which metrics should be used.

Performance-Measures

- Different evaluation metrics are used for both Regression and Classification tasks.
- In this topic, we will discuss metrics used for classification and regression tasks.

- **1. Performance Metrics for Regression**
- **2. Performance Metrics for Classification**

Types-of-errors

- Different evaluation metrics (ERRORS) are used for both Regression and Classification tasks. So errors are categorized on the basis of type of machine learning used.
- So mainly two types of errors (metrics) are :

- **1. Performance Metrics for Regression**
- **2. Performance Metrics for Classification**

Types-of-errors

- **Performance Metrics for Regression**
- Following are the popular metrics that are used to evaluate the performance of Regression models.

1. **Mean Absolute Error**
2. **Mean Squared Error**
3. **RMSE Mean Squared Error**
4. **R2 Score**

Types-of-errors

- **2. Performance Metrics for Classification**
- In a classification, machine learning model learns from the given dataset and then classifies the new data into classes or groups based on the training.
- It predicts class labels as the output, such as *Yes or No, 0 or 1, Spam or Not Spam,* etc.

Types-of-errors

- **2. Performance Metrics for Classification**
- To evaluate the performance of a classification model, different metrics are used, and some of them are as follows:

1. **Accuracy**
2. **Confusion Matrix**
3. **Precision**
4. **Recall**
5. **F-Score**
6. **AUC(Area Under the Curve)-ROC**

References

- https://www.javatpoint.com/performance-metrics-in-machine-learning
- https://www.geeksforgeeks.org/metrics-for-machine-learning-model/

Mean Absolute Error

- Mean Absolute Error (MAE)
- Mean Absolute Error or MAE is one of the simplest metrics, which measures the absolute difference between actual and predicted values, where absolute means taking a number as Positive.
- **Y is the Actual outcome, Y' is the predicted outcome**, and N is the total number of data points.

The below formula is used to calculate MAE:

$$MAE = 1/N \sum |Y - Y'|$$

Mean Squared Error

- It measures the average of the Squared difference between predicted values and the actual value given by the model.

Mean Squared Error

- The formula for calculating MSE is given below:

$$MSE = 1/N \sum (Y - Y')^2$$

Mean Squared Error

- In MSE, errors are squared, therefore it only assumes non-negative values, and it is usually positive and non-zero.
- Moreover, due to squared differences, it penalizes small errors also, and hence it leads to over-estimation of how bad the model is.

Root Mean Square Error(RMSE)

- We can say that RMSE is a metric that can be obtained by just taking the square root of the MSE value.
- This gives higher weightage to the large errors in predictions.

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{N} \left(y_j - \hat{y}_j\right)^2}{N}}$$

Marks in Mid Term of ML							
		Y	Y'				
SN	CPI	Actual	predicted	Y-Y'	|Y-Y'|	(Y-Y')^2	MAE
1	6.3	8	5	3	3	9	[(1/10)*22]
2	6.5	9	6	3	3	9	2.2
3	6.8	7	8	-1	1	1	
4	7.1	14	12	2	2	4	
5	7.5	15	16	-1	1	1	MSE
6	7.7	10	17	-7	7	49	[(1/10)*86]
7	7.9	22	22	0	0	0	8.6
8	8.1	27	24	3	3	9	
9	8.5	26	26	0	0	0	RMSE
10	8.8	29	27	2	2	4	SQRT(8.6)
			Sum	4	22	86	2.93

$$MAE = 1/N \sum |Y - Y'|$$

$$MSE = 1/N \sum (Y - Y')^2$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{N} (y_j - \hat{y}_j)^2}{N}}$$

Marks in Mid Term of COA							
		Y	Y'				
SN	CPI	Actual	predicted	Y-Y'	|Y-Y'|	(Y-Y')^2	MAE
1	6.3	5	5				
2	6.5	6	6				
3	6.8	7	8				
4	7.1	12	12				
5	7.5	11	16				MSE
6	7.7	14	17				
7	7.9	18	22				
8	8.1	19	24				
9	8.5	21	26				RMSE
10	8.8	23	27				
			Sum				

$$MAE = 1/N \sum |Y - Y'|$$

$$MSE = 1/N \sum (Y - Y')^2$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{N} (y_j - \hat{y}_j)^2}{N}}$$

Marks in Mid Term of ML							
		Y	Y'				
SN	CPI	Actual	predicted	Y-Y'	|Y-Y'|	(Y-Y')^2	MAE
1	6.3	5	5	0	0	0	[(1/10)*27]
2	6.5	6	6	0	0	0	2.7
3	6.8	7	8	-1	1	1	
4	7.1	12	12	0	0	0	
5	7.5	11	16	-5	5	25	MSE
6	7.7	14	17	-3	3	9	[(1/10)*117]
7	7.9	18	22	-4	4	16	11.7
8	8.1	19	24	-5	5	25	
9	8.5	21	26	-5	5	25	RMSE
10	8.8	23	27	-4	4	16	SQRT(11.7)
			Sum	-27	27	117	3.42

$$MAE = 1/N \sum |Y - Y'|$$

$$MSE = 1/N \sum (Y - Y')^2$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{N} (y_j - \hat{y}_j)^2}{N}}$$

R2 Score

- R squared error is also known as Coefficient of Determination.
- The R-squared metric enables us to compare our model with a **constant baseline to determine the performance of the model.**
- The R squared score will always be less than or equal to 1 without concerning if the values are too large or small.

$$R^2 = 1 - \frac{MSE(Model)}{MSE(Baseline)}$$

Types-of-errors

- **2. Performance Metrics for Classification**

1. Accuracy
2. Confusion Matrix
3. Precision
4. Recall
5. F-Score

Accuracy

- The accuracy can be determined as the number of correct predictions to the total number of predictions.
- It can be formulated as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

Accuracy : How to calculate

- Firstly, we need to import the *accuracy_score* function of the scikit-learn library as follows:
- from sklearn.metrics import accuracy_score
- print(“Accuracy Score = “,accuracy_score(y_test,y_hat)')
- Here y_hat = y_test_predicted
- Although it is simple to use and implement, it is suitable only for cases where an equal number of samples belong to each class.

Accuracy

- **When to Use Accuracy?**
- It is good to use the Accuracy metric when the target variable classes in data are approximately balanced.
- For example, if 60% of classes in a fruit image dataset are of Apple, 40% are Mango.
- In this case, if the model is asked to predict whether the image is of Apple or Mango, it will give a prediction with 97% of accuracy.

Accuracy

- **When not to use Accuracy?**
- It is recommended not to use the Accuracy measure when the target variable majorly belongs to one class.
- For example, Suppose there is a model for a disease prediction in which, out of 100 people, only five people have a disease, and 95 people don't have one.
- In this case, if our model predicts every person with no disease (which means a bad prediction), the Accuracy measure will be 95%, which is not correct.

Confusion-matrix

- A confusion matrix is a tabular representation of prediction outcomes of **any binary classifier**, which is used to describe the performance of the classification model on a set of test data when true values are known.
- The confusion matrix is simple to implement, but the terminologies used in this matrix might be confusing for beginners.

Confusion-matrix

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Confusion-matrix

- In the matrix, columns are for the prediction values, and rows specify the Actual values.
- Here Actual and prediction give two possible classes, Yes or No.
- So, if we are predicting the presence of a disease in a patient, the Prediction column with Yes means, Patient has the disease, and for NO, the Patient doesn't have the disease.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Confusion-matrix

- In this example, the total number of predictions are 165, out of which 110 time predicted yes, whereas 55 times predicted No.
- However, in reality, 60 cases in which patients don't have the disease, whereas 105 cases in which patients have the disease.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Confusion-matrix

- In general, the table is divided into four terminologies, which are as follows:
- **True Positive(TP):**
- True Negative(TN):
- False Positive(FP):
- False Negative(FN):

Confusion-matrix

- In general, the table is divided into four terminologies, which are as follows:
- **True Positive(TP):**
- True Negative(TN):
- False Positive(FP):
- False Negative(FN):

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP)**
	Positive +	**False Negative (FN)**	True Positive **(TP)**

Confusion-matrix

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP)**
	Positive +	**False Negative (FN)**	True Positive **(TP)**

Confusion-matrix

- **TP and TN**
- **True Positive(TP):**
  - In this case, the prediction outcome is true, and it is true in reality, also.
- **True Negative(TN):**
  - In this case, the prediction outcome is false, and it is false in reality, also.

Confusion-matrix

- **FP and FN**
- **False Positive(FP):**
  - In this case, prediction outcomes are true, but they are false in actuality.
- **False Negative(FN):**
  - In this case, predictions are false, and they are true in actuality.

- **True Positive(TP):** In this case, the prediction outcome is true, and it is true in reality, also.
- **True Negative(TN):** in this case, the prediction outcome is false, and it is false in reality, also.
- **False Positive(FP):** In this case, prediction outcomes are true, but they are false in actuality.
- **False Negative(FN):** In this case, predictions are false, and they are true in actuality.

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP)**
	Positive +	**False Negative (FN)**	True Positive **(TP)**

Confusion-matrix

n=165	Predicted: NO	Predicted: YES
Actual: NO	50 TN	10 FP
Actual: YES	5 FN	100 TP

Confusion-matrix

- Model

Confusion-matrix

Cancer prediction for 200 patients				Verified
	Positive	Negative		
Actual	100	100		
Prediction	95,5	93,7	(Correct,Incorrect)	
			Confusion Matrix	
TN			TN	FP
FP			FN	TP
FN				
TP				

Precision-recall

Cancer prediction for 200 patients				Verified
	Positive	Negative		
Actual	100	100		
Prediction	95,5	93,7	(Correct,Incorrect)	
			Confusion Matrix	
TN	93		TN	FP
FP	7		FN	TP
FN	5		93	7
TP	95		5	95

Precision-recall

Cancer prediction for 300 patients				Verified
	Positive	Negative		
Actual	250	50		
Prediction	225,25	40,10	(Correct,Incorrect)	
			Confusion Matrix	
TN			TN	FP
FP			FN	TP
FN				
TP				

Precision-recall

Cancer prediction for 300 patients				Verified
	Positive	Negative		
Actual	250	50		
Prediction	225,25	40,10	(Correct,Incorrect)	
			Confusion Matrix	
TN	40		TN	FP
FP	10		FN	TP
FN	25		40	10
TP	225		25	225

Precision-recall

Diabetes prediction for 1000 patients			Verified	
	Positive	Negative		
Actual	600	400		
Prediction	550,50	370,30	(Correct,Incorrect)	
			Confusion Matrix	
TN	370		TN	FP
FP	30		FN	TP
FN	50		370	30
TP	550		50	550

Precision-recall

Diabetes prediction for 1000 patients				Verified
	Positive	Negative		
Actual	600	400		
Prediction	550,50	370,30	(Correct,Incorrect)	
			Confusion Matrix	
TN			TN	FP
FP			FN	TP
FN				
TP				

Precision-recall

- **Precision**
- The precision metric is used to overcome the limitation of Accuracy.
- The precision determines the proportion of positive prediction that was actually correct.
- It can be calculated as the True Positive or predictions that are actually true to the total positive predictions (True Positive and False Positive).

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

Precision-recall

- **Recall or Sensitivity**
- It is also similar to the Precision metric;
- It can be calculated as True Positive or predictions that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative).

$$\text{Recall} = \frac{TP}{TP+FN}$$

Precision-recall

- **When to use Precision and Recall?**
- Recall determines the performance of a classifier with respect to a false negative, whereas precision gives information about the performance of a classifier with respect to a false positive.

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

$$\text{Recall} = \frac{TP}{TP+FN}$$

Precision-recall

- **When to use Precision and Recall?**
- So, if we want to minimize the false negative, then, Recall should be as near to 100%, and if we want to minimize the false positive, then precision should be close to 100% as possible.
- In simple words, *if we maximize precision, it will minimize the FP errors, and if we maximize recall, it will minimize the FN error.*

Calculate Precision and recall

Cancer prediction for 300 patients				Verified
	Positive	Negative		
Actual	250	50		
Prediction	225,25	40,10		
TN		Precision	?	
FP		Recall	?	
FN				
TP				

Calculate Precision and recall

Cancer prediction for 300 patients				Verified
	Positive	Negative		
Actual	250	50		
Prediction	225,25	40,10		
TN	40	Precision	=225/(225+10)=0.9574	
FP	10	Recall	=225/(225+25)=0.90	
FN	25			
TP	225			

Calculate Precision and recall ? Question.

- A judge take decision about total 100 cases in which he have take the decision for ‘Hang till death’.
- He ordered in 40 cases for ‘hang till death’ but among which only 30 was culprit.
- He do not punish 60 cases but from these 10 was culprit.
- Create the confusion matrix.
- Also find the value of precision and recall.

F-Score

- F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class.
- It is calculated with the help of Precision and Recall.
- It is a type of single score that represents both Precision and Recall.
- So, ***the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.***

$$F1 = 2 * \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

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

F-Score

- **When to use F-Score?**
- As F-score make use of both precision and recall, so it should be used if both of them are important for evaluation, but one (precision or recall) is slightly more important to consider than the other.
- For example, when False negatives are comparatively more important than false positives, or vice versa.

References

- https://www.javatpoint.com/performance-metrics-in-machine-learning
- https://www.geeksforgeeks.org/metrics-for-machine-learning-model/

END

BCAC 0017
FUNDAMENTALS OF MACHINE LEARNING

Section 7 Dimensionality Reduction :

Faculty Name
Mr. Sachin Sharma, Dr. Vinod Jain, Dr. Manu Banga,
Ms.Paromita Goswami, Ms.Chestha Bhardwaj

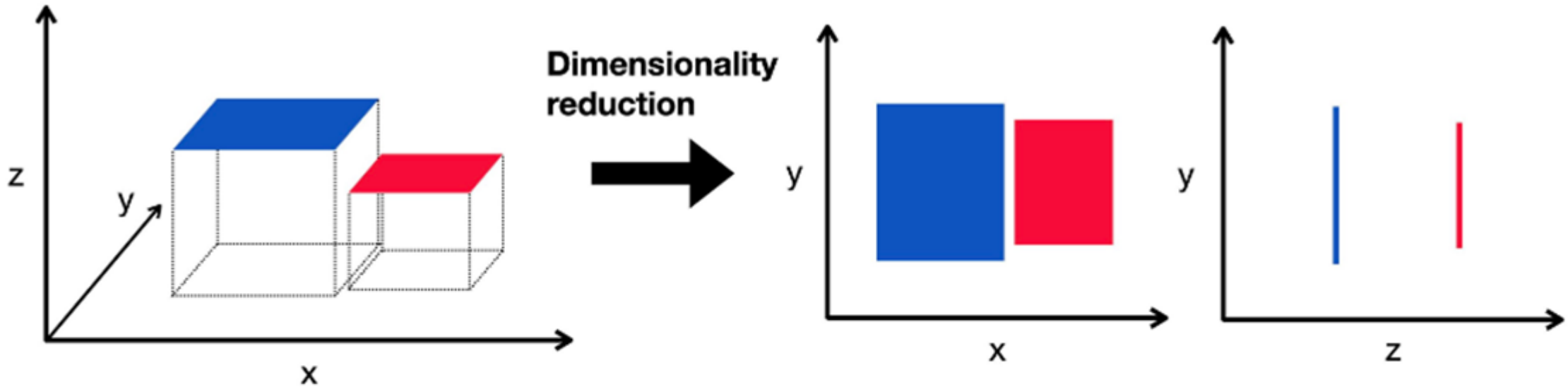
Section 7 Dimensionality Reduction :

- **Dimensionality Reduction :**
- Feature Selection vs. feature extraction
- Principal Component Analysis (PCA).

Dimensionality Reduction

- The number of input features, variables, or columns present in a given dataset is known as dimensionality

- The process to reduce these features is called dimensionality reduction.



Dimensionality reduction embeds the high-dimensional data into a lower dimensional space.

Dimensionality Reduction

- Dimensionality reduction technique can be defined as,

- ***"It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information."***

- These techniques are widely used in machine learning for obtaining a better fit predictive model while solving the classification and regression problems.

Dimensionality Reduction

- It is commonly used in the fields that deal with high-dimensional data, such as

1. Speech Recognition,
2. Signal Processing,
3. Bioinformatics, Etc.
4. Data Visualization,
5. Noise Reduction,
6. Cluster Analysis, Etc.

Dimensionality Reduction

- **The Curse of Dimensionality**
- As the number of features increases, the number of samples also gets increased proportionally, and the chance of overfitting also increases.
- **If the machine learning model is trained on high-dimensional data, it becomes over fitted and results in poor performance.**
- Hence, it is often required to reduce the number of features, which can be done with dimensionality reduction.

Dimensionality Reduction

- **Benefits of applying Dimensionality Reduction**
- Some benefits of applying dimensionality reduction technique to the given dataset are given below:

1. By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
2. Less Computation training time is required for reduced dimensions of features.
3. Reduced dimensions of features of the dataset help in visualizing the data quickly.
4. It removes the redundant features.

Disadvantages of dimensionality Reduction

- There are also some disadvantages of applying the dimensionality reduction, which are given below:

1. Some data may be lost due to dimensionality reduction.
2. In the PCA dimensionality reduction technique, sometimes the principal components required to consider are unknown.

Approaches of Dimension Reduction

- There are two ways to apply the dimension reduction technique, which are given below:

1. Feature Selection
2. Feature Extraction

Feature Selection

- Feature selection is the process of selecting the subset of the relevant features and leaving out the irrelevant features present in a dataset to build a model of high accuracy.
- In other words, it is a way of selecting the optimal features from the input dataset.

Feature Extraction:

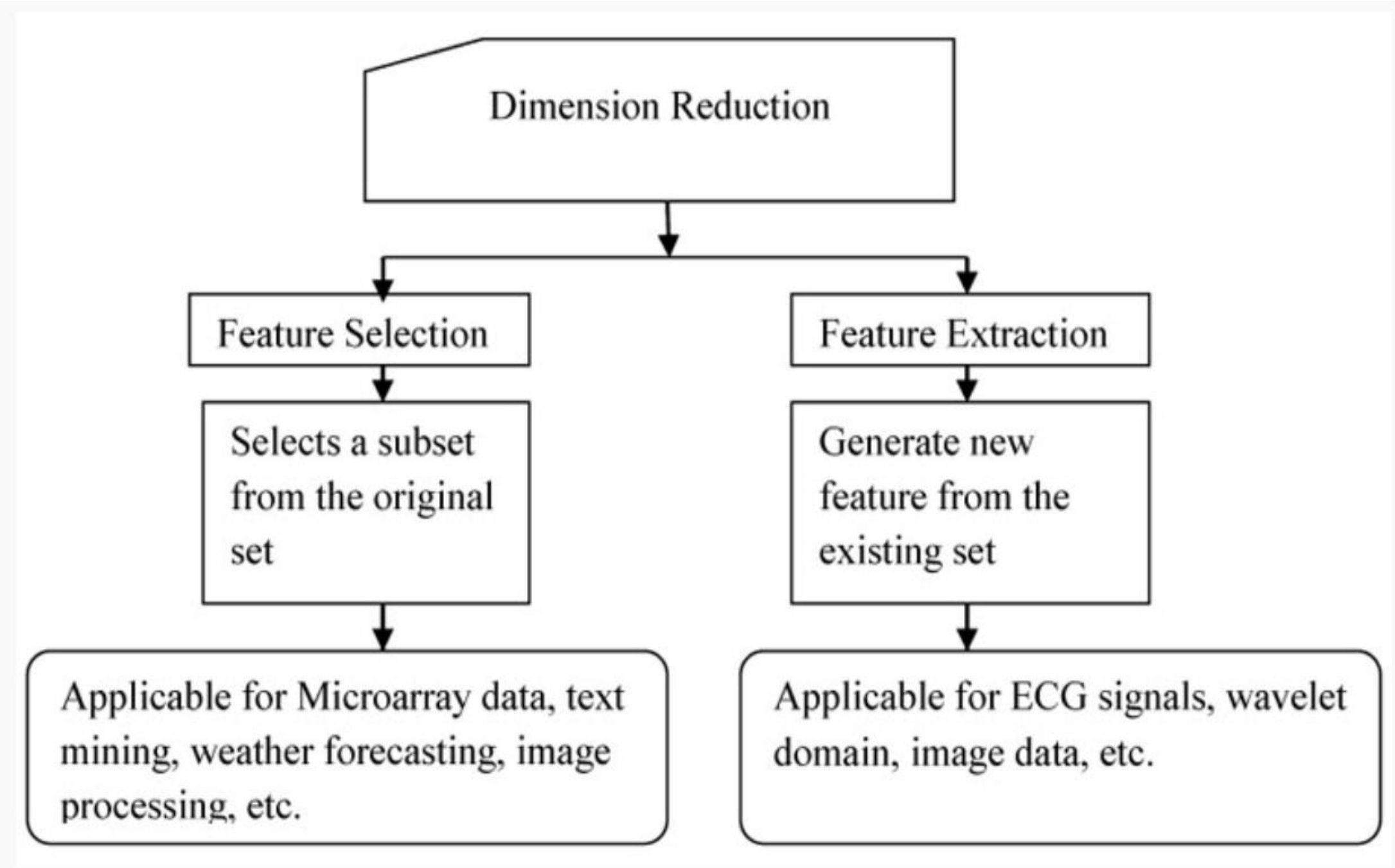
- Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions.
- This approach is useful when we want to keep the whole information but use fewer resources while processing the information.

Selection vs. Extraction

- In **feature selection** we try to find the best subset of the input feature set.

- In **feature extraction** we create new features based on transformation or combination of the original feature set.

- Both selection and extraction lead to the dimensionality reduction.
- No clear cut evidence that one of them is superior to the other on all types of task.


Dimension Reduction
Feature Selection
Feature Extraction
Selects a subset from the original set
Generate new feature from the existing set
Applicable for Microarray data, text mining, weather forecasting, image processing, etc.
Applicable for ECG signals, wavelet domain, image data, etc.

Feature Selection

- **Dimensionality Reduction**

Feature Selection

- Three methods are used for the feature selection:

1. Filter
2. Wrapper
3. Embedded

Feature Selection

- **1. Filters Methods**
- In this method, the dataset is filtered, and a subset that contains only the relevant features is taken. Some common techniques of filters method are:
- **Correlation**
- **Chi-Square Test**
- **ANOVA**
- **Information Gain, etc.**

Feature Selection

- **2. Wrappers Methods**
- In this method, some features are fed to the ML model, and evaluate the performance. The performance decides whether to add those features or remove to increase the accuracy of the model.
- This method is more accurate than the filtering method but complex to work. Some common techniques of wrapper methods are:
- Forward Selection
- Backward Selection
- Bi-directional Elimination

Feature Selection

- **3. Embedded Methods:** Embedded methods check the different training iterations of the machine learning model and evaluate the importance of each feature. Some common techniques of Embedded methods are:
- **LASSO**
- **Elastic Net**
- **Ridge Regression, etc.**

Feature Extraction

Feature Extraction

- Feature Extraction:
- Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions.
- This approach is useful when we want to keep the whole information but use fewer resources while processing the information.
- Some common feature extraction techniques are:

1. Principal Component Analysis
2. Linear Discriminant Analysis
3. Kernel PCA
4. Quadratic Discriminant Analysis

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

- **Principal Component Analysis (PCA)**
- As stated earlier, Principal Component Analysis is a technique of feature extraction that maps a higher dimensional feature space to a lower-dimensional feature space.
- While reducing the number of dimensions, PCA ensures that maximum information of the original dataset is retained in the dataset with the reduced no. of dimensions and the co-relation between the newly obtained Principal Components is minimum.

Principal Component Analysis (PCA)

- The new features obtained after applying PCA are called Principal Components and are denoted as ***PCi (i=1,2,3…n)***.
- Here, (Principal Component-1) PC1 captures the maximum information of the original dataset, followed by PC2, then PC3 and so on.

Principal Component Analysis (PCA)

- **Steps to Apply PCA in Python for Dimensionality Reduction**
- Step-1: Import necessary libraries
- Step-2: Load the dataset
- Step-3: Standardize the features
  - Step-3.1: Check the Co-relation between features without PCA (Optional)
- Step-4: Applying Principal Component Analysis
- Step-5: Checking Co-relation between features after PCA

Principal Component Analysis (PCA)

- **Step-3: Standardize the features**
- Before applying PCA or any other Machine Learning technique it is always considered good practice to standardize the data.
- For this, Standard Scalar is the most commonly used scalar.

Principal Component Analysis (PCA)

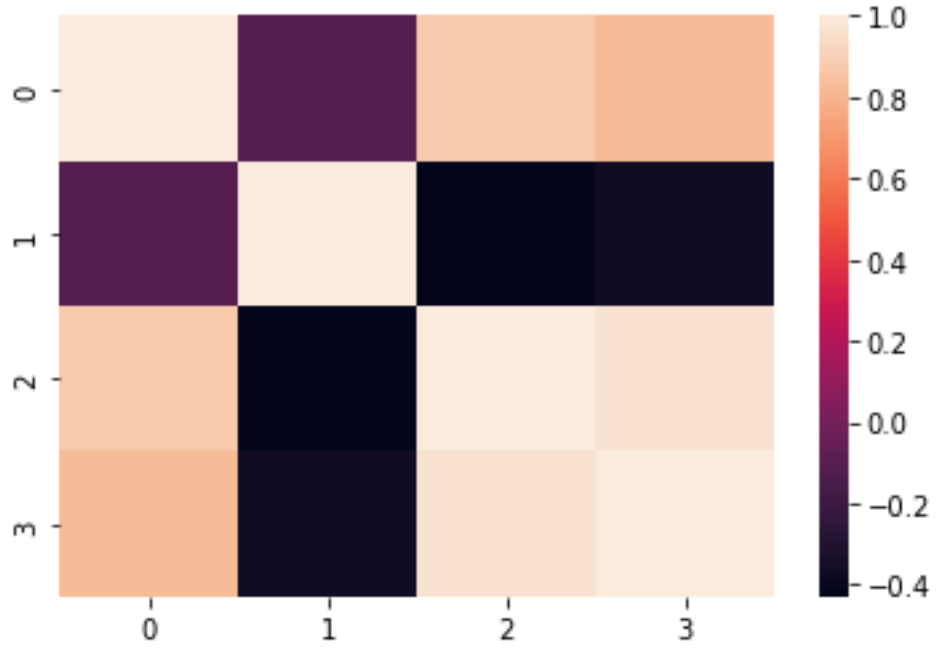
- **Check the Co-relation between features without PCA**
- Now, we will check the co-relation between our scaled dataset using a heat map.
- The correlation between various features is given by the *corr()* function and then the heat map is plotted by the heatmap() function.

Principal Component Analysis (PCA)

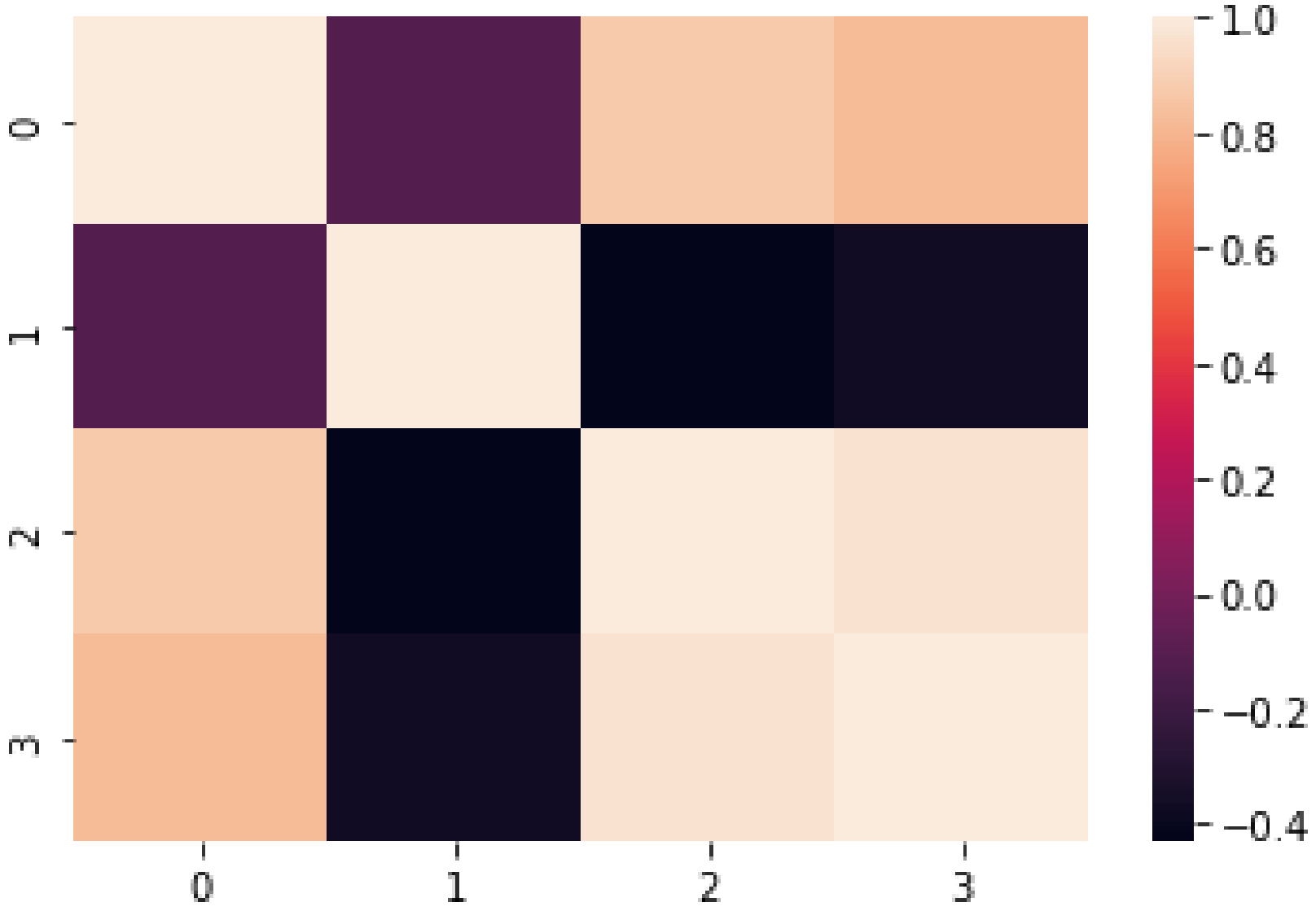
- **Check the Co-relation between features without PCA (Optional)**
- The colour scale on the side of the heatmap helps determine the magnitude of the co-relation.
- In our example, we can clearly see that a darker shade represents less co-relation while a lighter shade represents more co-relation.

Principal Component Analysis (PCA)

- **Check the Co-relation between features without PCA (Optional)**
- The diagonal of the heatmap represents the co-relation of a feature with itself – which is always 1.0, thus, the diagonal of the heatmap is of the highest shade.



darker shade represents less co-relation while a lighter shade represents more co-relation.



Principal Component Analysis (PCA)

Step-1: Import necessary libraries

```
# Import necessary libraries

from sklearn import datasets # to retrieve the iris Dataset

import pandas as pd

from sklearn.preprocessing import StandardScaler # to standardize the features

from sklearn.decomposition import PCA # to apply PCA

import seaborn as sns # to plot the heat maps
```

Principal Component Analysis (PCA)

Step-2: Load the dataset

```
#Load the Dataset
iris = datasets.load_iris()
#convert the dataset into a pandas data frame
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
#display the head (first 5 rows) of the dataset
print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

Principal Component Analysis (PCA)

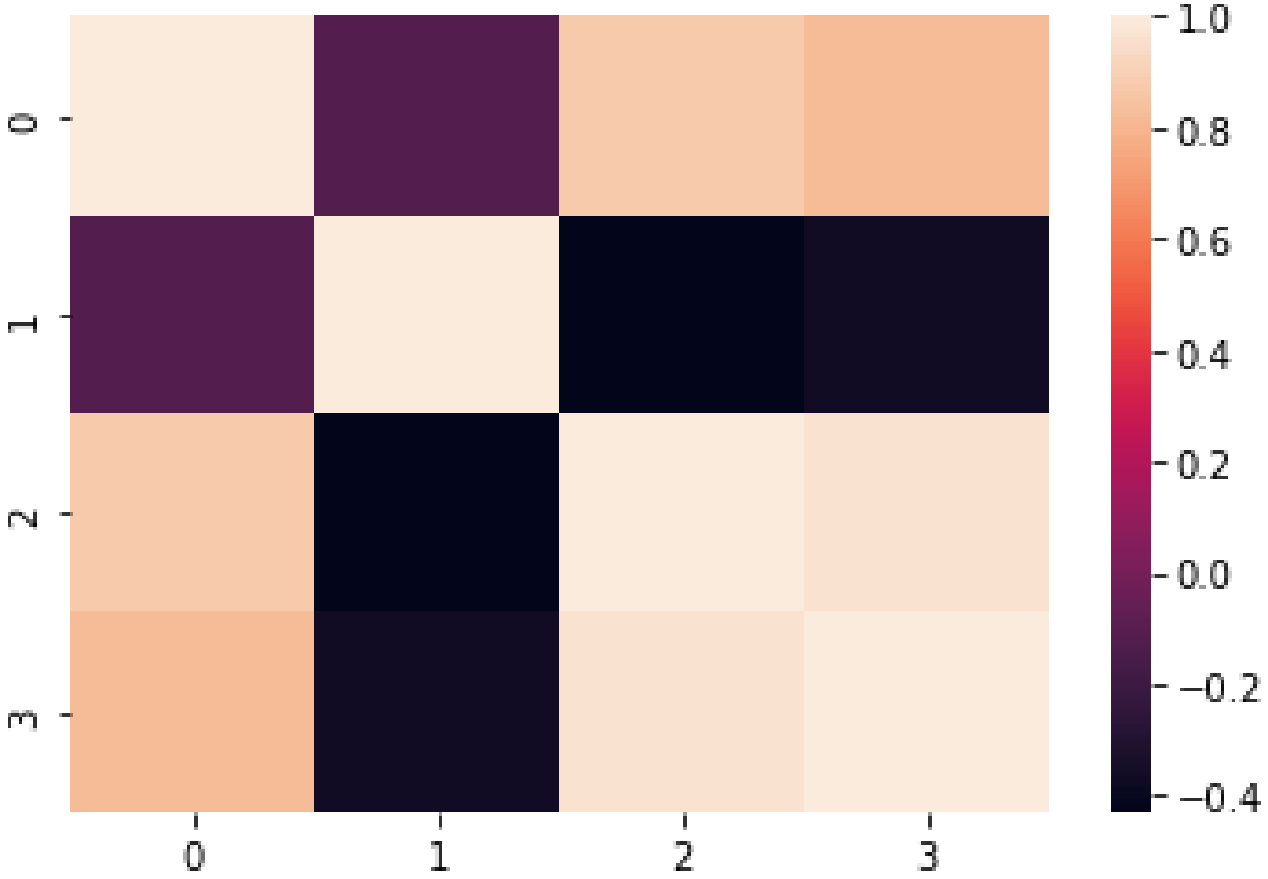
Step-3: Standardize the features

```
#Standardize the features
#Create an object of StandardScaler which is present in sklearn.preprocessing
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
print("scaled_data")
print(scaled_data)
```

```
scaled_data
          0         1         2         3
0 -0.900681  1.019004 -1.340227 -1.315444
1 -1.143017 -0.131979 -1.340227 -1.315444
2 -1.385353  0.328414 -1.397064 -1.315444
3 -1.506521  0.098217 -1.283389 -1.315444
4 -1.021849  1.249201 -1.340227 -1.315444
```

Principal Component Analysis (PCA)

#Check the Co-relation between features without PCA
sns.heatmap(scaled_data.corr())



Principal Component Analysis (PCA)

Step-4: Applying Principal Component Analysis

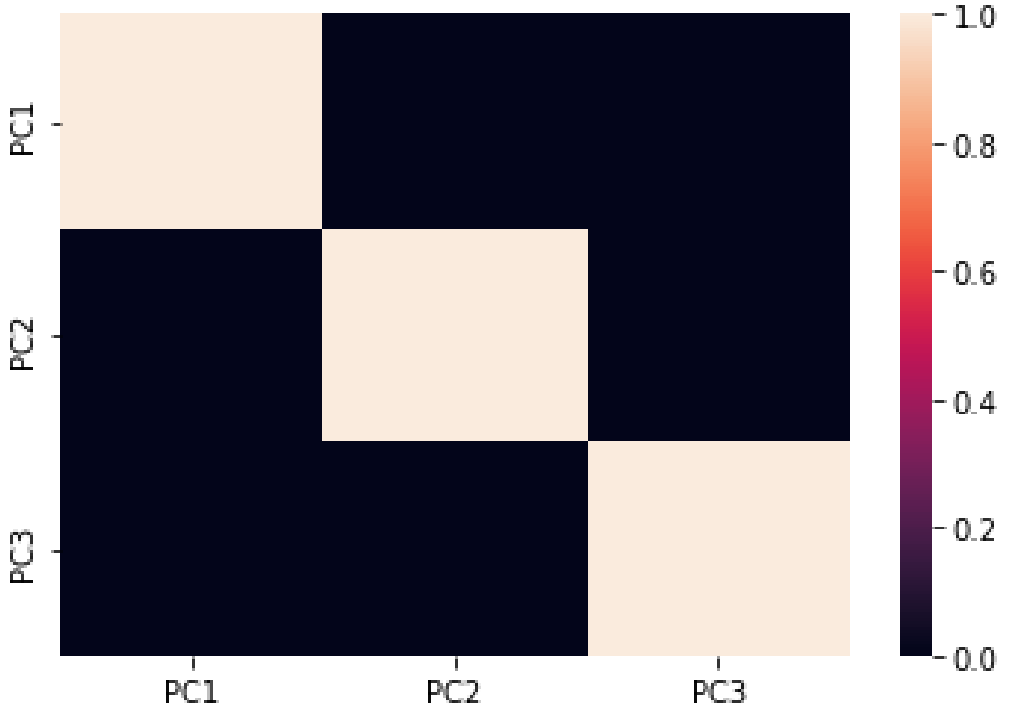
```
#Applying PCA
#Taking no. of Principal Components as 3
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])

print("data_pca.head()")
print(data_pca.head())
```

	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

Principal Component Analysis (PCA)

Step-5: Checking Co-relation between features after PCA

#Checking Co-relation between features after PCA
sns.heatmap(data_pca.corr())



Principal Component Analysis (PCA) Code

```
#################
# PCA
#     https://www.geeksforgeeks.org/reduce-data-dimentionality-using-pca-python/

# Import necessary libraries
from sklearn import datasets # to retrieve the iris Dataset
import pandas as pd # to load the dataframe
from sklearn.preprocessing import StandardScaler # to standardize the features
from sklearn.decomposition import PCA # to apply PCA
import seaborn as sns # to plot the heat maps

#Load the Dataset
iris = datasets.load_iris()
#convert the dataset into a pandas data frame
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
#display the head (first 5 rows) of the dataset
print(df.head())

#Standardize the features
#Create an object of StandardScaler which is present in sklearn.preprocessing
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
print("scaled_data")
print(scaled_data)

#Check the Co-relation between features without PCA
sns.heatmap(scaled_data.corr())

#Applying PCA
#Taking no. of Principal Components as 3
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
print("data_pca.head()")
print(data_pca.head())

#Checking Co-relation between features after PCA
sns.heatmap(data_pca.corr())
```

References

- https://www.javatpoint.com/dimensionality-reduction-technique
-
- https://www.geeksforgeeks.org/dimensionality-reduction/

END

BCAC 0017
FUNDAMENTALS OF MACHINE LEARNING

Section 8 Supervised Learning:

Faculty Name
Mr. Sachin Sharma, Dr. Vinod Jain, Dr. Manu Banga,
Ms.Paromita Goswami, Ms.Chestha Bhardwaj

Section 8 : Supervised Learning:

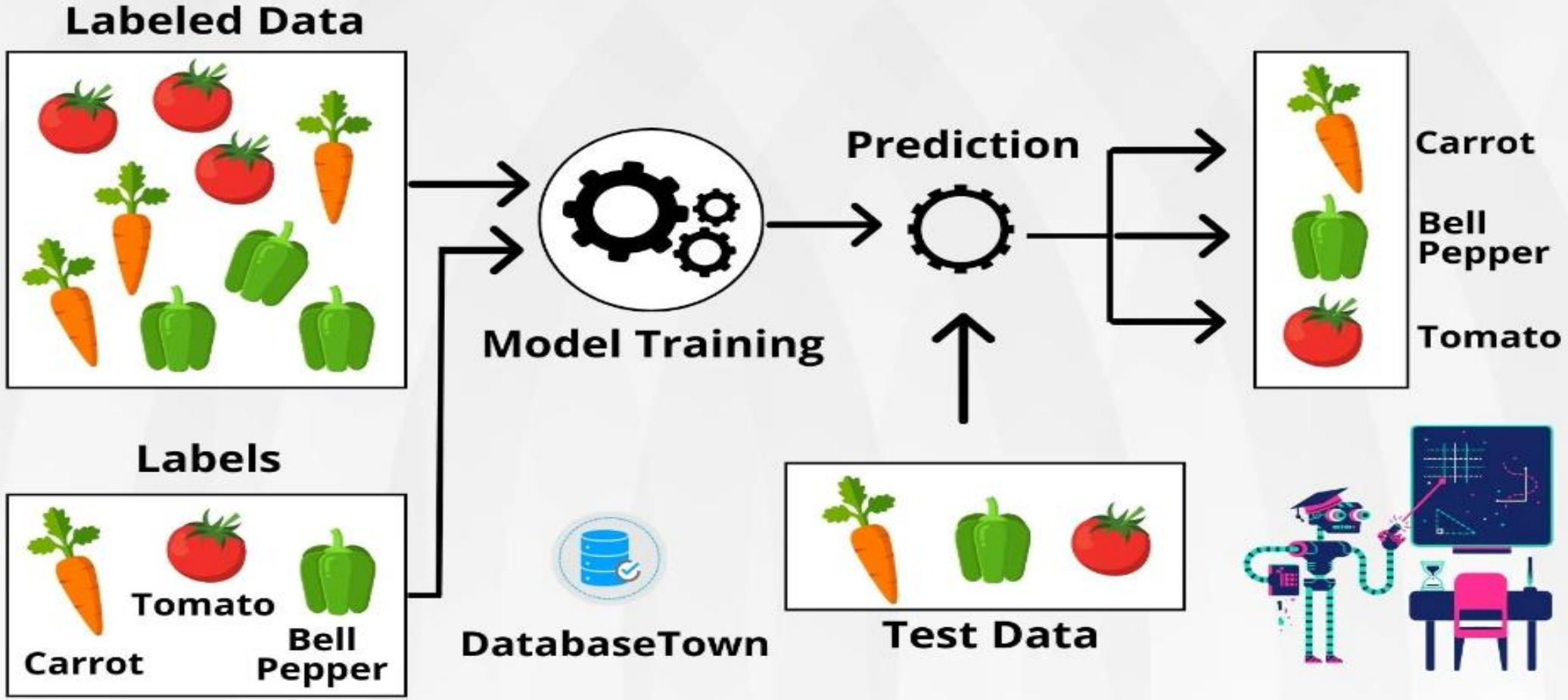
- **Supervised Learning:**
- Support Vector Machine
- Decision Tree
- Naïve Bayes Classifier.

Supervised Learning:

- **Supervised Learning:**
- Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output.
- The labelled data means some input data is already tagged with the correct output.

SUPERVISED LEARNING

Supervised machine learning is a branch of artificial intelligence that focuses on training models to make predictions or decisions based on labeled training data.



How Supervised Learning Works?

Supervised Learning:

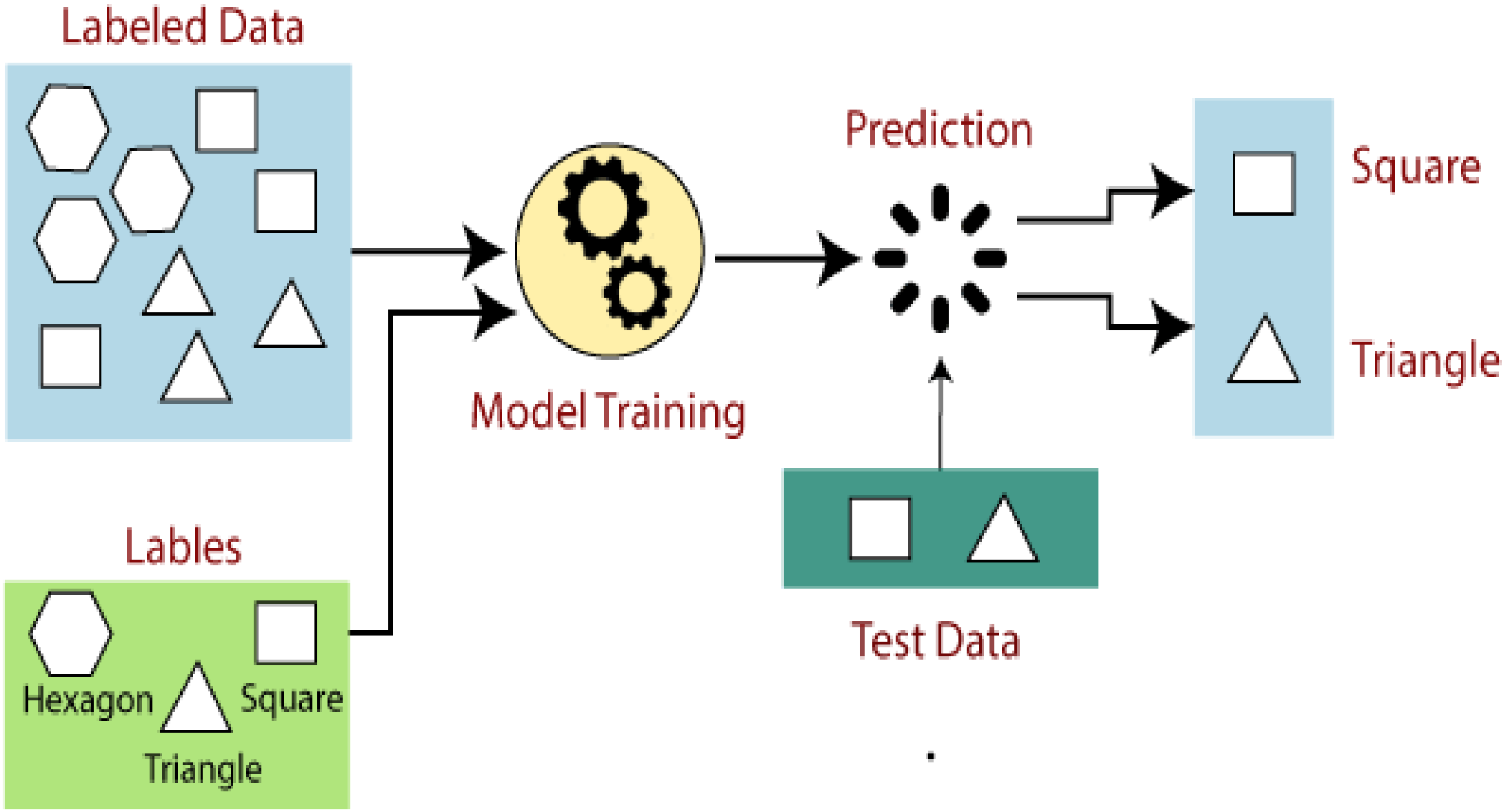
- **Supervised Learning:**
- In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly.
- It applies the same concept as a student learns in the supervision of the teacher.

Supervised Learning:

- **Supervised Learning:**
- Supervised learning is a process of providing input data as well as correct output data to the machine learning model.
- The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

Supervised Learning:

- **How Supervised Learning Works?**
- In supervised learning, models are trained using labelled dataset, where the model learns about each type of data.
- Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.


Labeled Data
Lables
Hexagon
Square
Triangle
Model Training
Prediction
Test Data
Square
Triangle

Supervised Learning:

- Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon.
- Now the first step is that we need to train the model for each shape.

1. If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
2. If the given shape has three sides, then it will be labelled as a **triangle**.
3. If the given shape has six equal sides then it will be labelled as **hexagon**.

Supervised Learning:

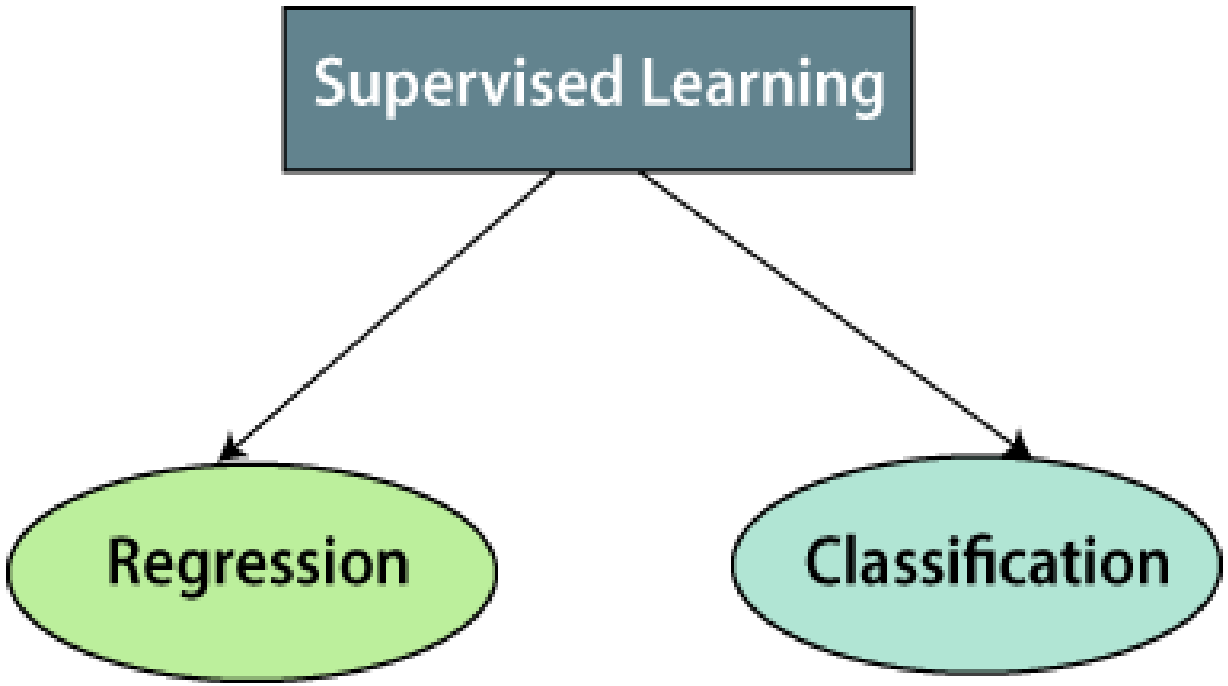
- Now, after training, we test our model using the test set, and the task of the model is to identify the shape.
- The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Steps Involved in Supervised Learning:

1. First Determine the type of training dataset
2. Collect/Gather the labelled training data.
3. Split the training dataset into training **dataset, test dataset, and validation dataset**.
4. Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
5. Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
6. Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
7. Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Supervised Learning:

- **Types of supervised Machine learning Algorithms:**
- Supervised learning can be further divided into two types of problems:



Supervised Learning:

- **1. Regression**
- Regression algorithms are used if there is a relationship between the input variable and the output variable.
- It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc.
- Below are some popular Regression algorithms which come under supervised learning:

1. Linear Regression
2. Regression Trees
3. Non-Linear Regression
4. Bayesian Linear Regression
5. Polynomial Regression

Supervised Learning:

- **2. Classification**
- Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

1. **Support Vector Machine**
2. **Decision Tree**
3. **Naïve Bayes Classifier.**
4. Random Forest
5. Logistic Regression
6. Etc.

Supervised Learning:

- **Advantages of Supervised learning:**

1. With the help of supervised learning, the model can predict the output on the basis of prior experiences.
2. In supervised learning, we can have an exact idea about the classes of objects.
3. Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Supervised Learning:

- **Disadvantages of supervised learning:**

1. Supervised learning models are not suitable for handling the complex tasks.
2. Supervised learning cannot predict the correct output if the test data is different from the training dataset.
3. Training required lots of computation times.
4. In supervised learning, we need enough knowledge about the classes of object.

References

- **https://www.javatpoint.com/supervised-machine-learning**

Support Vector Machine

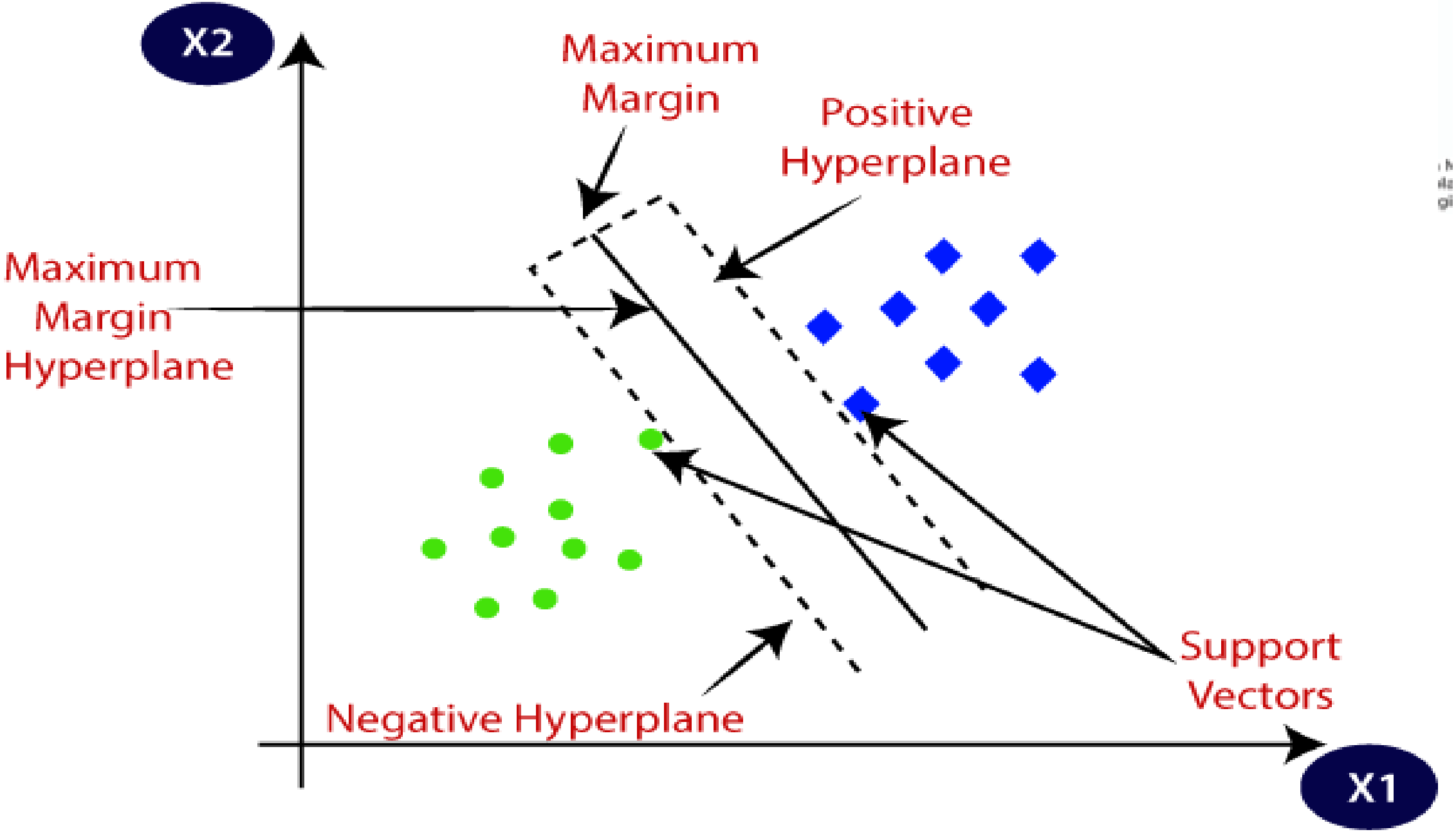
- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
- However, primarily, it is used for Classification problems in Machine Learning.

Support Vector Machine

- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.
- This best decision boundary is called a hyperplane.

Support Vector Machine

- SVM chooses the extreme points/vectors that help in creating the hyperplane.
- These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.
- Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:


X2
Maximum
Margin
Positive
Hyperplane
Maximum
Margin
Hyperplane
Support
Vectors
Negative Hyperplane
X1

Support Vector Machine

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane:

We need to find out the best decision boundary that helps to classify the data points.

This best boundary is known as the hyperplane of SVM.

If there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

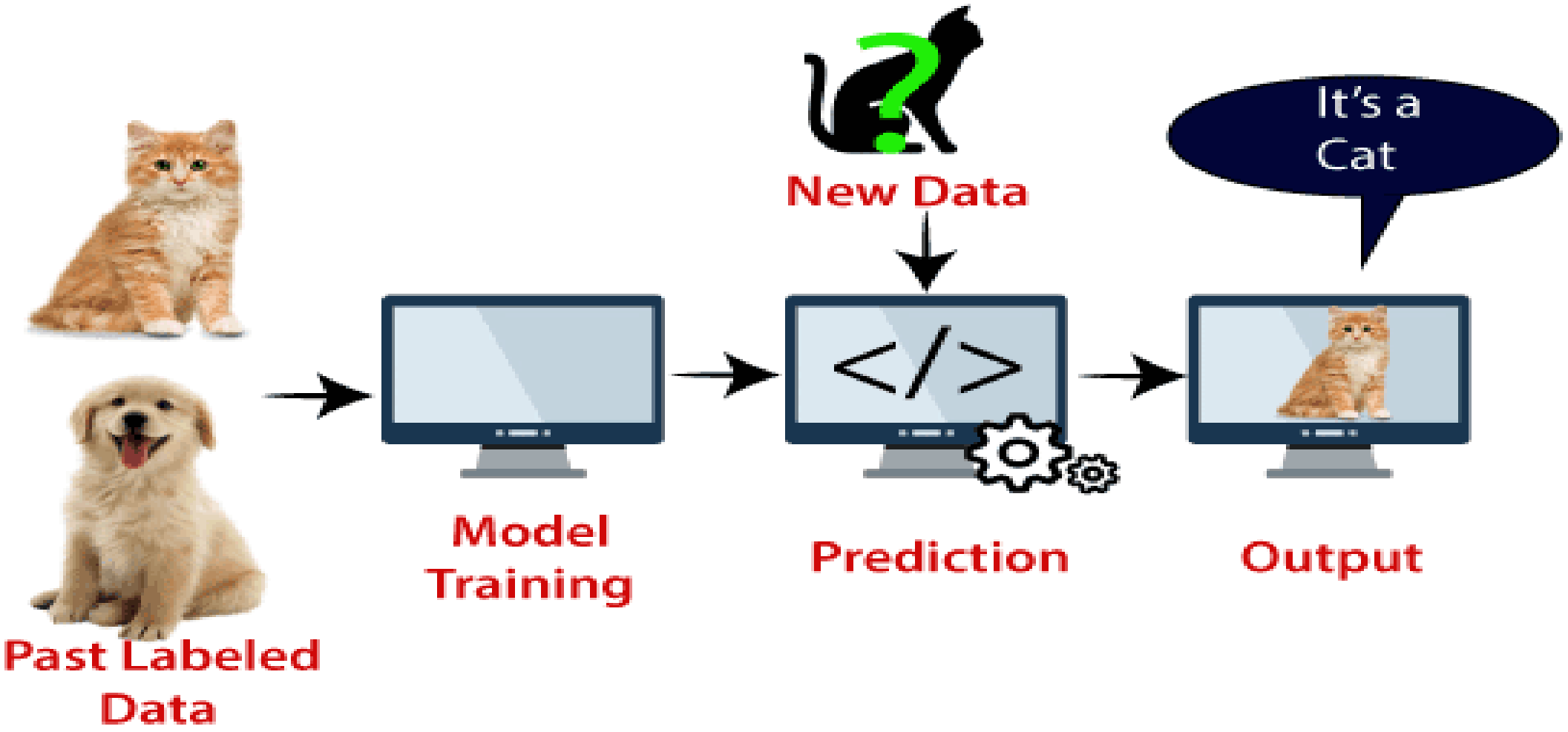
Support Vector Machine

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector.

Since these vectors support the hyperplane, hence called a Support vector.

Support Vector Machine



Support Vector Machine

- **Example**
- Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm.
- We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature.
- So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

Support Vector Machine

- SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Support Vector Machine

- **SVM can be of two types:**
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

```
# SVM
# https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)

#Predicting the test set result
y_pred= classifier.predict(x_test)

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print("Confusion Matrix is : ")
print(cm)

from sklearn.metrics import accuracy_score
score =accuracy_score(y_pred, y_test)
print("Test  Accuracy Score ")
print(score)
```

```
Confusion Matrix is :
[[66  2]
 [ 8 24]]
Test  Accuracy Score
0.9
```

Example 2

```
# SVM
# https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/

import numpy as np
from sklearn.datasets import make_classification
from sklearn import svm
from sklearn.model_selection import train_test_split

classes = 4
X,t= make_classification(100, 5, n_classes = classes, random_state= 40, n_informative = 2,
n_clusters_per_class = 1)

X_train, X_test, y_train, y_test=  train_test_split(X, t , test_size=0.50)

model = svm.SVC(kernel = 'linear', random_state = 0, C=1.0)

model.fit(X_train, y_train)

y=model.predict(X_test)
y2=model.predict(X_train)
```

```
from sklearn.metrics import accuracy_score
score =accuracy_score(y, y_test)
print("Test  Accuracy Score ")
print(score)

print("Train Accuracy Score ")
score2 =accuracy_score(y2, y_train)
print(score2)
```

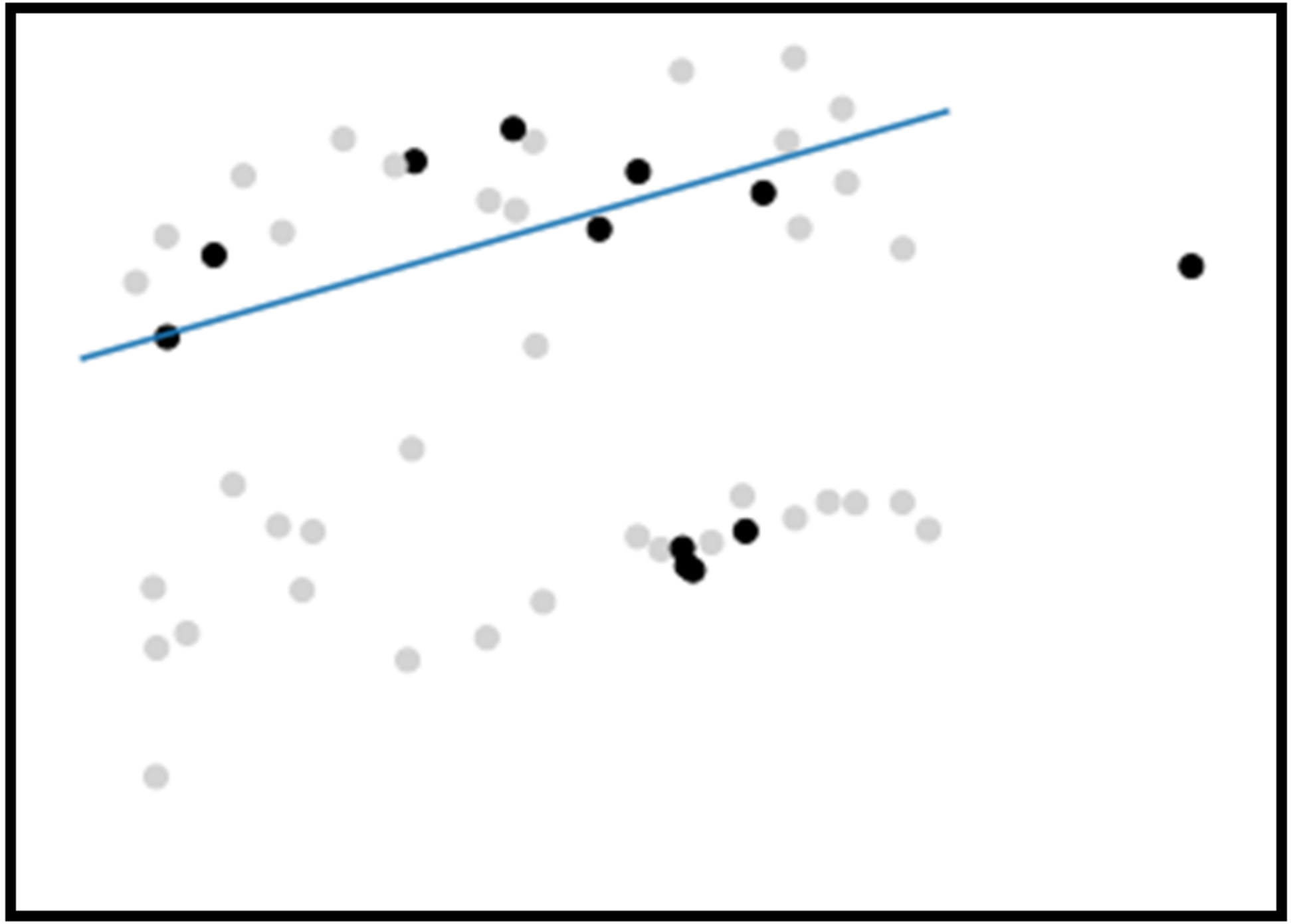
Test Accuracy Score

0.88

Train Accuracy Score

0.96

```
# Visualization of results.
import matplotlib.pyplot as plt
color = ['black' if c == 0 else 'lightgrey' for c in y]
plt.scatter(X_train[:,0], X_train[:,1], c=color)

# Create the hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the hyperplane
plt.plot(xx, yy)
plt.axis("off"), plt.show();
```



Support Vector Machine Code

```
# SVM
# https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/


import numpy as np
from sklearn.datasets import make_classification
from sklearn import svm
from sklearn.model_selection import train_test_split

classes = 4
X,t= make_classification(100, 5, n_classes = classes, random_state= 40, n_informative = 2, n_clusters_per_class = 1)


X_train, X_test, y_train, y_test= train_test_split(X, t , test_size=0.50)


model = svm.SVC(kernel = 'linear', random_state = 0, C=1.0)
model.fit(X_train, y_train)


y=model.predict(X_test)
y2=model.predict(X_train)


from sklearn.metrics import accuracy_score
score =accuracy_score(y, y_test)
print("Test  Accuracy Score ")
print(score)


print("Train Accuracy Score ")
score2 =accuracy_score(y2, y_train)
print(score2)


import matplotlib.pyplot as plt
color = ['black' if c == 0 else 'lightgrey' for c in y]
plt.scatter(X_train[:,0], X_train[:,1], c=color)

# Create the hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the hyperplane
plt.plot(xx, yy)
plt.axis("off"), plt.show();
```

sklearn.datasets.make_classification

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

sklearn.datasets.make_classification

```
sklearn.datasets.make_classification(n_samples=100, n_features=20, *, n_informative=2, n_redundant=2,
n_repeated=0, n_classes=2, n_clusters_per_class=2, weights=None, flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0,
scale=1.0, shuffle=True, random_state=None)                                                    [source]
```

Generate a random n-class classification problem.

sklearn.svm.SVC

Support Vector Classification.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

sklearn.svm.SVC

class sklearn.svm.**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

References

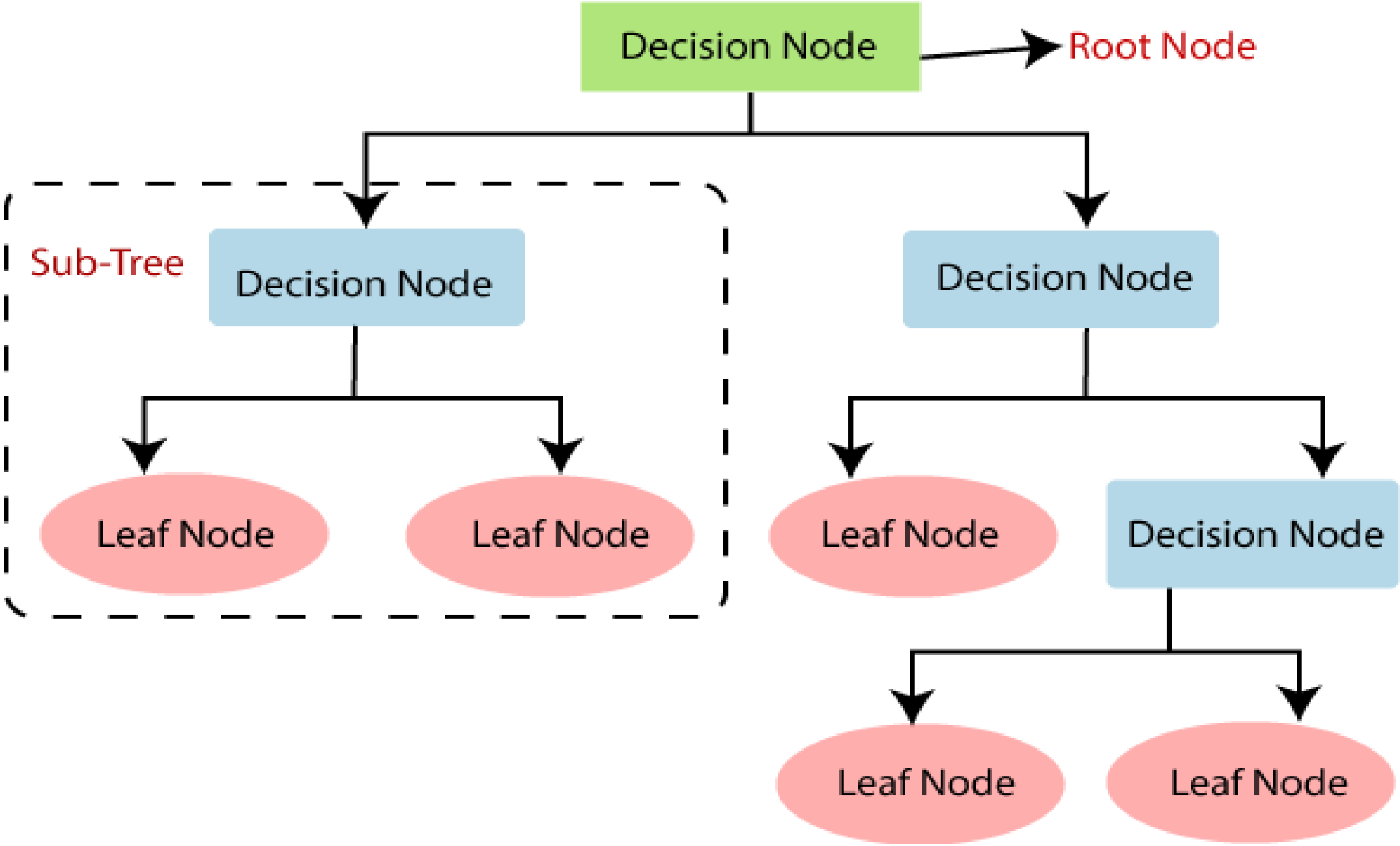
- https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/
- https://www.geeksforgeeks.org/support-vector-machine-algorithm/

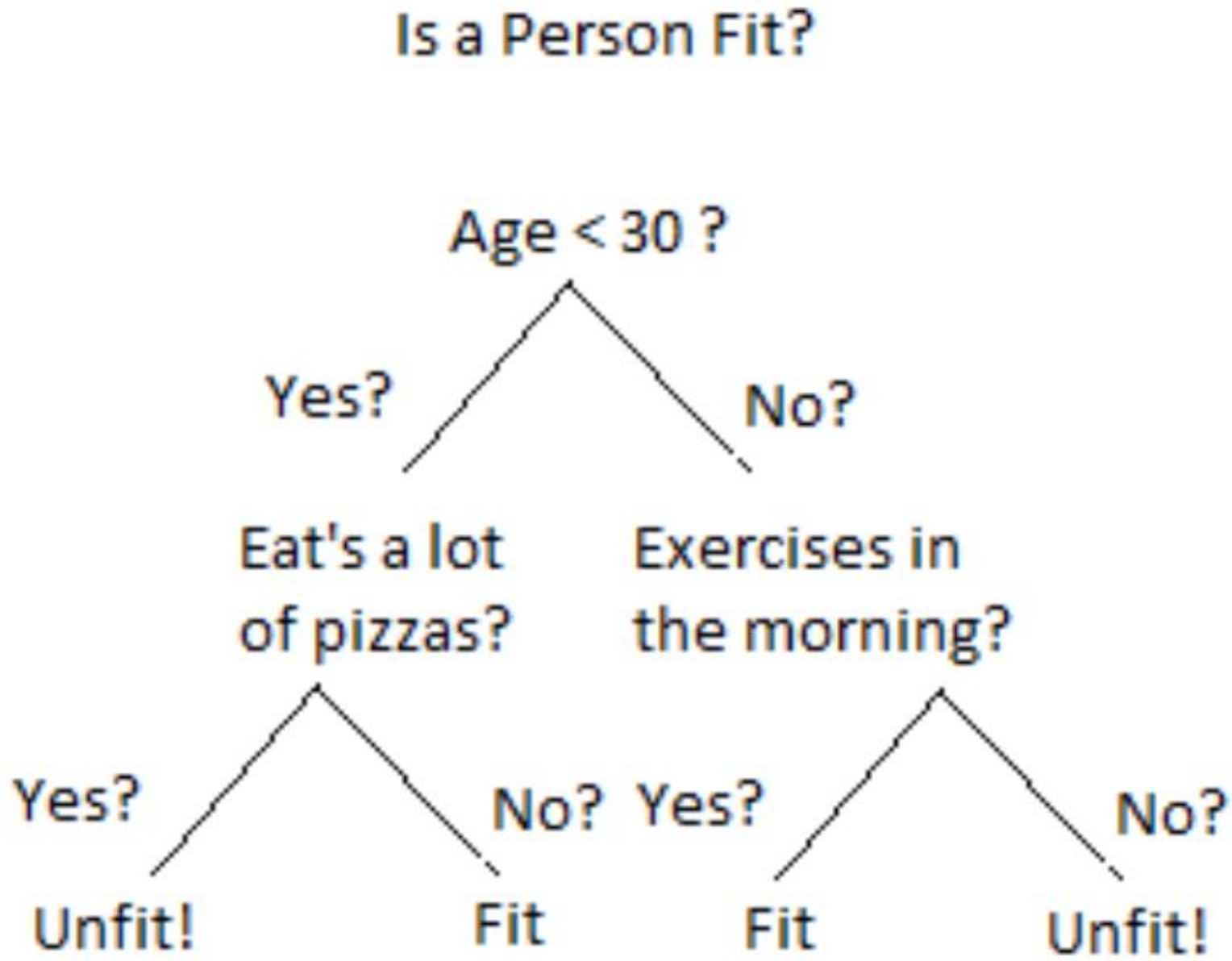
Decision Tree

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

Decision Tree

- It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**


Decision Node
Root Node
Sub-Tree
Decision Node
Decision Node
Leaf Node
Leaf Node
Leaf Node
Decision Node
Leaf Node
Leaf Node


Is a Person Fit?
Age < 30 ?
Yes?
No?
Eat's a lot of pizzas?
Exercises in the morning?
Yes?
No?
Yes?
No?
Unfit!
Fit
Fit
Unfit!

Decision Tree

- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.

Decision Tree

- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Decision Tree

- **Why use Decision Trees?**
- Below are the two reasons for using the Decision tree:
- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Decision Tree Terminologies

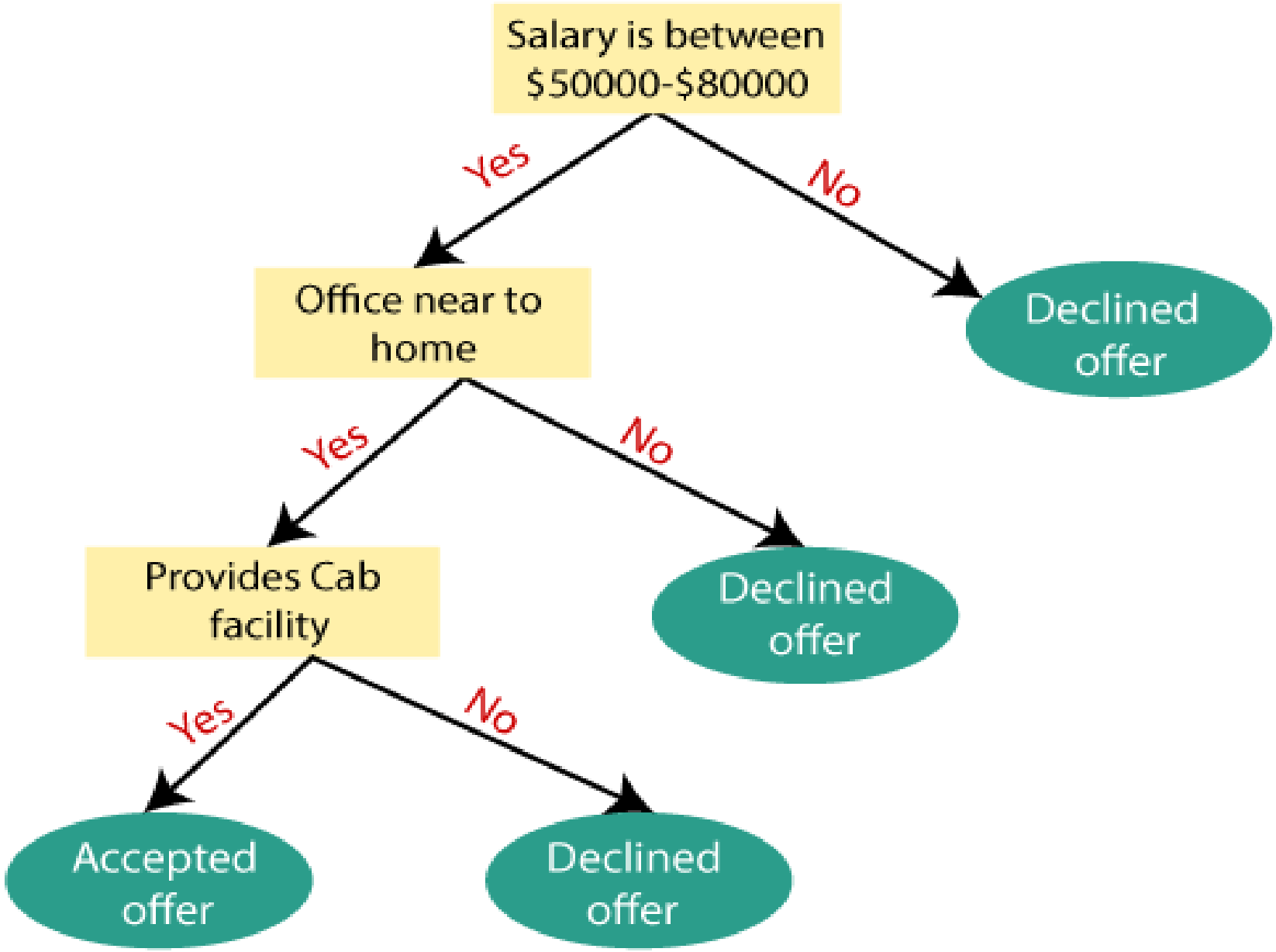
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

- **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.

Decision Tree

- **Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not.
- So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM).
- The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels.
- The next decision node further gets split into one decision node (Cab facility) and one leaf node.
- Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:


Salary is between $50000-$80000
Yes
No
Declined offer
Office near to home
Yes
No
Provides Cab facility
Declined offer
Yes
No
Accepted offer
Declined offer

Decision Tree

- **Advantages of the Decision Tree**
- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Decision Tree

- **Disadvantages of the Decision Tree**
- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

- Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models.
- By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, LogisticRegression, etc.

Decision Tree

- Steps will also remain the same, which are given below:
- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

```
######### Decision Tree 1

# https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

# data Set Download
# https://www.kaggle.com/datasets/erscodingzone/user-datacsv

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

#feature Scaling

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

#Fitting Decision Tree classifier to the training set

```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

#Predicting the test set result

```
y_pred= classifier.predict(x_test)
```

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

print(cm)
```

from sklearn.metrics import accuracy_score

```
score =accuracy_score(y_pred, y_test)
print("Test  Accuracy Score ")
print(score)
```

Confusion Matrix is :

[[62 6]

[3 29]]

Test Accuracy Score

0.91

Decision Tree : Code

```
######### Decision Tree 1

# https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

# data Set Download
# https://www.kaggle.com/datasets/erscodingzone/user-datacsv

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)

#Predicting the test set result
y_pred= classifier.predict(x_test)

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print("Confusion Matrix is : ")

print(cm)

from sklearn.metrics import accuracy_score
score =accuracy_score(y_pred, y_test)
print("Test  Accuracy Score ")
print(score)
```

References

- https://www.kaggle.com/datasets/erscodingzone/user-datacsv

- https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

Naïve Bayes Classifier.

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.

Naïve Bayes Classifier.

- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Naïve Bayes Classifier.

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

Naïve Bayes Classifier.

- **Working of Naïve Bayes' Classifier:**
- Working of Naïve Bayes' Classifier can be understood with the help of the below example:
- Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:
- Convert the given dataset into frequency tables.
- Generate Likelihood table by finding the probabilities of given features.
- Now, use Bayes theorem to calculate the posterior probability.
- **Problem**: If the weather is sunny, then the Player should play or not?
- **Solution**: To solve this, first consider the below dataset:

Naïve Bayes Classifier.

- **Advantages of Naïve Bayes Classifier:**
- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.
- **Disadvantages of Naïve Bayes Classifier:**
- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Naïve Bayes Classifier.

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

Naïve Bayes Classifier.

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

Python Implementation of the Naïve Bayes algorithm:

- **Steps to implement:**
- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Python Implementation of the Naïve Bayes algorithm:

```
# importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
# data Set Download
# https://www.kaggle.com/datasets/erscodingzone/user-datacsv

dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

 # Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix is : ")
print(cm)
```

```
from sklearn.metrics import accuracy_score
score =accuracy_score(y_pred, y_test)
print("Test  Accuracy Score ")
print(score)
```

Confusion Matrix is :
[[65 3]
[7 25]]
Test Accuracy Score
0.9

References

- # https://www.kaggle.com/datasets/erscodingzone/user-datacsv
- https://www.javatpoint.com/machine-learning-naive-bayes-classifier

END

BCAC 0017
FUNDAMENTALS OF MACHINE LEARNING

Section 9 : Unsupervised Learning:

Faculty Name
Mr. Sachin Sharma, Dr. Vinod Jain, Dr. Manu Banga,
Ms.Paromita Goswami, Ms.Chestha Bhardwaj

Section 9 : Unsupervised Learning:

- **Unsupervised Learning:**
- Clustering
- Hierarchical Clustering

Unsupervised Learning

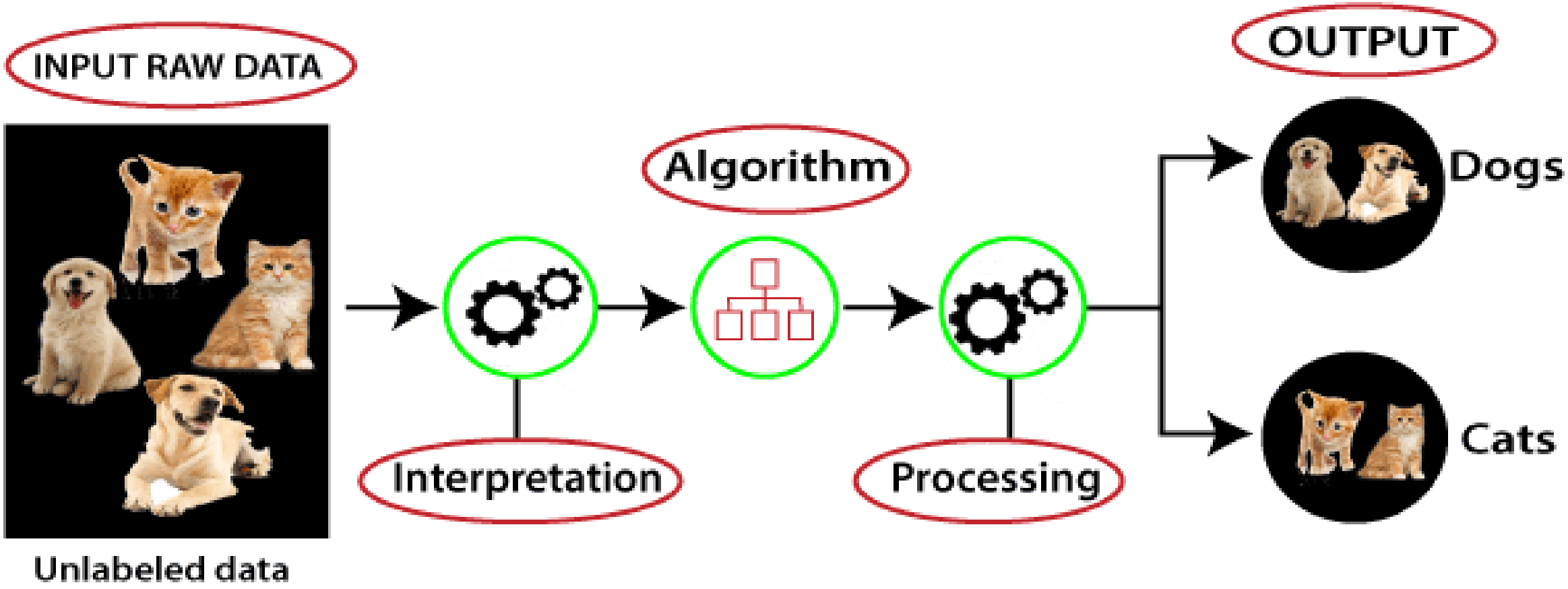
- *Unsupervised learning is a type of machine learning in which models are trained using **unlabeled dataset** and are allowed to act on that data without any supervision.*

Unsupervised Learning

- **Below are some main reasons which describe the importance of Unsupervised Learning:**

1. Unsupervised learning is helpful for finding useful insights from the data.
2. Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
3. Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
4. In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

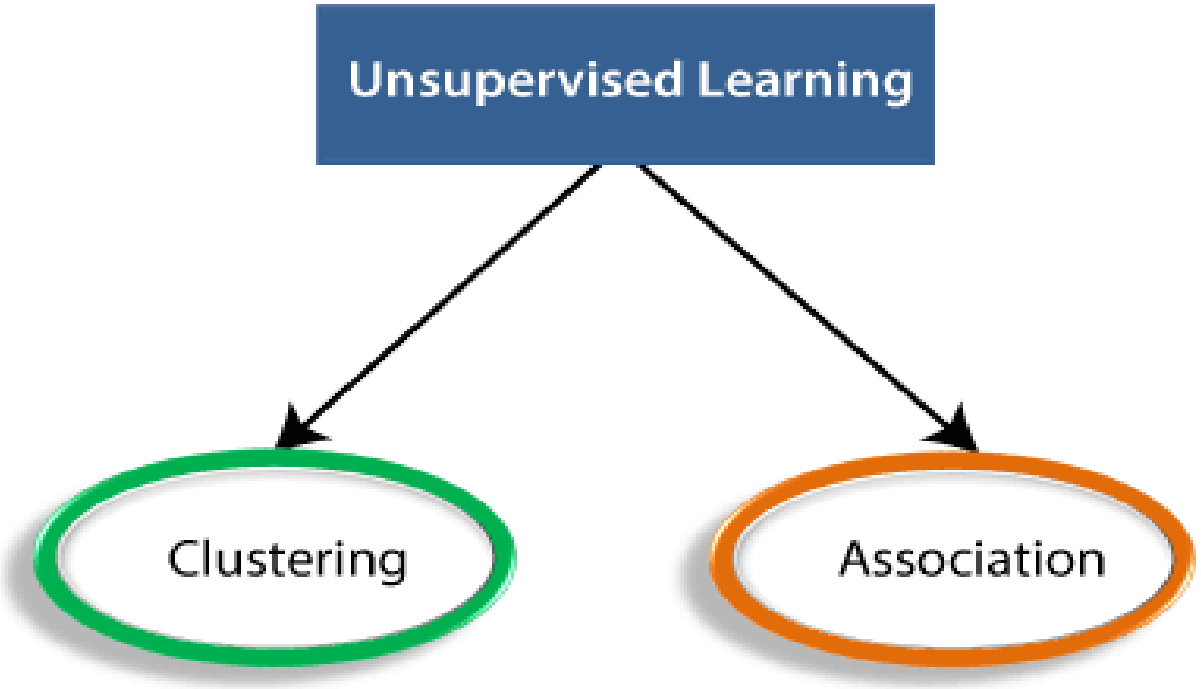


Unsupervised Learning

- Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given.
- Now, this unlabeled input data is fed to the machine learning model in order to train it.
- Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.
- Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Unsupervised Learning

- **Types of Unsupervised Learning Algorithm:**



Unsupervised Learning

- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group.
- Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

Unsupervised Learning

- **Association**:
- An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database.
- It determines the set of items that occurs together in the dataset.
- Association rule makes marketing strategy more effective.
- Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item.
- A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

- Below is the list of some popular unsupervised learning algorithms:
- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchal clustering**
- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Unsupervised Learning

- **Advantages of Unsupervised Learning**
- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data

Unsupervised Learning

- **Disadvantages of Unsupervised Learning**
- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

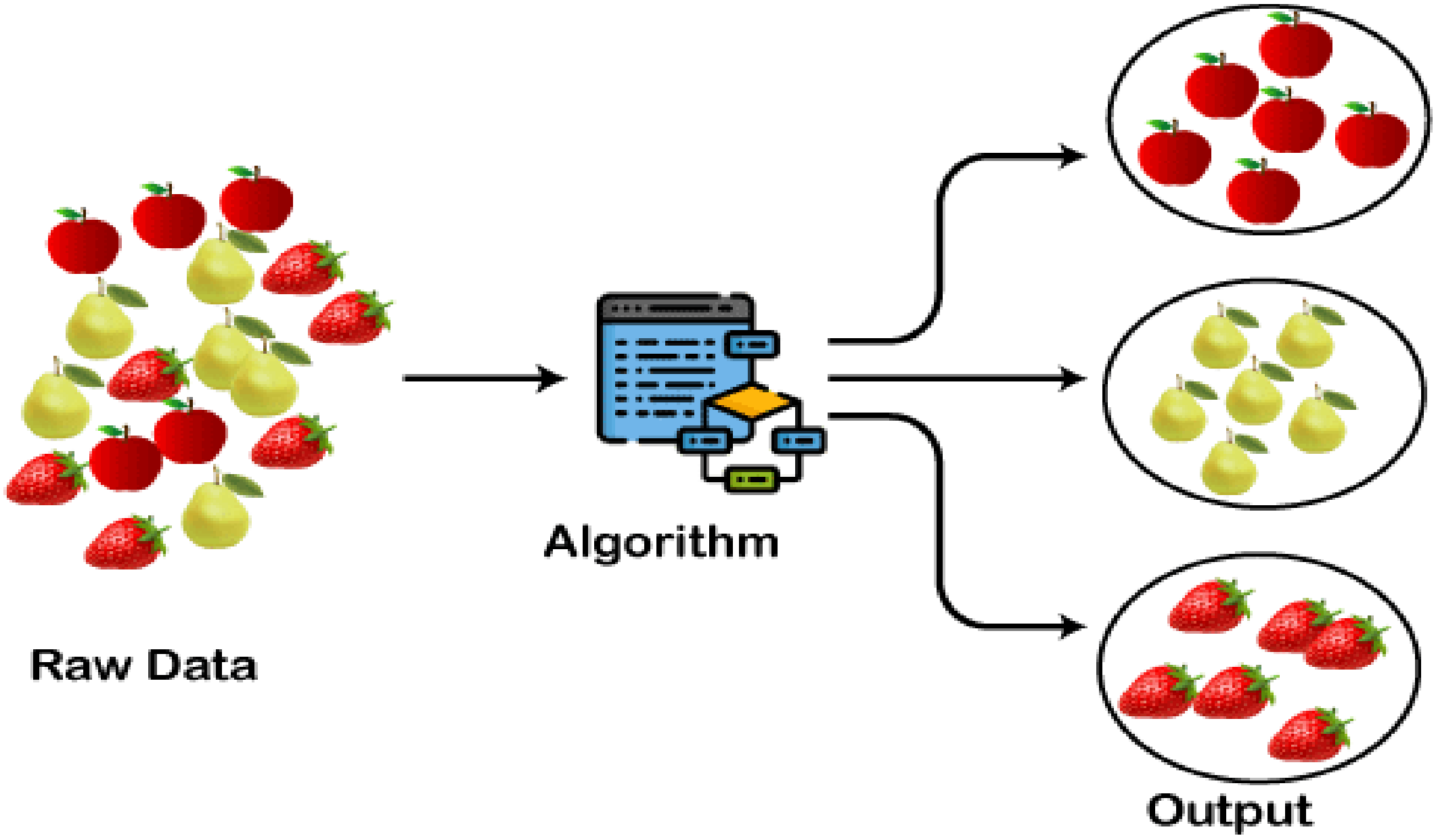
Clustering in Machine Learning

- Clustering

Clustering in Machine Learning

- It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

- It can be defined as
- ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***


Raw Data
Algorithm
Output

Clustering in Machine Learning

- The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

1. Market Segmentation
2. Statistical data analysis
3. Social network analysis
4. Image segmentation
5. Anomaly detection, etc.

Applications of Clustering

- **In Identification of Cancer Cells:** The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.
- **In Search Engines:** Search engines also work on the clustering technique. The search result appears based on the closest object to the search query.

Applications of Clustering

- **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.

- **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.

- **In Land Use:** The clustering technique is used in identifying the area of similar lands. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

Hierarchical Clustering

- Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

- In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

Hierarchical Clustering

- The hierarchical clustering technique has two approaches:

- **Agglomerative:**
  - **bottom-up** approach
- **Divisive:**
  - **top-down approach.**

Hierarchical Clustering

- The hierarchical clustering technique has two approaches:
- **Agglomerative:**
- Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

- **Divisive:**
- Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

Agglomerative Hierarchical clustering

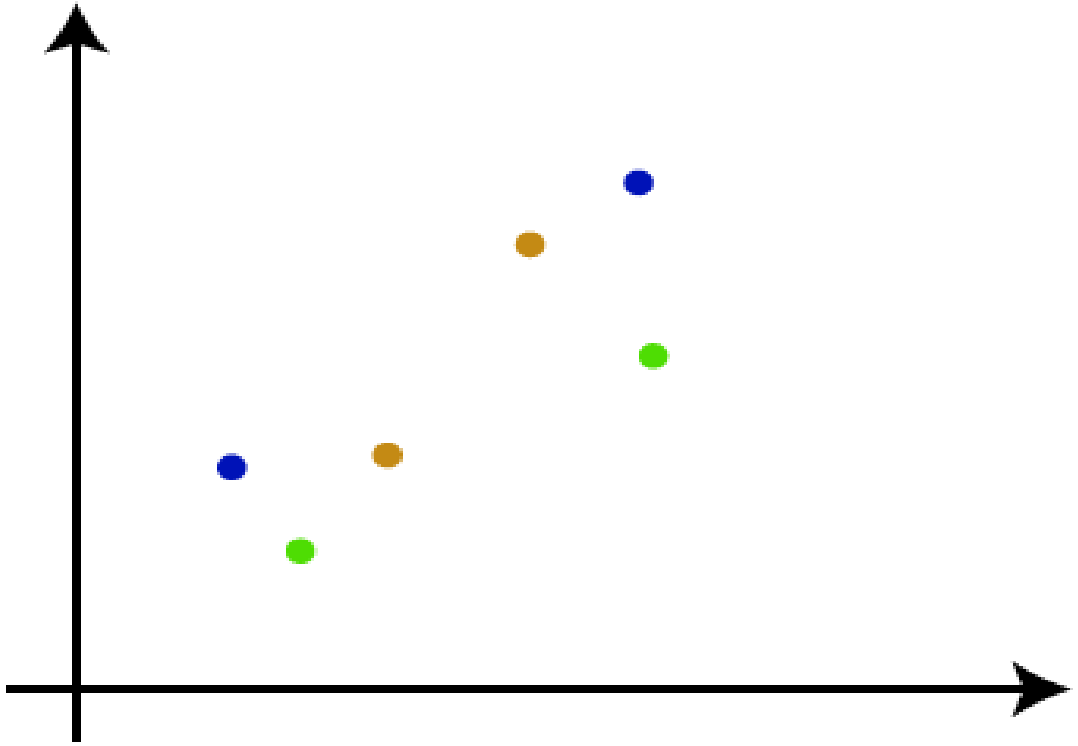
- The agglomerative hierarchical clustering algorithm is a popular example of HCA.
- To group the datasets into clusters, it follows the **bottom-up approach**.
- It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together

Agglomerative Hierarchical clustering

- **How the Agglomerative Hierarchical clustering Work?**
- The working of the AHC algorithm can be explained using the below steps:

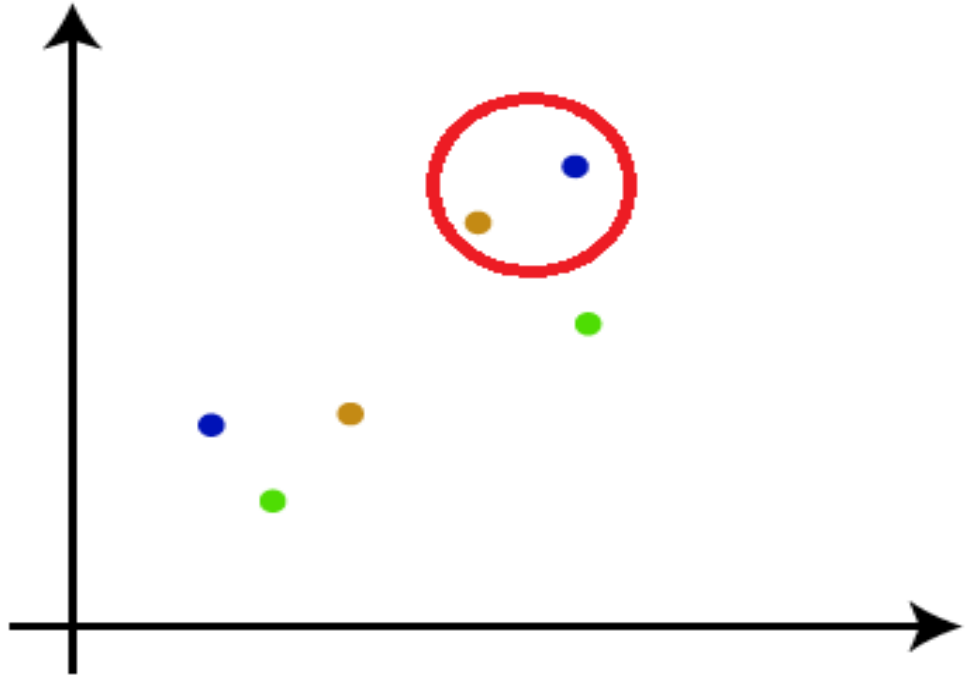
Agglomerative Hierarchical clustering

- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



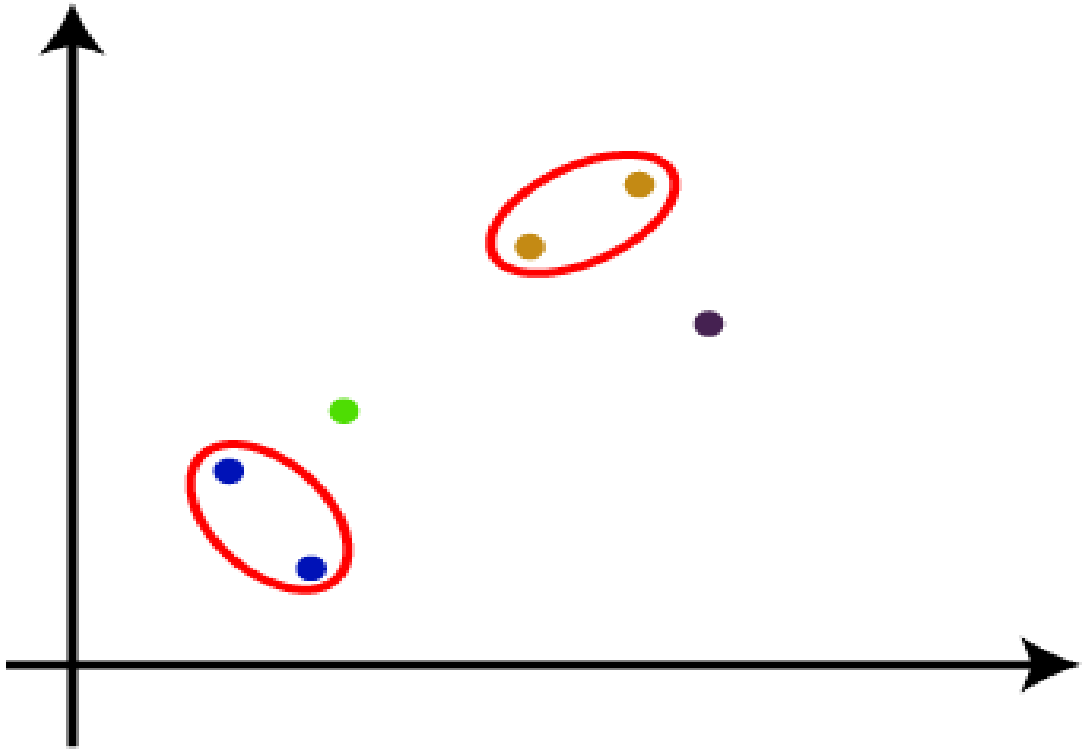
Agglomerative Hierarchical clustering

- **Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.



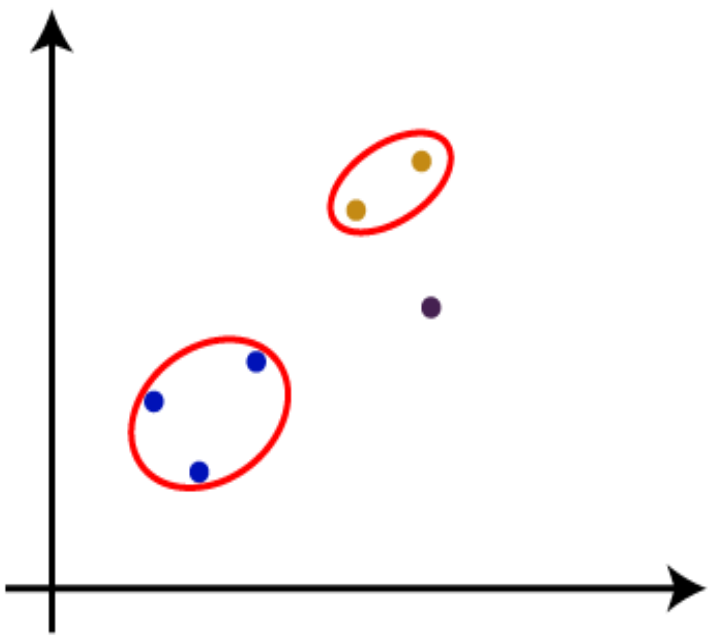
Agglomerative Hierarchical clustering

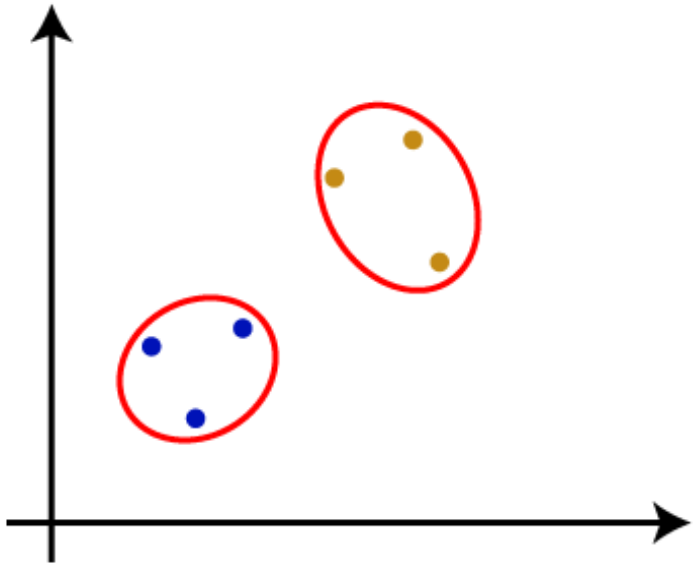
- **Step-3**: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.

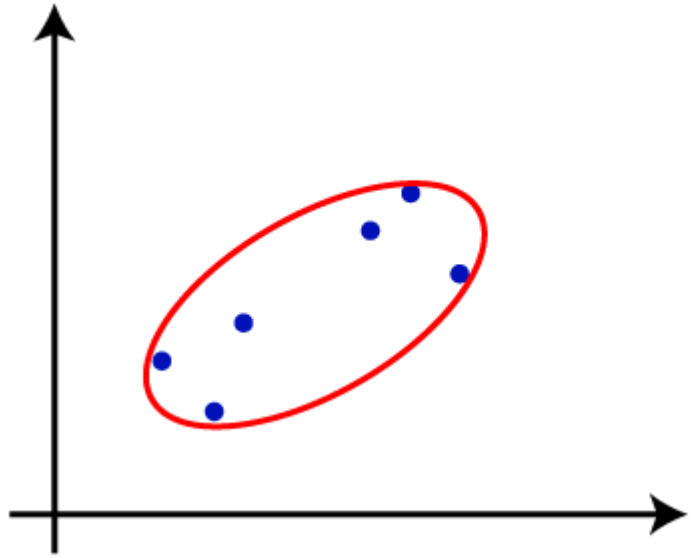


Agglomerative Hierarchical clustering

- **Step-4:** Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:

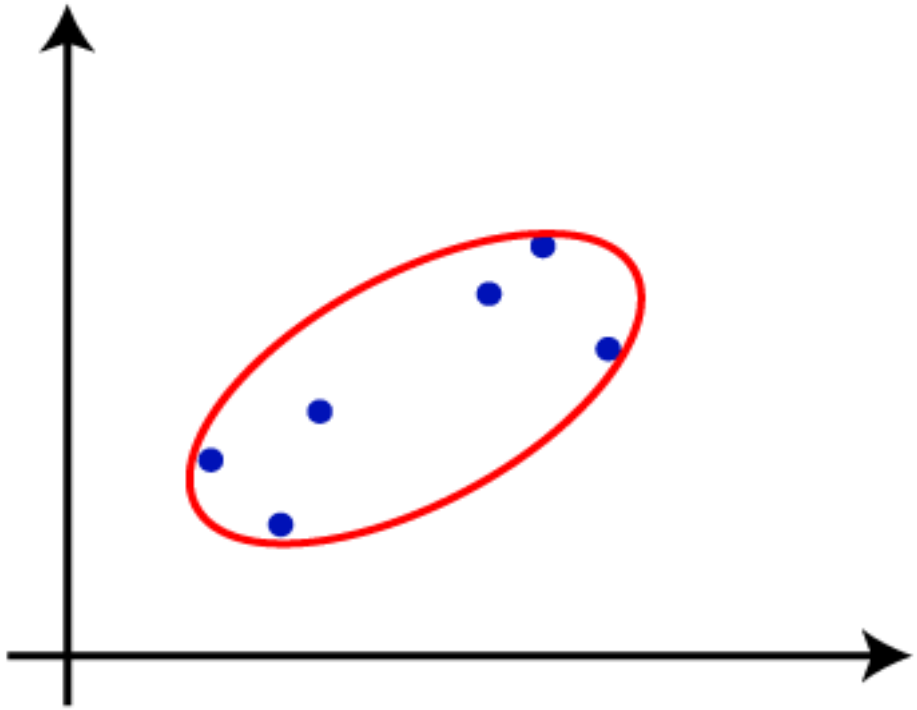






Agglomerative Hierarchical clustering

- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.



Measure for the distance between two clusters

- **Measure for the distance between two clusters**
- As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering.
- There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering.
- These measures are called **Linkage methods**.

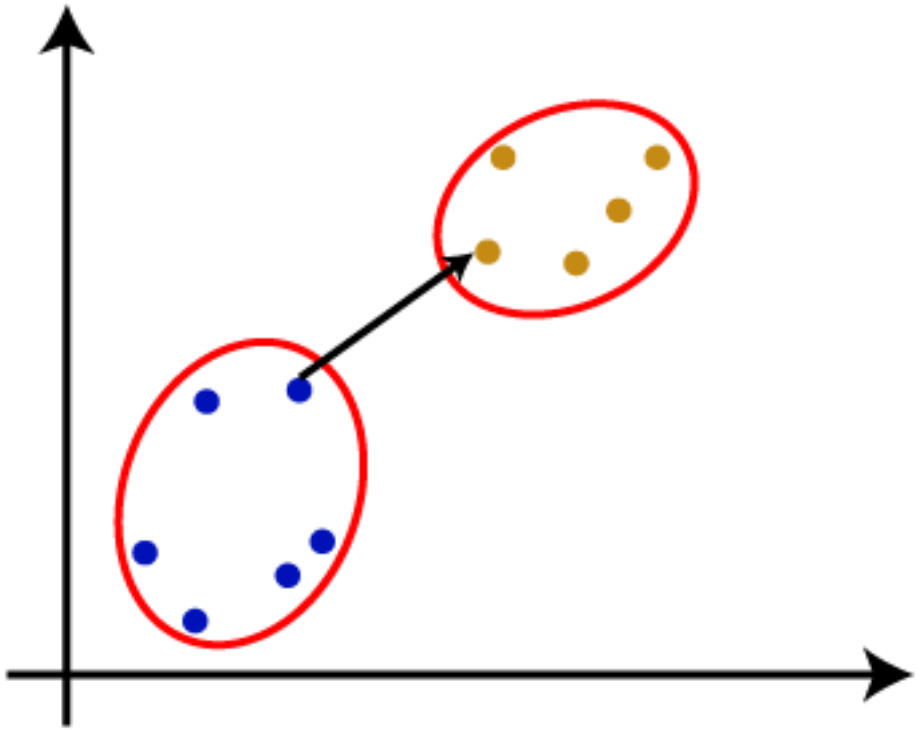
Measure for the distance between two clusters

- Some of the popular linkage methods are given below:

1. **Single Linkage**
2. **Complete Linkage**
3. **Average Linkage**
4. **Centroid Linkage**

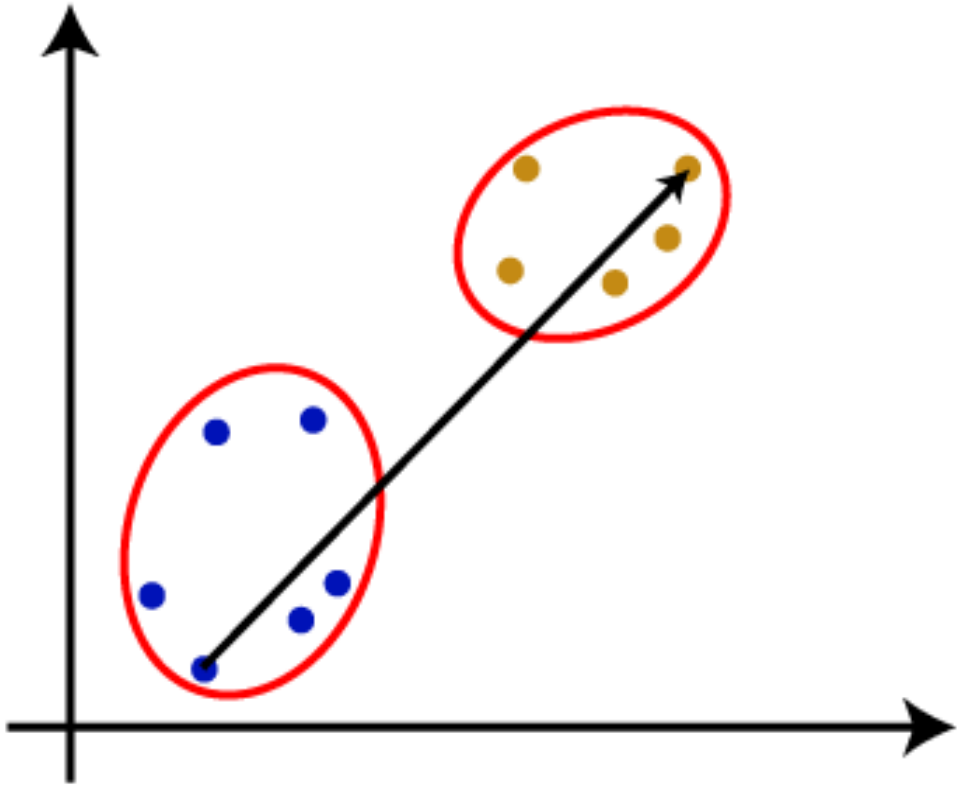
Measure for the distance between two clusters

- **Single Linkage:** It is the Shortest Distance between the **closest** points of the clusters. Consider the below image:



Measure for the distance between two clusters

- **Complete Linkage:** It is the **farthest** distance between the two points of two different clusters.

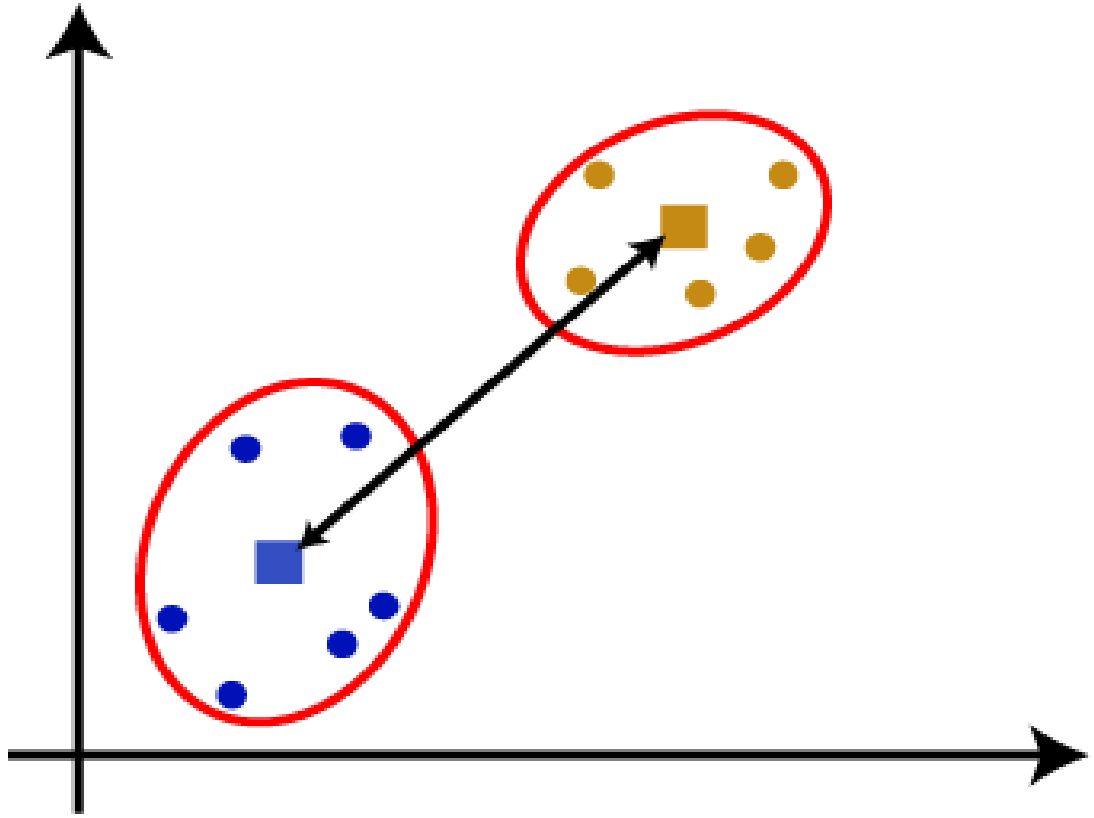


Measure for the distance between two clusters

- **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then the average is taken.

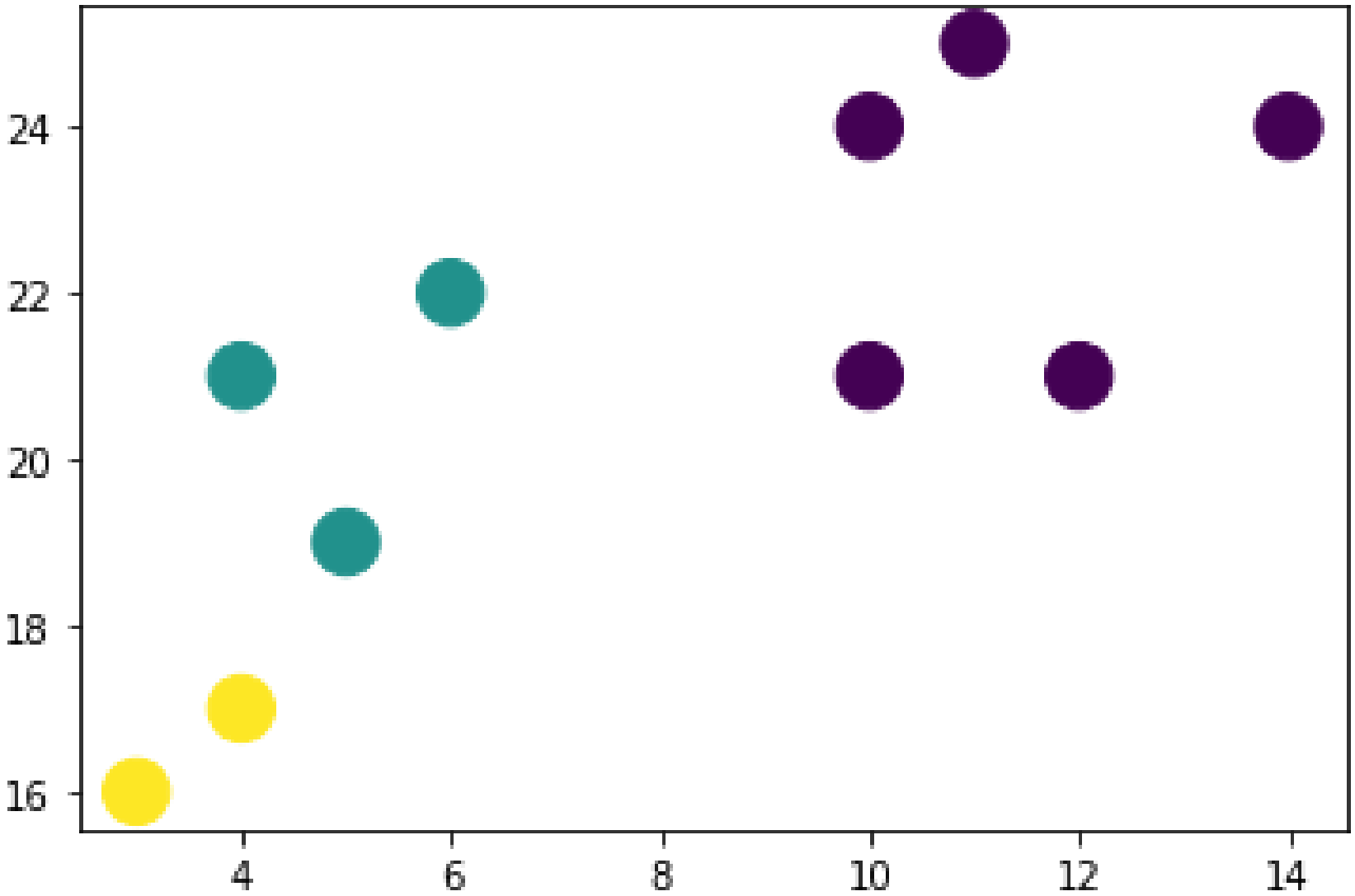
Measure for the distance between two clusters

- **Centroid Linkage:** It is the linkage method in which the distance between the **centroid** of the clusters is calculated. Consider the below image:

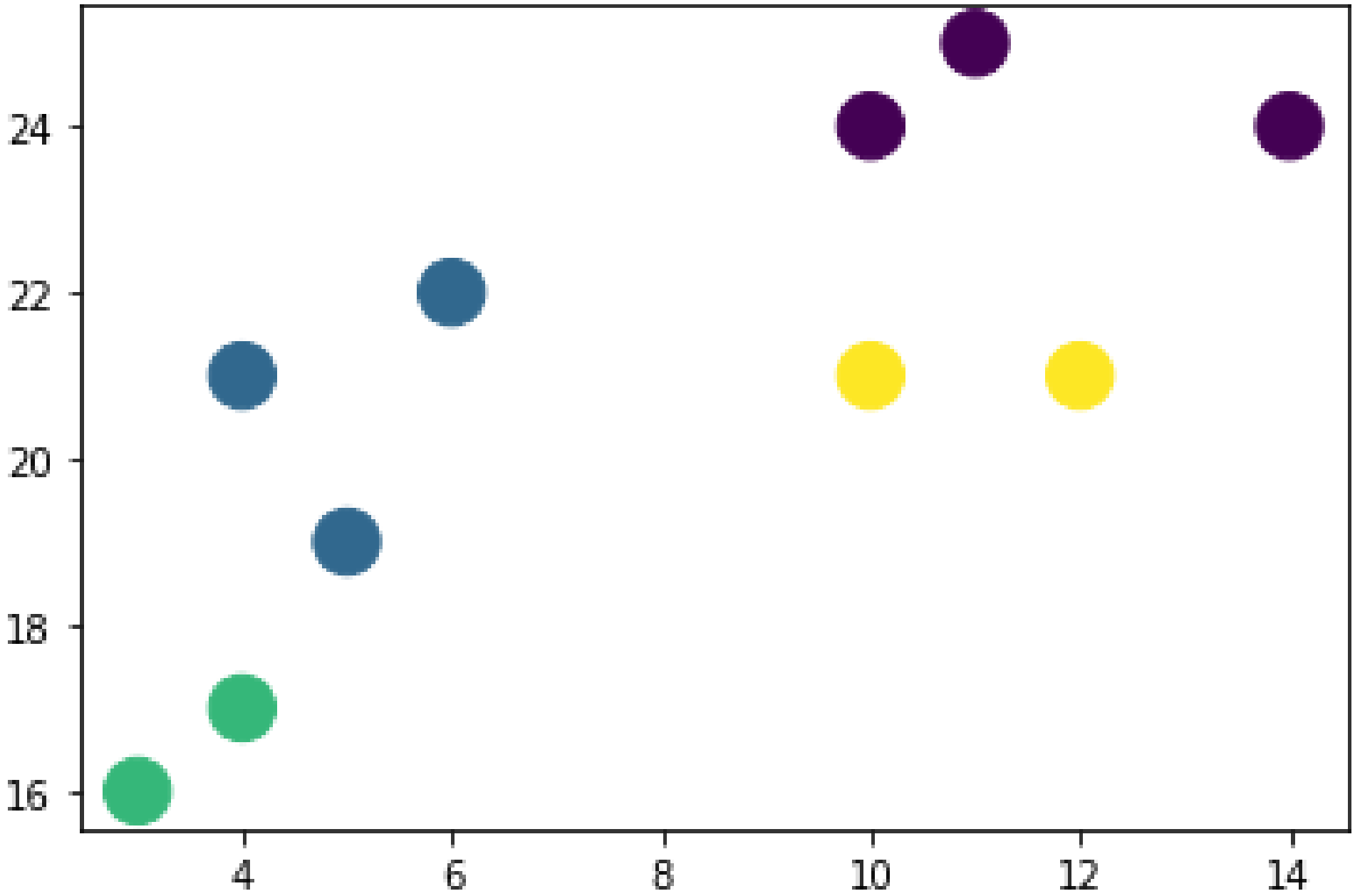


Python Implementation of Agglomerative Hierarchical Clustering

```
# AgglomerativeClustering
# https://www.w3schools.com/python/python_ml_hierarchial_clustering.asp
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
data = list(zip(x, y))
hierarchical_cluster = AgglomerativeClustering(n_clusters=3)    # 3 clusters
labels = hierarchical_cluster.fit_predict(data)
plt.scatter(x, y, c=labels, s=300) # s=marker size
plt.show()
```


24
22
20
18
16
4
6
8
10
12
14

```
# AgglomerativeClustering
# https://www.w3schools.com/python/python_ml_hierarchial_clustering.asp
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
data = list(zip(x, y))
hierarchical_cluster = AgglomerativeClustering(n_clusters=4)      # 4 clusters
labels = hierarchical_cluster.fit_predict(data)
plt.scatter(x, y, c=labels, s=300) # s=marker size
plt.show()
```

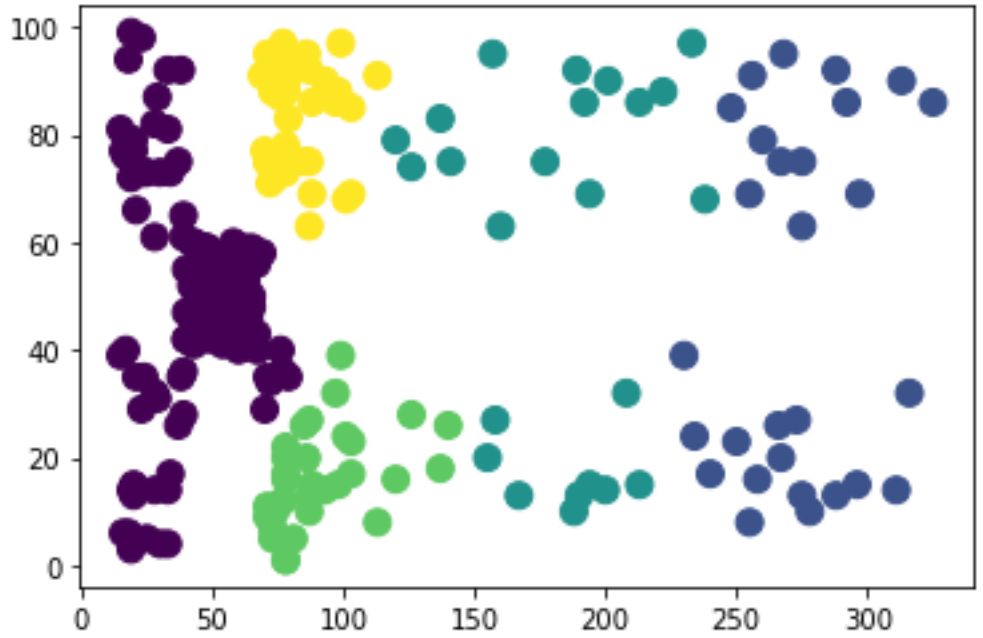

24
22
20
18
16
4
6
8
10
12
14

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=4)      # 4 clusters
labels = hierarchical_cluster.fit_predict(data)
Print(labels)
```

[1 1 0 2 2 0 0 1 3 3]

lables will show the cluster number of the data point

```
# Agglomerative Hierarchical Clustering
# https://www.javatpoint.com/hierarchical-clustering-in-machine-learning
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')

print(dataset.head())
x = dataset.iloc[:, [3, 4]].values

#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)

print(y_pred)

mtp.scatter(x[:,0], x[:,1], c=y_pred, s=100) # s=marker size
mtp.show()
```



References

- # https://www.javatpoint.com/hierarchical-clustering-in-machine-learning

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

- https://www.w3schools.com/python/python_ml_hierarchial_clustering.asp