

Artificial Intelligence

- ☐ Introduction
- ☐ EDA In Python
- ☐ Types Of Learning
- ☐ Regressions (3)
- ☐ Evaluation Metrics in Regression
- ☐ Classification
- ☐ Confusion Matrix in Classification
- ☐ Evaluation Metrics in Classification
- ☐ Validation, Cross validation : k-fold, stratified
- ☐ Hyperparameters

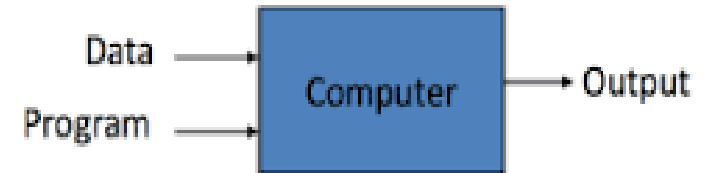
Machine Learning

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term “**Machine Learning**” as – “**Field of study that gives computers the capability to learn without being explicitly programmed**”.

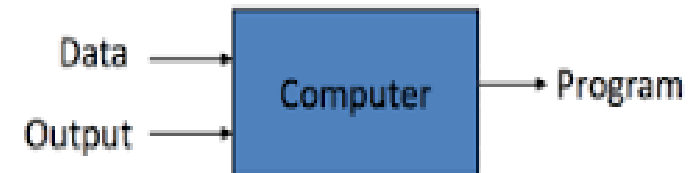
How it is different from traditional Programming:

- In Traditional Programming, we feed the Input, Program logic and run the program to get output.
- In Machine Learning, we feed the input, output and run it on machine during training and the machine creates its own logic, which is being evaluated while testing.

Traditional Programming



Machine Learning



Blocks of Machine Learning

Exploratory
Data
Analysis



PROCESSING

Blocks of Machine Learning

Data exploration

Formation of dataset

- ▶ Labelling
- ▶ Cleaning - NA,
non-compatible (data type, data form)

Qualifying for processing

Detailed report- correlations, indexes

Qualifying as per applied domain

Blocks- preprocessing,
outliers,
variable selections

EDA: Exploratory Data Analysis using Python

Exploratory Data Analysis - Data understanding is a crucial step in the data science process. Python has libraries (like Pandas) make this step easier and more efficient.

Also, this can be achieved using following Visual tools

- ▶ **Pygwalker**
- ▶ **PivotTableJS**
- ▶ **ydata-profiling**

Python Function For Performing EDA Of Given Dataset: .describe()

It returns description of the data in the DataFrame.

count - The number of not-empty values.

mean - The average (mean) value.

std - The standard deviation.

min - the minimum value.

Q1 - The 25% percentile.

Q2 - The 50% percentile.

Q3 - The 75% percentile.

max - the maximum value.

Example

```
import pandas as pd

data = [[10, 18, 11], [13, 15, 8], [9, 20, 3]]

df = pd.DataFrame(data)

print(df.describe())
```

	0	1	2
count	3.000000	3.000000	3.000000
mean	10.666667	17.666667	7.333333
std	2.081666	2.516611	4.041452
min	9.000000	15.000000	3.000000
25%	9.500000	16.500000	5.500000
50%	10.000000	18.000000	8.000000
75%	11.500000	19.000000	9.500000
max	13.000000	20.000000	11.000000

Customizing the “.describe()” Method

While the default summary statistics provided by .describe() are useful, you might want to customize them to better suit your needs. You can do this by passing a list of percentiles to the percentiles parameter.

```
print(df.describe(percentiles=[.10, .20, .30, .40, .50, .60, .70, .80, .90]))
```

This will display the 10th, 20th, 30th, 40th, 50th, 60th, 70th, 80th, and 90th percentiles of your data.

Terminologies that one should know before starting Machine Learning:

- ❑ **Model:** A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called **hypothesis**.
- ❑ **Feature:** A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc.**
- ❑ **Target(Label):** A target variable or label is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- ❑ **Training:** The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- ❑ **Prediction:** Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).

TYPES OF LEARNING

- Supervised Learning
- Unsupervised Learning
- Semi-Supervised Learning

1. **Supervised Learning:** Supervised learning is when the model is getting trained on a labelled dataset. **Labelled** dataset is one which have both input and output parameters. In this type of learning both training and validation datasets are labelled as shown in the figures below.

User ID	Gender	Age	Salary	Purchased	Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
15624510	Male	19	19000	0	10.69261758	986.882019	54.19337313	195.7150879	3.278597116
15810944	Male	35	20000	1	13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
15668575	Female	26	43000	0	17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
15603246	Female	27	57000	0	20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
15804002	Male	19	76000	1	22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
15728773	Male	27	58000	1	24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
15598044	Female	27	84000	0	24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
15694829	Female	32	150000	1	23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
15600575	Male	25	33000	1	22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
15727311	Female	35	65000	0	20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
15570769	Female	26	80000	1	17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
15606274	Female	26	52000	0	11.96680746	989.93861	60.16406036	1.650181436	

Classification

Regression

Types of Supervised Learning:

- Classification
- Regression

Classification : It is a Supervised Learning task where output is having defined labels(discrete value). For example in above Figure A, Output - Purchased has defined labels i.e. 0 or 1 ; 1 means the customer will purchase and 0 means that customer won't purchase. It can be either binary or multi class classification. In **binary** classification, model predicts either 0 or 1 ; yes or no but in case of **multi class** classification, model predicts more than one class.

Example: Gmail classifies mails in more than one classes like social, promotions, updates, offers.

Regression : It is a Supervised Learning task where output is having continuous value.

Example in before regression Figure, Output - Wind Speed is not having any discrete value but is continuous in the particular range. The goal here is to predict a value as much closer to actual output value as our model can and then evaluation is done by calculating error value. The smaller the error the greater the accuracy of our regression model.

Example of Supervised Learning Algorithms:

- ☐ Linear Regression
- ☐ Nearest Neighbor
- ☐ Gaussian Naive Bayes
- ☐ Decision Trees
- ☐ Support Vector Machine (SVM)
- ☐ Random Forest

Unsupervised Learning:

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised machine learning is more challenging than supervised learning due to the absence of labels.

Types of Unsupervised Learning:

- ☐ **Clustering**

- ☐ **Association**

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Examples of unsupervised learning algorithms are:

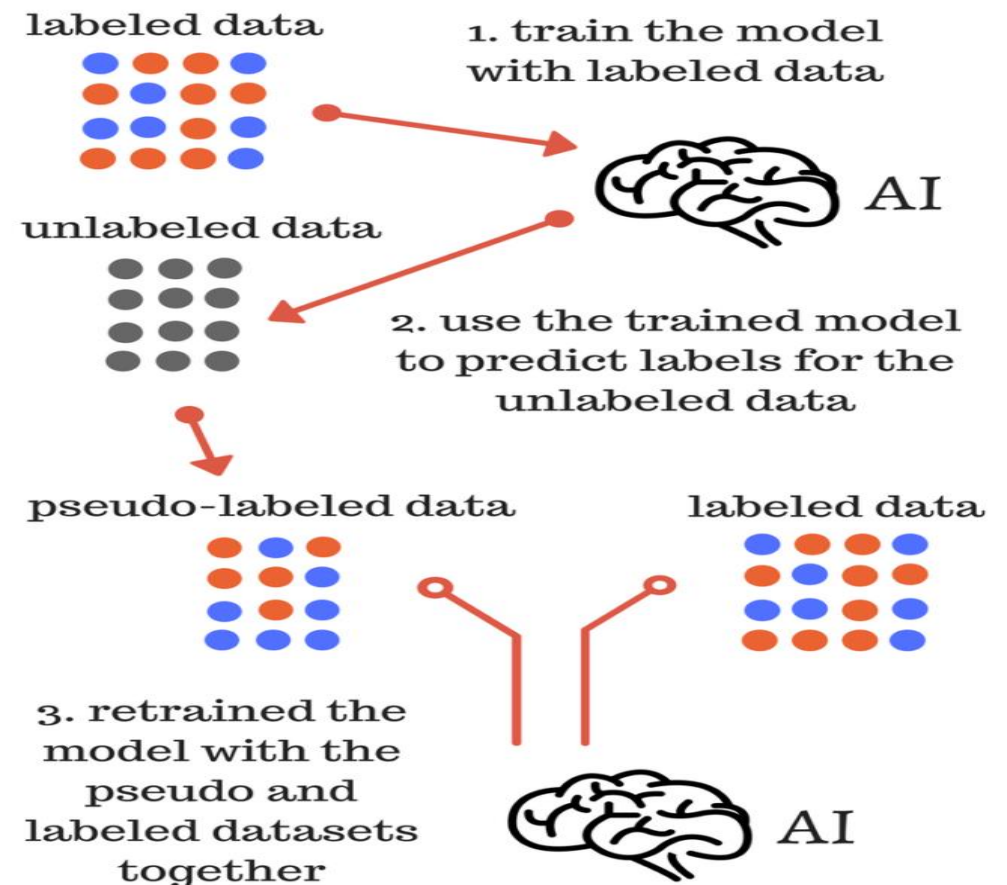
- ☐ k-means for clustering problems.
- ☐ Apriori algorithm for association rule learning problems

The most basic disadvantage of any **Supervised Learning** algorithm is that the dataset has to be hand-labeled either by a Machine Learning Engineer or a Data Scientist. This is a very *costly process*, especially when dealing with large volumes of data. The most basic disadvantage of any **Unsupervised Learning** is that its **application spectrum is limited**.

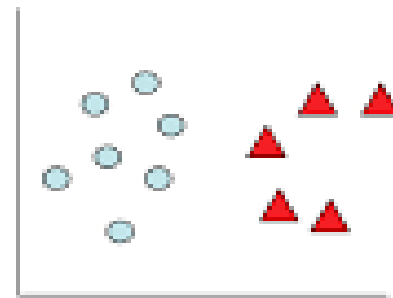
Semi-supervised machine learning:

To counter these disadvantages, the concept of **Semi-Supervised Learning** was introduced. In this type of learning, the algorithm is trained upon a combination of labeled and unlabeled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabeled data.

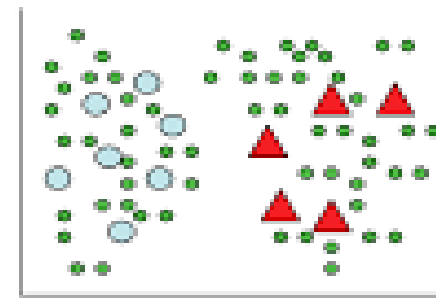
- In semi supervised learning labelled data is used to learn a model and using that model unlabeled data is labelled called pseudo labelling now using whole data model is trained for further use



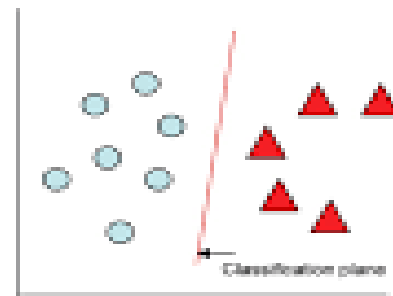
Model with labelled data and model with both labelled and unlabelled data



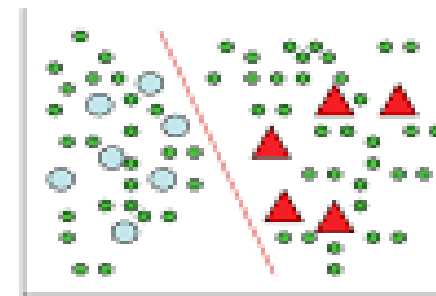
Labeled Data
(a)



Labeled and Unlabeled Data
(b)



Supervised Learning
(c)



Semi-Supervised Learning
(d)

Intuitively, one may imagine the three types of learning algorithms as Supervised learning where a student is under the supervision of a teacher at both home and school, Unsupervised learning where a student has to figure out a concept himself and Semi-Supervised learning where a teacher teaches a few concepts in class and gives questions as homework which are based on similar concepts.

REGRESSION

Regression is a statistical measurement that attempts to determine the strength of the relationship between one dependent variable and a series of other changing variables or independent variable.

Types of regression

- ▶ linear regression
 - ▶ Simple linear regression
 - ▶ Multiple linear regression
- ▶ Polynomial regression

Simple Linear regression

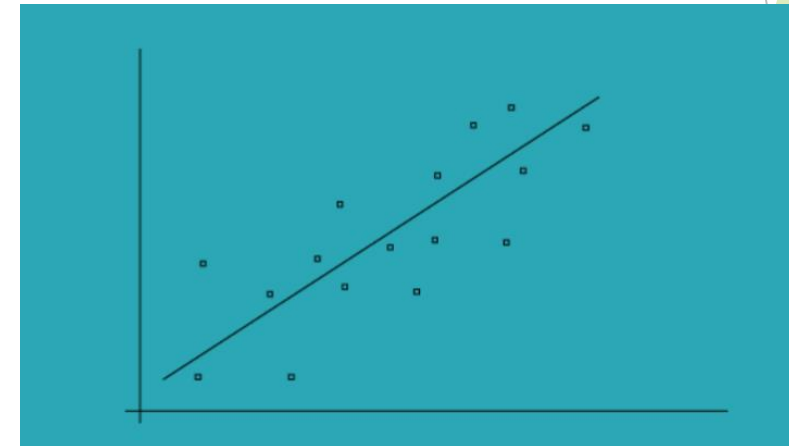
- ▶ The simple linear regression models are used to show or predict the relationship between the two variables or factors
- ▶ The factor that being predicted is called dependent variable and the factors that is are used to predict the dependent variable are called independent variables

Simple Linear Regression

$$y = b_0 + b_1 x_1$$

Constant Coefficient

Dependent variable (DV) Independent variable (IV)



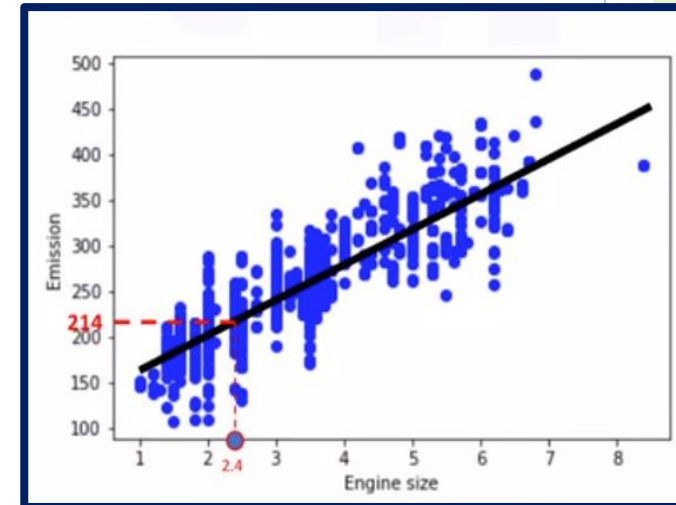
Simple Linear regression

Univariate Models

Multivariate Models

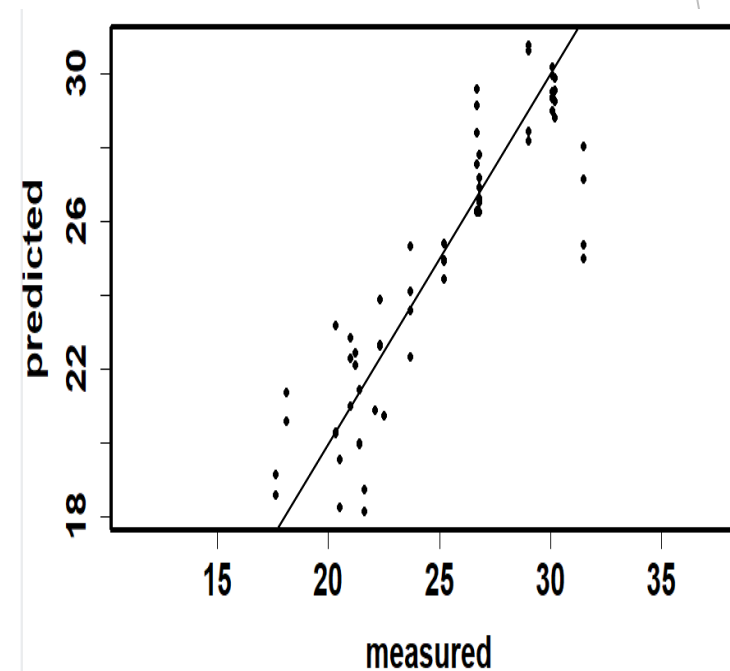
Predicting CO2 emission with engine size (only) is

	ENGINE SIZE	CYLINDERS	FUEL CONSUMPTION_COMB	CO2 EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	?



Predicting ‘standard/references’, from the given(Lab_1)data
is _____

standards/references	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359	1360
14.4	0.39855	0.39891	0.39926	0.39984	0.40015	0.40065	0.40116	0.40158	0.40195	0.40232	0.40277
14.4	0.40217	0.40265	0.40294	0.4034	0.40391	0.40439	0.40486	0.40536	0.40577	0.40623	0.4066
14.6	0.36531	0.3657	0.36601	0.36654	0.3668	0.36732	0.36764	0.36812	0.36852	0.36891	0.36942
14.6	0.43414	0.43456	0.43492	0.43543	0.43566	0.43627	0.43663	0.43715	0.43756	0.43799	0.43851
14.3	0.38688	0.3873	0.38767	0.38822	0.38851	0.38903	0.38941	0.38989	0.39031	0.39072	0.39125
14.3	0.39855	0.39891	0.39926	0.39984	0.40015	0.40065	0.40116	0.40158	0.40195	0.40232	0.40277
14.4	0.39475	0.39515	0.39546	0.39597	0.39623	0.39669	0.39705	0.39751	0.39796	0.39833	0.3988
14.4	0.38926	0.38957	0.39006	0.39033	0.39079	0.39104	0.39158	0.39182	0.39235	0.39267	0.3932
14.3	0.40225	0.4026	0.40305	0.40338	0.4038	0.40408	0.40462	0.40492	0.40541	0.40575	0.40624
14.3	0.35674	0.35709	0.35744	0.35779	0.3582	0.35849	0.35904	0.35928	0.35979	0.36011	0.36055
13.8	0.44596	0.44646	0.44698	0.44744	0.44801	0.44845	0.44916	0.44954	0.45015	0.45063	0.45122
13.8	0.4839	0.48441	0.48499	0.48542	0.486	0.4864	0.48712	0.48753	0.48822	0.48871	0.48934
13.7	0.48142	0.48191	0.48247	0.48291	0.4835	0.48393	0.48461	0.48503	0.48563	0.48613	0.48674
13.7	0.44761	0.44806	0.44859	0.44897	0.44952	0.4499	0.45052	0.45089	0.45146	0.4519	0.45251
13.8	0.46682	0.46726	0.46776	0.46814	0.46865	0.46905	0.46965	0.46999	0.4706	0.47104	0.47163
13.8	0.49728	0.49768	0.49823	0.49858	0.49911	0.49946	0.5001	0.50049	0.50106	0.50151	0.50213
13.7	0.44206	0.44249	0.44296	0.44336	0.44386	0.44423	0.44485	0.44522	0.44581	0.44625	0.44683
13.7	0.47218	0.47269	0.47327	0.47369	0.47425	0.4747	0.47539	0.47577	0.47641	0.47688	0.47754
13.8	0.40081	0.40123	0.40167	0.40198	0.40262	0.40315	0.40384	0.40446	0.4049	0.4056	0.40589
13.8	0.39647	0.39689	0.39741	0.3977	0.3983	0.39884	0.39951	0.40014	0.40056	0.40127	0.40154
12.2	0.49484	0.49526	0.49565	0.49608	0.49656	0.49737	0.49807	0.49894	0.49976	0.50073	0.5017
12.2	0.51092	0.51137	0.51197	0.51237	0.51302	0.51363	0.51444	0.5152	0.51572	0.51649	0.51687



```
from sklearn import linear_model

regr = linear_model.LinearRegression()

train_x = np.asanyarray(train[['ENGINE SIZE']])
train_y = np.asanyarray(train[['CO2 EMISSIONS']])

regr.fit (train_x, train_y)

# The coefficients

print ('Coefficients: ', regr.coef_)
print ('Intercept: ', regr.intercept_)
```

Multiple linear regression

- Multiple regression is an extension of simple linear regression. It is used when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots \beta_n X_n + \epsilon$$

The diagram illustrates the components of the multiple linear regression equation. The equation is shown as $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots \beta_n X_n + \epsilon$. Below the equation, four labels in yellow boxes are connected to their respective terms by lines: 'Dependent Variable' points to Y , 'Coefficients' points to β_0 and β_1 , 'Explanatory Variables' points to X_1 and X_2 , and 'Random Error Term/Residuals' points to ϵ . The coefficients β_1 and β_2 are circled in blue, while the explanatory variables X_1 and X_2 are circled in red.

Simple linear regression

- Predict CO2 emission vs Engine size of all cars
 - Independent variable(x) : Engine size
 - Dependent variable(y):CO2 emission

Multiple linear regression

- Predict CO2 emission vs Engine size and cylinders of all car
 - Independent variable(x) : engine size,cylinders
 - Dependent variable(y):CO2 emission

	ENGINE SIZE	CYLINDERS	FUEL CONSUMPTION_COMB	CO2 EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	?

```
from sklearn import linear_model

regr = linear_model.LinearRegression()

train_x = np.asanyarray(train[['ENGINE SIZE','CYLINDERS']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])

regr.fit (train_x, train_y)

# The coefficients

print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

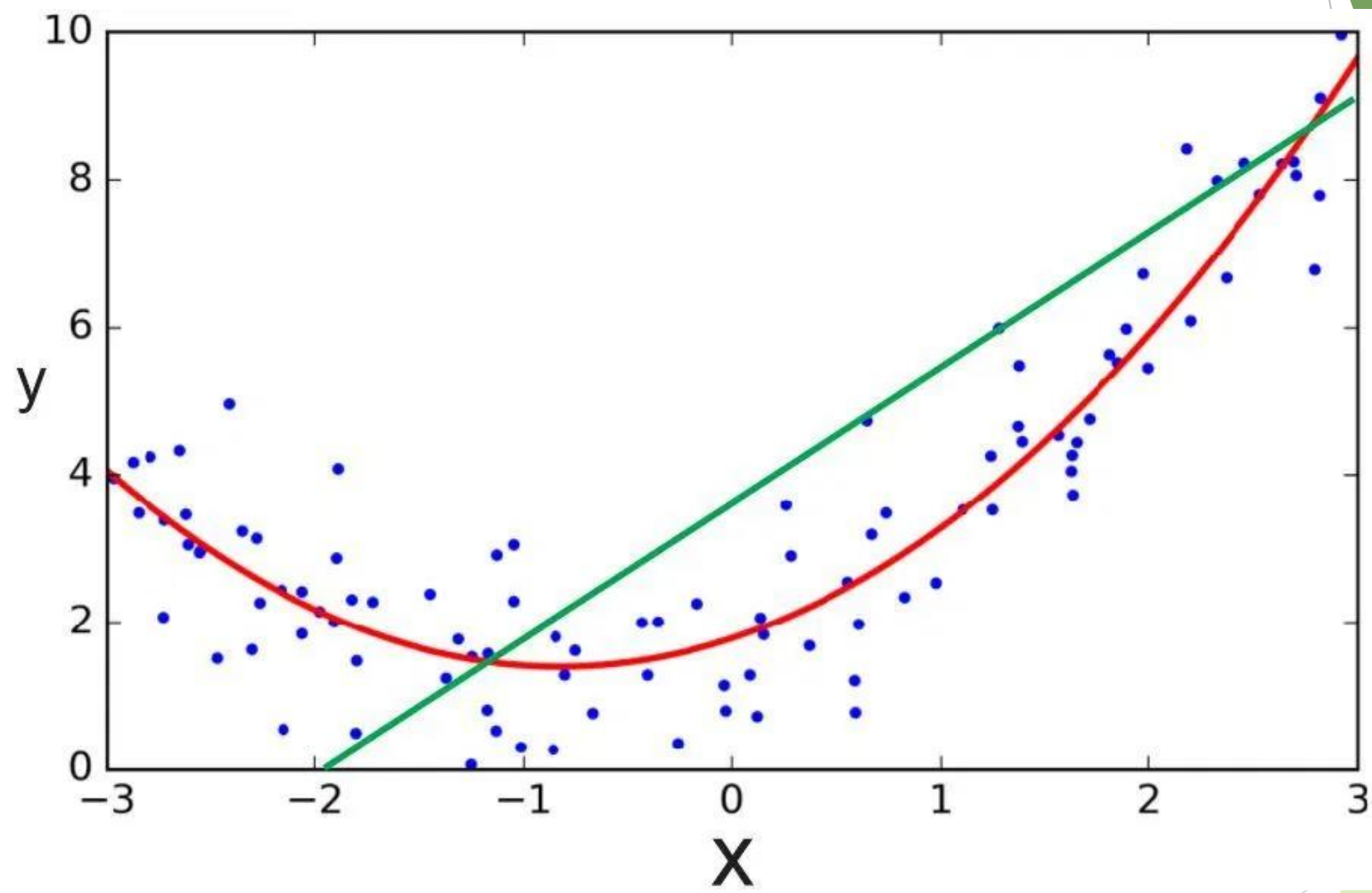

Polynomial regression

- Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an n th degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y , denoted $E(y | x)$

$$y_i = \beta_0 + \beta_1 \tilde{X}_i + \beta_2 Z_i + \beta_3 \tilde{X}_i Z_i + \beta_4 \tilde{X}_i^2 + \beta_5 \tilde{X}_i^2 Z_i + e_i$$

where:

- y_i = outcome score for the i th unit
- β_0 = coefficient for the *intercept*
- β_1 = linear pretest coefficient
- β_2 = mean difference for treatment
- β_3 = linear interaction
- β_4 = quadratic pretest coefficient
- β_5 = quadratic interaction
- \tilde{X}_i = transformed pretest
- Z_i = dummy variable for treatment (0 = control, 1 = treatment)
- e_i = residual for the i th unit



```
from sklearn.preprocessing import PolynomialFeatures

from sklearn import linear_model

train_x = np.asanyarray(train[['ENGINE SIZE','CYLINDERS']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])


test_x = np.asanyarray(test[['ENGINE SIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly.shape
```

Evaluation Metrics For Regression

Statistic	Units	Equation
Standard Error of Prediction (SEP)	Same as reference values	$SEP = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i - bias)^2}{n-1}}$
Root mean square of the error of prediction (RMSEP)	Same as reference values	$RMSEP = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}}$
Bias (d)	Same as reference values	$d = \frac{\sum_{i=1}^n (\hat{y}_i - y_i)}{n}$

The square root of mean standard error of prediction (RMSEP) is related to SEP and Bias according to following equation.

$$RMSEP^2 = SEP^2 + Bias^2$$

Since the RMSEP accounts for bias and provides information regarding calibration accuracy, it can be reported alone, especially when bias is small (then $RMSEP \sim SEP$).

Other Evaluation Metrics For Regression

Statistic	Units	Equation
Coefficient of Determination (r^2)	Unitless	$r^2 = \frac{\left(\sum_{i=1}^n \hat{y}_i y_i - \sum_{i=1}^n \hat{y}_i \sum_{i=1}^n y_i / n \right)^2}{\left(\sum_{i=1}^n \hat{y}_i^2 - \left(\sum_{i=1}^n \hat{y}_i \right)^2 / n \right) \left(\sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n y_i \right)^2 / n \right)}$
Ratio of performance of deviation (RPD)	Unitless	$RPD = \frac{Sd_y}{SEP}$

The coefficient of determination (R^2), which provides an estimation of how much variance between reference and predicted values is explained versus the total variance. Since it is highly dependent on the reference value range, (often ignored), seems to be one of the erroneously preferred guides for validation assessment.

Ratio of performance of deviation or relative predictive determinant (RPD) is dimensionless. It is related with the ability of the model to predict future data in relation to the initial variability of the calibration data. Basically, if a calibration leads to a low SEP but the calibration was carried out with a small range of reference values (standard deviation of reference values almost the same as SEP), the model would only be predicting the data average.

CLASSIFICATION

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.

In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc.**

Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc.

Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output. In classification algorithm, a discrete output function(y) is mapped to input variable(x).

Confusion matrix

What Is a Confusion Matrix?

- It is a diagram for indicating **actual (or target) and predicted (or output) values**, of a model.
- The confusion matrix is a $N \times N$ matrix, where N is the number of classes or outputs.
For 2 classes, we get a 2×2 confusion matrix.
For 3 classes, we get a 3×3 confusion matrix.

There are 4 terms you must understand in order to correctly interpret or read a Confusion Matrix:

- True Positive(TP)
- False Positive(FP)
- True Negative(TN)
- False Negative(FN)

Confusion matrix of 2-class problem

How to form the matrix out of the available **actual (or target)**
and **predicted (or output) values?**

Predicted Class	Actual Class
No Cancer	No Cancer
Has Cancer	Has Cancer
No Cancer	No Cancer
No Cancer	No Cancer
No Cancer	Has Cancer
Has Cancer	Has Cancer
No Cancer	No Cancer
Has Cancer	No Cancer
No Cancer	No Cancer
No Cancer	No Cancer
Has Cancer	No Cancer
No Cancer	No Cancer



		Actual Class	
		Has Cancer	No Cancer
Predicted Class	Has Cancer	2	2
	No Cancer	1	7

Q) From the given confusion matrix,

write down the explanation i.e. interpretation of the presence at the indicated cell, for '34'.

		Confusion Matrix		
Target Class	Covid	320	29	0.917
	Non-Covid	34	363	0.914
		0.904	0.926	0.916
		Covid	Non-Covid	
		Output Class		

Confusion matrix of 3-class problem

	Actual Class 1	Actual Class 2	Actual Class 3
Predicted Class 1	True Positive	False Positive	False Positive
Predicted Class 2	False Negative	True Positive	False Positive
Predicted Class 3	False Negative	False Negative	True Positive

How to Calculate FN, FP, TN, and TP Values, for multi-class model?

FN: The False-negative value for a class will be the sum of values of corresponding rows except for the TP value.

FP: The False-positive value for a class will be the sum of values of the corresponding column except for the TP value.

TN: The True-negative value for a class will be the sum of the values of all columns and rows except the values of that class that we are calculating the values for.

TP: the True-positive value is where the actual value and predicted value are the same.

The confusion matrix for the IRIS dataset is as shown:

		Predicted Values		
		Setosa	Versicolor	Virginica
Actual Values	Setosa	16 (cell 1)	0 (cell 2)	0 (cell 3)
	Versicolor	0 (cell 4)	17 (cell 5)	1 (cell 6)
	Virginica	0 (cell 7)	0 (cell 8)	11 (cell 9)

Let us calculate the TP, TN, FP, and FN values for the class Setosa using the Above tricks:

TP: The actual value and predicted value should be the same. So concerning Setosa class, the value of cell 1 is the TP value=16.

FN: The sum of values of corresponding rows except for the TP value

$$\text{FN} = (\text{cell 2} + \text{cell 3}) = (0 + 0) = 0$$

FP: The sum of values of the corresponding column except for the TP value.

$$\text{FP} = (\text{cell 4} + \text{cell 7}) = (0 + 0) = 0$$

TN: The sum of values of all columns and rows except the values of that class that we are calculating the values for.

$$\text{TN} = (\text{cell 5} + \text{cell 6} + \text{cell 8} + \text{cell 9}) = 17 + 1 + 0 + 11 = 29.$$

Similarly, for the Versicolor class, the values/metrics are calculated as below:

TP: 17 (cell 5)

FN : $0 + 1 = 1$ (cell 4 + cell 6)

FP : $0 + 0 = 0$ (cell 2 + cell 8)

TN : $16 + 0 + 0 + 11 = 27$ (cell 1 + cell 3 + cell 7 + cell 9).

Q) Calculate for the Virginica class.

Evaluation metrics for classification

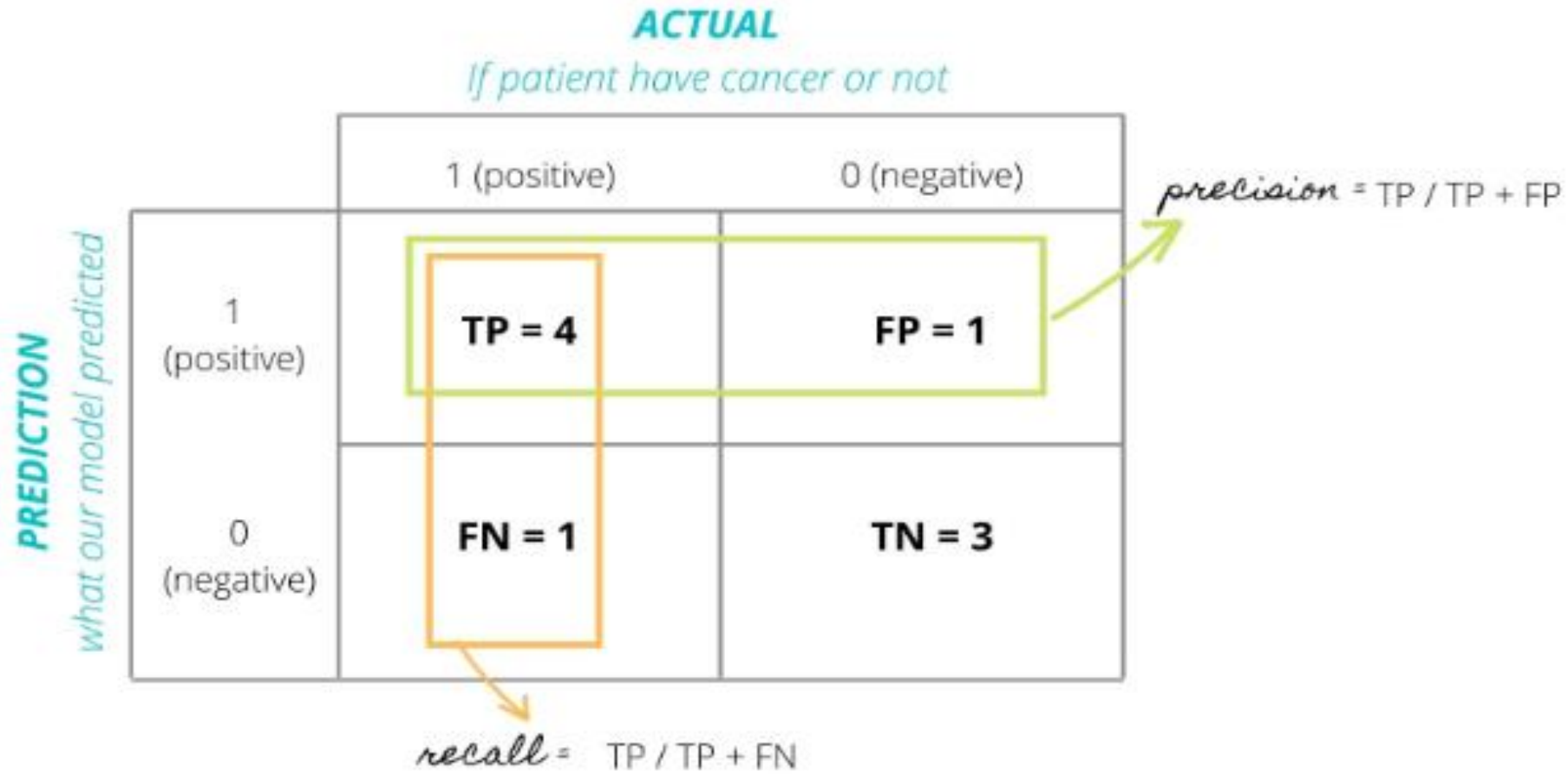
ACTUAL
If patient have cancer or not

PREDICTION
what our model predicted

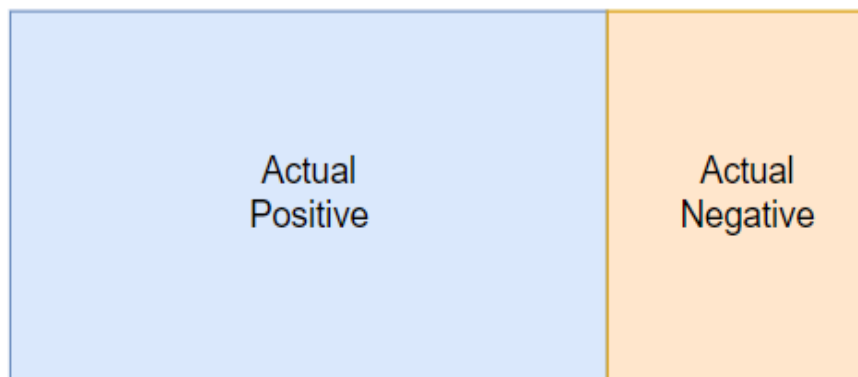
	1 (positive)	0 (negative)
1 (positive)	TP = 4	FP = 1
0 (negative)	FN = 1	TN = 3

$\text{precision} = \text{TP} / \text{TP} + \text{FP}$

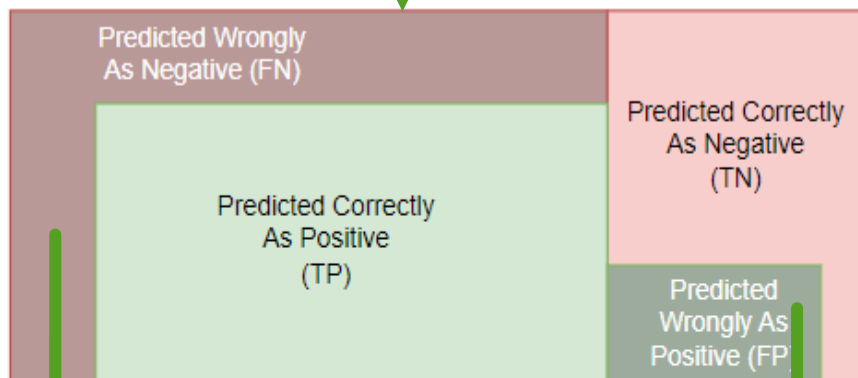
$\text{recall} = \text{TP} / \text{TP} + \text{FN}$



The diagram shows a confusion matrix for a binary classification task. The columns represent the actual status (1 for positive, 0 for negative) and the rows represent the predicted status (1 for positive, 0 for negative). The matrix contains four cells: TP=4 (True Positive), FP=1 (False Positive), FN=1 (False Negative), and TN=3 (True Negative). A green box highlights the TP and FP cells, with an arrow pointing to the precision formula. An orange box highlights the TP and FN cells, with an arrow pointing to the recall formula.



Prediction or
Classification



Minimize this to
improve Recall

Minimize this to
improve Precision
and Specificity

$$\text{TPR (Recall or Sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{Actual Positive}}$$

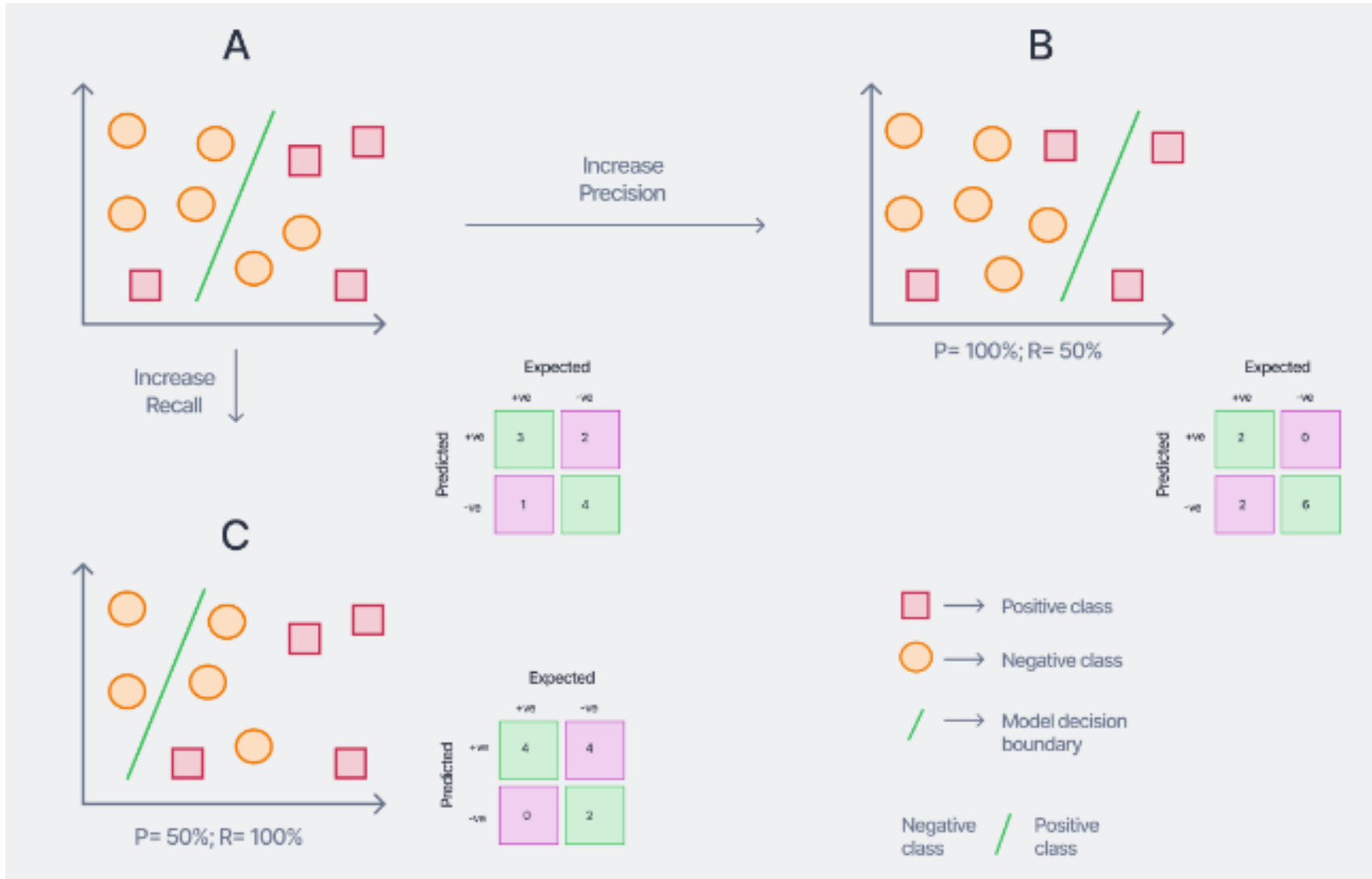
$$\text{TNR (Specificity)} = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{\text{TN}}{\text{Actual Negative}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

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

$$\text{Mis-classification Rate} = \frac{\text{FN} + \text{FP}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

Following diagram illustrates that, when we increase the precision of our model, the recall goes down, and vice-versa.



F1-score is a harmonic mean or weighted average of precision and recall.

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

In both precision and recall, there is false positive and false negative, gives a combined idea about these two metrics.

It is maximum when Precision is equal to Recall.

Disadvantage: The interpretability of the F1-score is poor. This means that we don't know what our classifier is maximizing – precision or recall. So, we use it in combination with other evaluation metrics which gives us a complete picture of the result.

Advantage : F1 score is more useful than accuracy, especially when you have an uneven class distribution.

Confusion Matrix Using Scikit-learn in Python

To put the theory in practice, we can use [Scikit-learn \(sklearn\) library](#) in Python. It has two great functions: **confusion_matrix()** and **classification_report()**.

- Sklearn [confusion_matrix\(\)](#) returns the values of the Confusion matrix. It takes the rows as Actual values and the columns as Predicted values. The rest of the concept remains the same.
- Sklearn [classification_report\(\)](#) outputs precision, recall, and f1-score for each target class.

In addition to this, it also has some extra values: **micro avg**, **macro avg**, and **weighted avg**

Micro average is the precision/recall/f1-score calculated for all the classes.

Macro average is the average of precision/recall/f1-score.

Weighted average is just the weighted average of precision/recall/f1-score.

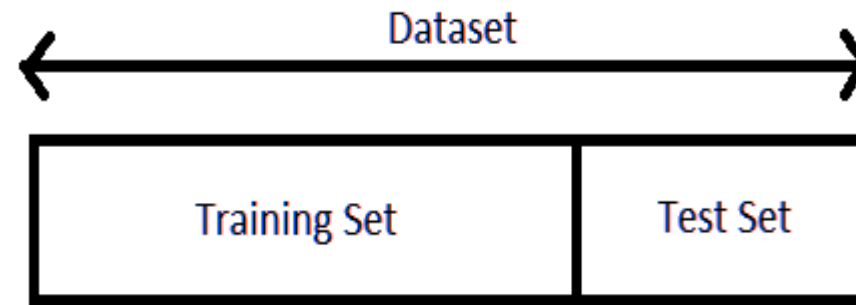
VALIDATION

Validation is a technique in which we train our model using the subset of the data-set (train) and then evaluate using the complementary subset of the data-set(test).

The three steps involved in cross-validation are as follows :

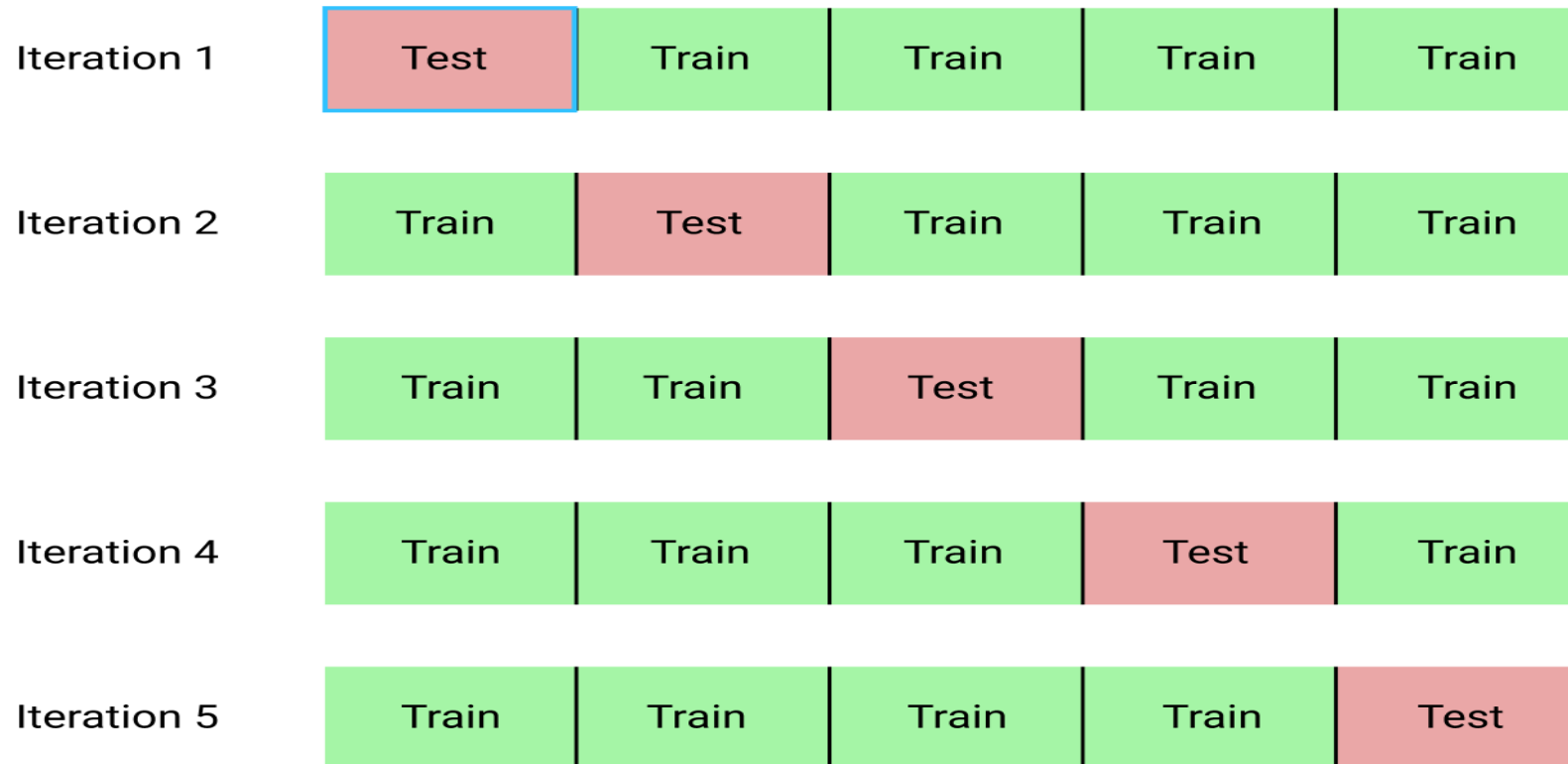
1. Split data set into training and test set
2. Using the training set train the model.
3. Test the model using the test set

USE: To get good out of sample accuracy



Even though we use cross validation technique we get variation in accuracy when we train our model for that we use K-fold cross validation technique

In **K-fold cross validation**, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.



Code in python for k-cross validation:

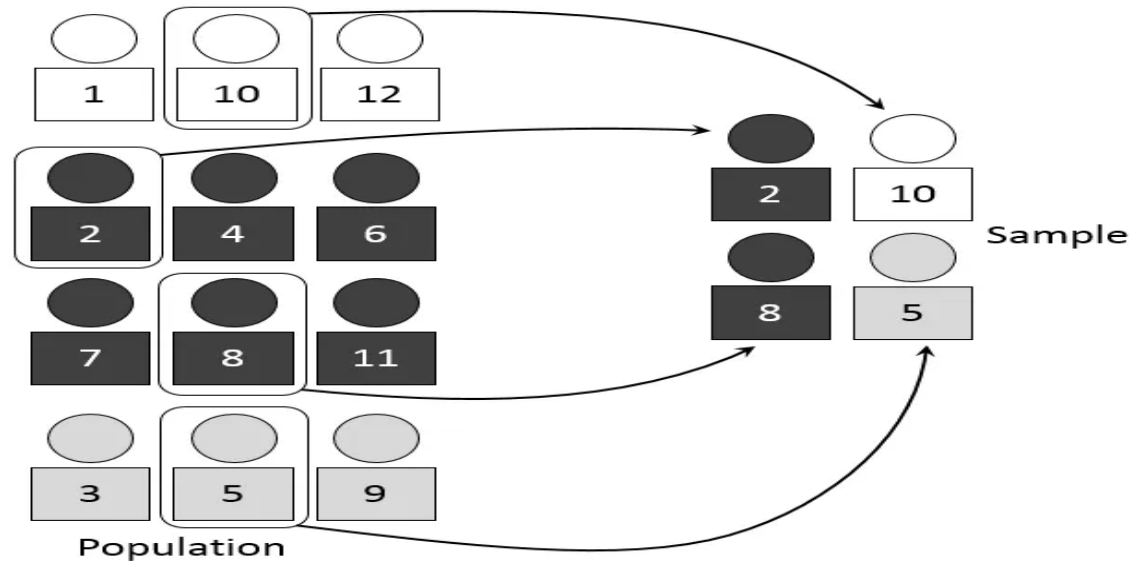
```
from sklearn.model_selection import cross_val_score  
List=cross_val_score(estimator=#name of your model object ,X=#your  
trained input,y=#correct output,cv=#number of folds(k))
```

- First line is importing k fold cross validation function from model_selection sublibrary from sklearn library
- second line will give a list of accuracies based on k value. we need to average them to get the model accurate accuracy

STRATIFIED CROSS-VALIDATION

- 1) **Stratified Sampling** : Before understanding the stratified cross-validation, it is important to know about stratified sampling. Stratified sampling is a sampling technique where the samples are selected in the same proportion (by dividing the population into groups called 'strata' based on a characteristic) as they appear in the population.

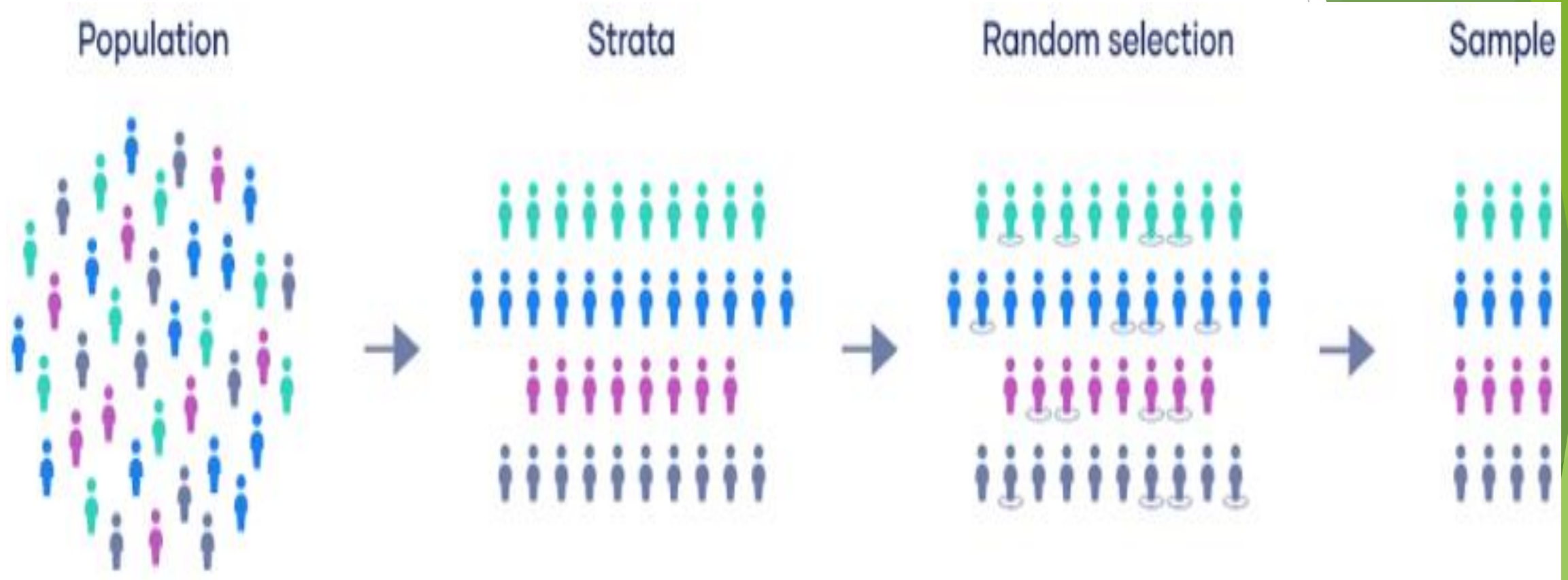
For example, if the population of interest has 30% male and 70% female subjects, then we divide the population into two ('male' and 'female') groups and choose 30% of the sample from the 'male' group and '70%' of the sample from the 'female' group.



Stratified sampling (Image by [Mathprofdk \(Dan Kernler\)](#) on Wikipedia)

- 2) **Cross-validation implemented using stratified sampling** ensures that the proportion of the feature of interest is the same across the original data, training set and the test set. This ensures **that no value is over/under-represented in the training and test sets**, which gives a more accurate estimate of performance/error.

STRATIFIED CROSS-VALIDATION Using Python



3) In k-fold cross-validation, Stratified sampling can be implemented using the '**StratifiedKFold**' class of Scikit-Learn.

4) In hold-out cross-validation, Stratified sampling can be achieved by setting the '**stratify**' argument of 'train_test_split' to the target variable.

HYPERPARAMETERS

Hyperparameters are the parameters that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Examples:

1. The k in k -nearest neighbours.

Choosing the optimal values for hyperparameters

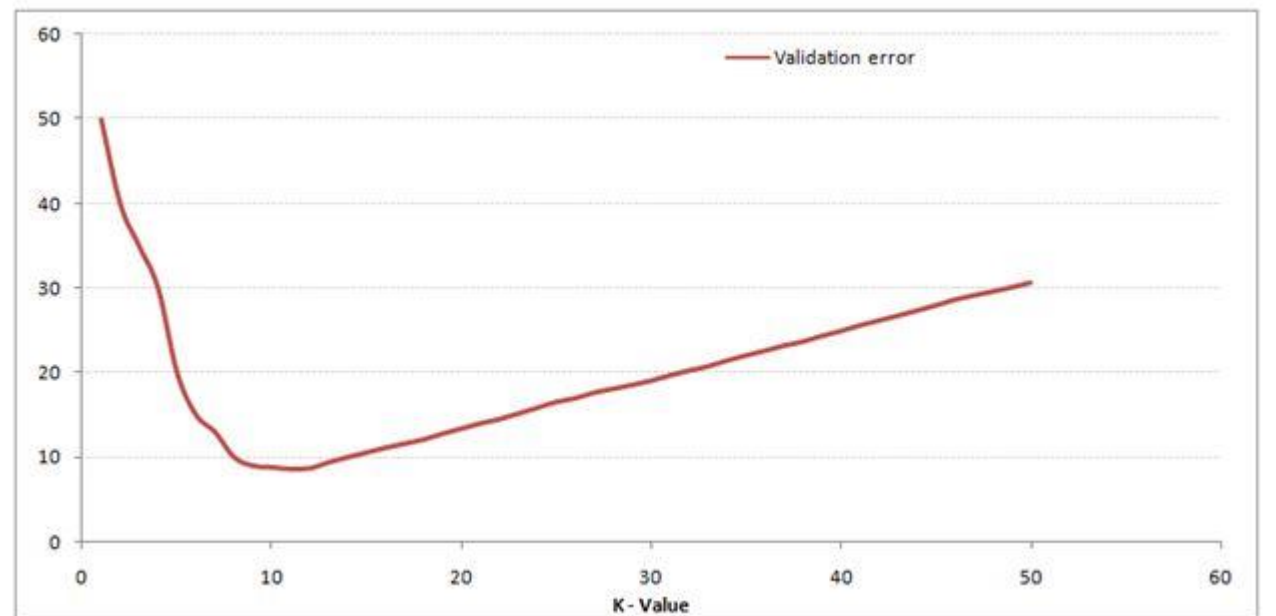
Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. One of the best strategies for Hyperparameter tuning is `grid_search`.

GridSearchCV

In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, it searches for best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set hyperparameter K-nearest neighbours model, with different set of values. The gridsearch technique will check model with all possible combinations of hyperparameters, and will return the best one.

From graph we can say best value for k is 10 and grid search will search all the values of k that we given in range and return the best one



Code in python for getting optimal hyperparameter

using gridsearchCV

for support vector machine (the algorithm)

#importing svm from svc library

```
from sklearn.svm import SVC
```

```
Classifier=SVC()
```

#To import gridsearchcv class from sklearn library

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

#creating a list of dictionaries that need to be inputted for grid search

```
parameters=[{'C':[1,10,100,1000],'kernel':['linear']},{'C':[1,10,100,1000],'kernel':['rbf'],'gamma':[0.5,0.1,0.01,0.001]}]
```

#creating grid search object

```
gridsearch=GridSearchCV(estimator=classifier,param_grid=parameters,scoring='accuracy',cv=10,n_jobs=-1)
```

#fitting gridsearch with data set

```
gd=gridsearch.fit(X_train,y_train)
```

#fitting gridsearch with data set

```
gd=gridsearch.fit(X_train,y_train)
```

#best score among all models in grid search

```
bestaccuracy=gd.best_score_
```

#return the parameters of best model

```
best_param=gd.best_params_
```

Particle Swarm Optimization was proposed by Kennedy and Eberhart in 1995.

Introduction: Resembles, a school of fish or a flock of birds that moves in a group “can profit from the experience of all other members”. In other words, while a bird flying and searching randomly for food, for instance, all birds in the flock can share their discovery and help the entire flock get the best hunt.

While we can simulate the movement of a flock of birds, we can also imagine each bird is to help us find the optimal solution in a high-dimensional solution space and the best solution found by the flock is the best solution in the space.

Theoram: Similar to the flock of birds looking for food, we start with a number of random points on the plane (call them **particles**) and let them look for the minimum point in random directions.

Functionality :At each step, every particle should search around the minimum point it ever found as well as around the minimum point found by the entire swarm of particles. After certain iterations, we consider the minimum point of the function as the minimum point ever explored by this swarm of particles.

Displaying a file

```
import pandas as pd
# making data frame
data = pd.read_csv("peach_spectra_brix.csv")
# calling head() method
data
```

	Brix	wl1	wl2	wl3	wl4	wl5	wl6	wl7	wl8	wl9	...	wl1591
0	15.5	-1.032355	-1.030551	-1.027970	-1.024937	-1.021866	-1.019143	-1.016866	-1.014910	-1.012907	...	0.692447
1	16.7	-1.139034	-1.137186	-1.134485	-1.131222	-1.127761	-1.124464	-1.121508	-1.118802	-1.115973	...	0.729328
2	18.1	-1.152821	-1.150937	-1.148288	-1.145165	-1.141951	-1.138977	-1.136366	-1.134011	-1.131516	...	0.736608
3	14.8	-1.087215	-1.085455	-1.082867	-1.079797	-1.076568	-1.073632	-1.071087	-1.068877	-1.066654	...	0.758695
4	15.1	-1.080364	-1.078436	-1.075784	-1.072693	-1.069562	-1.066691	-1.064214	-1.062025	-1.059787	...	0.719793
5	19.3	-0.981329	-0.979609	-0.977085	-0.974006	-0.970839	-0.967915	-0.965434	-0.963319	-0.961245	...	0.701138
6	12.5	-1.116596	-1.114783	-1.112159	-1.108967	-1.105666	-1.102590	-1.099915	-1.097510	-1.095016	...	0.715351
7	17.7	-1.161027	-1.159104	-1.156357	-1.153022	-1.149477	-1.146078	-1.142952	-1.140035	-1.136996	...	0.728290
8	18.5	-1.167002	-1.165128	-1.162439	-1.159234	-1.155926	-1.152886	-1.150260	-1.147956	-1.145572	...	0.740931
9	18.3	-1.135624	-1.133766	-1.131093	-1.127927	-1.124654	-1.121654	-1.119054	-1.116683	-1.114188	...	0.722930
10	18.6	-1.037029	-1.035263	-1.032739	-1.029744	-1.026651	-1.023877	-1.021587	-1.019614	-1.017614	...	0.708228

.head()

.head() is for displaying first 5 rows of few columns

```
import pandas as pd
# making data frame
data = pd.read_csv("peach_spectra_brix.csv")
# calling head() method
data_top = data.head()
# display
data_top
```

	Brix	wl1	wl2	wl3	wl4	wl5	wl6	wl7	
0	15.5	-1.032355	-1.030551	-1.027970	-1.024937	-1.021866	-1.019143	-1.016866	-1.0
1	16.7	-1.139034	-1.137186	-1.134485	-1.131222	-1.127761	-1.124464	-1.121508	-1.1
2	18.1	-1.152821	-1.150937	-1.148288	-1.145165	-1.141951	-1.138977	-1.136366	-1.1
3	14.8	-1.087215	-1.085455	-1.082867	-1.079797	-1.076568	-1.073632	-1.071087	-1.0
4	15.1	-1.080364	-1.078436	-1.075784	-1.072693	-1.069562	-1.066691	-1.064214	-1.0

5 rows × 601 columns

.tail()

.tail() is for displaying last 5 rows of few columns

```
import pandas as pd
# making data frame
data = pd.read_csv("peach_spectra_brix.csv")
# calling head() method
data_top = data.tail()
# display
data_top
```

	Brix	wl1	wl2	wl3	wl4	wl5	wl6	wl7	
45	18.1	-1.133235	-1.131388	-1.128708	-1.125488	-1.122157	-1.119086	-1.116383	-1
46	16.6	-1.074949	-1.073077	-1.070434	-1.067305	-1.064059	-1.061070	-1.058474	-1
47	18.8	-1.075072	-1.073220	-1.070531	-1.067333	-1.064043	-1.061038	-1.058450	-1
48	17.7	-1.135471	-1.133636	-1.131002	-1.127933	-1.124770	-1.121914	-1.119487	-1
49	18.1	-1.036062	-1.034260	-1.031678	-1.028609	-1.025445	-1.022537	-1.020050	-1

5 rows x 601 columns

.columns

```
data.columns
```

```
→ Index(['Brix', 'wl1', 'wl2', 'wl3', 'wl4', 'wl5', 'wl6', 'wl7', 'wl8', 'wl9',  
        ...,  
        'wl591', 'wl592', 'wl593', 'wl594', 'wl595', 'wl596', 'wl597', 'wl598',  
        'wl599', 'wl600'],  
        dtype='object', length=601)
```

Series and arange

```
[ ] import numpy as np  
    s=pd.Series(np.random.randint(11,23,size=12))  
    s
```



	0
0	20
1	14
2	18
3	12
4	17
5	21
6	14
7	21
8	15
9	16
10	17
11	14

dtype: int64

```
[1] w1= np.arange(11,23,2)  
    w1
```



array([11, 13, 15, 17, 19, 21])


```
# Constants
H1="Healthy"
H2="Infected"
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Consider the True values
y_true_S=np.array([H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2])
# Consider the Predicted values
y_pred_S=np.array([H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H1,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2,H2])
# Formation the confusion matrix.
cm_LB = confusion_matrix(y_pred_S,y_true_S)
#Plot the confusion matrix as diagram AND setting the properties of the figure.
sns.heatmap(cm_LB,
annot=True,
fmt='g', annot_kws={'size': 25},
xticklabels=['Healthy','Infected'],
yticklabels=['Healthy','Infected'],
cbar=False)
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.savefig('20-20_mat1.pdf')
plt.savefig('20-20_mat1.png', dpi=300)
# Accuracy, Precision, Recall, F1-score
from sklearn import metrics
print(metrics.classification_report(y true S, y pred S, digits=3))
```




	precision	recall	f1-score	support
Healthy	0.895	0.850	0.872	20
Infected	0.857	0.900	0.878	20
accuracy			0.875	40
macro avg	0.876	0.875	0.875	40
weighted avg	0.876	0.875	0.875	40



✓
1s

```
[1] import pandas as pd
# making data frame
data = pd.read_csv("peach_spectra_brix.csv")
💡 # calling head() method
data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Columns: 601 entries, Brix to wl600
dtypes: float64(601)
memory usage: 234.9 KB
```

✓
0s



```
data.shape 💡
```



```
(50, 601)
```