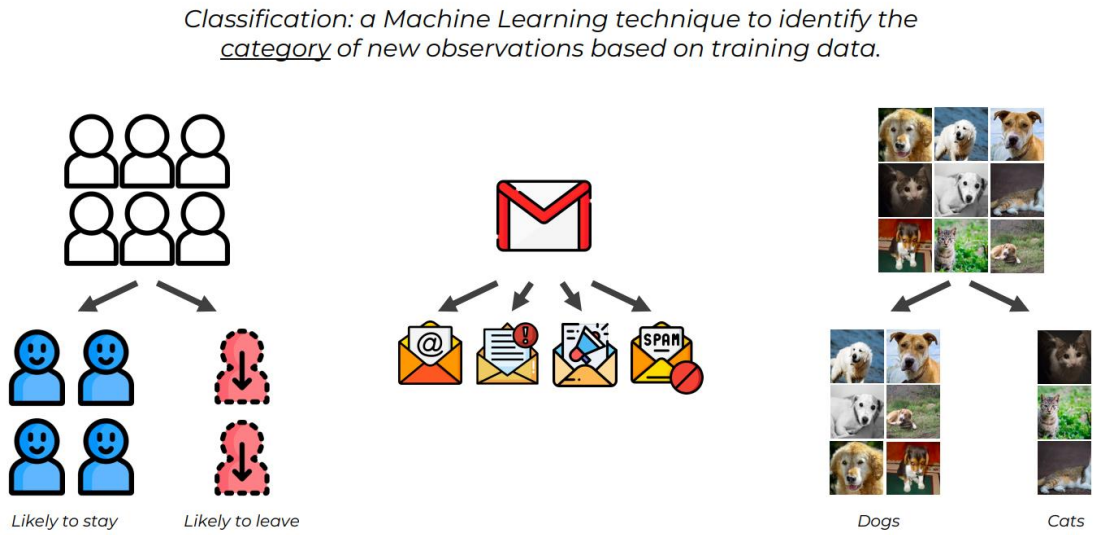
Classification

In machine learning, **classification** is a type of **supervised learning** technique where an algorithm is trained on a labeled dataset to predict the class or category of new, unseen data. The main objective of classification is to build a model that can accurately assign a label or category to a new observation based on its features. Classification algorithms can be broadly classified into **binary** and **multi-class** classifiers. Binary classifiers are used when the classification problem has only two possible outcomes, such as “Yes” or “No”, “Male” or “Female”, “Spam” or “Not Spam”, “Cat” or “Dog”, etc. Multi-class classifiers are used when a classification problem has more than two outcomes, such as classifications of types of crops, types of music, etc.



Unlike regression where you predict a continuous number, you use classification to predict a category. There is a wide variety of classification applications from medicine to marketing. Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

In this part, you will understand and learn how to implement the following Machine Learning Classification models:

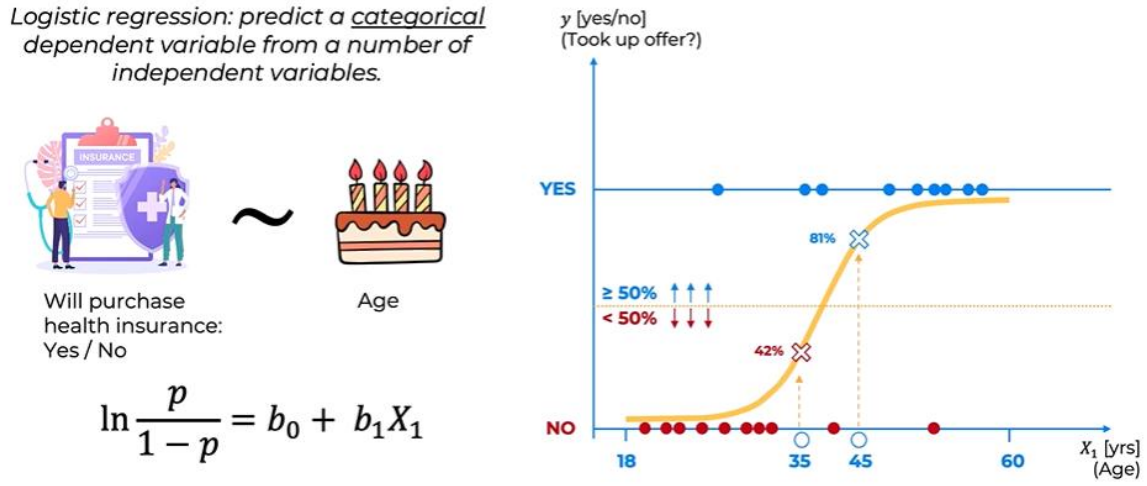
1. Logistic Regression
2. K-Nearest Neighbors (K-NN)
3. Support Vector Machine (SVM)
4. Kernel SVM
5. Naive Bayes
6. Decision Tree Classification
7. Random Forest Classification

Logistic Regression in Machine Learning

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. It is a kind of statistical algorithm, which analyzes the relationship between a set of independent variables and the dependent binary variables. Logistic regression is a powerful tool for decision-making.

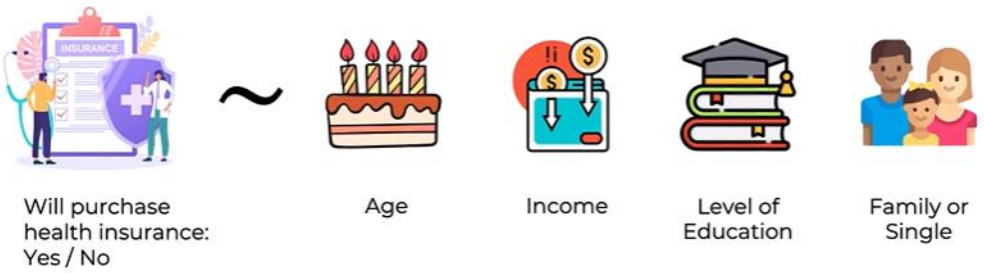
Logistic regression is a supervised machine learning algorithm mainly used for binary classification where we use a logistic function, also known as a sigmoid function that takes input as independent variables and produces a probability value between 0 and 1.

Logistic Regression for Single Independent variable:



Base on the Logistic Regression 35 years age would not purchase insurance but 45 years age would purchase insurance.

Multiple Independents Variable for Logistic Regression:



Logistic regression - Maximum Likelihood Estimation

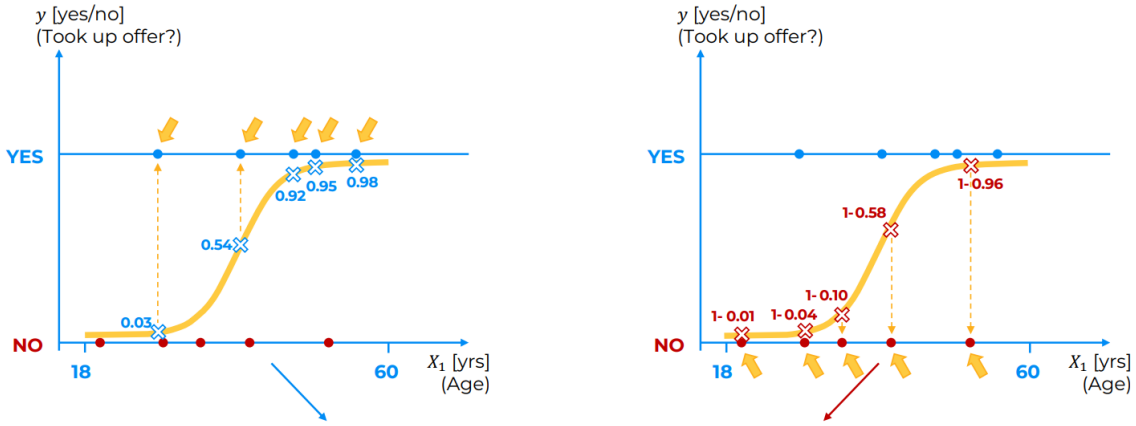
In logistic regression, the maximum likelihood estimation (MLE) is used to estimate the parameters of the model. The goal of MLE is to find the set of parameters that maximize the likelihood function, which is the probability of observing the data given the model parameters. The likelihood function is a product of probabilities of observing each data point, given the model parameters. The parameters are estimated by maximizing the log-likelihood function, which is the natural logarithm of the likelihood function. The maximum likelihood estimates of the parameters are the values that maximize the log-likelihood function.

$$\text{log-likelihood} = \sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

where y_i is the binary response variable (0 or 1) for the i-th observation, and p_i is the predicted probability of the positive class for the i-th observation. The predicted probability is calculated as:

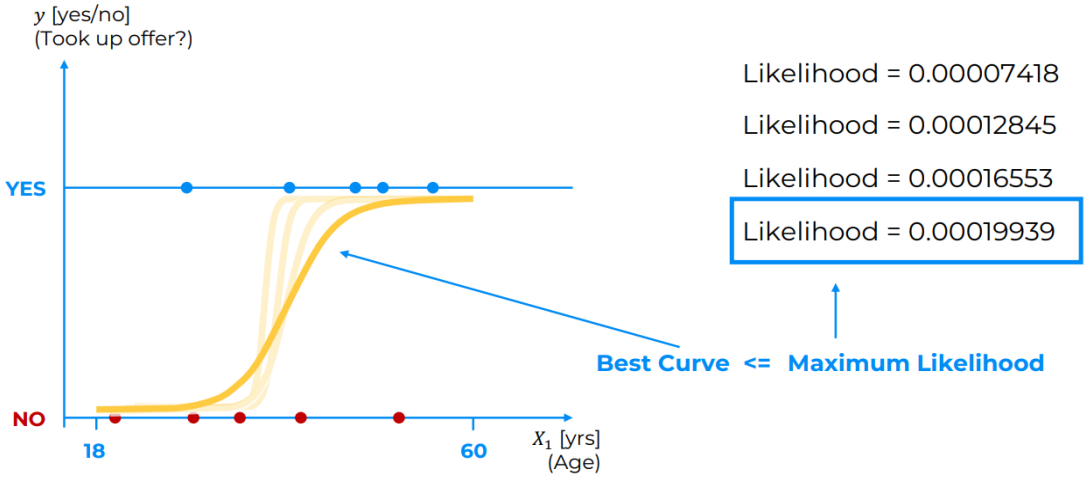
$$p_i = \frac{1}{1 + e^{-z_i}}$$

where z_i is the linear combination of the predictor variables for the i-th observation.



Likelihood = 0.03 x 0.54 x 0.92 x 0.95 x 0.98 x (1 – 0.01) x (1 – 0.04) x (1 – 0.10) x (1 – 0.58) x (1 – 0.96)

Likelihood = **0.00019939**



Dataset: Social_Network_Ads

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0
26	80000	0
26	52000	0
20	86000	0
32	18000	0
18	82000	0
29	80000	0
47	25000	1
45	26000	1
46	28000	1
48	29000	1
45	22000	1
47	49000	1
48	41000	1

On data classification model trained and need to predict which customer buy brand new model SUV. Our model will be trained on different Age, EstimatedSalary (both independent variable) and also purchase (dependent variable). Purchase '0' mean customer didn't buy any previous SUV and '1' mean bought previous SUV. Future prediction purchase '1' high chance to purchase New model SUV. Predicted model will target the customer and Advertise team use the result of predicted model and optimize the target the future customer that's why name of the dataset is Social_Network_Ads.

Python Code:

***Logistic Regression is linear models. You will get all code related to logistic regression from scikit-learn. Go to scikit-learn then API, API reference → classification → Linear Models → all the linear models you will get logistic regression and click on it you will get complete class of Logistic Regression.*

Importing the libraries

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

```
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

** Importing libraries and dataset are the same as data preprocessing.

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)
```

```
print(X_train)
```

** Dataset has 400 customer that's why test_size=0.25 mean 300 for training and 100 for test set

O/P: [39 106000]

```
 [    37  57000]
 [    26  72000]
 [    35  23000]
 [    54 108000]
 [    30  17000]
 [    39 134000]
 [    29  43000]
 [    33  43000]
```

…….

…….

```
print(y_train)
```

O/P:

```
[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0
0 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0
1 0 1 .....
.....
0 0 0 0]
```

```
print(X_test)
```

O/P:

```
 [    27  84000]
```

```
[    35  20000]
[    43 112000]
[    27  58000]
[    37  80000]
[    52  90000]
[    26  30000]
 ……..
 ……
```

```
print(y_test)
```

O/P:

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0
0 0…… 1 1]
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

** All the features (Age and EstimatedSalary) are same scale.

```
print(X_train)
```

O/P:

```
 [ 0.08648817  1.05583366]
 [-0.11157634 -0.3648304 ]
 [-1.20093113  0.07006676]
 [-0.30964085 -1.3505973 ]
 [ 1.57197197  1.11381995]
 [-0.80480212 -1.52455616]
 [ 0.08648817  1.8676417 ]

 ….    ….   …       … ….
```

```
print(X_test)
```

O/P:

```
 [-1.10189888  0.41798449]
 [-0.30964085 -1.43757673]
 [ 0.48261718  1.22979253]
 [-1.10189888 -0.33583725]
 [-0.11157634  0.30201192]
 [ 1.37390747  0.59194336]
 [-1.20093113 -1.14764529]

   ……
```

……..

Training the Logistic Regression model on the Training set

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

** sklearn.linear_model this is the model; LogisticRegression is the class; Here no parameter is required because simple logistic regression so random_state=0; classifier.fit method trained the logistic regression as X_train and y_train

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

** predict method for single observation. Purchase decision 1st customer whose is 30 and estimate salary 87000. Whether this customer bought SUV Yes or No. Our prediction Y test or result is 0 mean customer SUV not bought. So, prediction correct.

O/P: `[0]`

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

O/P:

```
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
```

…….

….

Making the Confusion Matrix

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

**From API of scikit-learn look into model→ metrics→ Classification metrics → `confusion_matrix`; output `confusion_matrix(y_test, y_pred)` in cm; `accuracy_score` function will give the accuracy.

O/P:
[[65 3]
[8 24]]
0.89

** 65 correct prediction for class 0 mean not to buy SUV; 24 correct prediction for class 1 mean to buy SUV; 8 incorrect prediction for class 0 mean not to buy SUV but practically they bought SUV; 3 incorrect prediction for class 1 mean customer to buy SUV but practically they did not buy SUV;

Visualising the Training set results

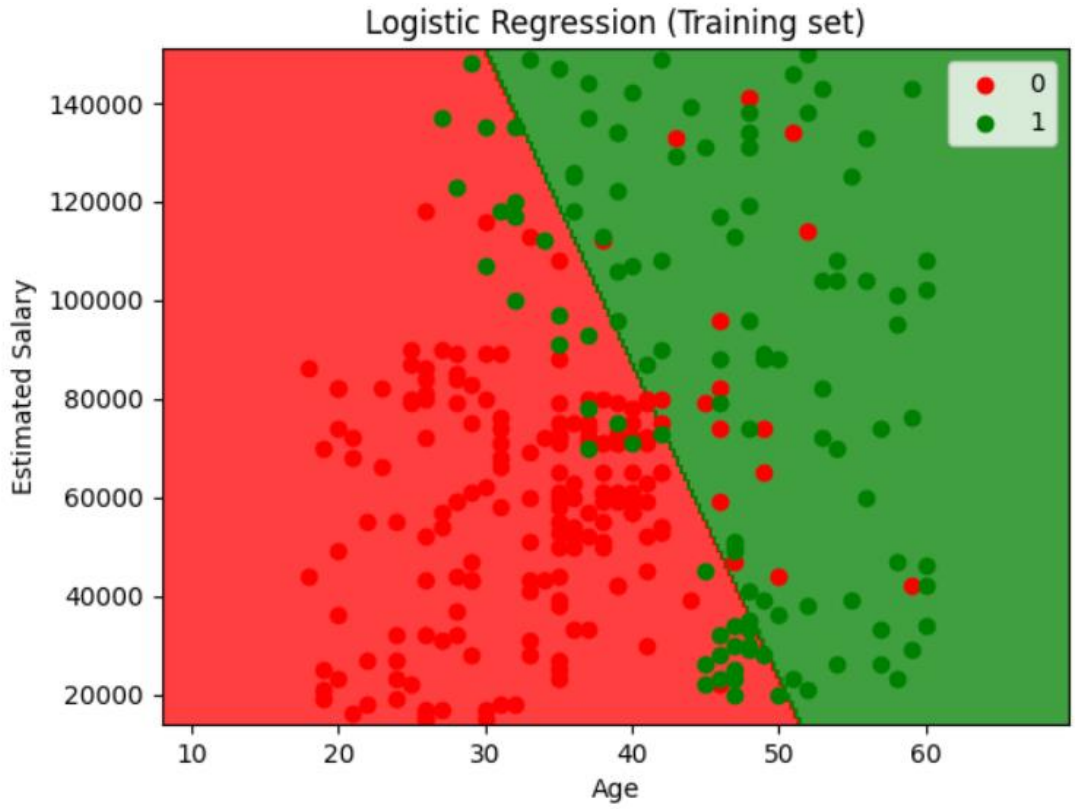
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

**Two-dimension plot. X-axis for Age and Y-axis for Estimated Salary; plot will be prediction region ; Here prediction boundary is linear region(straight line) because Logistic Regression is Linear; np.meshgrid function step=0.25 mean Age: 10,10.25,10.5,... & Estimated Salary 20000, 25000, 30000, ..; Green point customer bought SUV and Red point customer did not buy SUV. Red & Green region are prediction. Red point on green region mean customer did not buy SUV but predicted buy SUV same-way Green point on Red region mean customer bought SUV but predicted bought SUV. So, correct prediction as observation points as same colour region and incorrect prediction as observation points are different colour from belong region. Any linear classification boundary is straight line which separate two regions. These are training set result. All the customer are here in training set therefore observation is trained logistic regression. Logistic regression is able to perform new observation meaning observation of test set mean customer of the test set.

O/P: `<ipython-input-15-3277c112bab0>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x*`

```
& *y*.  Please use the *color* keyword-argument or provide a 2D array
with a single row if you intend to specify the same RGB or RGBA value
for all points.
```

```
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
```



Visualising the Test set results

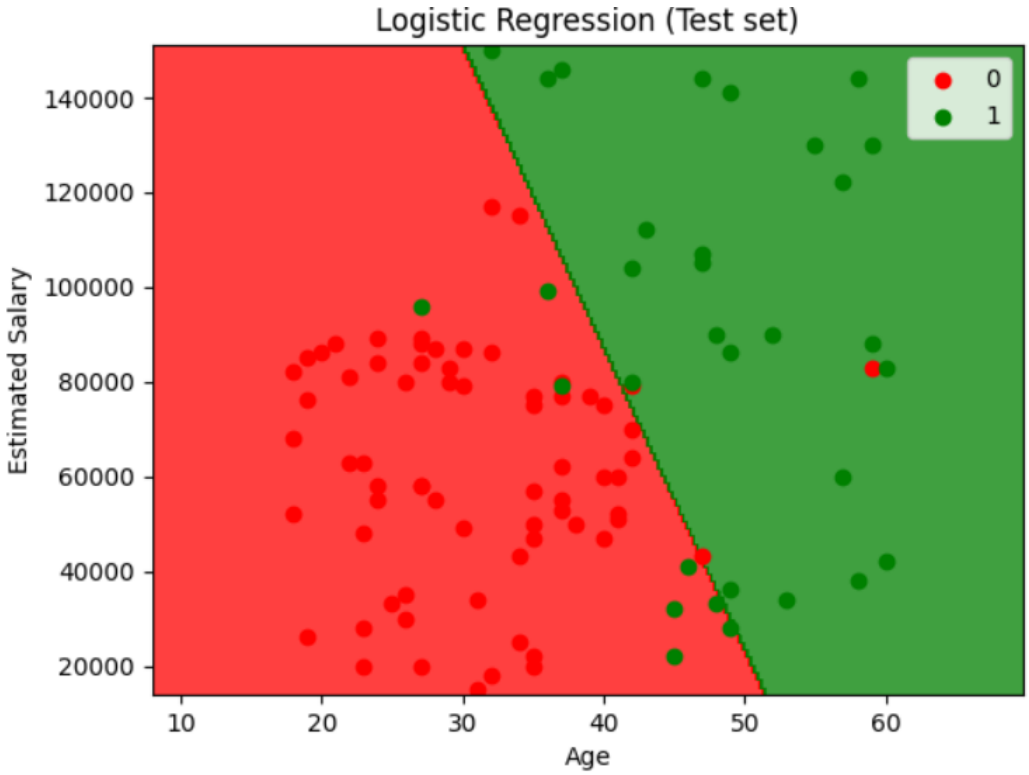
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

** Here is the test set result. Logistic regression model done very good job separate two classes. Some error also here.

O/P: <ipython-input-16-53d83417cfe6>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

```
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
```

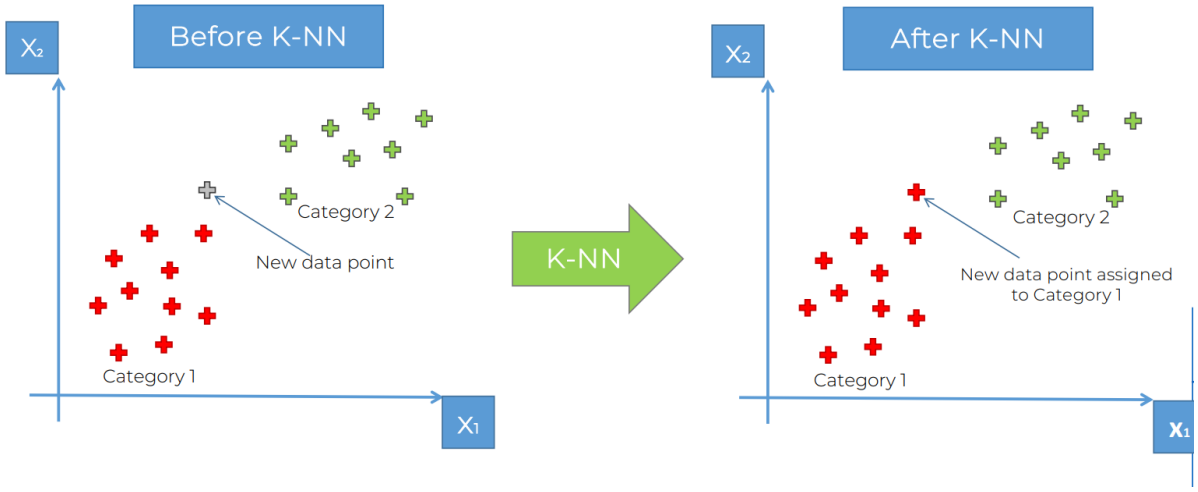


K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm used for both classification and regression tasks.
It is a **supervised learning** algorithm that predicts the label or value of a new data point by considering its K closest neighbors in the training dataset.

The algorithm works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors.
KNN is a versatile and widely used machine learning algorithm that is primarily used for its simplicity and ease of implementation. It does not require any assumptions about the underlying data distribution. It can also handle both numerical and categorical data, making it a flexible choice for various types of datasets in classification and regression tasks.

Here two category datasets Red as category1 and Green as category2. New data point, should it fall on category1 or category2? K-NN algorithm identify the red or green category.



How did it do that ?

STEP 1: Choose the number K of neighbors

STEP 2: Take the K nearest neighbors of the new data point, according to the Euclidean distance

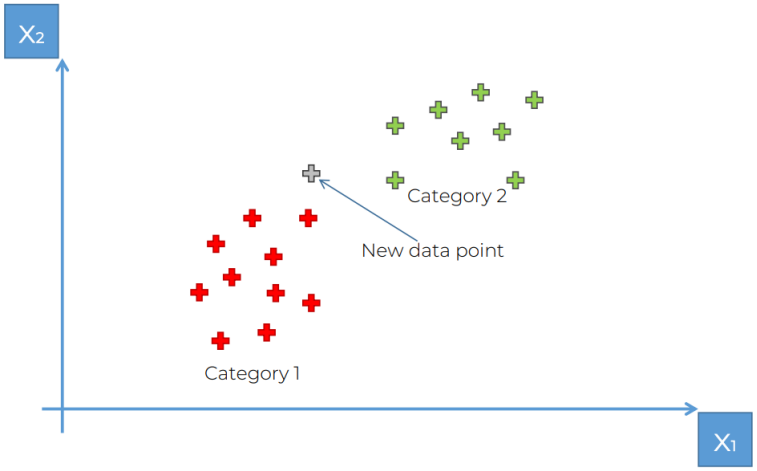
STEP 3: Among these K neighbors, count the number of data points in each category

STEP 4: Assign the new data point to the category where you counted the most neighbors

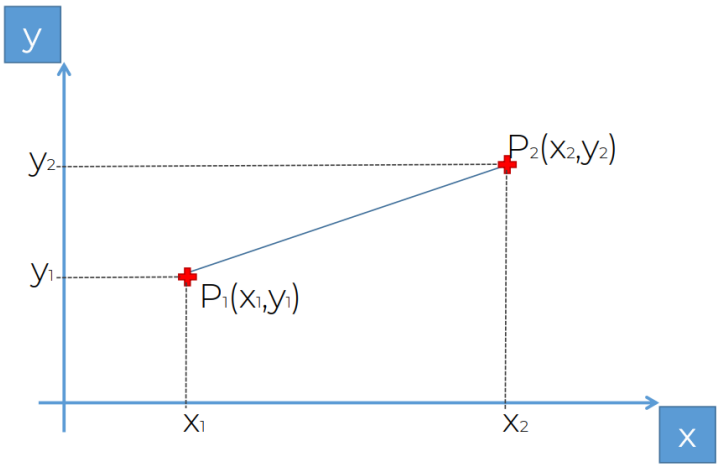
Your Model is Ready

K-NN algorithm

STEP 1: Choose the number K of neighbors: K = 5

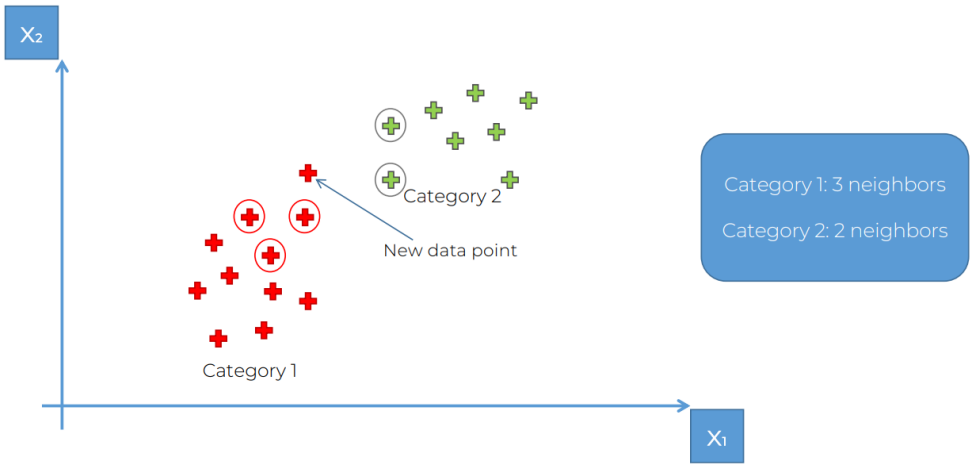


Euclidean Distance



Euclidean Distance between P_1 and $P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

K-NN algorithm



K-Nearest Neighbors (K-NN) Python Code:

Dataset: Social_Network_Ads

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0
26	80000	0
26	52000	0
20	86000	0
32	18000	0
18	82000	0
29	80000	0
47	25000	1
45	26000	1
46	28000	1
48	29000	1
45	22000	1
47	49000	1
48	41000	1

Importing the same Dataset for K-NN. All the codes for Logistic Regression & K-NN are the same except **Training the K-NN model on the Training set.

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].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)
```

```
print(X_train)
```

O/P:

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]

 -----------------

 [    29  43000]
 [    36  52000]
 [    27  54000]
 [    26 118000]]
```

```
print(y_train)
```

O/P: `[0 1 0 1 1 1 0 --------- 0 0 0 0]`

```
print(X_test)
```

O/P:

```
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]

 ---------------

 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]]
```

```
print(y_test)
```

O/P: `[0 0 0 0 0 0 0 ---------- 1 0 1 1 1]`

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print(X_train)
```

O/P:

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]

 -------------------------

 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
 [-1.10189888 -0.45180983]
 [-1.20093113  1.40375139]]
```

```
print(X_test)
```

O/P:

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]

 -------------------------

 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

** Go to Scikit-Learn → API → sklearn.neighbors: Nearest Neighbors (Model)→ neighbors.KNeighborsClassifier where `KNeighborsClassifier` →This is the class.

Instance of the class has the parameters **n_neighbors*int, default=5 (***Number of neighbors to use by default for **`kneighbors`** queries) and metric='minkowski' mean distance between observation point and neighbour which measure classic Euclidean Distance.

P=2 → Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2.

Classifier.fit() → fit method takes X_train and y_train .

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

O/P: `[0]`

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

O/P:

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]

 -----

 [0 1]
 [0 0]
 [1 1]
 [1 1]
 [1 1]]
```

Making the Confusion Matrix

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

O/P:

```
[[64  4]
 [ 3 29]]
0.93
```

Visualising the Training set results

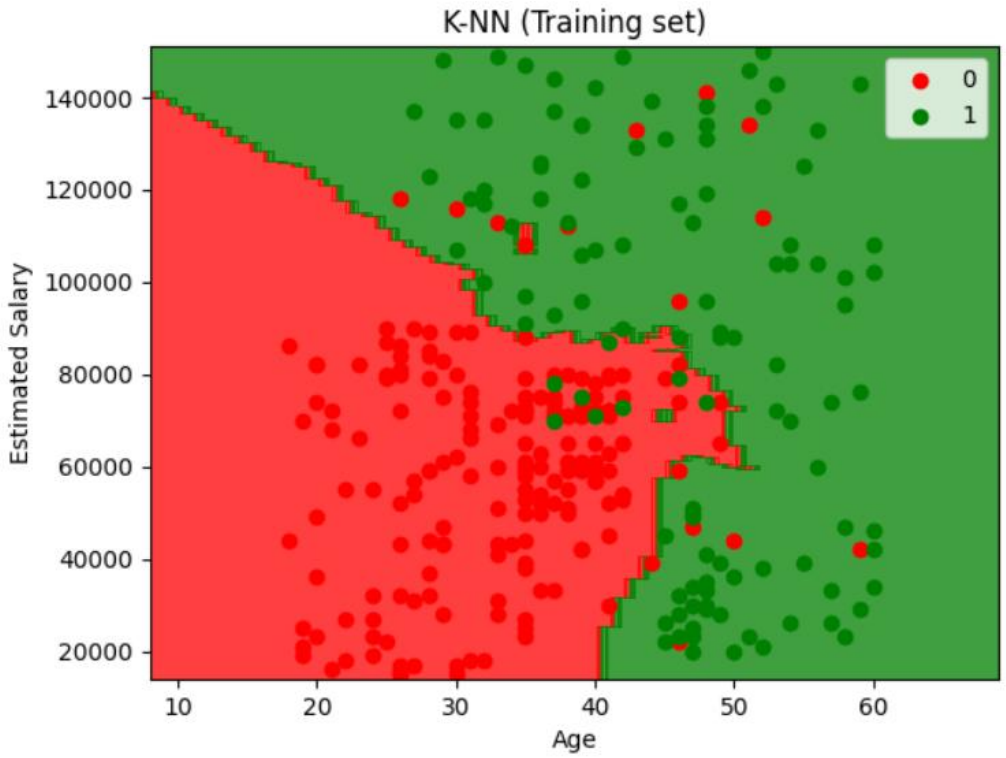
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

** K-NN model is very compute intensive so lot of computation that's why during execution time taking more. Here prediction boundary not a straight line (non linear) some kind of curve that's why prediction error is less. We want to avoid overfeeding for Machine Learning. Here we get excellent result. So, much better than logistic regression.

O/P:

```
<ipython-input-15-9061e2cf8fe3>:10: UserWarning: *c* argument looks
like a single numeric RGB or RGBA sequence, which should be avoided as
value-mapping will have precedence in case its length matches with *x*
& *y*.  Please use the *color* keyword-argument or provide a 2D array
```

```
with a single row if you intend to specify the same RGB or RGBA value
for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
```

```
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

O/P:

```
<ipython-input-16-4086b5de55b9>:10: UserWarning: *c* argument looks
like a single numeric RGB or RGBA sequence, which should be avoided as
value-mapping will have precedence in case its length matches with *x*
& *y*.  Please use the *color* keyword-argument or provide a 2D array
with a single row if you intend to specify the same RGB or RGBA value
for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
```