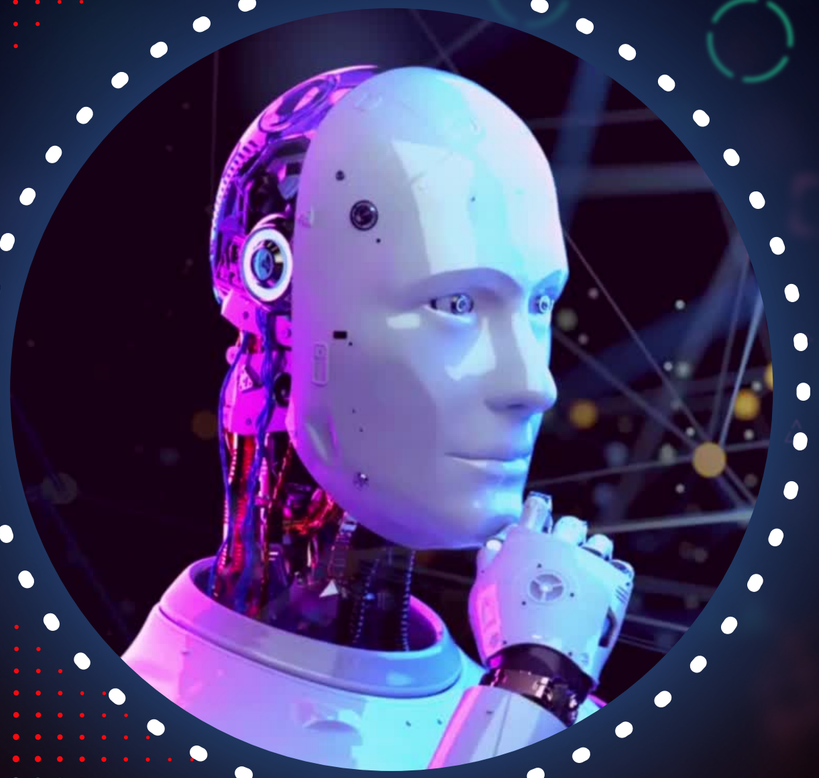
Pantech e Learning
DIGITAL LEARNING SIMPLIFIED



MACHINE LEARNING

DAY – 16

ENSEMBLE ALGORITHMS

1. Bagging - Random Forest
2. Boosting - AdaBoost, XGBoost

ENSEMBLE ALGORITHMS

- Ensemble algorithms are machine learning techniques that combine the predictions of multiple individual models (often called "base models" or "weak learners") to improve the overall performance and accuracy of a predictive model.

- Ensemble methods are powerful because they can reduce overfitting, improve model generalization, and enhance predictive performance compared to single models.

BAGGING ALGORITHMS

- Bagging involves creating multiple bootstrap samples (randomly selected subsets with replacement) from the training data and training a separate base model on each sample.

- The predictions from these models are combined, often by averaging (for regression) or voting (for classification), to produce the final ensemble prediction.

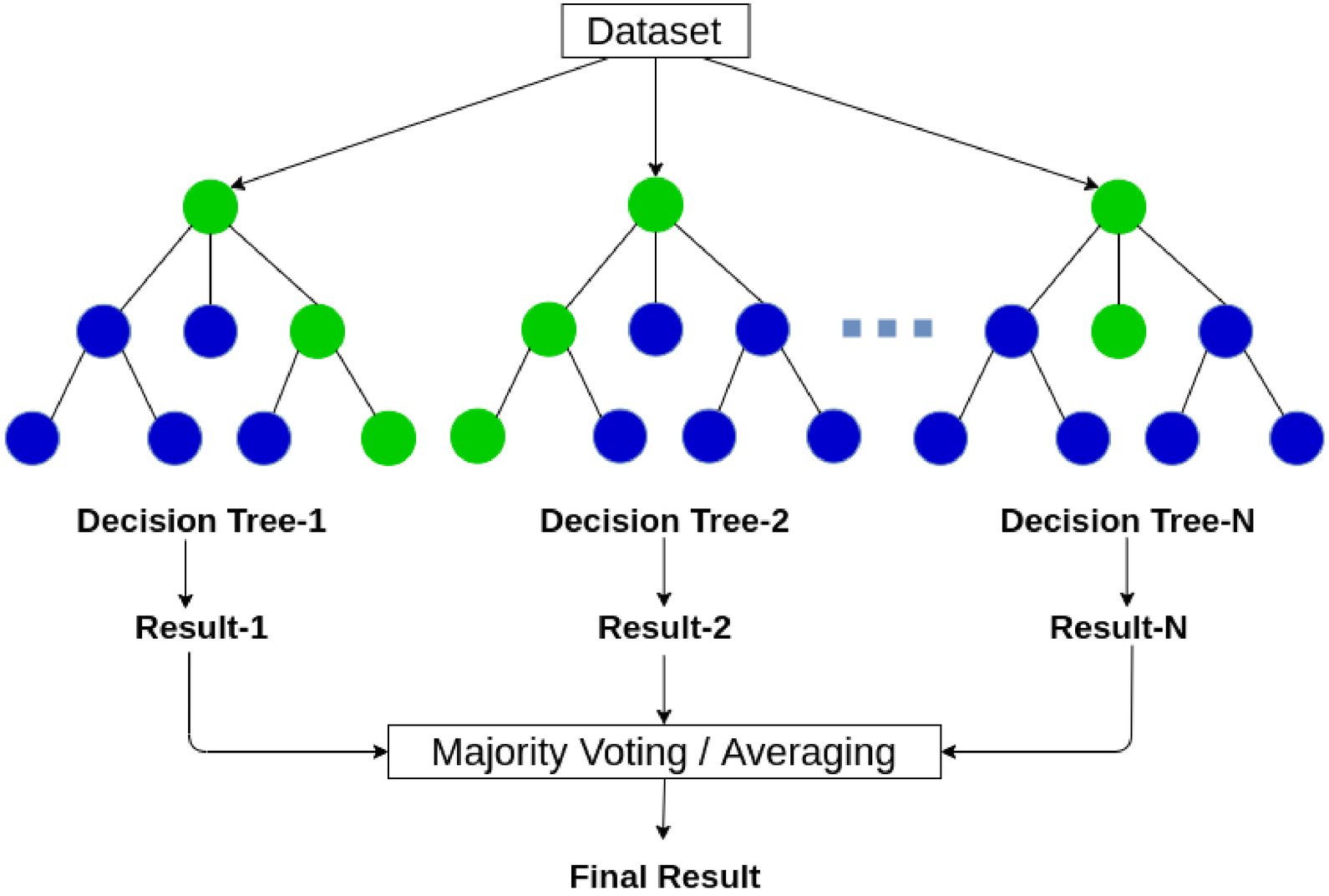
- Random Forests is a popular ensemble algorithm that uses bagging with decision trees as base models.

RANDOM FOREST

RANDOM FOREST ALGORITHM

- Random forest is a popular machine learning algorithm that is used for both classification and regression tasks.
- It is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model.
- The random forest algorithm works by building a large number of decision trees, each using a randomly selected subset of the available input features and a randomly selected subset of the training data.
- These decision trees are then combined to make predictions on new data by taking a vote or averaging the outputs of each individual tree.

RANDOM FOREST ALGORITHM



RANDOM FOREST ALGORITHM

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import RandomForestRegressor
```

BOOSTING **ALGORITHMS**

- Boosting focuses on improving the performance of weak learners by giving more weight to instances that are difficult to classify correctly.

- In boosting, base models are trained sequentially, and each subsequent model pays more attention to the instances that were misclassified by the previous models.

- Common boosting algorithms include AdaBoost, Gradient Boosting (e.g., XGBoost, LightGBM, and CatBoost), and Adaptive Boosting (AdaBoost).

ADABOOST

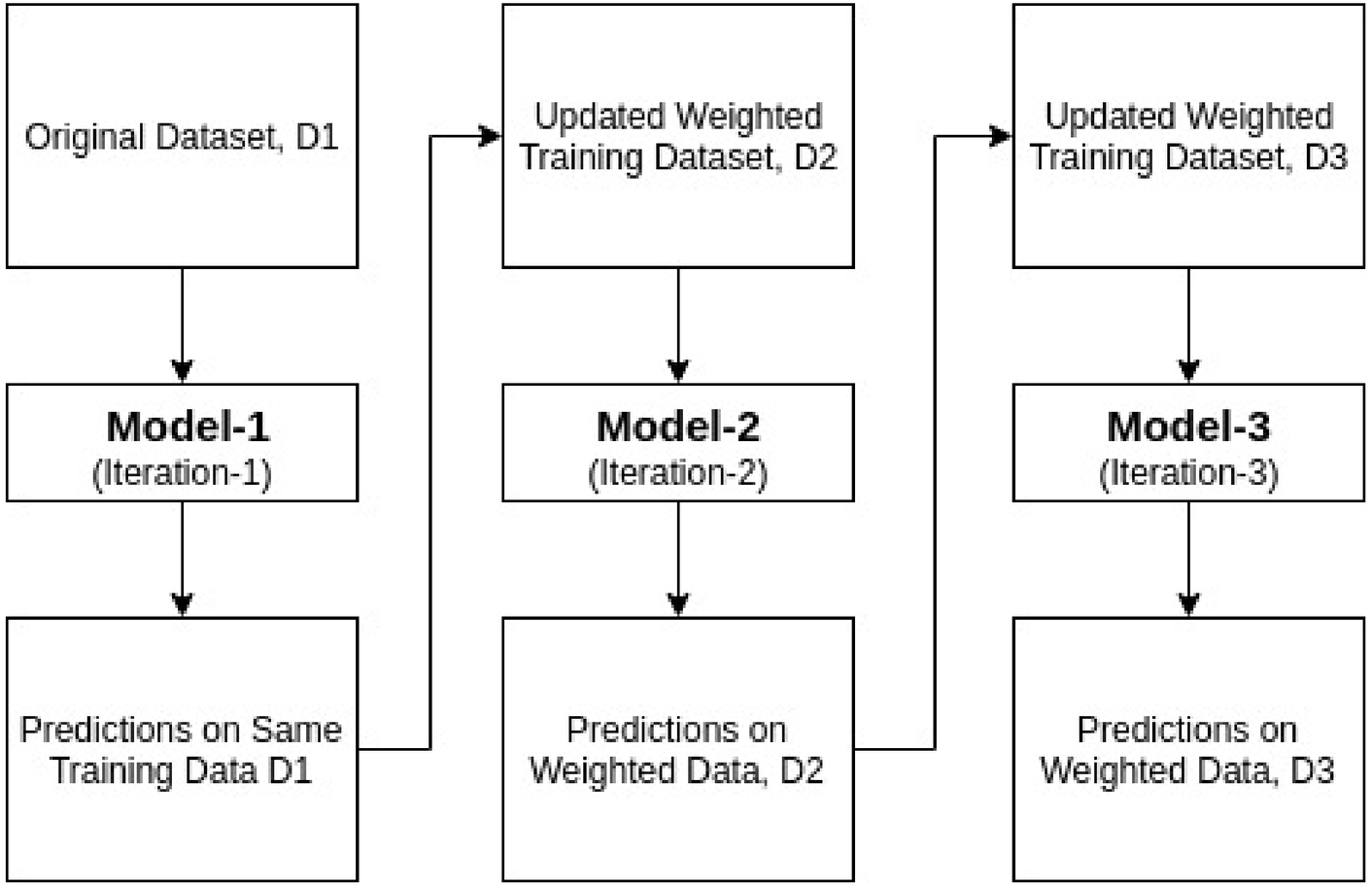
ADABOOST

- AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique used as an Ensemble Method in Machine Learning.

- It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances.

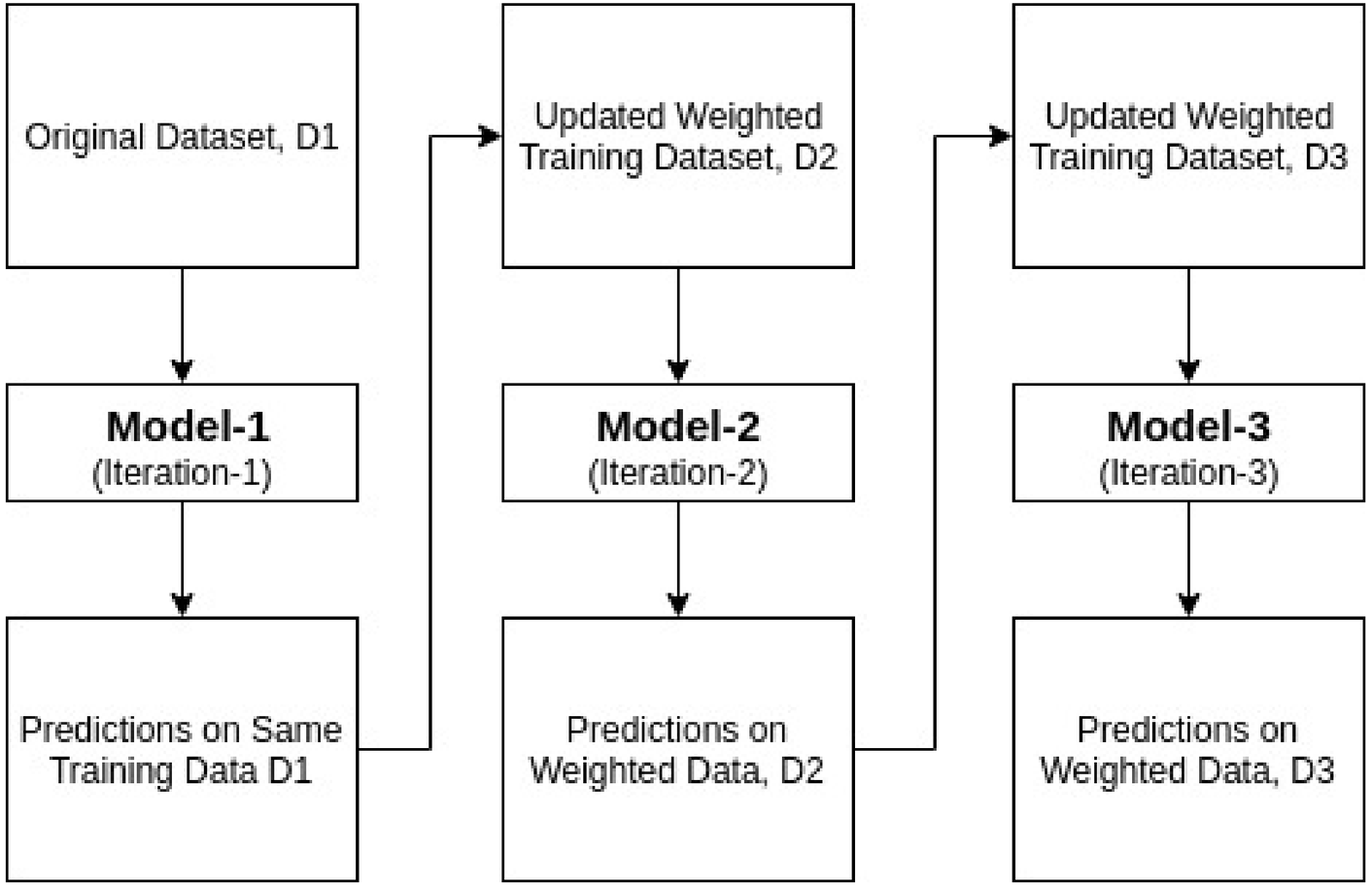
- The basic idea behind AdaBoost is to iteratively train weak classifiers on different subsets of the training data, and assign higher weights to the samples that were misclassified by previous weak classifiers. In this way, the subsequent weak classifiers focus more on the difficult samples that were not correctly classified by previous classifiers.

ADABOOST



It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.

ADABOOST



It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.

ADABOOST

```
from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import AdaBoostRegressor
```

XGBOOST

XGBOOST

- XGBoost (EXTREME GRADIENT Boost) works by building a series of decision trees in a gradient boosting framework.

- In this framework, each tree is built to correct the errors of the previous tree, with the final prediction being the sum of the predictions of all the individual trees.

- Another important feature of XGBoost is its ability to handle missing data.

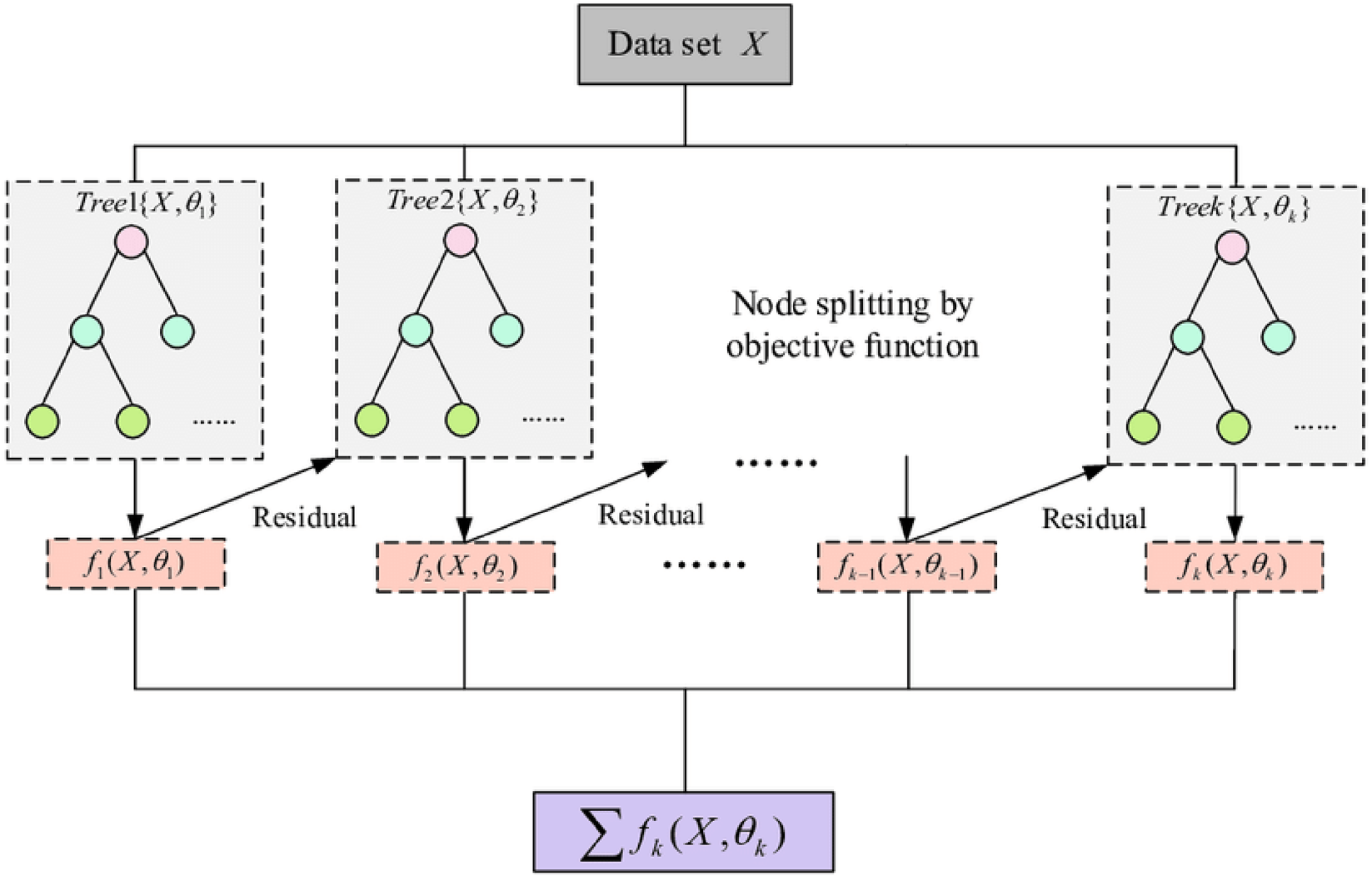
- XGBoost can automatically learn how to handle missing data by assigning different directions in the tree to the missing values and using the best split based on the available data.

XGBOOST

- XGBoost uses a gradient-based optimization approach to minimize the specified objective function, which typically includes a loss term to measure the difference between predictions and true values.

- XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization terms to control overfitting.

XGBOOST



XGBOOST

```
pip install xgboost

from xgboost import XGBClassifier

from xgboost import XGBRegressor
```

NAIVE BAYES

NAIVE BAYES CLASSIFIER

- Naive Bayes is a classification technique that is based on Bayes Theorem with an assumption that all the features that predicts the target value are independent of each other.

- It calculates the probability of each class and then pick the one with the highest probability.

- It has been successfully used for many purposes, but it works particularly well with natural language processing (NLP) problems.

NAIVE BAYES CLASSIFIER

- Naive Bayes classifier assumes that the features we use to predict the target are independent and do not affect each other.

- While in real-life data, features depend on each other in determining the target, but this is ignored by the Naive Bayes classifier.

MATH BEHIND NAIVE BAYES CLASSIFIER

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)}$$

calculating the posterior probability P(y | X) from the likelihood P(X | y) and prior probabilities P(y),P(X).

MATH BEHIND NAIVE BAYES CLASSIFIER

$$P(y|X) = \frac{P(x_1|y)*P(x_2|y) \ldots\ldots P(x_n|y) * P(y)}{P(x_1) * P(x_2) \ldots . P(x_n)}$$

NAIVE BAYES CLASSIFIER

	Not Spam	Spam
Dear	8	3
Visit	2	6
Invitation	5	2
Link	2	7
Friend	6	1
Hello	5	4
Discount	0	8
Money	1	7
Click	2	9
Dinner	3	0
Total Words	34	47

NAIVE BAYES CLASSIFIER

	Not Spam	Spam
Dear	8	3
Visit	2	6
Invitation	5	2
Link	2	7
Friend	6	1
Hello	5	4
Discount	0	8
Money	1	7
Click	2	9
Dinner	3	0
Total Words	34	47

exploring some probabilities:

- P(Dear|Not Spam) = 8/34
- P(Visit|Not Spam) = 2/34
- P(Dear|Spam) = 3/47
- P(Visit|Spam) = 6/47

and so on.

NAIVE BAYES CLASSIFIER

$$P(Hello\ Friend|Not\ Spam) = P(Hello|Not\ Spam) * P(Friend|Not\ Spam)$$

$$P(Not\ Spam|Hello\ Friend) = P(Hello|Not\ Spam) * P(Friend|Not\ Spam) * P(Not\ Spam)$$

$$P(Not\ Spam|Hello\ Friend) = \frac{5}{34} * \frac{6}{34} * \frac{15}{25} = 0.0155$$

now let's calculate the probability of being spam using the same procedure:

$$P(Spam|Hello\ Friend) = \frac{4}{47} * \frac{1}{47} * \frac{10}{25} = 0.00072$$

so, the message "*Hello friend*" is not a spam.

NAIVE BAYES CLASSIFIER

$$P(Spam|dear\ visit\ dinner\ money\ money\ money)$$

$$= P(dear\ visit\ dinner\ money\ money\ money|Spam) * P(Spam)$$

$$P(dear\ visit\ dinner\ money\ money\ money|Spam) = \frac{3}{47} * \frac{6}{47} * 0 * \left(\frac{7}{47}\right)^3 = 0$$

$$P(Spam|dear\ visit\ dinner\ money\ money\ money) = 0 * \frac{10}{25} = 0$$

oops!! Naive Bays says that this message is **not a spam**?!!!

This happened because the word "*dinner*" does not appear in the *spam* dataset, so that P(dinner | spam) = 0, hence all other probabilities will have no effect.

This is called the **Zero-Frequency Problem.**

LAPLACE SMOOTHING

- Laplace Smoothing is a technique for smoothing categorical data.

- A small-sample correction, or pseudo-count, will be incorporated in every probability estimate.

- Hence, no probability will be zero. this is a way of regularizing Naive Bayes.

NAIVE BAYES CLASSIFIER

$$P(dear\ visit\ dinner\ money\ money\ money|Spam) = \frac{3+1}{47+10} * \frac{6+1}{47+10} * \frac{0+1}{47+10} * \left(\frac{7+1}{47+10}\right)^3$$

$$= 0.000000418 = 4.18 * 10^{-7}$$

$$P(Spam|dear\ visit\ dinner\ money\ money\ money) = 4.18 * 10^{-7} * \frac{10}{25}$$

$$= 1.672 * 10^{-7}$$

Now it's correctly classified the message as spam.

TYPES OF NAIVE BAYES

- **Gaussian Naive Bayes (GNB):** This classifier assumes that the features follow a Gaussian (normal) distribution. It is suitable for continuous data where each feature can be modeled as a continuous variable. GNB works well for problems like text classification with continuous-valued features.

- **Multinomial Naive Bayes (MNB):** MNB is specifically designed for discrete data, such as text data. It's commonly used in natural language processing (NLP) tasks, such as text classification and spam email detection. In MNB, each feature represents the frequency of a term in a document.

- **Bernoulli Naive Bayes (BNB):** BNB is another variation suited for binary data, where features are either present or absent. It is commonly used in document classification tasks where each feature represents the presence or absence of a term in a document. BNB is also used in spam detection.

K NEAREST NEIGHBORS

K-NN ALGORITHM

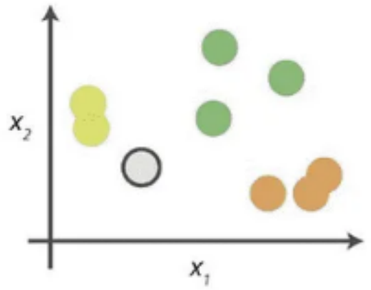
- K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems.

- The K-NN algorithm compares a new data entry to the values in a given data set (with different classes or categories).

- Based on its closeness or similarities in a given range (K) of neighbors, the algorithm assigns the new data to a class or category in the data set (training data).

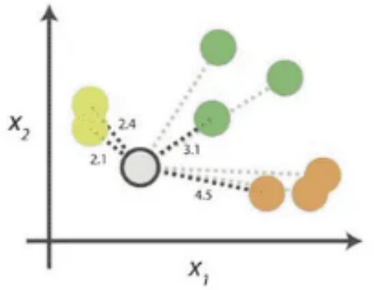
K-NN ALGORITHM

0. Look at the data



Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

1. Calculate distances



Start by calculating the distances between the grey point and all other points.

2. Find neighbours

Point	Distance	
○ ··· ●	2.1	→ 1st NN
○ ··· ●	2.4	→ 2nd NN
○ ··· ●	3.1	→ 3rd NN
○ ··· ●	4.5	→ 4th NN

Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

3. Vote on labels

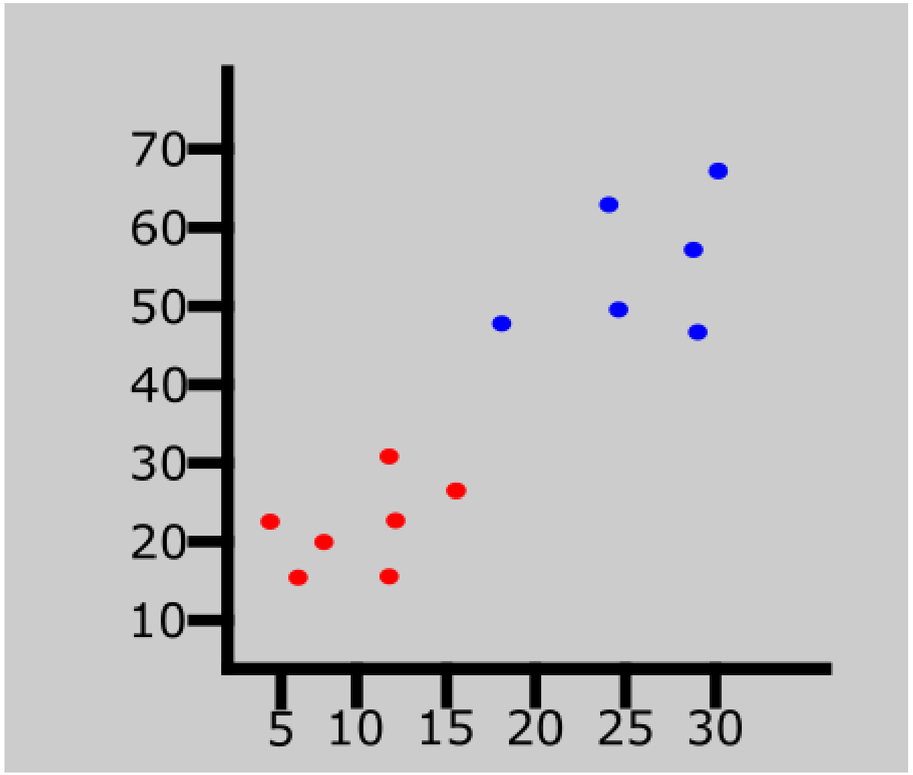
Class	# of votes
●	2
●	1
●	1

→ Class ● wins the vote!

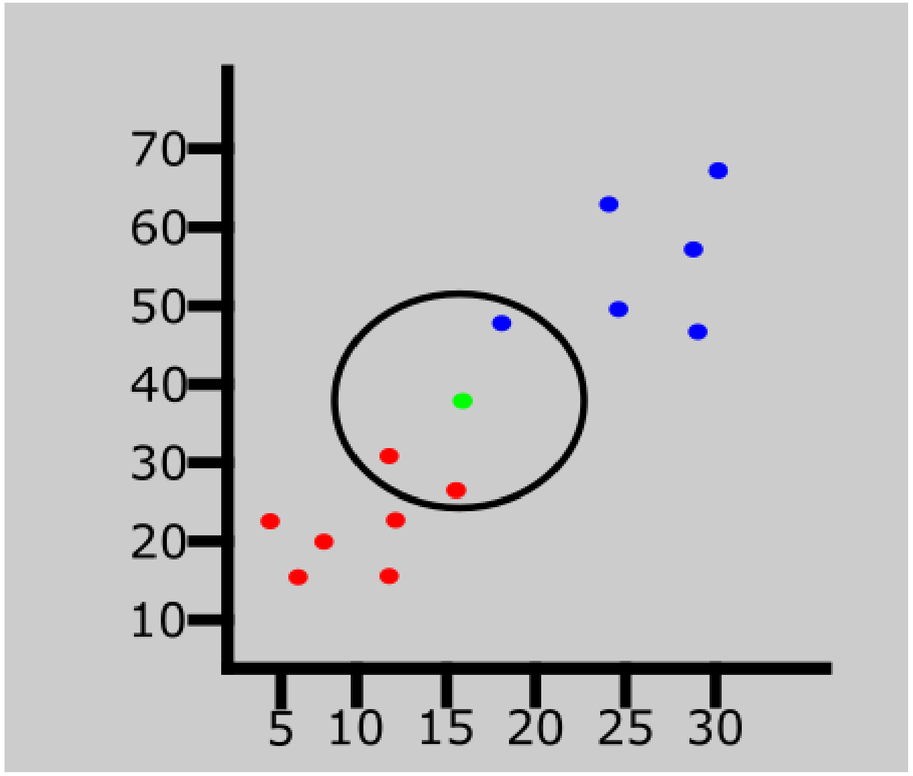
Point ○ is therefore predicted to be of class ●.

Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

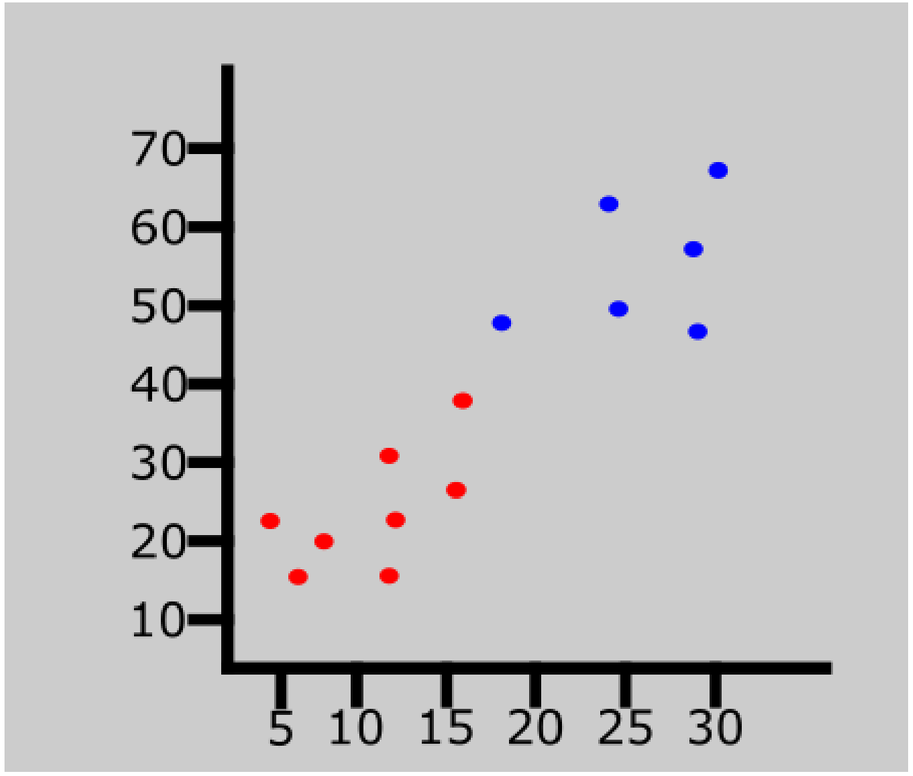
K-NN ALGORITHM



K-NN ALGORITHM



K-NN ALGORITHM



K-NN ALGORITHM

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

LET'S ASSUME THE VALUE OF K IS 5

K-NN ALGORITHM

BRIGHTNESS	SATURATION	CLASS
20	35	?

EUCLIDEAN DISTANCE

Here's the formula: $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Where:

- X_2 = New entry's brightness (20).
- X_1= Existing entry's brightness.
- Y_2 = New entry's saturation (35).
- Y_1 = Existing entry's saturation.

EUCLIDEAN DISTANCE

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17
70	70	Blue	61.03
60	90	Blue	68.01

K = 5

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17

EUCLIDEAN DISTANCE

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue
20	35	Red

STEPS IN KNN

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

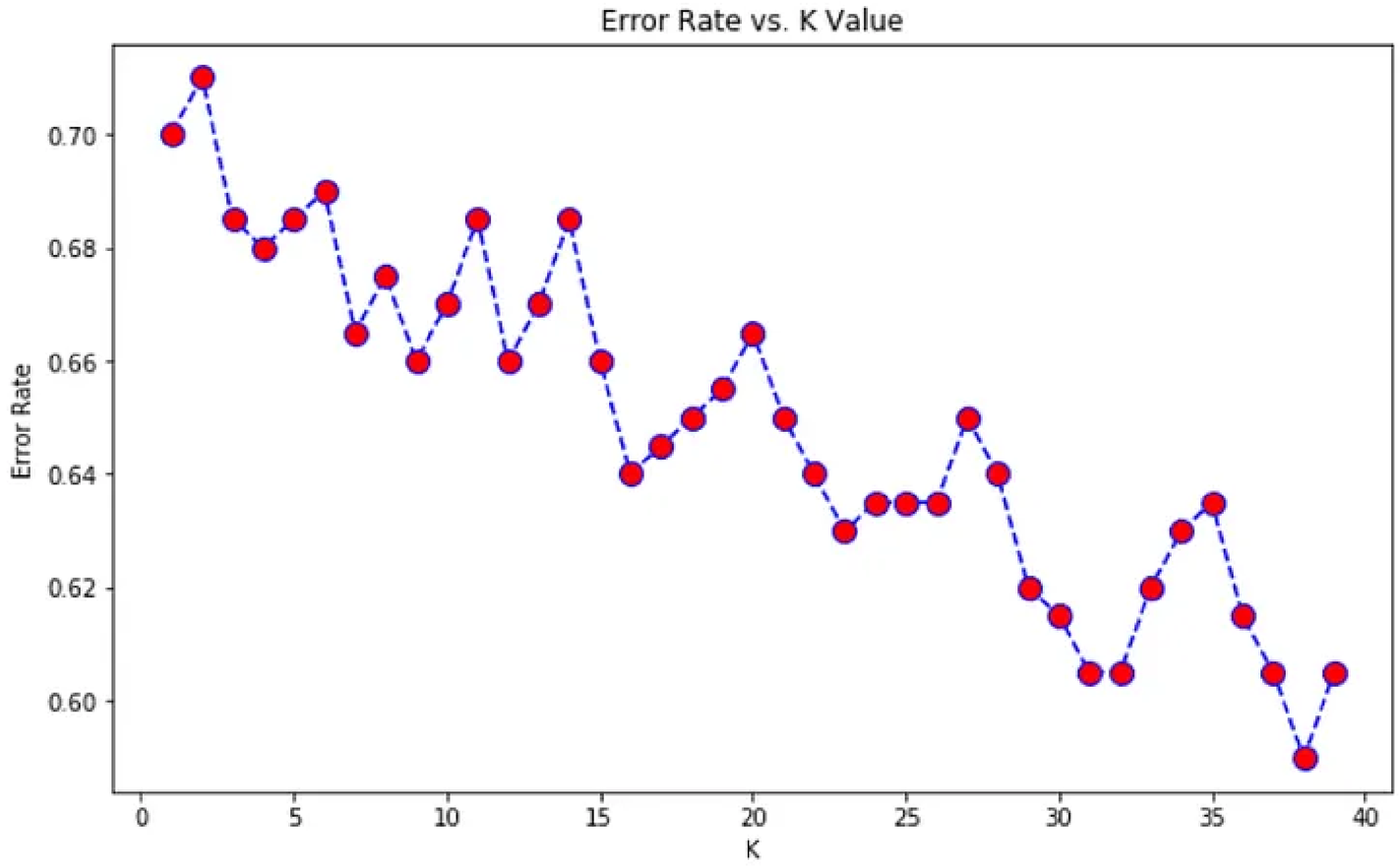
Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

STEPS IN KNN

```
Minimum error:- 0.59 at K = 37
```



KNN

```
error_rate = []

for i in range(1,40):

 knn = KNeighborsClassifier(n_neighbors=i)

 knn.fit(X_train,y_train)

 pred_i = knn.predict(X_test)

 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))

plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',

         marker='o',markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

knn = KNeighborsClassifier(n_neighbors=4)
```

THANK YOU