

**Experiment 1:**

1a) Implement and demonstrate Find-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Algorithm

1. Start with the most specific hypothesis.
 $h = \{\phi, \phi, \phi, \phi, \phi, \phi\}$
 2. Take the next example and if it is negative, then no changes occur to the hypothesis.
 3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
 4. Keep repeating the above steps till all the training examples are complete.
 5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.
-
1. Initialize h to the most specific hypothesis in H
 2. For each positive training instance x For each attribute constraint a, in h
If the constraint a, is satisfied by x Then do nothing
Else replace a, in h by the next more general constraint that is satisfied by x
 3. Output hypothesis h



1b) Implement and demonstrate Find-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

```
#Implementation
import pandas as pd
import numpy as np

data = pd.read_csv('/content/data.csv')
print('Data\n', data)

concepts = np.array(data[:, :-1])
print('Conceptes\n', concepts)
target = np.array(data[:, -1])
print('Target\n', target)

def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
            else:
                pass

    return specific_h
print('find-s\n', train(concepts, target))
```

Date :

```

Data
    sky air temp humidity    wind water forecast enjoy sport
0 sunny    warm    normal strong    warm    same      yes
1 sunny    warm    high  strong    warm    same      yes
2 rainy    cold    high  strong    warm    change    no
3 sunny    warm    high  strong    cool    change    yes
Conceptes
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
Target
['yes' 'yes' 'no' 'yes']
find-s
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

```



Experiment-2:

2(a) For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Definition

Candidate Elimination Algorithm

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of the Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified in the generalizing form.

Terms Used:

- **Concept learning:** Concept learning is basically the learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **$G = \{ '?', '?', '?', '?', ... \}$:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
- **$S = \{ 'p_1', 'p_1', 'p_1', ... \}$:** The number of p_i depends on a number of attributes.
- **Version Space:** It is an intermediate of general hypothesis and Specific hypothesis. It not only just writes one hypothesis but a set of all possible hypotheses based on training data- set.

Algorithm:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

if attribute_value == hypothesis_value:

Do nothing else:

replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

Make generalize hypothesis more specific.



Example:

Consider the dataset given below:

Sky	Temperature	Humid	Wind	Water	Forest	Output
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Algorithmic steps:

Initially : $G = [[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?]]$
 $S = [Null, Null, Null, Null, Null, Null]$

For instance 1 : $\langle 'sunny', 'warm', 'normal', 'strong', 'warm', 'same' \rangle$ and positive output.

$G1 = G$

$S1 = ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']$

For instance 2 : $\langle 'sunny', 'warm', 'high', 'strong', 'warm', 'same' \rangle$ and positive output.

$G2 = G$

$S2 = ['sunny', 'warm', '?', 'strong', 'warm', 'same']$

For instance 3 : $\langle 'rainy', 'cold', 'high', 'strong', 'warm', 'change' \rangle$ and negative output.

$G3 = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?]]$
 $S3 = S2$

For instance 4 : $\langle 'sunny', 'warm', 'high', 'strong', 'cool', 'change' \rangle$ and positive output.

$G4 = G3$

$S4 = ['sunny', 'warm', '?', 'strong', '?', ?]$

At last, by synchronizing the $G4$ and $S4$ algorithm produce the output.

Output :

$G = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?]]$

$S = ['sunny', 'warm', '?', 'strong', '?', ?]$



The Candidate Elimination Algorithm (CEA) is an improvement over the Find-S algorithm for classification tasks. While CEA shares some similarities with Find-S, it also has some essential differences that offer advantages and disadvantages. Here are some advantages and disadvantages of CEA in comparison with Find-S:

Advantages of CEA over Find-S:

1. Improved accuracy: CEA considers both positive and negative examples to generate the hypothesis, which can result in higher accuracy when dealing with noisy or incomplete data.
2. Flexibility: CEA can handle more complex classification tasks, such as those with multiple classes or non-linear decision boundaries.
3. More efficient: CEA reduces the number of hypotheses by generating a set of general hypotheses and then eliminating them one by one. This can result in faster processing and improved efficiency.
4. Better handling of continuous attributes: CEA can handle continuous attributes by creating boundaries for each attribute, which makes it more suitable for a wider range of datasets.

Disadvantages of CEA in comparison with Find-S:

1. More complex: CEA is a more complex algorithm than Find-S, which may make it more difficult for beginners or those without a strong background in machine learning to use and understand.
2. Higher memory requirements: CEA requires more memory to store the set of hypotheses and boundaries, which may make it less suitable for memory-constrained environments.
3. Slower processing for large datasets: CEA may become slower for larger datasets due to the increased number of hypotheses generated.
4. Higher potential for overfitting: The increased complexity of CEA may make it more prone to overfitting on the training data, especially if the dataset is small or has a high degree of noise.



Experiment-2

For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

#Implementation

```
import numpy as np
import pandas as pd

data = pd.read_csv(path+'/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1, "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = '?'

    print("Specific Boundary after ", i+1, "Instance is ", specific_h)
    print("Generic Boundary after ", i+1, "Instance is ", general_h)
    print("\n")
```



Date :

```
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
            for i in indices:
                general_h.remove(['?', '?', '?', '?', '?', '?'])
            return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Output :

G = [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

S = ['sunny', 'warm', '?', 'strong', '?', '?']

Experiment-3:

3a) Write a program to demonstrate the working of decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Analytical Analysis

ID3 algorithm, stands for **Iterative Dichotomiser 3**, is a **classification algorithm** that follows a **greedy approach** of **building a decision tree** by selecting a best attribute that yields **maximum Information Gain (IG)** or **minimum Entropy (H)**.

Information gain tells us how important a given attribute of the feature vectors is. We will use it to decide the ordering of attributes in the nodes of a decision tree.

Information Gain = entropy(parent) – [average entropy(children)]

$$\text{Entropy} = \sum_i -p_i \log_2 p_i$$

p_i is the probability of class i

Compute it as the proportion of class i in the set.

Entropy comes from information theory. The higher the entropy, the more the information content.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

-

$$\begin{aligned}
 H(S) &= -p(\text{yes}) * \log_2(p(\text{yes})) - p(\text{no}) * \log_2(p(\text{no})) \\
 &= - (9/14) * \log_2(9/14) - (5/14) * \log_2(5/14) \\
 &= - (-0.41) - (-0.53) \\
 &= 0.94
 \end{aligned}$$



First Attribute - Outlook

Categorical values - sunny, overcast and rain $H(\text{Outlook}=\text{sunny}) = -$

$$(2/5) \cdot \log(2/5) - (3/5) \cdot \log(3/5) = 0.971 \quad H(\text{Outlook}=\text{rain}) = -(3/5) \cdot \log(3/5) -$$

$$(2/5) \cdot \log(2/5) = 0.971 \quad H(\text{Outlook}=\text{overcast}) = -(4/4) \cdot \log(4/4) - 0 = 0$$

Average Entropy Information for Outlook -

$$I(\text{Outlook}) = p(\text{sunny}) \cdot H(\text{Outlook}=\text{sunny}) + p(\text{rain}) \cdot H(\text{Outlook}=\text{rain})$$

$$+ p(\text{overcast}) \cdot H(\text{Outlook}=\text{overcast})$$

$$= (5/14) \cdot 0.971 + (5/14) \cdot 0.971 + (4/14) \cdot 0$$

$$= 0.693$$

$$\text{Information Gain} = H(S) - I(\text{Outlook})$$

$$= 0.94 - 0.693$$

$$= 0.247$$

Second Attribute - Temperature

Categorical values - hot, mild, cool $H(\text{Temperature}=\text{hot}) = -(2/4) \cdot \log(2/4) -$

$$(2/4) \cdot \log(2/4) = 1$$

$$H(\text{Temperature}=\text{cool}) = -(3/4) \cdot \log(3/4) - (1/4) \cdot \log(1/4) = 0.811$$

$$H(\text{Temperature}=\text{mild}) = -(4/6) \cdot \log(4/6) - (2/6) \cdot \log(2/6) = 0.9179 \quad \text{Average}$$

Entropy Information for Temperature -

$$I(\text{Temperature}) = p(\text{hot}) \cdot H(\text{Temperature}=\text{hot})$$

$$+ p(\text{mild}) \cdot H(\text{Temperature}=\text{mild}) + p(\text{cool}) \cdot H(\text{Temperature}=\text{cool})$$

$$= (4/14) \cdot 1 + (4/14) \cdot 0.811 + (6/14) \cdot 0.9179$$

$$= 0.9108$$

$$\text{Information Gain} = H(S) - I(\text{Temperature})$$

$$= 0.94 - 0.9108$$

$$= 0.0292$$

Third Attribute - Humidity

Categorical values - high, normal

$$H(\text{Humidity}=\text{high}) = -(3/7) \cdot \log(3/7) - (4/7) \cdot \log(4/7) = 0.983$$

$$H(\text{Humidity}=\text{normal}) = -(6/7) \cdot \log(6/7) - (1/7) \cdot \log(1/7) = 0.591$$

Average Entropy Information for Humidity -

$$I(\text{Humidity}) = p(\text{high}) \cdot H(\text{Humidity}=\text{high}) + p(\text{normal}) \cdot H(\text{Humidity}=\text{normal})$$

$$= (7/14) \cdot 0.983 + (7/14) \cdot 0.591$$

$$= 0.787$$

$$\begin{aligned}\text{Information Gain} &= H(S) - I(\text{Humidity}) \\ &= 0.94 - 0.787 \\ &= 0.153\end{aligned}$$

Fourth Attribute - Wind

Categorical values - weak, strong

$$\begin{aligned}H(\text{Wind}=\text{weak}) &= -(6/8)*\log(6/8)-(2/8)*\log(2/8) = 0.811 \\ H(\text{Wind}=\text{strong}) &= -(3/6)*\log(3/6)-(3/6)*\log(3/6) = 1\end{aligned}$$

Average Entropy Information for Wind -

$$\begin{aligned}I(\text{Wind}) &= p(\text{weak})*H(\text{Wind}=\text{weak}) + p(\text{strong})*H(\text{Wind}=\text{strong}) \\ &= (8/14)*0.811 + (6/14)*1 \\ &= 0.892\end{aligned}$$

$$\begin{aligned}\text{Information Gain} &= H(S) - I(\text{Wind}) \\ &= 0.94 - 0.892 \\ &= 0.048\end{aligned}$$

$$\text{Information Gain(Outlook)} = 0.247 \quad \text{Information Gain (Temperature)}=0.0292$$



First Attribute - Temperature

Categorical values - hot, mild, cool

$$H(\text{Sunny, Temperature}=\text{hot}) = -(2/2)*\log(2/2) = 0 \quad H(\text{Sunny, Temperature}=\text{cool}) = -(1)*\log(1) = 0$$

$$H(\text{Sunny, Temperature}=\text{mild}) = -(1/2)*\log(1/2)-(1/2)*\log(1/2) = 1$$

Average Entropy Information for Temperature -

Entropy Information for Temperature -

$$\begin{aligned}I(\text{Sunny, Temperature}) &= p(\text{Sunny, hot})*H(\text{Sunny, Temperature}=\text{hot}) + \\ &\quad p(\text{Sunny, mild})*H(\text{Sunny, Temperature}=\text{mild}) + p(\text{Sunny, cool})*H(\text{Sunny, Temperature}=\text{cool}) \\ &= (2/5)*0 + (1/5)*0 + (2/5)*1 \\ &= 0.4\end{aligned}$$

$$\begin{aligned}\text{Information Gain} &= H(\text{Sunny}) - I(\text{Sunny, Temperature}) \\ &= 0.971 - 0.4 \\ &= 0.571\end{aligned}$$

Second Attribute - Humidity

Categorical values - high, normal

$$H(\text{Sunny, Humidity=high}) = -0 - (3/3) \cdot \log(3/3) = 0$$

$$H(\text{Sunny, Humidity=normal}) = -(2/2) \cdot \log(2/2) - 0 = 0$$

Average Entropy Information for Humidity -

$$I(\text{Sunny, Humidity}) = p(\text{Sunny, high}) \cdot H(\text{Sunny, Humidity=high}) +$$

$$p(\text{Sunny, normal}) \cdot H(\text{Sunny, Humidity=normal})$$

$$= (3/5) \cdot 0 + (2/5) \cdot 0$$

$$= 0$$

$$= 0$$

$$\text{Information Gain} = H(\text{Sunny}) - I(\text{Sunny, Humidity})$$

$$= 0.971 - 0$$

$$= 0.971$$

Third Attribute - Wind

Categorical values - weak, strong

$$H(\text{Sunny, Wind=weak}) = -(1/3) \cdot \log(1/3) - (2/3) \cdot \log(2/3) = 0.918$$

$$H(\text{Sunny, Wind=strong}) = -(1/2) \cdot \log(1/2) - (1/2) \cdot \log(1/2) = 1$$

Average Entropy Information for Wind -

$$I(\text{Sunny, Wind}) = p(\text{Sunny, weak}) \cdot H(\text{Sunny, Wind=weak}) + p(\text{Sunny, strong}) \cdot H(\text{Sunny, Wind=strong})$$

$$= (3/5) \cdot 0.918 + (2/5) \cdot 1$$

$$= 0.9508$$

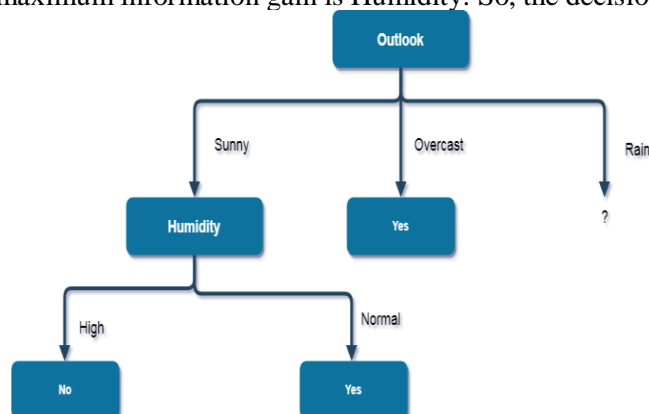
$$= 0.9508$$

$$\text{Information Gain} = H(\text{Sunny}) - I(\text{Sunny, Wind})$$

$$= 0.971 - 0.9508$$

$$= 0.0202$$

Here, the attribute with maximum information gain is Humidity. So, the decision tree built so far





Now, finding the best attribute for splitting the data with Outlook=Sunny values{ Dataset rows = [4, 5, 6, 10, 14]}.

Complete entropy of Rain is -

$$\begin{aligned} H(S) &= -p(\text{yes}) * \log_2(p(\text{yes})) - p(\text{no}) * \log_2(p(\text{no})) \\ &= - (3/5) * \log_2(3/5) - (2/5) * \log_2(2/5) \\ &= 0.971 \end{aligned}$$

First Attribute - Temperature

Categorical values - mild, cool

$$\begin{aligned} H(\text{Rain, Temperature=cool}) &= -(1/2)*\log_2(1/2) - (1/2)*\log_2(1/2) = 1 \\ H(\text{Rain, Temperature=mild}) &= -(2/3)*\log_2(2/3) - (1/3)*\log_2(1/3) = 0.918 \end{aligned}$$

Entropy Information for Temperature -

$$\begin{aligned} I(\text{Rain, Temperature}) &= p(\text{Rain, mild}) * H(\text{Rain, Temperature=mild}) + p(\text{Rain, cool}) * H(\text{Rain, Temperature=cool}) \\ &= (2/5) * 1 + (3/5) * 0.918 \\ &= 0.9508 \end{aligned}$$

$$\begin{aligned} \text{Information Gain} &= H(\text{Rain}) - I(\text{Rain, Temperature}) \\ &= 0.971 - 0.9508 \\ &= 0.0202 \end{aligned}$$

Second Attribute - Wind

Categorical values - weak, strong $H(\text{Wind=weak}) = -(3/3)*\log_2(3/3) - 0 = 0$

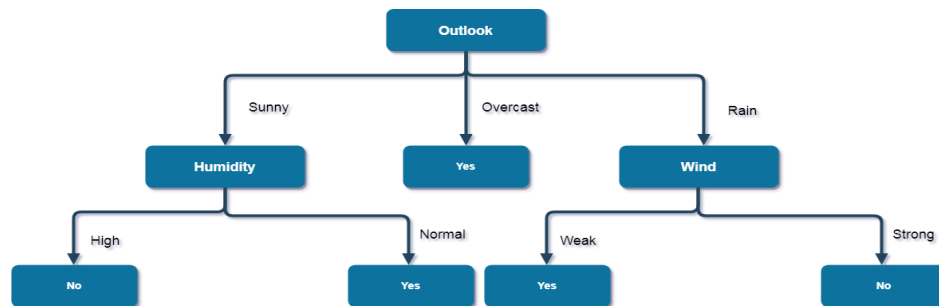
$$H(\text{Wind=strong}) = 0 - (2/2)*\log_2(2/2) = 0$$

Average Entropy Information for Wind -

$$\begin{aligned} I(\text{Wind}) &= p(\text{Rain, weak}) * H(\text{Rain, Wind=weak}) + p(\text{Rain, strong}) * H(\text{Rain, Wind=strong}) \\ &= (3/5) * 0 + (2/5) * 0 \\ &= 0 \end{aligned}$$

$$\begin{aligned} \text{Information Gain} &= H(\text{Rain}) - I(\text{Rain, Wind}) \\ &= 0.971 - 0 \\ &= 0.971 \end{aligned}$$

Here, the attribute with maximum information gain is Wind. So, the decision tree built so far



-

Here, when Outlook = Rain and Wind = Strong, it is a pure class of category "no". And When Outlook = Rain and Wind = Weak, it is again a pure class of category "yes". **And this is our final desired tree for the given dataset.**

3(b) Write a program to demonstrate the working of decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
#Implementation import pandas as pd import math import numpy as np

data = pd.read_csv("/content/PlayTennis.csv") features = [feat for feat in
data] features.remove("answer")

#Create a class named Node with four members children, value, isLeaf and
pred.

class Node:
def __init__(self): self.children = [] self.value = "" self.isLeaf = False self.pred =
""

#Define a function called entropy to find the entropy of the dataset.

def entropy(examples): pos = 0.0
neg = 0.0
for _, row in examples.iterrows(): if row["answer"] == "yes":
pos += 1 else:
neg += 1
if pos == 0.0 or neg == 0.0: return 0.0
else:
p = pos / (pos + neg) n = neg / (pos + neg)
return -(p * math.log(p, 2) + n * math.log(n, 2))

#Define a function named info_gain to find the gain of the attribute

def info_gain(examples, attr):
```



```
uniq = np.unique(examples[attr]) #print ("\n",uniq)
gain = entropy(examples) #print ("\n",gain)
for u in uniq:
    subdata = examples[examples[attr] == u] #print ("\n",subdata)
    sub_e = entropy(subdata)
    gain -= (float(len(subdata)) / float(len(examples))) * sub_e #print ("\n",gain)
return gain
#Define a function named ID3 to get the decision tree for the given dataset

def ID3(examples, attrs): root = Node()

    max_gain = 0 max_feat = ""
for feature in attrs: #print ("\n",examples)
    gain = info_gain(examples, feature) if gain > max_gain:
        max_gain = gain max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat) uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
for u in uniq: #print ("\n",u)
    subdata = examples[examples[max_feat] == u] #print ("\n",subdata)
if entropy(subdata) == 0.0: newNode = Node() newNode.isLeaf = True
    newNode.value = u
    newNode.pred = np.unique(subdata["answer"]) root.children.append(newNode)
else:
```




```

dummyNode = Node() dummyNode.value = u new_attrs = attrs.copy()
new_attrs.remove(max_feat) child = ID3(subdata, new_attrs)
dummyNode.children.append(child) root.children.append(dummyNode)

return root

#Define a function named printTree to draw the decision tree

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="") print(root.value, end="") if root.isLeaf:
            print(" -> ", root.pred) print()
    for child in root.children: printTree(child, depth + 1)

#Define a function named classify to classify the new example

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new, " is:", child.pred) exit
            else:
                classify (child.children[0], new)

#Finally, call the ID3, printTree and classify functions

root = ID3(data, features) print("Decision Tree is:") printTree(root)
print (" ----- ")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal",
"wind":"strong"} classify (root, new)

```

Decision Tree is:

outlook

overcast -> ['yes']

rain

wind

strong -> ['no']

weak -> ['yes']

sunny

humidity

high -> ['no']

normal -> ['yes']

=====

Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot',
'humidity': 'normal', 'wind': 'strong'} is: ['yes']

Experiment-4

4(a) Exercises to solve real world problems using the following machine learning methods:

a) Linear regression b) Logistic regression c) Binary classifier

a) Linear Regression

Linear regression shows the linear relationship between the independent(predictor) variable i.e. X-axis and the dependent(output) variable i.e. Y-axis, called linear regression. Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable. Linear regression is a fundamental machine learning algorithm used for predicting numerical values based on input features. It assumes a linear relationship between the features and the target variable.

Since the Linear Regression algorithm represents a linear relationship between a dependent (y) and one or more independent (x) variables, it is known as Linear Regression. This means it finds how the value of the dependent variable changes according to the change in the value of the independent variable. The relation between independent and dependent variables is a straight line with a slope. The model learns the coefficients that best fit the data and can make predictions for new inputs.

Linear regression can be expressed mathematically as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Here,

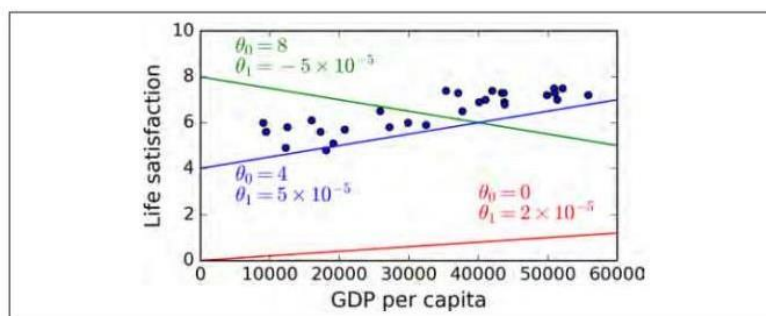
Y = Dependent Variable X = Independent Variable β_0 = intercept of the line

β_1 = Linear regression coefficient (slope of the line) ϵ = random error

The last parameter, random error ϵ , is required as the best fit line also doesn't include the data points perfectly.

Example:

$$life_satisfaction = \theta_0 + \theta_1 \times GDP_per_capita$$



Linear Models

Manhattan distance

Manhattan distance is a distance metric between two points in a N dimensional vector space.

$$|x_1 - x_2| + |y_1 - y_2|$$

Hamming distance

Hamming distance is the number of bit positions in which the two bits are different.

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

Euclidean distance

Euclidean distance is the distance between two points.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

p-norm

The p -norm is related to the generalized mean or power mean.

1, 2, and ∞ norms.

b) Logistic Regression

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a Sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

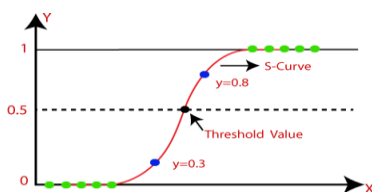
Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.





c) **Binary Classifier**

Binary classification is the task of classifying the elements of a set into one of two groups (each called class) on the basis of a classification rule.

In a binary classification task, the goal is to classify the input data into two mutually exclusive categories. The training data in such a situation is labeled in a binary format: true and false; positive and negative; 0 and 1; spam and not spam, etc. depending on the problem being tackled.

For instance, we might want to detect whether a given image is a truck or a boat.

In a medical diagnosis, a binary classifier for a specific disease could take a patient's symptoms as input features and predict whether the patient is healthy or has the disease. The possible outcomes of the diagnosis are positive and negative.



4(b) Exercises to solve real world problems using the following machine learning methods:

a) Linear regression b) Logistic regression c) Binary classifier

a) Linear regression

#Implementation

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
print('x',x)
print('y',y)
model = LinearRegression()
model.fit(x, y)
#model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)
print(f'coefficient of determination: {r_sq}')
print(f'intercept: {model.intercept_}')
print(f'slope: {model.coef_}')
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print(f'intercept: {new_model.intercept_}')
print(f'slope: {new_model.coef_}')
y_pred = model.predict(x)
print(f'predicted response:\n{y_pred}')
y_pred = model.intercept_ + model.coef_ * x
print(f'predicted response:\n{y_pred}')
x_new = np.arange(5).reshape((-1, 1))
print('x_new',x_new)
y_new = model.predict(x_new)
print('y_new',y_new)
```

```
x [[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
y [ 5 20 14 32 22 38]
coefficient of determination: 0.7158756137479542
intercept: 5.633333333333329
slope: [0.54]
intercept: [5.63333333]
slope: [[0.54]]
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
x_new [[0]
 [1]
 [2]
 [3]
 [4]]
y_new [5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]
```

**a1 Linear model coefficients**

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y) print(r)
print(slope) print(intercept) print(std_err) print(p)
#The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and
- 1) means 100% related.
Output
```

```
-0.758591524376155
-1.7512877115526118
103.10596026490066
0.453536157607742
0.0026468739224561064
```

b) Logistic Regression

```
import numpy
from sklearn import linear_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37,
4.96, 4.52, 3.69,
5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression() logr.fit(X,y)

predicted = logr.predict(numpy.array([3.46]).reshape(-1,1)) print(predicted)

Output: [0]
```




C) Binary Classifier

```
#Implementation
import pandas as pd
import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv')
df.head()
df.drop(columns=['#', 'Name', 'Type 1', 'Type 2'], inplace=True)
dfdf.Legendary.value_counts()
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['Legendary'] = label_encoder.fit_transform(df['Legendary'])
df['Legendary'].unique()
dfdf.Legendary.value_counts()
from sklearn.model_selection import train_test_split
pokemon_features = df.drop("Legendary", axis=1)
target = df["Legendary"]
X_train, X_test, Y_train, Y_test = train_test_split(pokemon_features, target, test_size=0.20, random_state=0)
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_pred_lr = lr.predict(X_test)
score_lr = round(accuracy_score(Y_pred_lr, Y_test)*100, 2)
print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```



```
The accuracy score achieved using Logistic Regression is: 92.5 %
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Experiment-5

5(a) Develop a program for Bias, Variance, Remove duplicates , Cross Validation

#bias, variance

```
import numpy as np # linear algebra
from mlxtend.evaluate import bias_variance_decomp
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)

from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from sklearn.neighbors
import KNeighborsClassifier from sklearn.neighbors import
KNeighborsRegressor from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.linear_model import LinearRegression import
matplotlib.pyplot as plt
%matplotlib inline data=pd.read_csv('/content/sample_data/heart.csv')
# separate into inputs and outputs
data = data.values
X, y = data[:, :-1], data[:, -1] # split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=1)
# define the model
model = LinearRegression() # estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train,
X_test, y_test, loss='mse', num_rounds=200, random_seed=1)
# summarize results print('MSE: %.3f' % mse) print('Bias: %.3f' %
bias) print('Variance: %.3f' % var)
```

#Output

```
➡ MSE: 0.150
Bias: 0.140
Variance: 0.010
```

Date :

#Removing duplicates

The datasets are first merged into a single dataframe, 'iris_df', using the concat() method. Then, the duplicates are removed from the merged dataset using the drop_duplicates() method.

#Removing duplicates

```
import pandas as pd
from sklearn.datasets import load_iris iris_data = load_iris()
iris_df = pd.DataFrame(iris_data.data,
columns=iris_data.feature_names) print('Shape\n',iris_df.shape)
print('Null\n',iris_df.isnull().sum())
print('Duplicates\n',iris_df.duplicated().sum())
print('Duplicates\n',iris_df.duplicated().sum())
print('after
re
moving
Duplicates\n',iris_df.duplicated().sum().shape)print('Describe\n',iris_df
.describe()) Output:
```

```
➡ Shape
(150, 4)
Null
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
Duplicates
1
after removing Duplicates
()
Describe
```

	sepal length (cm)	sepal width (cm)	petal length (cm)
count	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000
std	0.828066	0.435866	1.765298
min	4.300000	2.000000	1.000000
25%	5.100000	2.800000	1.600000
50%	5.800000	3.000000	4.350000
75%	6.400000	3.300000	5.100000
max	7.900000	4.400000	6.900000

```

petal width (cm)
count      150.000000
mean        1.199333
std         0.762238
min         0.100000
25%         0.300000
50%         1.300000
75%         1.800000
max         2.500000
```



#k-fold Cross Validation

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score X, y =
datasets.load_iris(return_X_y=True)
clf = DecisionTreeClassifier(random_state=42) k_folds =
KFold(n_splits = 5)
scores = cross_val_score(clf, X, y, cv = k_folds) print("Cross
Validation Scores: ", scores) print("Average CV Score: ",
scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```

Cross Validation Scores: [1.          1.          0.83333333 0.93333333 0.8
Average CV Score: 0.9133333333333333
Number of CV Scores used in Average: 5

```


#stratified cross fold

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold,
cross_val_score X, y = datasets.load_iris(return_X_y=True)
clf = DecisionTreeClassifier(random_state=42) sk_folds =
StratifiedKFold(n_splits = 5)
scores = cross_val_score(clf, X, y, cv = sk_folds) print("Cross
Validation Scores: ", scores) print("Average CV Score: ",
scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```
➡ Cross Validation Scores: [0.96666667 0.96666667 0.9          0.93333333 1.          ]
Average CV Score: 0.9533333333333334
Number of CV Scores used in Average: 5
```

#Leave-one-out-cut

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut, cross_val_score
X, y = datasets.load_iris(return_X_y=True)
clf = DecisionTreeClassifier(random_state=42) loo = LeaveOneOut()
scores = cross_val_score(clf, X, y, cv = loo) print("Cross Validation
Scores: ", scores) print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```
 Cross Validation Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.  
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.  
1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.  
1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.  
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.  
1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.  
1. 1. 1. 1. 1.]  
Average CV Score: 0.94  
Number of CV Scores used in Average: 150
```

```
#Leave-P-out cross validation from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeavePOut, cross_val_score X,

y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

lpo = LeavePOut(p=2)
scores = cross_val_score(clf, X, y, cv = lpo)

print("Cross Validation Scores: ", scores) print("Average CV Score:
", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```
➡ Cross Validation Scores: [1. 1. 1. ... 1. 1. 1.]
Average CV Score: 0.9485011185682327
Number of CV Scores used in Average: 11175
```

#Shuffle_split cross validation

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import ShuffleSplit, cross_val_score X,

y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

ss = ShuffleSplit(train_size=0.6, test_size=0.3, n_splits = 5) scores =

cross_val_score(clf, X, y, cv = ss)

print("Cross Validation Scores: ", scores) print("Average CV Score:
", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```
➡ Cross Validation Scores: [0.93333333 0.97777778 0.91111111 0.93333333 0.91111111]
Average CV Score: 0.9333333333333333
Number of CV Scores used in Average: 5
```



5(b) Develop a program for Bias, Variance, Remove duplicates, Cross Validation

A supervised Machine Learning model aims to train itself on the input variables(X) in such a way that the predicted values(Y) are as close to the actual values as possible. This difference between the actual values and predicted values is the error and it is used to evaluate the model. The error for any supervised Machine Learning algorithm comprises of 3 parts:

☐ Bias error ☐ Variance error ☐ The noise

☐ Bias is simply defined as the inability of the model because of that there is some difference or error occurring between the model's predicted value and the actual value. These differences between actual or expected values and the predicted values are known as error or bias error or error due to bias. Bias is a systematic error that occurs due to wrong assumptions in the machine learning process.

Variance is the measure of spread in data from its mean position. In machine learning variance is the amount by which the performance of a predictive model changes when it is trained on different subsets of the training data. More specifically, variance is the variability of the model that how much it is sensitive to another subset of the training dataset. i.e. how much it can adjust on the new subset of the training dataset.

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance.

The main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model's generalization performance, i.e., its ability to perform well on new, unseen data.

There are several types of cross validation techniques, including k-fold cross validation, leave-one-out cross validation, and stratified cross validation. The choice of technique depends on the size and nature of the data, as well as the specific requirements of the modeling problem.



Experiment-06

Svm_classifier

```
#SVM classifier import pandas as pd import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv') df.head()
df.drop(columns=['#','Name', 'Type 1', 'Type 2'],inplace=True) df
df.Legendary.value_counts() # Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels. label_encoder =
preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Legendary']= label_encoder.fit_transform(df['Legendary'])

df['Legendary'].unique() df df.Legendary.value_counts()
from sklearn.model_selection import train_test_split

pokemon_features = df.drop("Legendary",axis=1) target = df["Legendary"]

X_train,X_test,Y_train,Y_test =
train_test_split(pokemon_features,target,test_size=0.20,random_state=0) from sklearn.metrics import
accuracy_score
from sklearn import svm

sv = svm.SVC(kernel='linear') sv.fit(X_train, Y_train) Y_pred_svm = sv.predict(X_test)

score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

The accuracy score achieved using Linear SVM is: 94.38 %

Experiment-07

Gaussian_NB classifier:

Gaussian Naive Bayes is a machine learning classification technique based on a probabilistic approach that assumes each class follows a normal distribution. It assumes each parameter has an independent capacity of predicting the output variable.

```
# GaussianNB import pandas as pd import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv') df.head()
df.drop(columns=['#','Name', 'Type 1', 'Type 2'],inplace=True) df
df.Legendary.value_counts()
from sklearn import preprocessing label_encoder =

preprocessing.LabelEncoder()

df['Legendary']= label_encoder.fit_transform(df['Legendary'])

df['Legendary'].unique() df
df.Legendary.value_counts()
from sklearn.model_selection import train_test_split

pokemon_features = df.drop("Legendary",axis=1) target = df["Legendary"]

X_train,X_test,Y_train,Y_test =
train_test_split(pokemon_features,target,test_size=0.20,random_state=0)
from sklearn.metrics import accuracy_score

from sklearn.naive_bayes import GaussianNB nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+"
%")
```



The accuracy score achieved using Naive Bayes is: 91.88 %



EXPERIMENT-08

KNN classifier

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).

Algorithm

Step #1 - Assign a value to K.

Step #2 - Calculate the distance between the new data entry and all other existing data entries (you'll learn how to do this shortly). Arrange them in ascending order.

Step #3 - Find the K nearest neighbors to the new entry based on the calculated distances. Step #4 - Assign the new data entry to the majority class in the nearest neighbors.

```
#KNN classifier import pandas as pd import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv') df.head()
df.drop(columns=['#','Name', 'Type 1', 'Type 2'],inplace=True) df
df.Legendary.value_counts() # Import label encoder
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder() df['Legendary']=
label_encoder.fit_transform(df['Legendary'])

df['Legendary'].unique() df
df.Legendary.value_counts()
from sklearn.model_selection import train_test_split

pokemon_features = df.drop("Legendary",axis=1) target = df["Legendary"]

X_train,X_test,Y_train,Y_test =
train_test_split(pokemon_features,target,test_size=0.20,random_state=0)
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7) knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2) print("The

accuracy score achieved using KNN is: "+str(score_knn)+" %")
```



EXPERIMENT-09

Decision tree classifier

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

```
#Decision tree classifier import pandas as pd import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv') df.head()
df.drop(columns=['#','Name', 'Type 1', 'Type 2'],inplace=True) df
df.Legendary.value_counts()
from sklearn import preprocessing label_encoder =
preprocessing.LabelEncoder()
df['Legendary']= label_encoder.fit_transform(df['Legendary'])
df['Legendary'].unique() df
df.Legendary.value_counts()
from sklearn.model_selection import train_test_split pokemon_features =
df.drop("Legendary",axis=1) target = df["Legendary"]
X_train,X_test,Y_train,Y_test =
train_test_split(pokemon_features,target,test_size=0.20,random_state=0)
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier max_accuracy = 0
for x in range(200):
dt = DecisionTreeClassifier(random_state=x) dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
if(current_accuracy>max_accuracy):
max_accuracy = current_accuracy best_x = x
dt = DecisionTreeClassifier(random_state=best_x) dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
print("The accuracy score achieved using Decision Tree is:
"+str(score_dt)+" %")
```

➡ The accuracy score achieved using Decision Tree is: 96.25 %

EXPERIMENT-10

K-means

K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

#Analysis

make_blobs from **sklearn.datasets** module for doing this.

Object	X_value	Y_value
Object 1	1.005079	4.594642
Object 2	1.128478	4.328122
Object 3	2.117881	0.726845
Object 4	0.955626	4.385907
Object 5	-1.35402	2.769449
Object 6	-1.07295	2.627009
Object 7	-2.0375	3.048606
Object 8	2.354083	0.856632
Object 9	2.14404	0.964399
Object 10	1.166288	4.273516

Imports

```
from sklearn.datasets.samples_generator import make_blobs
```

Generate 2D data points

```
X, _ = make_blobs(n_samples=10, centers=3, n_features=2,  
cluster_std=0.2, random_state=0)
```

Convert the data points into a pandas DataFrame import pandas as pd

Generate indicators for the data points obj_names = []

```
for i in range(1, 11):
```

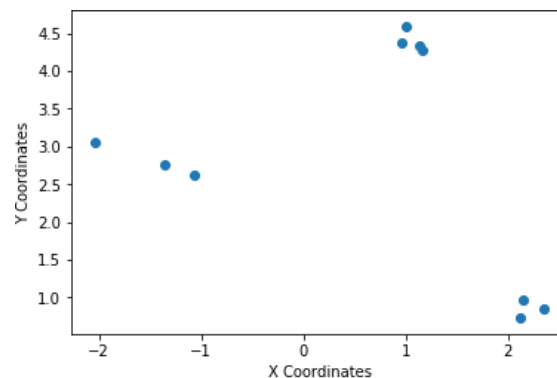
```
obj = "Object " + str(i) obj_names.append(obj)
```

Create a pandas DataFrame with the names and (x, y) coordinates data

```
= pd.DataFrame({  
'Object': obj_names, 'X_value': X[:, 0],  
'Y_value': X[:, -1]  
})
```

Preview the data print(data.head())

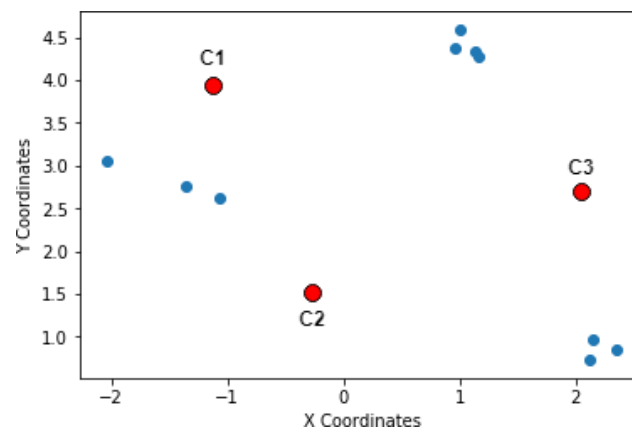
	Object	X_value	Y_value
0	Object 1	1.005079	4.594642
1	Object 2	1.128478	4.328122
2	Object 3	2.117881	0.726845
3	Object 4	0.955626	4.385907
4	Object 5	-1.354017	2.769449



Visual representation of the data points

- Initialize random centroids**

You start the process by taking three(as we decided K to be 3) random points (in the form of (x, y)). These points are called **centroids** which is just a fancy name for denoting *centers*. Let's name these three points - **C1**, **C2**, and **C3** so that you can refer them later.



Step 1 in K-Means: Random centroids

- Calculate distances between the centroids and the data points**

Next, you measure the distances of the data points from these three randomly chosen points. A very popular choice of distance measurement function, in this case, is the Euclidean distance.

Briefly, if there are n points on a 2D space (just like the above figure) and their coordinates are denoted by (x_i, y_i) , then the Euclidean distance between any two points $((x_1, y_1)$ and (x_2, y_2)) on this space is given by:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Equation for Euclidean distance

Suppose the coordinates of C_1 , C_2 and C_3 are $(-1, 4)$, $(-0.2, 1.5)$ and $(2, 2.5)$ respectively. Let's now write a few lines of Python code which will calculate the Euclidean distances between the data-points and these randomly chosen centroids. We start by initializing the centroids.

```
# Initialize the centroids c1 = (-1, 4)
```

```
c2 = (-0.2, 1.5)
```

```
c3 = (2, 2.5)
```

Next, we write a small helper function to calculate the Euclidean distances between the data points and centroids.

```
# A helper function to calculate the Euclidean distance between the data  
# points and the centroids
```

```
def calculate_distance(centroid, X, Y): distances = []
```

```
# Unpack the x and y coordinates of the centroid c_x, c_y = centroid
```

```
# Iterate over the data points and calculate the distance using the #  
given formula
```

```
for x, y in list(zip(X, Y)): root_diff_x = (x - c_x) ** 2  
root_diff_y = (y - c_y) ** 2
```

```
distance = np.sqrt(root_diff_x + root_diff_y) distances.append(distance)
```

```
return distances
```

We can now apply this function to the data points and assign the results in the DataFrame accordingly.

```
# Calculate the distance and assign them to the DataFrame accordingly
```

```
data['C1_Distance'] = calculate_distance(c1, data.X_value,
```

```
data.Y_value) data['C2_Distance'] = calculate_distance(c2,
```

```
data.X_value, data.Y_value) data['C3_Distance'] =
```

```
calculate_distance(c3, data.X_value, data.Y_value)
```

```
# Preview the data print(data.head())
```

The output should be like the following:



	Object	X_value	Y_value	C1_Distance	C2_Distance	C3_Distance
0	Object 1	1.005079	4.594642	2.091397	3.320997	2.318921
1	Object 2	1.128478	4.328122	2.153620	3.124601	2.025236
2	Object 3	2.117881	0.726845	4.520479	2.443428	1.777070
3	Object 4	0.955626	4.385907	1.993339	3.108686	2.155774
4	Object 5	-1.354017	2.769449	1.280462	1.715592	3.364823

Snapshot of the Euclidean distances between the data points and the centroids

Time to study the next step in the algorithm.

- **Compare, assign, mean and repeat**

This is fundamentally the last step of the K-Means clustering algorithm. Once you have the distances between the data points and the centroids, you compare the distances and take the *smallest ones*. The centroid to which the distance for a particular data point is the smallest, that centroid gets assigned as the cluster for that particular data point. Let's do this programmatically.

Get the minimum distance centroids

```
data['Cluster'] = data[['C1_Distance',
                        'C2_Distance', 'C3_Distance']].apply(np.argmin,
axis=1)
```

Map the centroids accordingly and rename them

```
data['Cluster'] = data['Cluster'].map({'C1_Distance': 'C1',
'C2_Distance': 'C2', 'C3_Distance': 'C3'})
```

Get a preview of the data print(data.head(10))

You get a nicely formatted output:

	Object	X_value	Y_value	C1_Distance	C2_Distance	C3_Distance	Cluster
0	Object 1	1.005079	4.594642	2.091397	3.320997	2.318921	C1
1	Object 2	1.128478	4.328122	2.153620	3.124601	2.025236	C3
2	Object 3	2.117881	0.726845	4.520479	2.443428	1.777070	C3
3	Object 4	0.955626	4.385907	1.993339	3.108686	2.155774	C1
4	Object 5	-1.354017	2.769449	1.280462	1.715592	3.364823	C1
5	Object 6	-1.072953	2.627009	1.374928	1.425551	3.075577	C1
6	Object 7	-2.037502	3.048606	1.407679	2.403038	4.074603	C1
7	Object 8	2.354083	0.856632	4.596807	2.633869	1.681081	C3
8	Object 9	2.144040	0.964399	4.370339	2.404453	1.542342	C3
9	Object 10	1.166288	4.273516	2.183487	3.091785	1.959703	C3

**Clusters after one iteration of K-means**

With this step, we complete an iteration of the K-Means clustering algorithm. Take a closer look at the output - **there's no C2 in there.**

Now comes the most interesting part of *updating the centroids* by determining the **mean** values of the coordinates of the data points (which should be belonging to some centroid by now). Hence the name **K-Means**. This is how the mean calculation looks like:

$$\frac{\sum_{i=1}^n x_i}{n}, \frac{\sum_{i=1}^n y_i}{n}$$

Mean update in K-Means (n denotes the number of data points belonging in a cluster)

The following lines of code does this for you:

```
# Calculate the coordinates of the new centroid from cluster 1
x_new_centroid1 = data[data['Cluster']=='C1']['X_value'].mean()
y_new_centroid1 = data[data['Cluster']=='C1']['Y_value'].mean()
```

```
# Calculate the coordinates of the new centroid from cluster 2
x_new_centroid2 = data[data['Cluster']=='C3']['X_value'].mean()
y_new_centroid2 = data[data['Cluster']=='C3']['Y_value'].mean()
```

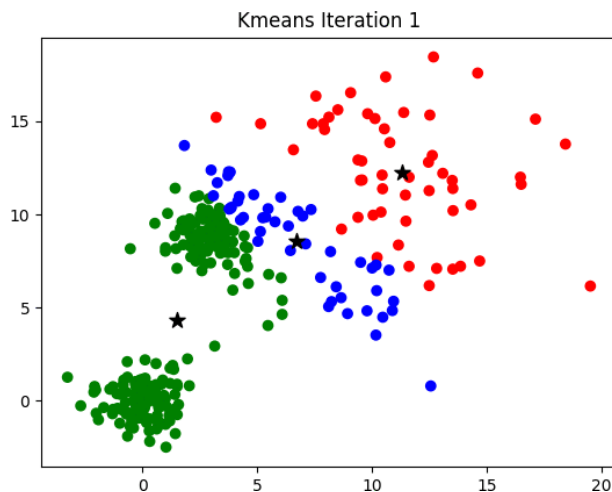
```
# Print the coordinates of the new centroids
print('Centroid 1 ({}, {})'.format(x_new_centroid1, y_new_centroid1))
print('Centroid 2 ({}, {})'.format(x_new_centroid2, y_new_centroid2))
```

You get:

```
Centroid 1 (-0.500753347459331, 3.4851226841526897)
Centroid 2 (1.7821539902873855, 2.2299026522421928)
```

Coordinates of the new centroids

Notice that the algorithm, in its first iteration, grouped the data points into two clusters although we specified this number to be 3. The following animation gives you a pretty good overview of how centroid updates take place in the K- Means algorithm.



Experiment-11

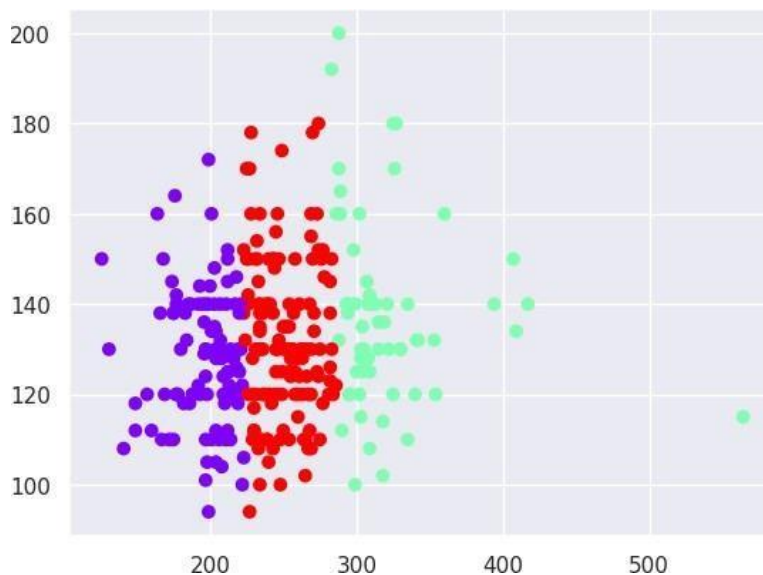
Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
#KMeans
import numpy as np import pandas as pd
import statsmodels.api as sm import matplotlib.pyplot as plt import
seaborn as sns
sns.set()
from sklearn.cluster import KMeans
data = pd.read_csv('/content/sample_data/heart.csv')

x = data.iloc[:,3:5] # 1st for rows and second for columns kmeans =

KMeans(3)
kmeans.fit(x)
identified_clusters = kmeans.fit_predict(x) data_with_clusters =
data.copy()

data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['chol'],data_with_clusters['trestbps'],c=data_
with_clusters['Clusters'],cmap='rainbow')
```





Date :

```
#KMeans
import numpy as np import pandas as pd
import statsmodels.api as sm import matplotlib.pyplot as plt
from sklearn.cluster import KMeans from sklearn import cluster
from sklearn import datasets df = datasets.load_iris()
x = df.data y=df.target print(x) print(y)
kmeans = KMeans(n_clusters=3) y_kmeans=kmeans.fit_predict(df)
print(y_kmeans) print(y_kmeans.cluster_centers_)
```

Output:

```
[[5.1 3.5 1.4 0.2]
[4.9 3. 1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5. 3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5. 3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3. 1.4 0.1]
[4.3 3. 1.1 0.1]
[5.8 4. 1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1. 0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5. 3. 1.6 0.2]
[5. 3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]]
```



Date :

[5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5. 3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3. 1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5. 3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5. 3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3. 1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5. 3.3 1.4 0.2]
 [7. 3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4. 1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1.]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5. 2. 3.5 1.]
 [5.9 3. 4.2 1.5]
 [6. 2.2 4. 1.]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3. 4.5 1.5]
 [5.8 2.7 4.1 1.]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4. 1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3. 4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3. 5. 1.7]
 [6. 2.9 4.5 1.5]
 [5.7 2.6 3.5 1.]
 [5.5 2.4 3.8 1.1]



[5.5 2.4 3.7 1.]
 [5.8 2.7 3.9 1.2]
 [6. 2.7 5.1 1.6]
 [5.4 3. 4.5 1.5]
 [6. 3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3. 4.1 1.3]
 [5.5 2.5 4. 1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3. 4.6 1.4]
 [5.8 2.6 4. 1.2]
 [5. 2.3 3.3 1.]
 [5.6 2.7 4.2 1.3]
 [5.7 3. 4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3. 1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6. 2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3. 5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3. 5.8 2.2]
 [7.6 3. 6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2.]
 [6.4 2.7 5.3 1.9]
 [6.8 3. 5.5 2.1]
 [5.7 2.5 5. 2.]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3. 5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6. 2.2 5. 1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2.]
 [7.7 2.8 6.7 2.]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6. 1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3. 4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3. 5.8 1.6]
 [7.4 2.8 6.1 1.9]

[illegible]



EXPERIMENT -12

XGBoost classifier

XGBoost is a machine learning algorithm that belongs to the ensemble learning category, specifically the gradient boosting framework. It utilizes decision trees as base learners and employs regularization techniques to enhance model generalization. Known for its computational efficiency, feature importance analysis, and handling of missing values, XGBoost is widely used for tasks such as regression, classification, and ranking.

```
#XGBoost classifier import pandas as pd import numpy as np
df = pd.read_csv('/content/sample_data/Pokemon.csv') df.head()
df.drop(columns=['#','Name', 'Type 1', 'Type 2'],inplace=True) df
df.Legendary.value_counts() # Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels. label_encoder =
preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Legendary']= label_encoder.fit_transform(df['Legendary'])

df['Legendary'].unique() df
df.Legendary.value_counts()
from sklearn.model_selection import train_test_split

pokemon_features = df.drop("Legendary",axis=1) target = df["Legendary"]

X_train,X_test,Y_train,Y_test
train_test_split(pokemon_features,target,test_size=0.20,random_state=0) from
sklearn.metrics import accuracy_score
import xgboost as xgb
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)
Y_pred_xgb = xgb_model.predict(X_test)
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)
print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
from keras.models import Sequential
from keras.layers import Dense import tensorflow as tf
model = Sequential()
```

=

Date :

```

model.add(Dense(32,activation='relu',input_dim=8))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X_train,Y_train,epochs=100,callbacks=
tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3))
Y_pred_nn = model.predict(X_test) rounded = [round(x[0]) for x in Y_pred_nn]
Y_pred_nn = rounded
score_nn = round(accuracy_score(Y_pred_nn,Y_test)*100,2)
print("The accuracy score achieved using Neural Network is: "+str(score_nn)+"
%")

```

```

➡ The accuracy score achieved using XGBoost is: 96.25 %
Epoch 1/100
20/20 [=====] - 1s 2ms/step - loss: 17.9147 - accuracy: 0.9219
Epoch 2/100
20/20 [=====] - 0s 2ms/step - loss: 8.8131 - accuracy: 0.9203
Epoch 3/100
20/20 [=====] - 0s 2ms/step - loss: 4.1243 - accuracy: 0.7297
Epoch 4/100
20/20 [=====] - 0s 2ms/step - loss: 3.4813 - accuracy: 0.8438
Epoch 5/100
20/20 [=====] - 0s 2ms/step - loss: 3.1440 - accuracy: 0.7875
Epoch 6/100
20/20 [=====] - 0s 2ms/step - loss: 2.3603 - accuracy: 0.8094
Epoch 7/100
20/20 [=====] - 0s 2ms/step - loss: 1.8960 - accuracy: 0.7922
Epoch 8/100
20/20 [=====] - 0s 2ms/step - loss: 1.7584 - accuracy: 0.7906
Epoch 9/100
20/20 [=====] - 0s 2ms/step - loss: 1.3752 - accuracy: 0.8281
Epoch 10/100
20/20 [=====] - 0s 2ms/step - loss: 1.2229 - accuracy: 0.8203
Epoch 11/100
20/20 [=====] - 0s 2ms/step - loss: 1.0404 - accuracy: 0.8266
Epoch 12/100
20/20 [=====] - 0s 2ms/step - loss: 0.8350 - accuracy: 0.8656
Epoch 13/100

```

Activ
Go to '...