2]synthetic dataset

```
from numpy import where
from collections import Counter
from sklearn.datasets import make_classification
from matplotlib import pyplot
X, y = make_classification(n_samples=1000, n_features=4, n_informative=2,
                           n_redundant=0, n_classes=3, n_clusters_per_class=1, weights=None,
                           random_state=1)
print(X.shape, y.shape)
counter = Counter(y)
print(counter)
for i in range(10):
    print(X[i], y[i])
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

3] find s

```
import pandas as pd

import io

import numpy as np

uploaded = files.upload()

df2 = pd.read_csv(io.BytesIO(uploaded['ws copy.csv']), header=None)

print("\nThe Given Training Dataset:\n")

print(df2)

print("\nThe most general hypothesis: ['?', '?', '?', '?', '?', '?']\n")

print("\nThe most specific hypothesis: ['0', '0', '0', '0', '0', '0']\n")

X = np.array(df2.iloc[:, :-1])

y = np.array(df2.iloc[:, -1])

print("\nX (features):\n", X)

print("\ny (target):\n", y)

m, n = X.shape

print("\nShape of X (m, n):", m, n)

print("\nThe initial value of hypothesis:")

hypothesis = ['0'] * (n - 1)

print(hypothesis)

print("\nFind S: Finding a Maximally Specific Hypothesis\n")

for i in range(m):

    if y[i] == 'Yes':

        for j in range(n - 1):

            if X[i][j] != hypothesis[j]:

                if hypothesis[j] == '0':

                    hypothesis = list(X[i, :n - 1])

                else:

                    hypothesis[j] = '?'

        print("The hypothesis {0} for training set {0}: ".format(i + 1), hypothesis)

print("\nThe Maximally Specific Hypothesis for the given Training Examples:\n")

print(hypothesis)
```

4] candidate elimination

```
import pandas as pd

import numpy as np

df2 = pd.read_csv('/content/ws.csv', header=None)

print(df2)

X = np.array(df2.iloc[:, :-1])

y = np.array(df2.iloc[:, -1])

print("\nX:")

print(X)

print("\ny:")

print(y)

m, n = X.shape

print("\nShape of X (m, n):", m, n)

print("\nMost specific hypothesis: ['0'] *", n)

print("Most general hypothesis: ['?'] *", n)

def candidate_elimination(X, y):

    print("\nInitialization of specific_h and general_h")

    specific_h = ['0'] * n

    print("Specific_h:", specific_h)

    general_h = ['?'] * n

    print("General_h:", general_h, "\n")

    for i in range(m):

        if y[i] == "Yes":

            for j in range(n):

                if specific_h[j] != X[i][j]:

                    if specific_h[j] == '0':

                        specific_h[j] = X[i][j]

                    else:

                        specific_h[j] = '?'

            print("Specific_h updated:", specific_h)

        if y[i] == "No":

            print("-ve training example:", X[i], "\n")
```

```
            new_general_h = []
            for gen_h in general_h:
                for j in range(n):
                    if X[i][j] != specific_h[j] and specific_h[j] != '?':
                        if gen_h[j] == '?':
                            new_gen = list(gen_h)
                            new_gen[j] = specific_h[j]
                            new_general_h.append(new_gen)
                        else:
                            new_general_h.append(gen_h)
            general_h = new_general_h
            print("General_h updated:", general_h)
    return specific_h, general_h
s_final, g_final = candidate_elimination(X, y)
print("\nFinal Specific_h:")
print(s_final)
print("\nFinal General_h:")
print(g_final)
```

5] ID3

```
import pandas as pd

import numpy as np

dataset = pd.read_csv("/content/drive/MyDrive/id3dataset.csv - Sheet1.csv",

                      names=['age', 'income', 'student', 'credit_rating', 'buys_computer'])

def entropy(target_col):

    elements, counts = np.unique(target_col, return_counts=True)

    entropy = np.sum([(-counts[i]/np.sum(counts)) * np.log2(counts[i]/np.sum(counts))

                      for i in range(len(elements))])

    return entropy

def InfoGain(data, split_attribute_name, target_name="buys_computer"):

    total_entropy = entropy(data[target_name])

    vals, counts = np.unique(data[split_attribute_name], return_counts=True)

    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts)) *
entropy(data.where(data[split_attribute_name] == vals[i]).dropna()[target_name])

                               for i in range(len(vals))])

    Information_Gain = total_entropy - Weighted_Entropy

    return Information_Gain

def ID3(data, originaldata, features, target_attribute_name="buys_computer", parent_node_class=None):

    if len(np.unique(data[target_attribute_name])) <= 1:

        return np.unique(data[target_attribute_name])[0]

    elif len(data) == 0:

        return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name
], return_counts=True)[1])]

    elif len(features) == 0:

        return parent_node_class

    else:

        parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],
return_counts=True)[1])]

        item_values = [InfoGain(data, feature, target_attribute_name) for feature in features]

        best_feature_index = np.argmax(item_values)
```

```
        best_feature = features[best_feature_index]
        tree = {best_feature: {}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = ID3(sub_data, dataset, features, target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree
        return tree
tree = ID3(dataset, dataset, dataset.columns[:-1])
print('\nDisplay Tree:\n', tree)
```

6] ANN ARTIFICIAL NN

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

expected_output = np.array([[0], [1], [1], [0]])

epochs = 10000

lr = 0.1

inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2, 2, 1

hidden_weights = np.random.uniform(size=(inputLayerNeurons, hiddenLayerNeurons))

hidden_bias = np.random.uniform(size=(1, hiddenLayerNeurons))

output_weights = np.random.uniform(size=(hiddenLayerNeurons, outputLayerNeurons))

output_bias = np.random.uniform(size=(1, outputLayerNeurons))

print("Initial hidden weights:")

print(hidden_weights)

print("Initial hidden biases:")

print(hidden_bias)

print("Initial output weights:")

print(output_weights)

print("Initial output biases:")

print(output_bias)

for _ in range(epochs):

    hidden_layer_activation = np.dot(inputs, hidden_weights)

    hidden_layer_activation += hidden_bias

    hidden_layer_output = sigmoid(hidden_layer_activation)

    output_layer_activation = np.dot(hidden_layer_output, output_weights)

    output_layer_activation += output_bias

    predicted_output = sigmoid(output_layer_activation)

    error = expected_output - predicted_output

    d_predicted_output = error * sigmoid_derivative(predicted_output)
```

```
    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
    output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
    output_bias += np.sum(d_predicted_output, axis=0, keepdims=True) * lr
    hidden_weights += inputs.T.dot(d_hidden_layer) * lr
    hidden_bias += np.sum(d_hidden_layer, axis=0, keepdims=True) * lr
print("\nFinal hidden weights:")
print(hidden_weights)
print("Final hidden bias:")
print(hidden_bias)
print("Final output weights:")
print(output_weights)
print("Final output bias:")
print(output_bias)
print("\nOutput from neural network after 10,000 epochs:")
print(predicted_output)
```

7] KNN

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
iris = load_iris()
print("Feature Names:", iris.feature_names)
print("Iris Data:", iris.data)
print("Target Names:", iris.target_names)
print("Target:", iris.target)
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.25, random_state=42)
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
print('\nClassification Report:')
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print('\nAccuracy:', accuracy)
print('Misclassification Rate:', 1 - accuracy)
print("\nPredicted Data:")
print(clf.predict(X_test))
print("Actual Test Data Labels:")
print(y_test)
diff = y_pred - y_test
print("\nResult of Misclassifications:")
print(diff)
print('Total number of misclassified samples =', sum(abs(diff)))
```

8] NAÏVE BAYES

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB

data = pd.read_csv("ws.csv")

print("the first 5 values of data is:\n",data.head())

X=data.iloc[:,:-1]

print("\n the first 5 values of train data is\n",X.head())

y=data.iloc[:,-1]

print("\n the first 5 values of train output is \n",y.head())

le_outlook=LabelEncoder()

X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()

X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()

X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()

X.Windy = le_Windy.fit_transform(X.Windy)

print("\n now the train data is:\n",X.head())

le_PlayTennis = LabelEncoder()

y = le_PlayTennis.fit_transform(y)

print("\n now the train output is \n",y)

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

classifier = GaussianNB()

classifier.fit(X_train, y_train)

print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

9] REGRESSION

```
from math import ceil

import numpy as np

from scipy import linalg

def lowess(x, y, f, iterations):

    n = len(x)

    r = int(ceil(f * n))

    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)

    w = (1 - w ** 3) ** 3

    yest = np.zeros(n)

    delta = np.ones(n)

    for iteration in range(iterations):

        for i in range(n):

            weights = delta * w[:, i]

            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])

            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x), np.sum(weights * x *
x)]])

            beta = linalg.solve(A, b)

            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest

        s = np.median(np.abs(residuals))

        delta = np.clip(residuals / (6.0 * s), -1, 1)

        delta = (1 - delta ** 2) ** 2

    return yest

import math

n = 100

x = np.linspace(0, 2 * math.pi, n)

y = np.sin(x) + 0.3 * np.random.randn(n)

f =0.25

iterations=3

yest = lowess(x, y, f, iterations)
```

```
import matplotlib.pyplot as plt

plt.plot(x,y,"r.")

plt.plot(x,yest,"b-")
```

10] EM ALG

```
import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

iris = datasets.load_iris()

X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])

y = pd.DataFrame(iris.target, columns=['Targets'])

model = KMeans(n_clusters=3)

model.fit(X)

plt.figure(figsize=(14, 7))

colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1, 3, 1)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.subplot(1, 3, 2)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns=X.columns)

gmm = GaussianMixture(n_components=3)

gmm.fit(xs)
```

```
plt.subplot(1, 3, 3)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm.predict(xs)], s=40)

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```