np.zeros(5)

.ones(5) or

.arrange(1, 10, 2, dtype = np.float64/np.int32)

.array([list]) -> convert list to np array

.reshape(2, 2) -> [2x2 matrix]

.identity(4)

.linspace(0, 10, 5)

.random.rand()

.random.randint(0, 400, size = (20, 20)) -> 0 to 400 => 20x20 matrix

.equal(temp1, temp2)

.flip(temp) -> reverse array

.min(temp)

.max(temp)

.mean(temp)

.concatenate((temp1, temp2), axis = 1)

.array\_split(temp1, 3, axis = 1) => will split and adjust at the end for unequal

.hsplit(temp1, 3) => same as axis = 1

.where(<condition>) => temp == 4 or temp%2 == 0

.sort(temp)

For i in np.nditer

temp.dtype

temp.copy()

temp.view() => changes original

temp.shape

Array with border 1’s:

Temp = np.ones((20, 20))

Temp[1:-1, 1:-1] = 0

Create an array of size 10X10 with 10 element valued as nan

temp1 = np.random.randint(0, 10, size = (10))

temp2 = np.random.randint(0, 10, size = (10))

temp = np.empty((10, 10))

print(temp)

temp[temp1, temp2] = np.nan

Temp

Pandas

Temp = pd.Series([list]) or pd.Series({“Name”: “John”, “Age”: “19”, “Gender”: “Male”})

Temp = pd.Series(np.random.normal(size = 7))

Temp.index

Temp.values

Temp = pd.Series([list1], index = [list2])

Temp.shape

Temp.size

data = {

"calories": [420, 380, 390],

"duration": [50, 40, 45]

}

#load data into a DataFrame object:

df = pd.DataFrame(data)

print(df)

calories duration

0 420 50

1 380 40

2 390 45

Df = pd.read\_csv(<dir>)

Df.loc[0] => row #0

Df.index

Df.columns

df.head(4) or df.tail(4)

Df = DataFrame(Data, columns = [list1])

Df[“Name”][0] => item

df[df.duplicated()]

df.isnull().sum()

df[“Name”] = df[“Name”].replace(np.NaN, df[“Name”].mean())

df.dropna(inplace = True)

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

dataset = pd.read\_csv('/content/sample\_data/500\_Person\_Gender\_Height\_Weight\_Index.csv')

dataset.head(10)

dataset = dataset.drop(['Gender', 'Index'], axis=1)

dataset['Height'] = dataset['Height'].multiply(0.0328084)

dataset['Weight'] = dataset['Weight'].multiply(1000)

dataset.head(10)

scaler = MinMaxScaler(feature\_range = (-10, 10))

rescaleDataset = scaler.fit\_transform(dataset)

np.set\_printoptions(precision = 6)

print(rescaleDataset)

scaler = StandardScaler().fit(dataset)

rescaleDataset = scaler.transform(dataset)

np.set\_printoptions(precision = 3)

print(rescaleDataset)

**CSV FILE:**

import csv

# fields = ['day', 'outlook', 'temp', 'humidity', 'wind', 'play']

# rows = [['D1', 'Sunny', 'Hot', 'High', 'Weak', 'No'],

# ['D2', 'Sunny', 'Hot', 'High', 'Strong', 'No'],

# ['D3', 'Overcast', 'Hot', 'High', 'Weak', 'Yes'],

# ['D4', 'Rain', 'Mild', 'High', 'Weak', 'Yes'],

# ['D5', 'Rain', 'Cool', 'Normal', 'Weak', 'Yes'],

# ['D6', 'Rain', 'Cool', 'Normal', 'Strong', 'No'],

# ['D7', 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes'],

# ['D8', 'Sunny', 'Mild', 'High', 'Weak', 'No'],

# ['D9', 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes'],

# ['D10', 'Rain', 'Mild', 'Normal', 'Weak', 'Yes'],

# ['D11', 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes'],

# ['D12', 'Overcast', 'Mild', 'High', 'Strong', 'Yes'],

# ['D13', 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes'],

# ['D14', 'Rain', 'Mild', 'High', 'Strong', 'No']]

# fields = ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']

rows = [['humidity','wind','play'],

['high','strong','no'],

['high','weak','yes'],

['normal','strong','yes'],

['normal','weak','yes']]

with open('/content/sample\_data/temp1.csv', 'w') as f:

csv\_writer = csv.writer(f)

#csv\_writer.writerow(fields)

csv\_writer.writerows(rows)

**FIND-S:**

#Question 1

import csv

num\_attributes = 5

a = []

print("\n The Given Training Data Set \n")

with open('/content/sample\_data/temp1.csv', 'r') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

a.append (row)

print(row)

print("\n The initial value of hypothesis: ")

hypothesis = ['0'] \* num\_attributes

print(hypothesis)

for j in range(0,num\_attributes):

hypothesis[j] = a[1][j]

print("\n The a[1] value of hypothesis: ")

print(hypothesis)

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

for i in range(0,len(a)):

if a[i][num\_attributes]=='Yes':

for j in range(0,num\_attributes):

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

hypothesis[j]='?'

else :

hypothesis[j]= a[i][j]

print(" For Training instance No:{} the hypothesis is ".format(i), hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")

print(hypothesis)

**CANDIDATE ELIMINATION ALGO:**

import csv

# Initialize the version space with the most specific and most general hypotheses

specific\_h = {'humidity': 'high', 'wind': 'strong', 'play': 'no'}

general\_h = {'humidity': '?', 'wind': '?', 'play': '?'}

# Read the data from a CSV file

examples = []

with open('/content/sample\_data/temp3.csv', 'r') as file:

reader = csv.DictReader(file)

for row in reader:

examples.append(row)

#printing dataset

print("Training Dataset:")

for example in examples:

print(example)

print()

# Iterate over each example

for example in examples:

print("Example:", example)

print("Specific hypothesis before update:", specific\_h)

print("General hypothesis before update:", general\_h)

# Check if the example is consistent with the specific hypothesis

if all(specific\_h[attr] == example[attr] for attr in specific\_h):

print("Example is consistent with specific hypothesis, no update needed")

else:

# Update the general hypothesis

general\_h = {attr: example[attr] if specific\_h[attr] != example[attr] else general\_h[attr] for attr in general\_h}

# Update the specific hypothesis

specific\_h = {attr: value for attr, value in specific\_h.items() if general\_h[attr] == value}

print("Specific hypothesis after update:", specific\_h)

print("General hypothesis after update:", general\_h)

print()

print("Final General hypothesis:", general\_h)

print("Final Specific hypothesis:", specific\_h)

**LINEAR REGRESSION:**

import numpy as np

import matplotlib.pyplot as plt

# Define the x and y points as numpy arrays

x = np.array([160, 280, 180, 200, 260, 240, 220, 170])

y = np.array([125, 120, 104, 85, 40, 80, 75, 79])

# Calculate the slope and y-intercept of the least squares regression line

m, b = np.polyfit(x, y, 1)

# Plot the points and the least squares regression line

plt.scatter(x, y)

plt.plot(x, m\*x + b, color='red')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Least Squares Regression Line')

plt.show()

**ID3:**

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("/content/sample\_data/3-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

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))

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

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

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)

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)

root = ID3(data, features)

print("Decision Tree is:")

printTree(root)

print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}

classify (root, new)