1) Write a program to implement S-algorithm to find the general hypothesis

```
table = [
['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']
]

hypo = ["phi" for i in range (len(table[0])-1)]

for i in table:
  if(i==table[0]):
    hypo=table[0]
  if(i[-1]=='yes'):
    for j in range (len(i)-1):
      if(i[j]!=hypo[j]):
        hypo[j]='?'
print(hypo)
```

```
['sunny', 'warm', '?', 'strong', '?', '?', 'yes']
```

2) Write a program to implement candidate elimination algorithm

```
table = [
['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']
]

x=[]
y=[]

for i in table:
  x.append(i[:len(table[0])-1])
  y.append(i[-1])
print("concept : ",x)
print("target : ",y,end="\n\n")


gen_hypo=[["?" for i in range (len(x[0]))] for i in range (len(x[0]))]
print("most general hypothesis : ",gen_hypo)
```

```
spe_hypo=["phi" for i in range (len(x[0]))]
print("most specific hypothesis : ",spe_hypo,end="\n\n")

spe_hypo=x[0]

for i,h in enumerate(x):
  if(y[i]=='yes'):
    for j in range (len(h)):
      if(spe_hypo[j]!=h[j]):
        spe_hypo[j]='?'
        gen_hypo[j][j]='?'

  else:
    for j in range (len(h)):
      if(spe_hypo[j]!=h[j]):
        gen_hypo[j][j]=spe_hypo[j]
      else:
        gen_hypo[j][j]='?'

i=0
while (i < (len(gen_hypo))):
  if(gen_hypo[i]==['?' for j in range (len(table[0])-1)]):
    gen_hypo.pop(i)
  else:
    i+=1

print("final specific boundry  : ",spe_hypo)
print("final general  : ",gen_hypo)
```

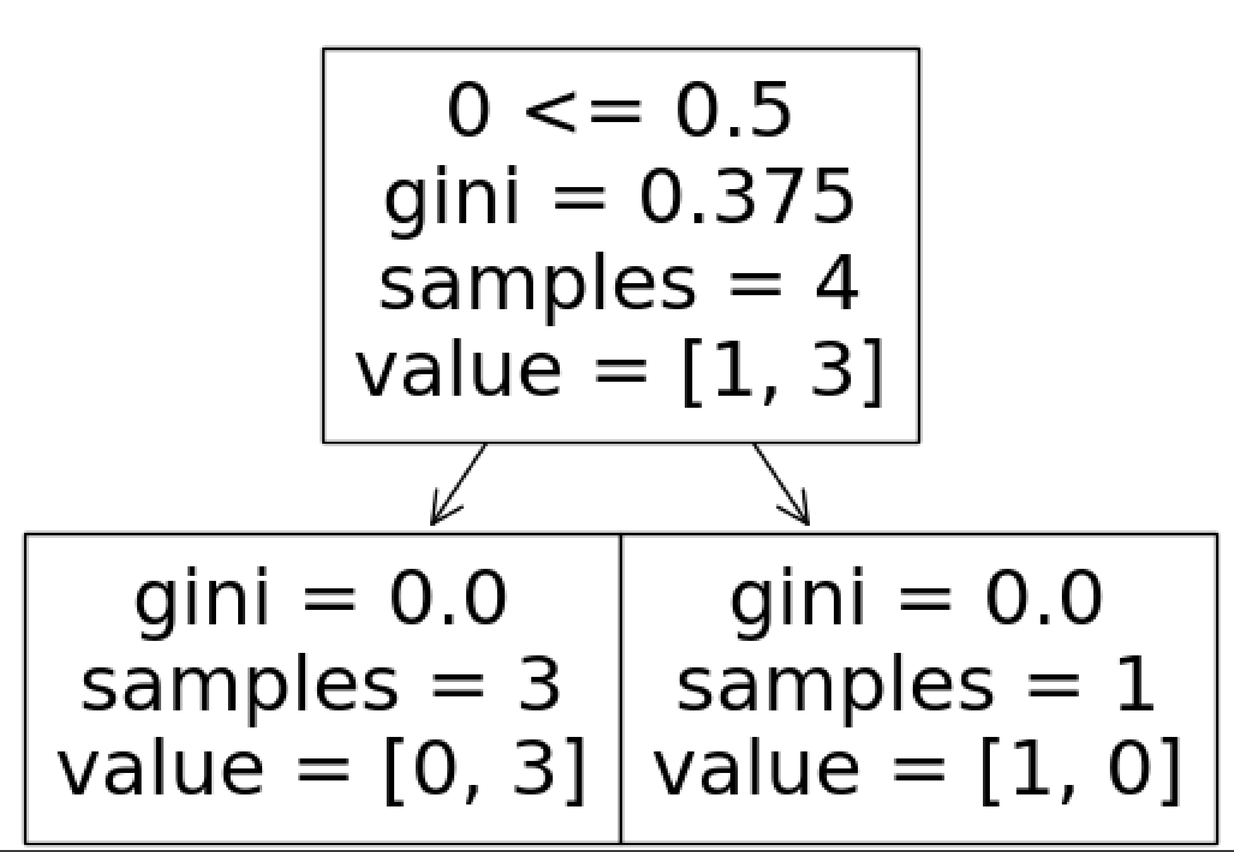
```
concept :  [['sunny', 'warm', 'normal', 'strong', 'warm', 'same'], ['sunny',
target :  ['yes', 'yes', 'no', 'yes']

most general hypothesis :  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '
most specific hypothesis :  ['phi', 'phi', 'phi', 'phi', 'phi', 'phi']

final specific boundry  :  ['sunny', 'warm', '?', 'strong', '?', '?']
final general  :  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?'
```

3) Implement decision-tree algorithm

```
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

```
import matplotlib.pyplot as plt


table = [
['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']
]

df = pd.DataFrame(table, columns =[i for i in range(len(table[0]))])

dd = {'sunny':0,'rainy':1}
df[0]=df[0].map(dd)

dd = {'warm':0,'cold':1}
df[1]=df[1].map(dd)

dd = {'normal':0,'high':1}
df[2]=df[2].map(dd)

dd = {'strong':0}
df[3]=df[3].map(dd)

dd = {'warm':0,'cool':1}
df[4]=df[4].map(dd)

dd = {'same':0,'change':1}
df[5]=df[5].map(dd)

dd = {'yes':1,'no':0}
df[6]=df[6].map(dd)

features=[i for i in range(len(table[0])-1)]

X = df[features]
y = df[6]

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)
```

```
[Text(0.5, 0.75, '0 <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]')]
```



4) The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

```
def bayes (AandB,B):
  return AandB/B

#P(Friday ^ Absent)=3%
PofFandAb = 0.03

#P(Friday)=20%
PofF=0.2

#P(Absent/Friday)=?

#according to baye's rule P(A/B)=p(A^B)/p(B)

# => P(Absent/Friday)=P(Friday ^ Absent)/P(Friday)

PofAbgivenF = bayes(PofFandAb,PofF)

print(PofAbgivenF)
```

```
0.15
```

5) Extract the data from database using python

```
with open("/content/enj.csv","r") as file:
  lines=file.readlines()

data=[]
for line in lines:
  data.append(line.strip().split(","))

for row in data[1:]:
  print(row)
```

```
['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']
```

6) Implement k-nearest neighbors classification using python

```
class knn:
  def __init__(self,k):
    self.k=k

  def fit(self,x,y):
    self.x=x
    self.y=y

  def predict(self,x_test):
    pred=[]
    for i in x_test:
      distances=[]
      for j in self.x:
        temp=[]
        temp.append(self.x.index(j))
        dist= ((j[0]-i[0])**2+(j[1]-i[1])**2)**0.5
        temp.append(dist)
        distances.append(temp)
      distances.sort(key=lambda x : x[1])

      c0=0
      c1=0
      for i in range (self.k):
        if(self.y[distances[i][0]]==0):
          c0+=1
```

```
          c0+=1
        else:
          c1+=1
      if(c0>c1):
        pred.append(0)
      else:
        pred.append(1)
    return pred

x_train = [[1,1],[3,4],[0,0],[5,5],[7,8]]
y_train = [0,1,0,1,1]

x_test=[[1,2],[6,6],[0,1],[8,7]]

knn=knn(3)
knn.fit(x_train,y_train)

print(knn.predict(x_test))
```

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

7) Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606,using the result of k-means clustering with 3 means (i.e., 3 centroids)

- VAR1 VAR2 CLASS
- 1.713 1.586 0
- 0.180 1.786 1
- 0.353 1.240 1
- 0.940 1.566 0
- 1.486 0.759 1
- 1.266 1.106 0
- 1.540 0.419 1
- 0.459 1.799 1
- 0.773 0.186 1

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

class kmeans:

  def dist(self,a,b):
    return (((a[0]-b[0])**2+(a[1]-b[1])**2)**0.5)
```

```
def __init__(self,k):
  self.k=k

def fit(self,x,y):
  self.x=x
  self.y=y
  self.tp_cent=x[:self.k]
  self.gps=[[x[i]] for i in range (3)]

def assigner(self,p):
  for i in p:
    min=float('inf')
    minidx=0
    for j in self.tp_cent:
      if(min>self.dist(i,j)):
        min=self.dist(i,j)
        minidx=self.x.index(j)
    self.gps[minidx].append(i)

def new_centroids(self):
  for i in self.gps:
    ncent=[[]]
    x_co=0
    y_co=0
    for j in range(1,len(i)):
      x_co+=i[j][0]
      y_co+=i[j][1]
    ncent[0].append(x_co/(len(i)-1))
    ncent[0].append(y_co/(len(i)-1))
    self.gps[self.gps.index(i)]=ncent


def predict(self,x_test):
  prev_gps=[]
  while(prev_gps!=self.gps):
    prev_gps=copy.deepcopy(self.gps)
    self.assigner(self.x)
    self.new_centroids()

  self.assigner(x_test+self.x)
  c0=0
  c1=0
  print(self.gps[0][0])
  print(self.gps[1][0])
  print(self.gps[2][0])
  for i in self.gps:
    if i.count(x_test[0])!=0:
      for j in range (2,len(i)):
        if(self.y[self.x.index(i[j])]==0):
```

```
            c0+=1
          else:
            c1+=1
      # i=np.array(i)
      # plt.scatter(i[:,0],i[:,1])
    if(c0>c1):
      return 0
    else:
      return 1

data = [
[1.713, 1.586, 0],
[0.180, 1.786, 1],
[0.353, 1.240, 1],
[0.940, 1.566, 0],
[1.486, 0.759, 1],
[1.266, 1.106, 0],
[1.540, 0.419, 1],
[0.459, 1.799, 1],
[0.773, 0.186, 1]
]

x_train=[]
y_train=[]

for i in data:
  x_train.append(i[:2])
  y_train.append(i[-1])

kmeans=kmeans(3)
kmeans.fit(x_train,y_train)
x_test=[[0.906,0.606]]

print(kmeans.predict(x_test))

# x_train= np.array(x_train)

# plt.scatter(0.906,0.606,color="g")
# [1.407, 1.152],
#     [0.394, 1.590],
#     [1.275, 0.454]
```

```
[1.50125, 0.9675000000000001]
[0.3195, 1.7925]
[0.6886666666666666, 0.9973333333333333]
1
```

```
##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gp
##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gp
##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gpt##shortcut gp

import math

# Function to calculate Euclidean distance between two points
def euclidean_distance(a, b):
    return (((a[0]-b[0])**2+(a[1]-b[1])**2)**0.5)

# Function to predict the classification for a given case
def predict_classification(data, centroids, new_case):
    min_distance = float('inf')
    predicted_class = None

    for i, centroid in enumerate(centroids):
        distance = euclidean_distance(new_case, centroid)
        if distance < min_distance:
            min_distance = distance
            predicted_class = data[i][2]  # Retrieve the class label from the data

    return predicted_class

# Given data
data = [
    [1.713, 1.586, 0],
    [0.180, 1.786, 1],
    [0.353, 1.240, 1],
    [0.940, 1.566, 0],
    [1.486, 0.759, 1],
    [1.266, 1.106, 0],
    [1.540, 0.419, 1],
    [0.459, 1.799, 1],
    [0.773, 0.186, 1]
]

# Centroids obtained from k-means clustering (3 means)
centroids = [
    [1.407, 1.152],
    [0.394, 1.590],
    [1.275, 0.454]
]

# New case to predict
new_case = [0.906, 0.606]

# Predict the classification for the new case
predicted_classification = predict_classification(data, centroids, new_case)
print("Predicted classification:", predicted_classification)
```

```
Predicted classification: 1
```

8) The following training examples map descriptions of individuals onto high, medium and low credit-worthiness. medium skiing design single twenties no -> highRisk high golf trading married forties yes -> lowRisk low speedway transport married thirties yes -> medRisk medium football banking single thirties yes -> lowRisk high flying media married fifties yes -> highRisk low football security single twenties no -> medRisk medium golf media single thirties yes -> medRisk medium golf transport married forties yes -> lowRisk high skiing banking single thirties yes -> highRisk low golf unemployed married forties yes -> highRisk Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the unconditional probability of `golf' and the conditional probability of`single' given `medRisk' in the dataset?

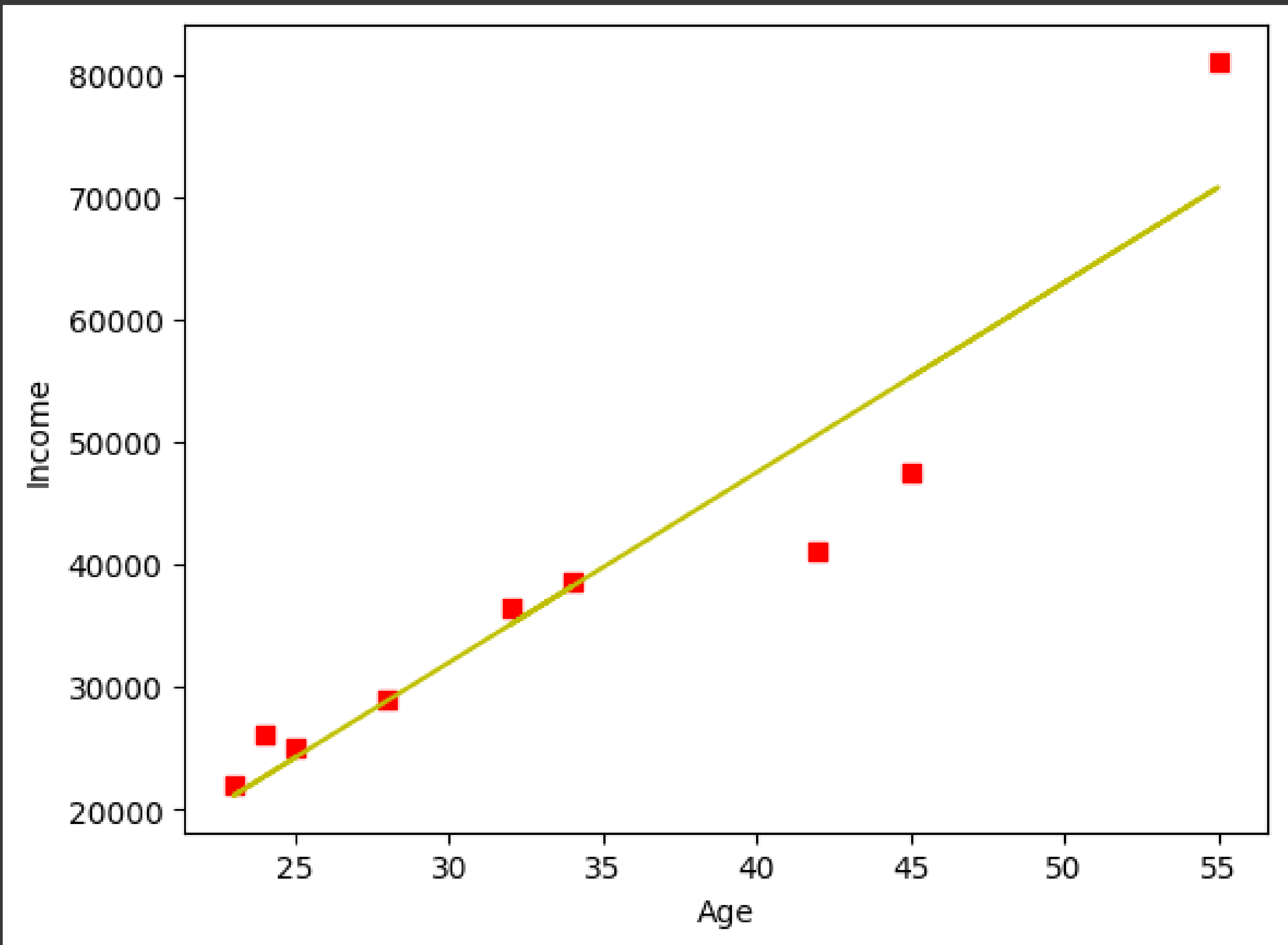
```
train_data = [
['medium', 'skiing', 'design', 'single', 'twenties', 'no', 'highRisk'],
['high', 'golf', 'trading', 'married', 'forties', 'yes', 'lowRisk'],
['low', 'speedway', 'transport', 'married', 'thirties', 'yes', 'medRisk'],
['medium', 'football', 'banking', 'single', 'thirties', 'yes', 'lowRisk'],
['high', 'flying', 'media', 'married', 'fifties', 'yes', 'highRisk'],
['low', 'football', 'security', 'single', 'twenties', 'no', 'medRisk'],
['medium', 'golf', 'media', 'single', 'thirties', 'yes', 'medRisk'],
['medium', 'golf', 'transport', 'married', 'forties', 'yes', 'lowRisk'],
['high', 'skiing', 'banking', 'single', 'thirties', 'yes', 'highRisk'],
['low', 'golf', 'unemployed', 'married', 'forties', 'yes', 'highRisk']
]

size_data=len(train_data)

nGolf = sum(1 for ex in train_data if ex[1]=='golf')


print("unconditional probability of golf : ")
print(nGolf/size_data)

nSingleMed = sum(1 for ex in train_data if ex[3]=='single' and ex[-1]=='medRisk')


nMed = sum(1 for ex in train_data if ex[-1]=='medRisk')


print("conditional probability ofsingle given medRisk")
print(nSingleMed/nMed)
```

```
unconditional probability of golf :
0.4
conditional probability ofsingle given medRisk
0.6666666666666666
```

9) Implement linear regression using python.

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

training_data=[
   [25,25000],
   [23,22000],
   [24,26000],
   [28,29000],
   [34,38600],
   [32,36500],
   [42,41000],
   [55,81000],
   [45,47500]
]

size = len(training_data)

training_data=np.array(training_data)

age=training_data[:,0]
income=training_data[:,1]

mean_age=np.mean(age)
mean_income=np.mean(income)

cd_ageincome=np.sum(age*income)-size*mean_age*mean_income
cd_ageage=np.sum(age*age)-size*mean_age*mean_age

b1=cd_ageincome/cd_ageage
b0=mean_income-b1*mean_age

plt.scatter(age,income,color="r",marker="s")

response_vec = b0+b1*age

plt.plot(age,response_vec,color="y")

plt.xlabel('Age')
plt.ylabel('Income')
```

```
plt.show()
```



10) Implement Naïve Bayes theorem to classify the English text

```
training_data = [
["I love this car", "positive"],
["This view is amazing", "positive"],
["I feel great", "positive"],
["I'm not happy with the product", "negative"],
["This is a terrible place", "negative"],
["I don't like this movie", "negative"],
["I hate things","negative"]
]

vocabulary=set()

for data in training_data:
  sentence=data[0]
  words=sentence.split()
  vocabulary.update(words)

class_probabilities={}
```

```
l1=0
l2=0
total_data=len(training_data)
for data in training_data:
  if(data[1]=="positive"):
    l1+=1
  else:
    l2+=1
class_probabilities["positive"]=l1/total_data
class_probabilities["negative"]=l2/total_data

word_counts={}
for data in training_data:
  sentence=data[0]
  words=sentence.split()
  label=data[1]
  if(label not in word_counts):
    word_counts[label]={}
  for word in words:
    if word in word_counts[label]:
      word_counts[label][word]+=1
    else:
      word_counts[label][word]=1

word_probabilities={}
for label in word_counts:
  word_probabilities[label]={}
  total_words=sum(word_counts[label].values())
  for word in vocabulary:
    if word in word_counts[label]:
      word_probabilities[label][word]=word_counts[label][word]/total_words
    else:
      word_probabilities[label][word]=0.0

def predict(text):
  text_probability={}
  words=text.split()
  for label in class_probabilities:
    score=0
    text_probability[label]=class_probabilities[label]
    for word in words:
      if word in vocabulary:
        score+=word_probabilities[label][word]
    text_probability[label]*=score
  if(text_probability["positive"]>text_probability["negative"]):
      return "positive"
  else:
    return "negative"
```

```
print(predict("I hate cars"))
```

```
negative
```

0s completed at 8:47 PM