**TMW1: Find S Algorithm**

**#implementation of Find S algorithm**

**import** csv  
a = []  
**with** open(**'enjoysport.csv'**, **'r'**) **as** csvfile:  
 next(csvfile)  
 **for** row **in** csv.reader(csvfile):  
 a.append(row)  
 print(a)  
  
print(**"\nThe total number of training instances are : "**,len(a))  
  
num\_attribute = len(a[0])-1  
  
print(**"\nThe initial hypothesis is : "**)  
hypothesis = [**'0'**]\*num\_attribute  
print(hypothesis)  
  
**for** i **in** range(0, len(a)):  
 **if** a[i][num\_attribute] == **'yes'**:  
 print (**"\nInstance "**, i+1, **"is"**, a[i], **" and is Positive Instance"**)  
 **for** j **in** range(0, num\_attribute):  
 **if** hypothesis[j] == **'0' or** hypothesis[j] == a[i][j]:  
 hypothesis[j] = a[i][j]  
 **else**:  
 hypothesis[j] = **'?'** print(**"The hypothesis for the training instance"**, i+1, **" is: "** , hypothesis, **"\n"**)  
  
 **if** a[i][num\_attribute] == **'no'**:  
 print (**"\nInstance "**, i+1, **"is"**, a[i], **" and is Negative Instance Hence Ignored"**)  
 print(**"The hypothesis for the training instance"**, i+1, **" is: "** , hypothesis, **"\n"**)  
  
print(**"\nThe Maximally specific hypothesis for the training instance is "**, hypothesis)

**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']]

The total number of training instances are : 4

The initial hypothesis is :

['0', '0', '0', '0', '0', '0']

Instance 1 is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'yes'] and is Positive Instance

The hypothesis for the training instance 1 is: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Instance 2 is ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'yes'] and is Positive Instance

The hypothesis for the training instance 2 is: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

Instance 3 is ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'no'] and is Negative Instance Hence Ignored

The hypothesis for the training instance 3 is: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

Instance 4 is ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'yes'] and is Positive Instance

The hypothesis for the training instance 4 is: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

The Maximally specific hypothesis for the training instance is ['Sunny', 'Warm', '?', 'Strong', '?', '?']

**TMW2: House Price Prediction using Linear Regression**

*#Implementation of Linear Regression*  
import sys

import subprocess

subprocess.check\_call([sys.executable,'-m','pip','install','sklearn'])

import pandas as pd

import numpy as np

from sklearn import linear\_model

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_boston

boston = load\_boston()

#print(boston)

df\_x = pd.DataFrame(boston.data,columns = boston.feature\_names)

df\_y = pd.DataFrame(boston.target)

df\_x.describe()

reg = linear\_model.LinearRegression()

x\_train,x\_test, y\_train, y\_test = train\_test\_split(df\_x , df\_y,test\_size=0.33, random\_state =42)

reg.fit(x\_train,y\_train)

print(reg.coef\_)

y\_pred = reg.predict(x\_test)

print(y\_pred)

y\_pred[2]

y\_test[0]

print(np.mean((y\_pred-y\_test)\*\*2))

from sklearn.metrics import mean\_squared\_error

print(mean\_squared\_error(y\_test,y\_pred))

**OUTPUT:**

[[-1.28749718e-01 3.78232228e-02 5.82109233e-02 3.23866812e+00

-1.61698120e+01 3.90205116e+00 -1.28507825e-02 -1.42222430e+00

2.34853915e-01 -8.21331947e-03 -9.28722459e-01 1.17695921e-02

-5.47566338e-01]]

[[28.53469469]

[36.6187006 ]

[15.63751079]

[25.5014496 ]

[18.7096734 ]

[23.16471591]

[17.31011035]

[14.07736367]

[23.01064388]

[20.54223482]

[24.91632351]

[18.41098052]

[-6.52079687]

[21.83372604]

[19.14903064]

[26.0587322 ]

[20.30232625]

[ 5.74943567]

[40.33137811]

[17.45791446]

[27.47486665]

[30.2170757 ]

[10.80555625]

[23.87721728]

[17.99492211]

[16.02608791]

[23.268288 ]

[14.36825207]

[22.38116971]

[19.3092068 ]

[22.17284576]

[25.05925441]

[25.13780726]

[18.46730198]

[16.60405712]

[17.46564046]

[30.71367733]

[20.05106788]

[23.9897768 ]

[24.94322408]

[13.97945355]

[31.64706967]

[42.48057206]

[17.70042814]

[26.92507869]

[17.15897719]

[13.68918087]

[26.14924245]

[20.2782306 ]

[29.99003492]

[21.21260347]

[34.03649185]

[15.41837553]

[25.95781061]

[39.13897274]

[22.96118424]

[18.80310558]

[33.07865362]

[24.74384155]

[12.83640958]

[22.41963398]

[30.64804979]

[31.59567111]

[16.34088197]

[20.9504304 ]

[16.70145875]

[20.23215646]

[26.1437865 ]

[31.12160889]

[11.89762768]

[20.45432404]

[27.48356359]

[10.89034224]

[16.77707214]

[24.02593714]

[ 5.44691807]

[21.35152331]

[41.27267175]

[18.13447647]

[ 9.8012101 ]

[21.24024342]

[13.02644969]

[21.80198374]

[ 9.48201752]

[22.99183857]

[31.90465631]

[18.95594718]

[25.48515032]

[29.49687019]

[20.07282539]

[25.5616062 ]

[ 5.59584382]

[20.18410904]

[15.08773299]

[14.34562117]

[20.85155407]

[24.80149389]

[-0.19785401]

[13.57649004]

[15.64401679]

[22.03765773]

[24.70314482]

[10.86409112]

[19.60231067]

[23.73429161]

[12.08082177]

[18.40997903]

[25.4366158 ]

[20.76506636]

[24.68588237]

[ 7.4995836 ]

[18.93015665]

[21.70801764]

[27.14350579]

[31.93765208]

[15.19483586]

[34.01357428]

[12.85763091]

[21.06646184]

[28.58470042]

[15.77437534]

[24.77512495]

[ 3.64655689]

[23.91169589]

[25.82292925]

[23.03339677]

[25.35158335]

[33.05655447]

[20.65930467]

[38.18917361]

[14.04714297]

[25.26034469]

[17.6138723 ]

[20.60883766]

[ 9.8525544 ]

[21.06756951]

[22.20145587]

[32.2920276 ]

[31.57638342]

[15.29265938]

[16.7100235 ]

[29.10550932]

[25.17762329]

[16.88159225]

[ 6.32621877]

[26.70210263]

[23.3525851 ]

[17.24168182]

[13.22815696]

[39.49907507]

[16.53528575]

[18.14635902]

[25.06620426]

[23.70640231]

[22.20167772]

[21.22272327]

[16.89825921]

[23.15518273]

[28.69699805]

[ 6.65526482]

[23.98399958]

[17.21004545]

[21.0574427 ]

[25.01734597]

[27.65461859]

[20.70205823]

[40.38214871]]

0 20.724023

dtype: float64

20.724023437339696

**TMW3: Spam/Ham Classification using Naïve Bayes**

import numpy as np

import pandas as pd

emails = pd.read\_csv('./emails.csv')

#emails[:10]

def process\_email(text):

text = text.lower()

return list(set(text.split()))

emails['words'] = emails['text'].apply(process\_email)

num\_emails = len(emails)

num\_spam = sum(emails['spam'])

print("Number of emails:", num\_emails)

print("Number of spam emails:", num\_spam)

print()

# Calculating the prior probability that an email is spam

print("Probability of spam:", num\_spam/num\_emails)

print()

model = {}

# Training process

for index, email in emails.iterrows():

for word in email['words']:

if word not in model:

model[word] = {'spam': 1, 'ham': 1}

if word in model:

if email['spam']:

model[word]['spam'] += 1

else:

model[word]['ham'] += 1

def predict\_bayes(word):

word = word.lower()

num\_spam\_with\_word = model[word]['spam']

num\_ham\_with\_word = model[word]['ham']

return 1.0\*num\_spam\_with\_word/(num\_spam\_with\_word + num\_ham\_with\_word)

print("Prediction using Bayes for word sale",predict\_bayes("sale"))

print("Prediction using Bayes for word lottery",predict\_bayes("lottery"))

print()

def predict\_naive\_bayes(email):

total = len(emails)

num\_spam = sum(emails['spam'])

num\_ham = total - num\_spam

email = email.lower()

words = set(email.split())

spams = [1.0]

hams = [1.0]

for word in words:

if word in model:

spams.append(model[word]['spam']/num\_spam\*total)

hams.append(model[word]['ham']/num\_ham\*total)

prod\_spams = np.compat.long(np.prod(spams)\*num\_spam)

prod\_hams = np.compat.long(np.prod(hams)\*num\_ham)

return prod\_spams/(prod\_spams + prod\_hams)

print("Prediction using NaiveBayes for word lottery sale",predict\_naive\_bayes("lottery sale"))

print("Prediction using NaiveBayes for word asdfgh",predict\_naive\_bayes("asdfgh"))

print("Prediction using NaiveBayes ",predict\_naive\_bayes('Hi mom how are you'))

**OUTPUT:**

**Number of emails: 5728**

**Number of spam emails: 1368**

**Probability of spam: 0.2388268156424581**

**Prediction using Bayes for word sale 0.48148148148148145**

**Prediction using Bayes for word lottery 0.9**

**Prediction using NaiveBayes for word lottery sale 0.9638144992048691**

**Prediction using NaiveBayes for word asdfgh 0.2388268156424581**

**Prediction using NaiveBayes 0.12554358867164464**