**ML 2020 Practical Solutions**

Q1. Implement & demonstrate the Find-s algorithm for finding the most specific hypothesis based on a given set of training data from a .csv file.

import csv

num\_attributes = 4

a = []

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

with open('PlayTennis.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 row 1 value of hypothesis: ")

print(hypothesis)

print("\n Find S: algorithm for finding most 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 Most Specific Hypothesis for a given Training Examples :\n")

print(hypothesis)

Q2. For a given set of training data examples stored in a .csv file, implement & demonstrate the Candidate Elimination algorithm to output a description of the set of all hypothesis consistent with the training examples.

import pandas as pd

import numpy as np

import csv

import random

def g\_0(n):

    return ("?",)\*n

def s\_0(n):

    return ('0',)\*n

def more\_general(h1, h2):

    more\_general\_parts = []

    for x, y in zip(h1, h2):

        mg = x == "?" or (x != "0" and (x == y or y == "0"))

        more\_general\_parts.append(mg)

    return all(more\_general\_parts)

def fulfills(example, hypothesis):

    return more\_general(hypothesis, example)

def min\_generalizations(h, x):

    h\_new = list(h)

    for i in range(len(h)):

        if not fulfills(x[i:i+1], h[i:i+1]):

            h\_new[i] = '?' if h[i] != '0' else x[i]

    return [tuple(h\_new)]

def min\_specializations(h, domains, x):

    results = []

    for i in range(len(h)):

        if h[i] == "?":

            for val in domains[i]:

                if x[i] != val:

                    h\_new = h[:i] + (val,) + h[i+1:]

                    results.append(h\_new)

        elif h[i] != "0":

            h\_new = h[:i] + ('0',) + h[i+1:]

            results.append(h\_new)

    return results

with open('PlayTennis.csv')  as csvFile:

        examples = [tuple(line) for line in csv.reader(csvFile)]

def get\_domains(examples):

    d = [set() for i in examples[0]]

    for x in examples:

        for i, xi in enumerate(x):

            d[i].add(xi)

    return [list(sorted(x)) for x in d]

def candidate\_elimination(examples):

    domains = get\_domains(examples)[:-1]

    G = set([g\_0(len(domains))])

    S = set([s\_0(len(domains))])

    i=0

    print("\n G[{0}]:".format(i),G)

    print("\n S[{0}]:".format(i),S)

    for xcx in examples:

        i=i+1

        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions

        if cx=='Y': # x is positive example

            G = {g for g in G if fulfills(x, g)}

            S = generalize\_S(x, G, S)

        else: # x is negative example

            S = {s for s in S if not fulfills(x, s)}

            G = specialize\_G(x, domains, G, S)

        print("\n G[{0}]:".format(i),G)

        print("\n S[{0}]:".format(i),S)

    return

def generalize\_S(x, G, S):

    S\_prev = list(S)

    for s in S\_prev:

        if s not in S:

            continue

        if not fulfills(x, s):

            S.remove(s)

            Splus = min\_generalizations(s, x)

            ## keep only generalizations that have a counterpart in G

            S.update([h for h in Splus if any([more\_general(g,h)

                                               for g in G])])

            ## remove hypotheses less specific than any other in S

            S.difference\_update([h for h in S if

                                 any([more\_general(h, h1)

                                      for h1 in S if h != h1])])

    return S

def specialize\_G(x, domains, G, S):

    G\_prev = list(G)

    for g in G\_prev:

        if g not in G:

            continue

        if fulfills(x, g):

            G.remove(g)

            Gminus = min\_specializations(g, domains, x)

            ## keep only specializations that have a conuterpart in S

            G.update([h for h in Gminus if any([more\_general(h, s)

                                                for s in S])])

            ## remove hypotheses less general than any other in G

            G.difference\_update([h for h in G if

                                 any([more\_general(g1, h)

                                      for g1 in G if h != g1])])

    return G

candidate\_elimination(examples)

Q9. Create a 4 x 4 matrix and find its inverse. Colour plot both the matrices as each row being values for different functions.

import numpy as np

import matplotlib.pyplot as plt

# Taking a 4 \* 4 matrix

A = np.array([[6, 1, 1, 3],

              [4, 10, 5, 1],

              [2, 8, 7, 6],

              [3, 1, -9, 7]])

# Calculating the inverse of the matrix

B = np.linalg.inv(A)

print("inverse of the given array : ", B)

plt.rcParams["figure.figsize"] = [7.00, 4.00]

plt.rcParams["figure.autolayout"] = True

colorPlotA = plt.imshow(A, cmap = "copper\_r")

plt.colorbar(colorPlotA)

plt.show()

Q10. Consider the function y=cos(x) and plot the function. Now from this function find discrete values of y for x values 0,1,2,3 and 4. Create a table for the x and y values. Now using linear regression find values of y for appropriate x values. Plot for the regressed value and original function.

import matplotlib.pyplot as plt

import numpy as np

import math as m

from tabulate import tabulate

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import statsmodels.api as sm

# Creating vectors X and Y

x = np.array([0,1,2,3,4])

y = x\*\*2

fig = plt.figure(figsize = (10, 5))

# Create the plot

plt.plot(x, y)

# Show the plot

plt.show()

myTable = [[0, m.cos(0)],[1, m.cos(1)], [2, m.cos(2)], [3, m.cos(3)], [4, m.cos(4)]]

head = ["x","y"]

table = tabulate(myTable, headers=head, tablefmt="grid")

print(table)

y = np.array([1,0.540,-0.416,-0.989,-0.653])

n = np.size(x)

x\_mean = np.mean(x)

y\_mean = np.mean(y)

x\_mean,y\_mean

Sxy = np.sum(x\*y)- n\*x\_mean\*y\_mean

Sxx = np.sum(x\*x)-n\*x\_mean\*x\_mean

b1 = Sxy/Sxx

b0 = y\_mean-b1\*x\_mean

print('slope b1 is', b1)

print('intercept b0 is', b0)

plt.scatter(x,y)

plt.xlabel('Independent variable X')

plt.ylabel('Dependent variable y')

Q6. Assuming a set of documentation that need to be classified, use the Naïve Bayesian Classifier model to perform this task.

total\_documents = int(input("Enter the Total Number of documents: "))

doc\_class = []

i = 0

keywords = []

while not i == total\_documents:

    doc\_class.append([])

    text = input(f"\nEnter the text of Doc-{i+1} : ").lower()

    cls = input(f"Enter the class of Doc-{i+1} : ")

    doc\_class[i].append(text.split())

    doc\_class[i].append(cls)

    keywords.extend(text.split())

    i = i+1

keywords = set(keywords)

keywords = list(keywords)

keywords.sort()

to\_find = input("\nEnter the Text to classify using Naive Bayes: ").lower().split()

probability\_table = []

for i in range(total\_documents):

    probability\_table.append([])

    for j in keywords:

        probability\_table[i].append(0)

doc\_id = 1

for i in range(total\_documents):

    for k in range(len(keywords)):

        if keywords[k] in doc\_class[i][0]:

            probability\_table[i][k] += doc\_class[i][0].count(keywords[k])

print('\n')

import prettytable

keywords.insert(0, 'Document ID')

keywords.append("Class")

Prob\_Table = prettytable.PrettyTable()

Prob\_Table.field\_names = keywords

Prob\_Table.title = 'Probability of Documents'

x=0

for i in probability\_table:

    i.insert(0,x+1)

    i.append(doc\_class[x][1])

    Prob\_Table.add\_row(i)

    x=x+1

print(Prob\_Table)

print('\n')

for i in probability\_table:

    i.pop(0)

totalpluswords=0

totalnegwords=0

totalplus=0

totalneg=0

vocabulary=len(keywords)-2

for i in probability\_table:

    if i[len(i)-1]=="+":

        totalplus+=1

        totalpluswords+=sum(i[0:len(i)-1])

    else:

        totalneg+=1

        totalnegwords+=sum(i[0:len(i)-1])

keywords.pop(0)

keywords.pop(len(keywords)-1)

#For positive class

temp=[]

for i in to\_find:

    count=0

    x=keywords.index(i)

    for j in probability\_table:

        if j[len(j)-1]=="+":

            count=count+j[x]

    temp.append(count)

    count=0

for i in range(len(temp)):

    temp[i]=format((temp[i]+1)/(vocabulary+totalpluswords),".4f")

print()

temp=[float(f) for f in temp]

print("Probabilities of Each word to be in '+' class are: ")

h=0

for i in to\_find:

    print(f"P({i}/+) = {temp[h]}")

    h=h+1

print()

pplus=float(format((totalplus)/(totalplus+totalneg),".8f"))

for i in temp:

    pplus=pplus\*i

pplus=format(pplus,".8f")

print("probability of Given text to be in '+' class is :",pplus)

print()

#For Negative class

temp=[]

for i in to\_find:

    count=0

    x=keywords.index(i)

    for j in probability\_table:

        if j[len(j)-1]=="-":

            count=count+j[x]

    temp.append(count)

    count=0

for i in range(len(temp)):

    temp[i]=format((temp[i]+1)/(vocabulary+totalnegwords),".4f")

print()

temp=[float(f) for f in temp]

print("Probabilities of Each word to be in '-' class are: ")

h=0

for i in to\_find:

    print(f"P({i}/-) = {temp[h]}")

    h=h+1

print()

pneg=float(format((totalneg)/(totalplus+totalneg),".8f"))

for i in temp:

    pneg=pneg\*i

pneg=format(pneg,".8f")

print("probability of Given text to be in '-' class is :",pneg)

print('\n')

if pplus>pneg:

    print(f"Using Naive Bayes Classification, We can clearly say that the given text belongs to '+' class with probability {pplus}")

else:

    print(f"Using Naive Bayes Classification, We can clearly say that the given text belongs to '-' class with probability {pneg}")

print('\n')