# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# MACHINE LEARNING

***Submitted by***

**POTANA KUNDANA SAI PRIYA (1BM19CS112)**

***in partial fulfillment for the award of the degree of***
**BACHELOR OF ENGINEERING**
***in***
**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**
**(Autonomous Institution under VTU)**
**BENGALURU-560019**
**May-2022 to July-2022**

# B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering

## CERTIFICATE

This is to certify that the Lab work entitled "MACHINE LEARNING" carried out by POTANA KUNDANA SAI PRIYA (1BM19CS112), who is bonafide student of B. M. S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a MACHINE LEARNING - (20CS6PCMAL) work prescribed for the said degree.

| | |
|---|---|
| Name of the Lab-Incharge | **Dr. Jyothi S Nayak** |
| Designation | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |

`

## Index Sheet

| Sl. No. | Experiment Title | Page No. |
|---|---|---|
| 1 | FIND S ALGORITHM | |
| 2 | CANDIDATE ELIMINATION ALGORITHM | |
| 3 | ID3 ALGORITHM | |
| 4 | NAÏVE BAYES ALGORITHM | |
| 5 | LINEAR REGRESSION | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## Course Outcome

| | |
|---|---|
| | |

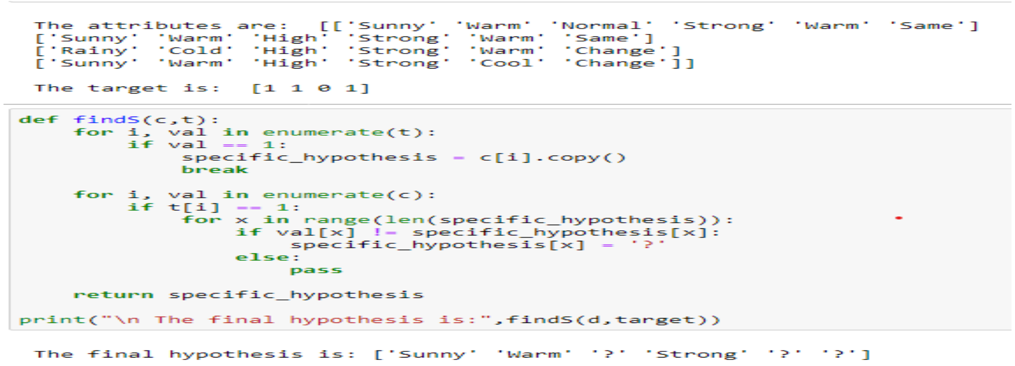
# LAB-1

# FIND-S ALGORITHM

- ## PROGRAM

```
import numpy as np
import pandas as pd

data=pd.read_csv("ENJOYSPORT.csv")
print(data,"\n")

d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)

def findS(c,t):
   for i, val in enumerate(t):
      if val == 1:
         specific_hypothesis = c[i].copy()
         break

   for i, val in enumerate(c):
      if t[i] == 1:
         for x in range(len(specific_hypothesis)):
            if val[x] != specific_hypothesis[x]:
               specific_hypothesis[x] = '?'
            else:
               pass

   return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

- ## OUTPUT



```
 The attributes are:  [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

 The target is:  [1 1 0 1]
```

```
def findS(c,t):
    for i, val in enumerate(t):
        if val == 1:
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == 1:
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

```
 The final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

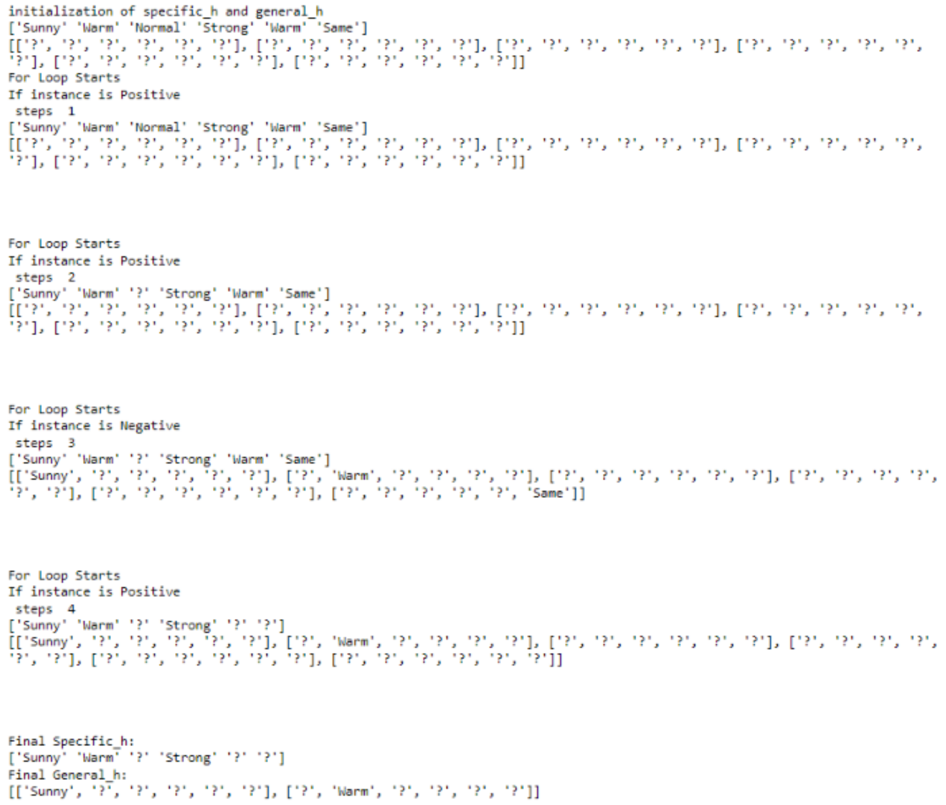
# LAB-2

# CANDIDATE ELIMINATION ALGORITHM

- ## PROGRAM

```
import numpy as np
import pandas as pd

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

concepts= np.array(data)[:,:-1]
print("\n The attributes are: ",concepts)
target = np.array(data)[:,-1]
print("\n The target is: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == 1:
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == 0:
            print("If instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(" steps ",i+1)
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

- **OUTPUT**



```
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps  1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps  2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Negative
 steps  3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]



For Loop Starts
If instance is Positive
 steps  4
['Sunny' 'Warm' '?' 'Strong' '?' '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

# LAB-3

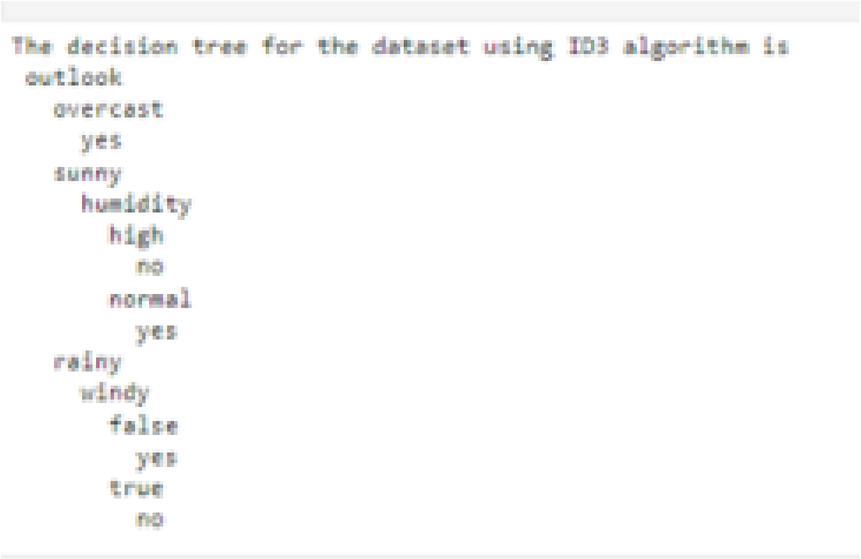
# ID-3 ALGORITHM

- ## PROGRAM

```
import math
import csv
def load_csv(filename):
   lines=csv.reader(open(filename,"r"));
   dataset = list(lines)
   headers = dataset.pop(0)
   return dataset,headers
class Node:
   def __init__(self,attribute):
      self.attribute=attribute
      self.children=[]
      self.answer=""
def subtables(data,col,delete):
   dic={}
   coldata=[row[col] for row in data]
   attr=list(set(coldata))

   counts=[0]*len(attr)
   r=len(data)
   c=len(data[0])
   for x in range(len(attr)):
      for y in range(r):
         if data[y][col]==attr[x]:
            counts[x]+=1

   for x in range(len(attr)):
      dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
      pos=0
      for y in range(r):
         if data[y][col]==attr[x]:
            if delete:
               del data[y][col]
            dic[attr[x]][pos]=data[y]
            pos+=1
   return attr,dic
def entropy(S):
   attr=list(set(S))
   if len(attr)==1:
      return 0

   counts=[0,0]
```

```
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums
def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)
def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
dataset,features=load_csv("tennis.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
```

- ## OUTPUT



```
The decision tree for the dataset using ID3 algorithm is
 outlook
   overcast
     yes
   sunny
     humidity
       high
         no
       normal
         yes
   rainy
     windy
       false
         yes
       true
         no
```

# LAB-4

# NAÏVE BAYES ALGORITHM

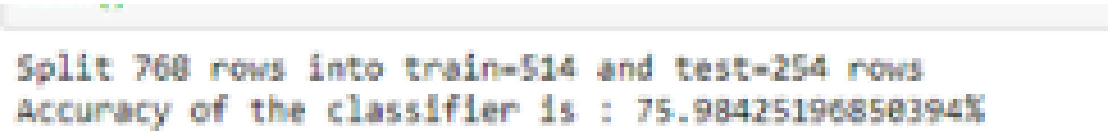
## • PROGRAM

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import csv
import random
import math


# In[2]:


def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));
        dataset = list(lines)
        for i in range(len(dataset)):
    #converting strings into numbers for processing
                dataset[i] = [float(x) for x in dataset[i]]

        return dataset


# In[3]:


def splitdataset(dataset, splitratio):
  #67% training size
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]


# In[5]:
```

```
def separatebyclass(dataset):
        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in separated):
                        separated[vector[-1]] = []
                separated[vector[-1]].append(vector)
        return separated


# In[6]:


def mean(numbers):
        return sum(numbers)/float(len(numbers))

def stdev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)


# In[7]:


def summarize(dataset): #creates a dictionary of classes
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
        del summaries[-1] #excluding labels +ve or -ve
        return summaries

def summarizebyclass(dataset):
        separated = separatebyclass(dataset);
  #print(separated)
        summaries = {}
        for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
                summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and
std
        return summaries


# In[8]:


def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
    probabilities = {} # probabilities contains the all prob of all class of test data
    for classvalue, classsummaries in summaries.items():#class and attribute information as mean
and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and sd of every attribute for class
0 and 1 seperaely
            x = inputvector[i] #testvector's first attribute
            probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal
dist
    return probabilities


# In[9]:


def predict(summaries, inputvector): #training and test data is passed
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions


# In[10]:


def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testset))) * 100.0


# In[12]:
```

```
def main():
        filename = 'naivedataset.csv'
        splitratio = 0.67
        dataset = loadcsv(filename);

        trainingset, testset = splitdataset(dataset, splitratio)
        print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))
        # prepare model
        summaries = summarizebyclass(trainingset);
        #print(summaries)
  # test model
        predictions = getpredictions(summaries, testset) #find the predictions of test data with the
training data
        accuracy = getaccuracy(testset, predictions)
        print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

- ## OUTPUT



```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is : 75.98425196850394%
```

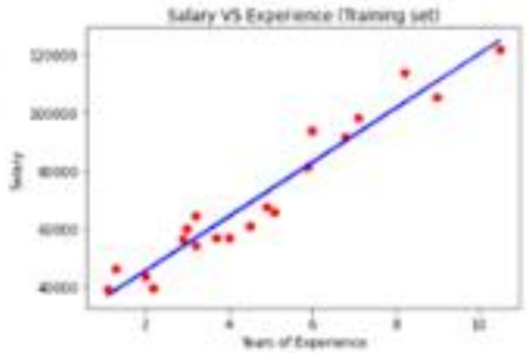
# LAB-5

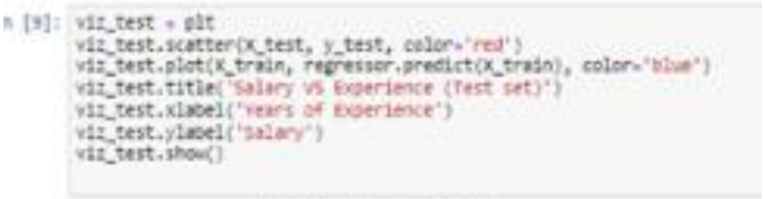
## LINEAR REGRESSION ALGORITHM

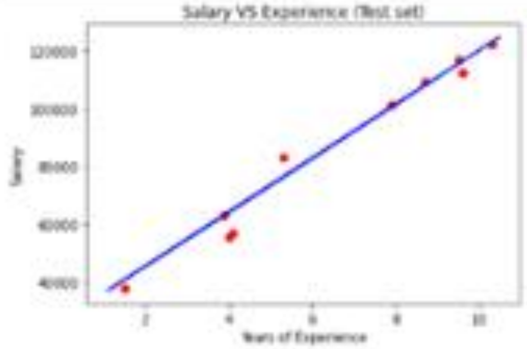
- ### PROGRAM

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()

viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

- **OUTPUT**




```
n [9]: viz_test = plt
       viz_test.scatter(X_test, y_test, color='red')
       viz_test.plot(X_train, regressor.predict(X_train), color='blue')
       viz_test.title('Salary VS Experience (Test set)')
       viz_test.xlabel('Years of Experience')
       viz_test.ylabel('Salary')
       viz_test.show()
```


Salary VS Experience (Test set)
Salary
Years of Experience