Machine Learning Laboratory

1. **Implement and demonstrate the FIND-S algorithm for finding the most specific**

**hypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

##### FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x For each attribute constraint ai in h

If the constraint ai is satisfied by x Then do nothing

Else replace ai in h by the next more general constraint that is satisfied by x

1. Output hypothesis h

##### Training Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| **1** | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| **2** | Sunny | Warm | High | Strong | Warm | Same | Yes |
| **3** | Rainy | Cold | High | Strong | Warm | Change | No |
| **4** | Sunny | Warm | High | Strong | Cool | Change | Yes |

***Program:***

import csv a = []

with open('enjoysport.csv', 'r') as csvfile: for row in csv.reader(csvfile):

a.append(row) print(a)

print("\n The total number of training instances are : ",len(a)) num\_attribute = len(a[0])-1

print("\n The initial hypothesis is : ") hypothesis = ['0']\*num\_attribute print(hypothesis)

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

if a[i][num\_attribute] == 'yes':

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("\n The hypothesis for the training instance {} is :

\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

#### Data Set:

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

**Output:**

The Given Training Data Set

['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']

The hypothesis for the training instance 1 is : ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is : ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is : ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is : ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is

#### ['sunny', 'warm', '?', 'strong', '?', '?']

1. **For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

##### CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H Initialize S to the set of maximally specific hypotheses in H For each training example d, do

* + If d is a positive example
    - Remove from G any hypothesis inconsistent with d
    - For each hypothesis s in S that is not consistent with d
      * Remove s from S
      * Add to S all minimal generalizations h of s such that
        + h is consistent with d, and some member of G is more general than h
      * Remove from S any hypothesis that is more general than another hypothesis in S
* If d is a negative example
  + Remove from S any hypothesis inconsistent with d
  + For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      * h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

CANDIDATE- ELIMINTION algorithm using version spaces

##### Training Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| **1** | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| **2** | Sunny | Warm | High | Strong | Warm | Same | Yes |
| **3** | Rainy | Cold | High | Strong | Warm | Change | No |
| **4** | Sunny | Warm | High | Strong | Cool | Change | Yes |

***Program:***

import numpy as np import pandas as pd

data = pd.DataFrame(data=pd.read\_csv('enjoysport.csv')) concepts = np.array(data.iloc[:,0:-1])

print(concepts)

target = np.array(data.iloc[:,-1]) print(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): if target[i] == "yes":

for x in range(len(specific\_h)): if h[x]!= specific\_h[x]:

specific\_h[x] ='?' general\_h[x][x] ='?'

print(specific\_h) print(specific\_h)

if target[i] == "no":

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 of Candidate Elimination Algorithm",i+1) print(specific\_h)

print(general\_h)

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")

#### Data Set:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| 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 |

**Output:**

Final Specific\_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General\_h:

[['sunny', '?', '?', '?', '?', '?'],

['?', 'warm', '?', '?', '?', '?']]

1. **Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

##### ID3 Algorithm

ID3(Examples, Target\_attribute, Attributes)

Examples are the training examples. Target\_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

* Create a Root node for the tree
* If all Examples are positive, Return the single-node tree Root, with label = +
* If all Examples are negative, Return the single-node tree Root, with label = -
* If Attributes is empty, Return the single-node tree Root, with label = most common value of Target\_attribute in Examples
* Otherwise Begin
  + A ← the attribute from Attributes that best\* classifies Examples
  + The decision attribute for Root ← A
  + For each possible value, *vi*, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = *vi*
    - Let *Examples vi*, be the subset of Examples that have value *vi* for *A*
    - If *Examples vi* , is empty
      * Then below this new branch add a leaf node with label = most common value of Target\_attribute in Examples
      * Else below this new branch add the subtree ID3(*Examples vi*, Targe\_tattribute, Attributes – {A}))
* End
* Return Root
* The best attribute is the one with highest information gain

**ENTROPY:**

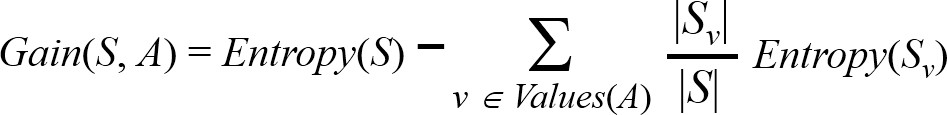
*Entropy measures the impurity of a collection of examples.*



Where, *p+* is the proportion of positive examples in S

*p-* is the proportion of negative examples in S.

**INFORMATION GAIN:**

* + ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
  + The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

##### Training Dataset:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | ***Outlook*** | ***Temperature*** | ***Humidity*** | ***Wind*** | ***PlayTennis*** |
| **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 |

***Test Dataset:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Day** | ***Outlook*** | ***Temperature*** | ***Humidity*** | ***Wind*** |
| **T1** | Rain | Cool | Normal | Strong |
| **T2** | Sunny | Mild | Normal | Strong |

##### 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)

'''Main program''' dataset,features=load\_csv("data3.csv") node1=build\_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")

print\_tree(node1,0) testdata,features=load\_csv("data3\_test.csv") for xtest in testdata:

print("The test instance:",xtest)

print("The label for test instance:",end=" ") classify(node1,xtest,features)

#### Output:

The decision tree for the dataset using ID3 algorithm is

Outlook rain

Wind

overcast

yes

strong weak

no yes

sunny

Humidity

normal

yes

high

no

The test instance: ['rain', 'cool', 'normal', 'strong'] The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong'] The label for test instance: yes

1. **Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

##### BACKPROPAGATION Algorithm

BACKPROPAGATION (*training\_example, ƞ, nin, nout, nhidden )*

*Each training example is a pair of the form (*𝑥⃗ →,

𝑡→ *), where (*𝑥→ *) is the vector of network*

*input values, (*𝑡→ *) and is the vector of target network output values.*

*ƞ is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units.*

*The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji*

### Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.

* Initialize all network weights to small random numbers

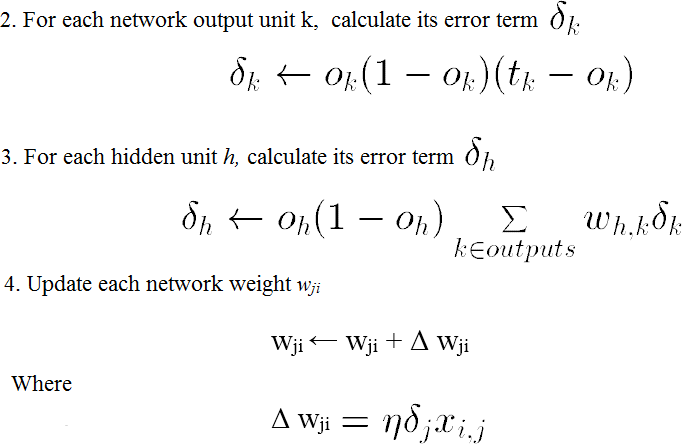
### Until the termination condition is met, Do

* + For each (⃗𝑥→,

### 𝑡→ ), in training examples, Do

*Propagate the input forward through the network:*

### 1. Input the instance ⃗𝑥→, to the network and compute the output ou of every unit u in the network.

*Propagate the errors backward through the network:*

##### Training Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | **Sleep** | **Study** | **Expected % in**  **Exams** |
| **1** | 2 | 9 | 92 |
| **2** | 1 | 5 | 86 |
| **3** | 3 | 6 | 89 |

Normalize the input

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | **Sleep** | **Study** | **Expected % in Exams** |
| **1** | 2/3 = 0.66666667 | 9/9 = 1 | 0.92 |
| **2** | 1/3 = 0.33333333 | 5/9 = 0.55555556 | 0.86 |
| **3** | 3/3 = 1 | 6/9 = 0.66666667 | 0.89 |

##### Program:

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neur ons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neuron s))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y for i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp) outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp)

#Backpropagation EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

#how much hidden layer wts contributed to error hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

print("Input: \n" + str(X)) print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

**Output:**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output: [[0.92]

[0.86]

[0.89]]

Predicted Output: [[0.89726759]

[0.87196896]

[0.9000671]]

1. **Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

#### finalBayes’ Theorem is stated as:

Where,

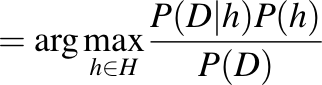
**P(h|D)** is the probability of hypothesis h given the data D. This is called the **posterior probability**.

**P(D|h)** is the probability of data d given that the hypothesis h was true.

**P(h)** is the probability of hypothesis h being true. This is called the **prior probability of h. P(D)** is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is ***hMAP*** is a MAP hypothesis provided

finalfinal

(Ignoring P(D) since it is a constant)

# Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It’s specifically used when the features have continuous values. It’s also assumed that all the features are following a Gaussian distribution i.e., normal distribution

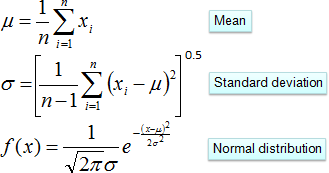
## Representation for Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real- valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

## Gaussian Naive Bayes Model from Data

The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.



Example: Refer the link

<http://chem-eng.utoronto.ca/~datamining/dmc/naive_bayesian.htm>

##### Examples:

* The data set used in this program is the ***Pima Indians Diabetes problem***.
* This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.

##### The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome

* Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Examples | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabetic Pedigree  Function | Age | Outcome |
| 1 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 2 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 3 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 5 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 6 | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |
| 7 | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 8 | 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | 0 |
| 9 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 10 | 8 | 125 | 96 | 0 | 0 | 0 | 0.232 | 54 | 1 |

##### Program:

import csv import random import math

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

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]

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

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)

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

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 contains the all prob of all class of test data probabilities = {}

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

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 the 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

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

def main():

filename = 'naivedata.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 : 71.65354330708661%

1. **Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.**

***Naive Bayes algorithms for learning and classifying text***

#### LEARN\_NAIVE\_BAYES\_TEXT (Examples, V)

*Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms P(wk |vj,), describing the probability that a randomly drawn word from a document in class vj will be the English word wk. It also learns the class prior probabilities P(vj).*

1. *collect all words, punctuation, and other tokens that occur in Examples*
   * *Vocabulary ←* ***c*** the set of all distinct words and other tokens occurring in any text document from *Examples*
2. *calculate the required P(vj) and P(wk|vj) probability terms*
   * For each target value *vj* in *V* do
     + *docsj* ← the subset of documents from *Examples* for which the target value is *vj*
     + *P(vj)* ← | *docsj* | / |Examples|
     + *Textj ←* a single document created by concatenating all members of *docsj*
     + *n* ← total number of distinct word positions in *Textj*
     + for each word *wk* in *Vocabulary*
       - *nk* **←** number of times word ***wk*** occurs in *Textj*
       - *P(wk|vj)* ← ( *nk* + 1) / (n + | *Vocabulary|* )

#### CLASSIFY\_NAIVE\_BAYES\_TEXT (Doc)

*Return the estimated target value for the document Doc. ai denotes the word found in the ith position within Doc.*

* *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
* Return *VNB,* where



##### Data set:

|  |  |  |
| --- | --- | --- |
|  | **Text Documents** | **Label** |
| **1** | I love this sandwich | pos |
| **2** | This is an amazing place | pos |
| **3** | I feel very good about these beers | pos |
| **4** | This is my best work | pos |
| **5** | What an awesome view | pos |
| **6** | I do not like this restaurant | neg |
| **7** | I am tired of this stuff | neg |
| **8** | I can't deal with this | neg |
| **9** | He is my sworn enemy | neg |
| **10** | My boss is horrible | neg |
| **11** | This is an awesome place | pos |
| **12** | I do not like the taste of this juice | neg |
| **13** | I love to dance | pos |
| **14** | I am sick and tired of this place | neg |
| **15** | What a great holiday | pos |
| **16** | That is a bad locality to stay | neg |
| **17** | We will have good fun tomorrow | pos |
| **18** | I went to my enemy's house today | neg |

***Program:***

import pandas as pd msg=pd.read\_csv('naivetext.csv',names=['message','label']) print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.message y=msg.labelnum

print(X) print(y)

#splitting the dataset into train and test data from sklearn.model\_selection import train\_test\_split xtrain,xtest,ytrain,ytest=train\_test\_split(X,y)

print ('\n The total number of Training Data :',ytrain.shape) print ('\n The total number of Test Data :',ytest.shape)

#output of count vectoriser is a sparse matrix from sklearn.feature\_extraction.text import CountVectorizer count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain) xtest\_dtm=count\_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n') print(count\_vect.get\_feature\_names())

df=pd.DataFrame(xtrain\_dtm.toarray(),columns=count\_vect.get\_fe ature\_names())

# Training Naive Bayes (NB) classifier on training data. from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain\_dtm,ytrain) predicted = clf.predict(xtest\_dtm)

#printing accuracy, Confusion matrix, Precision and Recall from sklearn import metrics

print('\n Accuracy of the classifer is’,

metrics.accuracy\_score(ytest,predicted))

3

print('\n Confusion matrix') print(metrics.confusion\_matrix(ytest,predicted))

print('\n The value of Precision' , metrics.precision\_score(ytest,predicted))

print('\n The value of Recall' , metrics.recall\_score(ytest,predicted))

#### Output:

The dimensions of the dataset (18, 2)

1. I love this sandwich
2. This is an amazing place
3. I feel very good about these beers
4. This is my best work
5. What an awesome view
6. I do not like this restaurant
7. I am tired of this stuff
8. I can't deal with this
9. He is my sworn enemy
10. My boss is horrible
11. This is an awesome place
12. I do not like the taste of this juice
13. I love to dance
14. I am sick and tired of this place
15. What a great holiday
16. That is a bad locality to stay
17. We will have good fun tomorrow
18. I went to my enemy's house today

4

Name: message, dtype: object 0 1

1 1

2 1

3 1

4 1

5 0

6 0

7 0

8 0

9 0

10 1

11 0

12 1

13 0

14 1

15 0

16 1

17 0

Name: labelnum, dtype: int64

The total number of Training Data: (13,) The total number of Test Data: (5,)

The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do', 'enemy', 'feel',

'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place',

'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

Accuracy of the classifier is 0.8 Confusion matrix

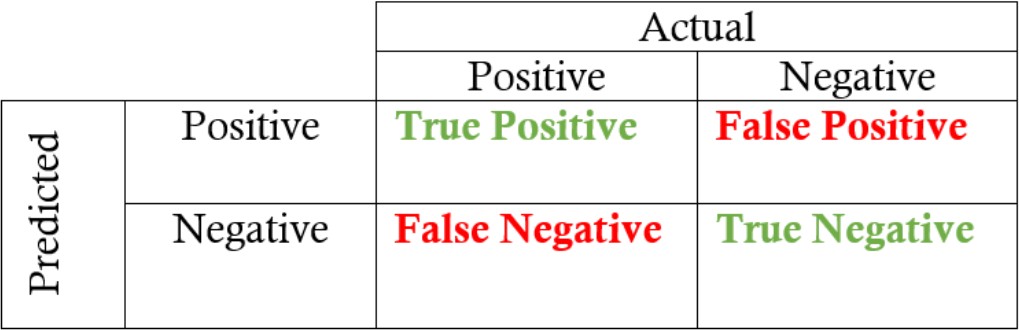
[[2 1]

[0 2]]

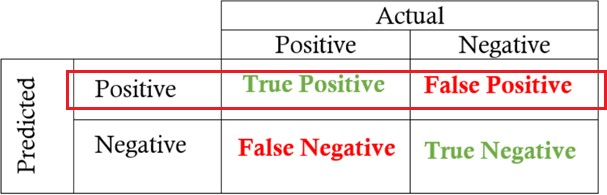
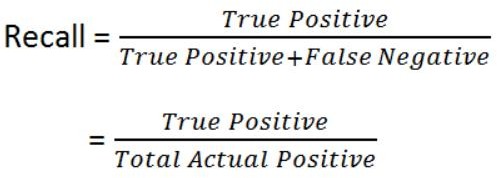
The value of Precision 0.6666666666666666 The value of Recall 1.0

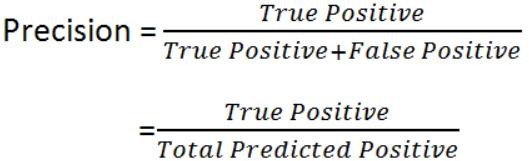
**Basic knowledge**

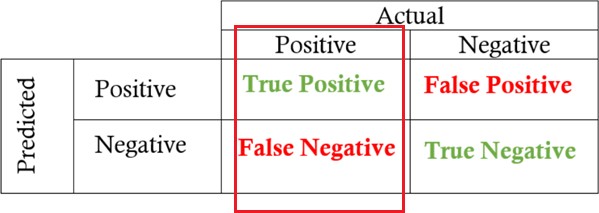
# Confusion Matrix



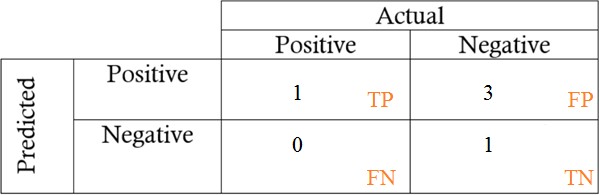
**True positives:** data points labelled as positive that are actually positive **False positives:** data points labelled as positive that are actually negative **True negatives:** data points labelled as negative that are actually negative **False negatives:** data points labelled as negative that are actually positive

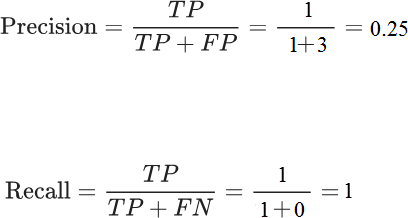






# Example:





**Accuracy:** how often is the classifier correct?



Example: Movie Review

|  |  |  |
| --- | --- | --- |
| Doc | Text | Class |
| 1 | I loved the movie | **+** |
| 2 | I hated the movie | **-** |
| 3 | a great movie. good movie | **+** |
| 4 | poor acting | **-** |
| 5 | great acting. good movie | **+** |

Unique word

< I, loved, the, movie, hated, a, great, good, poor, acting>

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Doc | I | loved | the | movie | hated | a | great | good | poor | acting | Class |
| 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  | + |
| 2 | 1 |  | 1 | 1 | 1 |  |  |  |  |  | - |
| 3 |  |  |  | 2 |  | 1 | 1 | 1 |  |  | + |
| 4 |  |  |  |  |  |  |  |  | 1 | 1 | - |
| 5 |  |  |  | 1 |  |  | 1 | 1 |  | 1 | + |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Doc | I | loved | the | movie | hated | a | great | good | poor | acting | Class |
| 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  | + |
| 3 |  |  |  | 2 |  | 1 | 1 | 1 |  |  | + |
| 5 |  |  |  | 1 |  |  | 1 | 1 |  | 1 | + |

𝑃(+) =

3

= 0.6

5

|  |  |
| --- | --- |
| 1 + 1  𝑃(𝐼 |+) = = 0.0833 14 + 10  1 + 1  𝑃(𝑙𝑜𝑣𝑒𝑑 |+) = = 0.0833 14 + 10  1 + 1  𝑃(𝑡ℎ𝑒 |+) = = 0.0833 14 + 10  4 + 1  𝑃(𝑚𝑜𝑣𝑖𝑒 |+) = = 0.2083 14 + 10  0 + 1  𝑃(ℎ𝑎𝑡𝑒𝑑 |+) = = 0.0416 14 + 10 | 1 + 1  𝑃(𝑎 |+) = = 0.0833 14 + 10  2 + 1  𝑃(𝑔𝑟𝑒𝑎𝑡 |+) = = 0.125 14 + 10  2 + 1  𝑃(𝑔𝑜𝑜𝑑 |+) = = 0.125 14 + 10  0 + 1  𝑃(𝑝𝑜𝑜𝑟 |+) = = 0.0416 14 + 10  1 + 1  𝑃(𝑎𝑐𝑡𝑖𝑛𝑔 |+) = = 0.0833 14 + 10 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Doc | I | loved | the | movie | hated | a | great | good | poor | acting | Class |
| 2 | 1 |  | 1 | 1 | 1 |  |  |  |  |  | - |
| 4 |  |  |  |  |  |  |  |  | 1 | 1 | - |

𝑃(−) =

2

= 0.4

5

|  |  |
| --- | --- |
| 1 + 1  𝑃(𝐼 |−) = = 0.125 6 + 10  0 + 1  𝑃(𝑙𝑜𝑣𝑒𝑑 |−) = = 0.0625 6 + 10  1 + 1  𝑃(𝑡ℎ𝑒 |−) = = 0.125 6 + 10  1 + 1  𝑃(𝑚𝑜𝑣𝑖𝑒|−) = = 0.125 6 + 10  1 + 1  𝑃(ℎ𝑎𝑡𝑒𝑑 |−) = = 0.125 6 + 10 | 0 + 1  𝑃(𝑎 |−) = = 0.0625 6 + 10  0 + 1  𝑃(𝑔𝑟𝑒𝑎𝑡 |−) = = 0.0625 6 + 10  0 + 1  𝑃(𝑔𝑜𝑜𝑑 |−) = = 0.0625 6 + 10  1 + 1  𝑃(𝑝𝑜𝑜𝑟|−) = = 0.125 6 + 10  1 + 1  𝑃(𝑎𝑐𝑡𝑖𝑛𝑔|−) = = 0.125 6 + 10 |

Let’s classify the new document

I hated the poor acting

If Vj = + then,

= P(+) P(I | +) P(hated | +) P(the | +) P(poor | +) P(acting | +)

= 0.6 \* 0.0833 \* 0.0416 \* 0.0833 \* 0.0416 \* 0.0833

= 6.03 X 10−2

If Vj = −

then,

= P(−) P(I | −) P(hated | −) P(the | −) P(poor | −) P(acting | −)

= 0.4 \* 0.125 \* 0.125 \* 0.125 \* 0.125 \* 0.125

= 1.22 X 10−5

= 1.22 X 10−5 > 6.03 X 10−2

So, the new document belongs to ( − ) class

1. **Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API**

##### Theory

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

* The directed acyclic graph is a set of random variables represented by nodes.
* The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.

The corresponding directed acyclic graph is depicted in below figure.

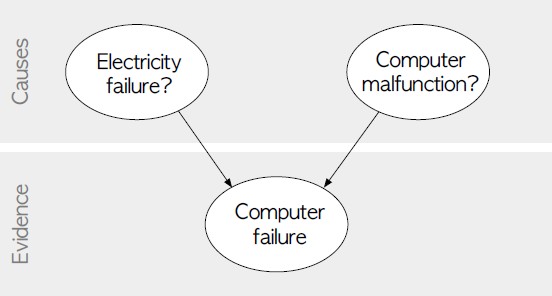


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

##### Data Set:

**Title:** Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database: 0 1 2 3 4 Total

Cleveland: 164 55 36 35 13 303

**Attribute Information:**

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
   * Value 1: typical angina
   * Value 2: atypical angina
   * Value 3: non-anginal pain
   * Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
   * Value 0: normal
   * Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   * Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    * Value 1: upsloping
    * Value 2: flat
    * Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Some instance from the dataset:** | | | | | | | | | | | | | | |
|  | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | Heartdisease |
| 63 | 1 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0 | 6 | 0 |
| 67 | 1 | 4 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3 | 3 | 2 |
| 67 | 1 | 4 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2 | 7 | 1 |
| 41 | 0 | 2 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0 | 3 | 0 |
| 62 | 0 | 4 | 140 | 268 | 0 | 2 | 160 | 0 | 3.6 | 3 | 2 | 3 | 3 |
| 60 | 1 | 4 | 130 | 206 | 0 | 2 | 132 | 1 | 2.4 | 2 | 2 | 7 | 4 |
| ***Program:***  import numpy as np import pandas as pd import csv  from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel  from pgmpy.inference import VariableElimination  #read Cleveland Heart Disease data heartDisease = pd.read\_csv('heart.csv')  heartDisease = heartDisease.replace('?',np.nan)  #display the data print('Sample instances from the dataset are given below') print(heartDisease.head())  #display the Attributes names and datatyes  print('\n Attributes and datatypes') print(heartDisease.dtypes)  #Creat Model- Bayesian Network model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),(  'exang','heartdisease'),('cp','heartdisease'),('heartdisease', 'restecg'),('heartdisease','chol')]) | | | | | | | | | | | | | | |

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators') model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network

print('\n Inferencing with Bayesian Network:') HeartDiseasetest\_infer = VariableElimination(model)

#computing the Probability of HeartDisease given restecg print('\n 1.Probability of HeartDisease given evidence= restecg :1') q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evi dence={'restecg':1})

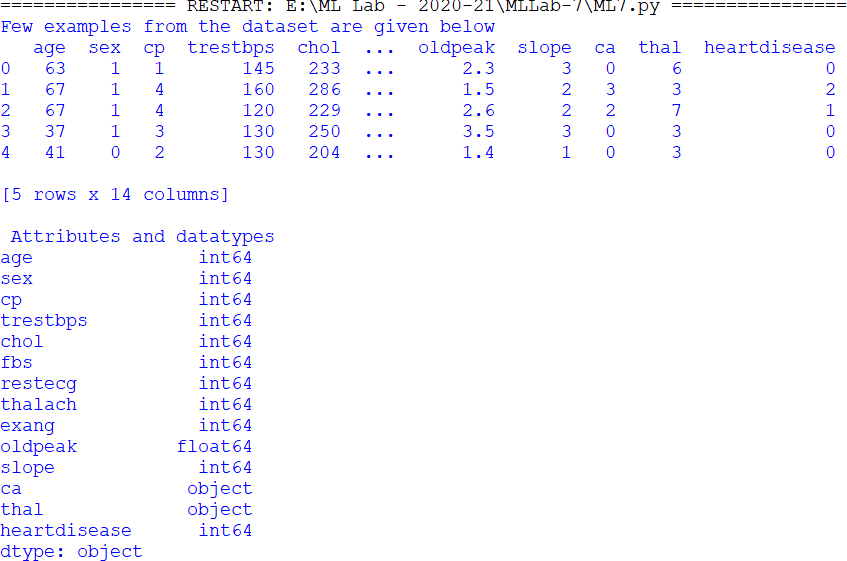
print(q1)

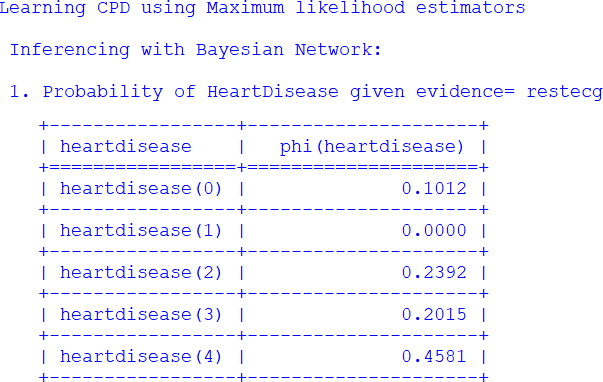
#computing the Probability of HeartDisease given cp

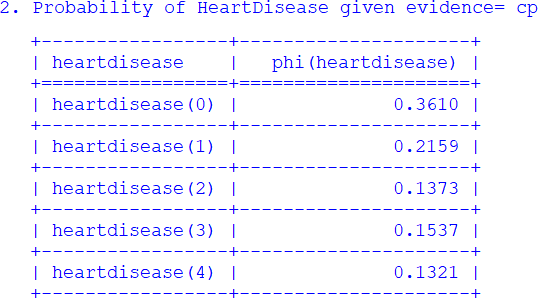
print('\n 2.Probability of HeartDisease given evidence= cp:2 ') q2=HeartDiseasetest\_infer.query(variables=['heartdisease'],evi dence={'cp':2})

print(q2)

##### Output:



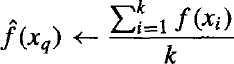




1. **Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.**

##### K-Nearest Neighbor Algorithm

Training algorithm:

* + For each training example (x, f (x)), add the example to the list training examples Classification algorithm:
  + Given a query instance xq to be classified,
    - Let x1 . . .xk denote the k instances from training examples that are nearest to xq
    - Return
      * Where, f(xi) function to calculate the mean value of the k nearest training examples.

##### Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class



**1**

##### Program:

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

from sklearn.metrics import classification\_report, confusion\_matrix from sklearn import datasets

""" Iris Plants Dataset, dataset contains 150 (50 in each of three classes)Number of Attributes: 4 numeric, predictive attributes and the Class

"""

iris=datasets.load\_iris()

""" The x variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.

"""

x = iris.data y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width') print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica') print(y)

""" Splits the dataset into 70% train data and 30% test data. This means that out of total 150 records, the training set will contain

105 records and the test set contains 45 of those records """

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

#To Training the model and Nearest nighbors K=5 classifier = KNeighborsClassifier(n\_neighbors=5) classifier.fit(x\_train, y\_train)

#to make predictions on our test data y\_pred=classifier.predict(x\_test)

""" For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics.

"""

print('Confusion Matrix') print(confusion\_matrix(y\_test,y\_pred)) print('Accuracy Metrics') print(classification\_report(y\_test,y\_pred))

**Output:**

sepal-length sepal-width petal-length petal-width

|  |  |  |  |
| --- | --- | --- | --- |
| [[5.1 | 3.5 | 1.4 | 0.2] |
| [4.9 | 3. | 1.4 | 0.2] |
| [4.7 | 3.2 | 1.3 | 0.2] |
| [4.6 | 3.1 | 1.5 | 0.2] |
| [5. | 3.6 | 1.4 | 0.2] |

. . . . .

. . . . .

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica

[0 0 0 ………0 0 1 1 1 …………1 1 2 2 2 ………… 2 2] Confusion Matrix

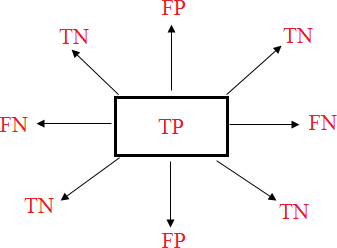
|  |  |  |
| --- | --- | --- |
| [[20 | 0 | 0] |
| [ 0 | 10 | 0] |
| [ 0 | 1 | 14]] |

Accuracy Metrics

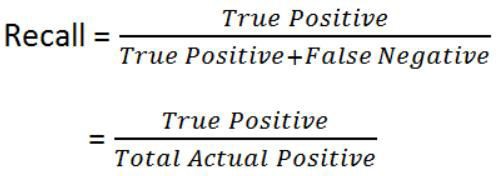
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 20 |
| 1 | 0.91 | 1.00 | 0.95 | 10 |
| 2 | 1.00 | 0.93 | 0.97 | 15 |
| avg / total | 0.98 | 0.98 | 0.98 | 45 |

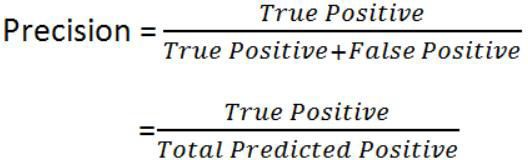
**Basic knowledge**

# Confusion Matrix



**True positives:** data points labelled as positive that are actually positive **False positives:** data points labelled as positive that are actually negative **True negatives:** data points labelled as negative that are actually negative **False negatives:** data points labelled as negative that are actually positive





**Accuracy:** how often is the classifier correct?



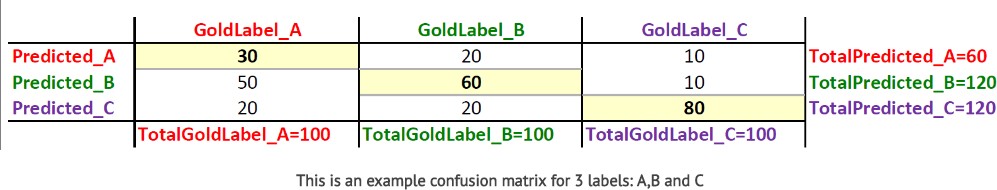
## F1-Score:

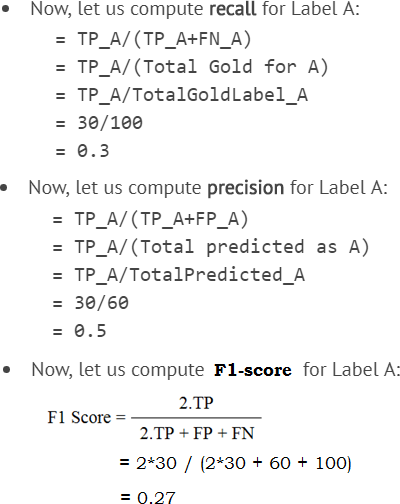


**Support:** Total Predicted of Class.

Support = TP + FN

# Example:





* + Support \_ A = TP\_A + FN\_A

= 30 + (20 + 10)

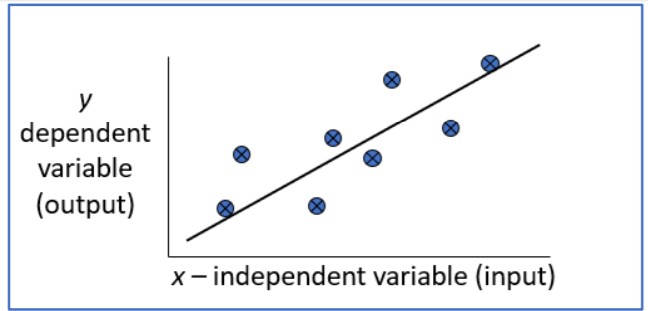
= 60

1. **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

***Locally Weighted Regression Algorithm***

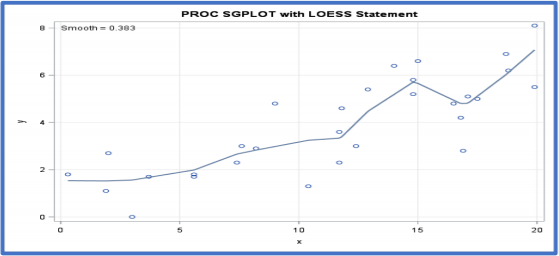
**Regression:**

* + Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous.
  + In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
    - y is called the dependent variable.
    - x is called the independent variable.



#### Loess/Lowess Regression:

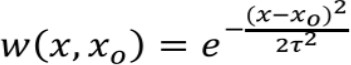
Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



#### Lowess Algorithm:

* + Locally weighted regression is a very powerful nonparametric model used in statistical learning.
  + Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
  + The weights are given by a kernel function (k or w) which can be chosen arbitrarily

##### Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :
5. Determine the value of model term parameter β using :



1. Prediction = x0\*β:

##### Program

import numpy as np

from bokeh.plotting import figure, show, output\_notebook from bokeh.layouts import gridplot

from bokeh.io import push\_notebook

def local\_regression(x0, X, Y, tau):# add bias term x0 = np.r\_[1, x0] # Add one to avoid the loss in

information

X = np.c\_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

# predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial\_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau))

# Weight or Radial Kernal Bias Function

n = 1000

# generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set ( 10 Samples) X :\n",X[1:10]) Y = np.log(np.abs(X \*\* 2 - 1) + .5)

print("The Fitting Curve Data Set (10 Samples) Y

:\n",Y[1:10])

# jitter X

X += np.random.normal(scale=.1, size=n) print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)

print(" Xo Domain Space(10 Samples) :\n",domain[1:10]) def plot\_lwr(tau):

# prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plot = figure(plot\_width=400, plot\_height=400) plot.title.text='tau=%g' % tau

plot.scatter(X, Y, alpha=.3)

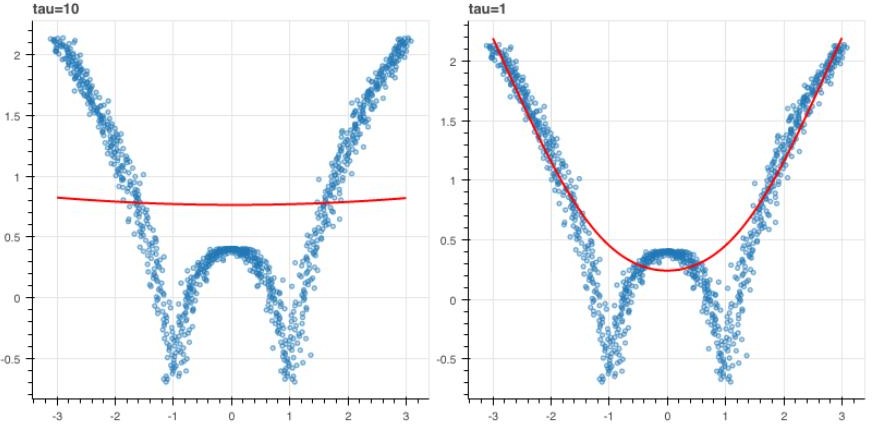
plot.line(domain, prediction, line\_width=2, color='red') return plot

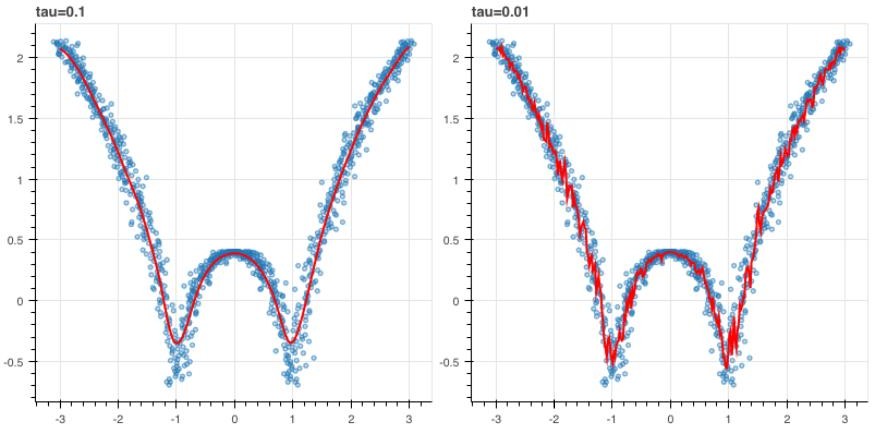
show(gridplot([

[plot\_lwr(10.), plot\_lwr(1.)],

[plot\_lwr(0.1), plot\_lwr(0.01)]]))

##### Output





# -\*- coding: utf-8 -\*- """

Spyder Editor

This is a temporary script file. """

from numpy import \* from os import listdir import matplotlib

import matplotlib.pyplot as plt import pandas as pd

import numpy as np1 import numpy.linalg as np

from scipy.stats.stats import pearsonr

def kernel(point,xmat, k): m,n = np1.shape(xmat)

weights = np1.mat(np1.eye((m))) for j in range(m):

diff = point - X[j]

weights[j,j] = np1.exp(diff\*diff.T/(-2.0\*k\*\*2)) return weights

def localWeight(point,xmat,ymat,k): wei = kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T)) return W

def localWeightRegression(xmat,ymat,k): m,n = np1.shape(xmat)

ypred = np1.zeros(m) for i in range(m):

ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k) return ypred

# load data points

data = pd.read\_csv('tips.csv') bill = np1.array(data.total\_bill) tip = np1.array(data.tip)

#preparing and add 1 in bill mbill = np1.mat(bill)

mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form m= np1.shape(mbill)[1]

# print(m) 244 data is stored in m one = np1.mat(np1.ones(m))

X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE #print(X)

#set k here

ypred = localWeightRegression(X,mtip,0.3) SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1) ax.scatter(bill,tip, color='green')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5) plt.xlabel('Total bill')

plt.ylabel('Tip') plt.show();

1. **Write a program to implement Categorical Encoding, One-hot Encoding**

**Program:**

import pandas as pd

# Sample data

data = {'Category': ['A', 'B', 'A', 'C', 'B', 'A']}

df = pd.DataFrame(data)

# Categorical Encoding

df['Category\_Cat\_Encoded'] = df['Category'].astype('category').cat.codes

print("Categorical Encoding Output:")

print(df)

# One-hot Encoding

one\_hot\_encoded = pd.get\_dummies(df['Category'], prefix='Category\_Onehot')

df = pd.concat([df, one\_hot\_encoded], axis=1)

print("\nOne-hot Encoding Output:")

print(df)

**output:**

Categorical Encoding Output:

Category Category\_Cat\_Encoded

0 A 0

1 B 1

2 A 0

3 C 2

4 B 1

5 A 0

One-hot Encoding Output:

Category Category\_Cat\_Encoded Category\_Onehot\_A Category\_Onehot\_B Category\_Onehot\_C

0 A 0 1 0 0

1 B 1 0 1 0

2 A 0 1 0 0

3 C 2 0 0 1

4 B 1 0 1 0

5 A 0 1 0 0

1. **Develop a program for Bias, Variance, Remove duplicates , Cross Validation**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Sample data

np.random.seed(0)

X = np.random.rand(100, 1) \* 10

y = 2 \* X.squeeze() + np.random.randn(100) # Generating y = 2X + noise

# Add duplicate samples

X = np.append(X, X[:10], axis=0)

y = np.append(y, y[:10], axis=0)

# Remove duplicates

X\_unique, indices = np.unique(X, axis=0, return\_index=True)

y\_unique = y[indices]

# Split data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_unique, y\_unique, test\_size=0.2, random\_state=42)

# Model training

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions

y\_pred\_train = model.predict(X\_train)

y\_pred\_test = model.predict(X\_test)

# Calculate Bias and Variance

train\_mse = mean\_squared\_error(y\_train, y\_pred\_train)

test\_mse = mean\_squared\_error(y\_test, y\_pred\_test)

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

variance = np.mean([((y - np.mean(y\_pred\_test)) \*\* 2) for y in y\_pred\_test])

print(f'Bias: {bias}')

print(f'Variance: {variance}')

# Cross Validation

cross\_val\_scores = cross\_val\_score(model, X\_unique, y\_unique, cv=5, scoring='neg\_mean\_squared\_error')

print(f'Cross Validation Scores: {cross\_val\_scores}')

**OUTPUT:**

Bias: 0.9255274971193731

Variance: 1.388268333621098

Cross Validation Scores: [-0.76131529 -1.21788725 -0.9329079 -0.87077993 -1.34808329]

1. **Exploratory Data Analysis for Classification using Pandas or Matplotlib.**

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

# Sample data (replace with your actual dataset)

data = {

'feature1': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],

'feature2': ['A', 'A', 'B', 'B', 'A', 'C', 'B', 'A', 'C', 'B'],

'target': [1, 1, 0, 0, 1, 0, 0, 1, 0, 1] # Binary classification example

}

df = pd.DataFrame(data)

# Check for missing values (optional)

print(df.isnull().sum())

# Get basic summary statistics

print(df.describe(include='all'))

# Univariate Analysis (Numerical Feature)

plt.hist(df['feature1'], bins=10, edgecolor='black')

plt.xlabel('feature1')

plt.ylabel('Frequency')

plt.title('Distribution of feature1')

plt.grid(True)

plt.show()

# Univariate Analysis (Categorical Feature)

plt.bar(df['feature2'].value\_counts().index, df['feature2'].value\_counts().values)

plt.xlabel('Category')

plt.ylabel('Count')

plt.title('Distribution of feature2')

plt.grid(True)

plt.show()

# Bivariate Analysis (Target vs. Feature)

plt.scatter(df['feature1'], df['target'], c=df['target'], cmap='viridis')

plt.xlabel('feature1')

plt.ylabel('Target')

plt.title('Relationship between feature1 and target')

plt.grid(True)

plt.show()

# Additional Analyses (consider based on data types and classification problem)

# - Correlation matrix (numerical features)

# - Boxplots (categorical features with numerical target)

# - Grouped histograms/distributions

print("Sample output after performing basic EDA for classification:")

**OUTPUT**:

**Output:**

feature1 0

feature2 0

target 0

dtype: int64

feature1 feature2 target

count 10.000000 10 10.000000

unique NaN 3 NaN

top NaN A NaN

freq NaN 4 NaN

mean 55.000000 NaN 0.500000

std 30.276504 NaN 0.527046

min 10.000000 NaN 0.000000

25% 32.500000 NaN 0.000000

50% 55.000000 NaN 0.500000

75% 77.500000 NaN 1.000000

max 100.000000 NaN 1.000000

1. **Write a program to Implement Principle Component Analysis**

**Program:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Standardize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Variance explained by each principal component

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

print("Variance explained by each principal component:")

print("Principal Component 1:", explained\_variance\_ratio[0])

print("Principal Component 2:", explained\_variance\_ratio[1])

# Plotting the PCA results

plt.figure(figsize=(8, 6))

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap=plt.cm.Set1)

plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()

**OUTPUT:**

Variance explained by each principal component:

Principal Component 1: 0.7296244541329986

Principal Component 2: 0.22850761786701768

**APPLY EM ALGORITHM TO CLUSTER A SET OF DATA STORED IN .CSV FILE.USE THE SAME DATA SET FOR CLUSTERING USING K-MEAN ALGORITHM.COMPARE THE RESULT OF TWO ALGORITHM AND**

**ON QUALITY OF CLUSTERING YOU CAN ADD ML/JAVA/ PYTHON LIBRARY CLASSES/API IN THE PROGRAM**

**SOURCE CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans

from sklearn import metrics

# Load the dataset from .csv file

data = pd.read\_csv('your\_dataset.csv')

# Check for missing values and handle them if necessary

if data.isnull().values.any():

data = data.dropna()

# Separate features from the target variable (if applicable)

X = data.drop(columns=['target\_variable']) # Adjust the column name if needed

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply the EM algorithm

em = GaussianMixture(n\_components=2, random\_state=42)

em\_clusters = em.fit\_predict(X\_scaled)

# Apply the K-Means algorithm

kmeans = KMeans(n\_clusters=2, random\_state=42)

kmeans\_clusters = kmeans.fit\_predict(X\_scaled)

# Compare the clustering results using Adjusted Rand Index (ARI)

ari\_em = metrics.adjusted\_rand\_score(data['target\_variable'], em\_clusters)

ari\_kmeans = metrics.adjusted\_rand\_score(data['target\_variable'], kmeans\_clusters)

print("Adjusted Rand Index (ARI) for EM algorithm:", ari\_em)

print("Adjusted Rand Index (ARI) for K-Means algorithm:", ari\_kmeans)

# Visualize the clustering results (you may need to adjust based on the number of features)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=em\_clusters, cmap='viridis')

plt.title('EM Algorithm Clustering')

plt.subplot(1, 2, 2)

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=kmeans\_clusters, cmap='viridis')

plt.title('K-Means Algorithm Clustering')

plt.show()from sklearn import preprocessing scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) #xs.sample(5)

from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=3) gmm.fit(xs)

y\_gmm = gmm.predict(xs) #y\_cluster\_gmm

plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y\_gmm], s=40) plt.title('GMM Classification')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy\_score(y, y\_gmm)) print('The Confusion matrix of EM: ',sm.confusion\_matrix(y, y\_gmm))

RFERENCE LINK

https://deepakdvallur.weebly.com/machine-learning-laboratory.html