Advanced Machine Learning Laboratory [0031]

Computing Lab II (CSE5261), and Information Systems lab II(CSE 5262)

LAB MANUAL II Sem M.Tech (CSE AND CSIS)

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book to every lab session
2. Be in time and adhere to the institution rules and maintain the decorum
3. Must Sign in the log register provided
4. Make sure to occupy the allotted system and answer the attendance

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- Copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Academic honesty is required in all your work. You must solve all programming assignments entirely on your own, except where group work is explicitly authorized. This means you must not take, neither show, give or otherwise allow others to take your program code, problem solutions, or other work.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
    - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs, and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
- The exercises for each week are divided into three sets:
    - Solved exercise
    - Lab exercises - to be completed during lab hours

- Additional Exercises - to be completed outside the lab or in the lab to enhance the skill

- Questions for lab tests and examination are not necessarily limited to the questions in the manual but may involve some variations and/or combinations of the questions.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

SCHEME OF EVALUATION

Subject	Marks
Week 1 – Week 8 Lab exercises	8x5 = 40 Marks
Mini-project with publishable work in IEEE/Srpinger conference/journal format	20 Marks
Final Lab exam	40 Marks
Total	100 Marks

Course Objectives

(1) To determine the correct working of a given algorithm.
(2) To pre-process the data for a given learning algorithm.
(3) To implement the various machine learning algorithms.
(4) To enhance Python skills.

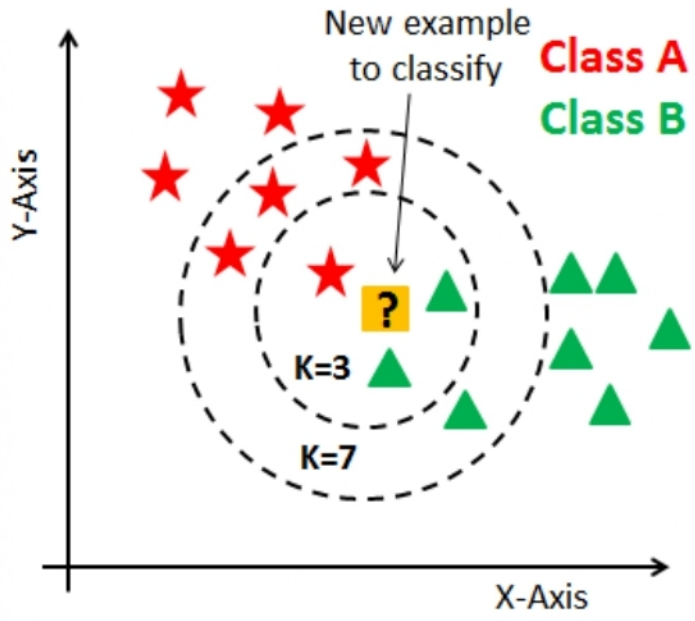
Course Outcomes:

After completing the course, the student will be able to:
CO1: Implement machine learning problems domains and algorithms.
CO2: Apply learning techniques like Neural Networks, Find S and Candidate Elimination Algorithms.
CO3: Apply text classification-based Bayes models for problem solving.

LAB NO. 01 IMPLEMENTATION OF K-NN on Iris Dataset using Google Colab

The class of an unknown instance is computed using the following steps:

1. The distance between the unknown instance and all other training instances is computed.
2. The k nearest neighbors are identified.
3. The class labels of the k nearest neighbors are used to determine the class label of the unknown instance by using techniques like majority voting.



k-NN classification example (Image Source)

1. Distance Metrics: Euclidean, Normalized Euclidean and Cosine Similarity
2. k-values: 1, 3, 5, and 7

LAB NO 02: IMPLEMENTATION OF FIND-S ALGORITHM

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 a_i in h

   If the constraint a_i is satisfied by x

   Then do nothing

   Else replace a_i in h by the next more general constraint that is satisfied by x
3. 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', '?', '?']
```

Exercise Questions:

1)Demonstrate the working of the program for different data provided by the teacher.

LAB NO : 03 DECISION TREE IMPLEMENTATION

3. 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, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = v_i
    - Let *Examples* $_{vi}$, be the subset of Examples that have value v_i 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

ENTROPY:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

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

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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

```
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
```

Exercise Questions:

1)Demonstrate the working of the program for different data provided by the teacher.

Note: Students need to form a team of maximum 2 students and submit a base paper for the mini-project implementation.

LAB NO: 04 ANN IMPLEMENTATION USING BACKPROPAGATION LEARNING ALGORITHM

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

BACKPROPAGATION Algorithm

BACKPROPAGATION (*training_example*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $(\vec{x}, \vec{t})$, *where* $(\vec{x})$ *is the vector of network input values,* $(\vec{t})$ *and is the vector of target network output values.*
η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*
The input from unit i into unit j is denoted x_{ji}, *and the weight from unit i to unit j is denoted* w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

- For each $(\vec{x}, \vec{t})$, in training examples, Do

  Propagate the input forward through the network:
  1. Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

  Propagate the errors backward through the network:

2. For each network output unit **k**, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

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]]
```

LAB NO: 05 IMPLEMENTATION OF NAÏVE-BAYESIAN CLASSIFIER

5. 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.

```
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 = {} # 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

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

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

confusion matrix is as
follows [[17 0 0]
[0 17 0]
[0 0 11]]
Accuracy metrics

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	1.00	1.00	1.00	17
2	1.00	1.00	1.00	11
avg / total	1.00	1.00	1.00	45

Exercise Questions:

1)Demonstrate the working of the program for different data provided by the teacher.

LAB NO 06: IMPLEMENTATION OF TEXT CLASSIFIER USING NAÏVE-BAYES

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 $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

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 $P(v_j)$ and $P(w_k|v_j)$ probability terms*

   - For each target value v_j in V do
     - $docs_j$ ← the subset of documents from *Examples* for which the target value is vj
     - $P(v_j)$ ← $|docs_j| / |\text{Examples}|$
     - $Text_j$ ← a single document created by concatenating all members of $docs_j$
     - n ← total number of distinct word positions in $Text_j$
     - for each word w_k in *Vocabulary*

       - n_k ← number of times word $\boldsymbol{w_k}$ occurs in $Text_j$
       - $P(w_k|v_j)$ ← $(n_k + 1) / (n + |Vocabulary|)$

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 V_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\text{argmax}}\, P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

Examples:

	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
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

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


#splitting the dataset into train and test data
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
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')

print(cv.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_na
mes())

# Training Naive Bayes (NB) classifier on training data.
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
print('\n Accuracy of the classifer is',
metrics.accuracy_score(ytest,predicted))

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


print('\n The value of Precision' ,
metrics.precision_score(ytest,predicted))
```

The value of Precision 1.0

The value of Recall 0.3333333333333333

```
print('\n The value of Recall' ,
metrics.recall_score(ytest,predicted))
```

Output:

The dimensions of the dataset (18, 2)

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', 'awesome', 'bad', 'beers', 'boss', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'stay', 'stuff', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with']

Accuracy of the classifier is 0.6

Confusion matrix

[[2 0]

[2 1]]

The value of Precision 1.0

The value of Recall 0.3333333333333333

Exercise Questions:

1)Demonstrate the working of the program for different data provided by the teacher.

LAB NO: 07 IMPLEMENTATION OF K-MEANS CLUSTERING ALGORITHM

8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

PROGRAM:-

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
#import matplotlib inline

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

#colormap = np.array(['red', 'lime', 'black'])

# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(X)
# This is what KMeans thought
model.labels_

# View the results

# Set the size of the plot
plt.figure(figsize=(14,7))
```

```
# Create a colormap
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')

# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
#print('The accuracy score : ',sm.accuracy_score(y, model.labels_))
#sm.confusion_matrix(y, model.labels_)

predY = np.choose(model.labels_, [0, 1, 2]).astype(np.int64)
```

```
print (predY)

#colormap = np.array(['red', 'lime', 'black'])
# Plot Orginal
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
# Plot Predicted with corrected values
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[predY], s=40)
plt.title('K Mean Classification')

print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))

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_cluster_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm], s=40)
plt.title('GMM Classification')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_cluster_gmm))
```

LAB NO 08: IMPLEMENTATION OF K-NN ALGORITHM

9. 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.

Training algorithm:

- For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

  - Where, $f(x_i)$ 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

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

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

Exercise Questions:

1)Demonstrate the working of the program for different data provided by the teacher.

Week 9- Week 12: Mini Project

Students need to work on the mini-project and submit the report in the form of conference/journal paper to be published. Note: we do not need regular report. Demo of mini-project will be in the Week 12 lab.

Week 13: Test

REFERENCE BOOKS:

1. Machine Learning – Tom M. Mitchell, - MGH, 2013.
2. Richard o. Duda, Peter E. Hart and David G. Stork, pattern classification, John Wiley & Sons Inc., 2001.
3. Ethem Alpaydin, "Introduction to Machine Learning", Prentice Hall of India, 2005
4. Albon C. Machine learning with python cookbook: Practical solutions from preprocessing to deep learning. " O'Reilly Media, Inc."; 2018.
5. Stephen Marsland, "Machine Learning –An Algorithmic Perspective", CRC Press, 2009.
6. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, by Aurelien Geron, OREILLY (2017).