lab 1:
Implementation of knn using iris dataset
--

```
# -*- coding: utf-8 -*-
"""iris.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1IIKPLzjQdFZP1LZnDyF4QwAqrEMvLU0E
"""

import math
from pprint import pprint
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

from google.colab import files
uploaded = files.upload()
file_columns =
['sepal_len','sepal_width','petal_len','petal_width','class']

import io
file_columns =
['sepal_len','sepal_width','petal_len','petal_width','class']
data = pd.read_csv(io.BytesIO(uploaded['iris.data']), header=None,
names=file_columns)
data.head()

data = data.sample(frac=1).reset_index(drop=True)
data['seq'] = data.index
data.head()

dev_size = int(data.shape[0]*0.75)
test_size = int(data.shape[0]*0.25)

# Take first 75% of the data as dev set
dev = data[:dev_size]

# Take last 25% of the data as test set
test = data[test_size:]

def get_euclidean(row1, row2):
    return math.sqrt(sum([(x1-x2)**2 for x1,x2 in zip(row1,row2)]))

def get_cosine_sim(row1, row2):
    return math.acos(
        sum([x1*x2 for x1,x2 in zip(row1,row2)])/(sum([i**2 for i in
row1]) * sum([i**2 for i in row2]))
```

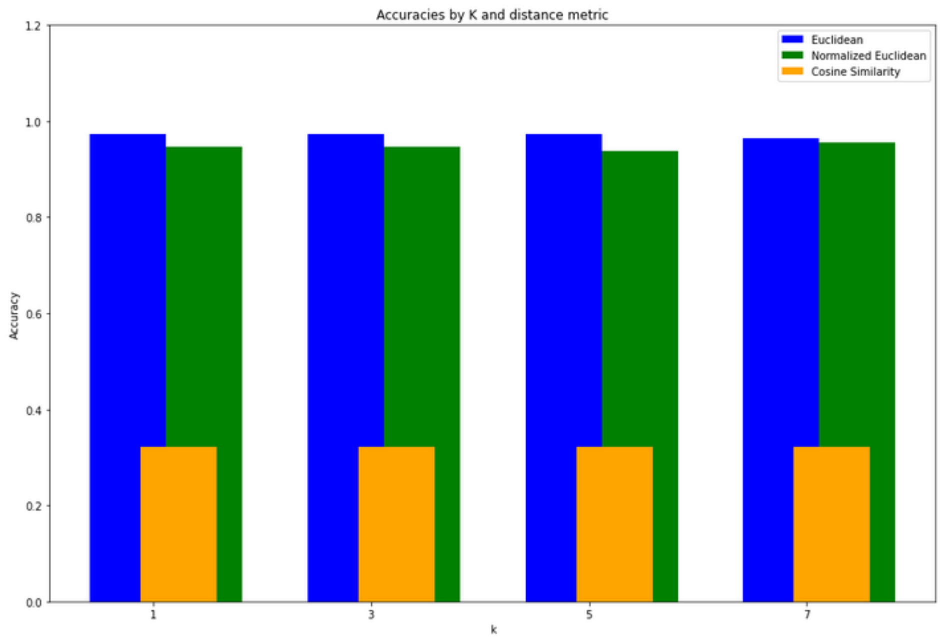
```
    )

ndev = pd.DataFrame(columns=['sepal_len',
'sepal_width','petal_len','petal_width', 'class', 'seq'])

ndev['class'] = dev['class'].copy()
ndev['seq'] = dev['seq'].copy()

ndev['sepal_len'] = dev['sepal_len'].apply(
    lambda x: (x-dev['sepal_len'].min())/(dev['sepal_len'].max()-
dev['sepal_len'].min()))

ndev['sepal_width'] = dev['sepal_width'].apply(
    lambda x: (x-dev['sepal_width'].min())/(dev['sepal_width'].max()-
dev['sepal_width'].min()))

ndev['petal_len'] = dev['petal_len'].apply(
    lambda x: (x-dev['petal_len'].min())/(dev['petal_len'].max()-
dev['petal_len'].min()))

ndev['petal_width'] = dev['petal_width'].apply(
    lambda x: (x-dev['petal_width'].min())/(dev['petal_width'].max()-
dev['petal_width'].min()))

dev2 = dev.values
ndev2 = ndev.values
eud = []
cosine_sim = []
neud = []
l = len(dev2)
for i in range(l):
    eu_distance = []
    cos_sim = []
    neu_distance = []
    for j in range(l):
        if(i!=j):
            index = dev2[j][5]
            nindex = ndev2[j][5]
            ed = get_euclidean(dev2[i][:-2], dev2[j][:-2])
            cs = get_cosine_sim(dev2[i][:-2], dev2[j][:-2])
            neu = get_euclidean(ndev2[i][:-2], ndev2[j][:-2])

            eu_distance.append((ed, index))
            cos_sim.append((cs, index))
            neu_distance.append((neu, nindex))

    eu_distance.sort(key= lambda x: x[0])
    cos_sim.sort(key= lambda x: x[0])
    neu_distance.sort(key= lambda x: x[0])

    eu_distance = [i[1] for i in eu_distance]
    cos_sim = [i[1] for i in cos_sim]
    neu_distance = [i[1] for i in neu_distance]
```

```
    eud.append(eu_distance)
    cosine_sim.append(cos_sim)
    neud.append(neu_distance)


dev['euclidean'] = eud
dev['cosine_sim'] = cosine_sim
dev['n_euclidean'] = neud

def get_nearest(row, distance_measure,k):
    return row[distance_measure][:k]

def get_dominant_class(df, neighbors):
    classes = df[df['seq'].isin(neighbors)]['class']
    return classes.value_counts().index[0]

k = 1
hyper_params = []
acc = {1: {}, 3:{}, 5:{}, 7:{}}

while k <= 7:
    dev['eud_{}'.format(k)] = dev.apply(lambda x: get_nearest(x,
'euclidean',k), axis=1)
    dev['cosim_{}'.format(k)] = dev.apply(lambda x: get_nearest(x,
'cosine_sim',k), axis=1)
    dev['neud_{}'.format(k)] = dev.apply(lambda x: get_nearest(x,
'n_euclidean',k), axis=1)

    dev['eud_{}_class'.format(k)] = dev['eud_{}'.format(k)].apply(lambda
row: get_dominant_class(dev, row))
    dev['cosim_{}_class'.format(k)] =
dev['cosim_{}'.format(k)].apply(lambda row: get_dominant_class(dev, row))
    dev['neud_{}_class'.format(k)] =
dev['neud_{}'.format(k)].apply(lambda row: get_dominant_class(dev, row))

    hyper_params.append('eud_{}_class'.format(k))
    hyper_params.append('cosim_{}_class'.format(k))
    hyper_params.append('neud_{}_class'.format(k))

    acc[k]['eud'] =
dev[dev['class']==dev['eud_{}_class'.format(k)]].shape[0]/dev.shape[0]
    acc[k]['cosine'] =
dev[dev['class']==dev['cosim_{}_class'.format(k)]].shape[0]/dev.shape[0]
    acc[k]['neud'] =
dev[dev['class']==dev['neud_{}_class'.format(k)]].shape[0]/dev.shape[0]

    k+=2

cols = ['class'] + hyper_params
dev[cols].head()

pprint(acc)

labels = [1,3,5,7]
```

```
eud_acc = [acc[i]['eud'] for i in list(acc)]
cos_acc = [acc[i]['cosine'] for i in list(acc)]
neud_acc = [acc[i]['neud'] for i in list(acc)]

width = 0.35
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(15,10))
eu_bar = ax.bar(x - width/3, eud_acc, width, label='Euclidean',
color='blue')
neu_bar = ax.bar(x + width*2/3, neud_acc, width, label='Normalized
Euclidean', color='green')
cosine_bar = ax.bar(x + width/3, cos_acc, width, label='Cosine
Similarity', color='orange')

ax.set_ylabel('Accuracy')
ax.set_ylim(0,1.2)
ax.set_title('Accuracies by K and distance metric')
ax.set_xlabel("k")
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc='best')
plt.show()

test['seq'] = test.index
test2 = test.values
test_eud = []
l = len(test)
for i in range(l):
    test_eu_distance = []
    for j in range(len(dev)):
        index = dev2[j][5]
        ed = get_euclidean(test2[i][:-2], dev2[j][:-2])
        test_eu_distance.append((ed, index))

    test_eu_distance.sort(key= lambda x: x[0])
    test_eu_distance = [i[1] for i in test_eu_distance]
    test_eud.append(test_eu_distance)

test['euclidean'] = test_eud

test['eu'] = test.apply(lambda x: get_nearest(x, 'euclidean',3), axis=1)
test[file_columns+['eu']].head()

test['eu_class'] = test['eu'].apply(lambda row: get_dominant_class(dev,
row))
test[file_columns+['eu_class']].head()

test_acc = test[test['class']==test['eu_class']].shape[0]/test.shape[0]
print('Test Accuracy: {:2.4f}%'.format(test_acc*100))
```

----

output:-




```
test_acc = test[test['class']==test['eu_class']].shape[0]/test.shape[0]
print('Test Accuracy: {:2.4f}%'.format(test_acc*100))
```

Test Accuracy: 98.2301%

```
----------------------------------------------------------------------------
--------------------------------------------------
```

lab2:Implementation of find-s Algorithm

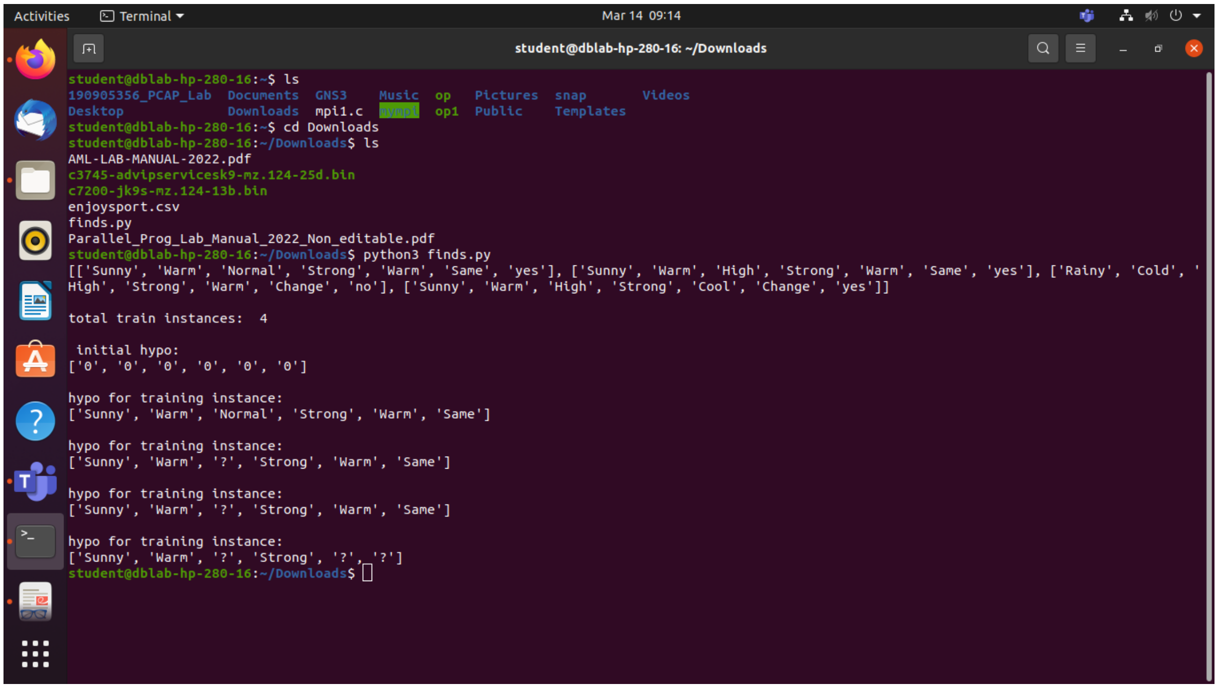
```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
      for row in csv.reader(csvfile):
            a.append(row)
      print(a)

print("\ntotal train instances: ", len(a))

num_attribute = len(a[0])-1

print("\n initial hypo: ")
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("\nhypo for training instance: ")
      print(hypothesis)
---------
Dataset:-
-----------
```

Clipboard Font Alignment

A1 Sunny

	A	B	C	D	E	F	G
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
5							

output:-



--

Lab3:

Decision Tree Implementation

```
import pandas as pd
import math
import numpy as np
data = pd.read_csv("3-dataset.csv")
features = [feat for feat in data]
features.remove("answer")
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
```

```
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
```

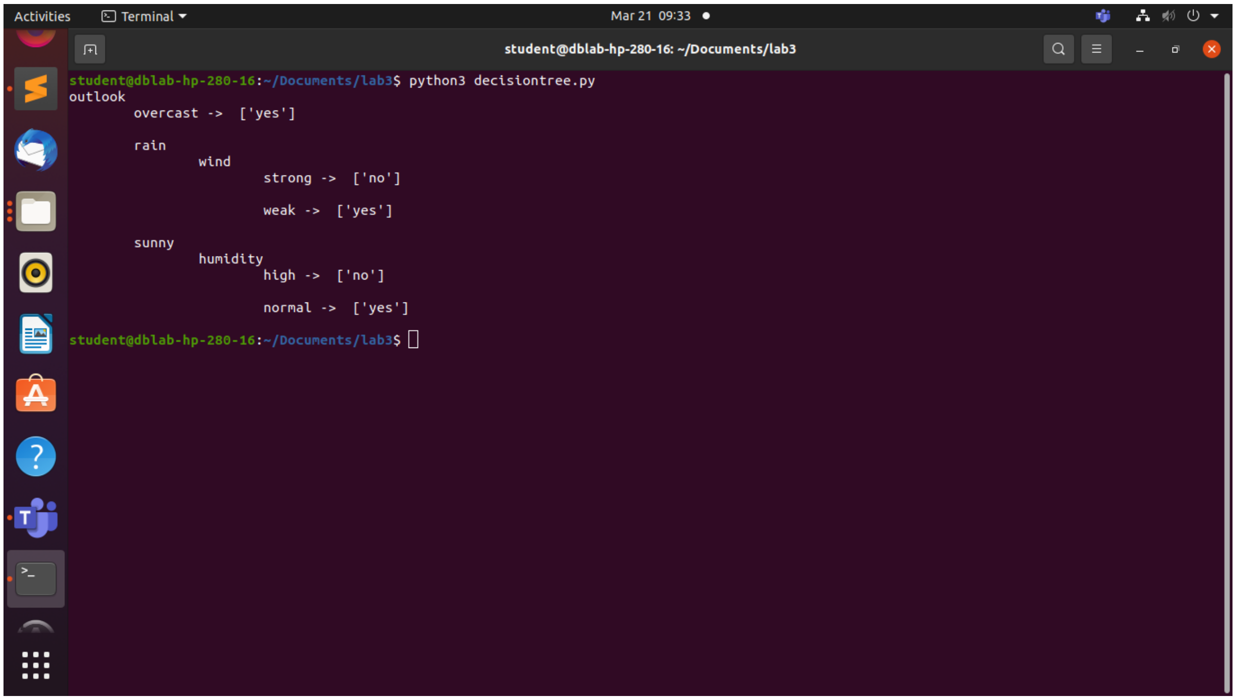
```
    return root
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
root = ID3(data, features)
printTree(root)
```

Dataset:-

	A	B	C	D	E
1	outlook	temperatu	humidity	wind	answer
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no
16					
17					

----

output:-



----

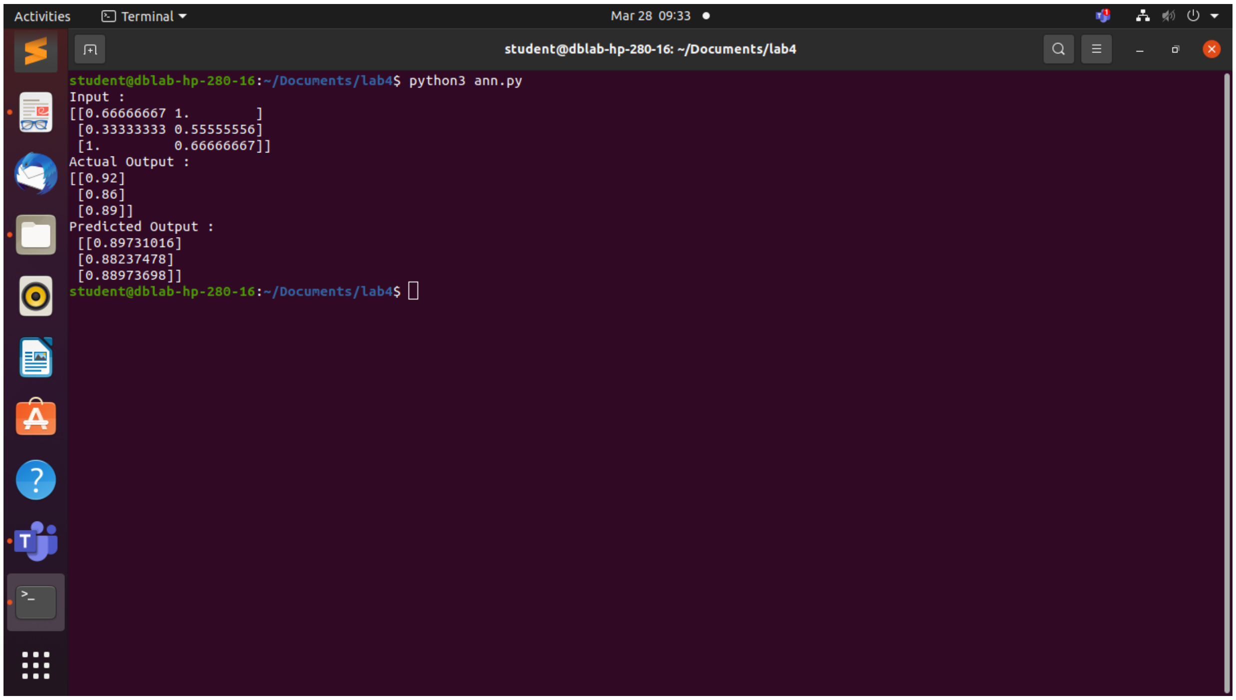
lab4:-Ann implementation using backpropogation Algorithm:

```
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)
y = y/100

def sigmoid(x):
```

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

def derivatives_sigmoid(x):
      return x*(1-x)

epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout = np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
      hinpl = np.dot(X,wh)
      hinp = hinpl + bh
      hlayer_act = sigmoid(hinp)
      outinpl = np.dot(hlayer_act,wout)
      outinp = outinpl + bout
      output = sigmoid(outinp)
      EO = y-output
      outgrad = derivatives_sigmoid(output)
      d_output = EO*outgrad
      EH = d_output.dot(wout.T)
      hiddengrad = derivatives_sigmoid(hlayer_act)
      d_hiddenlayer = EH*hiddengrad
      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:-



----

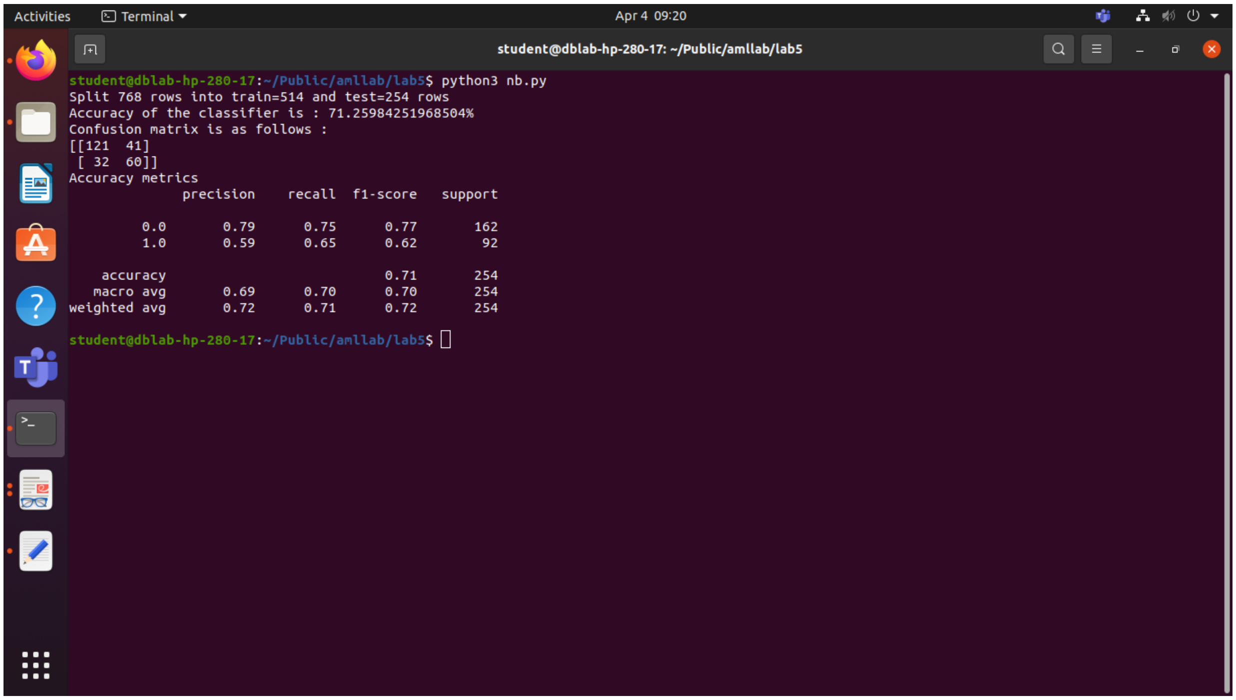
lab5:-
Implementation of nave-bayesian classifier

```
import csv
import random
import math
from sklearn.metrics import confusion_matrix, classification_report

def loadcsv(filename):
      lines = csv.reader(open(filename, "r"));
      dataset = list(lines)
      for i in range(len(dataset)):
            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:
            index = random.randrange(len(copy));
            trainset.append(copy.pop(index))
      return [trainset, copy]

def separatebyclass(dataset):
      separated = {}
```

```
    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():
        summaries[classvalue] = summarize(instances)
    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 = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]
            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))
      y_true = []
      for i in range(len(testset)):
            y_true.append(testset[i][-1])
      print("Confusion matrix is as follows :")
      print(confusion_matrix(y_true,predictions))
      print("Accuracy metrics")
      print(classification_report(y_true,predictions))

main()
```

----
output:-



------

lab6:-

Implementation of text classifiefr using nave bayes

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

# In[1]:


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("C:/Users/sinch/Downloads/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 the words or Tokens in the text documents
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_names())

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


# In[ ]:
```

----
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', 'an', 'and', 'awesome', 'bad', 'beers', 'boss', 'dance', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have',
'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwic
h', 'sick', 'stay', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went',
'what', 'will']

 Accuracy of the classifier is 0.8

 Confusion matrix
[[3 0]
 [1 1]]

 The value of Precision 1.0

 The value of Recall 0.5
```

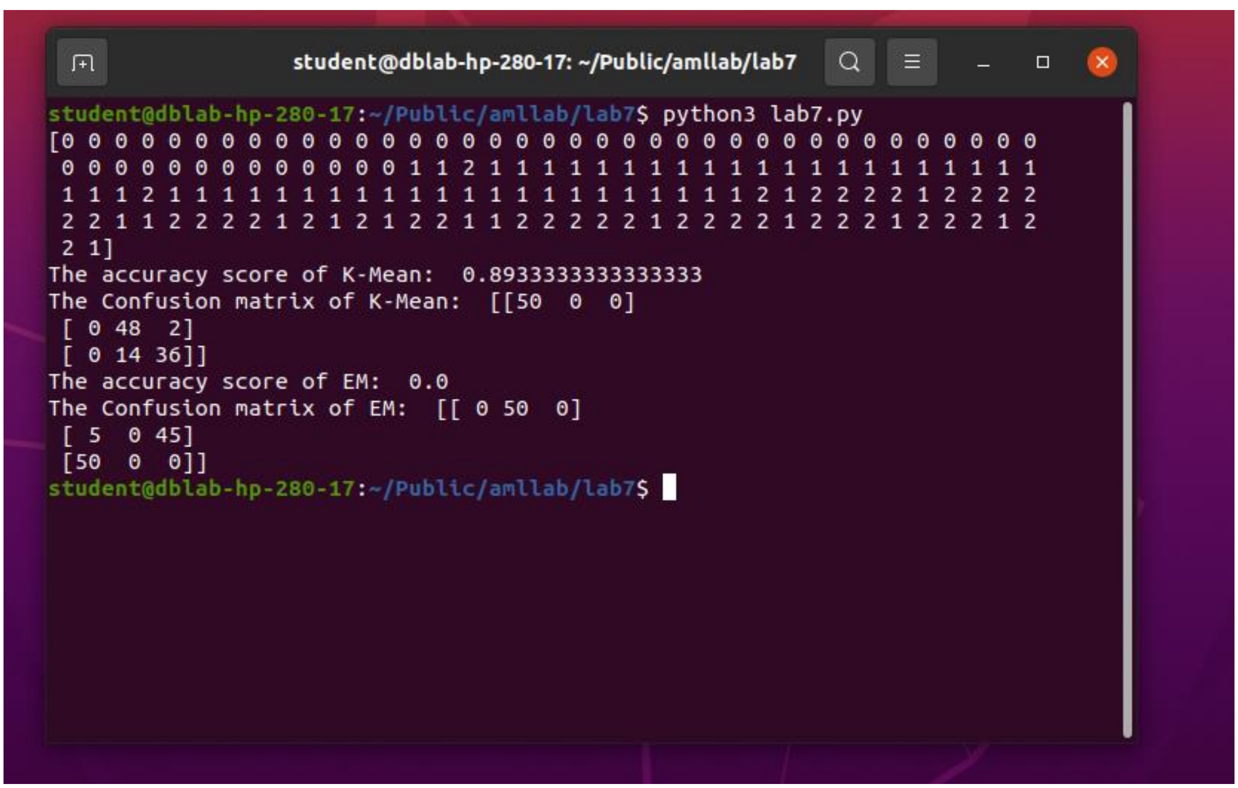
--
lab7:-

Implementation of k-means clustering Algorithm

```
# -*- coding: utf-8 -*-
"""210913006_AML_Lab7.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1HJ6dID8F8V-uP9jheemS0-
UgjFtof5KO
"""

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
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']
model = KMeans(n_clusters=3)
model.fit(X)
model.labels_
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_],
s=40)
plt.title('K Mean Classification')
plt.figure(figsize=(14,7))
predY = np.choose(model.labels_, [0, 1, 2]).astype(np.int64)
print (predY)
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
```

```
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 matrix of 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)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture (n_components=3)
gmm.fit(xs)
y_cluster_gmm = gmm.predict (xs)
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))
```

---
output:-



----
lab8:-
Implementation of knn Algorithm:-

```
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-Virginica')
 print(y)
 """Splits the dataset into 70% train data and 308 test data. This means
out of 150 records and the training set will 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 fiscore 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:-

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]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
```

```
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
```

```
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class:0-Iris-Setosa, 1-Iris-Versicolour, 2-Virginica
[00000000000000000000000000000000000000 0000000000000111111111111111111111111
 11111111111111111111111111222222222222 2222222222222222222222222222222222222
 2 2]

Confusion Matrix
[[12  0  0]
 [ 014 0]
 [ 0  0 19]]
 Accuracy Metrics
              precision    recall  f1-score   support
           0       1.00      1.00      1.00        12

           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        19
    accuracy                           1.00        45

   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```