Data Mining and Web algorithm

Lab Assignment 5:

[04 – 09 Apr, 2022]

Patil Amit Gurusidhappa
19104004
B11

Q1: Q1: Write a general program to compute the Euclidean distance between given users based on movie rating data and put them into two groups.

```
import pandas as pd
import numpy as np
```

```
df=pd.read_csv("movie_dataset.csv");
df
```

	Users	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7
0	1	2	4	3	3	2	3	3
1	2	1	4	3	2	1	5	3
2	3	4	3	3	3	4	1	5
3	4	1	2	4	3	1	2	1
4	5	0	3	5	5	0	2	2
5	6	2	3	1	1	2	2	2
6	7	5	5	2	2	5	4	4
7	8	2	1	0	2	2	3	3
8	9	3	2	2	2	3	4	2

```
def toSeries(df):
    ser1 = df.iloc[:,3]

    print("\nLast column as a Series:\n")
    print(ser1)
```

```
    # Checking type
    print(type(ser1))
```

```
seriesData=[];
for index,data in df.iterrows():
    l1=[]
    flag=False;
    for d in data:
#         print(d)
        if(flag):
            l1.append(d)
        flag=True
#     print(l1)
    seriesData.append(l1)

#     seriesData.append(pd.Series(l1))
#     print(data)
#     toSeries(data)
#     print(np.linalg.norm(pd.Series(l1)-pd.Series(l1)))
#     break
# for i in seriesData:
#     print(i)
#     break
print(seriesData)
```

```
[[2, 4, 3, 3, 2, 3, 3], [1, 4, 3, 2, 1, 5, 3], [4, 3, 3, 3, 4, 1,
5], [1, 2, 4, 3, 1, 2, 1], [0, 3, 5, 5, 0, 2, 2], [2, 3, 1, 1, 2,
2, 2], [5, 5, 2, 2, 5, 4, 4], [2, 1, 0, 2, 2, 3, 3], [3, 2, 2, 2,
3, 4, 2], [3, 2, 1, 4, 4, 5, 1], [3, 4, 4, 3, 5, 1, 4], [4, 3, 1,
4, 1, 2, 1], [5, 5, 2, 3, 2, 0, 0], [1, 1, 4, 3, 0, 2, 2], [2, 2,
3, 3, 2, 1, 5], [0, 0, 4, 4, 1, 4, 2], [2, 0, 3, 5, 4, 3, 4], [1,
2, 3, 1, 2, 3, 4], [4, 5, 3, 2, 4, 5, 3], [1, 2, 4, 0, 3, 1, 2]]
```

```
for i in range(len(seriesData)-1):
```

```
    print(seriesData[i])
    print(seriesData[i+1])
    sum=0;
    for j in seriesData[i]:
        for k in seriesData[i+1]:
            sum+=(k-j)**2
    print(np.sqrt(sum))
    print("\n")
#
print(np.linalg.norm(pd.Series(seriesData[i])-pd.Series(seriesData[i+1])));
#           np.
```

```
Output exceeds the size limit. Open the full output data in a text editor
[2, 4, 3, 3, 2, 3, 3]
[1, 4, 3, 2, 1, 5, 3]
10.723805294763608


[1, 4, 3, 2, 1, 5, 3]
[4, 3, 3, 3, 4, 1, 5]
13.2664991614216
```

```
[4, 3, 3, 3, 4, 1, 5]
[1, 2, 4, 3, 1, 2, 1]
14.247806848775006


[1, 2, 4, 3, 1, 2, 1]
[0, 3, 5, 5, 0, 2, 2]
15.652475842498529


[0, 3, 5, 5, 0, 2, 2]
[2, 3, 1, 1, 2, 2, 2]
14.696938456699069

...
[1, 2, 4, 0, 3, 1, 2]
17.233687939614086
```

Q2: Consider the given dataset and implement Naïve Bayes using python. Find the classification
accuracy.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
```

```
data = pd.read_csv("Breast_cancer_data.csv")
data.head(10)
```

0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
5	12.45	15.70	82.57	477.1	0.12780	0
6	18.25	19.98	119.60	1040.0	0.09463	0
7	13.71	20.83	90.20	577.9	0.11890	0
8	13.00	21.82	87.50	519.8	0.12730	0
9	12.46	24.04	83.97	475.9	0.11860	0

```
data = data[["mean_radius", "mean_texture", "mean_smoothness",
"diagnosis"]]
data.head(10)
```


```
data = data[["mean_radius", "mean_texture", "mean_smoothness",
"diagnosis"]]
data.head(10)
```


Calculate P(Y=y) for all possible y

```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

Calculate P(X=x|Y=y) using Gaussian dist

```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *
np.exp(-((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

Calculate P(X=x1|Y=y)P(X=x2|Y=y)...P(X=xn|Y=y) * P(Y=y) for all y and find the maximum

```
def naive_bayes_gaussian(df, X, Y):
```

```
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df,
features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="diagnosis")

from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

```
[[36  4]
 [ 0 74]]
0.9736842105263158
```

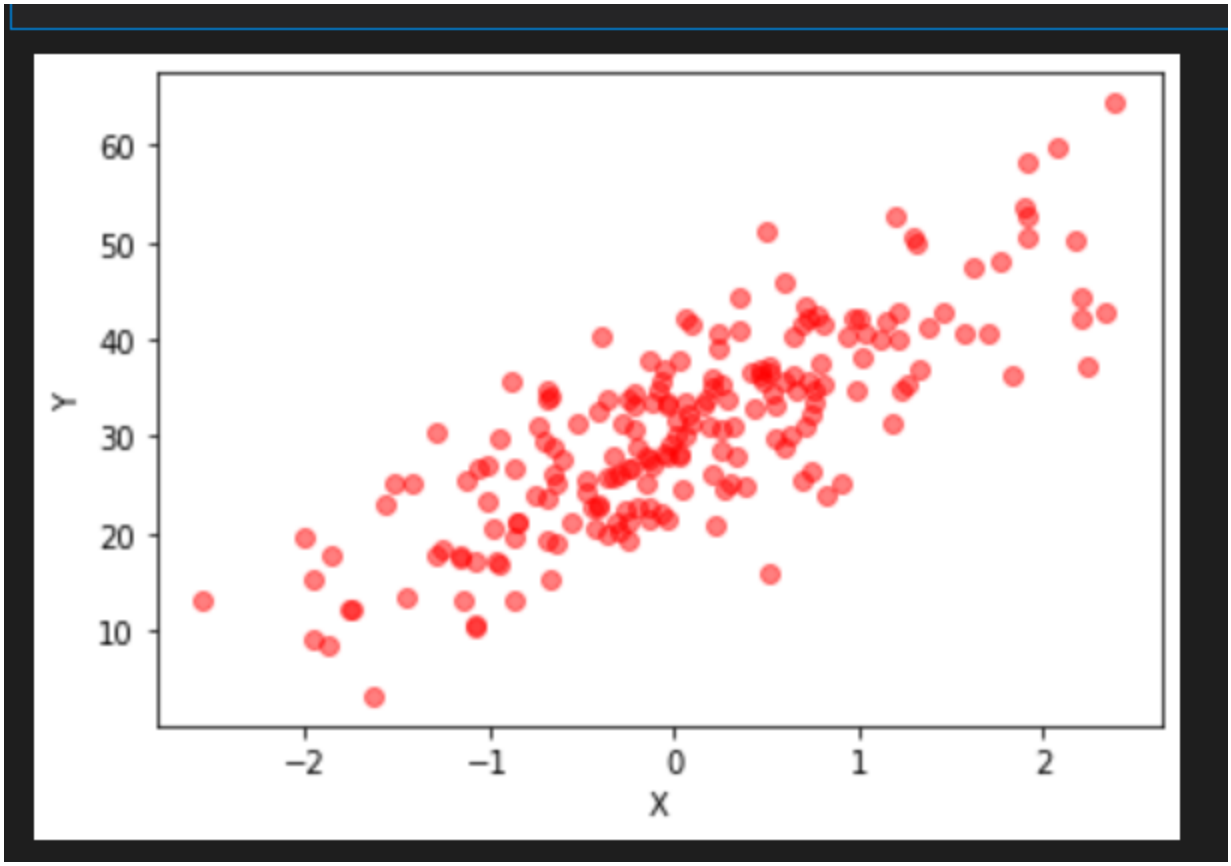
Q3: Consider IRIS dataset and perform the following operations.
a) Normalize the data values if necessary.
b) Use 80% training data and 20% testing data.
c) Use WEKA data mining tool for result statistics.
d) Write a python program for finding the KNN and Naïve Bayes accuracy.
e) Compare the accuracies obtained and give the suitable interpretation.

Q4: The following table gives information on ages and cholesterol levels for a random sample of 10 men.

a) WAP in python to implement linear regression of cholesterol level on age
b) Predict the cholesterol level of a 60-year-old man.

```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=200, n_features=1, n_informative=1,
noise=6, bias=30, random_state=200)
m = 200
```

```
from matplotlib import pyplot as plt
plt.scatter(X,y, c = "red",alpha=.5, marker = 'o')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Linear Model

```
import numpy as np
# hypothesis
def h(X,w):
    return (w[1]*np.array(X[:,0])+w[0])
```

Cost Function

```
# squred error
def cost(w,X,y):
    return (.5/m) * np.sum(np.square(h(X,w)-np.array(y)))
```

Gradient Descent

```
def grad(w,X,y):
    g = [0]*2
    g[0] = (1/m) * np.sum(h(X,w)-np.array(y))
    g[1] = (1/m) * np.sum((h(X,w)-np.array(y))*np.array(X[:,0]))
    return g
```

```
def descent(w_new, w_prev, lr):
    print(w_prev)
    print(cost(w_prev,X,y))
    j=0
    while True:
        w_prev = w_new
        w0 = w_prev[0] - lr*grad(w_prev,X,y)[0]
        w1 = w_prev[1] - lr*grad(w_prev,X,y)[1]
        w_new = [w0, w1]
        print(w_new)
        print(cost(w_new,X,y))
        if (w_new[0]-w_prev[0])**2 + (w_new[1]-w_prev[1])**2 <=
pow(10,-6):
            return w_new
        if j>500:
            return w_new
        j+=1
```

```
w = [0,-1]
w = descent(w,w,.1)
print(w)
```

```
[0, -1]
540.5360663843456
[3.0956308633447547, 0.11442770988081663]
437.91139336428444
[5.873446610978822, 1.10234542813828 54]
355.5039050187037
[8.366165526017987, 1.97786577832476 02]
289.3267499184995
[10.603129563187093, 2.753547324958939]
236.1799750745718
[12.610653489037027, 3.440564026385428]
193.49509649539323
```