**Artificial Intelligence Lab Evaluation Assignment 2**

**COE 9**

**Q1.** Create a dataset (.csv) of students having following attributes: Written score (out of 100), Personal interview Score (out of 50), Group discussion score (out of 150), Qualified for IIM (for qualified students mention the Name of IIM; otherwise mention "Not qualified”).

1. Implement KNN to predict whether following students are qualified for an IIM or not? Mention IIM name in case of Qualified otherwise predicted result should display not qualified

* 80% scored in Written test, 70% scored in Personal interview, and 70% scored in Group discussion.
* 50% scored in Written test, 80% scored in Personal interview, and 40% scored in Group discussion.

1. Compare the accuracy of prediction of KNN with Bayesian learning model and show it using plots.
2. Run your program for different values of K and find its best value. Also observe the effect on train test ratio on the best value of K.

**CODE Cells with output:**

**# Creating a dataset**

In [31]:

*# Kulpreet\_q1\_30/11/20*

**import** random **as** r

*# campus=['IIM-DL','IIM-GJ'] # possible IIM campuses*

fp**=**open('data\_1k.csv','w') *# Open the file in writing mode*

fp.write('written(100),pi(50),gd(150),qualified\n')

**for** i **in** range(10000):

written**=**r.randint(0,100)

pi**=**r.randint(0,50)

gd**=**r.randint(0,150)

tot**=**0.33**\***written**+**0.17**\***pi**+**0.50**\***gd *# merit qualification factor*

**if** tot**>**90:

college**=** r.randint(1,2) *# r.choice(campus)*

**else**:

college**=** 0 *# 'not qualified'*

stu**=**'%d,%d,%d,%d\n'**%**(written,pi,gd,college)

fp.write(stu) *# Writing to the file line by line*

​

fp.close()

print ('Done! \n Open the file to view the dataset.')

Done!

Open the file to view the dataset.

In [32]:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

In [33]:

df **=** pd.read\_csv('data\_1k.csv')

df.head()

Out[33]:

|  | **written(100)** | **pi(50)** | **gd(150)** | **qualified** |
| --- | --- | --- | --- | --- |
| **0** | 14 | 27 | 96 | 0 |
| **1** | 49 | 18 | 97 | 0 |
| **2** | 96 | 10 | 71 | 0 |
| **3** | 45 | 47 | 61 | 0 |
| **4** | 86 | 18 | 130 | 2 |

In [34]:

x **=** df[['written(100)', 'pi(50)', 'gd(150)']]

y **=** df['qualified']

print(x.shape, y.shape)

(10000, 3) (10000,)

**# Scaling the Input Data**

In [35]:

stu11**=** pd.Series(data**=**{'written(100)': 50, 'pi(50)' : 40, 'gd(150)' : 60})

x **=** x.append(stu11, ignore\_index**=True**)

stu22**=** pd.Series(data**=**{'written(100)': 80, 'pi(50)' : 35, 'gd(150)' : 105})

x **=** x.append(stu22, ignore\_index**=True**)

In [36]:

**from** sklearn.preprocessing **import** StandardScaler

sc **=** StandardScaler()

x **=** sc.fit\_transform(x)

np.set\_printoptions(suppress **=** **True**)

x

Out[36]:

array([[-1.24013672, 0.1528871 , 0.48910885],

[-0.04893826, -0.45567321, 0.51197282],

[ 1.5506711 , -0.9966157 , -0.08249043],

...,

[-1.7166161 , -0.52329102, -1.63724046],

[-0.01490402, 1.03191866, -0.33399411],

[ 1.00612323, 0.6938296 , 0.69488459]])

In [37]:

stu2, x **=** x[**-**1], x[:**-**1]

stu1, x **=** x[**-**1], x[:**-**1]

print(stu1,stu2)

[-0.01490402 1.03191866 -0.33399411] [1.00612323 0.6938296 0.69488459]

**# Splitting the training and testing data**

In [38]:

**from** sklearn.model\_selection **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size **=** 0.20, random\_state **=** 0)

print(x\_train.shape, x\_test.shape)

print(x\_test[0])

(8000, 3) (2000, 3)

[-1.7166161 -1.33470476 1.70089931]

**# Applying the KNN model for n=1 initially**

In [39]:

**from** sklearn.neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier(n\_neighbors**=**1)

knn.fit(x\_train,y\_train)

kpred **=** knn.predict(x\_test)

In [40]:

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

print('WITH K = 1 (initially)\n')

print('Confusion Matrix:')

print(confusion\_matrix(y\_test,kpred))

print('\nClassification Report:')

print(classification\_report(y\_test,kpred))

WITH K = 1 (initially)

Confusion Matrix:

[[1769 5 7]

[ 4 50 48]

[ 8 50 59]]

Classification Report:

precision recall f1-score support

0 0.99 0.99 0.99 1781

1 0.48 0.49 0.48 102

2 0.52 0.50 0.51 117

accuracy 0.94 2000

macro avg 0.66 0.66 0.66 2000

weighted avg 0.94 0.94 0.94 2000

**# Checking for the best value of n in range(1,40) for KNN**

In [41]:

error\_rate **=** []

**for** i **in** range(1,40):

knn **=** KNeighborsClassifier(n\_neighbors**=**i)

knn.fit(x\_train,y\_train)

kpred\_i **=** knn.predict(x\_test)

error\_rate.append(np.mean(kpred\_i **!=** y\_test))

**if** (error\_rate[i**-**1]**==**min(error\_rate)) :

dip **=** i

​

print('min error=', dip, min(error\_rate))

print(error\_rate)

min error= 11 0.054

[0.061, 0.062, 0.061, 0.062, 0.06, 0.0585, 0.055, 0.0555, 0.058, 0.058, 0.054, 0.0605, 0.062, 0.0585, 0.062, 0.0615, 0.0595, 0.059, 0.058, 0.0595, 0.0595, 0.0595, 0.0605, 0.061, 0.0625, 0.0615, 0.0645, 0.065, 0.064, 0.064, 0.064, 0.064, 0.066, 0.0655, 0.067, 0.0655, 0.067, 0.066, 0.066]

In [58]:

plt.figure(figsize**=**(10,6))

plt.plot(range(1,40),error\_rate,color**=**'blue', linestyle**=**'dashed', marker**=**'o',markerfacecolor**=**'red', markersize**=**10)

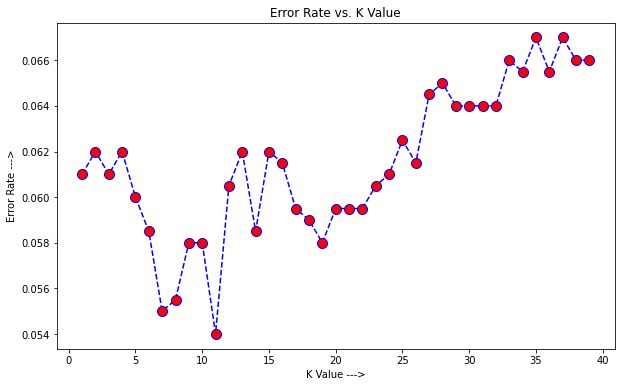
plt.title('Error Rate vs. K Value')

plt.xlabel('K Value --->')

plt.ylabel('Error Rate --->')

Out[58]:

Text(0, 0.5, 'Error Rate --->')



**# Again running KNN with best value of n i.e. 11 in our case**

In [55]:

knn **=** KNeighborsClassifier(n\_neighbors**=** 11)

knn.fit(x\_train,y\_train)

kpred **=** knn.predict(x\_test)

In [44]:

print('FINALLY WITH K = 11 (error rate minima)\n')

print('Confusion Matrix:')

print(confusion\_matrix(y\_test,kpred))

print('\nClassification Report:')

print(classification\_report(y\_test,kpred))

FINALLY WITH K = 11 (error rate minima)

Confusion Matrix:

[[1780 0 1]

[ 9 49 44]

[ 17 37 63]]

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 1781

1 0.57 0.48 0.52 102

2 0.58 0.54 0.56 117

accuracy 0.95 2000

macro avg 0.71 0.67 0.69 2000

weighted avg 0.94 0.95 0.94 2000

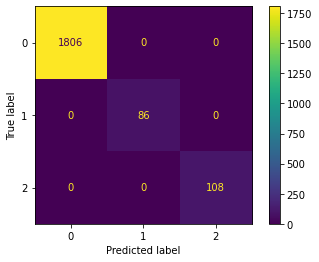
In [45]:

**from** sklearn.metrics **import** plot\_confusion\_matrix

plot\_confusion\_matrix(knn, x\_test, kpred)

Out[45]:

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x21d709096c8>



**# Predicting the result for the given two Inputs**

In [46]:

campus**=**['Not Qualified','Selected in IIM-DL','Selected in IIM-GJ']

**def** kpredictfor(testcase):

college **=** knn.predict(testcase)

**return** college

​

print('Custom Cases:')

*# testA = [[50,40,60]]*

collegeA **=** kpredictfor([stu1])

print('KNN: Student A was ---> ',campus[int(collegeA)])

*#testB = [[80,35,105]]*

collegeB **=** kpredictfor([stu2])

print('KNN: Student B was ---> ',campus[int(collegeB)])

​

Custom Cases:

KNN: Student A was ---> Not Qualified

KNN: Student B was ---> Not Qualified

**# Applying Bayesian Learning Model on the same dataset**

In [47]:

**from** sklearn.naive\_bayes **import** GaussianNB

gnb **=** GaussianNB()

gnb.fit(x\_train,y\_train)

bpred **=** gnb.predict(x\_test)

In [48]:

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

print('Confusion Matrix:')

print(confusion\_matrix(y\_test,bpred))

print('\nClassification Report:')

print(classification\_report(y\_test,bpred))

Confusion Matrix:

[[1781 0 0]

[ 31 22 49]

[ 43 31 43]]

Classification Report:

precision recall f1-score support

0 0.96 1.00 0.98 1781

1 0.42 0.22 0.28 102

2 0.47 0.37 0.41 117

accuracy 0.92 2000

macro avg 0.61 0.53 0.56 2000

weighted avg 0.90 0.92 0.91 2000

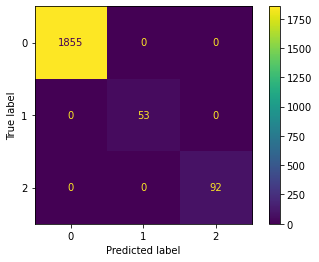
In [49]:

**from** sklearn.metrics **import** plot\_confusion\_matrix

plot\_confusion\_matrix(gnb, x\_test, bpred)

Out[49]:

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x21d70a37788>



**# Comparing the KNN with Bayesian learning model**

In [52]:

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.model\_selection **import** cross\_val\_score

​

print('Comparison KNN vs GNB \n')

print('knn', cross\_val\_score(knn, x\_test, y\_test, scoring**=**'accuracy', cv**=**10).mean())

print('gnb', cross\_val\_score(gnb, x\_test, y\_test, scoring**=**'accuracy', cv**=**10).mean())

Comparison KNN vs GNB (Custom)

knn 0.9405000000000001

gnb 0.9235000000000001

In [53]:

plt.figure(figsize**=**(10,6))

plt.plot(range(1,11),cross\_val\_score(knn, x\_test, y\_test, scoring**=**'accuracy', cv**=**10),color**=**'green', linestyle**=**'dashed', marker**=**'o',markerfacecolor**=**'green', markersize**=**10, label**=**'knn')

plt.plot(range(1,11),cross\_val\_score(gnb, x\_test, y\_test, scoring**=**'accuracy', cv**=**10),color**=**'red', linestyle**=**'dashed', marker**=**'o',markerfacecolor**=**'red', markersize**=**10, label**=**'gnb')

plt.legend()

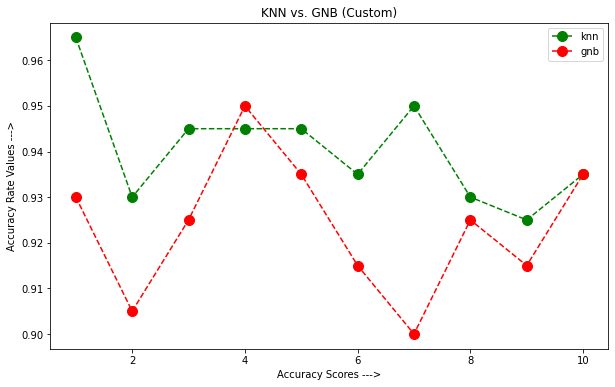
plt.title('KNN vs. GNB')

plt.xlabel('Accuracy Scores --->')

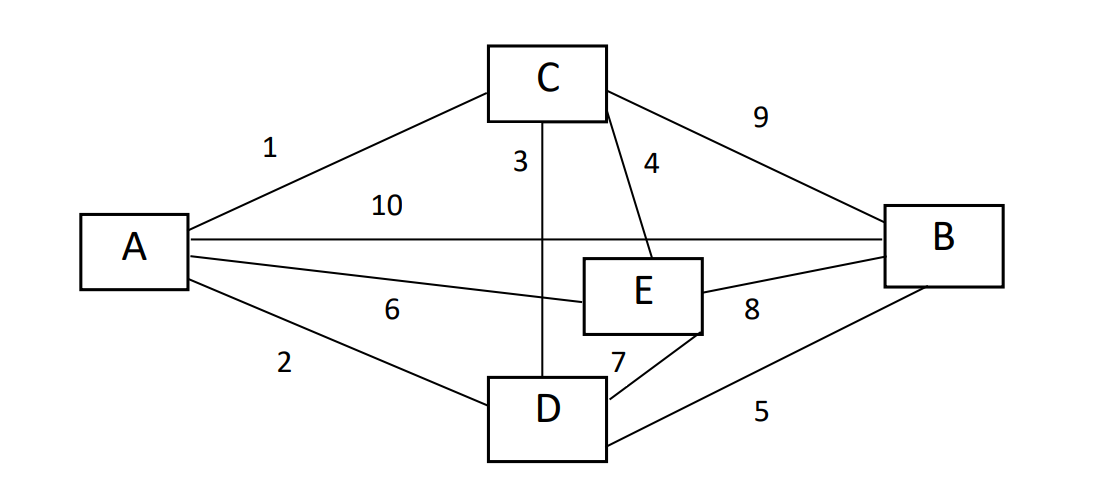
plt.ylabel('Accuracy Rate Values --->')

Out[53]:

Text(0, 0.5, 'Accuracy Rate Values --->')



**Q2.** Solve following TSP problem using prolog. Consider source node= ‘A’ and Goal Node = ‘B’



**CODE:**

edge(a,b,10).

edge(a,c,1).

edge(a,d,2).

edge(a,e,6).

edge(b,a,10).

edge(b,c,9).

edge(b,d,5).

edge(b,e,8).

edge(c,a,1).

edge(c,b,9).

edge(c,d,3).

edge(c,e,4).

edge(d,a,2).

edge(d,b,5).

edge(d,c,3).

edge(d,e,7).

edge(e,a,6).

edge(e,b,8).

edge(e,c,4).

edge(e,d,7).

len([], 0).

len([H|T], N):-

len(T, X),

N is X+1 .

best\_path(Visited, Total):- path(a, b, Visited, Total).

path(Start, Fin, Visited, Total) :-

path(Start, Fin, [Start], Visited, 0, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, StopLoc, Distance),

NewCostn is Costn + Distance,

\+ member(StopLoc, CurrentLoc),

path(StopLoc, Fin, [StopLoc|CurrentLoc], Visited, NewCostn, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, Fin, Distance), reverse([Fin|CurrentLoc], Visited),

len(Visited, Q),

(Q\=5 -> Total is 100000; Total is Costn + Distance).

shortest\_path(Path):-setof(Cost-Path, best\_path(Path,Cost), Holder),pick(Holder,Path).

best(Cost-Holder,Bcost-\_,Cost-Holder):-

Cost<Bcost,!.

best(\_,X,X).

pick([Cost-Holder|R],X):-

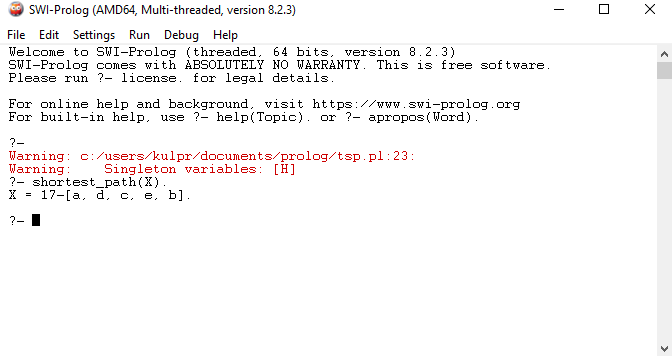
pick(R,Bcost-Bholder),

best(Cost-Holder,Bcost-Bholder,X),

!.

pick([X],X).

**OUTPUT:**



**Q3.** Write a prolog program to implement Medical Diagnosis Expert System (MDES). Create your own facts and rules and apply it in MDES implementation.

**CODE:**

:- dynamic known\_answers/3.

:- dynamic patient\_record/2.

:- dynamic illness/2.

%------------------

% symptom

% if you want to add more symptoms, just add more predicates below

%------------------

symptom(fever).

symptom(cough).

symptom(shivering).

symptom(runny\_nose).

%------------------

% new\_patient/1

% starts a new session

% It must be initiated with a patent name.

% This goal clears all known\_answers,

% and starts the examine subgoal followed by diagnosis subgoal.

%------------------

new\_patient(PatientName) :- not(patient\_already\_exists(PatientName)),

retractall(known\_answer(\_,\_,\_)),write('\nStarting examination...\n'),examine,confirmed\_symptoms(S), write('\nConfirmed symptoms are '),

write\_term(S, []), write('\n'), assert(patient\_record(PatientName,S)), !,

write('Determining illness...\n'), diagnose(S,I),!, write('AI diagnosed that you have '),

write\_term(I, []), !.

%------------------

% symptoms/1

% returns a list of all symptoms61

%------------------

symptoms(L) :-findall(X, symptom(X), L).

%------------------

% examine/0

% starts the examination process

% by asking for a yes/ no question against each symptom

%------------------

examine :- symptoms(L), check\_symptoms(L).

% ------------------

% diagnose/2

% starts the diagnosis process by checking PatientSymptoms and unifying the Illness

% This works by checking whether an Ilness exists with symptoms being

% subset of PatientSymptoms ------------------

diagnose(PS, I) :- length(PS, MustMatchCount), diagnose(PS, MustMatchCount, I).

%------------------

% diagnose/2

% The following predicate is expected to match when no other illness is identified.

%------------------

diagnose(\_, unknown\_disease).

%------------------

% diagnose/3

% recursively matches the illness symptoms and patient symptoms with

% decreasing number of matches PS: Patient Symptoms (expected to be

% passed as a parameter) I: Illness (expected to be unified)

% MustMatchCount: The number of symptoms that should exist in Illness

% ------------------

diagnose(PS, MustMatchCount, I) :- ( illness(I,S), length(S, MustMatchCount),

subset(S,PS),! ); (MustMatchCount > 1, NewCount is MustMatchCount-1,

diagnose(PS, NewCount, I) ).

%------------------

% check\_symptoms/1

% given a list of symptoms, ask quetions

%------------------

check\_symptoms([]) :- !.

check\_symptoms([H|T]) :- ask(symptom,H), check\_symptoms(T).

%------------------

% confirmed\_symptoms/1

% returns a list of symptoms for which the answer is yes

%------------------

confirmed\_symptoms(C) :- findall(X,known\_answer(yes,symptom,X),C).

%------------------

% ask/2

% given an attribute and a value, gets a yes/no answer from the user

% It works by writing a prompt and having a subgoal to assert the answer

%------------------

ask(Attr,Val) :-write(Attr:Val), write('? '), read(Y), asserta(known\_answer(Y,Attr,Val)).

%------------------

% fix\_diagnosis/2

% learns "actual" illness of a "patient" and improves the learning process

%------------------

fix\_diagnosis(PatientName, ActualIllness) :- patient\_record(PatientName, PS),

write('Confirmed patient symptoms '), write\_term(PS, []),

write(' will be related to '), write\_term(ActualIllness, []), !, update\_definition(ActualIllness, PS, FS),!, write('\nNew definition is '), write\_term(FS,[]).

%------------------

% update\_definition/3

% given an illness and new symptoms, returns the updated symptoms for that illness

% Following case is when the illness is not already defined.

%------------------

update\_definition(Illness, RelateSymptoms, RelateSymptoms) :- not(illness(Illness, \_)),

write('\nThere was no earlier definition of '), write\_term(Illness, []),assert(illness(Illness, RelateSymptoms)),!.

%------------------

% update\_definition/3

% given an illness and new symptoms, returns the updated symptoms for that illness

% Following case is when the illness is already defined, and hence takes

% an intsection of old and new symtpoms ------------------

update\_definition(Illness, RelateSymptoms, FinalSymptoms) :- illness(Illness, OldSymptoms),

write('\nEarlier definition of '),write(Illness),write(' was '),

illness(Illness, OldSymptoms),write\_term(OldSymptoms,[]),

intersection(OldSymptoms, RelateSymptoms, FinalSymptoms), retractall(illness(Illness,\_)),

assert(illness(Illness, FinalSymptoms)).

%----------------------

% rediagnose/1

% given a patient name, rechecks diagnosis based on existing symptoms

% This goal could be requested, for example, when illness predicates are updated

%----------------------

rediagnose(PatientName) :-patient\_record(PatientName, C),write('\nConfirmed symptoms were

'),

write\_term(C, []),write('\nRediagnosing...'), diagnose(C, NewIllness),!,

write('\nUpdated diagnosis is that Patient '), write\_term(PatientName, []),

write(' is having '),write\_term(NewIllness, []).

%----------------------

% patient\_already\_exists/1

% true if given patient name is already in the patient records

%----------------------

patient\_already\_exists(PatientName) :-patient\_record(PatientName,\_),

write('Patient '),write\_term(PatientName,[]),write(' already exists.\n').

%----------------------

% show\_patient\_records/0

% shows all patient records

%----------------------

show\_patient\_records :- findall((P,S), patient\_record(P,S), L), !,show\_records(L).

%----------------------

% show\_records/1

% Calls show\_record for each (PatientName, Symptom) pair

%----------------------

show\_records([]).

show\_records([(P,S)|T]) :- show\_record(P,S),show\_records(T).

show\_record(P, S) :- diagnose(S, I),!, write\_term(P, []),write(' has symptoms '),

write\_term(S, []), write(' and diagnosed '),

write\_term(I, []), write('\n').

%---------------------------

% change\_diagnosis/2

% Associate the symptoms from one illness to another for a given patient

%---------------------------

change\_diagnosis(Patient, NewIllness) :-patient\_record(Patient, Symptoms),!.

write\_term(Patient, []),write(' has symptoms '),

write\_term(Symptoms, []),diagnose(Symptoms, OldIllness),

write(' and was diagnosed '), write\_term(OldIllness, []),

write('\n'),!,write('Changing it to '),

write\_term(NewIllness, []),write('\n'),

retractall(illness(NewIllness, \_)), retractall(illness(OldIllness, \_)),

assert(illness(NewIllness, Symptoms)).

**OUTPUT:**

