Experiment No.05

A.1 Aim: Implementation of Naïve Bayes Algorithm using any programming language like JAVA, C++, Python or WEKA Tool.

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Blackboard access available)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 15/04/2021	**Date of Submission:** 15/04/2021
Grade:	

B.1 Software Code written by a student:

(Paste your problem statement related to your case study completed during the 2 hours of practice in the lab here)

```
import numpy as np
import pandas as pd
# Import necessary modules
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
d1 = pd. read_csv ('diabetes_csv.csv')

d1.head()

# Loading data
# Create feature and target arrays
X = d1[d1.columns[:-1]]
y = d1[d1.columns[-1]]
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=42)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
```

d1

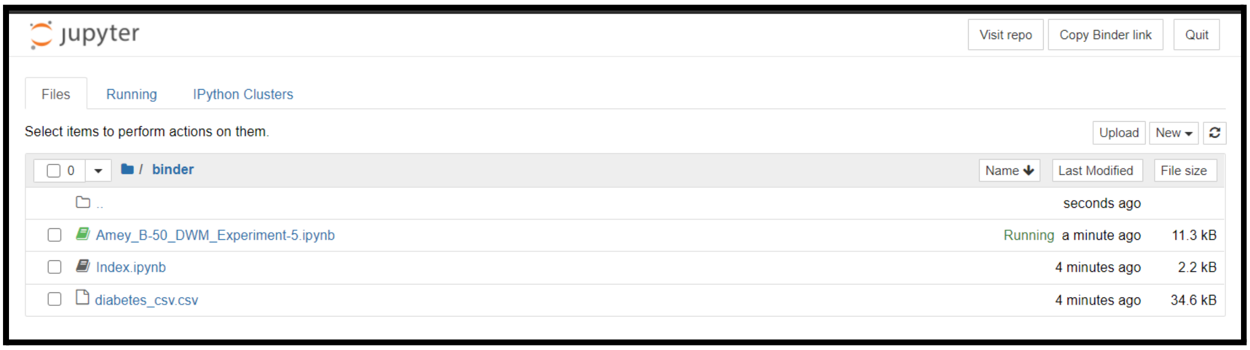
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
# Predict on dataset which model has not seen before
print(gnb.predict(X_test))

B.2 Input and Output:

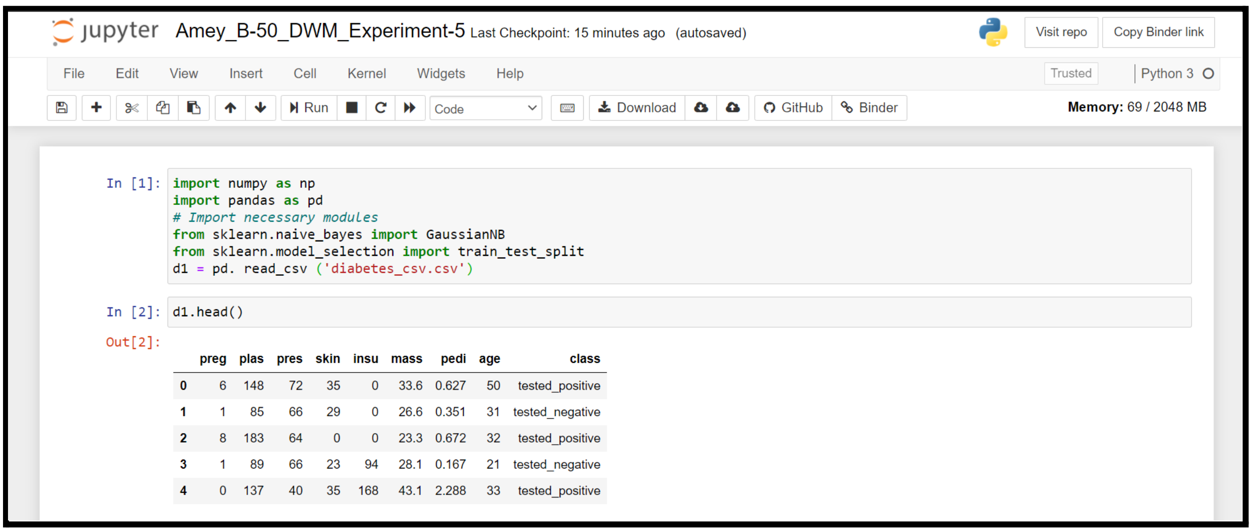
(Paste your program input and output in the following format, If there is an error then paste the specific error in the output part. In case of an error with the due permission of the faculty, an extension can be given to submit the error-free code with output in due course of time. Students will be graded accordingly.)

Jupyter Notebook

Jupyter/Binder:



Jupyter Notebook:



	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

jupyter Amey_B-50_DWM_Experiment-5 Last Checkpoint: 15 minutes ago (autosaved)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

Run | Code | Download | GitHub | Binder | Memory: 69 / 2048 MB

In [3]:
```python
# Loading data
# Create feature and target arrays
X = d1[d1.columns[:-1]]
y = d1[d1.columns[-1]]
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=42)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
```

In [4]:
```python
d1
```

Out[4]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	tested_negative
764	2	122	70	27	0	36.8	0.340	27	tested_negative
765	5	121	72	23	112	26.2	0.245	30	tested_negative
766	1	126	60	0	0	30.1	0.349	47	tested_positive
767	1	93	70	31	0	30.4	0.315	23	tested_negative

768 rows × 9 columns

jupyter Amey_B-50_DWM_Experiment-5 Last Checkpoint: 17 minutes ago (autosaved)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

Run | Code | Download | GitHub | Binder | Memory: 69 / 2048 MB

In [5]:
```python
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
# Predict on dataset which model has not seen before
print(gnb.predict(X_test))
```

```
Gaussian Naive Bayes model accuracy(in %): 76.62337662337663
['tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_positive'
 'tested_positive' 'tested_positive' 'tested_positive' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_positive' 'tested_positive' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative']
```

In [ ]:

Sample Dataset:

```
In [4]: d1
Out[4]:
```

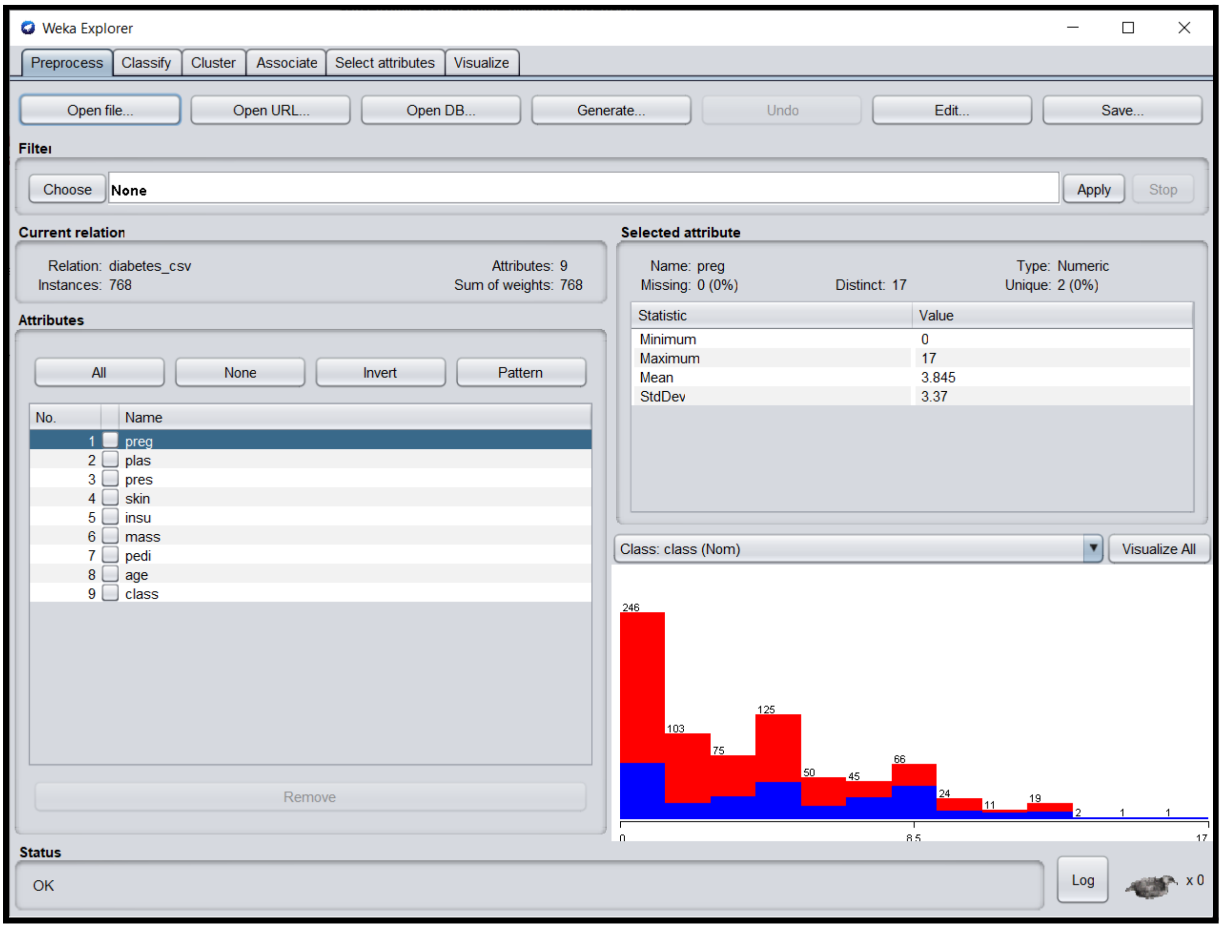
	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	tested_negative
764	2	122	70	27	0	36.8	0.340	27	tested_negative
765	5	121	72	23	112	26.2	0.245	30	tested_negative
766	1	126	60	0	0	30.1	0.349	47	tested_positive
767	1	93	70	31	0	30.4	0.315	23	tested_negative

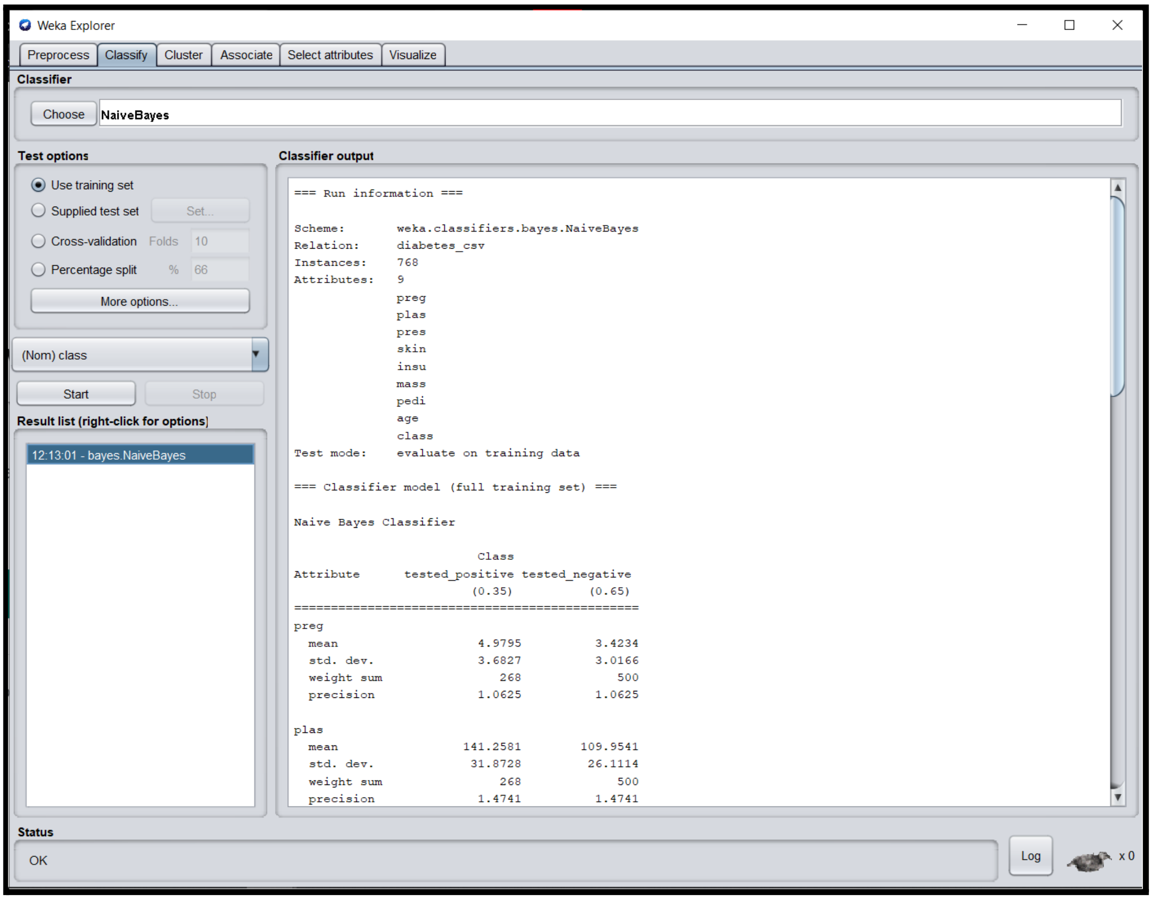
768 rows × 9 columns

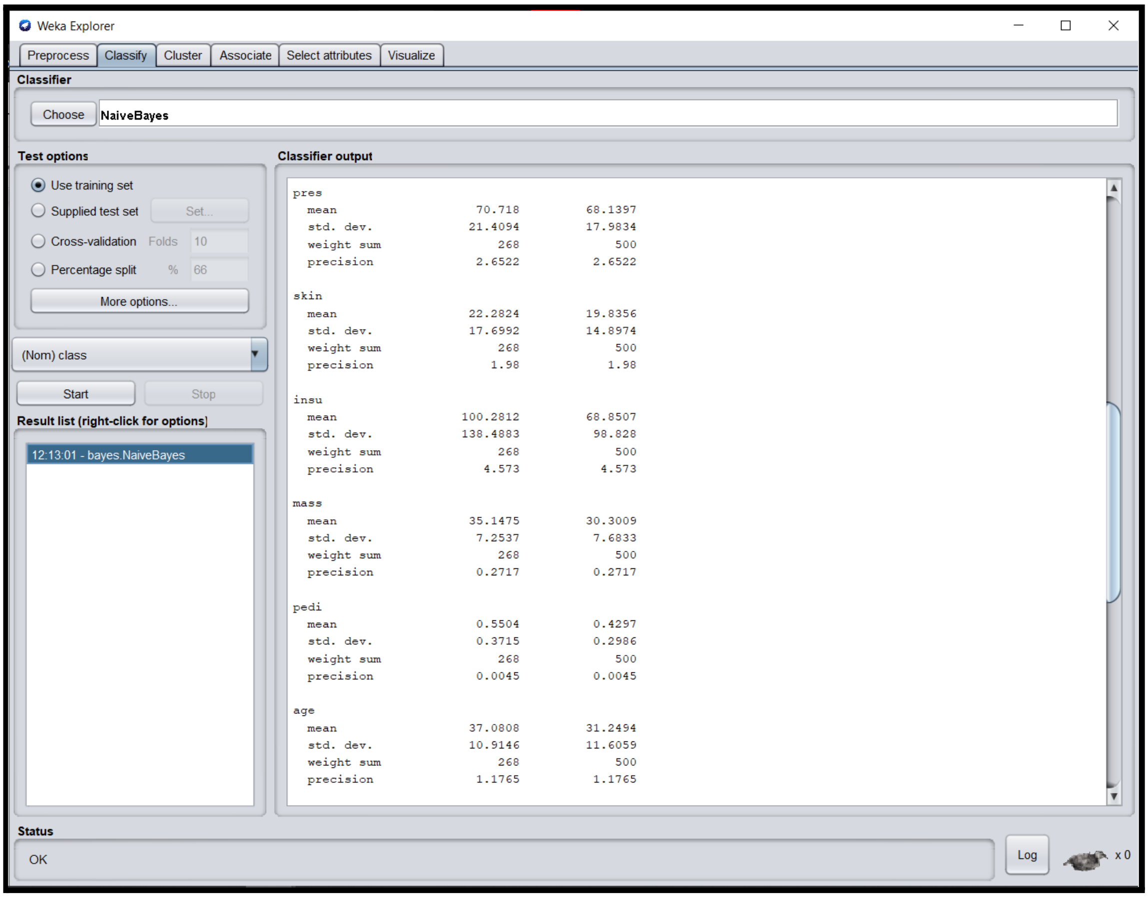
Python Output:

```
Gaussian Naive Bayes model accuracy(in %): 76.62337662337663
['tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_positive'
 'tested_positive' 'tested_positive' 'tested_positive' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_positive' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_positive'
 'tested_positive' 'tested_positive' 'tested_positive' 'tested_positive'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_positive' 'tested_negative'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_positive'
 'tested_negative' 'tested_negative' 'tested_negative' 'tested_negative'
 'tested_negative' 'tested_positive' 'tested_negative' 'tested_negative'
 'tested_positive' 'tested_negative']
```

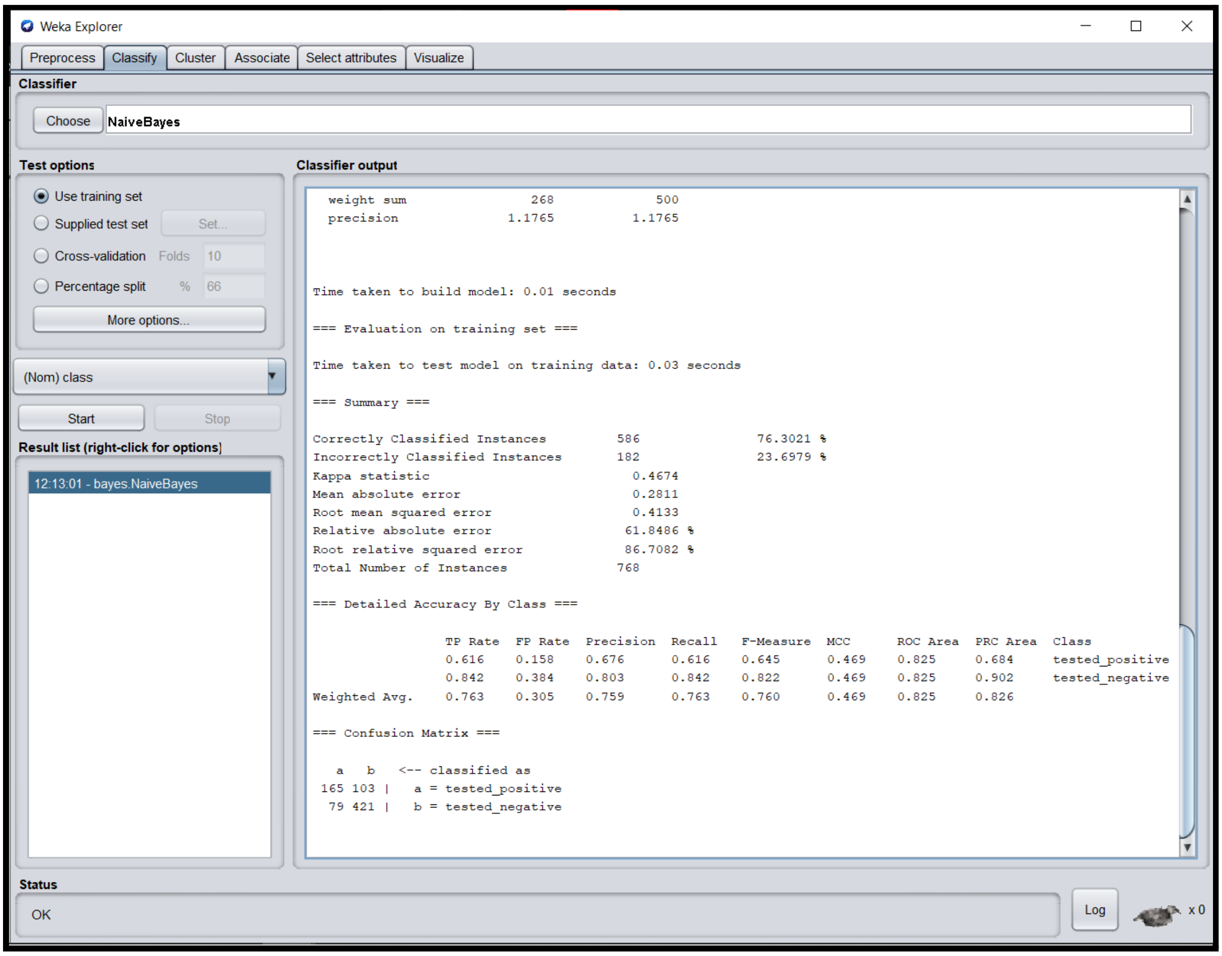
Weka Tool




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
12:13:01 - bayes.NaiveBayes
Classifier output
=== Run information ===
Scheme: weka.classifiers.bayes.NaiveBayes
Relation: diabetes_csv
Instances: 768
Attributes: 9
preg
plas
pres
skin
insu
mass
pedi
age
class
Test mode: evaluate on training data
=== Classifier model (full training set) ===
Naive Bayes Classifier
Class
Attribute tested_positive tested_negative
(0.35) (0.65)
preg
mean 4.9795 3.4234
std. dev. 3.6827 3.0166
weight sum 268 500
precision 1.0625 1.0625
plas
mean 141.2581 109.9541
std. dev. 31.8728 26.1114
weight sum 268 500
precision 1.4741 1.4741
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
12:13:01 - bayes.NaiveBayes
Classifier output
pres
mean 70.718 68.1397
std. dev. 21.4094 17.9834
weight sum 268 500
precision 2.6522 2.6522
skin
mean 22.2824 19.8356
std. dev. 17.6992 14.8974
weight sum 268 500
precision 1.98 1.98
insu
mean 100.2812 68.8507
std. dev. 138.4883 98.828
weight sum 268 500
precision 4.573 4.573
mass
mean 35.1475 30.3009
std. dev. 7.2537 7.6833
weight sum 268 500
precision 0.2717 0.2717
pedi
mean 0.5504 0.4297
std. dev. 0.3715 0.2986
weight sum 268 500
precision 0.0045 0.0045
age
mean 37.0808 31.2494
std. dev. 10.9146 11.6059
weight sum 268 500
precision 1.1765 1.1765
Status
OK
Log
x 0

Weka Output:



Naive Bayes model accuracy (in %):

76.62337662337663 (Using Python)
76.3021 (Using Weka)

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

We observed that Bayesian Classification represents a supervised learning method as well as a statistical classification method. The Bayesian classification is used as a probabilistic learning method as Naive Bayes classifiers are among the most successful known algorithms for learning to classify the datasets. Naïve Bayes classification Algorithm is one of the probabilistic algorithms which classifies the datasets according to its knowledge data and creates the Result as per the given knowledge.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

- Naïve Bayes classification Algorithm is one of the probabilistic algorithms which classify the datasets according to its knowledge data and creates the Result as per the given knowledge.
- Hence we've successfully implemented the Naive Bayesian Algorithm through Python as well as Weka Tool.

B.5 Question of Curiosity

(To be answered by the student based on the practical performed and learning/observations)

1. List out other classifiers and their efficiency.

Ans:

A. Decision trees
B. Bayesian classifiers
C. Neural networks
D. Nearest neighbour classifiers
E. Support vector machines
F. Linear regression.

Comparison of different classifier techniques with efficiency:

DT=decision tree,
NB=Naive Bayes classifier,
GB=general Bayesian classifier,
F F NN= feed-forward neural network,
K-nn=K-nearest neighbour classifier,
SV M=support vector machine, and
LR= linear regression.

	DT	*NB*	*GB*	*FFNN*	*K-nn*	*SVM*	*LR*
Non-linear boundaries	+	(+)	+	+	+	+	−
Accuracy on small data sets	−	+	+/−	−	−	+	+
Works with incomplete data	−	+	+	+	+	−	−
Supports mixed variables	+	+	+	−	+	−	−
Natural interpretation	+	+	+	−	(+)	−	+
Efficient reasoning	+	+	+	+	−	+	+
Efficient learning	+/−	+	−	−	+/−	+	+
Efficient updating	−	+	+	+	+	−	+

2. What is the accuracy of the classifier explain with an example?

Ans:

The accuracy of a classifier is given as the percentage of total correct predictions divided by the total number of instances.

Mathematically,

$$\boxed{\textbf{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}}$$

If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known.