

# DATA MINING LAB ASSIGNMENT

Name: P.Prasanna Kumar

Roll No:2451-17-733-096

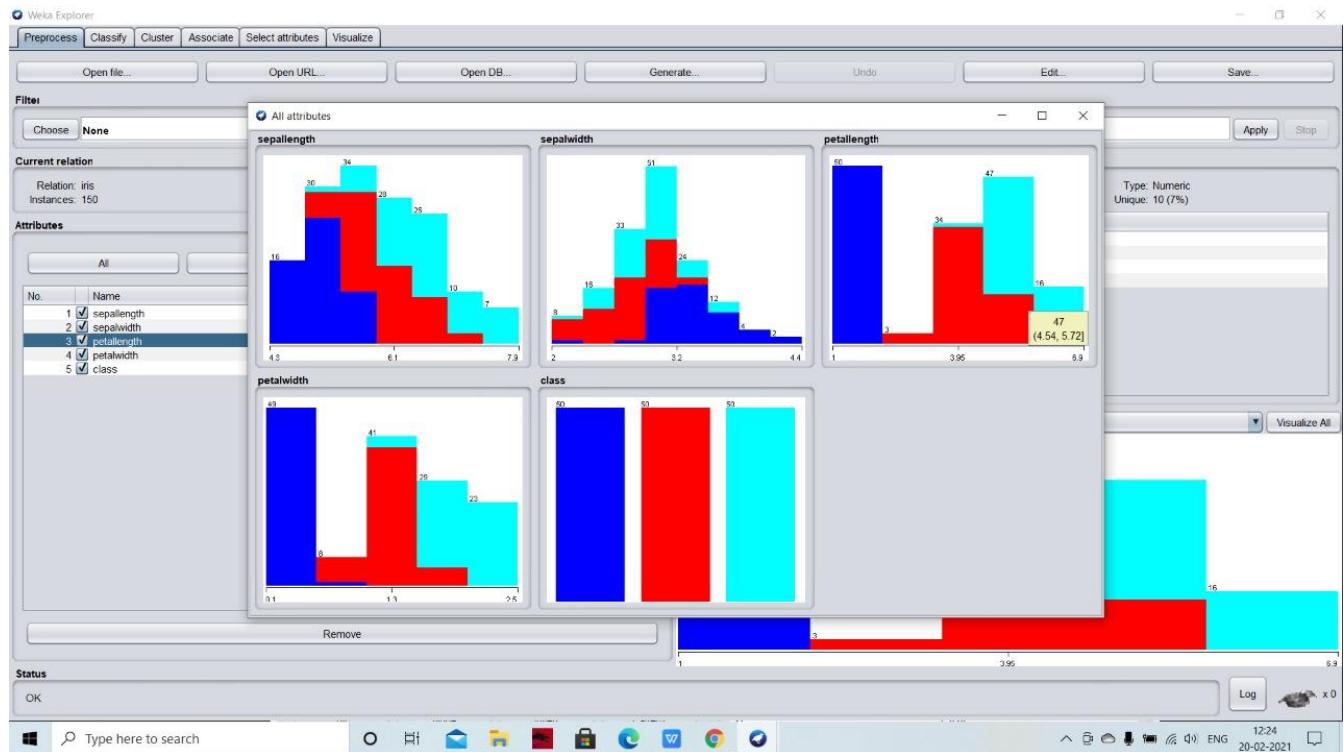
Section:CSE-2

## 1.DATAPREPROCESSING

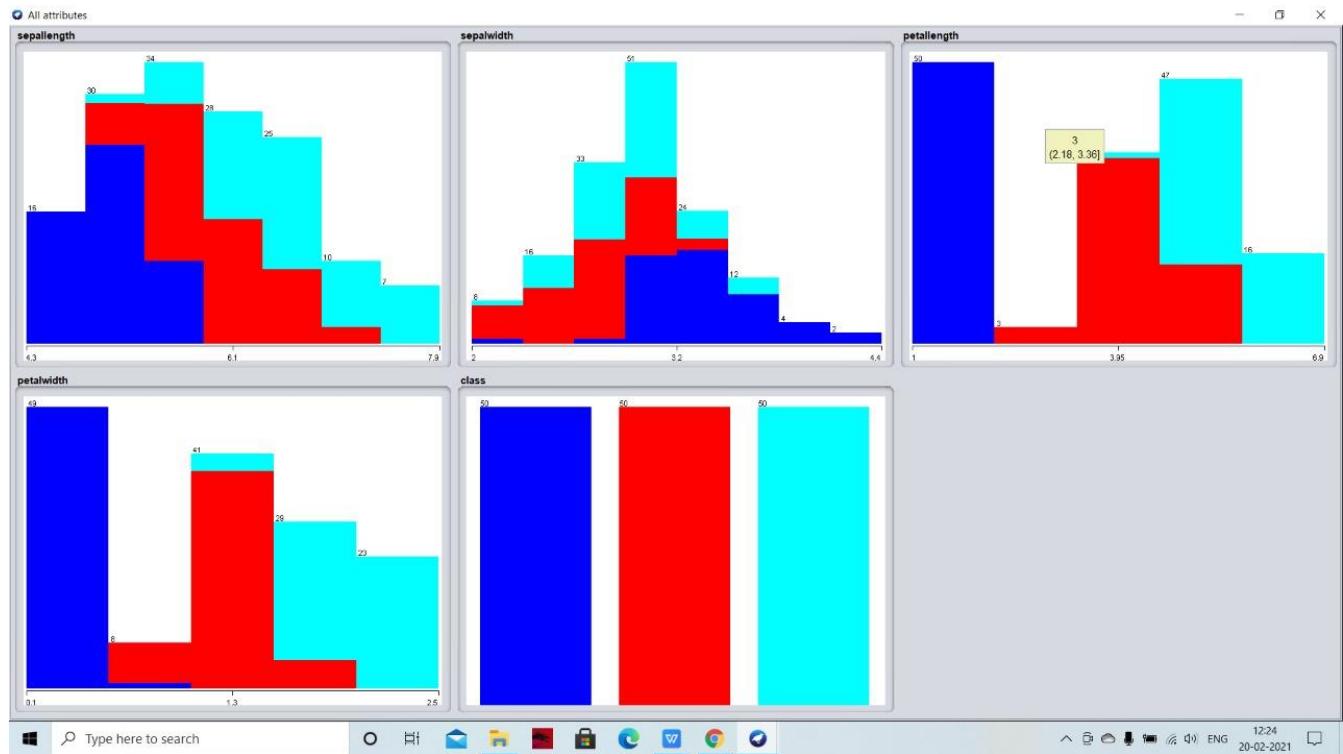
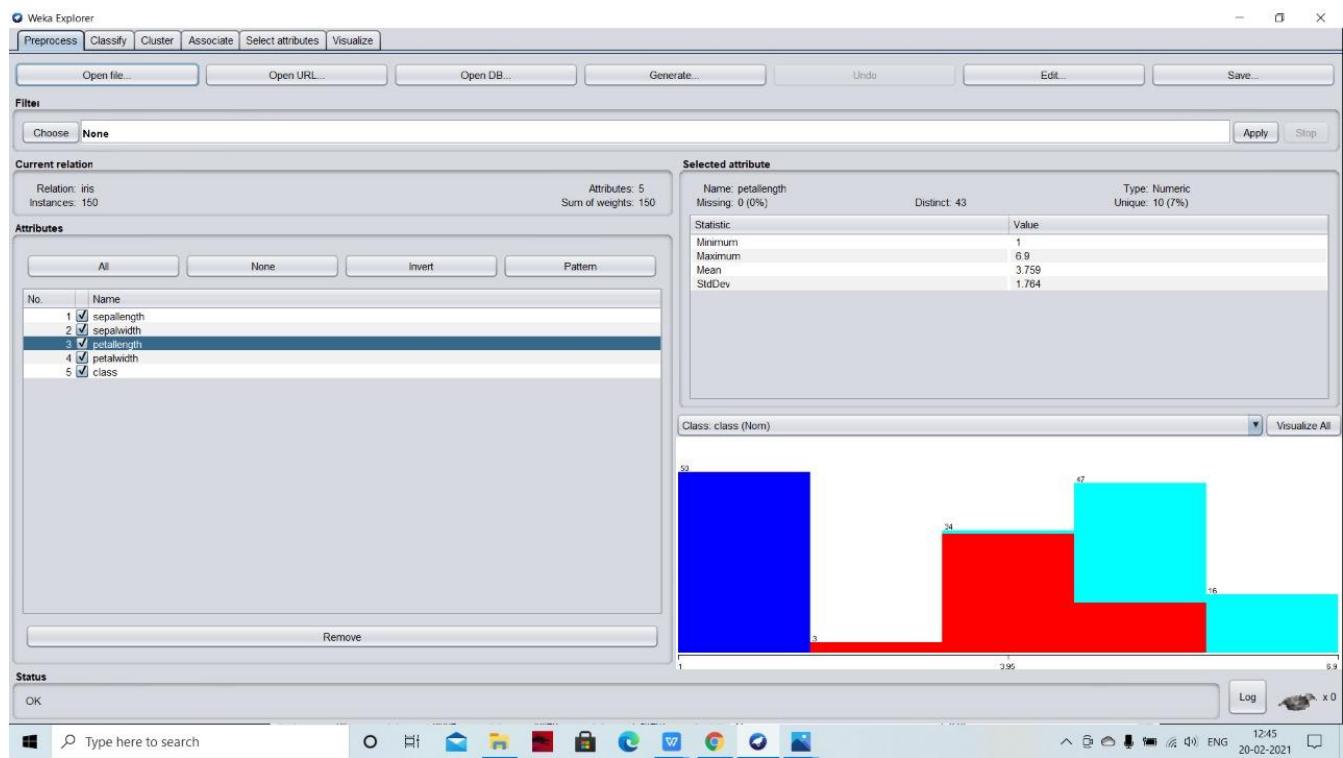
### 1.1Descriptive Stats and Visualization

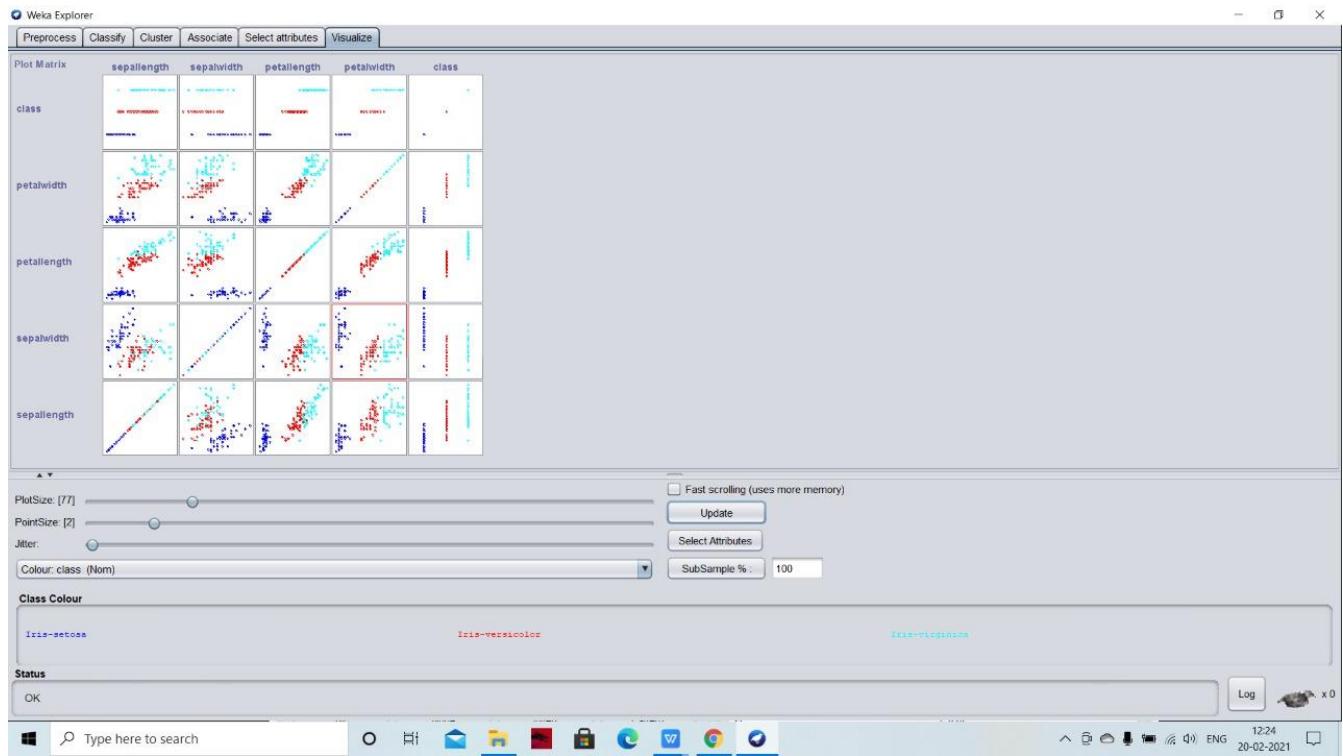
Weka allows you to review descriptive statistics calculated from your data. It also provides visualization tools.

1. Open the Weka GUI Chooser.
2. Open the Weka Explorer.
3. Load the iris.arff dataset.
4. Click on different attributes in the Attributes list and review the details in the Selected attribute pane.
5. Click the Visualize All button to review all attribute distributions.
6. Click the Visualize tab and review the scatter plot matrix for all attributes.



Get comfortable reviewing the details for different attributes in the Preprocess tab and tuning the scatter plot matrix in the Visualize tab.





## 1.2 Replacing Missing Values

Open the input file input.csv apply ReplaceMissingValues and save output.csv

Paste your screen shots here:

### Input.csv

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Viewer

Relation: kidney\_disease

No.	1: id	2: age	3: bp	4: sg	5: al	6: su	7: rbc	8: pc	9: pcc	10: ba	11: bgr	12: bu	13: sc	14: sod	15: pot	16: hemo	17: pcv	18: wc	19: rc	20: htn	21: dm	22: cad	23: appet	24: pe	25: ane	26: classifica
1	0.0	48.0	80.0	1.02	1.0	0.0	nor...	notp...	121.0	36.0	1.2				15.4	44.0	780...	5.2	yes	yes	no	good	no	no	ckd	
2	1.0	7.0	50.0	1.02	4.0	0.0	nor...	notp...		18.0	0.8				11.3	38.0	600...	no	no	no	no	good	no	no	ckd	
3	2.0	62.0	80.0	1.01	2.0	3.0	nor...	nor...	notp...	423.0	53.0	1.8			9.6	31.0	750...	no	yes	no	poor	no	yes	ckd		
4	3.0	48.0	70.0	1.005	4.0	0.0	nor...	abn...	pres...	117.0	56.0	3.8	111.0	2.5	11.2	32.0	670...	3.9	yes	no	no	poor	yes	yes	ckd	
5	4.0	51.0	80.0	1.01	2.0	0.0	nor...	nor...	notp...	106.0	26.0	1.4			11.6	35.0	730...	4.6	no	no	no	good	no	no	ckd	
6	5.0	60.0	90.0	1.015	3.0	0.0		notp...	notp...	74.0	25.0	1.1	142.0	3.2	12.2	39.0	780...	4.4	yes	yes	no	good	yes	no	ckd	
7	6.0	68.0	70.0	1.01	0.0	0.0	nor...	notp...	notp...	100.0	54.0	24.0	104.0	4.0	12.4	36.0		no	no	no	good	no	no	ckd		
8	7.0	24.0	50.0	1.015	2.0	4.0	nor...	abn...	notp...	410.0	31.0	1.1			12.4	44.0	690...	5.0	no	yes	no	good	yes	no	ckd	
9	8.0	52.0	100.0	1.015	3.0	0.0	nor...	abn...	pres...	138.0	60.0	1.9			10.8	33.0	960...	4.0	yes	no	good	no	yes	ckd		
10	9.0	53.0	90.0	1.02	2.0	0.0	abn...	abn...	pres...	70.0	107.0	7.2	114.0	3.7	9.5	29.0	121...	3.7	yes	yes	no	poor	no	yes	ckd	
11	10.0	50.0	60.0	1.01	2.0	4.0	abn...	abn...	pres...	490.0	55.0	4.0			9.4	28.0		yes	yes	no	good	no	yes	ckd		
12	11.0	63.0	70.0	1.01	3.0	0.0	abn...	abn...	pres...	380.0	60.0	2.7	131.0	4.2	10.8	32.0	450...	3.8	yes	yes	no	poor	yes	no	ckd	
13	12.0	68.0	70.0	1.015	3.0	1.0	nor...	pres...	pres...	208.0	72.0	2.1	138.0	5.8	9.7	28.0	122...	3.4	yes	yes	poor	yes	no	ckd		
14	13.0	68.0	70.0				notp...	notp...	notp...	98.0	85.0	4.6	135.0	3.4	9.8		yes	yes	yes	poor	yes	no	ckd			
15	14.0	68.0	80.0	1.01	3.0	2.0	nor...	abn...	pres...	157.0	90.0	4.1	130.0	6.4	5.6	16.0	110...	2.6	yes	yes	yes	poor	yes	no	ckd	
16	15.0	40.0	80.0	1.015	3.0	0.0	nor...	notp...	notp...	76.0	162.0	9.6	141.0	4.9	7.6	24.0	380...	2.8	yes	no	no	good	no	yes	ckd	
17	16.0	47.0	70.0	1.015	2.0	0.0	nor...	notp...	notp...	99.0	46.0	2.2	138.0	4.1	12.6			no	no	no	good	no	no	ckd		
18	17.0	47.0	80.0				notp...	notp...	notp...	114.0	87.0	5.2	139.0	3.7	12.1		yes	no	no	poor	no	no	ckd			
19	18.0	60.0	100.0	1.025	0.0	3.0	nor...	notp...	notp...	263.0	27.0	1.3	135.0	4.3	12.7	37.0	114...	4.3	yes	yes	yes	good	no	no	ckd	
20	19.0	62.0	60.0	1.015	1.0	0.0	abn...	abn...	pres...	100.0	31.0	1.6			10.3	30.0	530...	3.7	yes	no	yes	good	no	no	ckd	
21	20.0	61.0	80.0	1.015	2.0	0.0	abn...	abn...	pres...	173.0	148.0	3.9	135.0	5.2	7.7	24.0	920...	3.2	yes	yes	poor	yes	yes	ckd		
22	21.0	60.0	90.0				notp...	notp...	notp...	180.0	76.0	4.5			10.9	32.0	620...	3.6	yes	yes	yes	good	no	no	ckd	
23	22.0	48.0	80.0	1.025	4.0	0.0	nor...	abn...	notp...	95.0	163.0	7.7	136.0	3.8	9.8	32.0	690...	3.4	yes	no	no	good	no	yes	ckd	
24	23.0	21.0	70.0	1.01	0.0	0.0	nor...	notp...	notp...									no	no	no	poor	no	yes	ckd		
25	24.0	42.0	100.0	1.015	4.0	0.0	nor...	abn...	pres...		50.0	1.4	129.0	4.0	11.1	39.0	830...	4.6	yes	no	poor	no	no	ckd		
26	25.0	61.0	60.0	1.025	0.0	0.0	nor...	notp...	notp...	108.0	75.0	1.9	141.0	5.2	9.9	29.0	840...	3.7	yes	no	good	no	yes	ckd		
27	26.0	75.0	80.0	1.015	0.0	0.0	nor...	notp...	notp...	156.0	45.0	2.4	140.0	3.4	11.6	35.0	103...	4.0	yes	no	poor	no	no	ckd		
28	27.0	69.0	70.0	1.01	3.0	4.0	nor...	abn...	notp...	264.0	87.0	2.7	130.0	4.0	12.5	37.0	960...	4.1	yes	yes	good	yes	no	ckd		
29	28.0	75.0	70.0		1.0	3.0	nor...	notp...	notp...	123.0	31.0	1.4					no	yes	no	good	no	no	ckd			

Add instance Undo OK Cancel

OK Log x 0

## Output.csv

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Viewer

Relation: kidney\_disease-weka.filters.unsupervised.attribute.ReplaceMissingValues

No:	1: id	2: age	3: bp	4: sg	5: al	6: su	7: rbc	8: pc	9: pcc	10: ba	11: bgr	12: bu	13: sc	14: sod	15: pot	16: hemc	17: pcv	18: wc	19: rc	20: htn	21: dm	22: cad	23: appet	24: pe	25: ana	26: classifical
1	0.0	48.0	80.0	1.02	1.0	0.0	nor...	notp...	notp...	121.0	36.0	1.2	137...	4.62...	15.4	44.0	780...	5.2	yes	yes	no	good	no	no	no	ckd
2	1.0	7.0	50.0	1.02	4.0	0.0	nor...	nor...	notp...	148...	18.0	0.8	137...	4.62...	11.3	38.0	600...	4.70...	no	no	no	good	no	no	no	ckd
3	2.0	62.0	80.0	1.01	2.0	3.0	nor...	notp...	notp...	423.0	53.0	1.8	137...	4.62...	9.6	31.0	750...	4.70...	no	yes	no	poor	no	yes	yes	ckd
4	3.0	48.0	70.0	1.005	4.0	0.0	nor...	abn...	pres...	notp...	117.0	56.0	3.8	111.0	2.5	11.2	32.0	670...	3.9	yes	no	no	poor	yes	yes	ckd
5	4.0	51.0	80.0	1.01	2.0	0.0	nor...	notp...	notp...	106.0	26.0	1.4	137...	4.62...	11.6	35.0	730...	4.6	no	no	no	good	no	no	no	ckd
6	5.0	60.0	90.0	1.015	3.0	0.0	nor...	nor...	notp...	74.0	25.0	1.1	142.0	3.2	12.2	39.0	780...	4.4	yes	yes	no	good	yes	no	no	ckd
7	6.0	68.0	70.0	1.01	0.0	0.0	nor...	notp...	notp...	100.0	54.0	24.0	104.0	4.0	12.4	36.0	840...	4.70...	no	no	no	good	no	no	no	ckd
8	7.0	24.0	76.4...	1.015	2.0	4.0	abn...	notp...	notp...	410.0	31.0	1.1	137...	4.62...	12.4	44.0	690...	5.0	no	yes	no	good	yes	no	no	ckd
9	8.0	52.0	100.0	1.015	3.0	0.0	nor...	abn...	pres...	notp...	138.0	60.0	1.9	137...	4.62...	10.8	33.0	960...	4.0	yes	yes	no	good	no	yes	ckd
10	9.0	53.0	90.0	1.02	2.0	0.0	abn...	pres...	notp...	70.0	107.0	7.2	114.0	3.7	9.5	29.0	121...	3.7	yes	no	poor	no	yes	ckd		
11	10.0	50.0	60.0	1.01	2.0	4.0	nor...	abn...	pres...	490.0	55.0	4.0	137...	4.62...	9.4	28.0	840...	4.70...	yes	yes	no	good	no	yes	ckd	
12	11.0	63.0	70.0	1.01	3.0	0.0	abn...	pres...	notp...	380.0	60.0	2.7	131.0	4.2	10.8	32.0	450...	3.8	yes	yes	no	poor	yes	no	ckd	
13	12.0	68.0	70.0	1.015	3.0	1.0	nor...	pres...	notp...	208.0	72.0	2.1	138.0	5.8	9.7	28.0	122...	3.4	yes	yes	no	poor	yes	no	ckd	
14	13.0	68.0	70.0	1.01...	1.01...	0.45...	nor...	notp...	notp...	98.0	86.0	4.6	135.0	3.4	9.8	38.8...	840...	4.70...	yes	yes	yes	poor	yes	no	ckd	
15	14.0	68.0	80.0	1.01	3.0	2.0	nor...	abn...	pres...	157.0	90.0	4.1	130.0	6.4	5.6	16.0	110...	2.6	yes	yes	yes	good	yes	no	ckd	
16	15.0	40.0	80.0	1.015	3.0	0.0	nor...	notp...	notp...	76.0	162.0	9.6	141.0	4.9	7.6	24.0	380...	2.8	yes	no	no	good	no	yes	ckd	
17	16.0	47.0	70.0	1.015	2.0	0.0	nor...	notp...	notp...	99.0	46.0	2.2	138.0	4.1	12.6	38.8...	840...	4.70...	no	no	no	good	no	no	ckd	
18	17.0	47.0	80.0	1.01...	1.01...	0.45...	nor...	notp...	notp...	114.0	87.0	5.2	139.0	3.7	12.1	38.8...	840...	4.70...	yes	no	no	poor	no	no	ckd	
19	18.0	60.0	100.0	1.025	0.0	3.0	nor...	notp...	notp...	263.0	27.0	1.3	135.0	4.3	12.7	37.0	114...	4.3	yes	yes	yes	good	no	no	ckd	
20	19.0	62.0	60.0	1.015	1.0	0.0	nor...	abn...	pres...	100.0	31.0	1.6	137...	4.62...	10.3	30.0	530...	3.7	yes	no	yes	good	no	no	ckd	
21	20.0	61.0	80.0	1.015	2.0	0.0	abn...	notp...	notp...	173.0	148.0	3.9	135.0	5.2	7.7	24.0	920...	3.2	yes	yes	yes	poor	yes	yes	ckd	
22	21.0	60.0	90.0	1.01...	1.01...	0.45...	nor...	notp...	notp...	148...	180.0	76.0	4.5	4.62...	10.9	32.0	620...	3.6	yes	yes	yes	good	no	no	ckd	
23	22.0	48.0	80.0	1.025	4.0	0.0	nor...	abn...	notp...	95.0	163.0	7.7	136.0	3.8	9.8	32.0	690...	3.4	yes	no	no	good	no	yes	ckd	
24	23.0	21.0	70.0	1.01	0.0	0.0	nor...	notp...	notp...	148...	57.4...	3.07...	137...	4.62...	12.52...	38.8...	840...	4.70...	no	no	no	poor	no	yes	ckd	
25	24.0	42.0	100.0	1.015	4.0	0.0	nor...	abn...	pres...	148...	50.0	1.4	129.0	4.0	11.1	39.0	830...	4.6	yes	no	no	poor	no	no	ckd	
26	25.0	61.0	60.0	1.025	0.0	0.0	nor...	notp...	notp...	108.0	75.0	1.9	141.0	5.2	9.9	29.0	840...	3.7	yes	yes	no	good	no	yes	ckd	
27	26.0	75.0	80.0	1.015	0.0	0.0	nor...	notp...	notp...	156.0	45.0	2.4	140.0	3.4	11.6	35.0	103...	4.0	yes	yes	no	poor	no	no	ckd	
28	27.0	69.0	70.0	1.01	3.0	4.0	nor...	abn...	notp...	264.0	87.0	2.7	130.0	4.0	12.5	37.0	960...	4.1	yes	yes	yes	good	yes	no	ckd	
29	28.0	75.0	70.0	1.01	1.0	3.0	nor...	nor...	notp...	123.0	31.0	1.4	137...	4.62...	12.52...	38.8...	840...	4.70...	no	yes	yes	yes	good	yes	no	ckd

Add instance Undo OK Cancel Log x 0

## 1.3 Discretization

An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes. Discretization is by simple binning. Skips the class attribute if set.

Apply weka.filters.unsupervised.attribute.Discretize on Iris dataset, take a screen shot of output.

**Insert here your sample Input and Output**

input.csv

Viewer

Relation: pima\_diabetes

No:	1: preg	2: plas	3: pres	4: skin	5: insu	6: mass	7: pedi	8: age	9: class
1	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	teste...
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	teste...
3	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	teste...
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	teste...
5	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	teste...
6	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	teste...
7	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	teste...
8	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	teste...
9	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	teste...
10	8.0	125.0	96.0	0.0	0.0	0.0	0.232	54.0	teste...
11	4.0	110.0	92.0	0.0	0.0	37.6	0.191	30.0	teste...
12	10.0	168.0	74.0	0.0	0.0	38.0	0.537	34.0	teste...
13	10.0	139.0	80.0	0.0	0.0	27.1	1.441	57.0	teste...
14	1.0	189.0	60.0	23.0	846.0	30.1	0.398	59.0	teste...
15	5.0	166.0	72.0	19.0	175.0	25.8	0.587	51.0	teste...
16	7.0	100.0	0.0	0.0	0.0	30.0	0.484	32.0	teste...
17	0.0	118.0	84.0	47.0	230.0	45.8	0.551	31.0	teste...
18	7.0	107.0	74.0	0.0	0.0	29.6	0.254	31.0	teste...
19	1.0	103.0	30.0	38.0	83.0	43.3	0.183	33.0	teste...
20	1.0	115.0	70.0	30.0	96.0	34.6	0.529	32.0	teste...
21	3.0	126.0	88.0	41.0	235.0	39.3	0.704	27.0	teste...
22	8.0	99.0	84.0	0.0	0.0	35.4	0.388	50.0	teste...
23	7.0	196.0	90.0	0.0	0.0	39.8	0.451	41.0	teste...
24	9.0	119.0	80.0	35.0	0.0	29.0	0.263	29.0	teste...
25	11.0	143.0	94.0	33.0	146.0	36.6	0.254	51.0	teste...
26	10.0	125.0	70.0	26.0	115.0	31.1	0.205	41.0	teste...
27	7.0	147.0	76.0	0.0	0.0	39.4	0.257	43.0	teste...
28	1.0	97.0	66.0	15.0	140.0	23.2	0.487	22.0	teste...
29	13.0	145.0	82.0	19.0	110.0	22.2	0.245	57.0	teste...
30	5.0	117.0	87.0	0.0	0.0	34.1	0.227	38.0	teste...

Add instance Undo OK Cancel

## output.csv

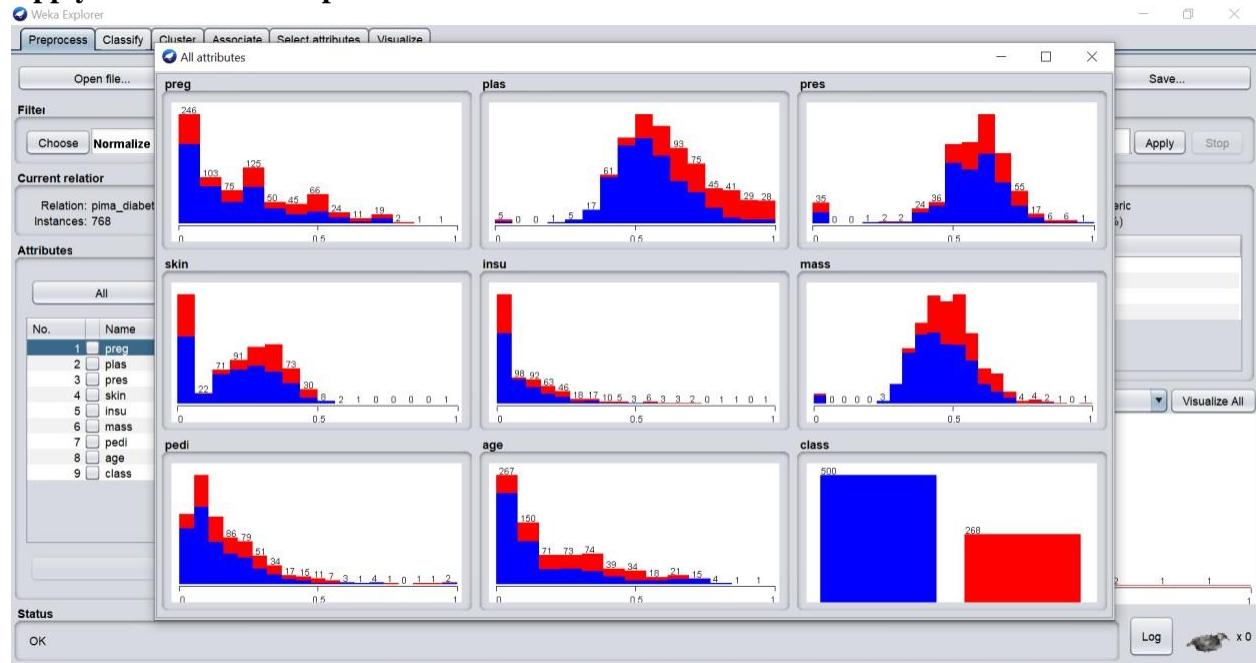
No.	1: preg	2: plas	3: pres	4: skin	5: insu	6: mass	7: pedi	8: age	9: class
	Normal								
1	(5.1...	(13...	(61...	(29...	'-inf...	(33.5...	(0.5...	(45...	teste...
2	'-inf...	(79...	(61...	(19...	'-inf...	(20.1...	(0.3...	(27...	teste...
3	(6.8...	(17...	(61...	'-inf...	(20.1...	(0.5...	(27...	teste...	
4	'-inf...	(79...	(61...	(19...	(84...	(26.8...	'-inf...	'-inf...	teste...
5	'-inf...	(11...	(36...	(29...	(84...	(40.2...	(2.1...	(27...	teste...
6	(3.4...	(99...	(73...	'-inf...	(20.1...	'-inf...	(27...	teste...	
7	(1.7...	(59...	(48...	(29...	(84...	(26.8...	'-inf...	'-inf...	teste...
8	(8.5...	(99...	'-inf...	'-inf...	(33.5...	'-inf...	(27...	teste...	
9	(1.7...	(17...	(61...	(39...	(50...	(26.8...	'-inf...	(51...	teste...
10	(6.8...	(11...	(85...	'-inf...	'-inf...	'-inf...	(51...	teste...	
11	(3.4...	(99...	(85...	'-inf...	'-inf...	(33.5...	'-inf...	(27...	teste...
12	(8.5...	(15...	(73...	'-inf...	'-inf...	(33.5...	(0.3...	(33...	teste...
13	(8.5...	(11...	(73...	'-inf...	'-inf...	(26.8...	(1.2...	(51...	teste...
14	'-inf...	(17...	(48...	(19...	(76...	(26.8...	(0.3...	(57...	teste...
15	(3.4...	(15...	(61...	(9.9...	(16...	(20.1...	(0.5...	(45...	teste...
16	(6.8...	(99...	'-inf...	'-inf...	(26.8...	(0.3...	(27...	teste...	
17	'-inf...	(99...	(73...	(39...	(16...	(40.2...	(0.5...	(27...	teste...
18	(6.8...	(99...	(73...	'-inf...	'-inf...	(26.8...	'-inf...	(27...	teste...
19	'-inf...	(99...	(24...	(29...	'-inf...	(40.2...	'-inf...	(27...	teste...
20	'-inf...	(99...	(61...	(29...	(84...	(33.5...	(0.3...	(27...	teste...
21	(1.7...	(11...	(85...	(39...	(16...	(33.5...	(0.5...	'-inf...	teste...
22	(6.8...	(79...	(73...	'-inf...	'-inf...	(33.5...	(0.3...	(45...	teste...
23	(6.8...	(17...	(85...	'-inf...	'-inf...	(33.5...	(0.3...	(39...	teste...
24	(8.5...	(99...	(73...	(29...	'-inf...	(26.8...	'-inf...	(27...	teste...
25	(10...	(13...	(85...	(29...	(84...	(33.5...	'-inf...	(45...	teste...
26	(8.5...	(11...	(61...	(19...	(84...	(26.8...	'-inf...	(39...	teste...
27	(6.8...	(13...	(73...	'-inf...	'-inf...	(33.5...	'-inf...	(39...	teste...
28	'-inf...	(79...	(61...	(9.9...	(84...	(20.1...	(0.3...	'-inf...	teste...
29	(11...	(13...	(73...	(9.9...	(84...	(20.1...	'-inf...	(51...	teste...
30	'-inf...	teste...							

Add instance Undo OK Cancel

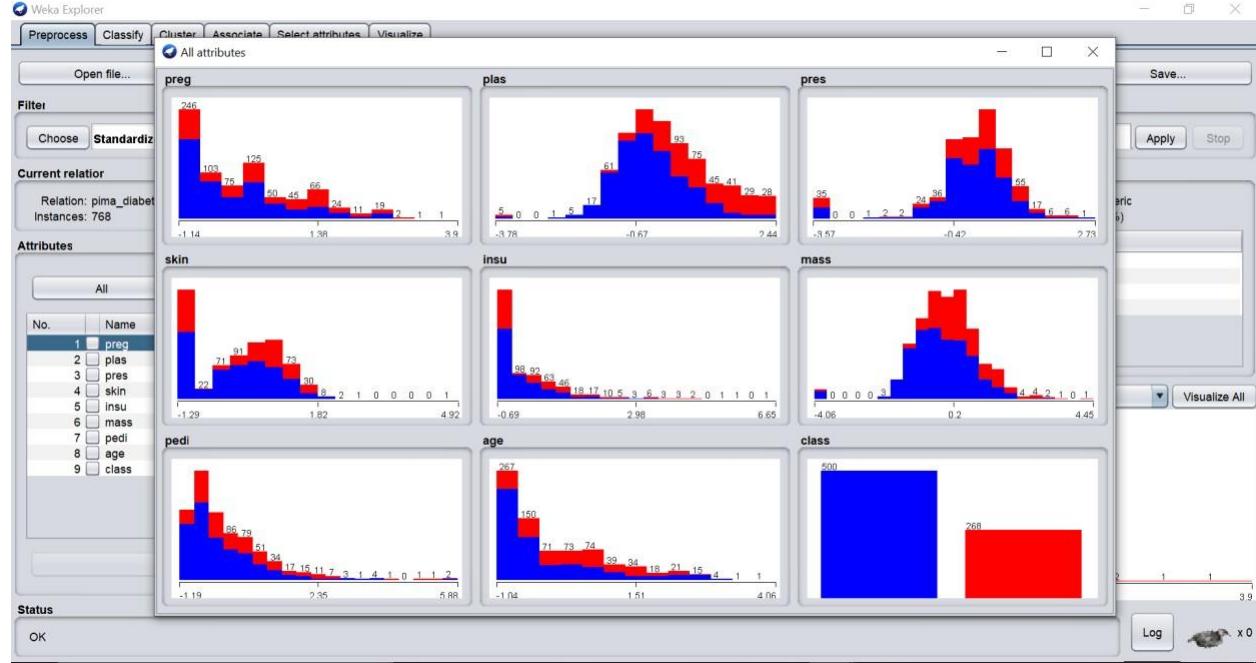
## 1.4 Normalisation

Where the attribute data are scaled so as to fall within a smaller range, such as 1.0 to 1.0, or 0.0 to 1.0.

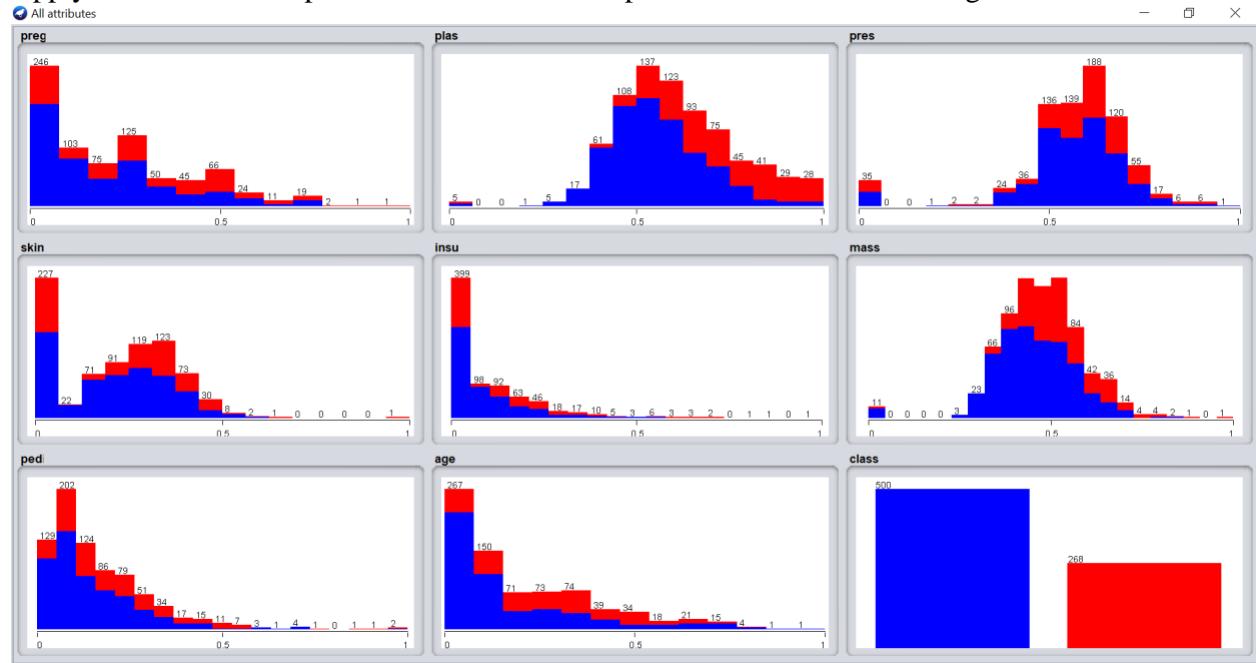
### Apply weka.filters.unsupervised.attribute.Normalize for Min-Max Normalization



## Apply weka.filters.unsupervised.attribute.Standardize (Z-score)



## Apply weka.filters.unsupervised.attribute.MathExpression for Decimal Scaling



**Fill the table below: (for diabetes.arff)**

Attribute(Initial scale)	Min-Max	Standardize	Decimal Scaling
preg (0-17)	0-1	-1.141- 3.904	0 - 1
plas(0-199)	0-1	-3.781- 2.443	0 - 1
pres(0-122)	0-1	-3.57- 2.733	0 - 1
skin(0-99)	0-1	-1.287- 4.919	0 - 1
insu(0-846)	0-1	-0.692- 6.649	0 - 1
mass(0-67.1)	0-1	-4.058- 4.453	0 - 1
pedi(0.078-2.42)	0-1	-1.189- 5.88	0 - 1
age(21-81)	0-1	-1.041- 4.061	0 - 1

## 1.5 Sampling

Apply weka.filters.unsupervised.instance.Resample on iris dataset

Produces a random subsample of a dataset using either sampling with replacement or without replacement.

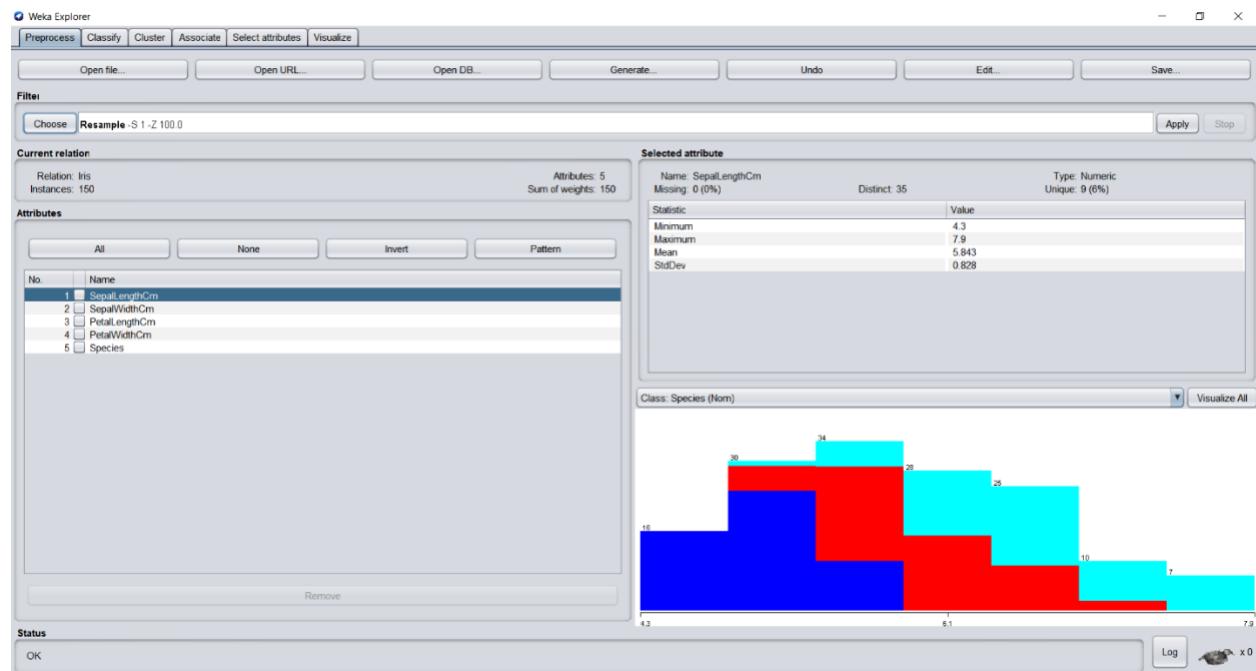
sampleSizePercent – Set Size of the subsample as a percentage of the original data say 50%.

**Output: Fill the table below with No of tuples in your input and output .**

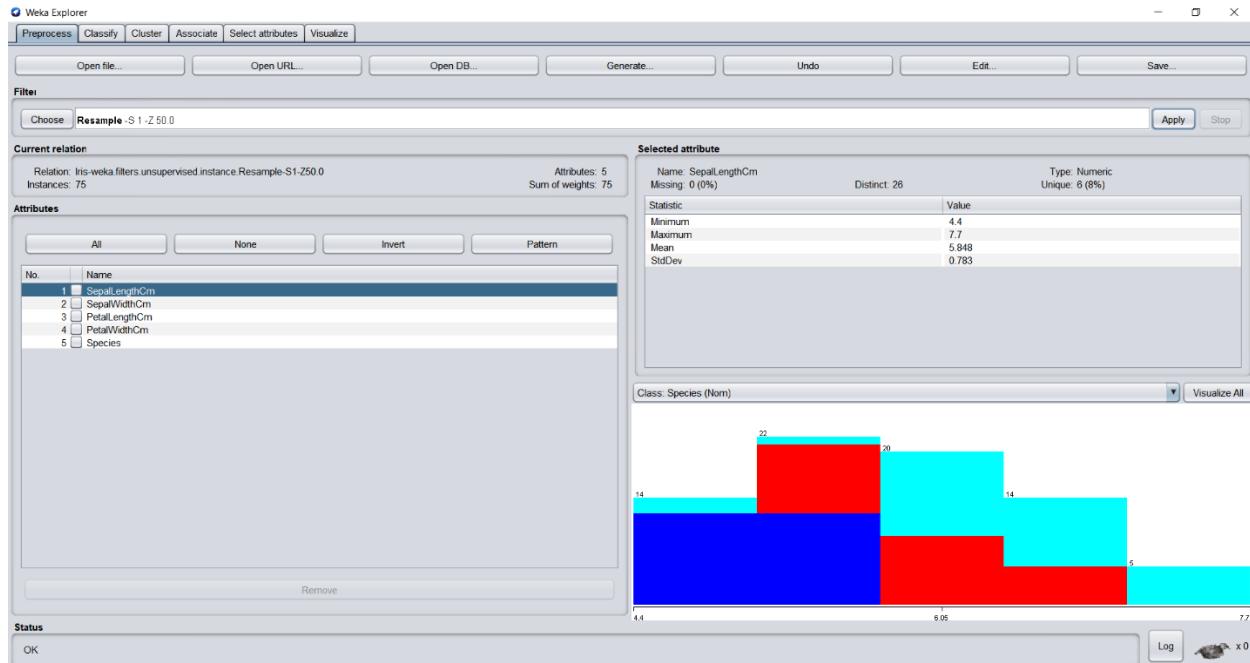
And take a scree shot of input and output and paste here

No. of tuples Before Sampling	No. of tuples After Sampling
150	150

## Input



## Output

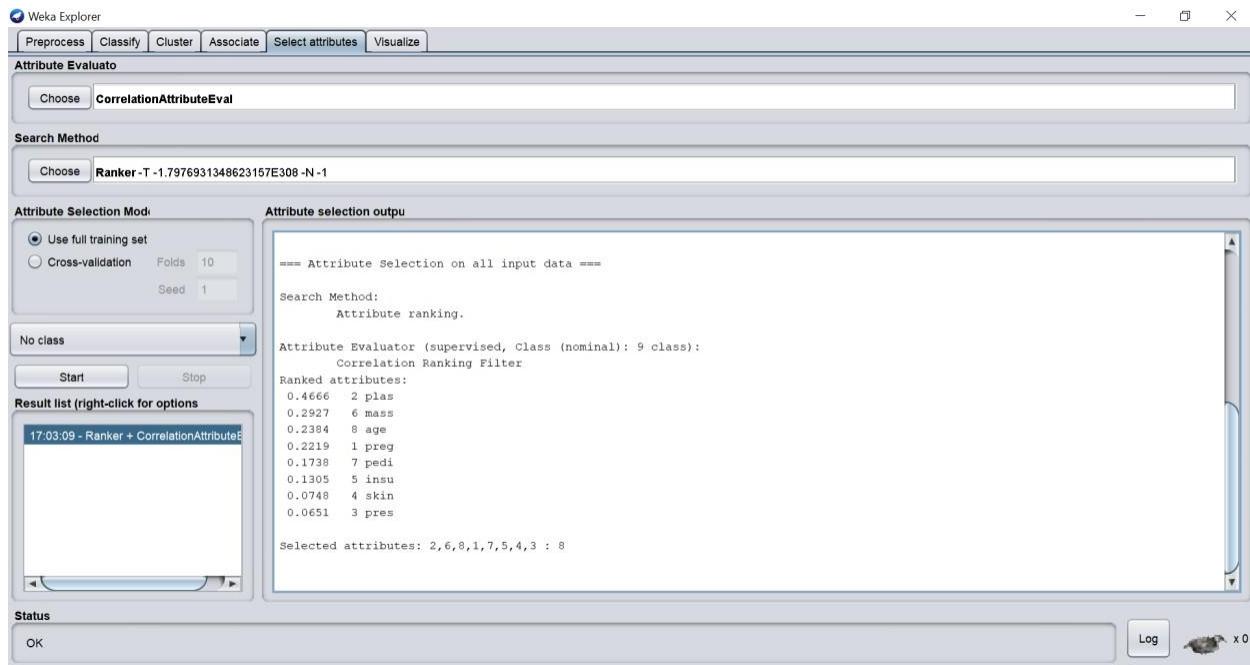


## 1.5 Perform Feature Selection

Not all of the attributes in your dataset may be relevant to the attribute you want to predict. You can use feature selection to identify those attributes that are most relevant to your output variable. The following procedure can be used for feature selection

1. Open the Weka GUI Chooser and then the Weka Explorer.
2. Load the iris.arff dataset.
3. Click the Select attributes tab.
4. Click the Choose button in the Attribute Evaluator pane and select the CorrelationAttributeEval.  
(a) You will be presented with a dialog to change to the Ranker search method, needed when using this feature selection method. Click the Yes button.
5. Click the Start button to run the feature selection method.

### Determining the Correlation score of attributes in diabetes dataset



Review the output in the Attribute selection output pane and note the correlation scores for each attribute, the larger numbers indicating the more relevant features. Explore other feature selection methods such as the use of information gain (entropy). Using feature selection, find out the important features and remove other features from dataset using remove button in Preprocess tab.

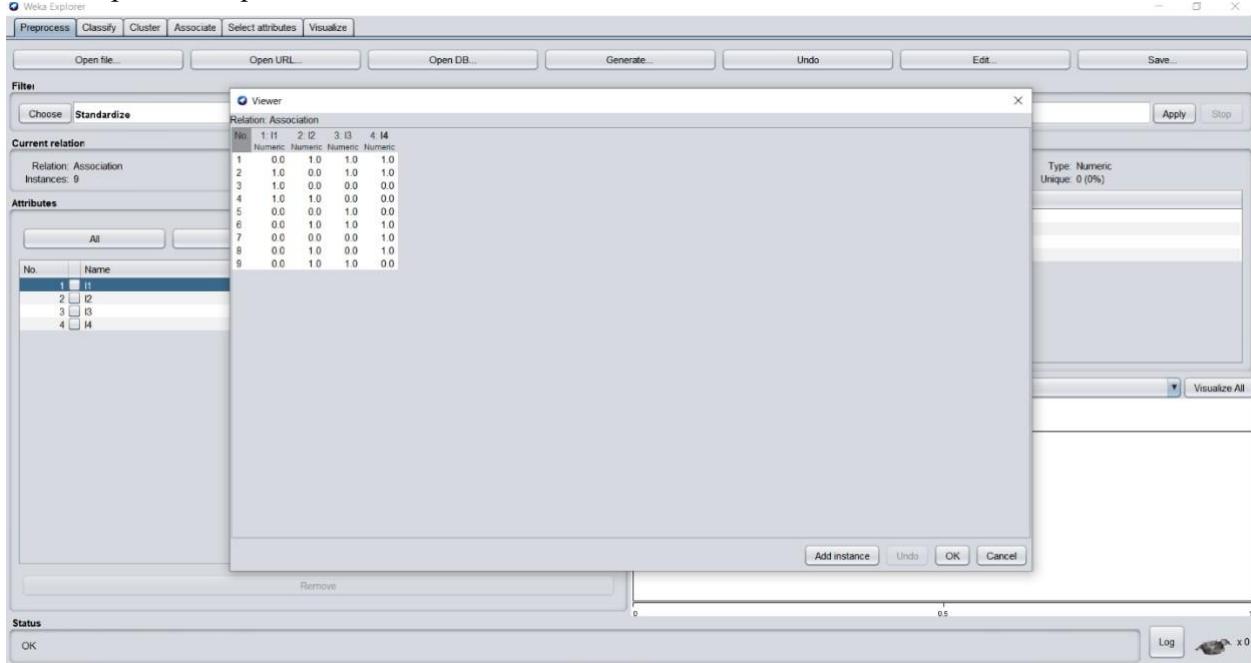
## 2. ASSOCIATION MINING

### Step by Step Procedure

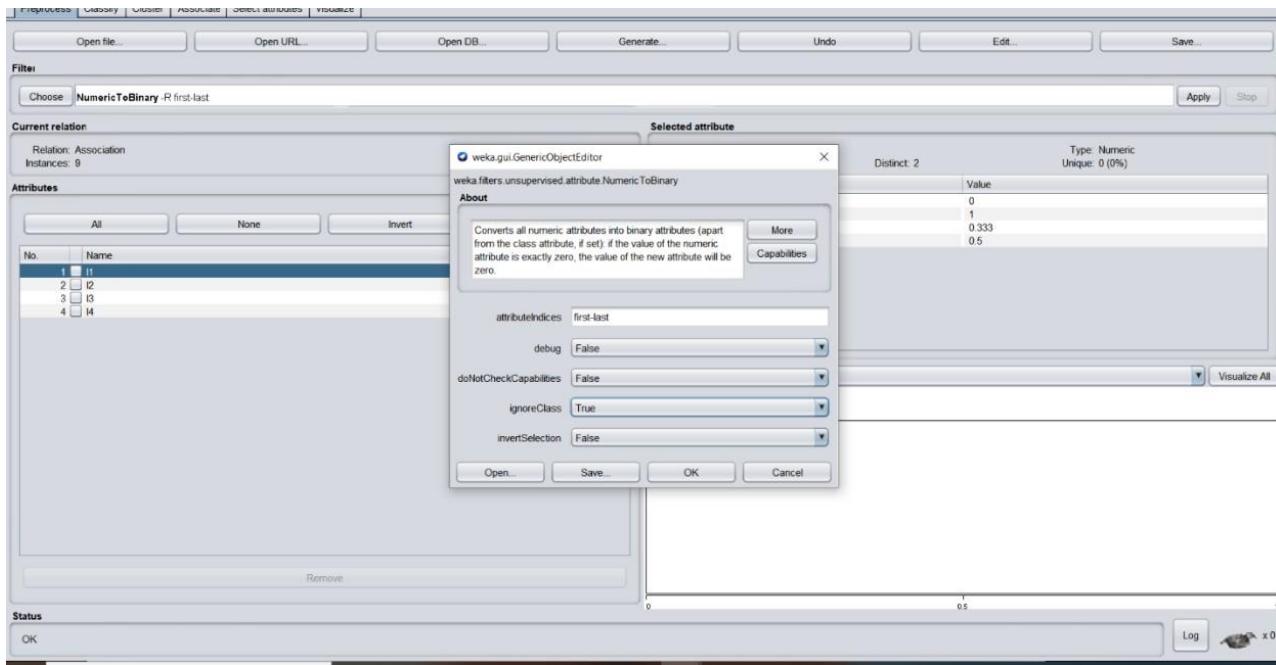
Practice these steps with your own dataset and upload the screen shots after each step here.

Find Frequent patterns using Apriori and FP Growth algorithms and generate strong association rules using WEKA on the following dataset 2.1 Apriori algorithm

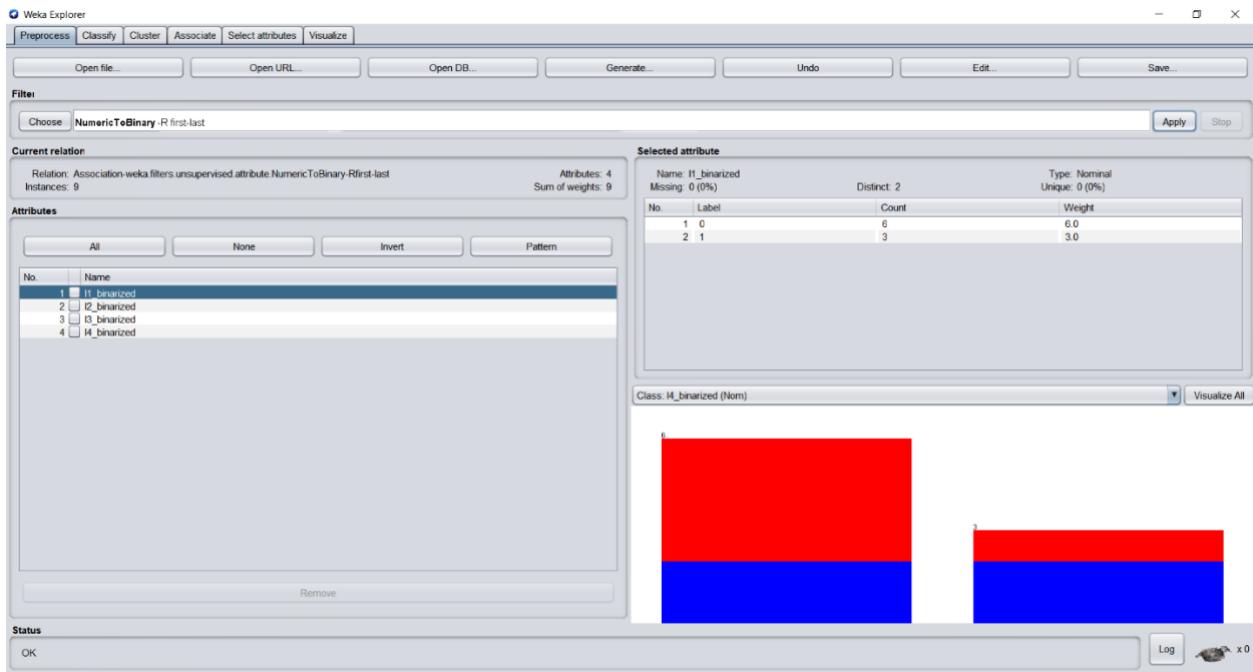
- i. Open the required .csv file and click on ‘edit’ to view the dataset.



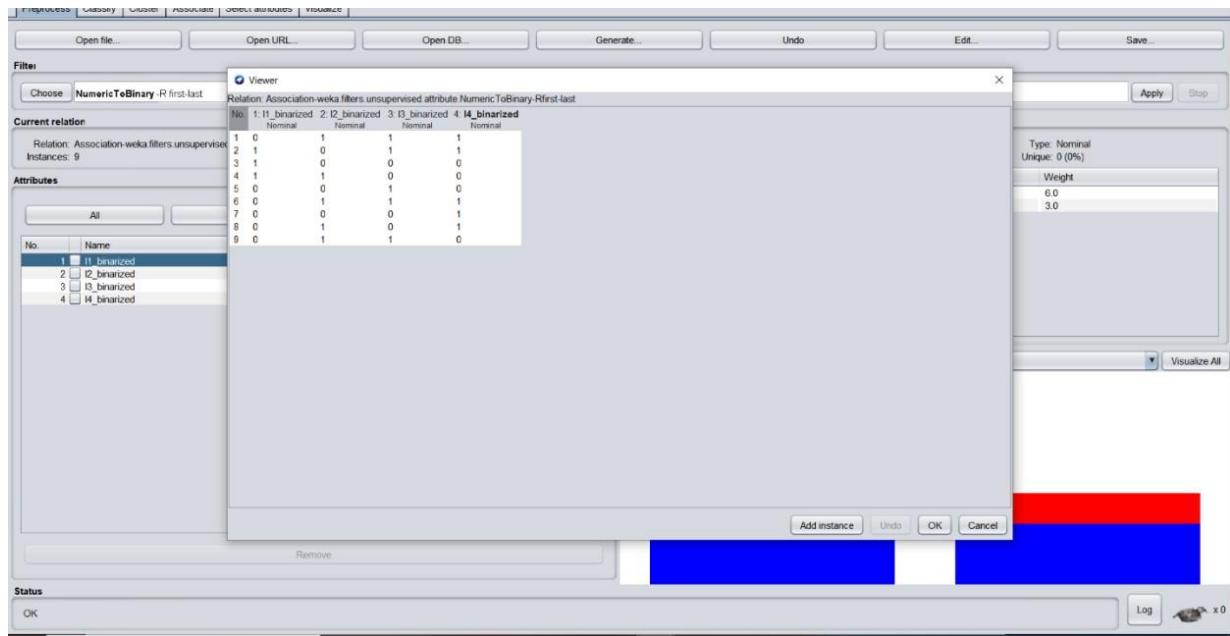
- ii. Use NumericToBinary filter to convert the given numeric data to binary and click on that text box to see the description of that filter and change the settings of that filter.



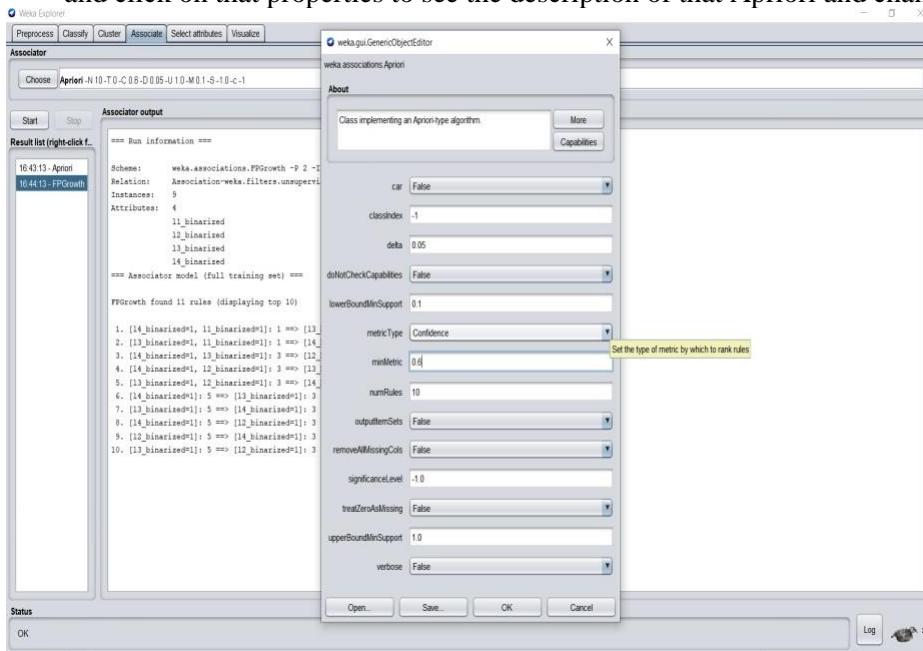
iii. In the below window we can see the binarized attributes.



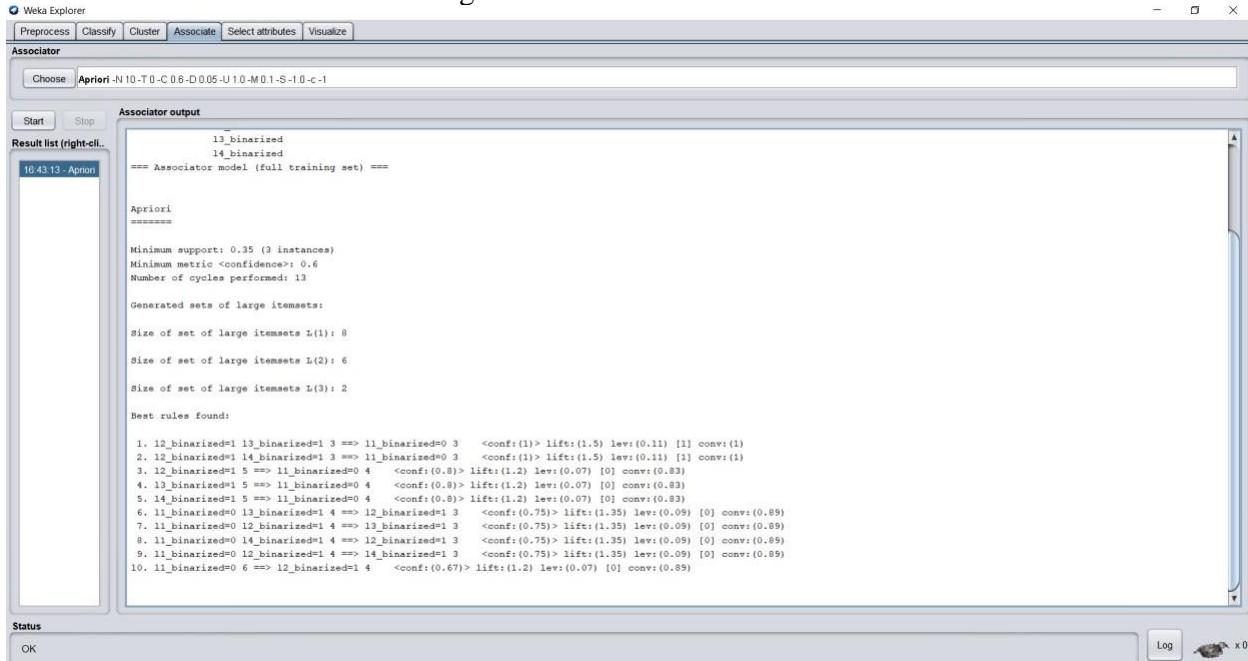
iv. Click on edit to check the dataset values after conversion.



v. Go to Associate tab and click on choose to select Apriori Algorithm on the given data set and click on that properties to see the description of that Apriori and change the settings.



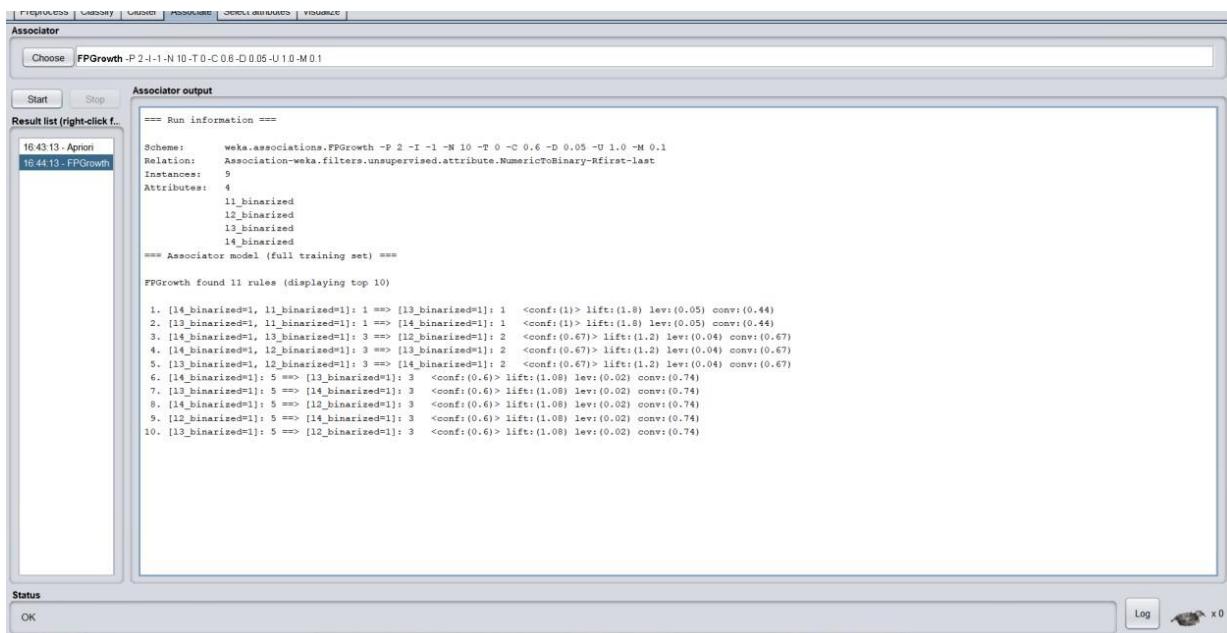
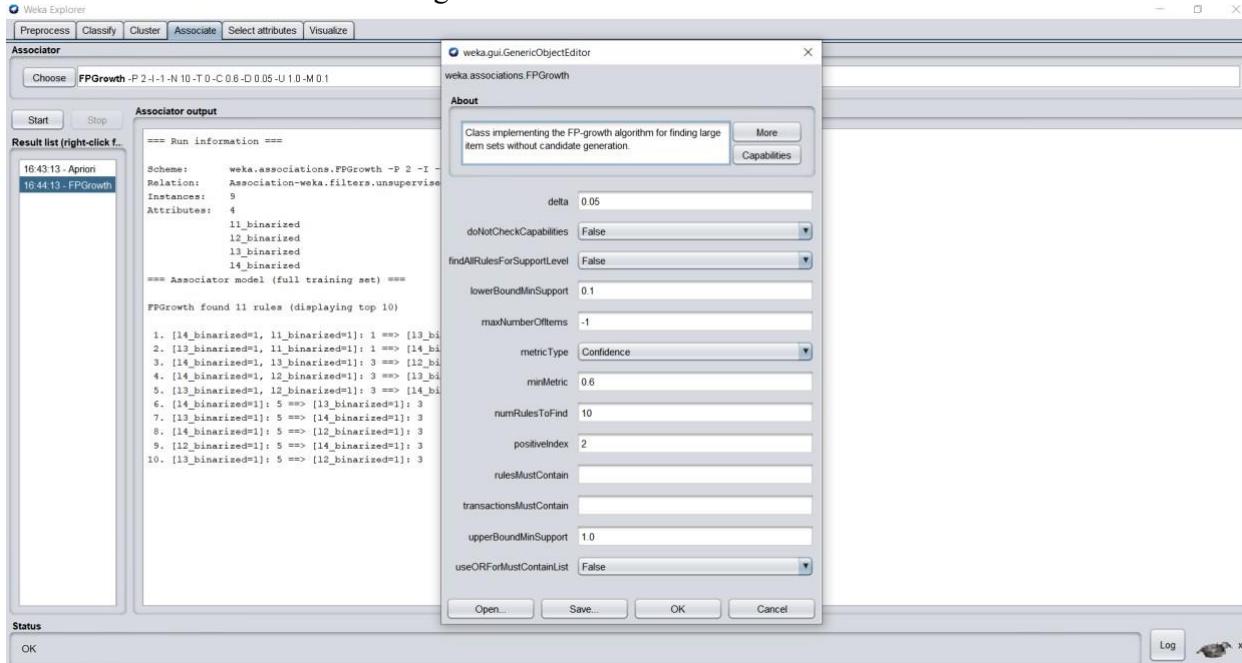
vi. Now click on ‘start’ to start the algorithm



## 2.2 FP-Growth algorithm

i. Repeat above (i) to (iv) steps and go to Associate tab and choose to select FP Growth Algorithm on the given data set and click on properties to see the description of that FP Growth and change the settings.

ii. Now click on ‘start’ to start the algorithm



### 3.CLASSIFICATION

Classification is a two step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data).

#### Dataset:iris.arff

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

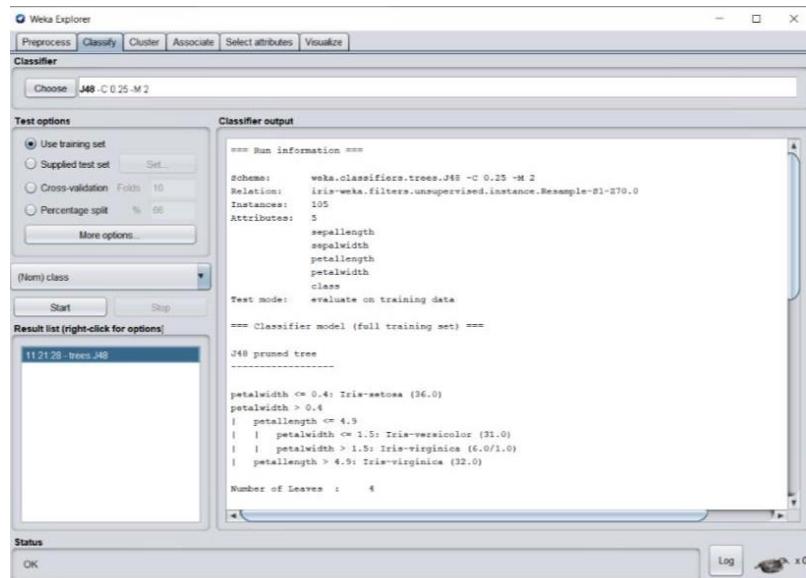
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

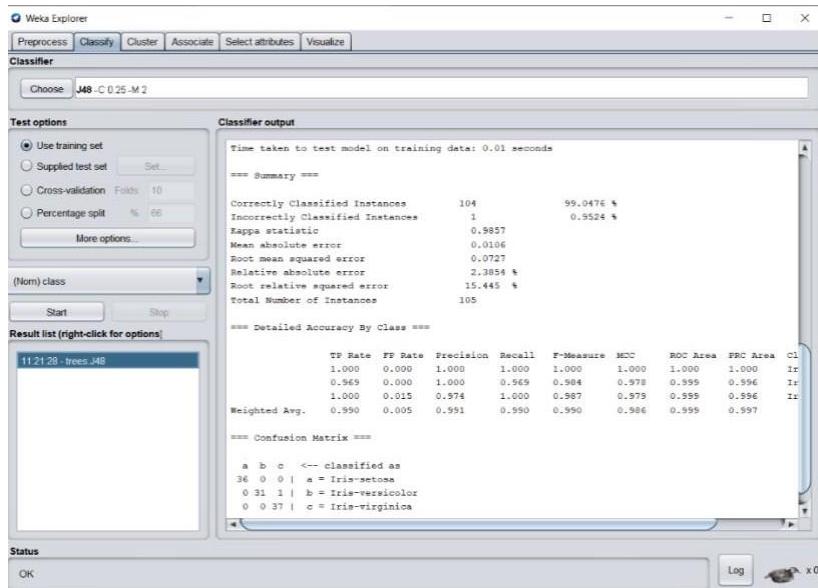
#### 3.1 Desicion Tree Algorithm:

Decision tree is a flow chart like structure where each internal node denote test on attribute , each branch represents an outcome of test and leaf node represents classes.

Browse the iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize% = 70 and apply Now save the file as iris\_train.arff , again select iris.arff apply Resample and give samplesize% =30 and apply save the file as iris\_test.arff Browse iris\_train.arff apply weka.classifiers.trees.J48.

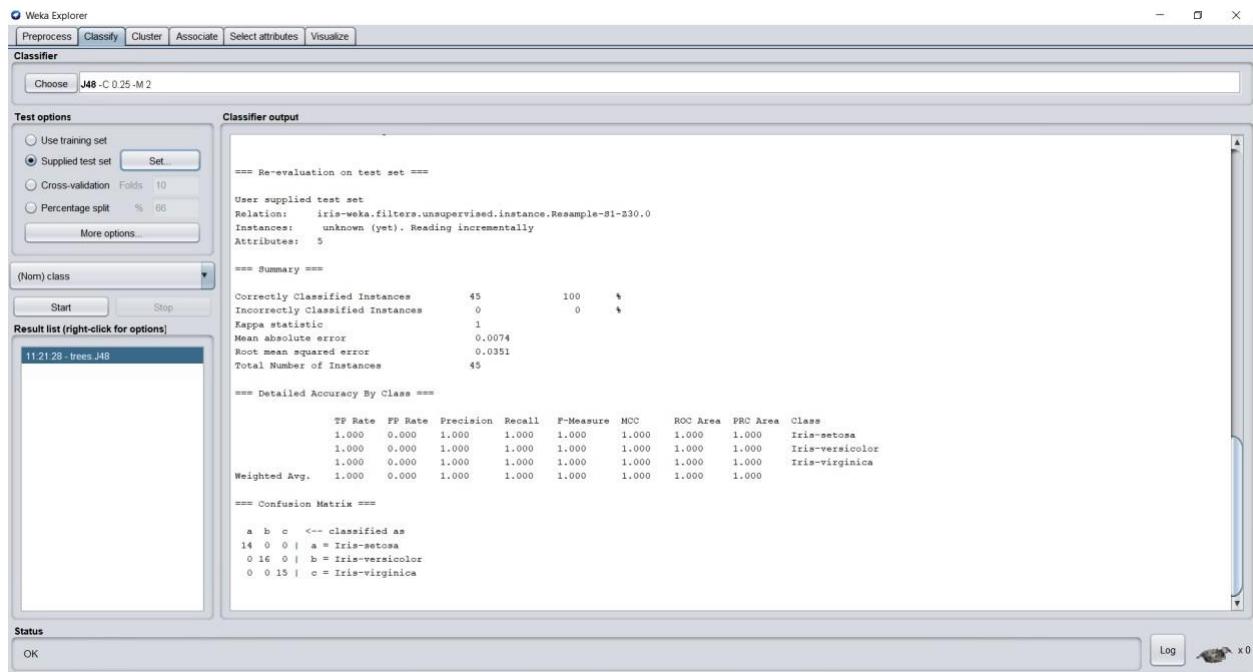
**Take a screen shot of your output and paste here**





Click on supplied set > set >open file > iris\_test.arff. Right click & re-evaluate model on current test set

### Take a screen shot of your output and paste here



And observe accuracy for given dataset.(in %).-100%

## Prediction

Prediction is a process of predicting the values of data by making use of known results from a different set of sample data. Click on supplied set > set > open file > iris\_prediction.arff

Click on more options > choose plain text .click ok and start

**Take a screen shot of your output and paste here**

Iris\_prediction input

No.	1: sepal length	2: sepal width	3: petal length	4: petal width	5: class
	Numeric	Numeric	Numeric	Numeric	Nominal
1	5.1	3.5	1.4	0.2	Iris-v.
2	4.9	3.0	1.4	0.2	Iris-v.
3	4.7	3.2	1.3	0.2	Iris-v.
4	4.6	3.1	1.5	0.2	Iris-v.
5	5.0	3.6	1.4	0.2	Iris-v.
6	5.4	3.9	1.7	0.4	Iris-v.
7	4.6	3.4	1.4	0.2	Iris-v.
8	4.5	2.3	1.3	0.3	Iris-v.
9	4.9	3.0	1.5	0.2	Iris-v.
10	4.7	3.2	1.3	0.2	Iris-v.
11	4.6	3.1	1.5	0.2	Iris-v.
12	5.0	3.6	1.4	0.2	Iris-v.
13	5.4	3.9	1.7	0.4	Iris-v.
14	4.6	3.4	1.4	0.2	Iris-v.
15	4.8	3.0	1.4	0.2	Iris-v.
16	4.9	3.1	1.5	0.1	Iris-s.
17	5.0	3.6	1.4	0.2	Iris-v.
18	4.8	3.0	1.4	0.1	Iris-s.
19	5.7	2.6	4.2	1.3	Iris-v.
20	5.5	2.7	4.0	1.3	Iris-v.
21	5.0	3.4	1.5	0.4	Iris-s.
22	5.2	4.1	1.5	0.1	Iris-s.
23	5.4	3.7	1.5	0.2	Iris-s.
24	5.9	3.0	5.1	1.8	Iris-v.
25	6.4	2.9	4.3	1.3	Iris-v.
26	5.2	2.7	3.9	1.4	Iris-v.
27	6.2	3.4	5.4	2.0	Iris-v.
28	5.5	2.3	4.0	1.3	Iris-v.
29	5.3	2.3	4.4	1.3	Iris-v.
30	4.7	3.2	1.3	0.2	Iris-v.

output

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose J48 -C 0.25 -M 2

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)

11:21:28 - trees.J48  
11:30:16 - trees J48

Classifier output

```
==== Run information ====
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: iris-weka.filters.unsupervised.instance.Resample-S1-Z70.0
Instances: 105
Attributes: 5
Class: class

Test mode: user supplied test set: size unknown (reading incrementally)

==== Classifier model (full training set) ====
J48 pruned tree
-----
petalwidth <= 0.4: Iris-setosa (36.0)
petalwidth > 0.4
|  petallength <= 4.9
|  |  petalwidth <= 1.5: Iris-versicolor (31.0)
|  |  petalwidth > 1.5: Iris-virginica (6.0/1.0)
|  petallength > 4.9: Iris-virginica (32.0)

Number of Leaves : 4
Size of the tree : 7

Time taken to build model: 0 seconds

==== Predictions on test set ====

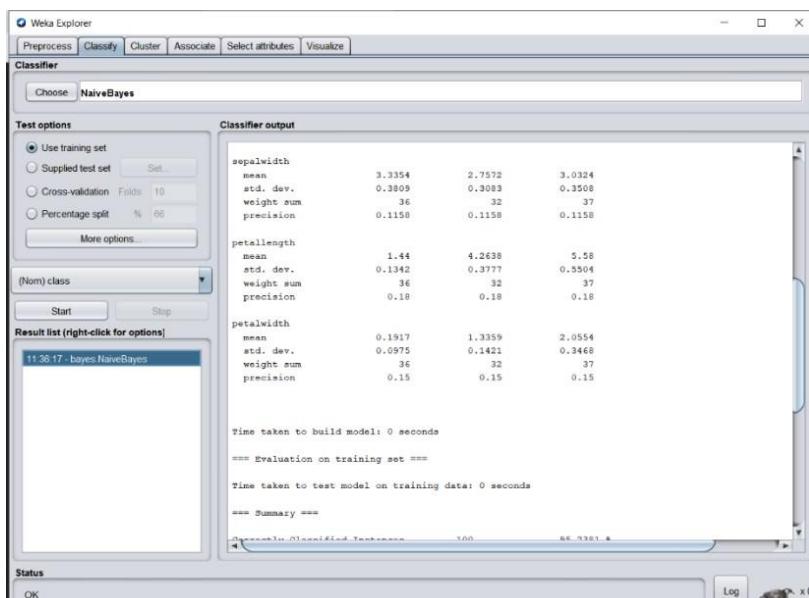
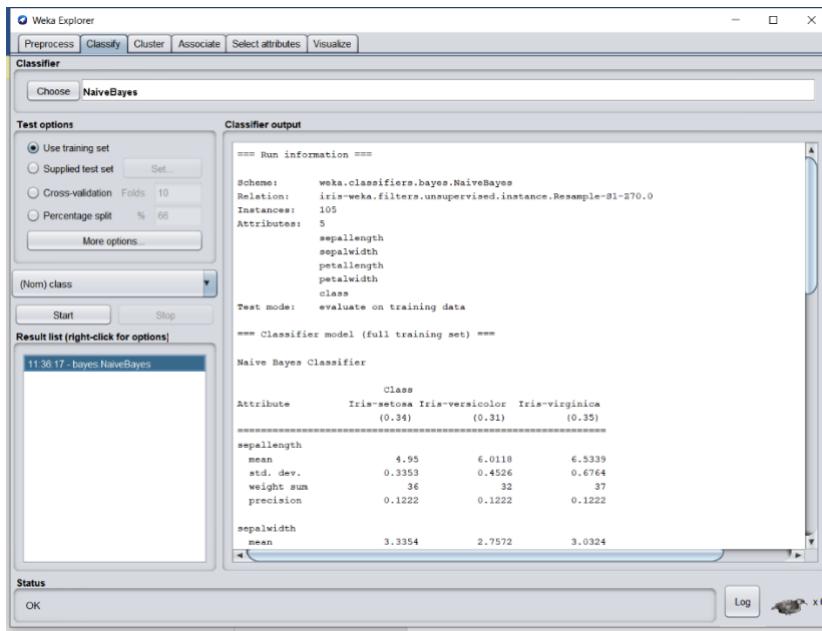
Status OK Log x 0
```

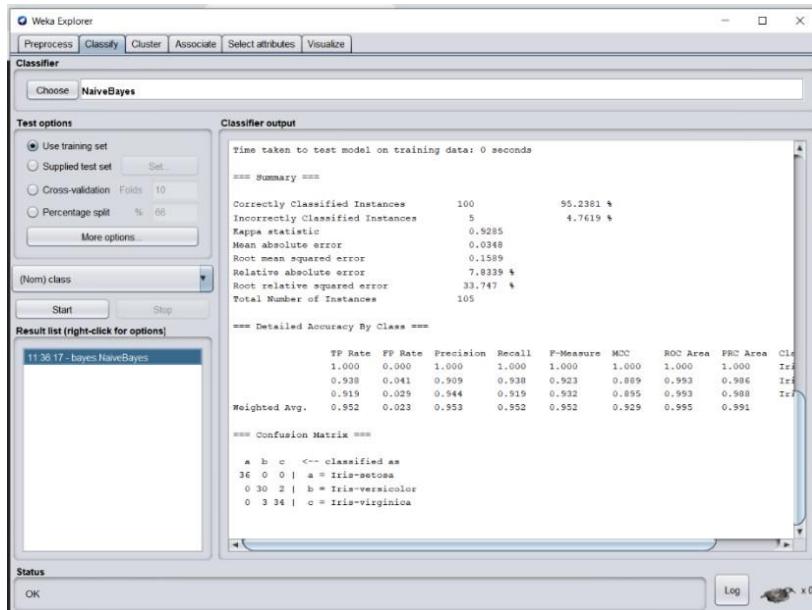
## 1. Naive Bayes Algorithm:

This algorithm is based on Bayes Theorem.

Browse the iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize% = 70 and apply now save the file as iris\_train.arff. Again select iris.arff apply Resample and give samplesize% = 30 and apply save the file as iris\_test.arff .Browse iris\_train.arff apply weka.classifiers.bayes.naivebayes.

**Take a screen shot of your output and paste here**



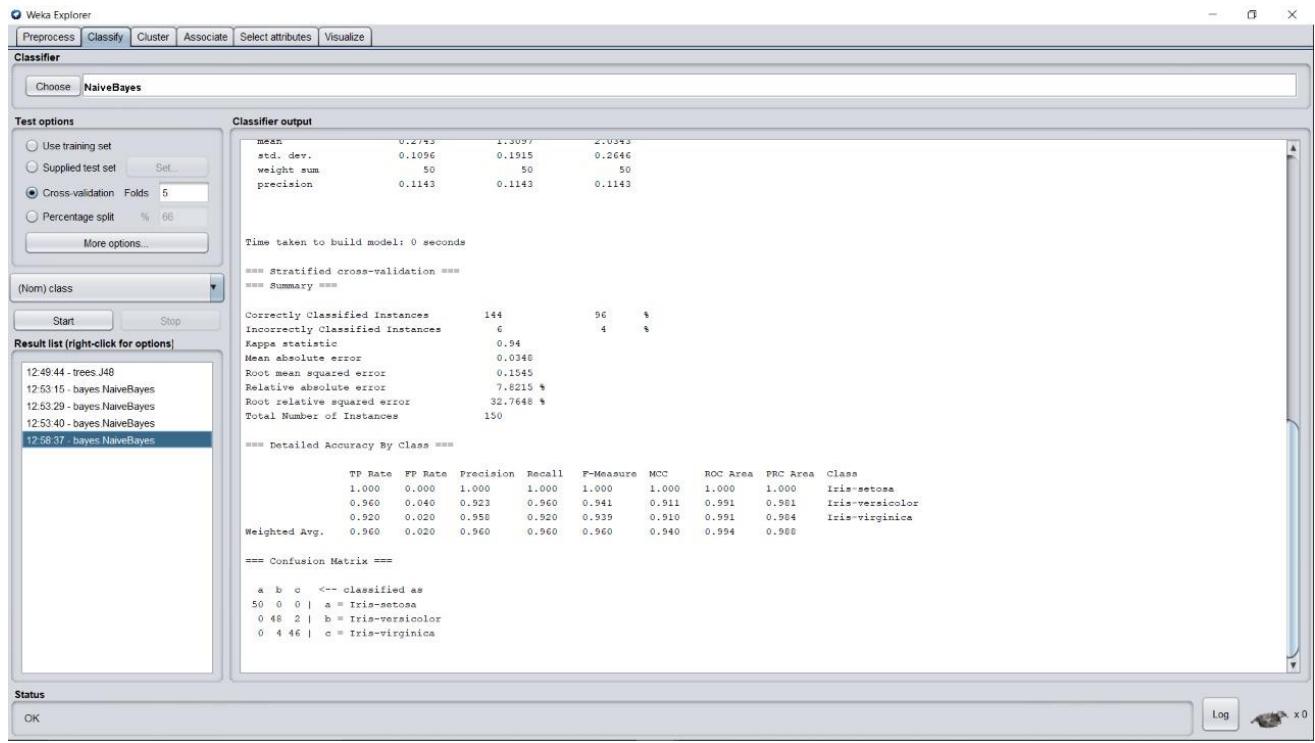


**Practice these additional Exercises and observe the output**

### Cross Validation:

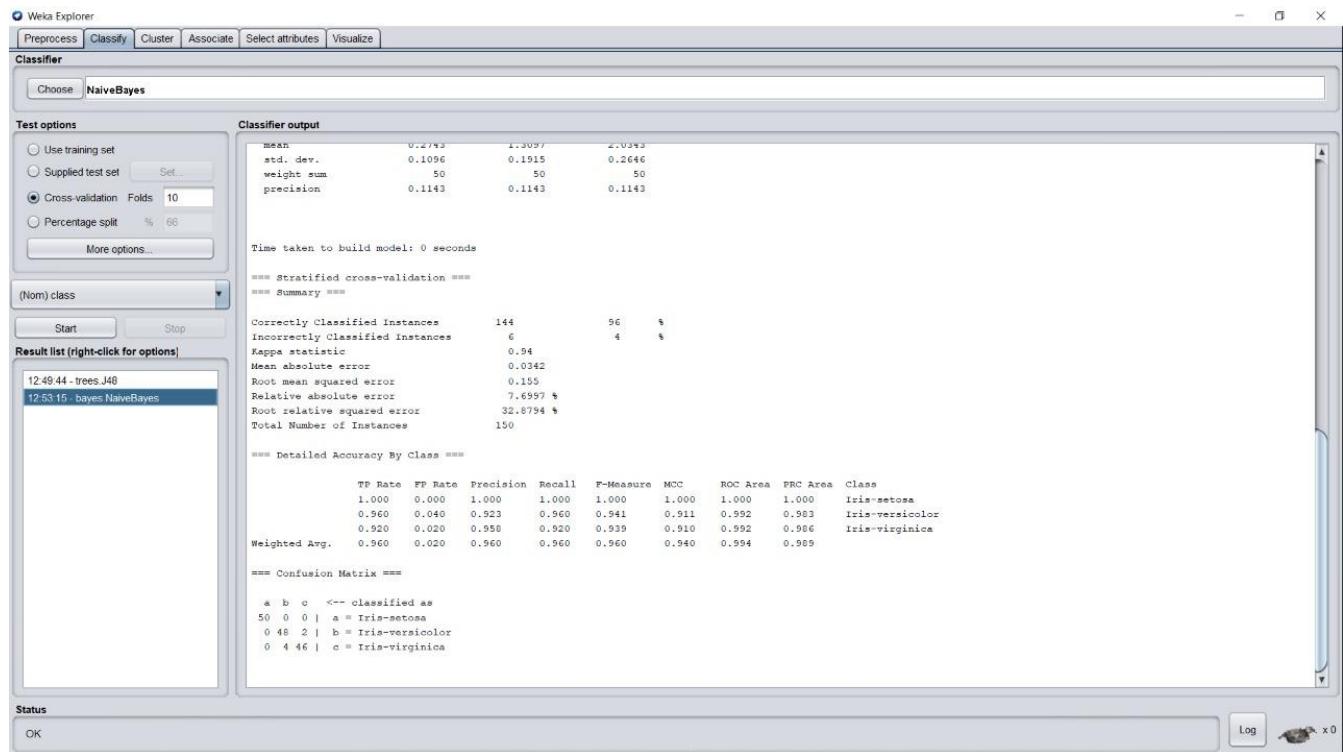
In **K-fold cross-validation**, the initial data are randomly partitioned into  $K$  mutually exclusive subsets or “folds,”  $D_1, D_2, \dots, D_K$ , each of approximately equal size. Training and testing is performed  $K$  times. In iteration  $I$ , partition  $D_I$  is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets  $D_2, \dots, D_K$  collectively serve as the training set to obtain a first model, which is tested on  $D_1$ ; the second iteration is trained on subsets  $D_1, D_3, \dots, D_K$  and tested on  $D_2$ ; and so on. the accuracy estimate is the overall number of correct classifications from the  $K$  iterations, divided by the total number of tuples in the initial data.

No of folds	Accuracy
5	96 %
10	96 %
15	95.33 %

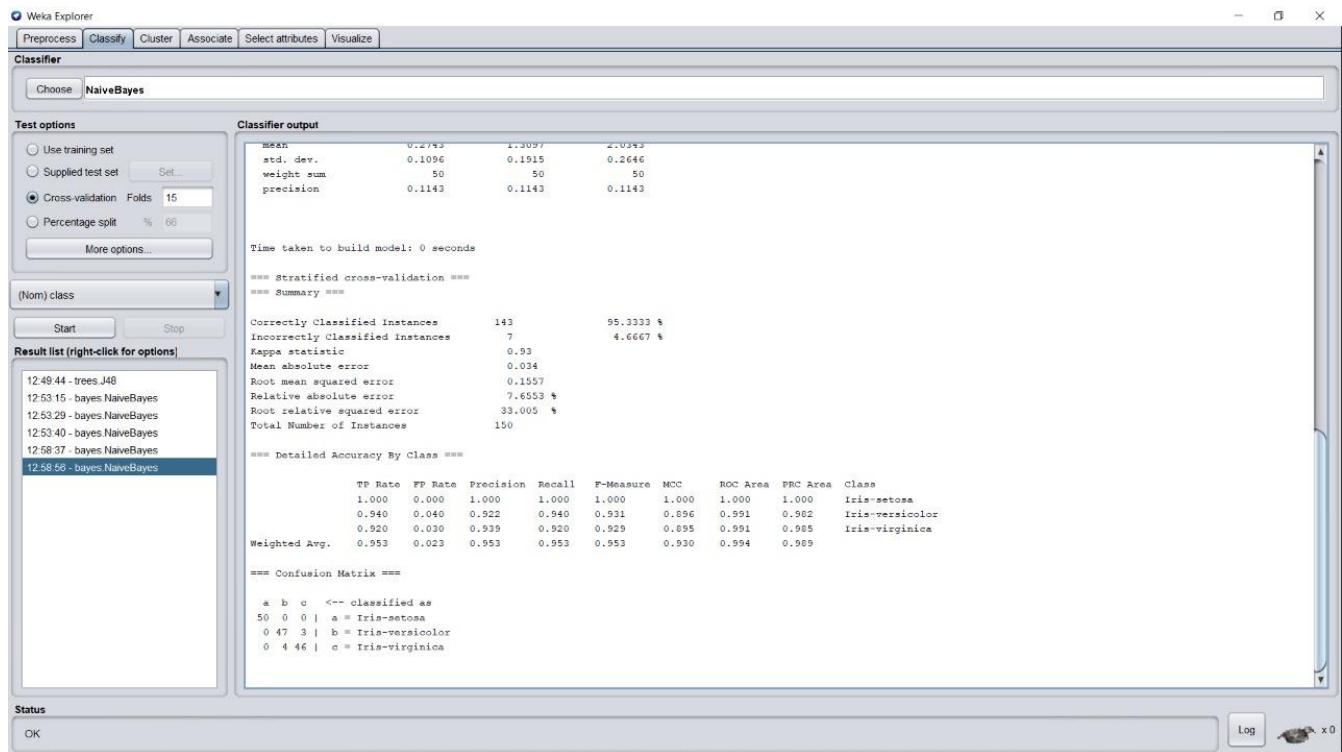


5-folds:

10-folds:



15-folds:



### 3.3KNN

KNN searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k nearest neighbours of the unknown tuple.

Browse iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize% = 70 and apply Now save the file as **iris\_train.arff**, Again select iris.arff apply Resample and give samplesize% = 30 and apply save the file as **iris\_test.arff**

Browse iris\_train.arff apply weka.classifiers.lazy.IBK

Distance Measure	K v a l u e	Time Taken(sec)	Accuracy %
LinearNNsearch	1	0.02	100%
LinearNNsearch	2	0.02	97.33%
LinearNNsearch	3	0.02	96.6667 %
LinearNNsearch	5	0.02	96%

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

**Classifier**

Choose: IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A weka.core.EuclideanDistance -R first-last"

**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)**

- 12:49:44 - trees J48
- 12:53:15 - bayes NaiveBayes
- 12:53:29 - bayes NaiveBayes
- 12:53:40 - bayes NaiveBayes
- 12:58:37 - bayes NaiveBayes
- 12:58:58 - bayes NaiveBayes
- 13:00:02 - lazy.IBk
- 13:00:47 - lazy.IBk

**Classifier output**

```
==== Classifier model (full training set) ====
IBI instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      143      95.3333 %
Incorrectly Classified Instances    7       4.6667 %
Kappa statistic                   0.93
Mean absolute error               0.0399
Root mean squared error          0.1747
Relative absolute error           8.9763 %
Root relative squared error     37.0695 %
Total Number of Instances        150

==== Detailed Accuracy By Class ====


|               | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC   | ROC Area | PRC Area | Class           |
|---------------|---------|---------|-----------|--------|-----------|-------|----------|----------|-----------------|
| 1.000         | 0.000   | 1.000   | 1.000     | 1.000  | 1.000     | 1.000 | 1.000    | 1.000    | Iris-setosa     |
| 0.940         | 0.040   | 0.922   | 0.940     | 0.931  | 0.896     | 0.952 | 0.887    | 0.887    | Iris-versicolor |
| 0.920         | 0.030   | 0.939   | 0.920     | 0.929  | 0.895     | 0.947 | 0.894    | 0.894    | Iris-virginica  |
| Weighted Avg. | 0.953   | 0.023   | 0.953     | 0.953  | 0.953     | 0.930 | 0.966    | 0.927    |                 |


==== Confusion Matrix ====


| a b c  | -- classified as    |
|--------|---------------------|
| 50 0 0 | a = Iris-setosa     |
| 0 47 3 | b = Iris-versicolor |
| 0 4 46 | c = Iris-virginica  |


```

**Status**

OK Log x 0

k=1

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

**Classifier**

Choose: IBk -K 2 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A weka.core.EuclideanDistance -R first-last"

**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)**

- 12:49:44 - trees J48
- 12:53:15 - bayes NaiveBayes
- 12:53:29 - bayes NaiveBayes
- 12:53:40 - bayes NaiveBayes
- 12:58:37 - bayes NaiveBayes
- 12:58:58 - bayes NaiveBayes
- 13:00:02 - lazy.IBk
- 13:00:47 - lazy.IBk
- 13:01:18 - lazy.IBk

**Classifier output**

```
==== Classifier model (full training set) ====
IBI instance-based classifier
using 2 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      142      94.6667 %
Incorrectly Classified Instances    8       5.3333 %
Kappa statistic                   0.92
Mean absolute error               0.04
Root mean squared error          0.1693
Relative absolute error           8.9989 %
Root relative squared error     35.9168 %
Total Number of Instances        150

==== Detailed Accuracy By Class ====


|               | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC   | ROC Area | PRC Area | Class           |
|---------------|---------|---------|-----------|--------|-----------|-------|----------|----------|-----------------|
| 1.000         | 0.000   | 1.000   | 1.000     | 1.000  | 1.000     | 1.000 | 1.000    | 1.000    | Iris-setosa     |
| 0.960         | 0.060   | 0.889   | 0.960     | 0.923  | 0.884     | 0.963 | 0.911    | 0.911    | Iris-versicolor |
| 0.880         | 0.020   | 0.957   | 0.880     | 0.917  | 0.879     | 0.959 | 0.917    | 0.917    | Iris-virginica  |
| Weighted Avg. | 0.947   | 0.027   | 0.946     | 0.947  | 0.947     | 0.921 | 0.974    | 0.942    |                 |

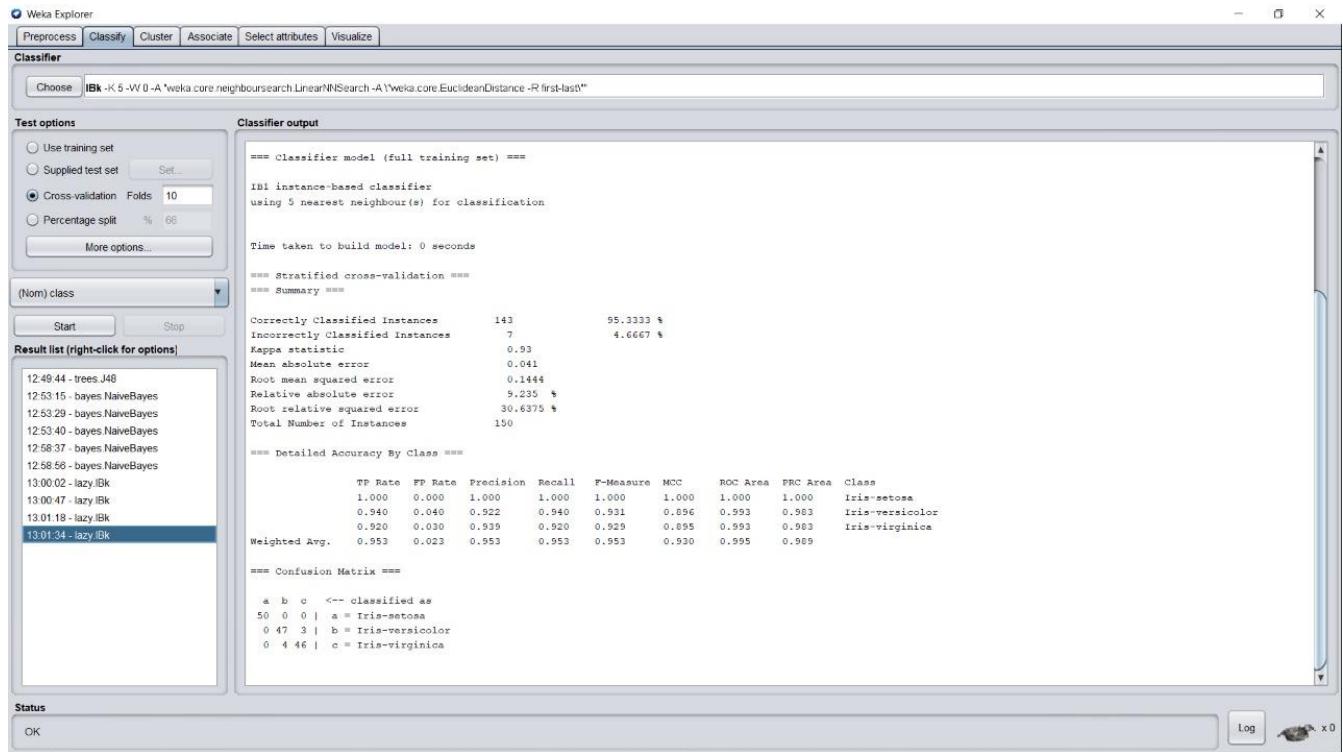

==== Confusion Matrix ====


| a b c  | -- classified as    |
|--------|---------------------|
| 50 0 0 | a = Iris-setosa     |
| 0 48 2 | b = Iris-versicolor |
| 0 6 44 | c = Iris-virginica  |


```

**Status**

OK Log x 0



k=5

### 3.4 Multilayer Perceptron

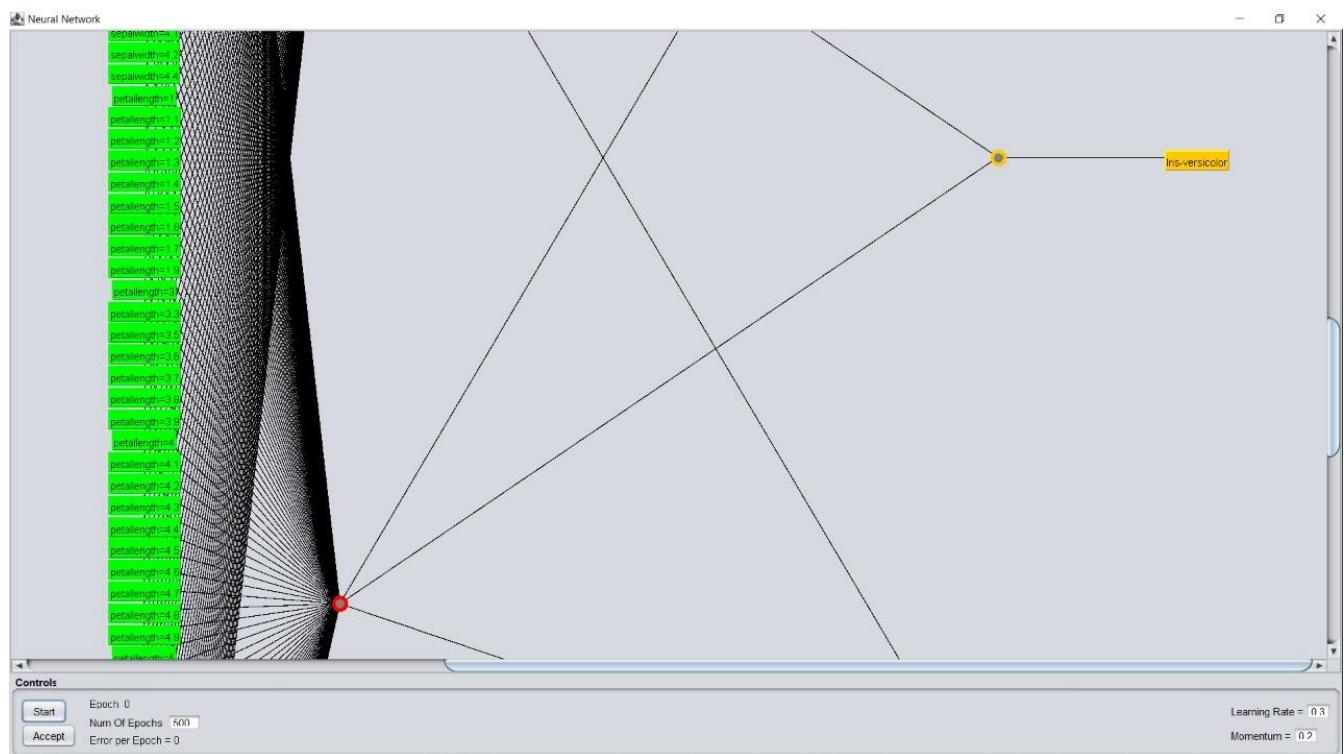
Apply weka.filters.unsupervised.attribute.NumericToNominal

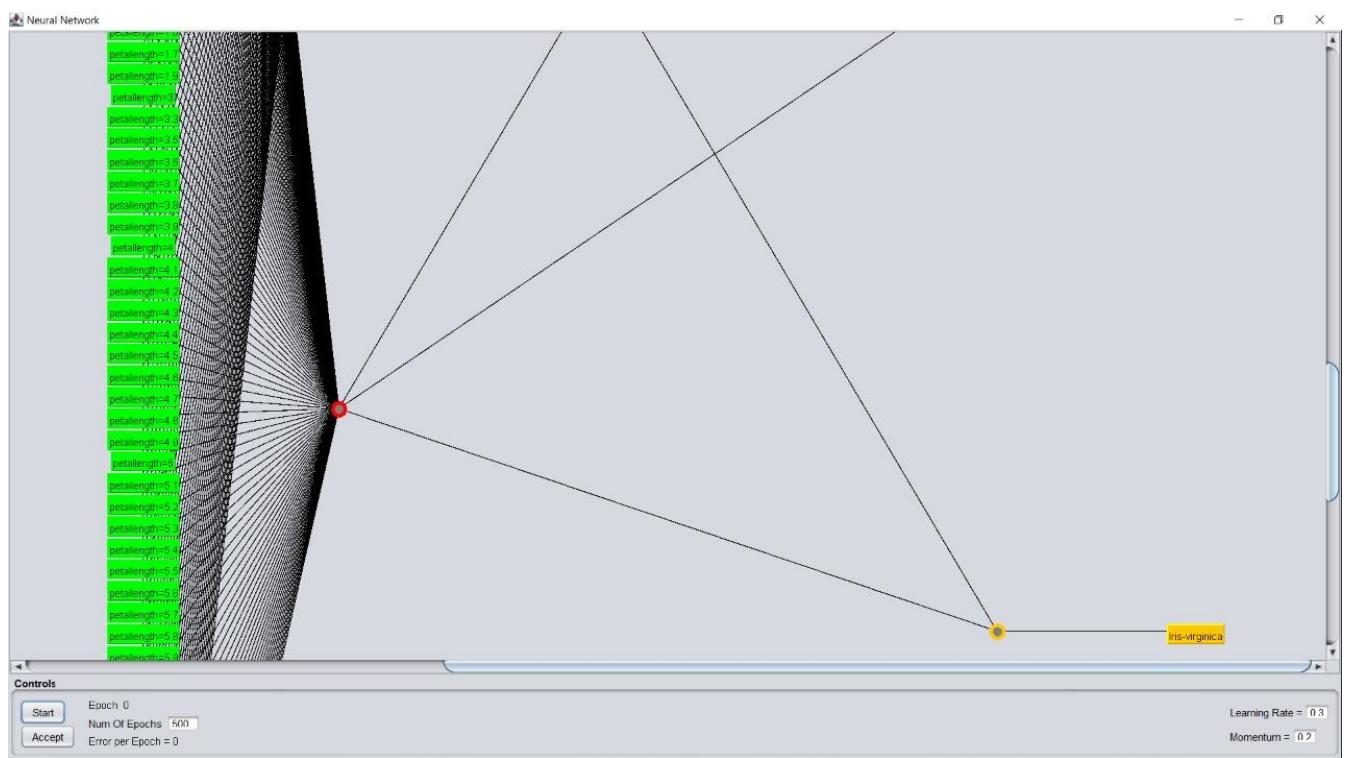
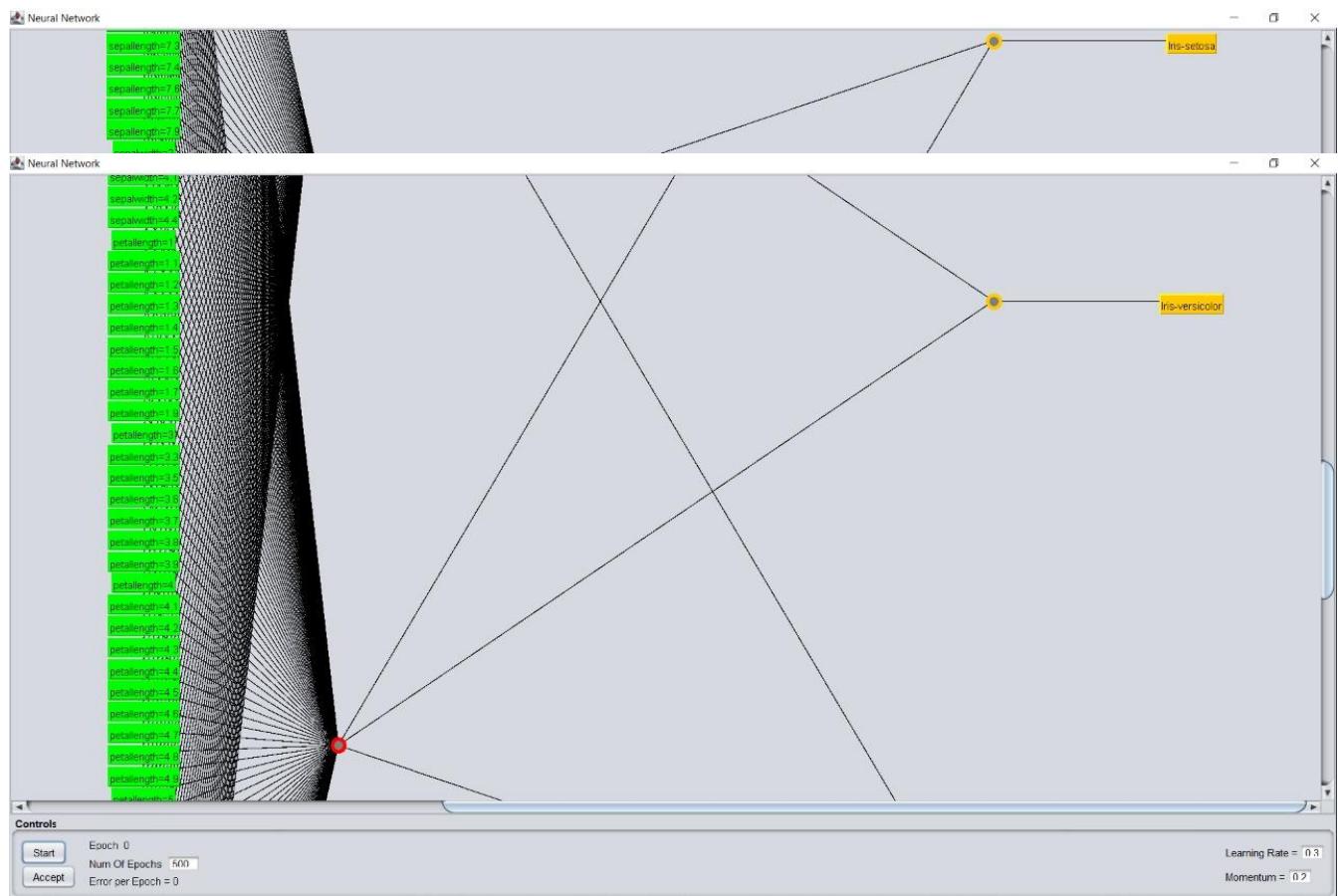
Goto classify ->functions->multilayer preceptron

Click on properties tab and make GUI as true, hidden layers as 2

Click on start and accept and observe the output.

Correlation coefficient	0.7048
Mean absolute error	0.3038
Root mean squared error	0.3734
Relative absolute error	60.7581 %
Root relative squared error	74.6824 %
Total Number of Instances	4





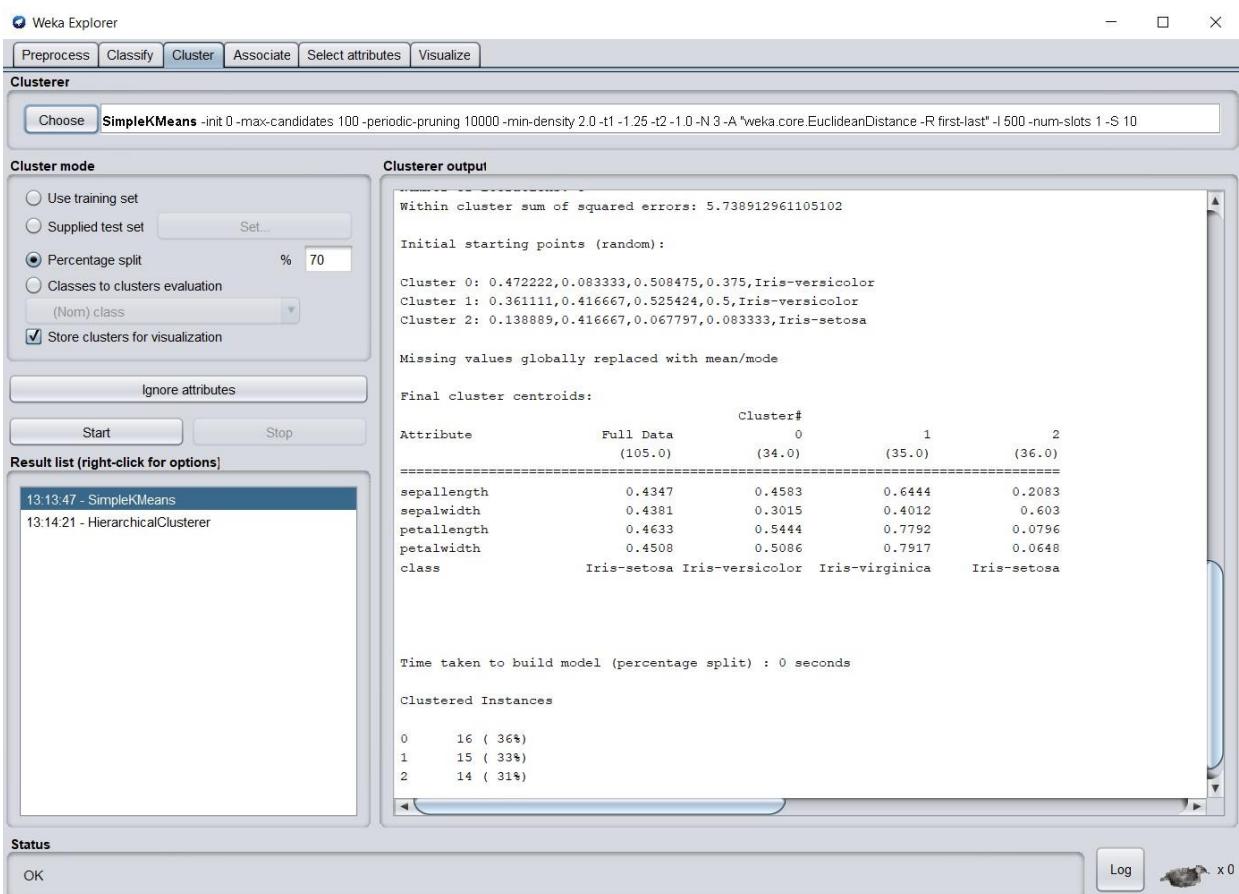
## 4.CLUSTERING

### 4.1 SimpleKMeans

Apply weka.clusterers.SimpleKMeans

Set NumCluster as 3.

Cluster data using the k means algorithm. Can use either the Euclidean distance



(default) or the Manhattan distance.

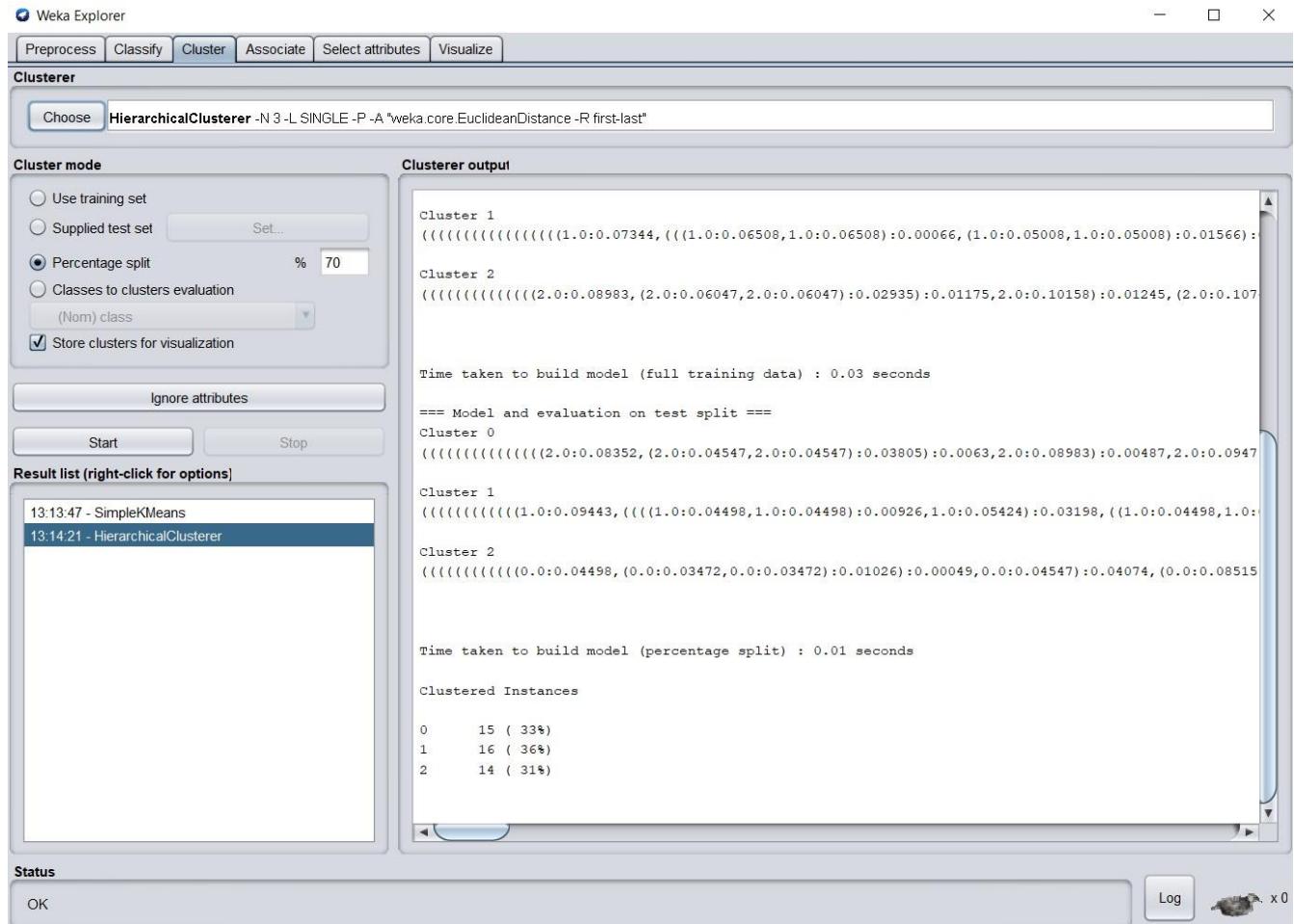
### 4.2 Hierarchical Clustering

Apply weka.clusterers.HierarchicalClusterer

Hierarchical clustering class.

Implements a number of classic agglomerative (i.e. bottom up) hierarchical clustering methods

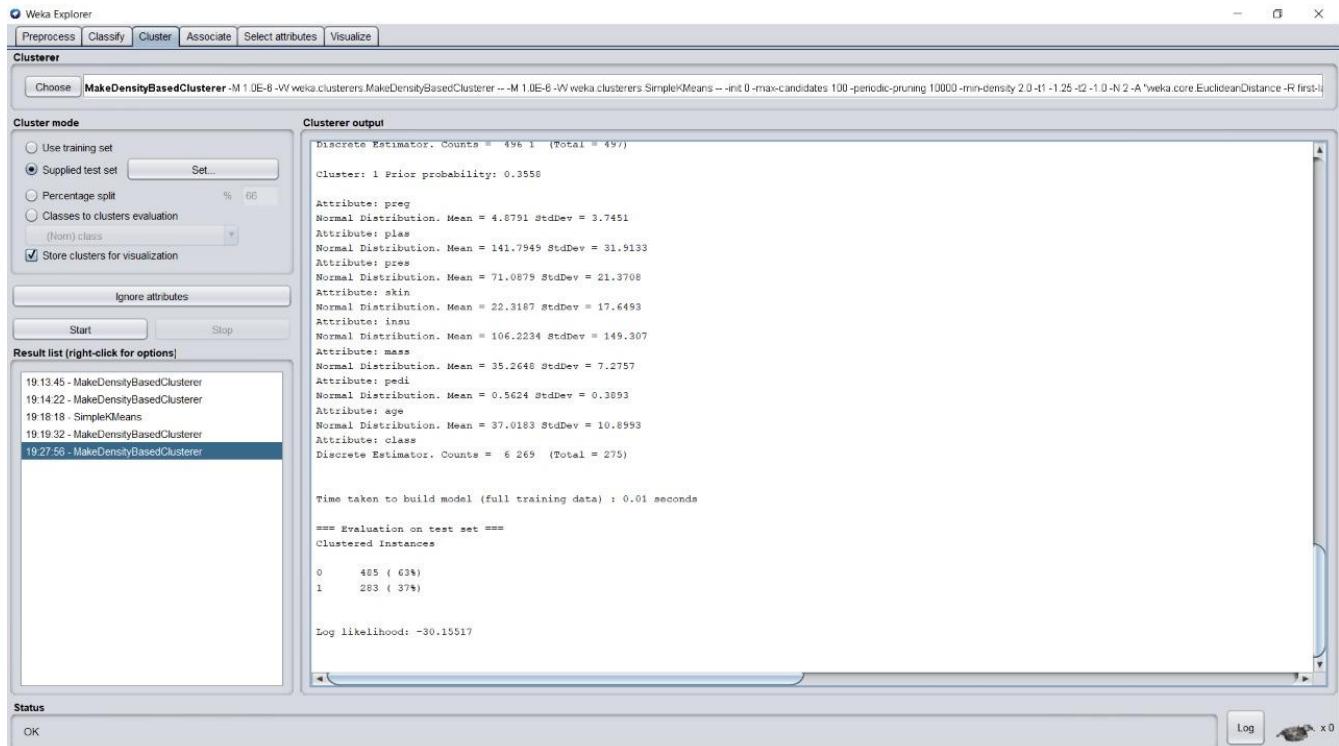
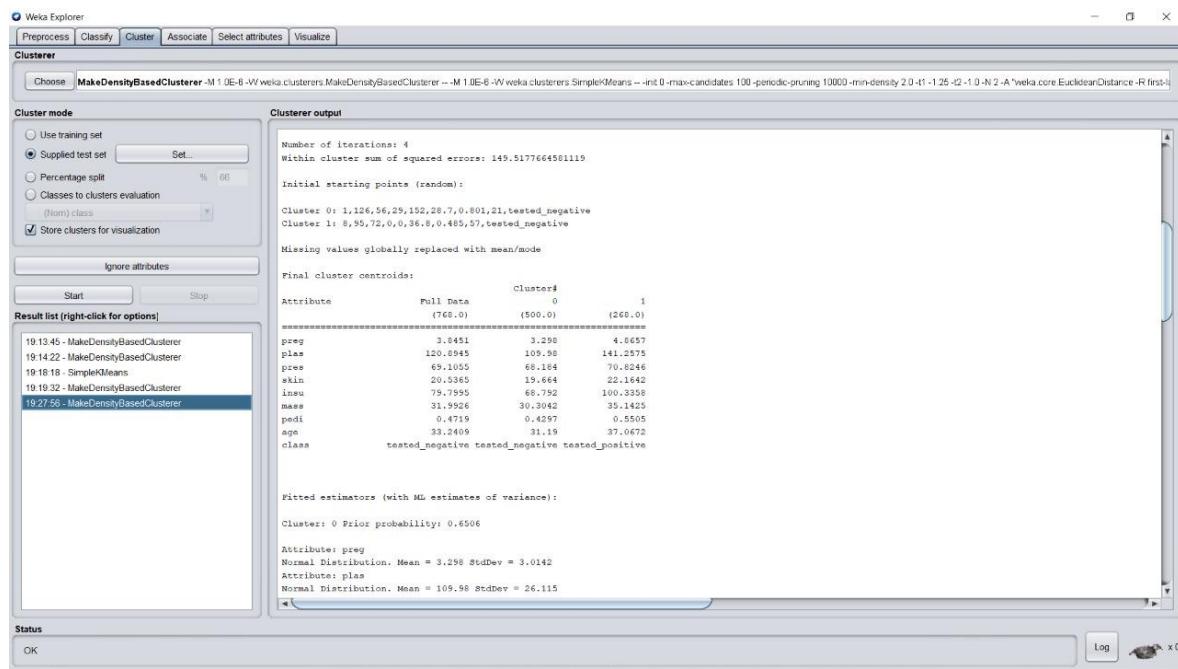
Set linkType as SINGLE.



### 4.3 MakeDensityBasedClusterer

Apply weka.clusterers.MakeDensityBasedClusterer

Class for wrapping a Clusterer to make it return a distribution and density. Fits normal distributions and discrete distributions within each cluster produced by the wrapped clusterer. Supports the NumberOfClustersRequestable interface only if the wrapped



Clusterer does.

## 5.Compare the Performance of Algorithms in Experiment environment

Weka provides a different tool specifically designed for comparing algorithms called the Weka Experiment Environment. The Weka Experiment Environment allows to design and execute controlled experiments with machine learning algorithms and then analyze the results.

Experiment Environment to compare the performance of machine learning algorithms.

1. Open the Weka Chooser GUI.
2. Click the Experimenter button to open the Weka Experiment Environment.
3. Click the New button.
4. Click the Add new... button in the Datasets pane and select iris.arff.
5. Click the Add new... button in the Algorithms panel and add trees.J48, bayes.NaiveBayes, functions.LibSVM and lazy.IBk
6. Click the Run tab and click the Start button.
7. Click the Analyse tab and click the Experiment button and then the Perform test button.
8. Save output.

The screenshot shows the 'Test output' window of the Weka Experiment Environment. It displays the configuration of the PairedCorrectedTTester and the results for the iris dataset.

**Test output**

```
Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-
Analysing: Percent_correct
Datasets: 1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 20/02/2021, 11:13
```

Dataset	(1) trees.J4   (2) bayes (3) lazy.
iris	(100) 94.73   95.53 95.40 (v/ /*)   (0/1/0) (0/1/0)

**Key:**

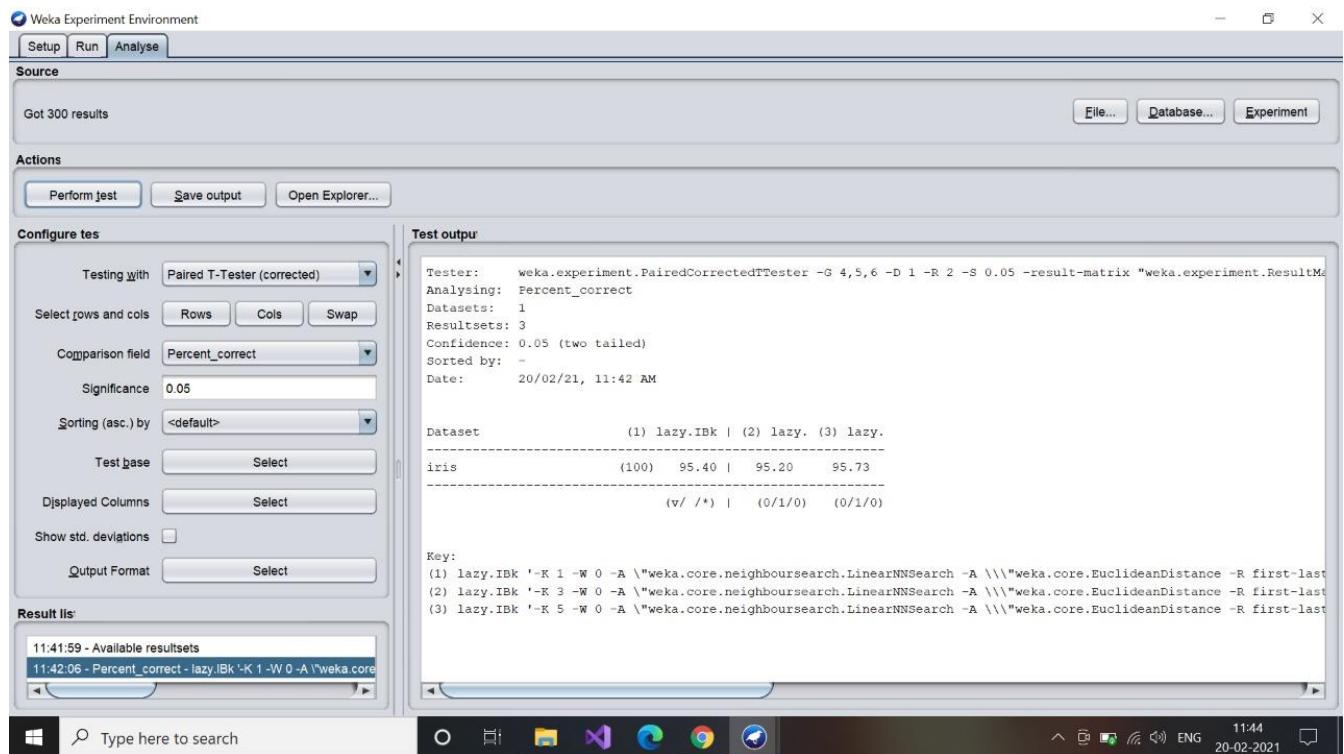
```
(1) trees.J48 '-C 0.25 -M 2' -2177331683936444444
(2) bayes.NaiveBayes '' 5995231201785697655
(3) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core
```

## 6.Tune Algorithm Parameters

To get the most out of a machine learning algorithm you must tune the parameters of the method to your problem. You cannot know how to best do this beforehand, therefore you must try out lots of different parameters. The Weka Experiment Environment allows you to design controlled experiments to compare the results of different algorithm parameters and whether the differences are statistically significant.

This experiment compares the performance of k-Nearest Neighbors algorithm by varying its parameter value.

1. Open the Weka Chooser GUI.
2. Click the Experimenter button to open the Weka Experiment Environment
3. Click the New button.
4. Click the Add new... button in the Datasets pane and select iris.arff.
5. Click the Add new... button in the Algorithms pane and add 3 copies of the IBk algorithm.
6. Click each IBk algorithm in the list and click the Edit selected... button and change KNN to 1, 3, 5 for each of the 3 different algorithms.
7. Click the Run tab and click the Start button.
8. Click the Analyse tab and click the Experiment button and then the Perform test button.
9. Save output.



## 1. Five Number Summary

jupyter DMlab Last Checkpoint: 4 minutes ago (autosaved)  Logout

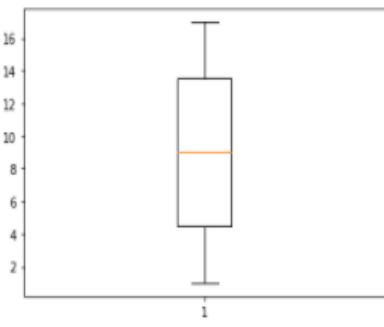
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]:

```
import matplotlib.pyplot as plt
from numpy import percentile
import numpy as np
```

In [2]:

```
dataip = [1,2,15,3,6,17,8,16,8,3,10,12,16,12,9]
data=np.array(dataip)
plt.boxplot(data)
plt.show()
```



In [3]:

```
# calculate quartiles
quartiles = percentile(data, [25, 50, 75])
# calculate min/max
data_min, data_max = data.min(), data.max()
# print 5-number summary
print('Min: %.3f' % data_min)
print('Q1: %.3f' % quartiles[0])
print('Median: %.3f' % quartiles[1])
print('Q3: %.3f' % quartiles[2])
print('Max: %.3f' % data_max)
```

Min: 1.000  
Q1: 4.500  
Median: 9.000  
Q3: 13.500  
Max: 17.000

## 2.Normalisation

jupyter DMlab Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

### Normalization

```
In [1]: import pandas as pd
```

```
In [17]: dataset=pd.read_csv("C:\\Users\\LALITHA\\Downloads\\Iris.csv")
dataset.drop(['Species'],axis=1,inplace=True)
dataset.head()
```

```
Out[17]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

```
In [18]: print("Min-Max Normalization")
new_min,new_max=0,1
output=(dataset-dataset.min())*((new_max-new_min)/(dataset.max()-dataset.min())))
print(output.minmax)
```

```
Min-Max Normalization
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	0.000000	0.222222	0.625000	0.067797	0.041667
1	0.006711	0.166667	0.416667	0.067797	0.041667
2	0.013323	0.111111	0.333333	0.067797	0.041667
3	0.020034	0.066667	0.458333	0.067797	0.041667
4	0.026846	0.194444	0.666667	0.067797	0.041667
5	0.033557	0.305555	0.791667	0.118644	0.125000
6	0.040268	0.083333	0.583333	0.067797	0.083333
7	0.046979	0.144444	0.333333	0.067797	0.041667
8	0.053690	0.027778	0.375000	0.067797	0.041667
9	0.060401	0.166667	0.458333	0.067797	0.000000
10	0.067114	0.305555	0.708333	0.084746	0.041667
11	0.073826	0.198889	0.583333	0.101695	0.041667
12	0.080537	0.188889	0.216667	0.067797	0.041667
13	0.087249	0.200000	0.666667	0.016949	0.000000
14	0.093960	0.416667	0.833333	0.033898	0.041667
15	0.100671	0.388889	1.000000	0.084746	0.125000
16	0.107383	0.305555	0.791667	0.050847	0.125000
17	0.114094	0.222222	0.625000	0.067797	0.083333
18	0.120805	0.466667	0.740000	0.118644	0.083333

```
In [20]: print("z-score normalization")
output_zscore=(dataset-dataset.mean())/(dataset.std())
print(output_zscore)
```

```
z-score normalization
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-1.714795	-0.297674	1.028611	-1.336794	-1.308593
1	-1.691780	-1.139200	-0.124540	-1.336794	-1.308593
2	-1.668762	-1.380727	0.336720	-1.393470	-1.308593
3	-1.645745	-1.501490	0.160690	-1.280118	-1.308593
4	-1.622728	-1.818437	1.259242	-1.336794	-1.308593
5	-1.599710	-0.535384	1.112337	-1.280118	-1.308593
6	-1.576693	-1.501490	0.797981	-1.336794	-1.177559
7	-1.553675	-1.018437	0.797981	-1.280118	1.308593
8	-1.530658	-1.743017	-0.355171	-1.336794	1.308593
9	-1.507641	-1.210300	0.806900	-1.280118	1.308593
10	-1.484623	-0.535384	1.489872	-1.280118	1.308593
11	-1.461600	-1.259964	0.797981	-1.232442	1.308593
12	-1.438588	-1.259964	-0.124540	-1.336794	-1.439627
13	-1.415574	-1.865794	-0.816000	-1.136822	-1.439627
14	-1.392554	-0.413311	1.181763	-1.350049	-1.439627
15	-1.369536	-0.173094	3.104284	-1.280118	-1.046525
16	-1.346519	-0.535384	1.951133	-1.393470	-1.046525
17	-1.323508	-0.897674	1.028611	-1.336794	-1.177559
18	-1.300484	-0.173094	1.794000	-1.146767	-1.177559

```
In [22]: print("Normalization by decimal scaling")
output_decimalScaling=(pow(10,int(str(abs(dataset['SepalWidthCm']).max()))))
print(output_decimalScaling)
```

```
Normalization by decimal scaling
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	0.001	0.0001	0.0001	0.0014	0.0002
1	0.002	0.0004	0.0020	0.0014	0.0002
2	0.003	0.0047	0.0032	0.0013	0.0002
3	0.004	0.0046	0.0031	0.0015	0.0002
4	0.005	0.0050	0.0036	0.0014	0.0002
5	0.006	0.0054	0.0039	0.0017	0.0004
6	0.007	0.0046	0.0034	0.0014	0.0003
7	0.008	0.0050	0.0034	0.0015	0.0002
8	0.009	0.0044	0.0029	0.0014	0.0002
9	0.010	0.0042	0.0031	0.0015	0.0001
10	0.011	0.0054	0.0037	0.0015	0.0002
11	0.012	0.0040	0.0034	0.0016	0.0002
12	0.013	0.0048	0.0030	0.0014	0.0001
13	0.014	0.0043	0.0039	0.0011	0.0001
14	0.015	0.0050	0.0040	0.0010	0.0002
15	0.016	0.0057	0.0044	0.0015	0.0004
16	0.017	0.0054	0.0039	0.0013	0.0003
17	0.018	0.0051	0.0035	0.0014	0.0003
18	0.019	0.0047	0.0038	0.0017	0.0003

### 3.Binning

```
In [1]: M import numpy as np

In [2]: M price = np.array([5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215])
dataset=np.sort(price)
nob=3

print(dataset.size)

12

In [3]: M binsize=(int)(dataset.size/nob)
print(binsize)
print('.....Equal height binning...')
print(dataset[0:binsize],dataset[binsize:2*binsize],dataset[2*binsize:3*binsize])

4
.....Equal height binning...
[ 5 10 11 13] [15 35 50 55] [ 72 92 204 215]

In [5]: M print('.....Equal width binning...')

width=int((dataset.max()-dataset.min())/nob)

bin1=[]
bin2=[]
bin3=[]

for i in range(len(dataset)):
    if dataset[i]>=dataset.min() and dataset[i]<=(width+dataset.min()):
        bin1.append(dataset[i])
    elif dataset[i]>(width+dataset.min()) and dataset[i]<=2*(width+dataset.min()):
        bin2.append(dataset[i])
    else:
        bin3.append(dataset[i])

print(bin1)
print(bin2)
print(bin3)

.....Equal width binning...
[5, 10, 11, 13, 15, 35, 50, 55, 72]
[92]
[204, 215]
```

```
In [ ]: M
```

## 4.Distance Metrics

The screenshot shows a Jupyter Notebook interface with two code cells. The top cell, titled "Distance Metrics", contains Python code for implementing various distance metrics. The bottom cell, titled "In [2]", contains code to test these metrics with specific data points.

```
In [1]: import math
from decimal import Decimal
"""Five similarity measures function """
def euclidean_distance(x,y):

    """return euclidean distance between two lists """
    return math.sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))

def manhattan_distance(x,y):

    """return manhattan distance between two lists """
    return sum(abs(a-b) for a,b in zip(x,y))

def minkowski_distance(x,y,p_value):

    """return minkowski distance between two lists """
    return nth_root(sum(pow(abs(a-b),p_value) for a,b in zip(x, y)), p_value)

def nth_root(value, n_root):

    """returns the n_root of an value """
    root_value = 1/float(n_root)
    return round (Decimal(value) ** Decimal(root_value),3)

def cosine_similarity(x,y):

    """return cosine similarity between two lists """
    numerator = sum(a*b for a,b in zip(x,y))
    denominator =square_rooted(x)*square_rooted(y)
    return round(numerator/float(denominator),3)

def square_rooted(x):
    """return 3 rounded square rooted value """
    return round(math.sqrt(sum([a*a for a in x])),3)

def jaccard_similarity(x,y):

    """returns the jaccard similarity between two lists """
    intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
    union_cardinality = len(set.union(*[set(x), set(y)]))
    return intersection_cardinality/float(union_cardinality)
```

```
In [2]: x=[22, 1, 42, 10]
y=[20, 0, 36, 8]

print(euclidean_distance(x,y))
print(manhattan_distance(x,y))
print(minkowski_distance(x,y,3))
print(cosine_similarity(x,y))
print(jaccard_similarity(x,y))

6.708203932499369
11
6.153
0.999
0.0
```

## 6.Apriori

### Apriori

```
In [3]: M import itertools as it
```

```
In [4]: M data=[[1,2,3],[2,3,5],[1,2,4],[3,4]]
l=[]
u=[]
for x in data:
    print(x)
c1=[1,2,3,4,5]
for i in range(1,len(c1)):
    y=it.combinations(c1,i)
    for z in y:
        count=0
        for x in data:
            if (set(z).issubset(x)):
                count=count+1
            if(count>=2):
                l.append(z)
                break
print(l)
```

```
[1, 2, 3]
[2, 3, 5]
[1, 2, 4]
[3, 4]
[(1,), (2,), (3,), (4,), (1, 2), (2, 3)]
```

```
In [5]: M l1=[3]
r1=[1,2]
count1=0.0
count2=0.0
for x in data:
    if(set(l1).issubset(x) and set(r1).issubset(x)):
        count1=count1+1
print(count1)
for y in data:
    if(set(l1).issubset(y)):
        count2=count2+1
print(count2)
conf=(count1/count2)*100
print(conf)
```

```
1.0
3.0
33.33333333333333
```

```
In [ ]: M
```

## 7. Classification

jupyter DMlab Last Checkpoint: 3 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [6]:

```
from sklearn import datasets
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

In [7]:

```
iris = datasets.load_iris()
x = iris.data
y = iris.target
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=.5)
```

Create Decision Tree classifier object

In [8]:

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
```

Train Decision Tree Classifier

In [9]:

```
clf = clf.fit(x_train,y_train)
```

Predict the response for test dataset

In [10]:

```
y_pred = clf.predict(x_test)
```

Model Accuracy, how often is the classifier correct?

In [11]:

```
print("Accuracy:",accuracy_score(y_test, y_pred))
```

Accuracy: 0.9733333333333334

In [12]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	22
1	1.00	0.92	0.96	26
2	0.93	1.00	0.96	27

	accuracy	macro avg	weighted avg	
accuracy	0.97	0.97	0.97	75
macro avg	0.98	0.97	0.97	75
weighted avg	0.98	0.97	0.97	75

## 8.More Classifiers

The screenshot shows a Jupyter Notebook interface with the title "More Classifiers". The notebook has three cells:

- In [12]:** Python code imports various machine learning models and metrics from the sklearn library.
- In [16]:** Python code reads the "iris.csv" dataset and prints its first five rows.
- Out[16]:** The output shows the first five rows of the iris dataset:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

**Split-out validation dataset**

**In [17]:** Python code splits the dataset into training and validation sets using `train_test_split`.

**Test options and evaluation metric**

**In [9]:** Python code sets the random seed and scoring metric to "accuracy".

jupyter DMLab Last Checkpoint: 4 hours ago (autosaved)

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

### Spot Check Algorithms

```
In [7]: models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

### Evaluate each model in turn

```
In [18]: results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, scoring=scoring1, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

LR: 0.966667 (0.040825)
LDA: 0.975000 (0.031818)
KNN: 0.983333 (0.031333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

### Compare Algorithms

```
In [19]: fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

### Make predictions on validation dataset

```
In [20]: knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
precision    recall  f1-score   support
      0.0      1.00      1.00       7
      1.0      0.85      0.92      12
      2.0      0.90      0.82      11
accuracy                           0.90      30
macro avg      0.92      0.91      0.91      30
weighted avg     0.90      0.90      0.90      30
```

In [ ]:

## 9.Prediction

jupyter DMlab Last Checkpoint: 4 hours ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3



### Prediction

```
In [23]: M import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

```
In [24]: M dataset=pandas.read_csv("C:\\\\Users\\\\LALITHA\\\\Anaconda3\\\\lib\\\\site-packages\\\\sklearn\\\\datasets\\\\data\\\\iris.csv")
dataset.head()
```

```
Out[24]:
```

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

### For test

```
In [25]: M a=np.array([5.1,3.5,1.4,0.2]).reshape(1,-1)
b=np.array([7,3.2,4.7,1.4]).reshape(1,-1)
c=np.array([6.3,3.3,6,2.5]).reshape(1,-1)
d=np.array([6.3,2.3,6,1.5]).reshape(1,-1)
```

### Split-out validation dataset

```
In [26]: M array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_stat
```

jupyter DMlab Last Checkpoint: 4 hours ago (autosaved) Logout

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

Split-out validation dataset

```
In [26]: M array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_state=seed)
```

Test options and evaluation metric

```
In [27]: M seed = 7
scoring = 'accuracy'
```

Spot Check Algorithms

```
In [28]: M models = []
models.append('LR', LogisticRegression())
models.append('LDA', LinearDiscriminantAnalysis())
models.append('KNN', KNeighborsClassifier())
models.append('CART', DecisionTreeClassifier())
models.append('NB', GaussianNB())
models.append('SVM', SVC())
```

Evaluate each model in turn

```
In [29]: M results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, scoring=scoring, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

Make predictions on validation dataset

```
In [30]: M knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)

print("prediction of new instances")
predictionsa = knn.predict(a)
```



### Make predictions on validation dataset

```
In [30]: knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)

print("prediction of new instances")
predictionsa = knn.predict(a)
print(predictionsa)

predictionsb = knn.predict(b)
print(predictionsb)

predictionsc = knn.predict(c)
print(predictionsc)

predictionsd = knn.predict(d)
print(predictionsd)
```

```
prediction of new instances
[0.]
[1.]
[2.]
[2.]
```

```
In [31]: predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
      precision    recall  f1-score   support
          0.0       1.00     1.00      7
          1.0       0.85     0.92     0.88     12
          2.0       0.90     0.82     0.86     11
  accuracy                           0.90      30
 macro avg       0.92     0.91     0.91      30
 weighted avg       0.90     0.90     0.90      30
```

```
In [32]: svc = SVC()
svc.fit(X_train, Y_train)
predictions = svc.predict(a)
print(predictions)
```

```
[0.]
```

```
In [ ]:
```

## 10. Regression

jupyter DMlab Last Checkpoint: 4 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [33]: M from pandas import read_csv
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression

In [34]: M # Load dataset
import numpy as np
dataframe = read_csv("C:\\\\Users\\\\LALITHA\\\\Downloads\\\\pima-indians-diabetes.csv")

In [35]: M array = dataframe.values
X = array[:,0:8]
Y = array[:,8]

x_train, x_test, y_train, y_test = model_selection.train_test_split(X, Y, test_size=.5)
a=np.array([1,85,66,29,0,26.6,0.351,31]).reshape(1,-1)

# Create Linear regression object

"""
regr = LinearRegression()
# Train the model using the training sets
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)

print(regr.predict(a))
# Model Accuracy, how often is the classifier correct?
print("mean_absolute_error:",mean_absolute_error(y_test, y_pred))

print("mean_squared_error:",mean_squared_error(y_test, y_pred))
print("r2_score:",r2_score(y_test, y_pred))

"""

# Train the model using the training sets
regr = LogisticRegression()
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)

print(regr.predict(a))
# Model Accuracy, how often is the classifier correct?
print("mean_absolute_error:",mean_absolute_error(y_test, y_pred))

print("mean_squared_error:",mean_squared_error(y_test, y_pred))
print("r2_score:",r2_score(y_test, y_pred))

[0.]
mean_absolute_error: 0.25
mean_squared_error: 0.25
r2_score: -0.002484216700266
```

## 11. Finalize and Save your model

jupyter DMLab Last Checkpoint: 4 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Finalize and Save your model

```
In [36]: # Save Model Using Pickle
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pickle
import numpy as np

dataframe = read_csv("iris.csv")
array = dataframe.values

X = array[:,0:4]
Y = array[:,4]

test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
# Fit the model on 33%
print(X_test)
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later...

# Load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, Y_test)
print(result)

a=np.array([5.1,3.5,1.4,0.2]).reshape(1,-1)
p=loaded_model.predict(a)
print(p)
```

```
[[150 5.9 3.0 5.1]
 [85 5.4 3.0 4.5]
 [45 5.0 3.5 1.3]
 [67 5.6 3.0 4.5]
 [107 4.9 2.5 4.5]
 [42 4.5 2.3 1.3]
 [53 6.9 3.1 4.9]
 [95 5.6 2.7 4.2]
 [12 4.8 3.4 1.6]
 [52 6.4 3.2 4.5]
 [78 6.7 3.0 5.0]
 [86 6.0 3.4 4.5]
 [33 5.2 4.1 1.5]
 [110 7.2 3.6 6.1]
 [29 5.2 3.4 1.4]
 [71 5.9 3.2 4.8]
 [109 6.7 2.5 5.8]
 [138 6.4 3.1 5.5]
 [47 5.1 3.8 1.6]
 [38 4.9 3.1 1.5]
 [82 5.8 2.7 2.9]
```

In [ ]:

## 12. Clustering

Jupyter DMlab Last Checkpoint: 4 hours ago (unsaved changes)  Logon

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [37]:

```
from sklearn import cluster
import pandas as pd

dataset=pd.read_csv("C:\\\\Users\\\\LALITHA\\\\Downloads\\\\pima-indians-diabetes.csv")
km = cluster.KMeans(n_clusters=3).fit(dataset)
print(km.cluster_centers_)
print(km)

from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering().fit(dataset)
print(ac.labels_)
print(type(ac.labels_))

ac.labels_.tofile("D:/output.csv")

from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=3, min_samples=3).fit(dataset)
print(dbs)
print(dbs.components_)

[[3.97773279e+00 1.13939271e+02 6.77631579e+01 1.49574899e+01
 1.44291498e+01 3.07997976e+01 4.31536437e-01 3.37267206e+01
 2.97570850e-01]
 [4.02631579e+00 1.58447368e+02 7.20000000e+01 3.22631579e+01
 4.41289474e+02 3.51078947e+01 5.69210526e-01 3.47631579e+01
 5.78947368e-01]
 [3.52765957e+00 1.29327660e+02 7.14468085e+01 3.03063830e+01
 1.59102128e+02 3.39893617e+01 5.40276596e-01 3.19021277e+01
 4.17021277e-01]]
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
[1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 0 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1
 0 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1
 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 0 0 1 0
 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1
 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0
 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1
 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 0 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0]
```

<class 'numpy.ndarray'>

DBSCAN(algorithm='auto', eps=3, leaf\_size=30, metric='euclidean',
 metric\_params=None, min\_samples=3, n\_jobs=None, p=None)

[]

A case study

On

## **Prediction of Structure in Ionosphere**

By

**Sanikommu Narendra (2451-17-733-062)**

**Bobbala Sanjay Yadav (2451-17-733-074)**

**Papaganti Prasanna Kuma (2451-17-733-096)**



Department of Computer Science and Engineering

**M.V.S.R. ENGINEERING COLLEGE**

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510

2020-21.

## Objective

In this case study, we are predicting whether or not there is structure in the ionosphere or not

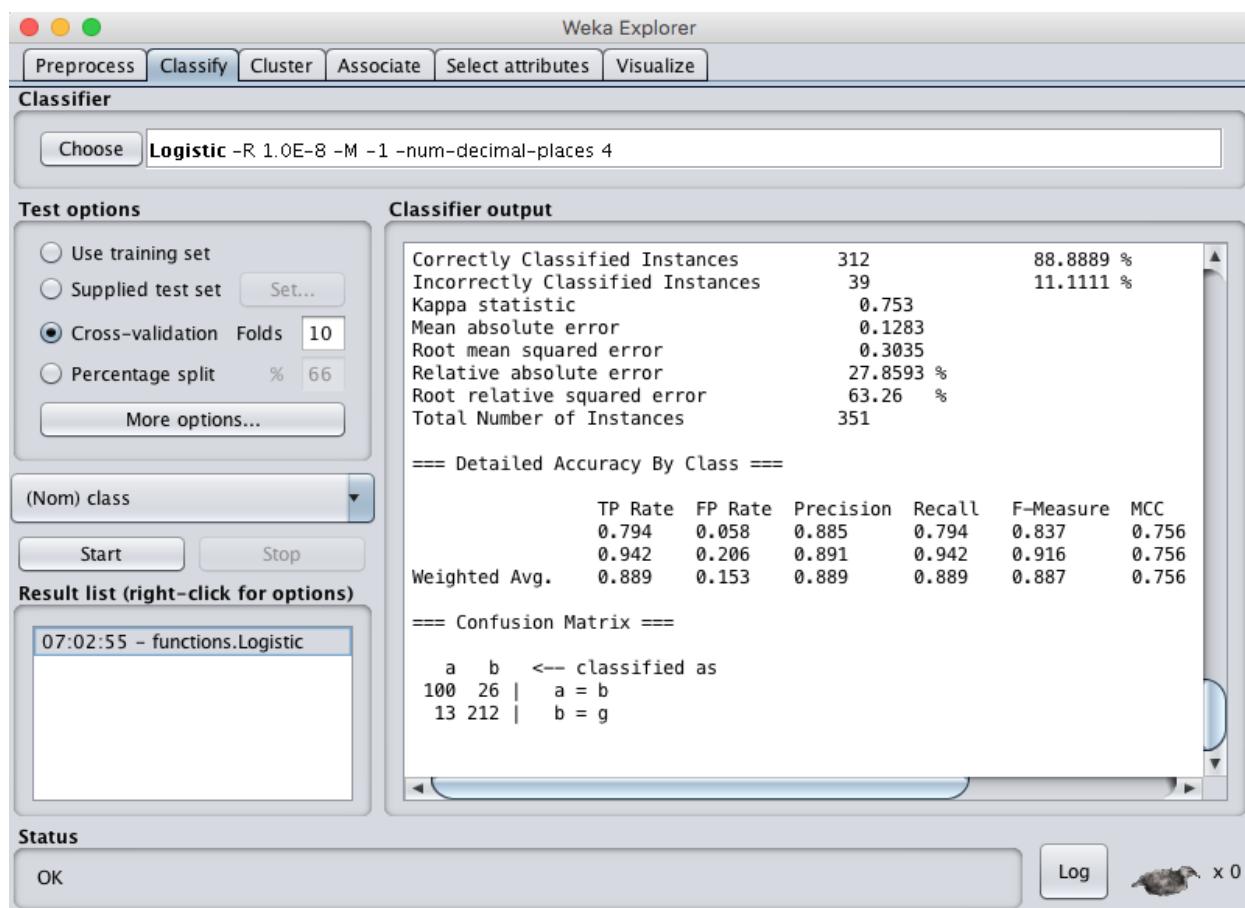
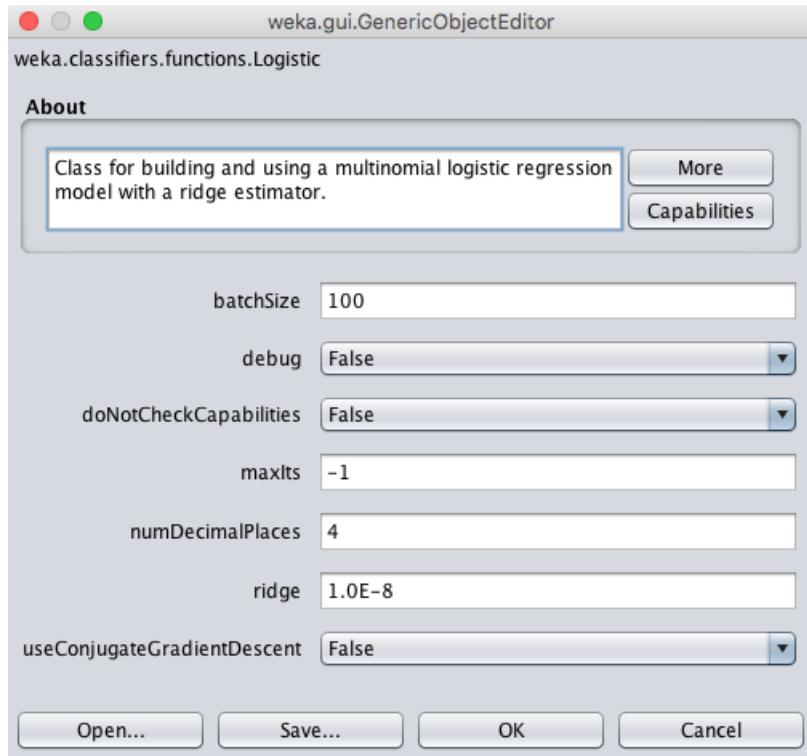
## Data Set Information

<b>Data Set Characteristics:</b>	Multivariate	<b>Number of Instances:</b>	351
<b>Attribute Characteristics:</b>	Integer, Real	<b>Number of Attributes:</b>	34
<b>Associated Tasks:</b>	Classification	<b>Missing Values?</b>	No

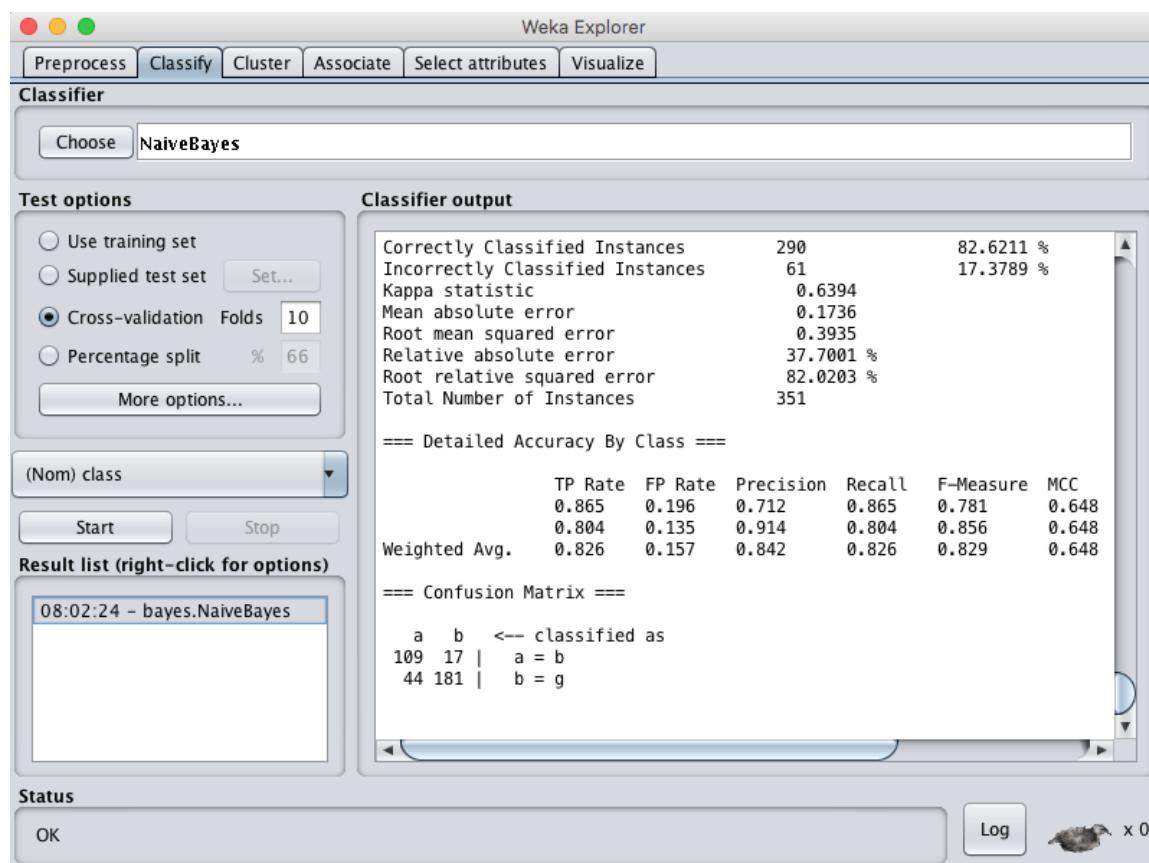
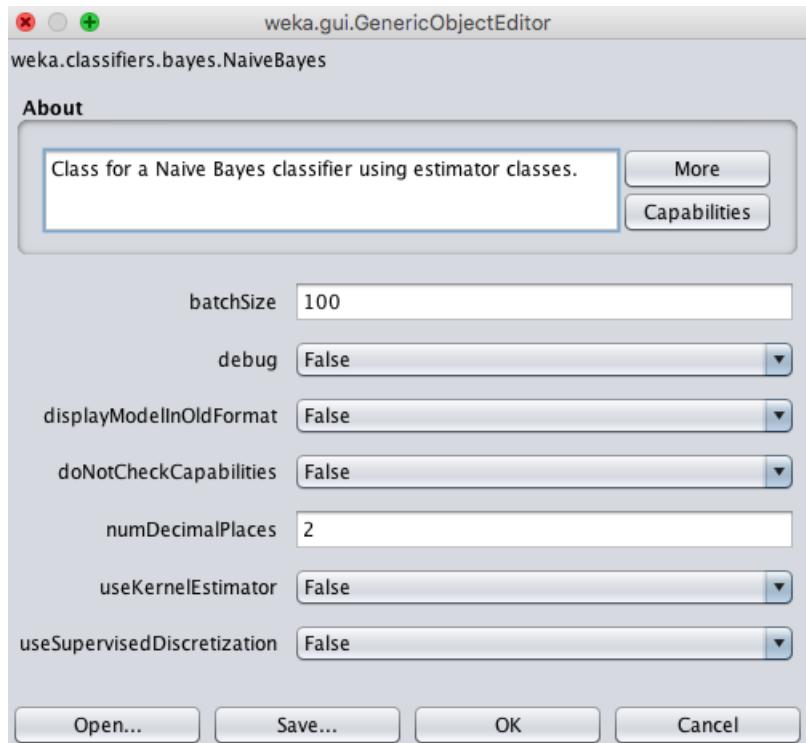
## Attribute Information:

- All 34 are continuous.
- The 35th attribute is either "good" or "bad".

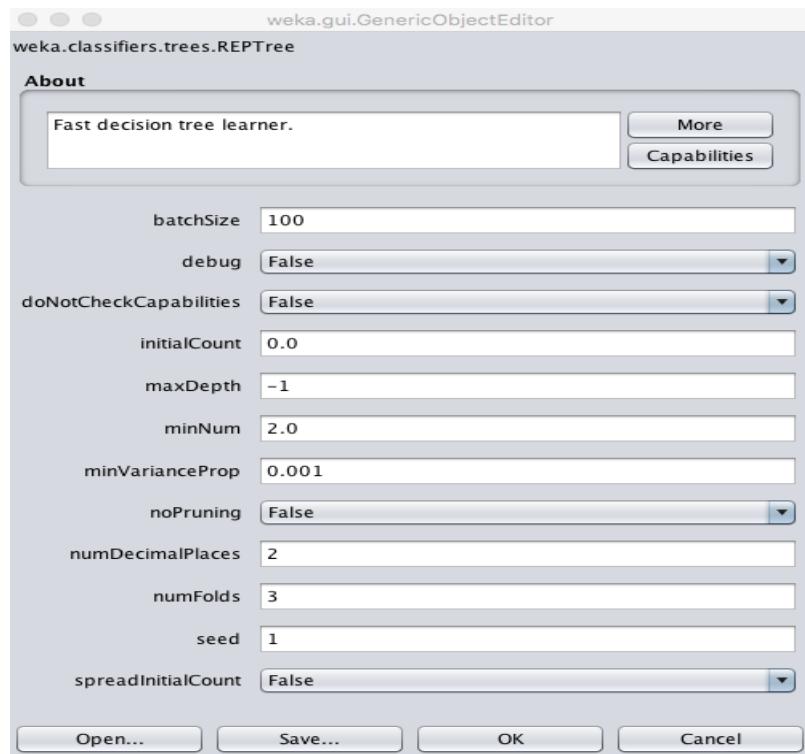
# Logistic Regression



## Naive Bayes



## Decision Tree



The screenshot shows the 'Weka Explorer' interface with the 'Classifier' tab selected. The 'Choose' button displays the command: **REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0**. The 'Test options' panel includes radio buttons for 'Use training set', 'Supplied test set', 'Cross-validation' (selected, Folds 10), and 'Percentage split' (66%), along with a 'More options...' button. The 'Classifier output' panel displays classification statistics and detailed accuracy by class. The 'Result list' panel shows a log entry: 08:04:15 - trees.REPTree. The 'Status' panel at the bottom has a 'Log' button.

**Classifier output:**

	Correctly Classified Instances	89.4587 %
Incorrectly Classified Instances	37	10.5413 %
Kappa statistic	0.7689	
Mean absolute error	0.158	
Root mean squared error	0.3001	
Relative absolute error	34.3084 %	
Root relative squared error	62.5544 %	
Total Number of Instances	351	

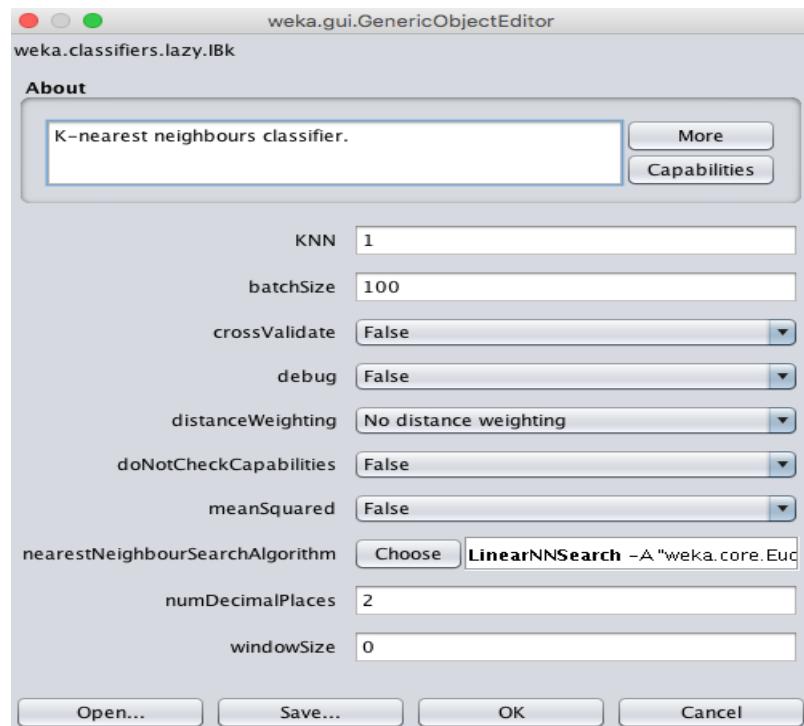
**==== Detailed Accuracy By Class ====**

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
0.833	0.071	0.868	0.833	0.850	0.769	
0.929	0.167	0.909	0.929	0.919	0.769	
Weighted Avg.	0.895	0.132	0.894	0.895	0.894	0.769

**==== Confusion Matrix ====**

		<-- classified as	
a	b	a = b	b = g
105	21		a = b
16	209		b = g

## k-Nearest Neighbors



The screenshot shows the Weka Explorer interface. The title bar reads 'Weka Explorer'. The 'Classifier' tab is selected, showing the command 'IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A {"weka.core.EuclideanDistance -R first-last}"' in the 'Choose' field.

**Test options**

- Use training set
- Supplied test set
- Cross-validation Folds 10
- Percentage split % 66

**Classifier output**

	Correctly Classified Instances	303	86.3248 %
Incorrectly Classified Instances	48	13.6752 %	
Kappa statistic	0.6841		
Mean absolute error	0.139		
Root mean squared error	0.3686		
Relative absolute error	30.1815 %		
Root relative squared error	76.8426 %		
Total Number of Instances	351		

== Detailed Accuracy By Class ==

(Nom) class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
0	0.675	0.031	0.924	0.675	0.780	0.702
1	0.969	0.325	0.842	0.969	0.901	0.702
Weighted Avg.	0.863	0.220	0.871	0.863	0.857	0.702

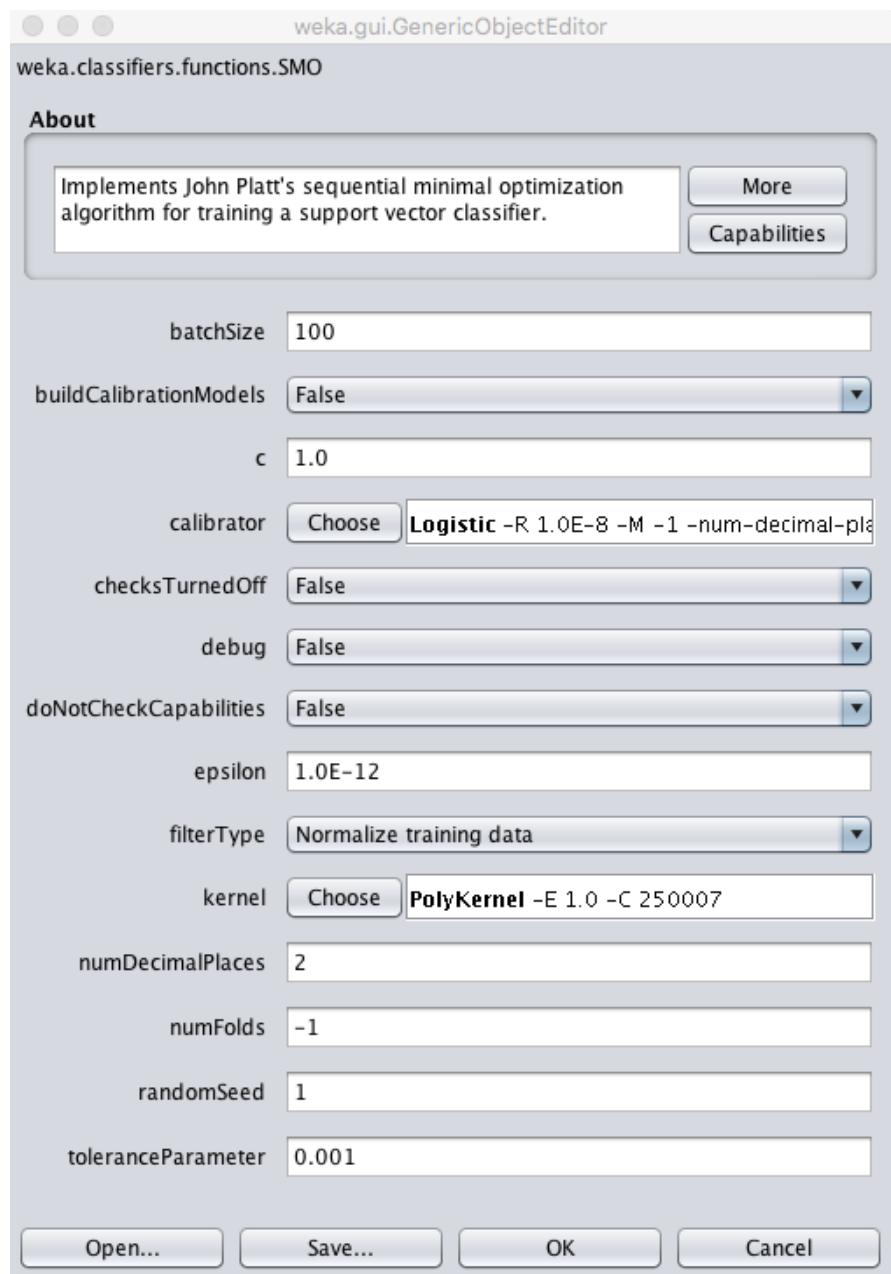
== Confusion Matrix ==

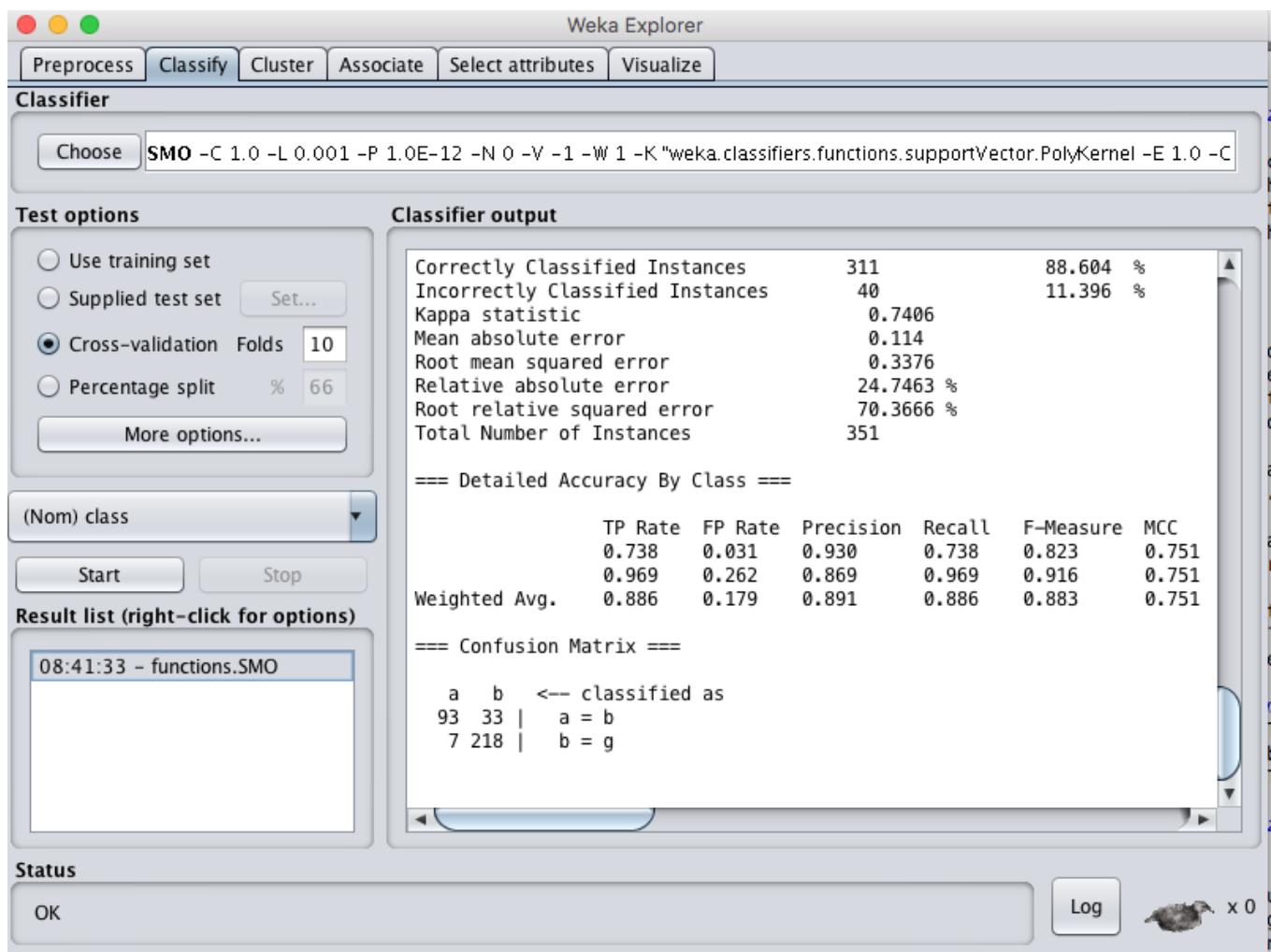
		a	b	<-- classified as
		85	41	a = b
		7	218	b = g

**Status**

OK  x 0

## Support Vector Machines





## Observations

Comparing all the algorithms SVM showed the least Relative absolute errors.

