DATA MINING

CDCSC16

Submitted to:
Vandana Bhatia

Submitted by:
Balraj Gahlot
2020UCD2152
CSDS – I
Semester - 5

Teachers Sign:

INDEX

Practical – 1

AIM OF EXPERIMENT:

To Create an Employee Table with the help of Data Mining Tool WEKA.

Description:

WEKA - an open source software provides tools for data pre-processing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems.

The default file type in WEKA is arff.

An **Arff** file contains two sections - header and data.

- The header describes the attribute types.
- The data section contains a comma separated list of data.

We need to create a Weather table with training data set which includes attributes like outlook, temperature, humidity, windy, play.

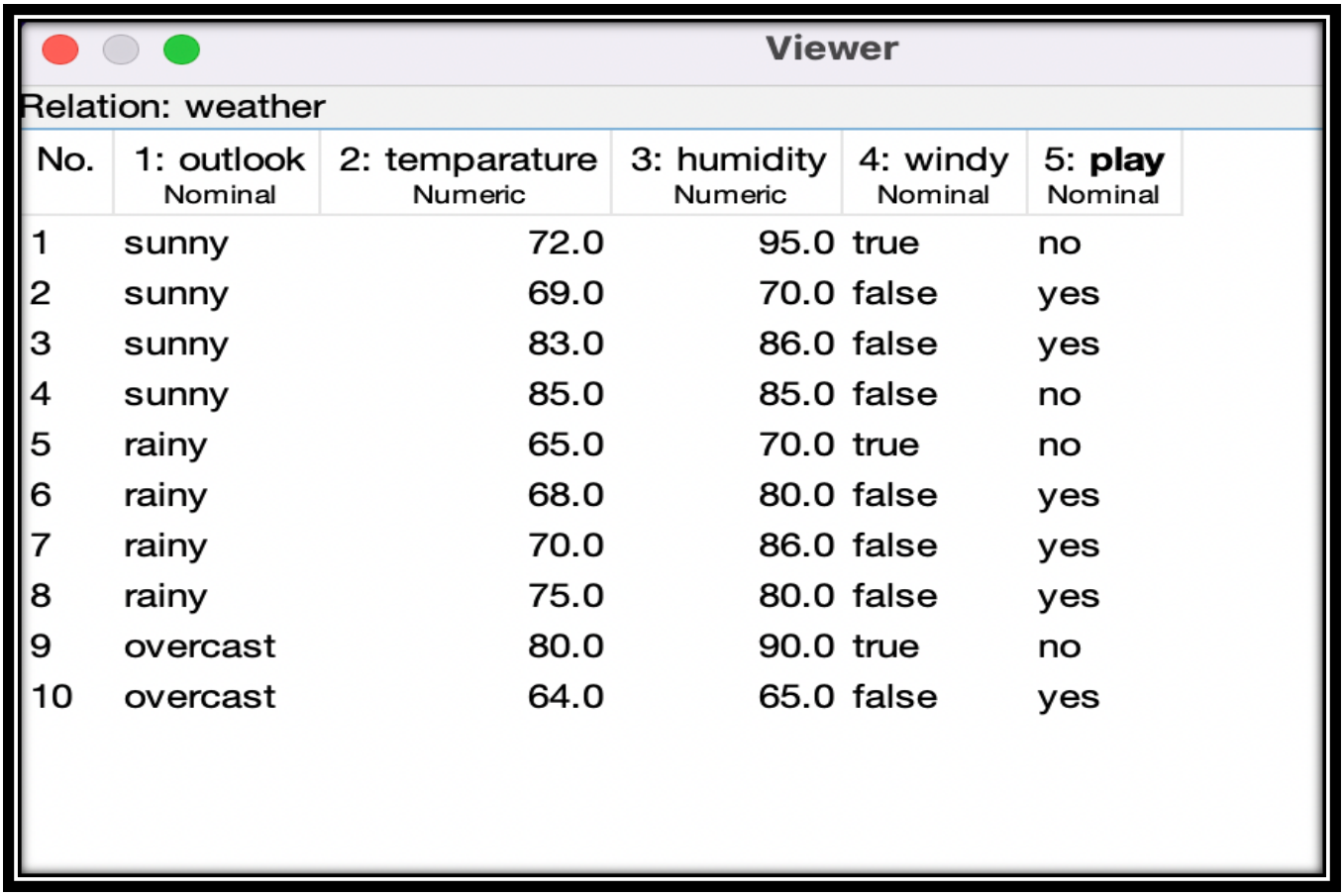
Procedure:

1. Open a text editor
2. Type the following training data set with the help of Notepad for Weather Table.

```
@relation weather

@attribute outlook {sunny,rainy,overcast}
@attribute temparature numeric
@attribute humidity numeric
@attribute windy {true,false}
@attribute play {yes,no}

@data
sunny,85.0,85.0,false,no
overcast,80.0,90.0,true,no
sunny,83.0,86.0,false,yes
rainy,70.0,86.0,false,yes
rainy,68.0,80.0,false,yes
rainy,65.0,70.0,true,no
overcast,64.0,65.0,false,yes
sunny,72.0,95.0,true,no
sunny,69.0,70.0,false,yes
rainy,75.0,80.0,false,yes
```

3. After that the file is saved with .arff file format
4. The file is saved in arff format. Now open the weka application.
5. Click on weka-3-4, then Weka dialog box is displayed on the screen.
6. In that dialog box there are four modes, click on explorer.
7. Explorer shows many options. In that click on 'open file' and select the arff file
8. Click on edit button which shows weather table on weka.

Result:



Viewer

Relation: weather

No.	1: outlook Nominal	2: temparature Numeric	3: humidity Numeric	4: windy Nominal	5: **play** Nominal
1	sunny	72.0	95.0	true	no
2	sunny	69.0	70.0	false	yes
3	sunny	83.0	86.0	false	yes
4	sunny	85.0	85.0	false	no
5	rainy	65.0	70.0	true	no
6	rainy	68.0	80.0	false	yes
7	rainy	70.0	86.0	false	yes
8	rainy	75.0	80.0	false	yes
9	overcast	80.0	90.0	true	no
10	overcast	64.0	65.0	false	yes

The Datafile was successfully created and loaded into the WEKA explorer.

Conclusion:

Successful procedure to load arff dataset in WEKA was followed.

Practical – 2

Aim of Experiment:

(i) Build Data Warehouse/Data Mart (using open source tools like Pentaho Data Integration Tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft-SSIS,Informatica,Business Objects, etc.,)

(ii) Identify source tables and populate sample data.

Description:

The data warehouse contains 4 tables:

1. Date dimension: contains every single date from 2006 to 2016.
2. Customer dimension: contains 100 customers. To be simple we'll make it type 1 so we
don't create a new row for each change.
3. Van dimension: contains 20 vans. To be simple we'll make it type 1 so we don't create a
new row for each change.
4. Hire fact table: contains 1000 hire transactions since 1st Jan 2011. It is a daily snapshot fact table so that every day we insert 1000 rows into this fact table. So over time we can track
the changes of total bill, van charges, satnav income, etc.

Procedure:

1. So now we are going to create the 3 tables in HireBase database: Customer, Van, and Hire. Then we populate them.

Following script is used to create and populate the source tables:

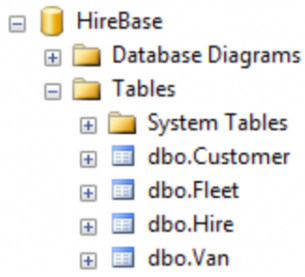
```
1.  -- Create database
2.  create database HireBase
3.  go
4.  use HireBase
5.  go
6.
7.  -- Create customer table
8.  if exists (select * from sys.tables where name = 'Customer')
9.  drop table Customer
10. go
11.
12. create table Customer
13. ( CustomerId varchar(20) not null primary key,
14.  CustomerName varchar(30), DateOfBirth date, Town varchar(50),
15.  TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
```

```
16. )
17. go
18.
19. -- Populate Customer
20. truncate table Customer
21. go
22.
23. declare @i int, @si varchar(10), @startdate date
24. set @i = 1
25. while @i <= 100
26. begin
27.  set @si = right('0'+CONVERT(varchar(10), @i),2)
28.  insert into Customer
29.  ( CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo, DrivingLicenceNo, Occupation)
30.  values
31.  ( 'N'+@si, 'Customer'+@si, DATEADD(d,@i-1,'2000-01-01'), 'Town'+@si, 'Phone'+@si, 'Licence'+@si,
    'Occupation'+@si)
32.  set @i = @i + 1
33. end
34. go
35.
36. select * from Customer
37.
38. -- Create Van table
39. if exists (select * from sys.tables where name = 'Van')
40. drop table Van
41. go
42.
43. create table Van
44. ( RegNo varchar(10) not null primary key,
45.  Make varchar(30), Model varchar(30), [Year] varchar(4),
46.  Colour varchar(20), CC int, Class varchar(10)
47. )
48. go
49.
50. -- Populate Van table
51. truncate table Van
52. go
53.
54. declare @i int, @si varchar(10)
55. set @i = 1
56. while @i <= 20
57. begin
58.  set @si = convert(varchar, @i)
59.  insert into Van
60.  ( RegNo, Make, Model, [Year], Colour, CC, Class)
61.  values
62.  ( 'Reg'+@si, 'Make'+@si, 'Model'+@si,
63.  case @i%4 when 0 then 2008 when 1 then 2009 when 2 then 2010 when 3 then 2011 end,
64.  case when @i%5<3 then 'White' else 'Black' end,
65.  case @i%3 when 0 then 2000 when 1 then 2500 when 2 then 3000 end,
66.  case @i%3 when 0 then 'Small' when 1 then 'Medium' when 2 then 'Large' end)
67.  set @i = @i + 1
68. end
69. go
70.
71. select * from Van
72.
73. -- Create Hire table
74. if exists (select * from sys.tables where name = 'Hire')
75. drop table Hire
76. go
77.
78. create table Hire
79. ( HireId varchar(10) not null primary key,
80.  HireDate date not null,
81.  CustomerId varchar(20) not null,
82.  RegNo varchar(10), NoOfDays int, VanHire money, SatNavHire money,
83.  Insurance money, DamageWaiver money, TotalBill money
84. )
85. go
86.
87. -- Populate Hire table
```

```
truncate table Hire
go

declare @i int, @si varchar(10), @DaysFrom1stJan int, @CustomerId int, @RegNo int, @mi int
set @i = 1
while @i <= 1000
begin
 set @si = right('000'+convert(varchar(10), @i),4) -- string of i
 set @DaysFrom1stJan = (@i-1)%200 --The Hire Date is derived from i modulo 200
 set @CustomerId = (@i-1)%100+1 --The CustomerId is derived from i modulo 100
 set @RegNo = (@i-1)%20+1 --The Van RegNo is derived from i modulo 20
 set @mi = (@i-1)%3+1 --i modulo 3
insert into Hire (HireId, HireDate, CustomerId, RegNo, NoOfDays, VanHire, SatNavHire, Insurance,
    DamageWaiver, TotalBill)
values ('H'+@si, DateAdd(d, @DaysFrom1stJan, '2011-01-01'),
left('N0'+CONVERT(varchar(10),@CustomerId),3), 'Reg'+CONVERT(varchar(10), @RegNo),
@mi, @mi*100, @mi*10, @mi*20, @mi*40, @mi*170)
set @i += 1
end
go

select * from Hire
```

Result:

The Final Outcome is as follow :



Customer table:

	CustomerId	CustomerName	DateOfBirth	Town	TelephoneNo	DrivingLicenceNo	Occupation
1	N00	Customer00	2000-04-09	Town00	Phone00	Licence00	Occupation00
2	N01	Customer01	2000-01-01	Town01	Phone01	Licence01	Occupation01
3	N02	Customer02	2000-01-02	Town02	Phone02	Licence02	Occupation02
4	N03	Customer03	2000-01-03	Town03	Phone03	Licence03	Occupation03
5	N04	Customer04	2000-01-04	Town04	Phone04	Licence04	Occupation04
6	N05	Customer05	2000-01-05	Town05	Phone05	Licence05	Occupation05

Van table:

	RegNo	Make	Model	Year	Colour	CC	Class
1	Reg1	Make1	Model1	2009	White	2500	Medium
2	Reg10	Make10	Model10	2010	White	2500	Medium
3	Reg11	Make11	Model11	2011	White	3000	Large
4	Reg12	Make12	Model12	2008	White	2000	Small
5	Reg13	Make13	Model13	2009	Black	2500	Medium

Hire table:

HireId	HireDate	CustomerId	RegNo	NoOfDays	VanHire	SatNavHire	Insurance	DamageWaiver	TotalBill
H0001	2011-01-01	N01	Reg1	1	100.00	10.00	20.00	40.00	170.00
H0002	2011-01-02	N02	Reg2	2	200.00	20.00	40.00	80.00	340.00
H0003	2011-01-03	N03	Reg3	3	300.00	30.00	60.00	120.00	510.00
H0004	2011-01-04	N04	Reg4	1	100.00	10.00	20.00	40.00	170.00
H0005	2011-01-05	N05	Reg5	2	200.00	20.00	40.00	80.00	340.00

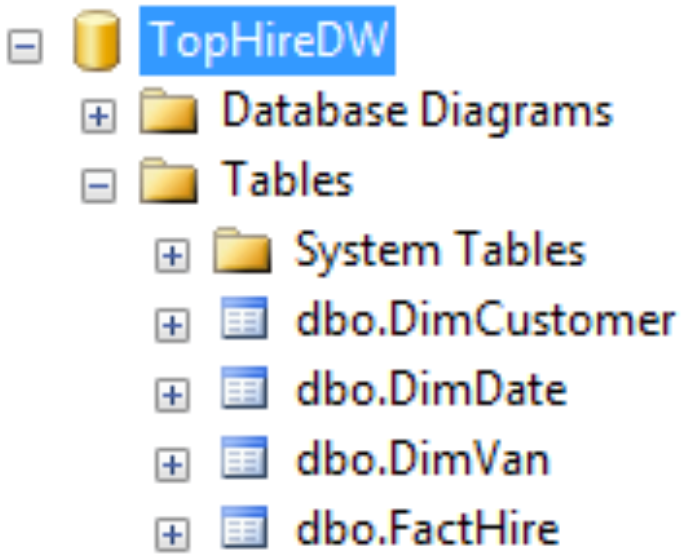
2. Create Data WareHouse

So now we are going to create the 3 dimension tables and 1 fact table in the data warehouse: DimDate, DimCustomer, DimVan and FactHire. We are going to populate the 3 dimensions but we'll leave the fact table empty. The purpose of this article is to show how to populate the fact table using SSIS.

```
1.  -- Create the data warehouse
2.  create database TopHireDW
3.  go
4.  use TopHireDW
5.  go
6.
7.  -- Create Date Dimension
8.  if exists (select * from sys.tables where name = 'DimDate')
9.  drop table DimDate
10. go
11.
12. create table DimDate
13. ( DateKey int not null primary key,
14.  [Year] varchar(7), [Month] varchar(7), [Date] date, DateString varchar(10))
15. go
16.
17. -- Populate Date Dimension
18. truncate table DimDate
19. go
20.
21. declare @i int, @Date date, @StartDate date, @EndDate date, @DateKey int,
22.  @DateString varchar(10), @Year varchar(4),
23.  @Month varchar(7), @Date1 varchar(20)
24. set @StartDate = '2006-01-01'
25. set @EndDate = '2016-12-31'
26. set @Date = @StartDate
27.
28. insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
29.  values (0, 'Unknown', 'Unknown', '0001-01-01', 'Unknown') --The unknown row
30.
31. while @Date <= @EndDate
32. begin
33.  set @DateString = convert(varchar(10), @Date, 20)
34.  set @DateKey = convert(int, replace(@DateString,'-',''))
35.  set @Year = left(@DateString,4)
36.  set @Month = left(@DateString, 7)
37.  insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
38.  values (@DateKey, @Year, @Month, @Date, @DateString)
39.  set @Date = dateadd(d, 1, @Date)
40. end
41. go
42.
43. select * from DimDate
44.
45. -- Create Customer dimension
46. if exists (select * from sys.tables where name = 'DimCustomer')
47. drop table DimCustomer
48. go
49.
50. create table DimCustomer
51. ( CustomerKey int not null identity(1,1) primary key,
52.  CustomerId varchar(20) not null,
53.  CustomerName varchar(30), DateOfBirth date, Town varchar(50),
54.  TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
55. )
56. go
57.
58. insert into DimCustomer (CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo,
59.  DrivingLicenceNo, Occupation)
60. select * from HireBase.dbo.Customer
61.
62. select * from DimCustomer
63.
```

```
64. -- Create Van dimension
65. if exists (select * from sys.tables where name = 'DimVan')
66. drop table DimVan
67. go
68.
69. create table DimVan
70. ( VanKey int not null identity(1,1) primary key,
71.  RegNo varchar(10) not null,
72.  Make varchar(30), Model varchar(30), [Year] varchar(4),
73.  Colour varchar(20), CC int, Class varchar(10)
74. )
75. go
76.
77. insert into DimVan (RegNo, Make, Model, [Year], Colour, CC, Class)
78. select * from HireBase.dbo.Van
79. go
80.
81. select * from DimVan
82.
83. -- Create Hire fact table
84. if exists (select * from sys.tables where name = 'FactHire')
85. drop table FactHire
86. go
87.
88. create table FactHire
89. ( SnapshotDateKey int not null, --Daily periodic snapshot fact table
90.  HireDateKey int not null, CustomerKey int not null, VanKey int not null, --Dimension Keys
91.  HireId varchar(10) not null, --Degenerate Dimension
92.  NoOfDays int, VanHire money, SatNavHire money,
93.  Insurance money, DamageWaiver money, TotalBill money
94. )
95. go
96.
97. select * from FactHire
```

Result :



Date Dimension:

	DateKey	Year	Month	Date	DateString
1	0	Unknown	Unknown	0001-01-01	Unknown
2	20060101	2006	2006-01	2006-01-01	2006-01-01
3	20060102	2006	2006-01	2006-01-02	2006-01-02
4	20060103	2006	2006-01	2006-01-03	2006-01-03
5	20060104	2006	2006-01	2006-01-04	2006-01-04
6	20060105	2006	2006-01	2006-01-05	2006-01-05

Customer Dimension:

	CustomerKey	CustomerId	CustomerName	DateOfBirth	Town	TelephoneNo	DrivingLicenceNo	Occupation
1	1	N01	Customer01	2000-01-01	Town01	Phone01	Licence01	Occupation01
2	2	N02	Customer02	2000-01-02	Town02	Phone02	Licence02	Occupation02
3	3	N03	Customer03	2000-01-03	Town03	Phone03	Licence03	Occupation03
4	4	N04	Customer04	2000-01-04	Town04	Phone04	Licence04	Occupation04
5	5	N05	Customer05	2000-01-05	Town05	Phone05	Licence05	Occupation05

Van Dimension:

	VanKey	RegNo	Make	Model	Year	Colour	CC	Class
1	1	Reg1	Make1	Model1	2009	White	2500	Medium
2	2	Reg10	Make10	Model10	2010	White	2500	Medium
3	3	Reg11	Make11	Model11	2011	White	3000	Large
4	4	Reg12	Make12	Model12	2008	White	2000	Small
5	5	Reg13	Make13	Model13	2009	Black	2500	Medium

Hire Fact Table:

Results Messages

	SnapshotDateKey	HireDateKey	CustomerKey	VanKey	HireId	NoOfDays	VanHire	SatNavHire	Insurance	DamageWaiver	TotalBill

Conclusion:

We Build a data warehouse and populated the source tables with sample data.

Practical – 3

Aim Of Experiment:

1. Conversion of various Datafile.
2. Training the dataset for an application and applying test options to it.

Description:

Dataset files can be of many different formats. We sometimes may require to convert the dataset file to a particular format as per need. WEKA provide support for many file formats. The arff file format is the default file format to work in WEKA.
The file formats supported by WEKA are:

Arff	arff.gz	bsi	csv	dat	data
Json	json.gz	libsvm	m	names	xrff

Arff, JSON, CSV are the widely used file format in dataset.

WEKA provide support of many in-built machine learning algorithms. The machine learning algorithms can be used to perform classification, clustering and regression analysis. The machine learning algorithms can be trained using the dataset and can be checked for metrics against the test options provided in WEKA.

Test Options

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box.

There are four test modes:

1. **Use training set.** The classifier is evaluated on how well it predicts the class of the instances it was trained on.
2. **Supplied test set.** The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.
3. **Cross-validation.** The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.
4. **Percentage split.** The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

Unless you have your own training set or a client supplied test set, you would use cross-validation or percentage split options. Under cross-validation, you can set the number of folds in which entire data would be split and used during each iteration of training. In the percentage split, you will split the data between training and testing using the set split percentage.

Procedure:

Conversion of Datafile Formats

1. Open WEKA
2. Open WEKA explorer.
3. Click on the open file button and load the datafile in any of the supported format.
4. Click on the Save button and select the file format we want for the datafile.
5. Confirm by naming the file , choosing the file format and click on save

The Data File is saved in that file format.
This can be used to convert the datafile in/from various file format supported by Weka.

Training the dataset for an application

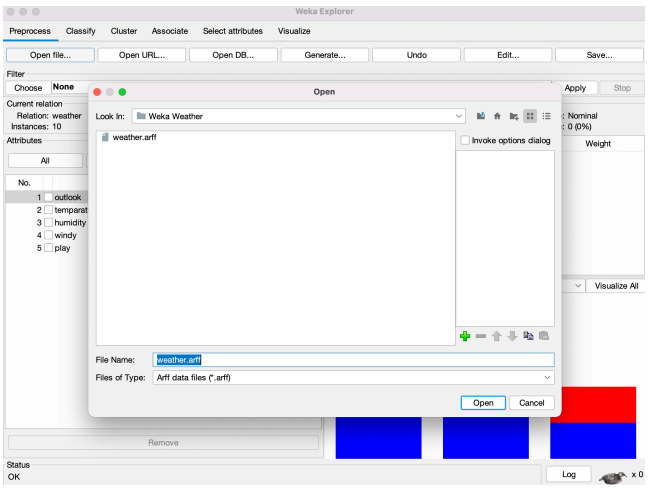
We will use the weather data to train a classifier to predict if play is possible or not .

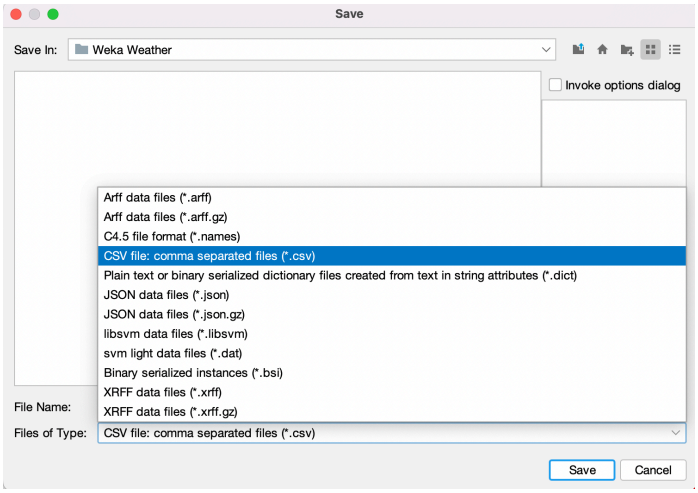
1. Open WEKA. Go to Explorer.
2. Load the dataset into the explorer using open file button.
3. We will use the nominal weather dataset to predict if play /not play, using naïve bayes classifier.
4. We see the result of applying the classifier by choosing one of the test options and looking at the metrics in classifier output window.

RESULT:

Conversion of file format:

We were able to convert the arff file to Csv and vice versa.





Training the Dataset

We trained the dataset to classify the play attribute.

Test mode: evaluate on training data.

```
=== Summary ===

Correctly Classified Instances          13               92.8571 %
Incorrectly Classified Instances         1                7.1429 %
Kappa statistic                          0.8372
Mean absolute error                      0.2917
Root mean squared error                  0.3392
Relative absolute error                 62.8233 %
Root relative squared error             70.7422 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.200    0.900      1.000    0.947      0.849    0.922     0.947     yes
                 0.800    0.000    1.000      0.800    0.889      0.849    0.911     0.911     no
Weighted Avg.    0.929    0.129    0.936      0.929    0.926      0.849    0.918     0.934

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 1 4 | b = no
```

Test mode: 10-fold cross-validation

```
=== Summary ===

Correctly Classified Instances           8               57.1429 %
Incorrectly Classified Instances         6               42.8571 %
Kappa statistic                         -0.0244
Mean absolute error                      0.4374
Root mean squared error                  0.4916
Relative absolute error                 91.8631 %
Root relative squared error             99.6492 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.778    0.800    0.636      0.778    0.700      -0.026   0.578     0.697     yes
                 0.200    0.222    0.333      0.200    0.250      -0.026   0.578     0.557     no
Weighted Avg.    0.571    0.594    0.528      0.571    0.539      -0.026   0.578     0.647

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 4 1 | b = no
```

Test mode: 5-fold cross-validation

```
=== Summary ===

Correctly Classified Instances           8               57.1429 %
Incorrectly Classified Instances         6               42.8571 %
Kappa statistic                         -0.0244
Mean absolute error                      0.4423
Root mean squared error                  0.4964
Relative absolute error                 94.7104 %
Root relative squared error            103.1962 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.778    0.800    0.636      0.778    0.700      -0.026   0.556     0.690     yes
                 0.200    0.222    0.333      0.200    0.250      -0.026   0.544     0.524     no
Weighted Avg.    0.571    0.594    0.528      0.571    0.539      -0.026   0.552     0.631

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 4 1 | b = no
```

Test mode: split 66.0% train, remainder test

```
=== Summary ===

Correctly Classified Instances           3               60      %
Incorrectly Classified Instances         2               40      %
Kappa statistic                          0
Mean absolute error                      0.4437
Root mean squared error                  0.5023
Relative absolute error                 93.8582 %
Root relative squared error            102.2471 %
Total Number of Instances                5

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.600      1.000    0.750      ?        0.667     0.867     yes
                 0.000    0.000    ?          0.000    ?          ?        0.667     0.583     no
Weighted Avg.    0.600    0.600    ?          0.600    ?          ?        0.667     0.753

=== Confusion Matrix ===

 a b   <-- classified as
 3 0 | a = yes
 2 0 | b = no
```

Conclusion:

We converted arff to CSV file .
We trained a naïve bayes classification model in the weather dataset and tested the model with various test options. The best was against the training data.

Practical – 4

Aim of Experiment:

To apply pre-processing techniques on the dataset.

Description:

Real world databases are highly influenced to noise, missing and inconsistency due to their queue size so the data can be pre-processed to improve the quality of data and missing results and it also improves the efficiency. For example, the data may contain null fields, it may contain columns that are irrelevant to the current analysis, and so on. Thus, the data must be pre-processed to meet the requirements of the type of analysis you are seeking. This is the done in the pre-processing module.

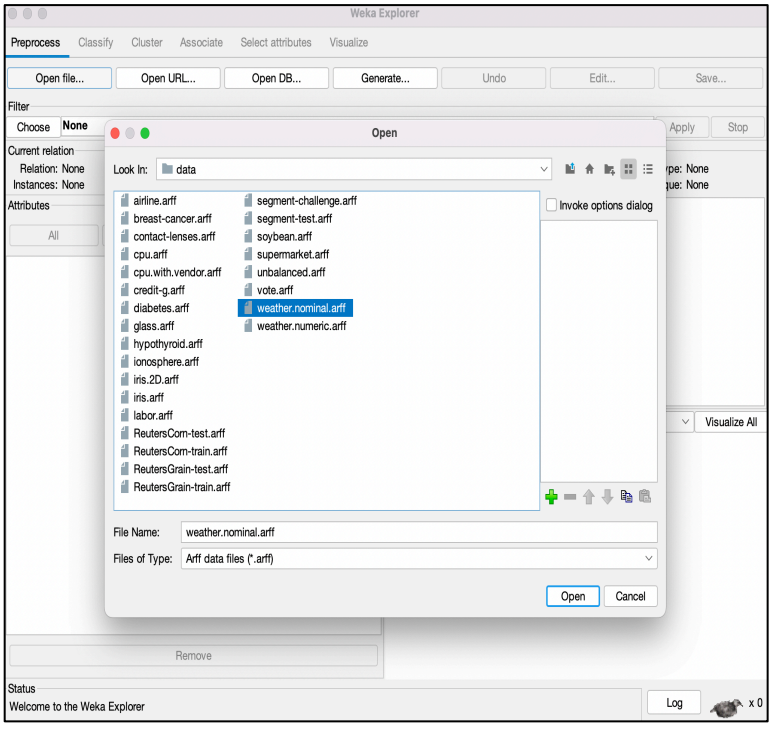
There are 3 pre-processing techniques they are:

1) Add
2) Remove
3) Normalization

Procedure:

Load the dataset into WEKA :

1. Open WEKA.
2. Click on the 'Open File' button.
3. Navigate to Weka Data Folder containing sample datasets.
4. Choose the Weather Dataset and click Open.

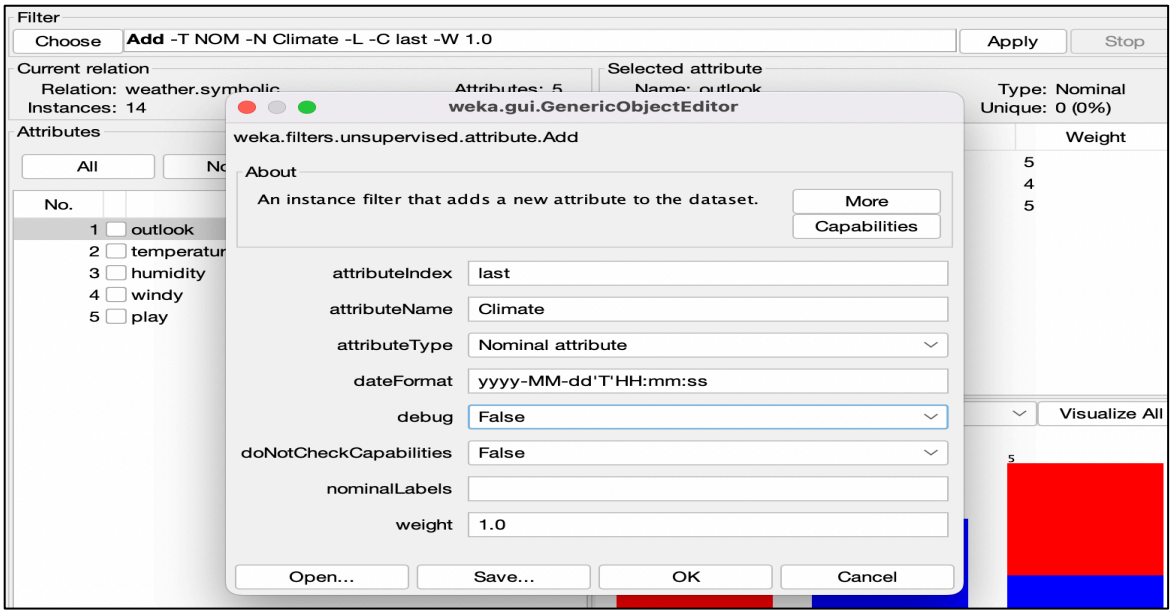


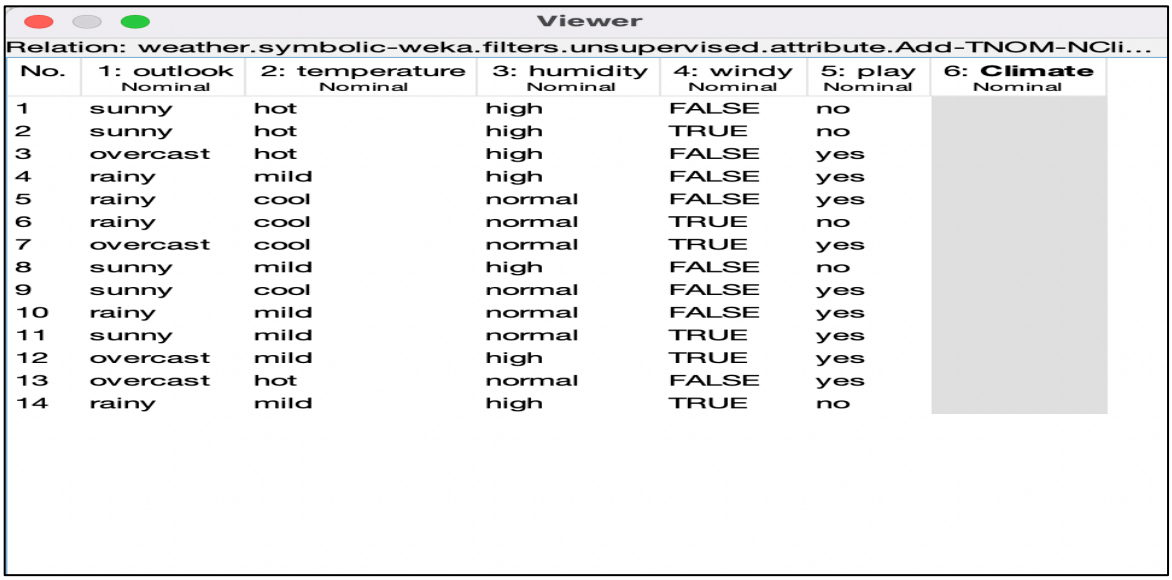
Relation: weather.symbolic

No.	1: outlook Nominal	2: temperature Nominal	3: humidity Nominal	4: windy Nominal	5: play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

Adding Attribute :

1. After loading the Dataset into Explorer.
2. Under the Filter Tab. Click on the choose button and select filter option.
3. In Filters, we have Supervised and Unsupervised data.
4. Click on Unsupervised data.
5. Select the Attribute add. A new window is opened.
6. In that we add attribute index, type, data format, nominal label values for Climate. Click on OK.
7. Click on the Apply button.
8. A new Attribute is added to Weather Table.
9. Save the File and preview the new Data File.

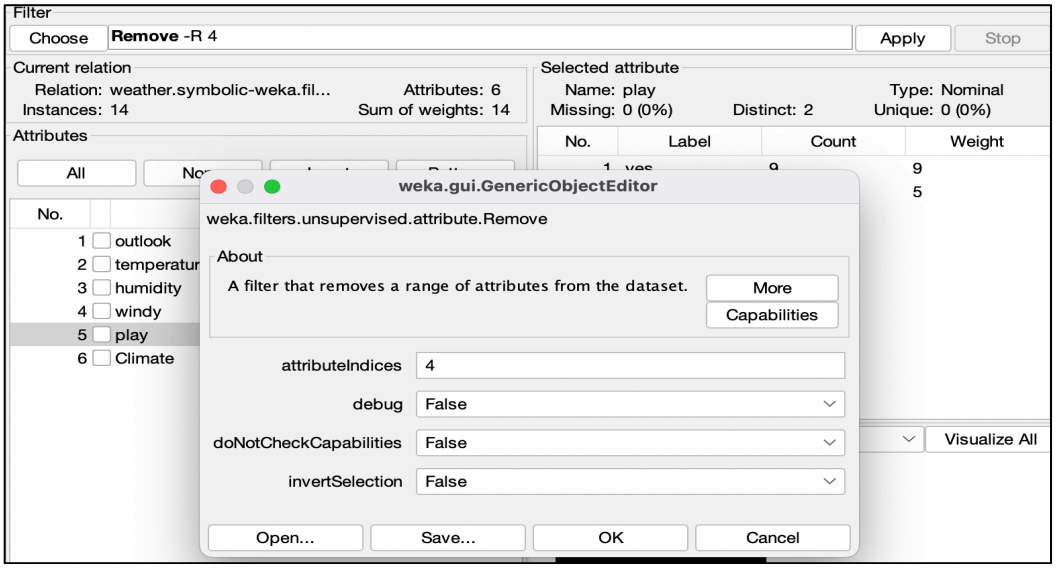


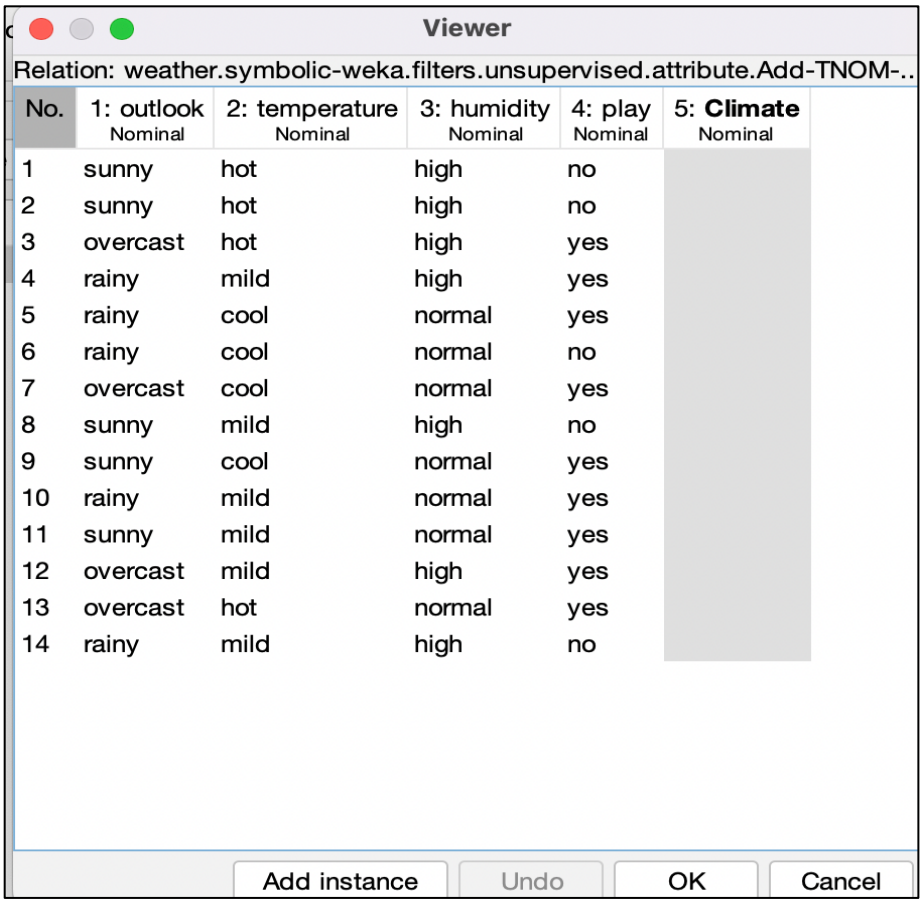


No.	1: outlook Nominal	2: temperature Nominal	3: humidity Nominal	4: windy Nominal	5: play Nominal	6: Climate Nominal
1	sunny	hot	high	FALSE	no	
2	sunny	hot	high	TRUE	no	
3	overcast	hot	high	FALSE	yes	
4	rainy	mild	high	FALSE	yes	
5	rainy	cool	normal	FALSE	yes	
6	rainy	cool	normal	TRUE	no	
7	overcast	cool	normal	TRUE	yes	
8	sunny	mild	high	FALSE	no	
9	sunny	cool	normal	FALSE	yes	
10	rainy	mild	normal	FALSE	yes	
11	sunny	mild	normal	TRUE	yes	
12	overcast	mild	high	TRUE	yes	
13	overcast	hot	normal	FALSE	yes	
14	rainy	mild	high	TRUE	no	

Removing Attribute :

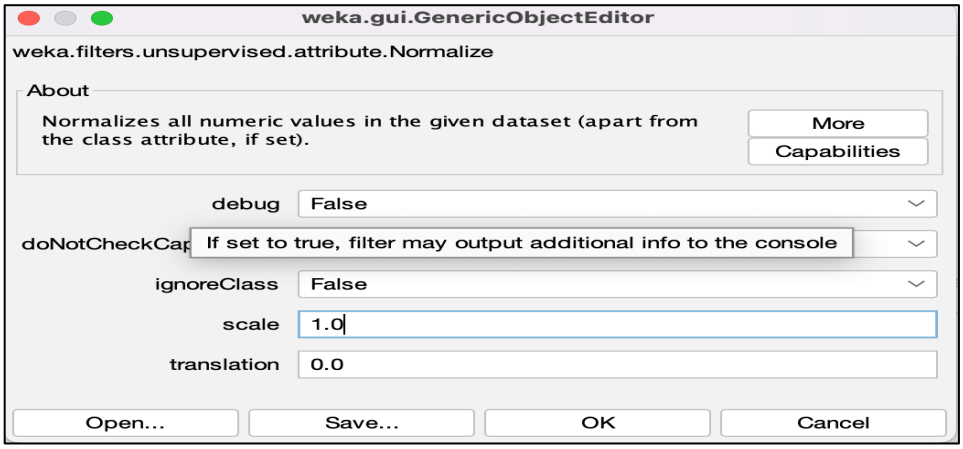
1. After Loading the dataset into WEKA explorer.
2. Under the Filter Tab. Click on the choose button and select filter option.
3. In Filters, we have Supervised and Unsupervised data.
4. Click on Unsupervised data.
5. Select the Attribute Remove.
6. Select the attributes to Remove.(windy in our case)
7. Click the Remove Button.
8. Apply the changes. The Weather Dataset is changed it can be Viewed in Edit.

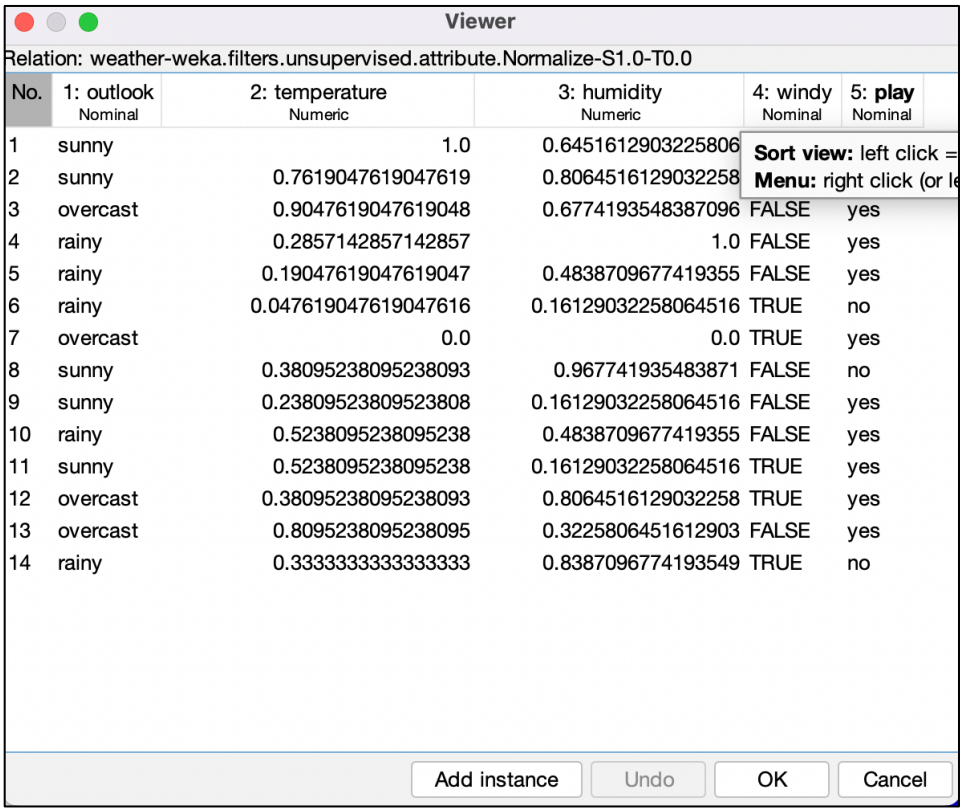




Normalizing an attribute :

1. After Loading the dataset into WEKA explorer.
2. Under the Filter Tab. Click on the choose button and select filter option.
3. In Filters, we have Supervised and Unsupervised data.
4. Click on Unsupervised data.
5. Select the Attribute Normalize.
6. The numeric attributes are normalized.(temperature, humidity in our case)
7. Click the OK Button and continue.
8. Apply the changes. The Weather Dataset is Normalized it can be Viewed in Edit.





No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	1.0	0.6451612903225806		
2	sunny	0.7619047619047619	0.8064516129032258		
3	overcast	0.9047619047619048	0.6774193548387096	FALSE	yes
4	rainy	0.2857142857142857	1.0	FALSE	yes
5	rainy	0.19047619047619047	0.4838709677419355	FALSE	yes
6	rainy	0.047619047619047616	0.16129032258064516	TRUE	no
7	overcast	0.0	0.0	TRUE	yes
8	sunny	0.38095238095238093	0.967741935483871	FALSE	no
9	sunny	0.23809523809523808	0.16129032258064516	FALSE	yes
10	rainy	0.5238095238095238	0.4838709677419355	FALSE	yes
11	sunny	0.5238095238095238	0.16129032258064516	TRUE	yes
12	overcast	0.38095238095238093	0.8064516129032258	TRUE	yes
13	overcast	0.8095238095238095	0.3225806451612903	FALSE	yes
14	rainy	0.3333333333333333	0.8387096774193549	TRUE	no

Sort view: left click =
Menu: right click (or l

Conclusion :

We performed pre-processing techniques on the Weather dataset including adding attribute, removing attribute and normalizing attribute.

Practical – 5

Aim of Experiment :

Perform Normalization of Dataset using Knowledge Flow in WEKA.

Description :

The knowledge flow provides an alternative way to the explorer as a graphical front end to WEKA's algorithm.

The Knowledge Flow presents a data-flow inspired interface to WEKA. The user can select WEKA components from a palette, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analysing data.
At present, all of WEKA's classifiers, filters, clusterers, associators, loaders and savers are available in the Knowledge Flow along with some extra tools.

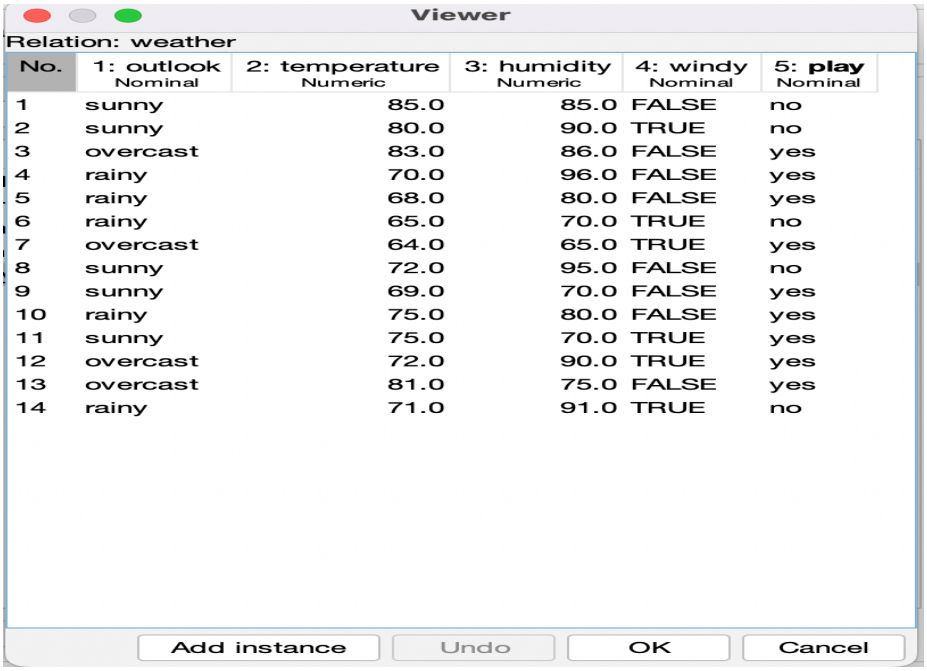
Knowledge flow presents a dataflow interface to WEKA. The user can select WEKA components from a toolbar placed them on a layout campus and connect them together in order to form a knowledge flow for processing and analysing the data.

Procedure :

A. Using Weather Dataset

Open and View the Weather Dataset

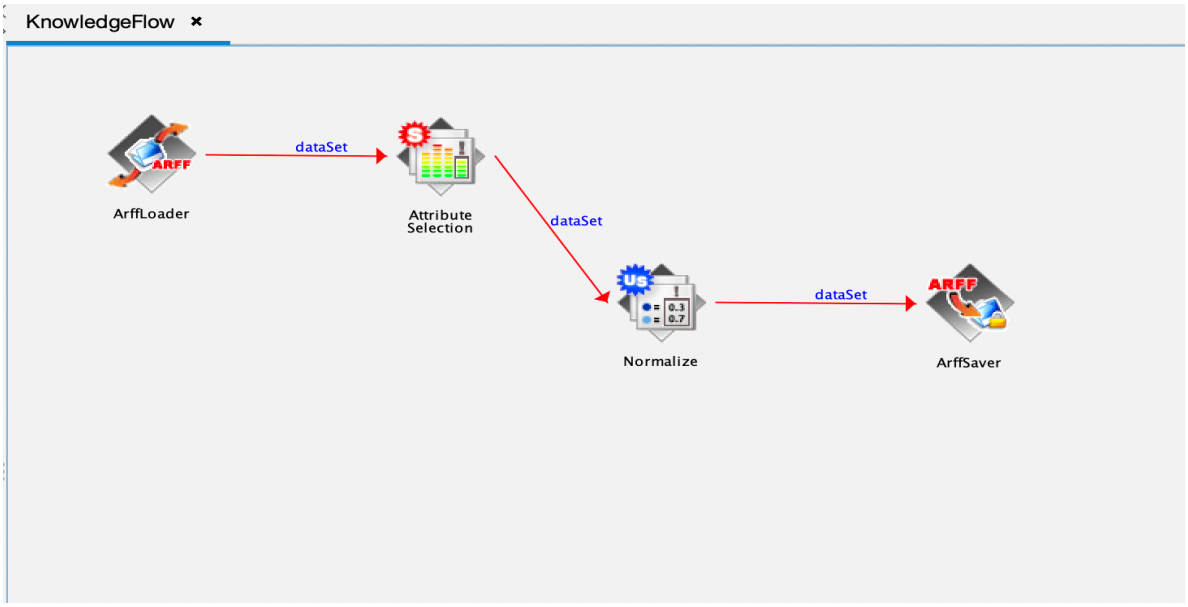
1. Open WEKA.
2. Load the numeric weather dataset into explorer.
3. Click on Edit button to view the dataset in viewer.



Knowledge Flow Procedure :

1. Open the WEKA Application.
2. Open the Knowledge Flow Window.
3. Select the Data Source component and add Arff Loader into the knowledge layout canvas.
4. Select the Filters component and add Attribute Selection and Normalize into the knowledge layout canvas
5. Select the Data Sinks component and add Arff Saver into the knowledge layout canvas.
6. Right click on Arff Loader and select Configure option then the new window will be opened and select Weather.arff
7. Right click on Arff Loader and select Dataset option then establish a link between Arff Loader and Attribute Selection.
8. Right click on Attribute Selection and select Dataset option then establish a link between Attribute Selection and Normalize
9. Right click on Attribute Selection and select Configure option and choose the best attribute for Weather data.
10. Right click on Normalize and select Dataset option then establish a link between Normalize and Arff Saver.
11. Right click on Arff Saver and select Configure option then new window will be opened and set the path, enter .arff in look in dialog box to save normalize data.
12. Right click on Arff Loader and click on Start Loading option then everything will be executed one by one.
13. Check whether output is created or not by selecting the preferred path.
14. Rename the data name as a.arff
15. Double click on a.arff then automatically the output will be opened in MS-Excel.

Result :



Viewer

Relation: weather-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: **play** Nominal
1	sunny	1.0	0.645161	FALSE	no
2	sunny	Right click (or left+alt) for context menu			
3	overcast	0.904762	0.677419	FALSE	yes
4	rainy	0.285714	1.0	FALSE	yes
5	rainy	0.190476	0.483871	FALSE	yes
6	rainy	0.047619	0.16129	TRUE	no
7	overcast	0.0	0.0	TRUE	yes
8	sunny	0.380952	0.967742	FALSE	no
9	sunny	0.238095	0.16129	FALSE	yes
10	rainy	0.52381	0.483871	FALSE	yes
11	sunny	0.52381	0.16129	TRUE	yes
12	overcast	0.380952	0.806452	TRUE	yes
13	overcast	0.809524	0.322581	FALSE	yes
14	rainy	0.333333	0.83871	TRUE	no

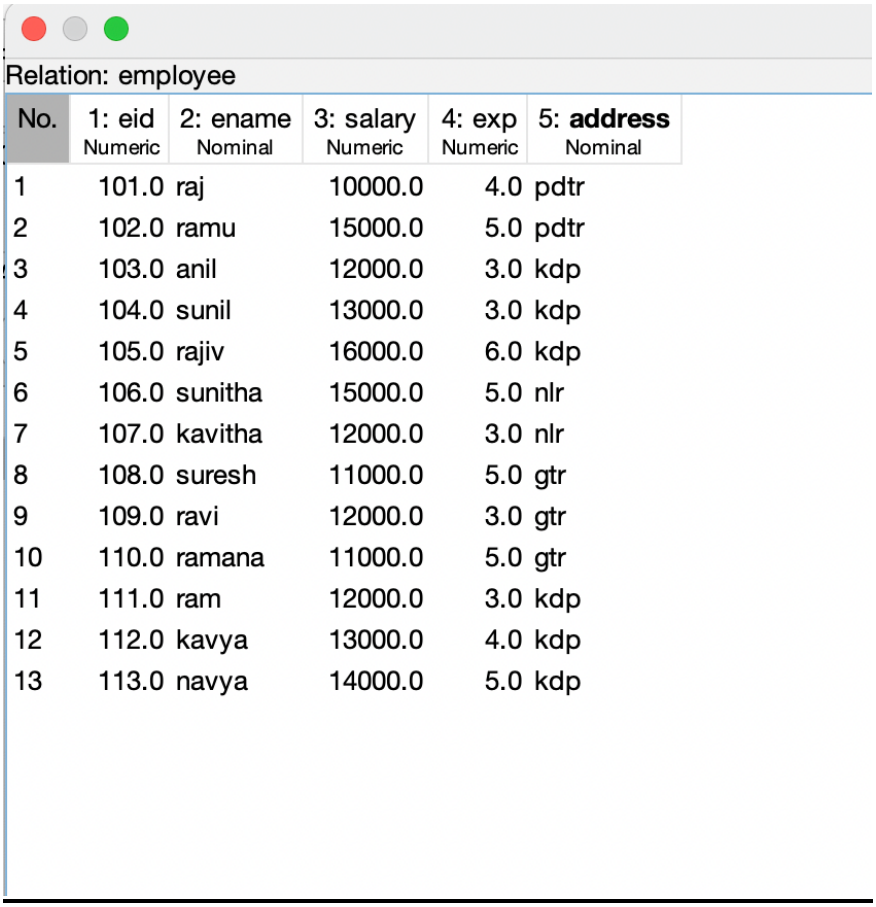
B. Using Employee Dataset

Create the Employee Table:

9. Open a text editor
10. Type the following training data set with the help of Notepad for Weather Table.

```
@relation employee

@attribute eid numeric
@attribute ename
{raj,ramu,anil,sunil,rajiv,sunitha,kavitha,suresh,ravi,ramana,ram,kavya,navya}
@attribute salary numeric
@attribute exp numeric
@attribute address {pdtr,kdp,nlr,gtr}

@data
101,raj,10000,4,pdtr
102,ramu,15000,5,pdtr
103,anil,12000,3,kdp
104,sunil,13000,3,kdp
105,rajiv,16000,6,kdp
106,sunitha,15000,5,nlr
107,kavitha,12000,3,nlr
108,suresh,11000,5,gtr
109,ravi,12000,3,gtr
110,ramana,11000,5,gtr
111,ram,12000,3,kdp
112,kavya,13000,4,kdp
113,navya,14000,5,kdp
```

11. After that the file is saved with .arff file format
12. The file is saved in arff format. Now open the weka application.
13. Click on weka-3-4, then Weka dialog box is displayed on the screen.
14. In that dialog box there are four modes, click on explorer.
15. Explorer shows many options. In that click on 'open file' and select the arff file
16. Click on edit button which shows weather table on weka.



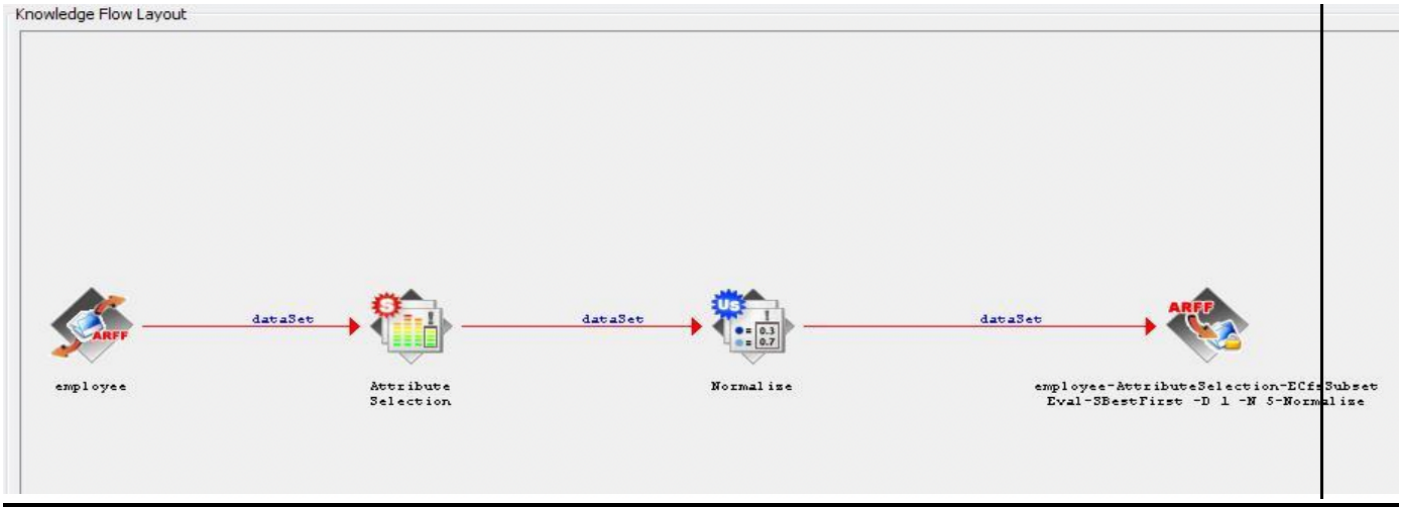
Relation: employee

No.	1: eid Numeric	2: ename Nominal	3: salary Numeric	4: exp Numeric	5: **address** Nominal
1	101.0	raj	10000.0	4.0	pdtr
2	102.0	ramu	15000.0	5.0	pdtr
3	103.0	anil	12000.0	3.0	kdp
4	104.0	sunil	13000.0	3.0	kdp
5	105.0	rajiv	16000.0	6.0	kdp
6	106.0	sunitha	15000.0	5.0	nlr
7	107.0	kavitha	12000.0	3.0	nlr
8	108.0	suresh	11000.0	5.0	gtr
9	109.0	ravi	12000.0	3.0	gtr
10	110.0	ramana	11000.0	5.0	gtr
11	111.0	ram	12000.0	3.0	kdp
12	112.0	kavya	13000.0	4.0	kdp
13	113.0	navya	14000.0	5.0	kdp

Knowledge Flow Procedure :

1. Open the WEKA Application.
2. Open the Knowledge Flow Window.
3. Select the Data Source component and add Arff Loader into the knowledge layout canvas.
4. Select the Filters component and add Attribute Selection and Normalize into the knowledge layout canvas
5. Select the Data Sinks component and add Arff Saver into the knowledge layout canvas.
6. Right click on Arff Loader and select Configure option then the new window will be opened and select Employee.arff

7. Right click on Arff Loader and select Dataset option then establish a link between Arff Loader and Attribute Selection.
8. Right click on Attribute Selection and select Dataset option then establish a link between Attribute Selection and Normalize
9. Right click on Attribute Selection and select Configure option and choose the best attribute for Employee data.
10. Right click on Normalize and select Dataset option then establish a link between Normalize and Arff Saver.
11. Right click on Arff Saver and select Configure option then new window will be opened and set the path, enter .arff in look in dialog box to save normalize data.
12. Right click on Arff Loader and click on Start Loading option then everything will be executed one by one.
13. Check whether output is created or not by selecting the preferred path.
14. Rename the data name as a.arff
15. Double click on a.arff then automatically the output will be opened in MS-Excel.



Conclusion :

We performed the normalisation of data using knowledge floe interface .

Practical – 6

Aim Of Experiment :

To Perform Visualization on
a. Weather Data.
b. Banking Data.

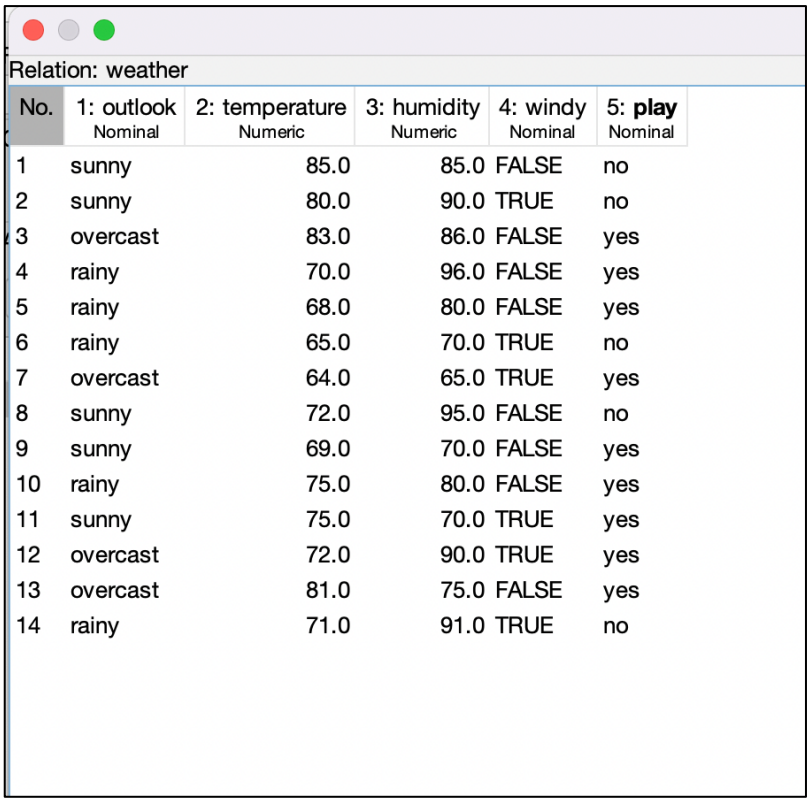
Description :

WEKA's visualization allows you to visualize a 2-D plot of the current working relation. Visualization is very useful in practice, it helps to determine difficulty of the learning problem. WEKA can visualize single attributes (1-d) and pairs of attributes (2-d), rotate 3-d visualizations (Xgobi-style). WEKA has "Jitter" option to deal with nominal attributes and to detect "hidden" data points.

Procedure :

Weather Dataset

1. Open Weka application
2. Load the Weather numeric arff file into the explore window.
3. Click on edit button to view the dataset in viewer.



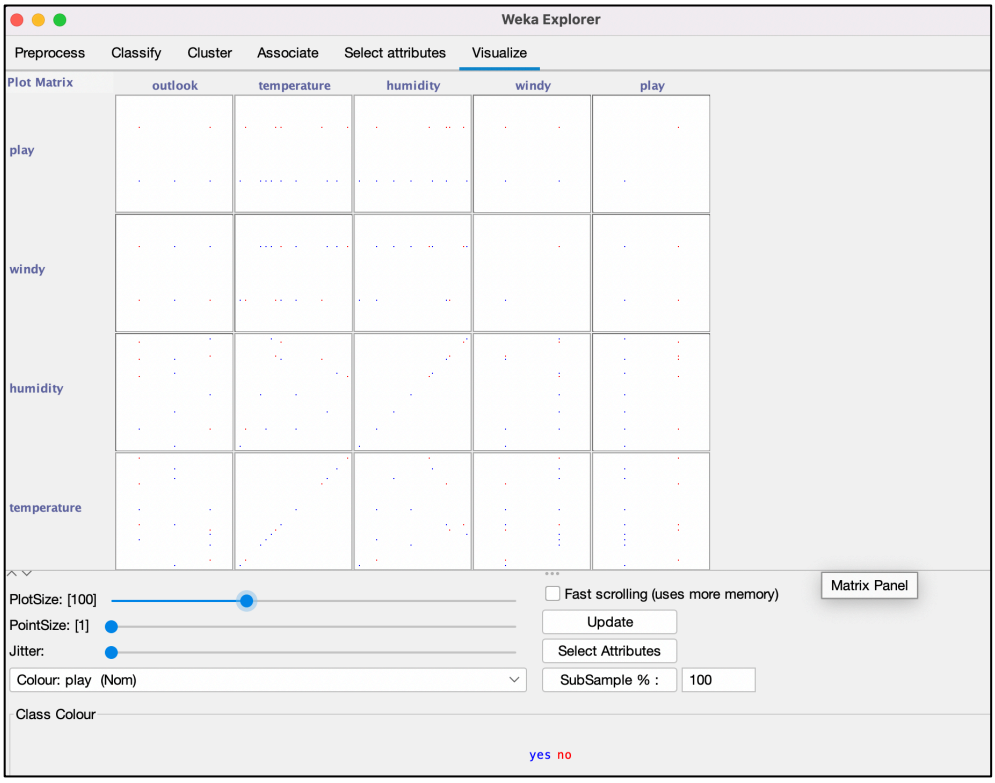
Relation: weather

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: **play** Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

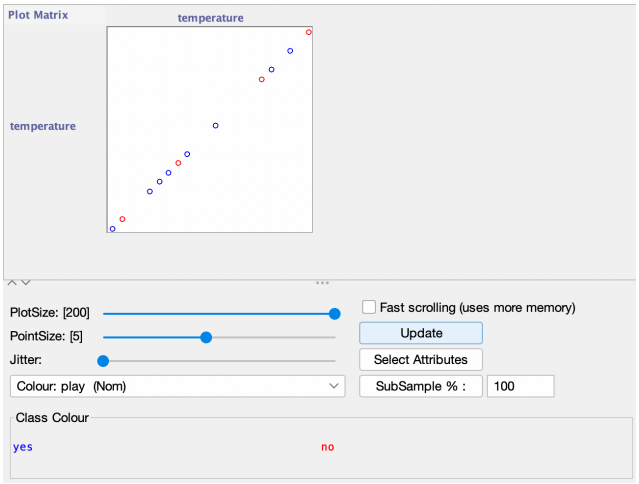
4. Click on the visualise tab on top bar of explorer window.
5. Open the visualise tab then Plot Matrix is displayed on the Screen.
6. After that we select the Select Attribute button, then select Outlook attribute and click OK.
7. Click on the Update button to display the output.
8. After that select the Select Attribute button and select Temperature attribute and then click OK.
9. Increase the Plot Size and Point Size.
10. Click on the Update button to display the output.
11. After that we select the Select Attribute button, then select Humidity attribute and click OK.
12. Click on the Update button to display the output.
13. After that select the Select Attribute button and select Windy attribute and then click OK.
14. Increase the Jitter Size.
15. Click on the Update button to display the output.
16. After that we select the Select Attribute button, then select Play attribute and clock OK.
17. Click on the Update button to display the output.

Result :

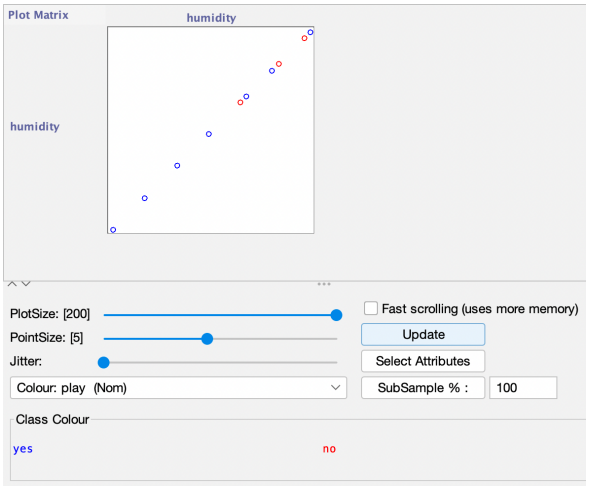
Plot Matrix



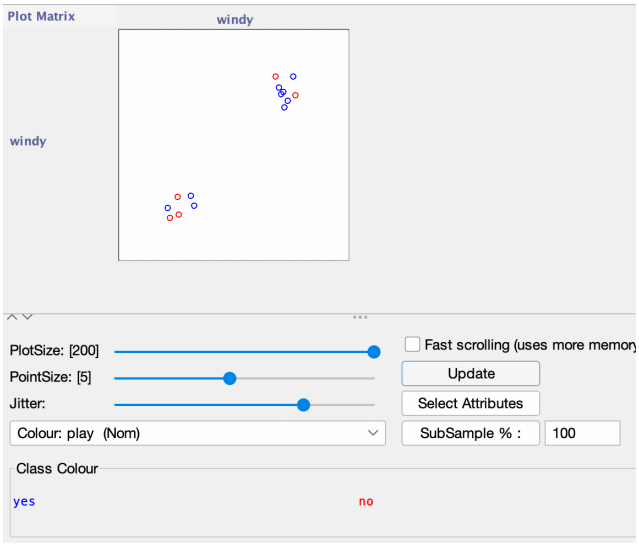
Attribute: temperature



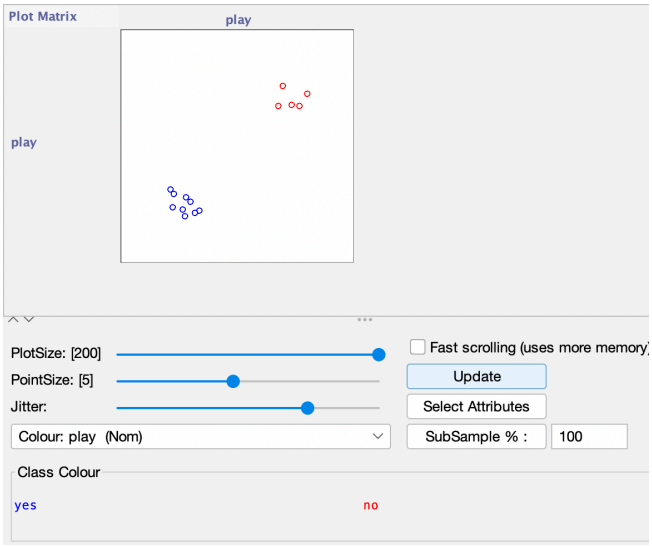
Attribute: Humidity



Attribute: Windy (jitter increased)



Attribute: play (jitter)



Banking Dataset

Creation of Banking Table:

17. Open a text editor
18. Type the following training data set with the help of Notepad for Weather Table.

```
@relation bank

@attribute cust {male,female}
@attribute
accno{0101,0102,0103,0104,0105,0106,0107,0108,0109,0110,0111,0112,0113
,0114,0115}
@attribute bankname {sbi,hdfc,sbh,ab,rbi}
@attribute location {hyd,jmd,antp,pdtr,kdp}
@attribute deposit {yes,no}

@data
male,0101,sbi,hyd,yes
female,0102,hdfc,jmd,no
male,0103,sbh,antp,yes
male,0104,ab,pdtr,yes
female,0105,sbi,jmd,no
male,0106,ab,hyd,yes
female,0107,rbi,jmd,yes
female,0108,hdfc,kdp,no
male,0109,sbh,kdp,yes
male,0110,ab,jmd,no
female,0111,rbi,kdp,yes
male,0112,sbi,jmd,yes
female,0113,rbi,antp,no
male,0114,hdfc,pdtr,yes
female,0115,sbh,pdtr,no
```

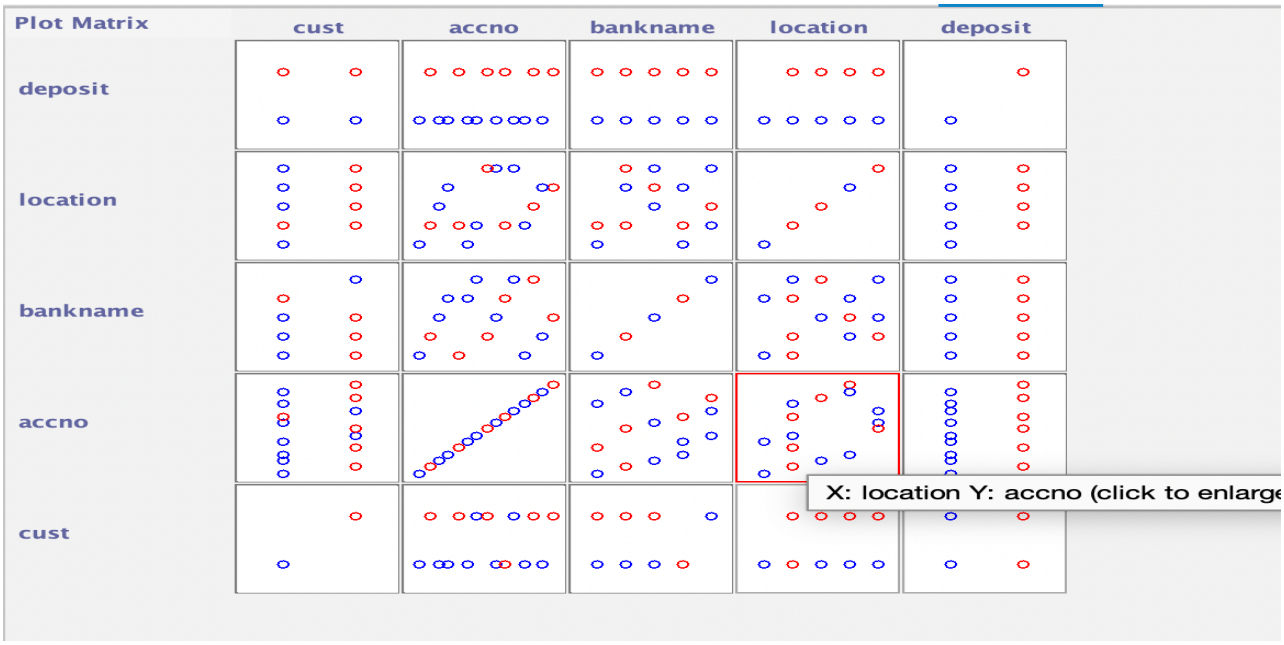
19. After that the file is saved with .arff file format
20. The file is saved in arff format. Now open the weka application.
21. Click on weka-3-4, then Weka dialog box is displayed on the screen.
22. In that dialog box there are four modes, click on explorer.
23. Explorer shows many options. In that click on ‘open file’ and select the arff file
24. Click on edit button which shows weather table on weka.

Visualisation Procedure:

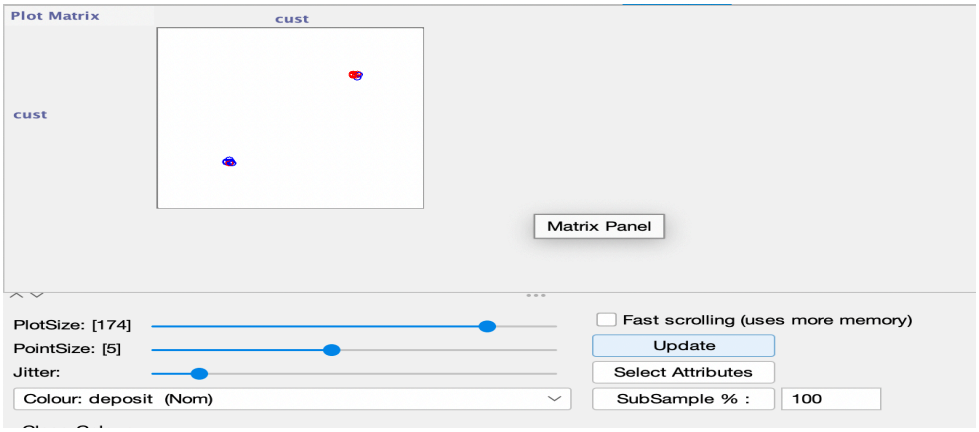
1. Click on the visualise tab on top bar of explorer window.
2. Open the visualise tab then Plot Matrix is displayed on the Screen.
3. After that we select the Select Attribute button, then select Outlook attribute and click OK.
4. Click on the Update button to display the output.
5. After that select the Select Attribute button and select Temperature attribute and then click OK.
6. Increase the Plot Size and Point Size.
7. Click on the Update button to display the output.
8. After that we select the Select Attribute button, then select Humidity attribute and click OK.
9. Click on the Update button to display the output.
10. After that select the Select Attribute button and select Windy attribute and then click OK.
11. Increase the Jitter Size.
12. Click on the Update button to display the output.
13. After that we select the Select Attribute button, then select Play attribute and clock OK.
14. Click on the Update button to display the output.

Result :

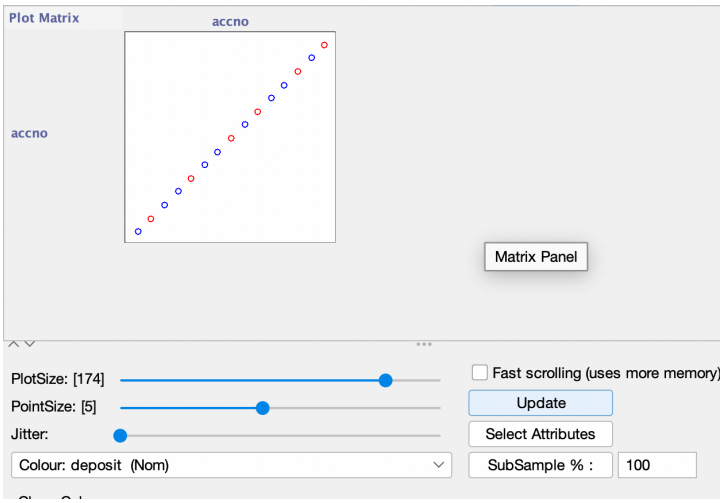
Plot Matrix



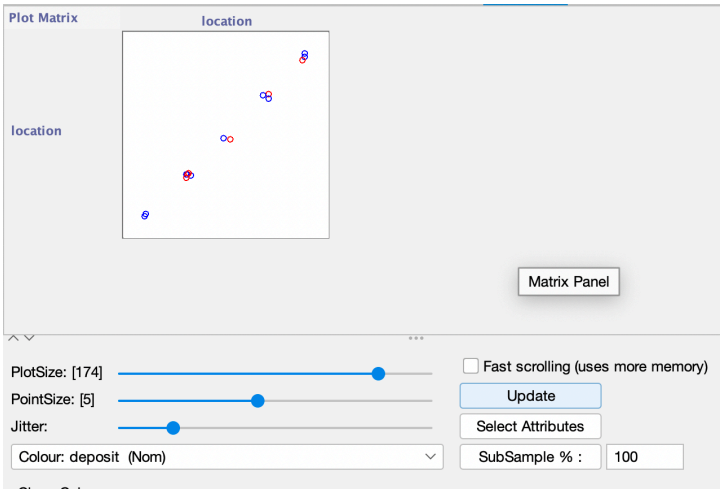
Attribute : Cust



Attribute : Account no



Attribute : Location



Conclusion :

We performed Visualisation on various attributes and changes in jitter and visualisation.

Practical – 7a

Aim Of Experiment :

Implement 1-R algorithm.

Description :

One Rule Algorithm

OneR, short for "One Rule", is a simple, yet accurate, classification algorithm that generates one rule for each predictor in the data, then selects the rule with the smallest total error as its "one rule".
To create a rule for a predictor, we construct a frequency table for each predictor against the target. It has been shown that OneR produces rules only slightly less accurate than state-of-the-art classification algorithms while producing rules that are simple for humans to interpret.

Procedure

```
# For each predictor,
     # For each value of that predictor, make a rule as follows;
           # Count how often each value of target (class) appears
           # Find the most frequent class
           # Make the rule assign that class to this value of the predictor
     # Calculate the total error of the rules of each predictor
# Choose the predictor with the smallest total error.
```

1R Python Implementation:

```
import numpy as np
import pandas as pd

df = pd.read_csv("data.csv")

df.head()
```

	Outlook	Temp	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes

```
df.shape
```

```
(14, 5)
```

```
class OneR:
  def __init__(self):
    self.rule = None
    self.accuracy = 0.0
    self.classes = []

  def fit(self, X_train,y_train):
    if len(X_train)!=len(y_train):
      raise Exception("Invalid training data")
    X_train = np.array(X_train).T
    y_train = np.array(y_train)
    training_samples = len(y_train)
    self.classes = list(np.unique(y_train))

    if len(self.classes)!=2:
      raise Exception("Algorithm is made for binary classification only")

    accuracy = 0.0
    for idx,column in enumerate(X_train):
      data = {k:{key:0 for key in self.classes} for k in np.unique(column)}
      for d,y in zip(column,y_train):
        data[d][y]+=1
      data = pd.DataFrame(data)
      correct_samples = 0
      rule = {}
      for col in data.columns:
        prediction = np.argmax(data[col])
        rule[col] = prediction
        correct_samples+= data[col][prediction]
      accuracy = correct_samples/training_samples
      if accuracy>self.accuracy:
        self.accuracy = accuracy
        self.rule = idx,rule
    return self.rule

  def predict(self,X_test):
    X_test = np.array(X_test)
    if not self.rule:
      raise Exception("Model not trained")
    idx,rule = self.rule
    y_pred = []
    for itr in X_test:
      y_pred.append(rule[itr[idx]])
    return y_pred

  def evaluate(self,X_test,y_test):
    if len(X_test)!=len(y_test):
      raise Exception("Invalid data")
    y_pred = self.predict(X_test)
    correct = 0
    total = len(y_test)
    for pred,actual in zip(y_pred,y_test):
      if self.classes[pred]==actual:
        correct+=1
    return correct/total
```

```
y = df['Play Golf']
X = df.drop('Play Golf',axis=1)

model = OneR()
model.fit(X,y)
```

```
(0, {'Overcast': 1, 'Rainy': 0, 'Sunny': 1})
```

```
model.accuracy
```

```
0.7142857142857143
```

1R Weka Implementation:

1. Open Weka and load the Weather.arff Dataset in the Explorer.
2. Open the Classify Tab on top .
3. Click on Choose Button
4. Under Rules Click on **OneR** Algorithm
5. Choose Test Option as training set itself.
6. Choose (Nom) Play as the target variable.
7. Click start.

Result

Classifier output

```
Scheme:       weka.classifiers.rules.OneR -B 6
Relation:     weather.symbolic
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

outlook:
        sunny    -> no
        overcast        -> yes
        rainy    -> yes
(10/14 instances correct)


Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances          10               71.4286 %
Incorrectly Classified Instances         4               28.5714 %
Kappa statistic                          0.3778
Mean absolute error                      0.2857
Root mean squared error                  0.5345
Relative absolute error                 61.5385 %
Root relative squared error            111.4773 %
Total Number of Instances               14
```

Conclusion:

We Implemented One Rule algorithm using Python and Weka to classify the Weather data.

Practical – 7b

Aim Of Experiment :

Implement Decision Tree.

Description :

Decision Tree

A decision tree is a classification and prediction tool having a tree-like structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A decision tree is a tree where each -

- Node - a feature(attribute)
- Branch - a decision(rule)
- Leaf - an outcome(categorical or continuous)

There are many algorithms there to build a decision tree. They are

1. **CART** (Classification and Regression Trees) — This makes use of Gini impurity as the metric.
2. **ID3** (Iterative Dichotomiser 3) — This uses entropy and information gain as metric.

Python Implementation

```
import pandas as pd

df = pd.read_csv("salaries.csv")
df.head()
```

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

```
inputs = df.drop('salary_more_then_100k',axis='columns')

target = df['salary_more_then_100k']
```

```
from sklearn.preprocessing import LabelEncoder
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()

inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])

inputs
```

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

```
inputs_n = inputs.drop(['company','job','degree'],axis='columns')

inputs_n
```

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

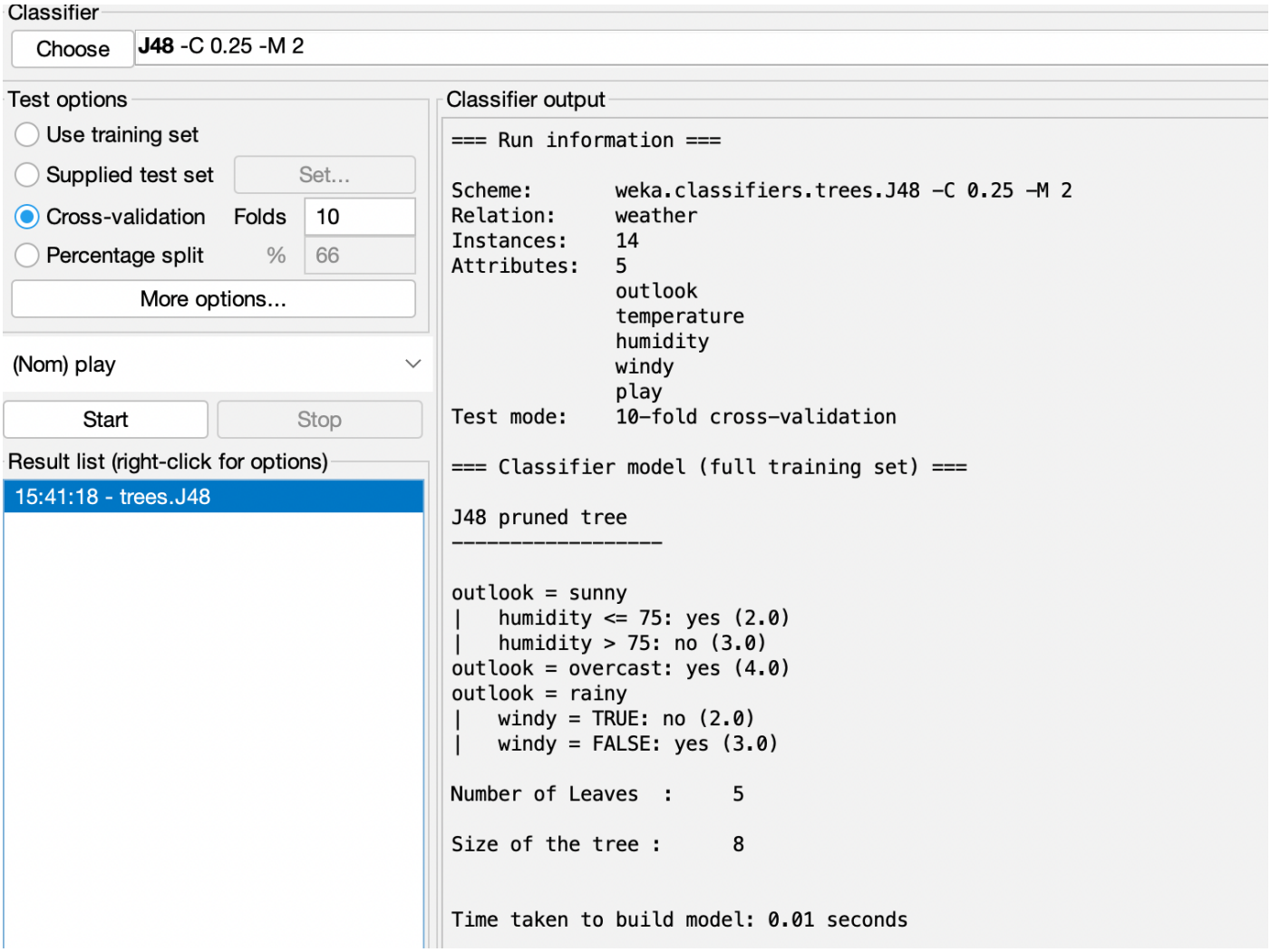
	target
0	0
1	0
2	1
3	1
4	0
5	1
6	0
7	0
8	0
9	1
10	1
11	1
12	1
13	1
14	1
15	1

Name: salary_more_then_100k, dtype: int64

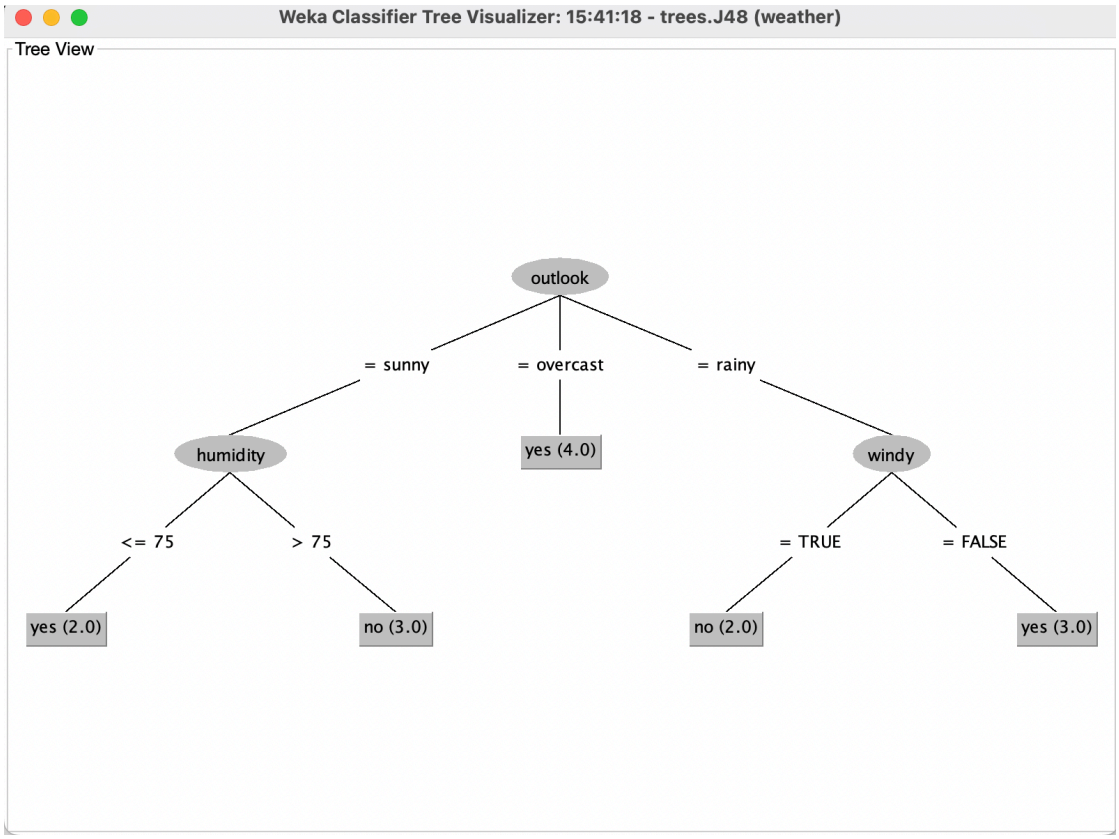
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs_n,target,test_size=0.3)

from sklearn import tree
model = tree.DecisionTreeClassifier()

model.fit(X_train,y_train)

DecisionTreeClassifier()

model.score(X_test,y_test)

0.8
```

Decision Tree Using WEKA:

1. Open Weka and load the Weather_numeric.csv Dataset in the Explorer.
2. Open the Classify Tab on top .
3. Click on Choose Button
4. Under Tree Click on **J48** Algorithm
5. Choose Test Option as Cross Validation -fold 10.
6. Choose (Nom) Play as the target variable.
7. Click start.



- Right Click on Result
- Visualise the Tree



Conclusion:

We Implemented Decision Tree Algorithm in python and in WEKA.

Practical – 8a

Aim Of Experiment :

Implement Naïve Bayes Classifier.

Description :

Naïve Bayes Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Bayes Theorem

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Python Implementation

```
import pandas as pd

df = pd.read_csv("titanic.csv")
df.head()
```

PassengerId			Name	Pclass	\
0	1		Braund, Mr. Owen Harris	3	
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...		1	
2	3		Heikkinen, Miss. Laina	3	
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)		1	
4	5		Allen, Mr. William Henry	3	

	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	\
0	male	22.0	1	0	A/5 21171	7.2500	NaN	S	
1	female	38.0	1	0	PC 17599	71.2833	C85	C	
2	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	
3	female	35.0	1	0	113803	53.1000	C123	S	
4	male	35.0	0	0	373450	8.0500	NaN	S	

	Survived
0	0
1	1
2	1
3	1
4	0

```
df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked'],axis='colum
ns',inplace=True)
df.head()
```

	Pclass	Sex	Age	Fare	Survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

```
inputs = df.drop('Survived',axis='columns')
target = df.Survived

#inputs.Sex = inputs.Sex.map({'male': 1, 'female': 2})

dummies = pd.get_dummies(inputs.Sex)
dummies.head(3)
```

	female	male
0	0	1
1	1	0
2	1	0

```
inputs = pd.concat([inputs,dummies],axis='columns')
inputs.head(3)
```

	Pclass	Sex	Age	Fare	female	male
0	3	male	22.0	7.2500	0	1
1	1	female	38.0	71.2833	1	0
2	3	female	26.0	7.9250	1	0

```
inputs.drop(['Sex','male'],axis='columns',inplace=True)
inputs.head(3)
```

	Pclass	Age	Fare	female
0	3	22.0	7.2500	0
1	1	38.0	71.2833	1
2	3	26.0	7.9250	1

```
inputs.columns[inputs.isna().any()]
```

```
Index(['Age'], dtype='object')
```

```
inputs.Age[:10]
```

```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
```

```
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs.head()
```

	Pclass	Age	Fare	female
0	3	22.0	7.2500	0
1	1	38.0	71.2833	1
2	3	26.0	7.9250	1
3	1	35.0	53.1000	1
4	3	35.0	8.0500	0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs,target,test_size=0.3)

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

model.fit(X_train,y_train)

GaussianNB(priors=None, var_smoothing=1e-09)

model.score(X_test,y_test)
```

```
0.7835820895522388
```

```
X_test[0:10]
```

	Pclass	Age	Fare	female	male
309	1	30.000000	56.9292	1	0
839	1	29.699118	29.7000	0	1
110	1	47.000000	52.0000	0	1
872	1	33.000000	5.0000	0	1
235	3	29.699118	7.5500	1	0
411	3	29.699118	6.8583	0	1
32	3	29.699118	7.7500	1	0
562	2	28.000000	13.5000	0	1
542	3	11.000000	31.2750	1	0
250	3	29.699118	7.2500	0	1

```
y_test[0:10]
```

```
309    1
839    1
110    0
872    0
235    0
411    0
32     1
562    0
542    0
250    0
Name: Survived, dtype: int64
```

```
model.predict(X_test[0:10])
```

```
array([1, 0, 0, 0, 1, 0, 1, 0, 1, 0], dtype=int64)
```

```
model.predict_proba(X_test[:10])
```

```
array([[0.00455992, 0.99544008],
       [0.91382024, 0.08617976],
       [0.88164575, 0.11835425],
       [0.92347978, 0.07652022],
       [0.09084386, 0.90915614],
       [0.99093305, 0.00906695],
       [0.09094857, 0.90905143],
       [0.97923786, 0.02076214],
       [0.0516967 , 0.9483033 ],
       [0.9909573 , 0.0090427 ]])
```

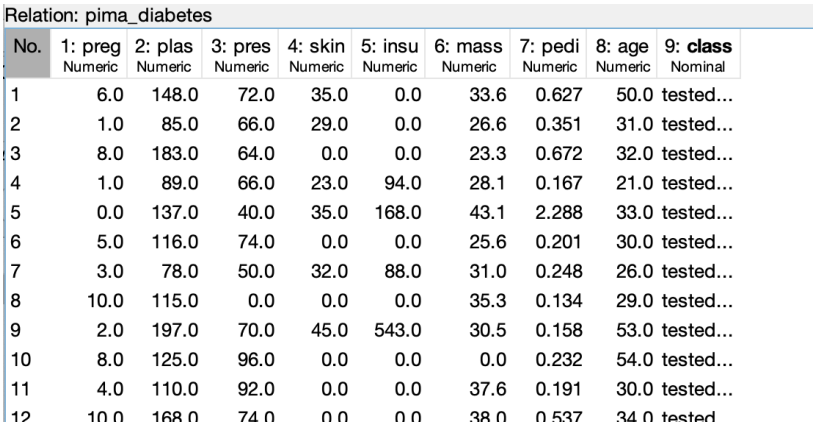
Calculate the score using cross validation

```
from sklearn.model_selection import cross_val_score
cross_val_score(GaussianNB(),X_train, y_train, cv=5)
```

```
array([0.75396825, 0.784     , 0.76612903, 0.82258065, 0.77419355])
```

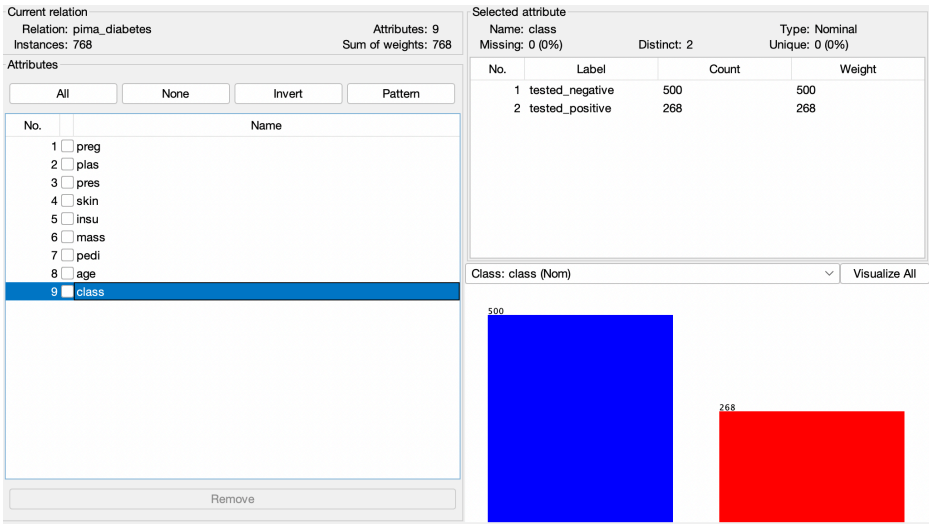
WEKA Implementation

1. Open Weka Application
2. Open the Explorer Window and Click on open file.
3. Under the data files Open the diabetes.arff file.
4. Click On edit to View the Dataset.

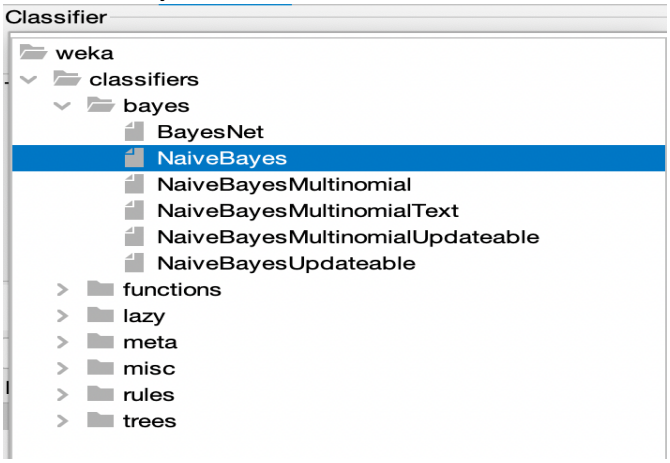


Relation: pima_diabetes

No.	1: preg Numeric	2: plas Numeric	3: pres Numeric	4: skin Numeric	5: insu Numeric	6: mass Numeric	7: pedi Numeric	8: age Numeric	9: **class** Nominal
1	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	tested...
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	tested...
3	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	tested...
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	tested...
5	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	tested...
6	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	tested...
7	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	tested...
8	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	tested...
9	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	tested...
10	8.0	125.0	96.0	0.0	0.0	0.0	0.232	54.0	tested...
11	4.0	110.0	92.0	0.0	0.0	37.6	0.191	30.0	tested...
12	10.0	168.0	74.0	0.0	0.0	38.0	0.537	34.0	tested



5. Switch to the classify tab.
6. Under the classify tab click on choose
7. Choose the bayes folder and select Naïve Bayes.



8. Choose 10 fold cross validation as test Option
9. Click on Start to build the model

Result :

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     pima_diabetes
Instances:    768
Attributes:   9
              preg
              plas
              pres
              skin
              insu
              mass
              pedi
              age
              class
Test mode:    10-fold cross-validation
```

```
=== Classifier model (full training set) ===

Naive Bayes Classifier

                        Class
Attribute       tested_negative tested_positive
                         (0.65)          (0.35)
================================================
preg
  mean                   3.4234          4.9795
  std. dev.              3.0166          3.6827
  weight sum                500             268
  precision                 1.0625          1.0625

plas
  mean                 109.9541        141.2581
  std. dev.             26.1114         31.8728
  weight sum                500             268
  precision                 1.4741          1.4741

pres
  mean                  68.1397         70.718
  std. dev.             17.9834         21.4094
  weight sum                500             268
  precision                 2.6522          2.6522

skin
  mean                  19.8356         22.2824
  std. dev.             14.8974         17.6992
  weight sum                500             268
  precision                   1.98            1.98
```

```
insu
  mean                  68.8507        100.2812
  std. dev.              98.828         138.4883
  weight sum                500             268
  precision                4.573           4.573

mass
  mean                  30.3009         35.1475
  std. dev.              7.6833          7.2537
  weight sum                500             268
  precision                0.2717          0.2717

pedi
  mean                   0.4297          0.5504
  std. dev.              0.2986          0.3715
  weight sum                500             268
  precision                0.0045          0.0045

age
  mean                  31.2494         37.0808
  std. dev.             11.6059         10.9146
  weight sum                500             268
  precision                 1.1765          1.1765


Time taken to build model: 0.01 seconds
```

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         586               76.3021 %
Incorrectly Classified Instances       182               23.6979 %
Kappa statistic                          0.4664
Mean absolute error                      0.2841
Root mean squared error                  0.4168
Relative absolute error                 62.5028 %
Root relative squared error             87.4349 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.844    0.388    0.802      0.844    0.823      0.468    0.819     0.892     tested_negative
                 0.612    0.156    0.678      0.612    0.643      0.468    0.819     0.671     tested_positive
Weighted Avg.    0.763    0.307    0.759      0.763    0.760      0.468    0.819     0.815

=== Confusion Matrix ===

   a   b   <-- classified as
 422  78 |   a = tested_negative
 104 164 |   b = tested_positive
```

Conclusion:

We implemented the Naïve Bayes Classifier using Python on titanic Dataset and Using Weka using Diabetes dataset.

Practical – 8b

Aim Of Experiment :

Implement Random Forest .

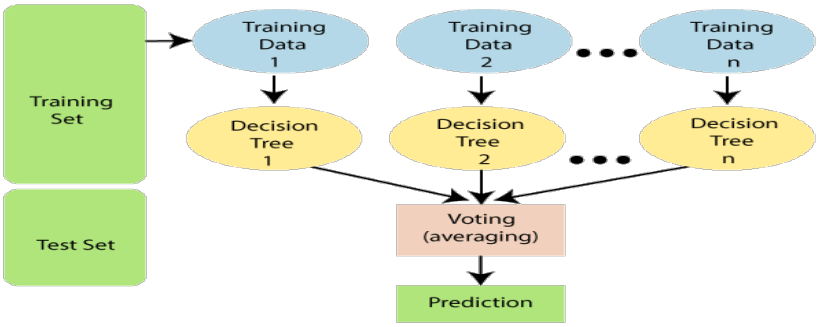
Description :

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.
Step-2: Build the decision trees associated with the selected data points (Subsets).
Step-3: Choose the number N for decision trees that you want to build.
Step-4: Repeat Step 1 & 2.
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Random Forest is capable of performing both Classification and Regression tasks.
It is capable of handling large datasets with high dimensionality.
It enhances the accuracy of the model and prevents the overfitting issue.

Python Implementation

Digits dataset from sklearn

```
import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()
```

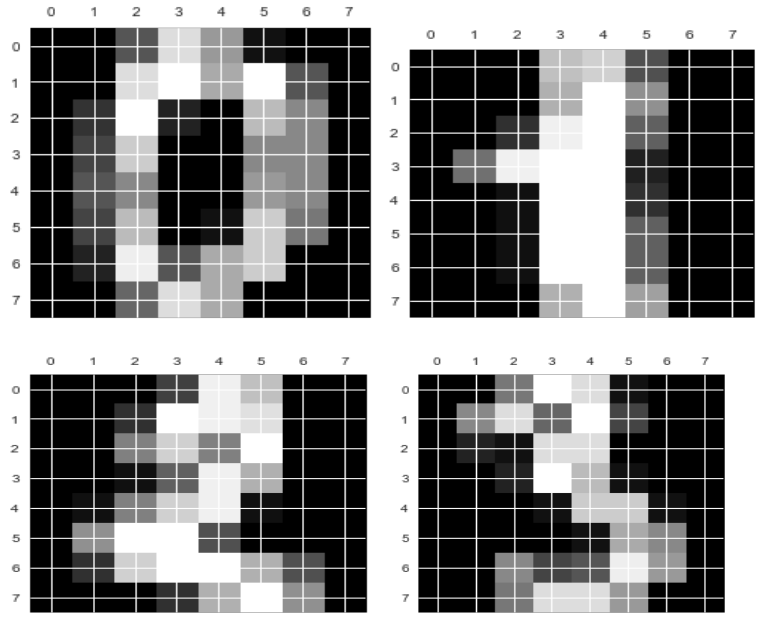
```
dir(digits)
```

```
['DESCR', 'data', 'images', 'target', 'target_names']
```

```
%matplotlib inline
import matplotlib.pyplot as plt
```

```
plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])
```

```
<matplotlib.figure.Figure at 0x2406e6b9a90>
```



```
df = pd.DataFrame(digits.data)
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	\
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

	57	58	59	60	61	62	63
0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	2.0	16.0	4.0	0.0	0.0

```
[5 rows x 64 columns]
```

```
df['target'] = digits.target
```

```
df[0:12]
```

```
      0    1     2     3     4     5     6    7    8    9  ...    55   56  \
0   0.0  0.0   5.0  13.0   9.0   1.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
1   0.0  0.0   0.0  12.0  13.0   5.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
2   0.0  0.0   0.0   4.0  15.0  12.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
3   0.0  0.0   7.0  15.0  13.0   1.0   0.0  0.0  0.0  8.0  ...   0.0  0.0
4   0.0  0.0   0.0   1.0  11.0   0.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
5   0.0  0.0  12.0  10.0   0.0   0.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
6   0.0  0.0   0.0  12.0  13.0   0.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
7   0.0  0.0   7.0   8.0  13.0  16.0  15.0  1.0  0.0  0.0  ...   0.0  0.0
8   0.0  0.0   9.0  14.0   8.0   1.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
9   0.0  0.0  11.0  12.0   0.0   0.0   0.0  0.0  0.0  2.0  ...   0.0  0.0
10  0.0  0.0   1.0   9.0  15.0  11.0   0.0  0.0  0.0  0.0  ...   0.0  0.0
11  0.0  0.0   0.0   0.0  14.0  13.0   1.0  0.0  0.0  0.0  ...   0.0  0.0

     57    58    59    60    61   62   63  target
0   0.0   6.0  13.0  10.0   0.0  0.0  0.0       0
1   0.0   0.0  11.0  16.0  10.0  0.0  0.0       1
2   0.0   0.0   3.0  11.0  16.0  9.0  0.0       2
3   0.0   7.0  13.0  13.0   9.0  0.0  0.0       3
4   0.0   0.0   2.0  16.0   4.0  0.0  0.0       4
5   0.0   9.0  16.0  16.0  10.0  0.0  0.0       5
6   0.0   1.0   9.0  15.0  11.0  3.0  0.0       6
7   0.0  13.0   5.0   0.0   0.0  0.0  0.0       7
8   0.0  11.0  16.0  15.0  11.0  1.0  0.0       8
9   0.0   9.0  12.0  13.0   3.0  0.0  0.0       9
10  0.0   1.0  10.0  13.0   3.0  0.0  0.0       0
11  0.0   0.0   1.0  13.0  16.0  1.0  0.0       1

[12 rows x 65 columns]
```

Train and the model and prediction

```
X = df.drop('target',axis='columns')
y = df.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=20, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)

model.score(X_test, y_test)

      0.9666666666666667

y_predicted = model.predict(X_test)
```

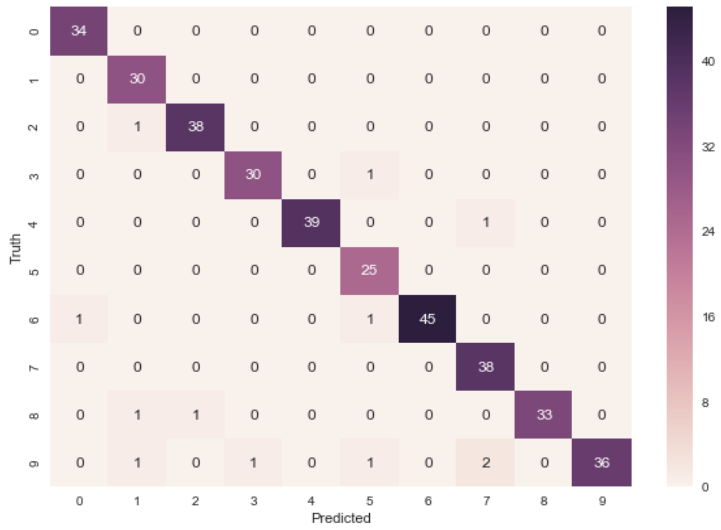
Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[34,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 30,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1, 38,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 30,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 39,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 25,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  1, 45,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 38,  0,  0],
       [ 0,  1,  1,  0,  0,  0,  0,  0, 33,  0],
       [ 0,  1,  0,  1,  0,  1,  0,  2,  0, 36]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
<matplotlib.text.Text at 0x2406e03e080>
```



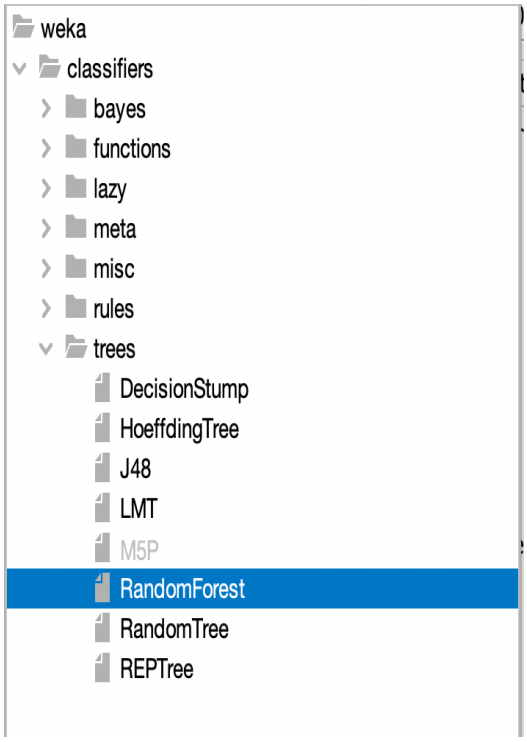
WEKA Implementation

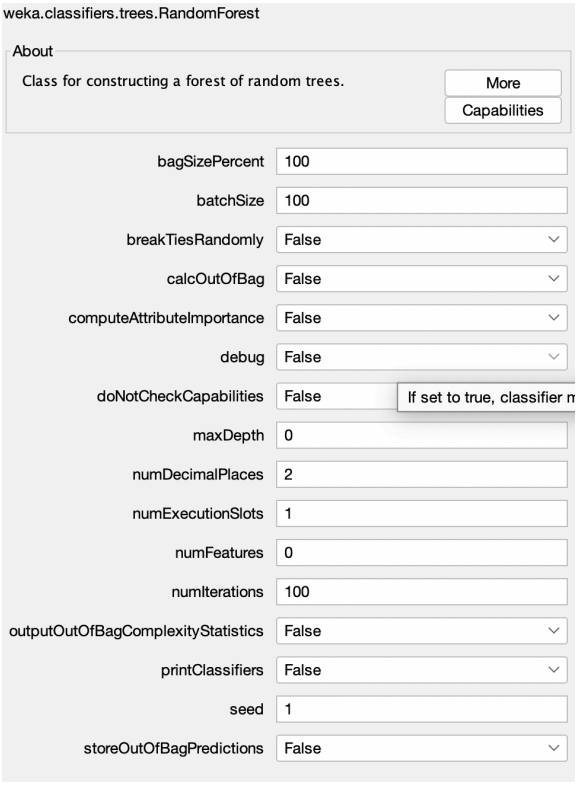
1. Open Weka Application
2. Open the Explorer Window and Click on open file.
3. Under the data files Open the Weather_numeric.arff file.
4. Click On edit to View the Dataset.

Relation: weather

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: **play** Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

5. Switch to the classify tab.
6. Under the classify tab click on choose
7. Choose the trees folder and select Random Forest.





8. Choose 10 fold cross validation as test Option
9. Click on Start to build the model

Result :

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
Test mode:    10-fold cross-validation
```

```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0.186
Mean absolute error                      0.4717
Root mean squared error                  0.5192
Relative absolute error                 99.0669 %
Root relative squared error            105.2408 %
Total Number of Instances               14

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

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

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.778    0.600    0.700      0.778    0.737      0.189    0.444     0.669     yes
               0.400    0.222    0.500      0.400    0.444      0.189    0.444     0.385     no
Weighted Avg.  0.643    0.465    0.629      0.643    0.632      0.189    0.444     0.567

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```

Conclusion:

We implemented the Naïve Bayes Classifier using Python on titanic Dataset and Using Weka using Diabetes dataset.

Practical – 9a

AIM OF EXPERIMENT :

TO IMPLEMENT APRIORI ALGORITHM

DESCRIPTION :

Apriori Algorithm:

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule leaning that analyses that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions.

To construct association rules between elements or items, the algorithm considers 3 important factors which are, **support, confidence** and **lift.**

$$support(I) = \frac{Number\ of\ transactions\ containing\ I}{Total\ number\ of\ transactions} \quad lift(I1 \rightarrow I2) = \frac{confidence(I1 \rightarrow I2)}{support(I2)}$$

$$confidence(I1 \rightarrow I2) = \frac{Number\ of\ transactions\ containing\ items\ I1\ and\ I2}{Total\ number\ of\ transactions\ containing\ I1}$$

Python Implementation

Dataset Load

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

Load require packages

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
```

Transform the data

```
te = TransactionEncoder()
te_try = te.fit(dataset).transform(dataset)
```

Generate Dataframe

```
df = pd.DataFrame(te_try, columns=te.columns_)
```

```
df
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

Model Training

```
from mlxtend.frequent_patterns import apriori

apriori(df,min_support=0.5)
```

	support	itemsets
0	0.8	(3)
1	1.0	(5)
2	0.6	(6)
3	0.6	(8)
4	0.6	(10)
5	0.8	(3, 5)
6	0.6	(8, 3)
7	0.6	(5, 6)
8	0.6	(8, 5)
9	0.6	(10, 5)
10	0.6	(8, 3, 5)

Model Training with Column Result return

```
apriori(df,min_support=0.5, use_colnames=True)
```

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Eggs, Onion)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Kidney Beans, Eggs, Onion)

Calculate the length of Itemset

```
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

Length is 2 and Support is > 0.8

```
frequent_itemsets[ (frequent_itemsets['length'] == 2) & (frequent_itemsets['support'] >= 0.8) ]
```

	support	itemsets	length
5	0.8	(Eggs, Kidney Beans)	2

```
frequent_itemsets[ frequent_itemsets['itemsets'] == {'Onion', 'Eggs'} ]
```

	support	itemsets	length
6	0.6	(Eggs, Onion)	2

```
oht_ary = te.fit(dataset).transform(dataset, sparse=True)
sparse_df = pd.SparseDataFrame(te_try, columns=te.columns_, default_fill_value=False)
sparse_df
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

Verbose return the number of iteration and itemset default size

```
apriori(sparse_df, min_support=0.6, use_colnames=True, verbose=1)
```

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Eggs, Onion)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Kidney Beans, Eggs, Onion)

WEKA implementation of Apriori:

1. Load the Supermarket Dataset into Explorer.
2. Open the association tab.
3. Choose Apriori algorithm for association
4. Click on start.

Associator output

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1
```

```
Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Conclusion :

We Implemented the apriori algorithm for association rule analyses on dataset .

Practical – 9b

AIM OF EXPERIMENT :

TO IMPLEMENT FP-Growth ALGORITHM

DESCRIPTION :

FP-Growth:

The FP-Growth Algorithm proposed by ***Han in***. This is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree).

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

Python Implementation:

Dataset Load

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

Load require packages

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
```

Transform the data

```
te = TransactionEncoder()
te_try = te.fit(dataset).transform(dataset)
```

Generate Dataframe

```
df = pd.DataFrame(te_try, columns=te.columns_)
```

```
df
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

Model Training

```
from mlxtend.frequent_patterns import fpgrowth
```

```
fpgrowth(df,min_support=0.5)
```

	support	itemsets
0	1.0	(5)
1	0.8	(3)
2	0.6	(10)
3	0.6	(8)
4	0.6	(6)
5	0.8	(3, 5)
6	0.6	(10, 5)
7	0.6	(8, 3)
8	0.6	(8, 5)
9	0.6	(8, 3, 5)
10	0.6	(5, 6)

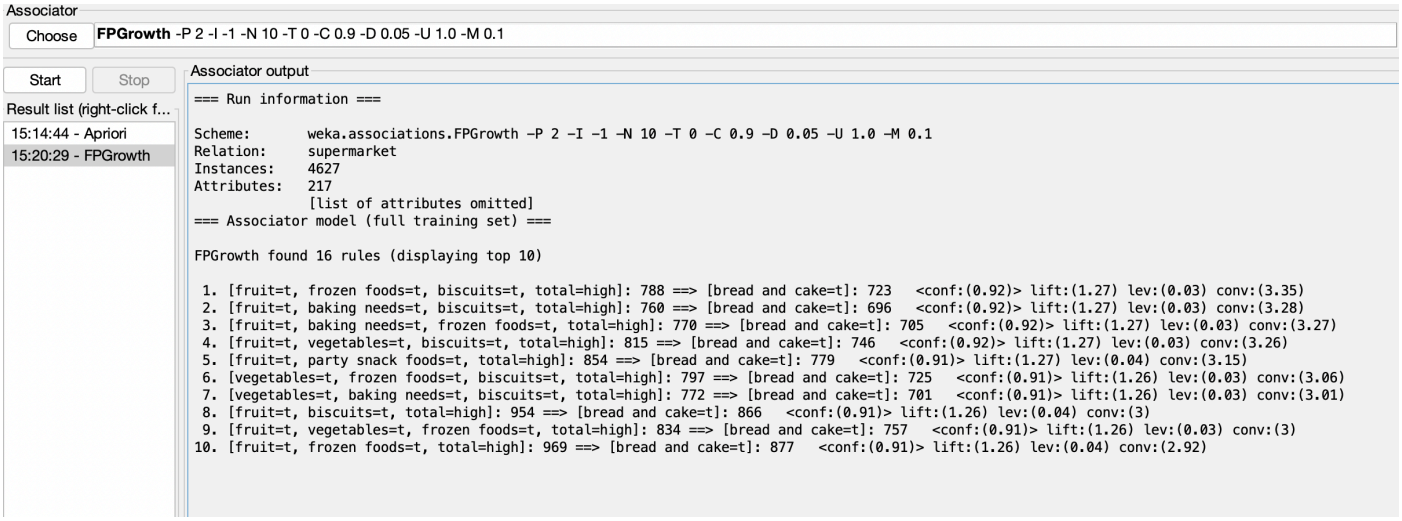
Model Training with Column Result return

```
apriori(df,min_support=0.5, use_colnames=True)
```

	support	itemsets
0	1.0	(Kidney Beans)
1	0.8	(Eggs)
2	0.6	(Yogurt)
3	0.6	(Onion)
4	0.6	(Milk)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Yogurt, Kidney Beans)
7	0.6	(Eggs, Onion)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Eggs, Onion, Kidney Beans)
10	0.6	(Milk, Kidney Beans)

Weka Implementation of FP Growth:

1. Load the Supermarket Dataset into Explorer.
2. Open the association tab.
3. Choose FP Growth algorithm for association
4. Click on start.



Conclusion:

We Implemented the FP Growth algorithm for association rule analyses on dataset In Python and Weka

Practical – 10a

AIM OF EXPERIMENT:

To Implement K-Means Clustering Algorithm.

Description:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled dataset into different clusters.

"It is an iterative algorithm that divides the unlabelled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties."
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training.

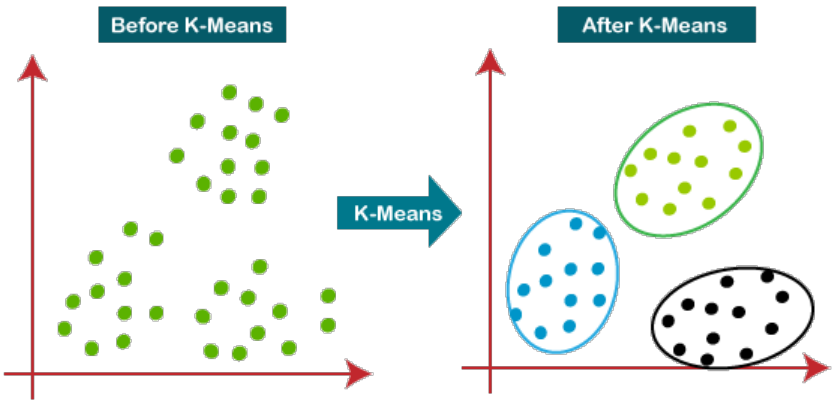
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K centre points or centroids by an iterative process.
- Assigns each data point to its closest k-centre. Those data points which are near to the particular k-centre, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.
Step-2: Select random K points or centroids. (It can be other from the input dataset).
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.



Python Implementation:

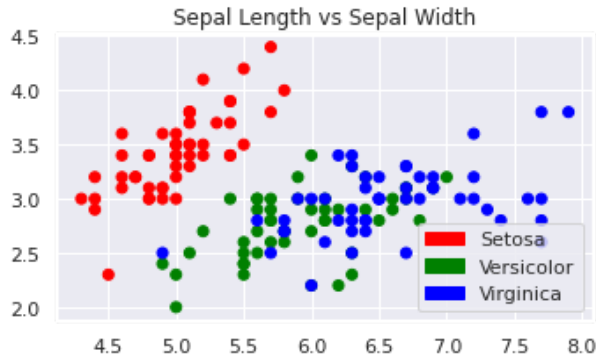
```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn.metrics as sm
%matplotlib inline

iris = datasets.load_iris()

x = pd.DataFrame(iris.data, columns=['Sepal Length', 'Sepal Width', 'Petal Length',
'Petal Width'])
y = pd.DataFrame(iris.target, columns=['Target'])

x.head()
```

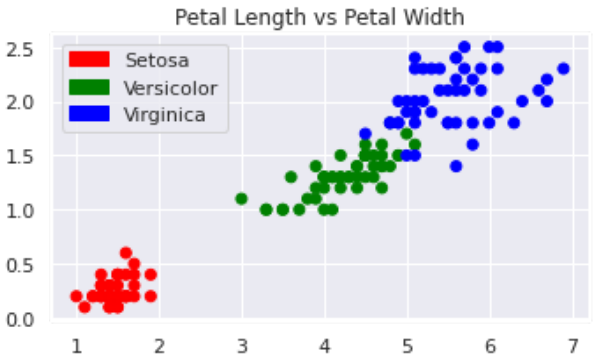
	Sepal Length	Sepal Width	Petal Length	Petal Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
y.head()
```

	Target
0	0
1	0
2	0
3	0
4	0

```
plt.figure(figsize=(12,3))
colors = np.array(['red', 'green', 'blue'])
iris_targets_legend = np.array(iris.target_names)
red_patch = mpatches.Patch(color='red', label='Setosa')
green_patch = mpatches.Patch(color='green', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')


plt.subplot(1, 2, 1)
plt.scatter(x['Sepal Length'], x['Sepal Width'], c=colors[y['Target']])
plt.title('Sepal Length vs Sepal Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1,2,2)
plt.scatter(x['Petal Length'], x['Petal Width'], c= colors[y['Target']])
plt.title('Petal Length vs Petal Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])
```






```
iris_k_mean_model = KMeans(n_clusters=3)
iris_k_mean_model.fit(x)
```

```
KMeans(n_clusters=3)
```

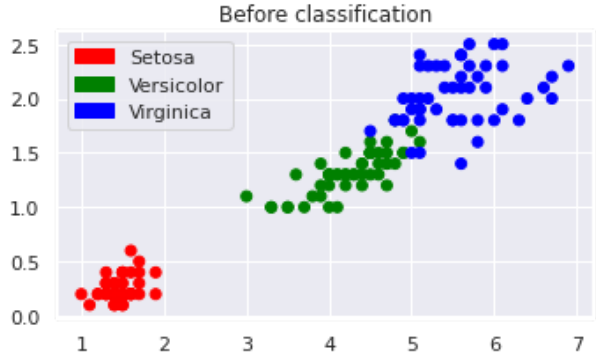
```
print (iris_k_mean_model.labels_)
```

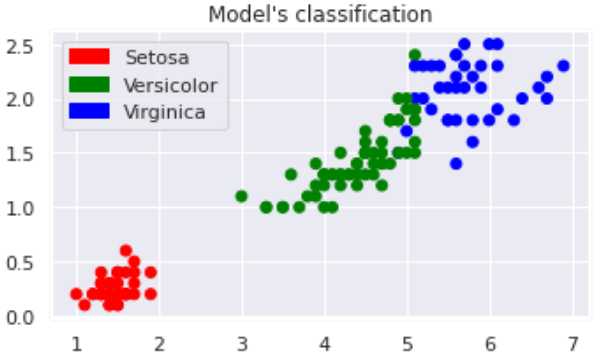
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

```
print (iris_k_mean_model.cluster_centers_)
```

```
[[5.9016129  2.7483871  4.39354839 1.43387097]
 [5.006      3.428      1.462      0.246     ]
 [6.85       3.07368421 5.74210526 2.07105263]]
```

```
plt.figure(figsize=(12,3))

colors = np.array(['red', 'green', 'blue'])

predictedY = np.choose(iris_k_mean_model.labels_, [1, 0, 2]).astype(np.int64)

plt.subplot(1, 2, 1)
plt.scatter(x['Petal Length'], x['Petal Width'], c=colors[y['Target']])
plt.title('Before classification')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1, 2, 2)
plt.scatter(x['Petal Length'], x['Petal Width'], c=colors[predictedY])
plt.title("Model's classification")
plt.legend(handles=[red_patch, green_patch, blue_patch])
```





```
sm.accuracy_score(predictedY, y['Target'])
```

```
0.8933333333333333
```

```
sm.confusion_matrix(predictedY, y['Target'])
```

```
array([[50,  0,  0],
       [ 0, 48, 14],
       [ 0,  2, 36]])
```

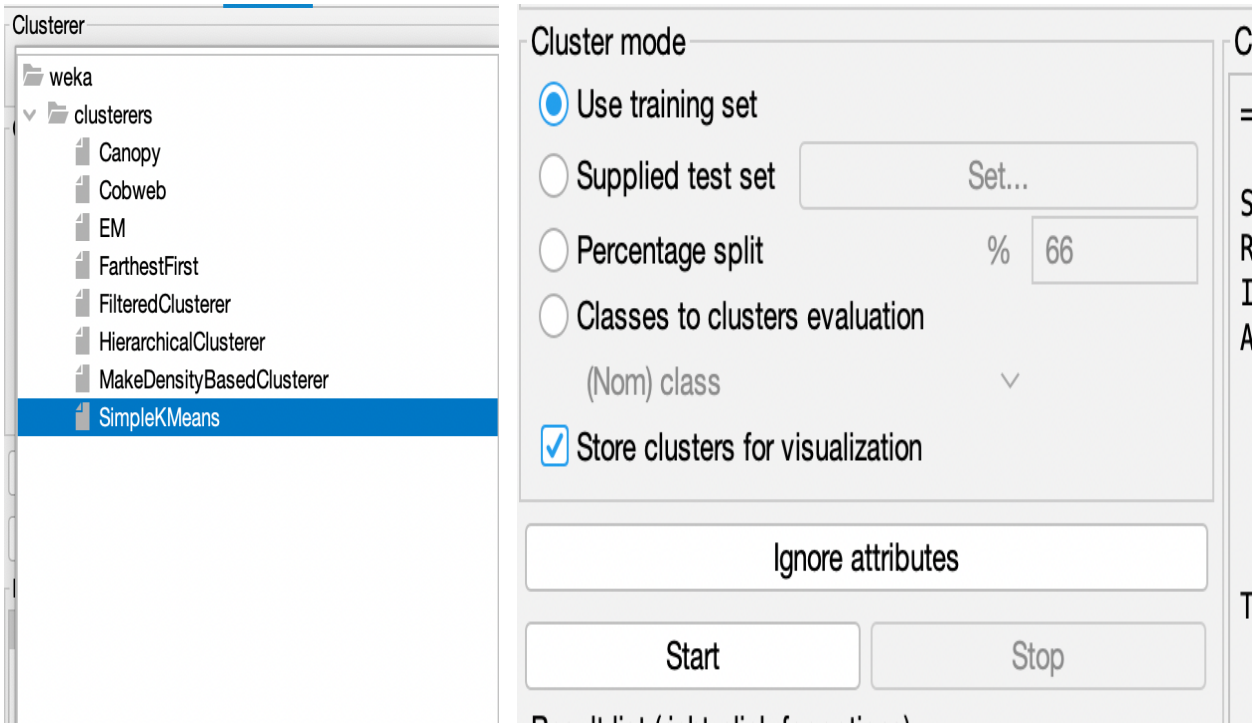
Weka Implementation of K-Means Clustering

1. Open Weka Application
2. Open the Explorer Window.
3. Click the Open file button and select Iris Dataset.
4. Click on Edit to View the dataset

Relation: iris

No.	1: sepallength Numeric	2: sepalwidth Numeric	3: petallength Numeric	4: petalwidth Numeric	5: **class** Nominal
1	5.1	3.5	1.4	0.2	Iris-se...
2	4.9	3.0	1.4	0.2	Iris-se...
3	4.7	3.2	1.3	0.2	Iris-se...
4	4.6	3.1	1.5	0.2	Iris-se...
5	5.0	3.6	1.4	0.2	Iris-se...
6	5.4	3.9	1.7	0.4	Iris-se...
7	4.6	3.4	1.4	0.3	Iris-se...
8	5.0	3.4	1.5	0.2	Iris-se...
9	4.4	2.9	1.4	0.2	Iris-se...
10	4.9	3.1	1.5	0.1	Iris-se...
11	5.4	3.7	1.5	0.2	Iris-se...
12	4.8	3.4	1.6	0.2	Iris-se...
13	4.8	3.0	1.4	0.1	Iris-se...
14	4.3	3.0	1.1	0.1	Iris-se...
15	5.8	4.0	1.2	0.2	Iris-se...
16	5.7	4.4	1.5	0.4	Iris-se...
17	5.4	3.9	1.3	0.4	Iris-se...
18	5.1	3.5	1.4	0.3	Iris-se...
19	5.7	3.8	1.7	0.3	Iris-se...
20	5.1	3.8	1.5	0.3	Iris-se...
21	5.4	3.4	1.7	0.2	Iris-se...
22	5.1	3.7	1.5	0.4	Iris-se...
23	4.6	3.6	1.0	0.2	Iris-se...
24	5.1	3.3	1.7	0.5	Iris-se...

5. Now Click on the Cluster Tab on the top bar.
6. Click the choose button.
7. Select the SimpleKmeans Clustering.



8. Select the Use training set Test Option.
9. Check Box to store for visualization. Click on Start.

Result :

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode
```

```
Final cluster centroids:
                                        Cluster#
Attribute               Full Data             0              1
                          (150.0)         (100.0)         (50.0)
=================================================================
sepallength                5.8433           6.262           5.006
sepalwidth                  3.054           2.872           3.418
petallength                3.7587           4.906           1.464
petalwidth                 1.1987           1.676           0.244
class                 Iris-setosa Iris-versicolor     Iris-setosa



Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)
```

Conclusion:

We Implemented the K-Means Clustering Algorithm using python and also using Weka.

Practical – 10b

Aim of Experiment:

Calculating Information Gain Measure.

Description:

Information gain can be defined as the amount of information gained about a random variable or signal from observing another random variable. It can be considered as the difference between the entropy of parent node and weighted average entropy of child nodes.

$$IG(S, A) = H(S) - H(S, A)$$

Alternatively,

$$IG(S, A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

In machine learning, **entropy** is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

Information gain (IG) measures how much "information" a feature gives us about the class. – Features that perfectly partition should give maximal information. – Unrelated features should give no information. It measures the reduction in entropy. CfsSubsetEval aims to identify a subset of attributes that are highly correlated with the target while not being strongly correlated with one another. It searches through the space of possible attribute subsets for the "best" one using the BestFirst search method by default, although other methods can be chosen. To use the wrapper method rather than a filter method, such as CfsSubsetEval, first select WrapperSubsetEval and then configure it by choosing a learning algorithm to apply and setting the number of crossvalidation folds to use when evaluating it on each attribute subset.

Python Implementation:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
play_data = pd.read_csv('https://raw.githubusercontent.com/edyoda/data-science-complete-tutorial/master/Data/tennis.csv.txt')
play_data
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

```
play_data.play.value_counts()
```

```
yes    9
no     5
Name: play, dtype: int64
```

```
Entropy_Play = -(9/14)*np.log2(9/14) -(5/14)*np.log2(5/14)
Entropy_Play
```

```
0.9402859586706309
```

```
play_data[play_data.outlook == 'sunny']
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
10	sunny	mild	normal	True	yes

```
# Entropy(Play|Outlook=Sunny)
Entropy_Play_Outlook_Sunny =-(3/5)*np.log2(3/5) -(2/5)*np.log2(2/5)
Entropy_Play_Outlook_Sunny
```

```
0.9709505944546686
```

```
play_data[play_data.outlook == 'overcast']
# Entropy(Play|Outlook=overcast)
```

```
# Since, it's a homogenous data entropy will be 0
```

	outlook	temp	humidity	windy	play
2	overcast	hot	high	False	yes
6	overcast	cool	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes

```
play_data[play_data.outlook == 'rainy']
```

	outlook	temp	humidity	windy	play
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
9	rainy	mild	normal	False	yes
13	rainy	mild	high	True	no

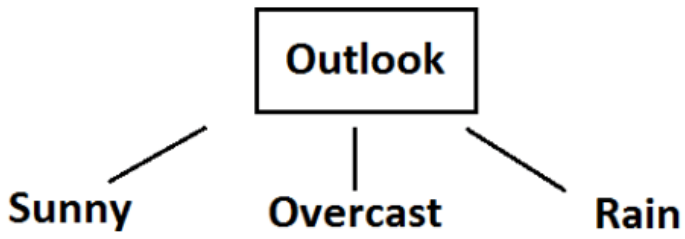
```
# Entropy(Play|Outlook=rainy)
Entropy_Play_Outlook_Rain = -(2/5)*np.log2(2/5) - (3/5)*np.log2(3/5)
Entropy_Play_Outlook_Rain
```

```
0.9709505944546686
```

```
"""Gain on splitting by attribute outlook"""
Entropy_Play - (5/14)*Entropy_Play_Outlook_Sunny - (4/14)*0 - (5/14) *
Entropy_Play_Outlook_Rain
```

```
0.2467498197744391
```

```
"""Other gains

Gain(Play, Temperature) - 0.029

Gain(Play, Humidity) - 0.151

Gain(Play, Wind) - 0.048

Conclusion - Outlook is winner & thus becomes root of the tree
```



```
Time to find the next splitting criteria
"""

play_data[play_data.outlook == 'overcast']
```

	outlook	temp	humidity	windy	play
2	overcast	hot	high	False	yes
6	overcast	cool	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes

```
"""Conclusion - If outlook is overcast, play is true

Let's find the next splitting feature
"""
play_data[play_data.outlook == 'sunny']
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
10	sunny	mild	normal	True	yes

```
# Entropy(Play_Sunny|)
Entropy_Play_Outlook_Sunny =-(3/5)*np.log2(3/5) -(2/5)*np.log2(2/5)
Entropy_Play_Outlook_Sunny
```

```
0.9709505944546686
```

```
"""Information Gain for humidity"""

#Entropy for attribute high = 0, also entropy for attribute normal = 0
Entropy_Play_Outlook_Sunny - (3/5)*0 - (2/5)*0
```

```
0.9709505944546686
```

```
"""Information Gain for windy

False -> 3 -> [1+ 2-]

True -> 2 -> [1+ 1-]
"""

Entropy_Wind_False = -(1/3)*np.log2(1/3) - (2/3)*np.log2(2/3)
Entropy_Wind_False
```

```
0.9182958340544896
```

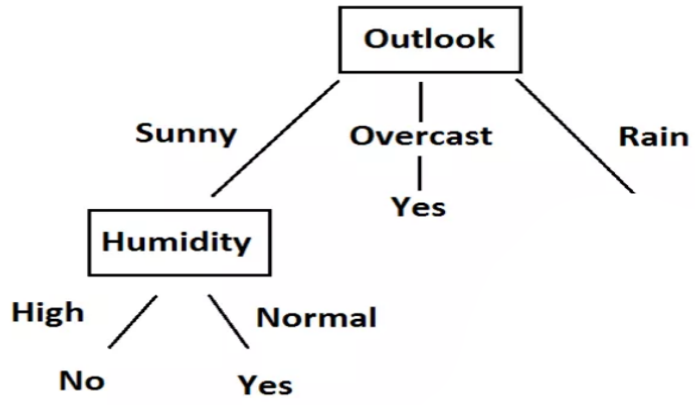
```
Entropy_Play_Outlook_Sunny - (3/5)* Entropy_Wind_False  - (2/5)*1
```

```
0.01997309402197489
```

```
"""Information Gain for temperature'

hot -> 2 -> [2- 0+]

mild -> 2 -> [1+ 1-]

cool -> 1 -> [1+ 0-]
"""

Entropy_Play_Outlook_Sunny - (2/5)*0 - (1/5)*0 - (2/5)* 1
```

```
0.5709505944546686
```

```
"""Conclusion : Humidity is the best choice on sunny branch
```



Weka Implementation :

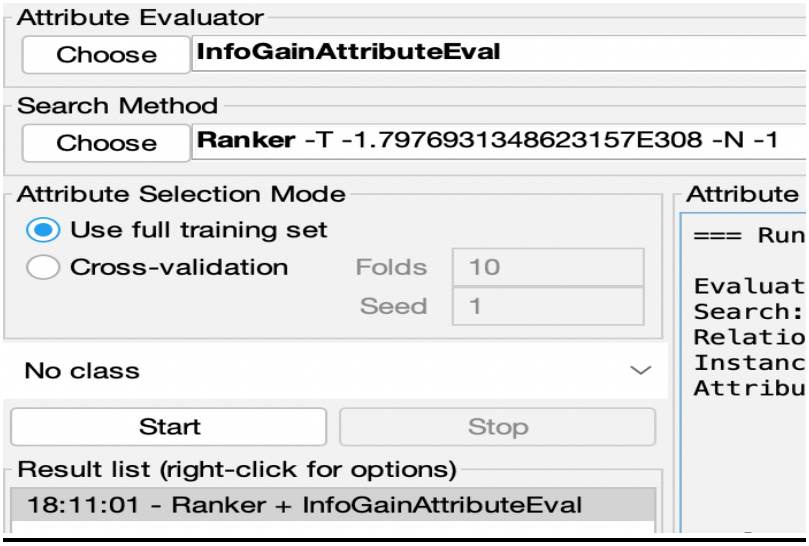
1. Open Weka Application
2. Open the Explorer Window
3. Click on Open File button.
4. Choose the Weather.arff file or any other dataset.
5. Click on edit to see the dataset table

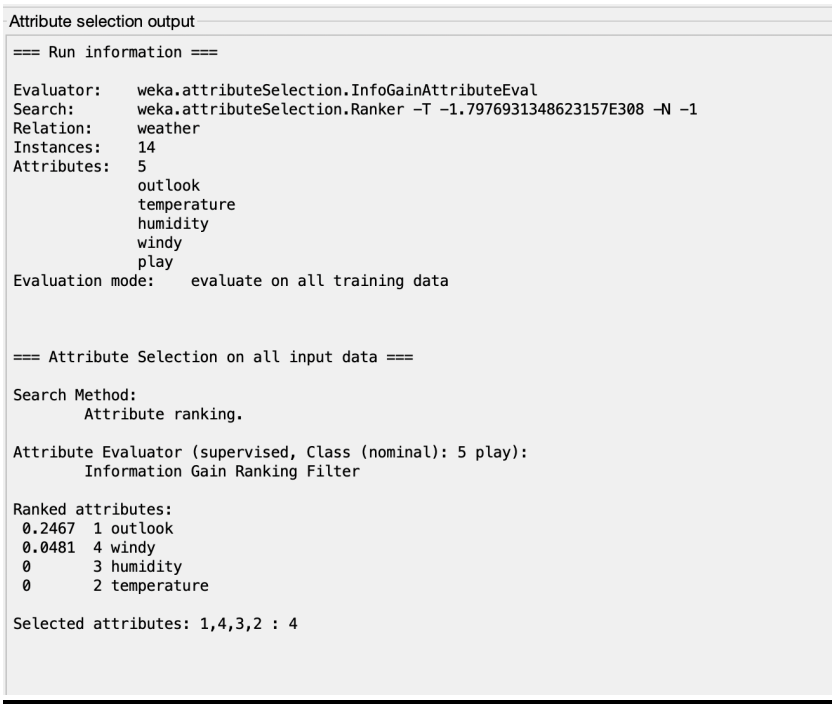
Relation: weather

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: **play** Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

6. Click on the select attribute tab to open it.
7. Choose under attribute evaluator InfoGainAttributeEval
8. Under search method choose Ranker method
9. Select full training set under attribute selection mode.
10. Click on start.

Result:





Conclusion:

We implemented Attribute Selection Using Information Gain using python and Weka.