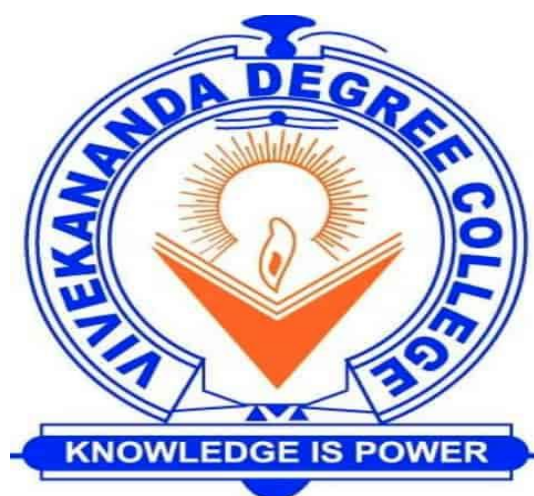


MACHINE LEARNING PROGRAMMING

B.Sc. IV-Semester (Data Science)



1. Write a Python program using Scikit-learn to split the iris dataset into 70% train data and 30% test data. Out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records. Print both datasets.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)
print("\n70% train data:")
print(X_train)
print(y_train)
print("\n30% test data:")
print(X_test)
print(y_test)
```

OUTPUT:

```
70% train data:
Squeezed text (120 lines).
[2 1 1 2 1 1 2 2 1 0 2 1 1 1 2 2 2 1 2 2 1 0 0 2 1 0 1 0 1 1 0 1 0 2 0 2 2
0 1 1 0 0 2 1 0 2 1 0 1 1 0 0 2 2 1 0 0 0 2 1 2 0 1 1 0 0 1 0 2 0 0 2 2 1
0 0 0 2 2 1 2 1 2 1 0 2 1 0 1 1 2 2 2 0 0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 2 2
2 2 2 0 0 1 0 0 2]

30% test data:
[[5.2 3.5 1.5 0.2]
 [6.4 3.2 4.5 1.5]
 [6.8 2.8 4.8 1.4]
 [5.9 3. 4.2 1.5]
 [5.2 2.7 3.9 1.4]
 [6.5 3. 5.5 1.8]
 [7. 3.2 4.7 1.4]
 [6.9 3.2 5.7 2.3]
 [5.8 2.6 4. 1.2]
 [5.8 2.7 4.1 1. ]
 [7.1 3. 5.9 2.1]
 [4.6 3.6 1. 0.2]
 [6.4 3.2 5.3 2.3]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [6.7 2.5 5.8 1.8]
 [5. 3.6 1.4 0.2]
 [6.3 2.9 5.6 1.8]
 [5.5 3.5 1.3 0.2]
 [7.7 2.6 6.9 2.3]
 [6.8 3.2 5.9 2.3]
 [6.9 3.1 5.4 2.1]
 [5.7 2.9 4.2 1.3]
 [6.7 3.3 5.7 2.5]
 [7.2 3.6 6.1 2.5]
 [5.9 3.2 4.8 1.8]
 [6.2 2.8 4.8 1.8]
 [4.7 3.2 1.6 0.2]
 [7.2 3.2 6. 1.8]
 [6.1 2.8 4.7 1.2]]
[0 1 1 1 1 2 1 2 1 1 2 0 2 0 0 2 0 2 0 2 2 2 1 2 2 1 2 0 2 1]
```


2. Write a python program to use sklearn's Decision Tree Classifier to build a decision tree for the sklearn's datasets. Implement functions to find the importance of a split (entropy, information gain, gini measure).

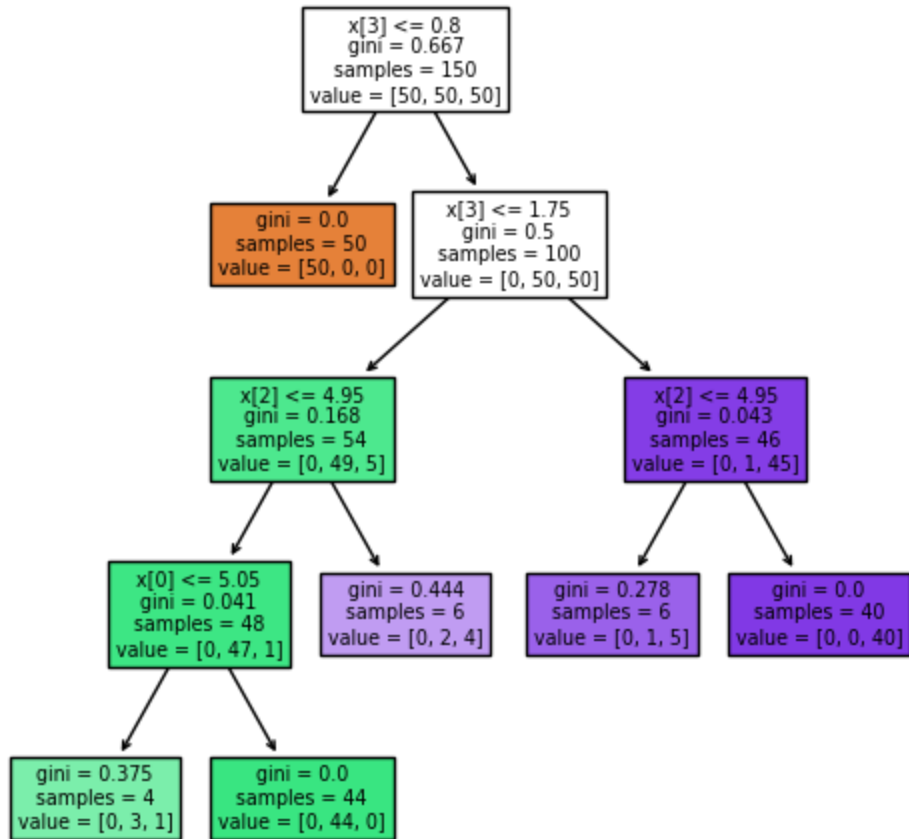
Program:

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
iris = load_iris()
fig, ax = plt.subplots(figsize=(6, 6))
clf = DecisionTreeClassifier(criterion='gini',
max_depth=4,min_samples_leaf=4)
clf.fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.title("Decision tree trained on gini")
plt.show()
fig, ax = plt.subplots(figsize=(6, 6))
clf = DecisionTreeClassifier(criterion='entropy',
max_depth=4,min_samples_leaf=4)
clf.fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.title("Decision tree trained on entropy")
plt.show()
value = [[1.2,3.5,1.9,2.9]]
pred = clf.predict(value)
if pred[0] == 0:
    print('Iris-Setosa')
elif pred[0] == 1:
    print('Iris-Versicolour')
else:
    print('Iris-Virginica')
```

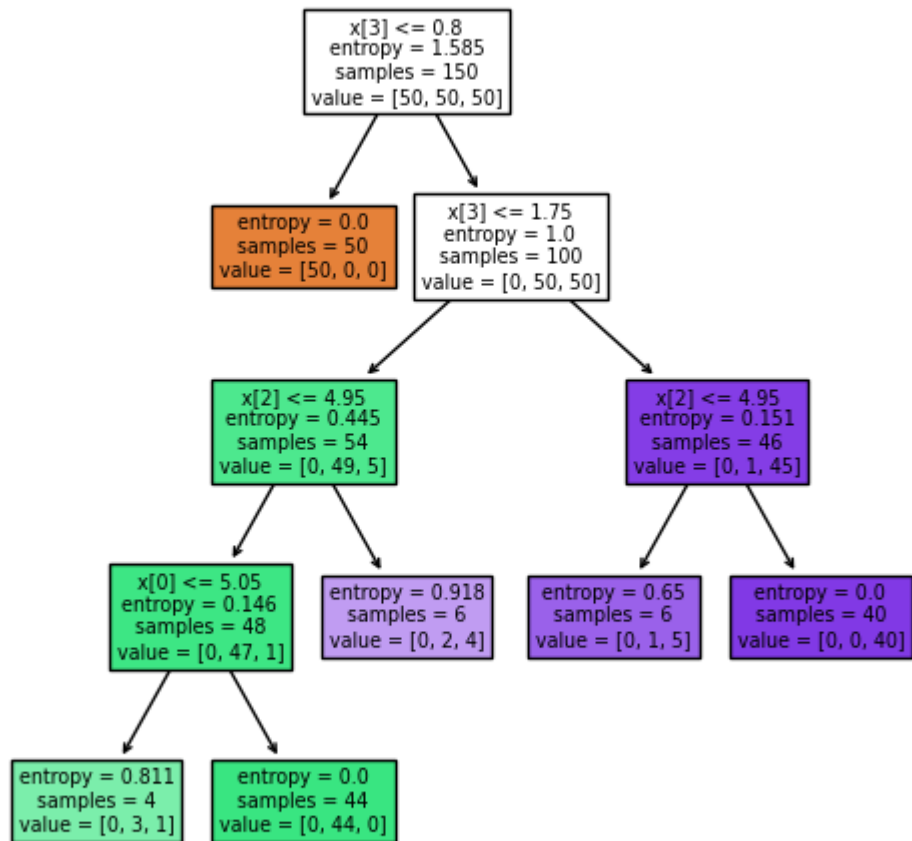
OUTPUT:

>>>Iris-Virginica

Decision tree trained on gini



Decision tree trained on entropy



3. Write a python program to implement your own version of K-mean algorithm. Then apply it to different datasets and evaluate the performance.

Program:

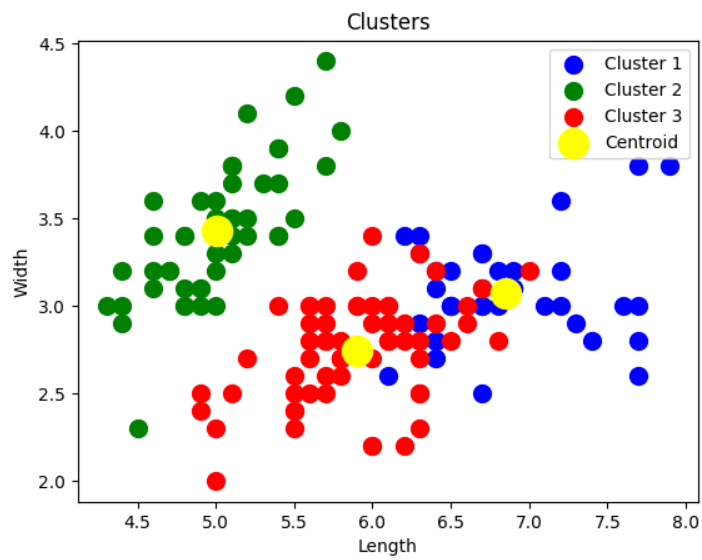
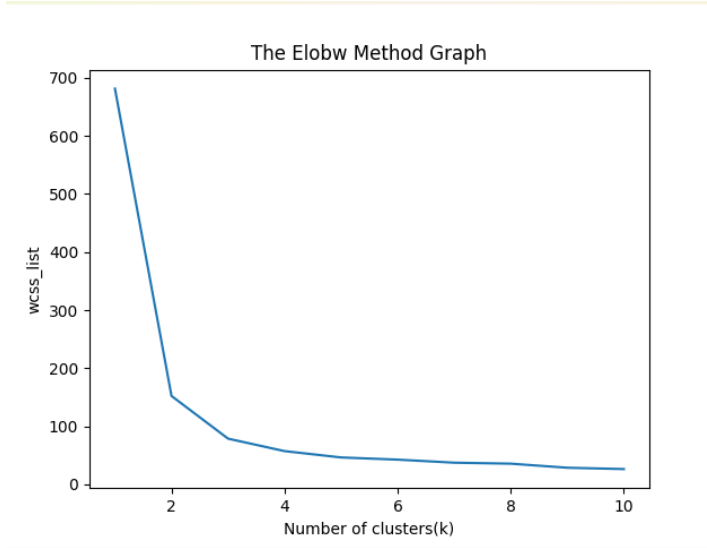
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
wcss_list= []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i,n_init='auto')
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elbow Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()

kmeans = KMeans(n_clusters=3,n_init='auto')
y_predict= kmeans.fit_predict(x)
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue',
label = 'Cluster 1')
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green',
label = 'Cluster 2')
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red',
label = 'Cluster 3')
mtp.scatter(kmeans.cluster_centers_[0, 0], kmeans.cluster_centers_[0, 1], s
= 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters')
mtp.xlabel('Length')
mtp.ylabel('Width')
mtp.legend()
mtp.show()

test = x[0::15]
predict = kmeans.predict(test)
print(test)
```

OUTPUT:

```
===== RESTART: C:\Users'\n[[5.1 3.5 1.4 0.2]\n [5.7 4.4 1.5 0.4]\n [4.8 3.1 1.6 0.2]\n [4.8 3.  1.4 0.3]\n [5.  2.  3.5 1. ]\n [6.6 3.  4.4 1.4]\n [5.5 2.6 4.4 1.2]\n [7.6 3.  6.6 2.1]\n [6.9 3.2 5.7 2.3]\n [7.7 3.  6.1 2.3]]\n|
```



4. Design a perceptron classifier to classify handwritten numerical digits (0-9). Implement using scikit or weka.

Program:

#modelpreperation

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
X_train[0]
print('shape i.e the dimension of the images', X_train[0].shape)
X_train = X_train / 255
X_test = X_test / 255
print(X_train[0])
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
print(y_train[15])
num_classes = 10
def cnn():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    model.add(MaxPooling2D())
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model
model = cnn()
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=200,
verbose=2)
model.predict(X_test[:1])
def expected_result(y_data):
```

```

for i in range(10):
    if(y_data[i]):
        print(i)
        break
def predicted_result(pred_Xtrain, index):
    ans = pred_Xtrain[index].argsort()[-8:][:-1] #sorting in descending
    print(ans[0])
pred_Xtest = model.predict(X_test)
index = 20
predicted_result(pred_Xtest, index)
expected_result(y_test[index])
model.save('digitrecognition.h5')
test1 = X_test[0]
test1 = test1.reshape(1, 28, 28, 1).astype('float32')
model.predict(test1)

```

OUTPUT:

```

shape i.e the dimension of the images (28, 28, 1)
Squeezed text (811 lines).
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
Epoch 1/5
300/300 - 15s - loss: 0.2536 - accuracy: 0.9271 - val_loss: 0.0803 - val_accuracy: 0.9761 - 15s/epoch - 51ms/step
Epoch 2/5
300/300 - 14s - loss: 0.0764 - accuracy: 0.9769 - val_loss: 0.0537 - val_accuracy: 0.9821 - 14s/epoch - 48ms/step
Epoch 3/5
300/300 - 15s - loss: 0.0525 - accuracy: 0.9844 - val_loss: 0.0424 - val_accuracy: 0.9857 - 15s/epoch - 48ms/step
Epoch 4/5
300/300 - 16s - loss: 0.0443 - accuracy: 0.9865 - val_loss: 0.0369 - val_accuracy: 0.9867 - 16s/epoch - 52ms/step
Epoch 5/5
300/300 - 15s - loss: 0.0346 - accuracy: 0.9888 - val_loss: 0.0331 - val_accuracy: 0.9884 - 15s/epoch - 51ms/step

```

#GUI implementaion

```
from tkinter import *
import tkinter.font as tkFont
import win32gui
from PIL import ImageGrab, Image
import numpy as np
from keras.models import load_model

class application(Frame):
    def __init__(self, master):
        super().__init__(master)
        self.fontStyle = tkFont.Font(family="Lucida Grande", size=20)
        self.master = master
        self.pack()
        self.createWidget()
    def createWidget(self):
        self.canvas = Canvas(self, width=224, height=224, bg='black')
        self.canvas.pack(expand=YES, fill=BOTH)
        self.canvas.bind('<B1-Motion>', self.activate_paint)
    def activate_paint(self, event):
        global lastx, lasty
        self.canvas.bind('<B1-Motion>', self.paint)
        lastx, lasty = event.x, event.y
    def paint(self, event):
        global lastx, lasty
        x, y = event.x, event.y
        self.canvas.create_line((lastx, lasty, x, y), width=12, fill='white')
        lastx, lasty = x, y
    def clearCanvas(self):
        self.canvas.delete("all")
        answer.configure(text='Answer Goes Here', font=self.fontStyle)
        self.canvas.bind('<B1-Motion>', self.activate_paint)
    def predicted_result(self, data):
        ans = data.argsort()[-8:][::-1] #sorting in descending
        return ans
    def predictDigit(self):
        HWND = self.canvas.winfo_id() # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas
        im = ImageGrab.grab(rect) # get image of the current location
        im.save('file.png')
        img = Image.open('file.png').convert('L')
```

```

img = img.resize((28,28), Image.ANTIALIAS)
img.save('resized.png')
data = np.array(img)
data = data/255.0 # for range b/w 0-1
data = data.reshape(1, 28, 28, 1).astype('float32')

model = load_model('digitrecognition.h5')

result = model.predict(data)
ans = self.predicted_result(result)

answer.configure(text='Predicted Digit: '+str(ans[0][-1]), font=self.fontStyle)
if __name__ == '__main__':
    root = Tk()
    root.geometry('300x400')
    app=application(root)
    clear = Button(root, text='Clear', command=app.clearCanvas)
    clear.pack()
    predict = Button(root, text='Predict', command=app.predictDigit)
    predict.pack()
    answer = Label(root, text="Answer Goes Here", font=app.fontStyle)
    answer.pack()
    root.title('Draw a Digit')
    root.mainloop()

```

OUTPUT:



5. Write a python program to classify text as spam or not spam using Naïve Bayes Classifier.

Program:

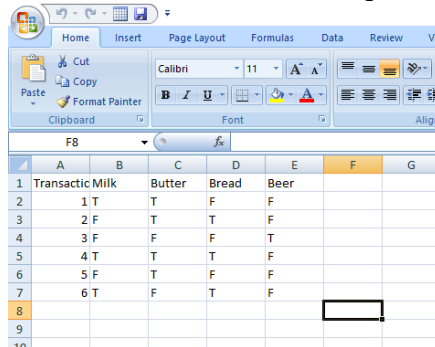
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
# Load the data
data = pd.read_csv(r"spam.csv",encoding="latin-1")
# Preprocess the data
count_vect = CountVectorizer()
X_counts = count_vect.fit_transform(data['v2'])
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X_counts)
y = data['v1']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2)
# Train the Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)
# Test the model
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, pos_label='spam'))
print("Recall:", recall_score(y_test, y_pred, pos_label='spam'))
print("F1-score:", f1_score(y_test, y_pred, pos_label='spam'))
# Classify new text
new_text = ["Claim your free gift today!", "Hey, how's it going?"]
new_text_counts = count_vect.transform(new_text)
new_text_tfidf = tfidf_transformer.transform(new_text_counts)
new_text_pred = clf.predict(new_text_tfidf)
print(new_text_pred)
```

OUTPUT:

```
Accuracy: 0.9614349775784753
Precision: 1.0
Recall: 0.7094594594594594
F1-score: 0.8300395256916996
['spam' 'ham']
```

6. Use Weka and experiment with the following classifiers: Association Rule Mining (Apriori). Agglomerative and Divisive Clustering

#1) Prepare an excel file dataset and name it as “apriori.csv”



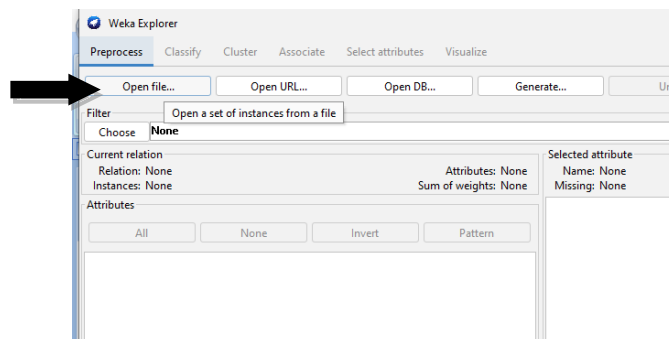
	A	B	C	D	E	F	G
1	Transactic Milk		Butter	Bread	Beer		
2	1 T	T		F	F		
3	2 F	T		T	F		
4	3 F	F		F	T		
5	4 T	T		T	F		
6	5 F	T		F	F		
7	6 T	F		T	F		

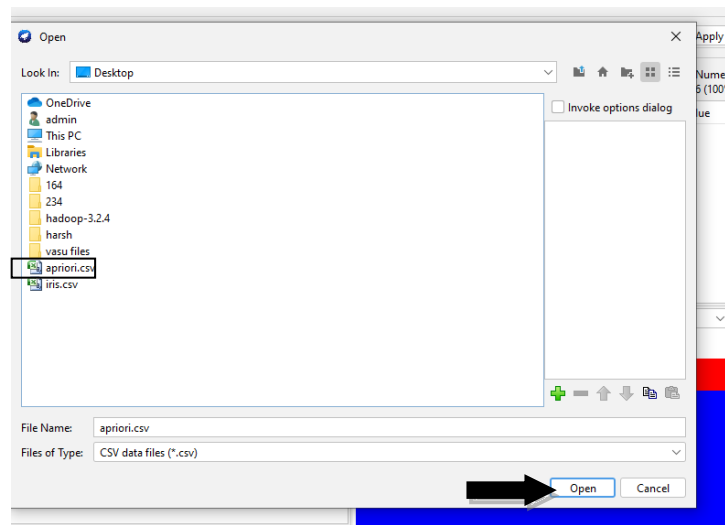
Program:

i) Apriori

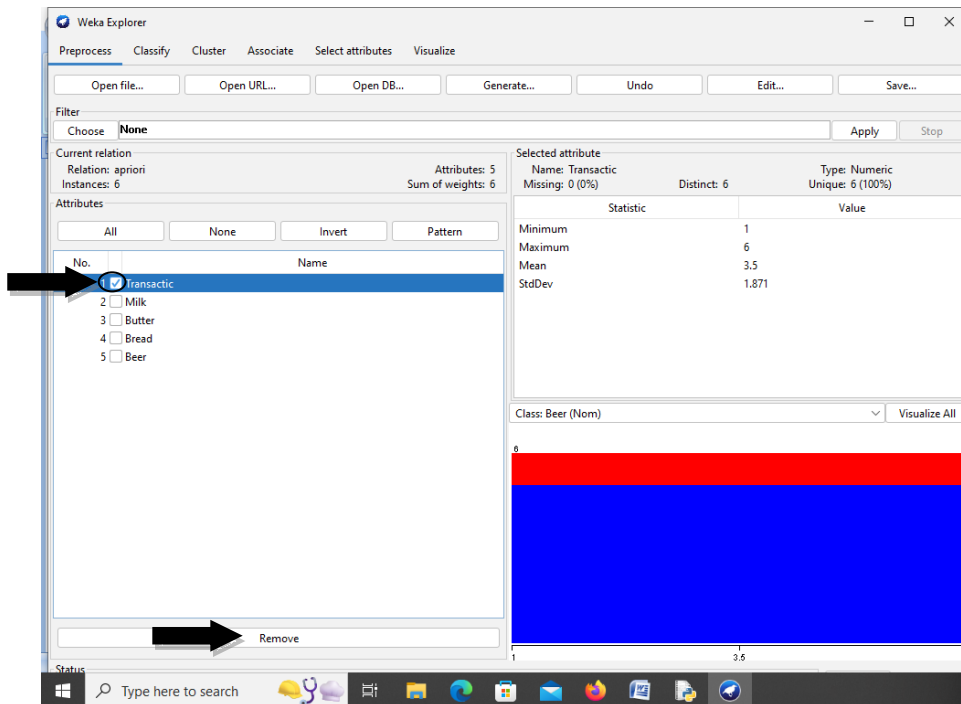


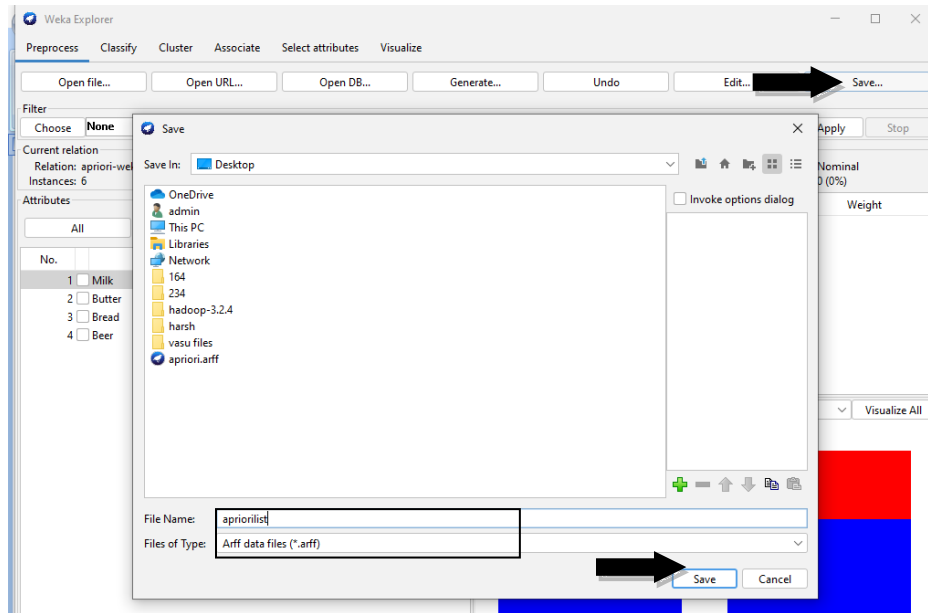
2) Open Weka explorer and under preprocess tab choose “apriori.csv”





3) Remove the transaction field by clicking the check box and click on the remove as shown in the image below. Now save the file as “apriorilist.arff”.

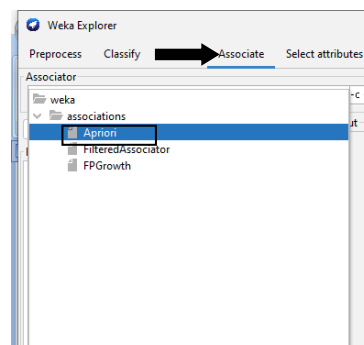


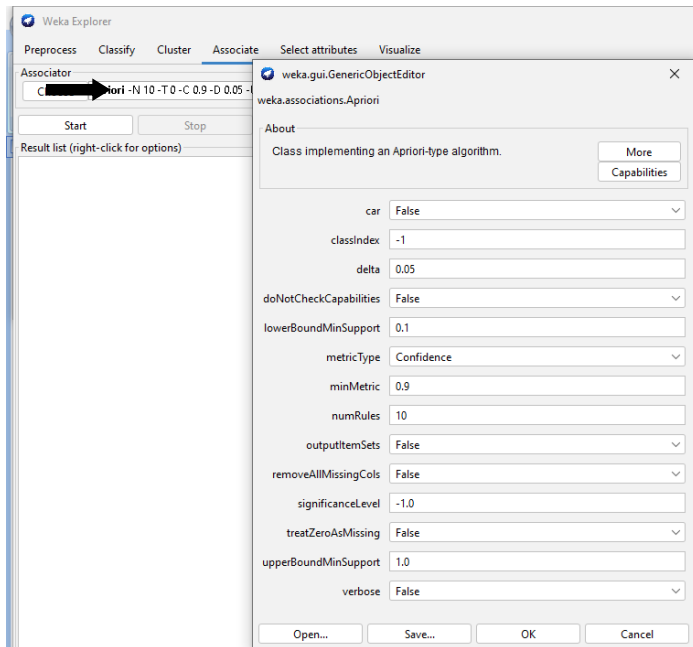


5) Go to the associate tab. The apriori rules can be mined from here.

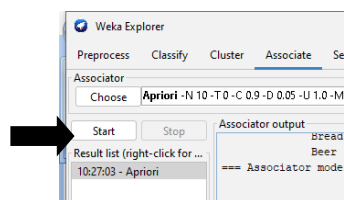
6) Click on Choose to set the support and confidence parameters. The various parameters that can be set here are:

- "lowerBoundMinSupport" and "upperBoundMinSupport", this is the support level interval in which our algorithm will work.
- Delta is the increment in the support. In this case, 0.05 is the increment of support from 0.1 to 1.
- metricType can be "Confidence", "Lift", "Leverage" and "Conviction". This tells us how we rank the association rules. Generally, Confidence is chosen.
- numRules tells the number of association rules to be mined. By default, it is set as 10.
- significanceLevel depicts what is the significance of the confidence level.

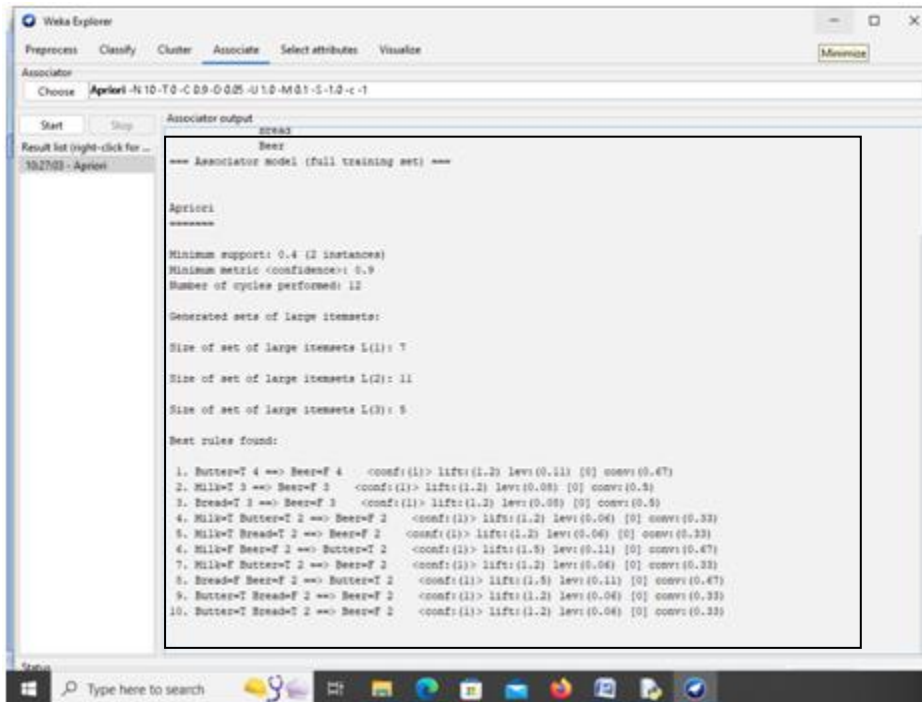




7) The textbox next to choose button, shows the “Apriori-N-10-T-0-C-0.9-D 0.05-U1.0-M0.1-S-1.0-c-1” , which depicts the summarized rules set for the algorithm in the settings tab.

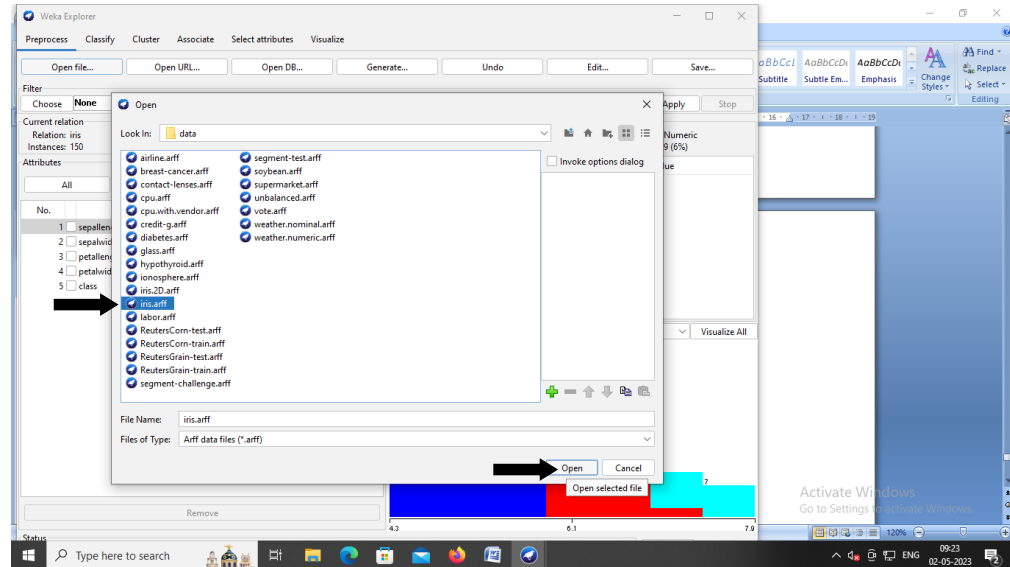


8) Click on Start Button. The association rules are generated in the right panel. This panel consists of 2 sections. First is the algorithm, dataset chosen to run. The second part shows the Apriori Information



ii) Agglomerative and Divisive Clustering

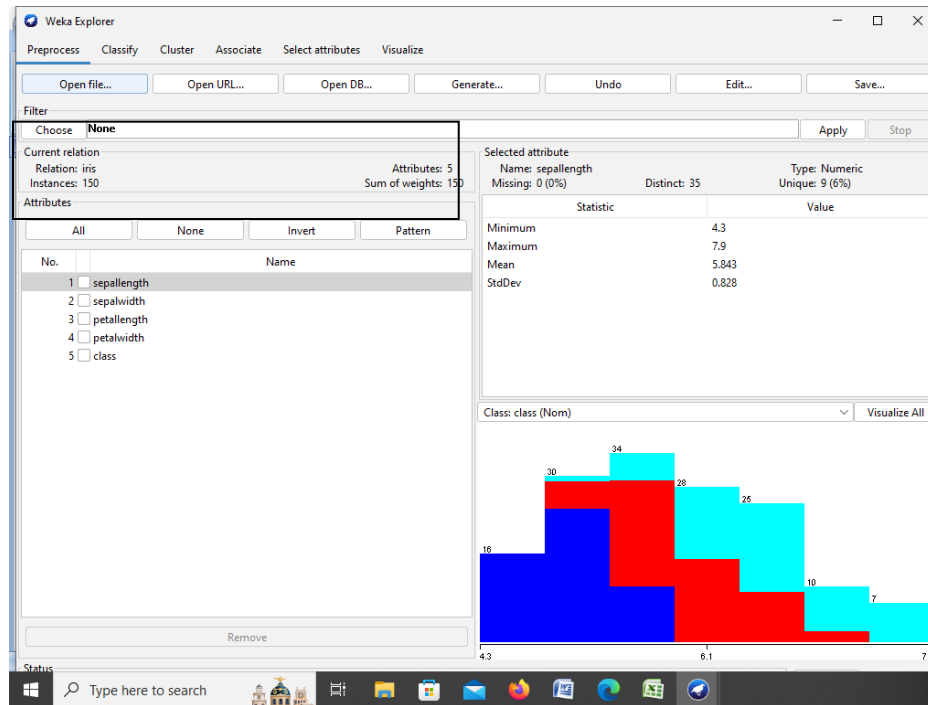
path : C:\Program Files\Weka-3-8-6\data\iris.arff



As in the case of classification, WEKA allows you to visualize the detected clusters graphically. To demonstrate the clustering, we will use the provided iris database. The data set contains three classes of 50 instances each. Each class refers to a type of iris plant.

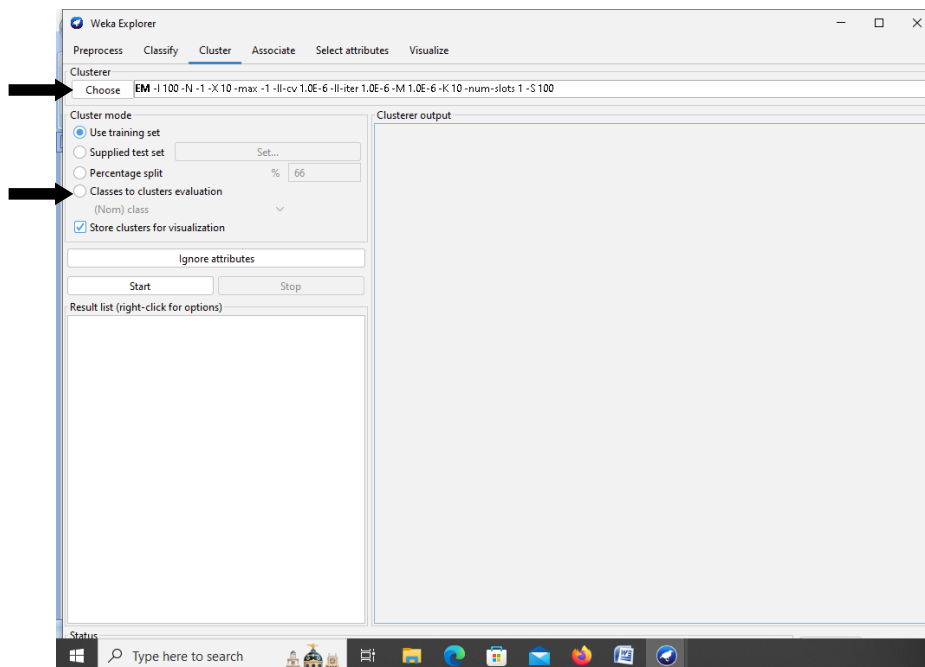
Loading Data

In the WEKA explorer select the Preprocess tab. Click on the Open file → option and select the iris.arff file in the file selection dialog. When you load the data, the screen looks like as shown below -

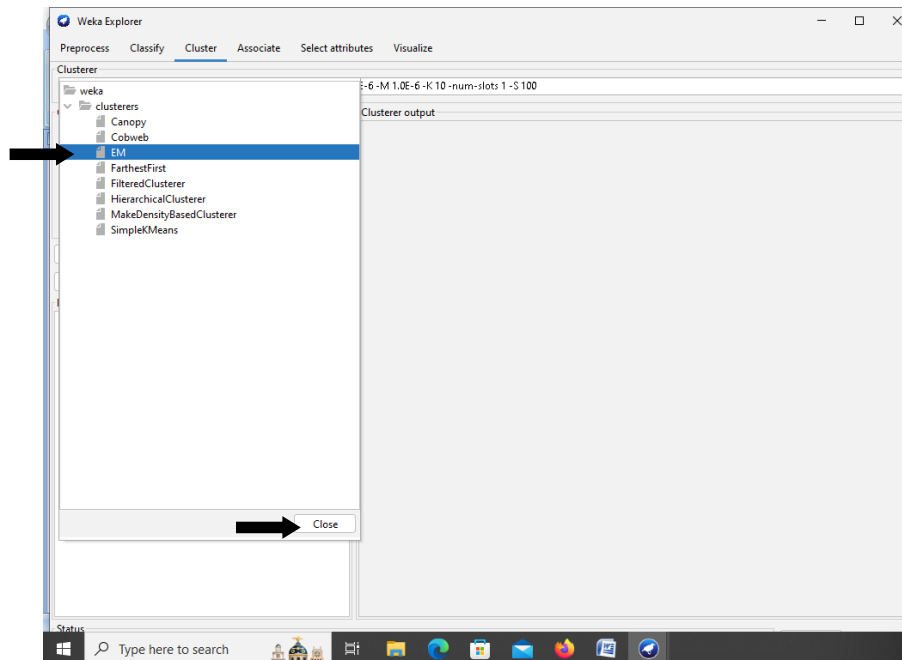


Machine Learning

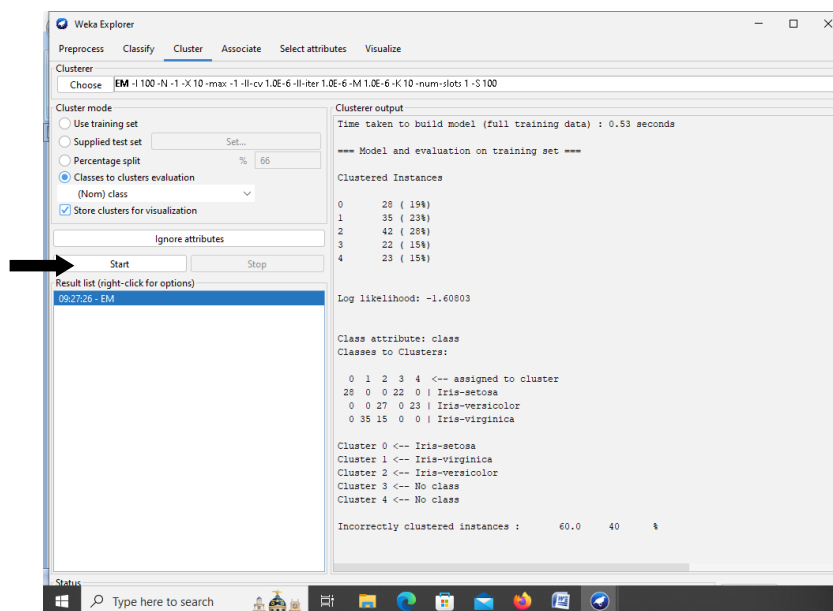
You can observe that there are 150 instances and 5 attributes. The names of attributes are listed as sepalength, sepalwidth, petallength, petalwidth and class. The first four attributes are of numeric type while the class is a nominal type with 3 distinct values. Examine each attribute to understand the features of the database. We will not do any preprocessing on this data and straight-away proceed to model building.



Now , select EM as the clustering algorithm. In the cluster mode sub window , select the classes to clusters evaluation option as shown nin the screenshot below-



Click on the start button to process the data. After a while, the results will be presented on the screen. Next, let us study the results.



The output of the data processing is shown in the screen below -

```

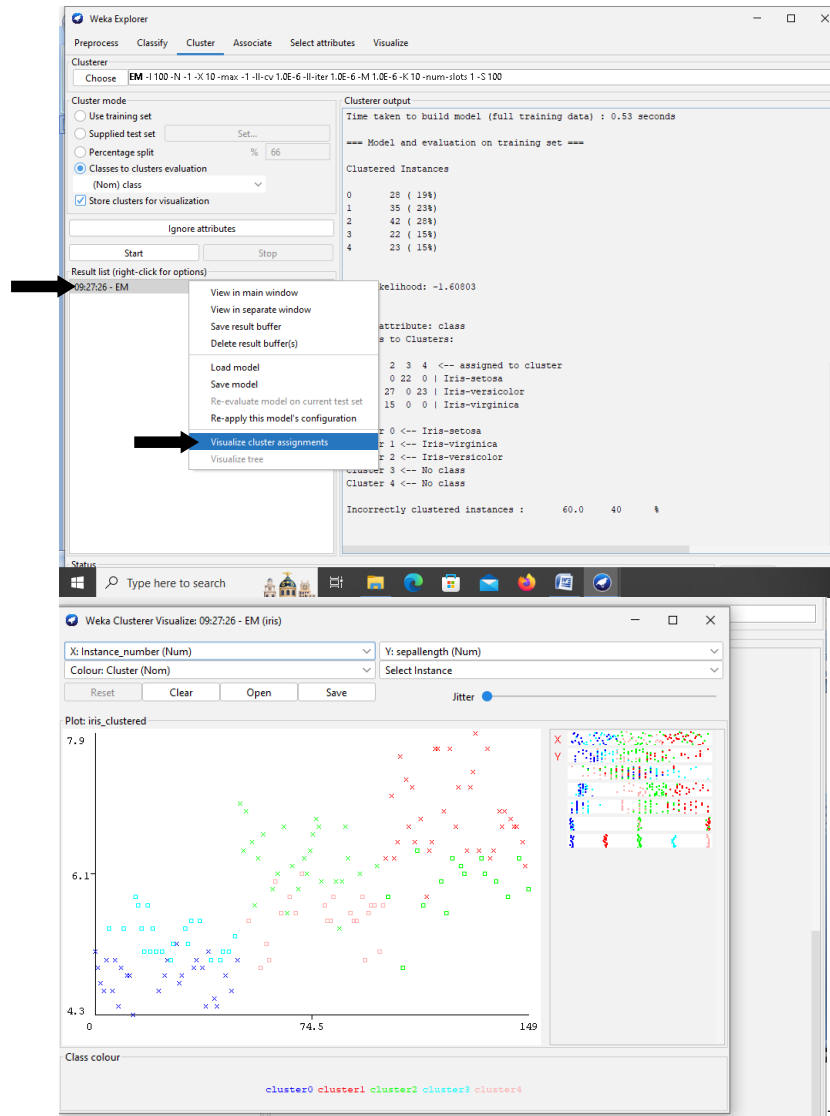
Cluster <- Iris-setosa
Cluster 1<< Iris-virginica
Cluster 2 <- Iris-versicolor Cluster 3- No class
Cluster 4 <- No class
  
```

The Cluster 0 represents setosa, Cluster 1 represents virginica, Cluster 2 represents versicolor, while the last two clusters do not have any class associated with them.

If you scroll up the output window, you will also see some statistics that gives the mean and standard deviation for each of the attributes in the various detected clusters. This is shown in the screenshot given below –

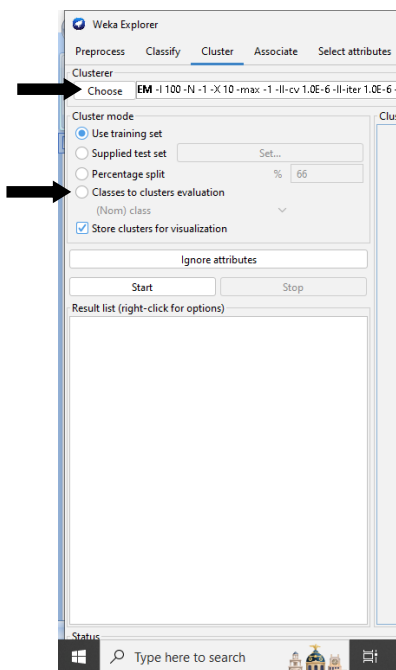
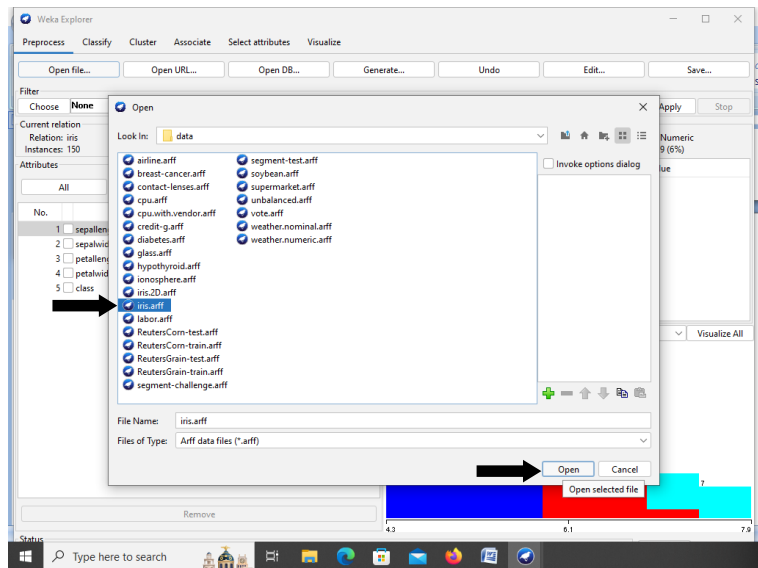
Visualizing Clusters

To visualize the clusters, right click on the EM result in the Result list. You will see the following options -



ii). Hierarchical cluster :

path : C:\Program Files\Weka-3-8-6\data\iris.arff



Applying Hierarchical Clusterer

To demonstrate the power of WEKA, let us now look into an application of another clustering algorithm. In the WEKA explorer, select the Hierarchical Clusterer as your ML. algorithm as shown in the screenshot shown below -

