Machine Learning Assignment

2018-2019

Student Name: Valerija Holomjova

Course Code: ICS2207

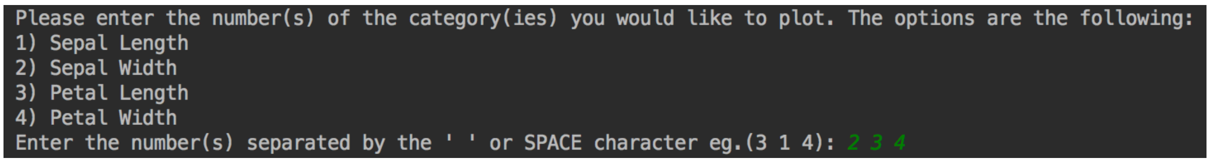
Table of Contents

Description 4
Overview 4
Completion 5
Third-Party Libraries/Packages 6
Executing the Code 7
PyCharm IDE 7
Linux Terminal (Ubuntu) 8
Source Code 9
start() function 9
Artifact 1 9
Global Variables 9
Data class 10
getData() function 10
askProp() function 11
checkValid() function 11
getCoord() function 12
getCoordCluster() function 13
plotPoints() function 13
Artifact 2 16
Cluster class 16
getMinAndMax() function 17
getNewMean() function 17
updateMeans() function 17
euclidDistance() function 18
getAssignedCluster() function 19
getCommonAttr() function 19
createKMClusters() function 19
Artifact 3 20
splitData() function 20
findNeighbours() function 21
testAccuracy() function 22
KNNClustering() function 22
doKNNClustering() function 23

Description

Overview

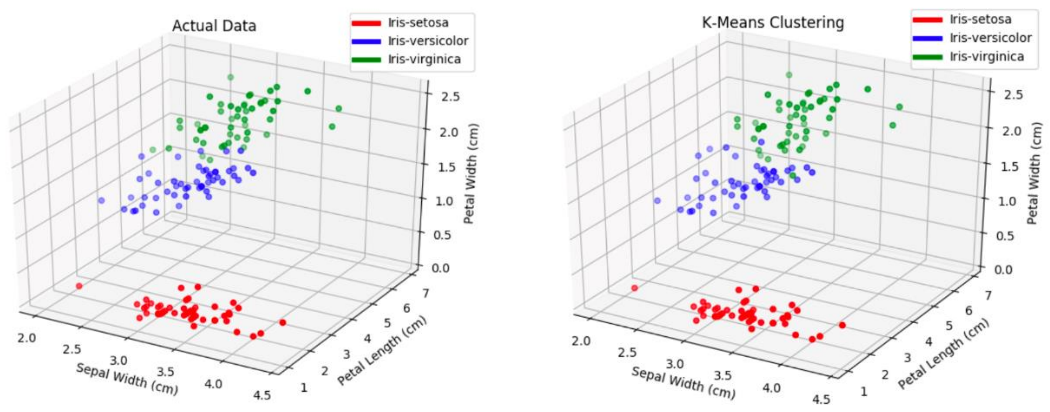
The following program plots chosen features of a data set and then uses an implementation of the K-Means Clustering algorithm and the KNN Clustering Algorithm. At the start of the program the user chooses 1-3 features from the Iris data set which they want to plot. A scatter plot graph called 'Actual Data' of appropriate dimensions will display the data points of these chosen features. The graph on the left below is an example of the chosen features being plotted.



```
Please enter the number(s) of the category(ies) you would like to plot. The options are the following:
1) Sepal Length
2) Sepal Width
3) Petal Length
4) Petal Width
Enter the number(s) separated by the ' ' or SPACE character eg.(3 1 4): 2 3 4
```

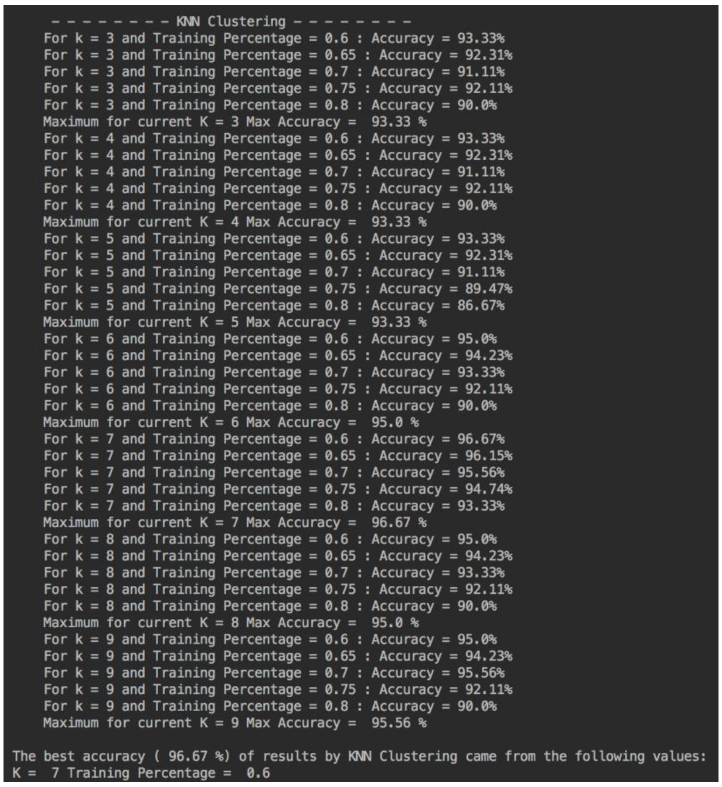
(Figure 1 - Example of asking the user to choose which features to plot)

Next, the K-Means Clustering Algorithm will attempt to classify each point in the dataset to one of 3 clusters (each of which will represent a particular plant class). The clusters will be plotted in a graph called 'K-Means Clustering' and displayed to the user so the user can compare the results to the 'Actual Data' graph.



(Figure 2 - The user's chosen features plotted and the plotted results of the K-Means Clustering algorithm for the data chosen in Figure 1)

In the end, the program will under-go KNN Clustering for different k-values and training percentages and display their accuracy. The k-value and training percentage that gives the highest accuracy is displayed at the end of the program. For more detailed description on how these algorithms were implemented please refer to the 'Source Code' part of this documentation. Please refer to the screenshot below for an example of the output of the KNN Clustering algorithm.



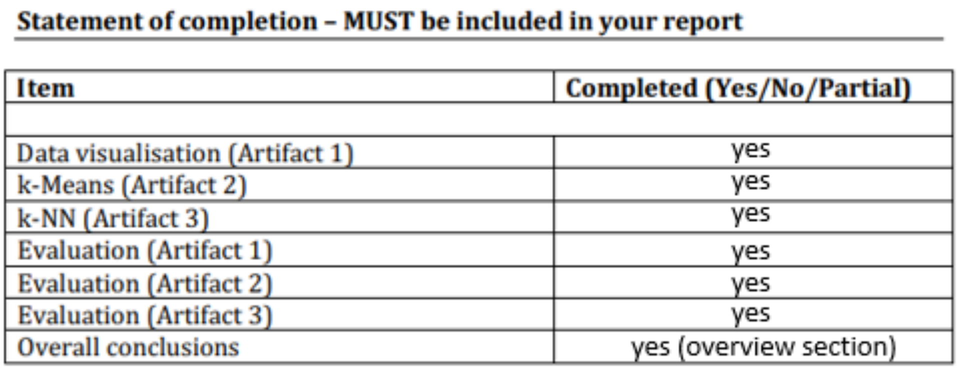
```
 - - - - - - - - KNN Clustering - - - - - - - -
For k = 3 and Training Percentage = 0.6 : Accuracy = 93.33%
For k = 3 and Training Percentage = 0.65 : Accuracy = 92.31%
For k = 3 and Training Percentage = 0.7 : Accuracy = 91.11%
For k = 3 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 3 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 3 Max Accuracy =  93.33 %
For k = 4 and Training Percentage = 0.6 : Accuracy = 93.33%
For k = 4 and Training Percentage = 0.65 : Accuracy = 92.31%
For k = 4 and Training Percentage = 0.7 : Accuracy = 91.11%
For k = 4 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 4 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 4 Max Accuracy =  93.33 %
For k = 5 and Training Percentage = 0.6 : Accuracy = 93.33%
For k = 5 and Training Percentage = 0.65 : Accuracy = 92.31%
For k = 5 and Training Percentage = 0.7 : Accuracy = 91.11%
For k = 5 and Training Percentage = 0.75 : Accuracy = 89.47%
For k = 5 and Training Percentage = 0.8 : Accuracy = 86.67%
Maximum for current K = 5 Max Accuracy =  93.33 %
For k = 6 and Training Percentage = 0.6 : Accuracy = 95.0%
For k = 6 and Training Percentage = 0.65 : Accuracy = 94.23%
For k = 6 and Training Percentage = 0.7 : Accuracy = 93.33%
For k = 6 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 6 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 6 Max Accuracy =  95.0 %
For k = 7 and Training Percentage = 0.6 : Accuracy = 96.67%
For k = 7 and Training Percentage = 0.65 : Accuracy = 96.15%
For k = 7 and Training Percentage = 0.7 : Accuracy = 95.56%
For k = 7 and Training Percentage = 0.75 : Accuracy = 94.74%
For k = 7 and Training Percentage = 0.8 : Accuracy = 93.33%
Maximum for current K = 7 Max Accuracy =  96.67 %
For k = 8 and Training Percentage = 0.6 : Accuracy = 95.0%
For k = 8 and Training Percentage = 0.65 : Accuracy = 94.23%
For k = 8 and Training Percentage = 0.7 : Accuracy = 93.33%
For k = 8 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 8 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 8 Max Accuracy =  95.0 %
For k = 9 and Training Percentage = 0.6 : Accuracy = 95.0%
For k = 9 and Training Percentage = 0.65 : Accuracy = 94.23%
For k = 9 and Training Percentage = 0.7 : Accuracy = 95.56%
For k = 9 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 9 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 9 Max Accuracy =  95.56 %

The best accuracy ( 96.67 %) of results by KNN Clustering came from the following values:
K =  7 Training Percentage =  0.6
```

(Figure 3 - KNN Clustering Algorithm for different k-values and training percentages for the same data chosen in Figure 1)

Some things to note:

- The data the user chooses (by selecting the features) will also be used in both Clustering Algorithms.
- The k-values for both algorithms can be changed but for this assignment, the k-value of the K-Means Clustering algorithm has been set to 3 and the k-values of the KNN Clustering algorithm has been set to a range from 3 – 9.
- The data is shuffled for better results.
- Validation was added to user input to avoid possible errors.

Completion



Statement of completion – MUST be included in your report

Item	Completed (Yes/No/Partial)
Data visualisation (Artifact 1)	yes
k-Means (Artifact 2)	yes
k-NN (Artifact 3)	yes
Evaluation (Artifact 1)	yes
Evaluation (Artifact 2)	yes
Evaluation (Artifact 3)	yes
Overall conclusions	yes (overview section)

Third-Party Libraries/Packages

For this project, the package called ‘matplotlib’ [1] was used in order to display the graphs. Its module ‘matplotlib.lines’ was used to create a legend for the graphs. It’s toolkit extension ‘mplot3d’ was used for 3D plotting. For information on how this package was installed for this assignment, please refer to the ‘Executing the Code’ section of this documentation.

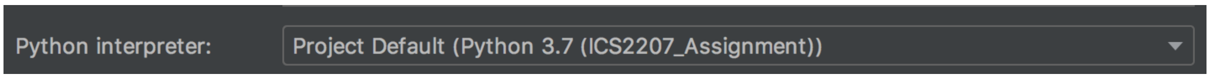
[1] https://matplotlib.org/index.html

Executing the Code

I coded project using the PyCharm IDE [2] by the company JetBrains. I will describe two ways to run the program, both of which I have tested. I will proceed by describing how I loaded the libraries and executed the code in PyCharm. I will also include a section on how the program can be executed in Ubuntu.

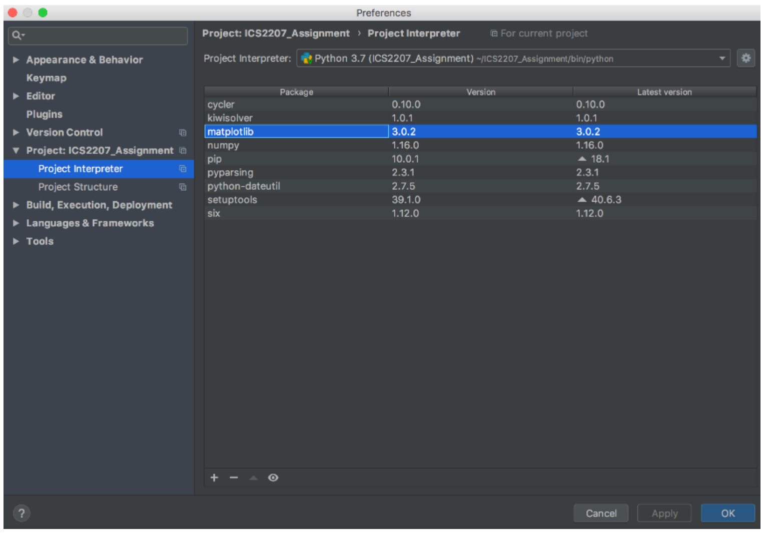
PyCharm IDE

First, I installed Python 3.7 to be used as the Python interpreter for the project. Next, I opened the Assignment file and installed the necessary packages (matplotlib) of this program in the Project Interpreter settings which can be found in the Settings (or Preferences if you're using a MacOS) of the IDE.



Python interpreter: Project Default (Python 3.7 (ICS2207_Assignment))

(Using Python 3.7 as the Python Interpreter)



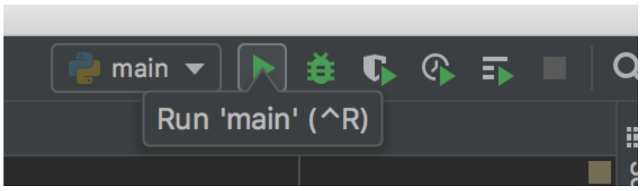
Project: ICS2207_Assignment › Project Interpreter   For current project

Project Interpreter: Python 3.7 (ICS2207_Assignment) ~/ICS2207_Assignment/bin/python

Package	Version	Latest version
cycler	0.10.0	0.10.0
kiwisolver	1.0.1	1.0.1
matplotlib	3.0.2	3.0.2
numpy	1.16.0	1.16.0
pip	10.0.1	▲ 18.1
pyparsing	2.3.1	2.3.1
python-dateutil	2.7.5	2.7.5
setuptools	39.1.0	▲ 40.6.3
six	1.12.0	1.12.0

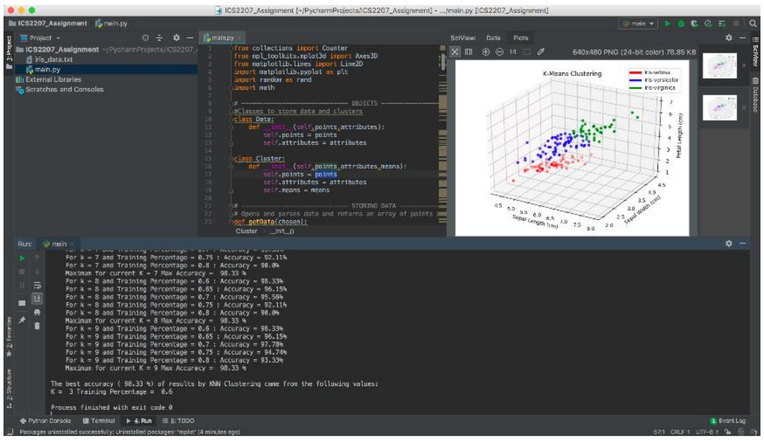
(Installing matplotlib package in the Project Interpreter)

The code can then be run by pressing the 'Play' button in the top right corner of the IDE. The output of the program is displayed in the console and the graphs are displayed on the side.



(Running the Program)

[2] https://www.jetbrains.com/pycharm/



```
from collections import Counter
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
import random as rand
import math

# ------------------------------ OBJECTS ------------------------------
#Classes to store data and clusters
class Data:
    def __init__(self,points,attributes):
        self.points = points
        self.attributes = attributes

class Cluster:
    def __init__(self,points,attributes,means):
        self.points = points
        self.attributes = attributes
        self.means = means

# ------------------------------ STORING DATA ------------------------------
# Opens and parses data and returns an array of points
def getData(chosen):
```

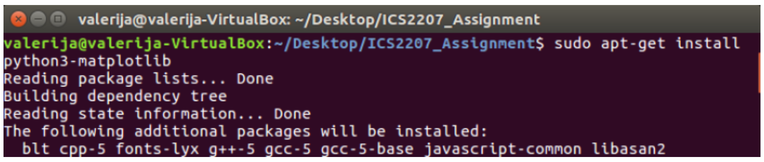
```
For k = 7 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 7 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 7 Max Accuracy =  98.33 %
For k = 8 and Training Percentage = 0.6 : Accuracy = 98.33%
For k = 8 and Training Percentage = 0.65 : Accuracy = 96.15%
For k = 8 and Training Percentage = 0.7 : Accuracy = 95.56%
For k = 8 and Training Percentage = 0.75 : Accuracy = 92.11%
For k = 8 and Training Percentage = 0.8 : Accuracy = 90.0%
Maximum for current K = 8 Max Accuracy =  98.33 %
For k = 9 and Training Percentage = 0.6 : Accuracy = 98.33%
For k = 9 and Training Percentage = 0.65 : Accuracy = 96.15%
For k = 9 and Training Percentage = 0.7 : Accuracy = 97.78%
For k = 9 and Training Percentage = 0.75 : Accuracy = 94.74%
For k = 9 and Training Percentage = 0.8 : Accuracy = 93.33%
Maximum for current K = 9 Max Accuracy =  98.33 %

The best accuracy ( 98.33 %) of results by KNN Clustering came from the following values:
K =  3 Training Percentage =  0.6

Process finished with exit code 0
```

Packages uninstalled successfully: Uninstalled packages: 'mplot' (4 minutes ago)

(Example of Program output in PyCharm)

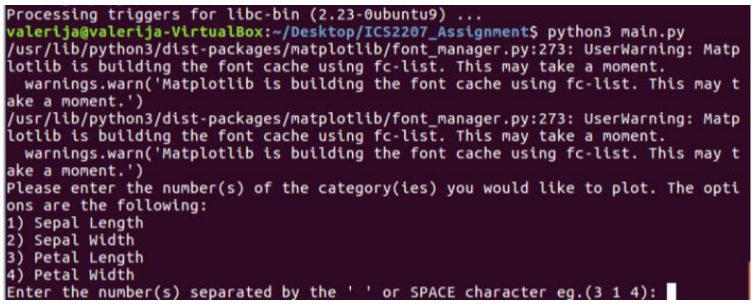
Linux Terminal (Ubuntu)

To run the code in Ubuntu, I first installed the matplotlib library using the "`sudo apt-get install python3-matplotlib`" command. Then I navigated to the Assignment folder containing the mani.py file and run the "`python3 main.py`" command to run the program. When given valid user input, the program will open two graphs and after they are closed, the results from the KNN Clustering algorithm results will be displayed in the terminal.



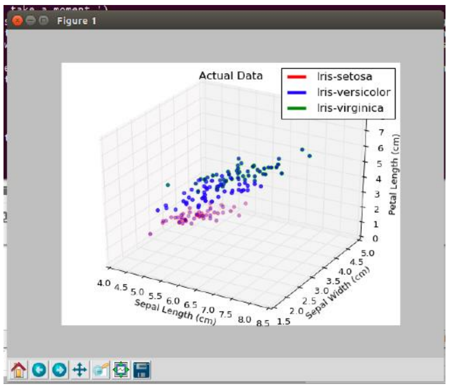
```
valerija@valerija-VirtualBox:~/Desktop/ICS2207_Assignment$ sudo apt-get install
python3-matplotlib
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  blt cpp-5 fonts-lyx g++-5 gcc-5 gcc-5-base javascript-common libasan2
```

(Installing matplotlib package)

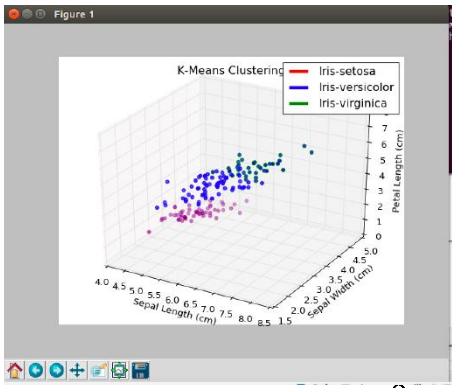


```
Processing triggers for libc-bin (2.23-0ubuntu9) ...
valerija@valerija-VirtualBox:~/Desktop/ICS2207_Assignment$ python3 main.py
/usr/lib/python3/dist-packages/matplotlib/font_manager.py:273: UserWarning: Matp
lotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may t
ake a moment.')
/usr/lib/python3/dist-packages/matplotlib/font_manager.py:273: UserWarning: Matp
lotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may t
ake a moment.')
Please enter the number(s) of the category(ies) you would like to plot. The opti
ons are the following:
1) Sepal Length
2) Sepal Width
3) Petal Length
4) Petal Width
Enter the number(s) separated by the ' ' or SPACE character eg.(3 1 4):
```

(Running the Program)



(Example of graphs popping up)



Source Code

start() function

The following function links together all three artifacts.

The first part of the function retrieves the x , y and z co-ordinates and attributes from the global data object (which was declared and initialized outside the function as a global variable) using the 'getCoord()' function. This was done so that the data could be plotted using the 'plotPoints()' function. Note that the title of the graphs displaying the global data will be called 'Actual Data' to avoid confusion between the K-Means Clustering graph.

```
def start():
    #Create a normal graph with given points.
    x,y,z,attr = getCoord(data)
    plotPoints(x, y, z,attr,'Actual Data')
```

The second part of the function obtains k cluster objects from K-Means Clustering and then retrieves the x, y and z co-ordinates and attributes from them using the 'getCoordCluster()' function. However, this time the clusters will be plotted on a graph titled 'K-Means Clustering' so that the results to the actual data can be compared. To see an example of how the graphs are displayed, please refer to the Overview section of this documentation.

```
#Do K-Means Clustering
clusters = createKMClusters()
x, y, z, attr = getCoordCluster(clusters)
plotPoints(x, y, z, attr,'K-Means Clustering')
```

The last part of the function calls the 'doKNNClustering()' method which will perform KNN clustering for different k-values and training percentages and output the best results obtained to the user.

Artifact 1

The functions described in this section of the documentation were implemented to complete Artifact 1 of the project. These functions are responsible for loading the dataset and allowing the user to select one-three features from the data which are then plotted in a scatter plot graph with the appropriate dimensions. This section will proceed by describing each of the functions and classes that were used.

Global Variables

The following variables were defined so that they could be used in all the functions if needed. The variable 'no_choices' stores the number of features the user chooses in the 'askProp()' function. The variable 'k' stores the k value which will be used in the K-Means Clustering algorithms. The variable 'global_points' will store an array of points corresponding to their values of the features that were chosen by the user. The variable 'global_attributres' will store an array of points corresponding to their plant class name (eg. Iris-setosa). The variable' global_axes' will store the appropriate axes labels that will be needed for better visualization when plotting the graph. The previous three variables are obtain from the 'getData()' function which will be described in the section further on. The 'data' variable stores a Data object which is defined by the Data class which will store the 'global_points' and 'global_attributes' variables.

```
no_choices = 0 #no. of features the person chose - defined in askProp()
k = 3 #K value - remains constant
```

```
#Creating a global variable data
global_points, global_attributes, global_axes = getData(askProp())
data = Data(global_points, global_attributes)
```

Data class

The following class was used to define objects that will store the data the user selects. The points variable stores an array where each point is an array of plant attribute values that were chosen. For instance, if the user decides to choose the sepal width and sepal length feature from the data, each point in the points array will store an array containing their sepal width and sepal length value.

```
class Data:
    def __init__(self,points,attributes):
        self.points = points
        self.attributes = attributes
```

getData() function

The following function is used to return an array of points, attributes and axes labels from an array of features the that were chosen by the user from the 'askProp()' function. The 'chosen' variable will contain these selected features which are stored in their numerical values as inputted by the user.

The 'axes_choices' variables stores all the possible axes labels which can be used in the same order as the feature options were displayed to the user in the 'askProp()' function. It then retrieves the appropriate axes labels from the 'axes_choices' array based on the order and features the user chose and places them into an array. For instance, if the user chose the 3rd and 2nd features from the 'askProp()' function, the 'axes' array will store the 3rd and 2nd element of the 'axes_choices' array in the exact same order. This will later represent the labels of the x axis and y axis when plotting the data.

```
def getData(chosen):
    axes_choices = ["Sepal Length (cm)", "Sepal Width (cm)", "Petal Length (cm)",
"Petal Width (cm)"]
    #Get axes needed for plotting later
    axes=[axes_choices[int(i)-1] for i in chosen]
```

Next, an empty array of points and attributes are created which will store the feature values and plant class of each data point respectively. The 'iris_data.txt' file is then opened and converted into a list and stored in the variable 'dataset'. The 'dataset' variable is then shuffled for better clustering results. Next, each item in the list is iterated through. For each item, a temporary array called 'temp_point' is made which will store the chosen feature values of the current data point (or item in the list).

The item is stripped for any newline characters and split using the ',' character as a delimiter to retrieve each feature. This results in an array of features which is stored in the variable 'point'. If the item has 1 or less features, this means the item was empty and the loop reiterates. Otherwise, the features which the user chose are extracted from the 'point' array (which stores all the features of the current item) and added to the 'temp_point' variables. The 'temp_point' array is then added to the final array of points. The last feature of the item which is the plant class name is added to the attributes array. The loop then reiterates until the points and attributes array is populated with the required data of each data point in the dataset. The arrays containing the points, attributes and axes labels is then returned.

```
points,attributes = [],[]
# Open the dataset and store it as a list
```

```
with open('iris_data.txt') as f:
    dataset = list(f)
rand.shuffle(dataset)
# Go through each item in the list and obtain the required features of each point
and its attribute
for item in dataset:
    temp_point = []
    point = item.rstrip("\n").split(",")
    if len(point) <= 1:
        continue
    for i in range(len(chosen)):
        temp_point.append(float(point[int(chosen[i])-1]))
    # Push the chosen feature values and attribute of the point
    points.append(temp_point)
    attributes.append(point[4])
return points, attributes, axes
```

askProp() function

The following function displays a list of features the user can select and returns the chosen features as an array. Each feature is assigned a numeric value, the user has to input the features they want by inputting the number it represents. For instance, if the user wants to choose the ‘Petal Width’ and ‘Petal Length’ features, the user needs to input the numbers 3 and 4 separated by a space character. The input the user gives is separated using the space characters as a delimiter and stored in an array called ‘chosen’. Next, the global ‘no_choices’ is set to the number of features the user chose. Next the function checks whether the input given by the user is valid using the ‘checkValid()’ function. Note that the ‘askProp()’ function loops until the user input is valid, hence, if the ‘checkValid()’ function returns false, the user will be asked to input the features they want again. If the ‘checkValid()’ function returns true, the array of chosen features is returned. These will be used in the ‘getData()’ function.

```
# Ask user to enter select properties to plot - and returns a list of the
properties they chose
def askProp():
    # Loops until tbe user enters valid input
    while True:
        print('Please enter the number(s) of the category(ies) you would like to
plot. The options are the following:')
        print('1) Sepal Length')
        print('2) Sepal Width')
        print('3) Petal Length')
        print('4) Petal Width')
        chosen = input("Enter the number(s) separated by the ' ' or SPACE character
eg.(3 1 4): ").split(' ')
        # Setting the global no. of choices variable
        global no_choices
        no_choices = len(chosen)
        # Check if arguments are valid
        if checkValid(chosen) == False:
            continue
        break
    return chosen
```

checkValid() function

The follow functions take an array of features that were chosen by the user from the ‘askProp()’ function and returns true or false based on whether they are valid or not. First, the function checks whether one of the features is an empty string indicating that the user added an extra space as an argument. If this condition is true, the functions returns false.

```
#Extra Validation
def checkValid(chosen):
    # Check if extra space was entered
    if any(i is '' for i in chosen):
        print('Please input choices without extra space.\n')
        return False
```

Next, each feature is iterated through. If one of the features is not numeric or is not any of the given numerical options (i.e. 1,2,3,4), then the function returns false. Otherwise, the function proceeds by checking whether the user entered the right number of features. Since the program only accepts 1 – 3 chosen features, if the number of features the user inputting is less than 1 or greater than 3, the function will return false. If the user input is valid, the function will return true.

```
# Check if given option is not 1,2,3,4
for i in chosen:
    if(i.isnumeric() == False):
        print('Please enter 1,2,3 or 4 as options.\n')
        return False
    i = int(i)
    if not(i in [1,2,3,4]):
        print("Please enter 1,2,3 or 4 as options.\n")
        return False
# Validation so that the right amount of properties are used.
if len(chosen) < 1 or len(chosen) > 3:
    print('Please enter 1-3 categories.\n')
    return False
```

Below is a table indicating what the 'checkValid()' function will do based on the input the user gives.

User Input	Program Output
"1 2"	Valid Input – function returns true.
"1 2 "	Invalid Input – function display the message 'Please input choices without extra space.\n' and returns false.
"a 1"	Invalid Input – function display the message 'Please enter 1,2,3 or 4 as options.\n' and returns false.
"3 6"	Invalid Input – function display the message 'Please enter 1,2,3 or 4 as options.\n' and returns false.
"1 2 3 4"	Invalid Input – function display the message 'Please enter 1-3 categories.\n' and returns false.

getCoord() function

The following function takes the point array of an object to create an array of x, y and z values which are then returned. Note that the object can be a cluster object or a data object, both have an array of points. The function works by accessing each point in the object's array of points. It then populates the co-ordinate arrays using each point based on the number of features the user has chosen:

- If the user chosen only one feature, the function will take the first feature value of the point and push it into the X co-ordinate array and the Y and Z co-ordinate arrays will remain empty.

- If the user chose two features, the function will push the first feature value into the X co-ordinate array and the second feature value into the Y co-ordinate array and Z co-ordinate array will remain empty.
- If the user chose three features, the function will push the first feature value into the X co-ordinate array, the second feature value into the Y co-ordinate array and third feature value into the Z co-ordinate array.

Thus, the co-ordinate arrays are arrays of feature values for each point. At the end of the function all three co-ordinate arrays are returned and the object's array of attributes.

```
#Return X, Y, Z values and attributes for an object
def getCoord(object):
    x,y,z=[],[],[]
    for point in object.points:
        if(no_choices >= 1):
          # Add the first feature value of point to X array
          x.append(point[0])
        if(no_choices >= 2):
          # Add the second feature value of point to Y array
          y.append(point[1])
        if(no_choices == 3):
          # Add the third feature value of point to Z array
          z.append(point[2])
    return x,y,z,object.attributes
```

getCoordCluster() function

The following function creates an array of x, y and z co-ordinate values for an array cluster objects. It does this by looping through each cluster and getting its x, y, z co-ordinate values and its array of attributes. It then merges these x, y, z co-ordinates and attributes with the variables holding the overall x, y, z-co-ordinates of the clusters. The process is repeated for each cluster. At the end of the function, the 'x', 'y' and 'z' variables contain the x, y and z values for all of the clusters merged together and the 'attr' variable contains all the attributes of the clusters merged together. These values are then returned. This function is needed to plot the K-Means clusters from Artifact 2 of the project.

```
#Return X, Y, Z values and attributes for a cluster
def getCoordCluster(clusters):
    x,y,z,attr = [],[],[],[]
    for cluster in clusters:
       # Get the x, y, z and attr for the cluster
       temp_x,temp_y,temp_z,temp_attr = getCoord(cluster)
       # Merge values
       x = x + temp_x
       y = y + temp_y
       z = z + temp_z
       attr = attr + temp_attr
    return x,y,z,attr
```

plotPoints() function

The following function creates scatter plot graphs given an array of x, y and z co-ordinate values, an array of attributes and a title for the graph. At the start of the function an array representing the color of each co-ordinate is made by looping through each attribute in the given attributes array and assigning the color 'red' for the attribute 'Iris-setosa', 'blue' for the attribute 'Iris-versicolor' and 'green' for the attribute 'Iris-virginica'. In order to create a legend for the graph, an array of the possible attributes called 'options' is created as well as an array of lines with the appropriate colors to match the ‘option’ array called ‘custom_lines’. The lines are creating using the Line2D instance from the matplotlib.lines module. This was done for better visualization of the graphs.

```
#Plot points given x,y,z values and attribute
def plotPoints(x,y,z,attr,title):
    global global_axes
    #Get colours for data points from attribute array
    color = [('red' if i == 'Iris-setosa' else 'blue' if i == "Iris-versicolor"
else 'green') for i in attr]
    #Custom Legend
    options = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
    custom_lines = [Line2D([0], [0], color='red', lw=4),
                    Line2D([0], [0], color='blue', lw=4),
                    Line2D([0], [0], color='green', lw=4)]
```

In the next part of the function the scatter plot graphs are plotted and displayed using the matplotlib.pyplot library and the Axes3D objects from the mpl_toolkits.mplot3d toolkit. The dimension of the graph depends on the number of features the user chose in the ‘askProp()’ function. The next part of the section describes how the 3 different types graphs were created. Note that the end of the function, the title is added to the final graph using the ‘title()’ function and then displayed using the ‘show()’ function.

```
# Adding title and displaying graph
plt.title(title)
plt.show()
```

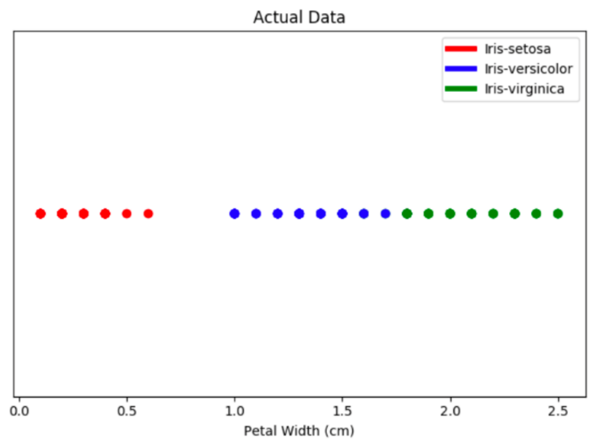
Plotting 1D scatter plot

If only one feature was chosen, a 1D scatter plot is made using the functions from matplotlib. This is done by creating a figure using ‘plt.figure()’ and then creating an axes object of a 1x1 grid in the figure using the ‘add_subplot()’ function. Next the scatter plot points are plotted using the ‘scatter()’ function. This function plots these points using the x co-ordinate values and an array of y co-ordinate values where each element is 0. The ‘scatter()’ also accepts the ‘color’ array so that each point could be displayed in a colour appropriate to its attribute.

Next, a legend is created for the graph using the ‘legend()’ function which uses the ‘custom_lines’ and ‘options’ variables as arguments which were defined earlier. The labels of the x axis is added using the ‘set_xlabel()’ function which gets its argument from the global variable ‘global_axes’. Also, the values on the y axis were hidden since this graph is meant to be a 1D scatter plot and the y values were all set to 0’s anyways.

```
# Plotting a 1D scatter plot using just x coordinates
if(no_choices == 1):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(x, [0]*len(x), color=color)
    ax.legend(custom_lines, options)
    ax.set_xlabel(global_axes[0])
    # Hide the values on the y-axis.
    ax.get_yaxis().set_visible(False)
```

Below is an example of a 1D Scatter plot illustrating the Petal Width feature of the dataset.

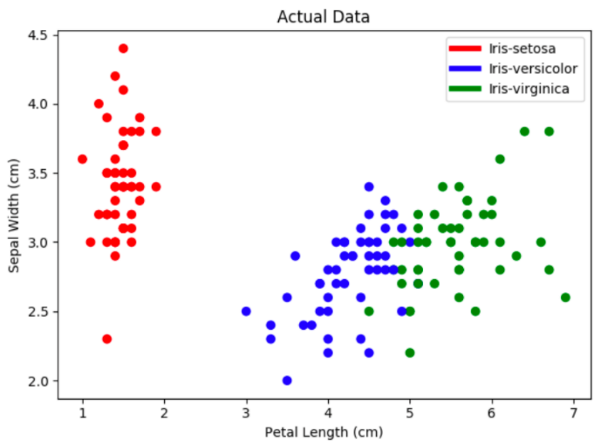


Plotting 2D scatter plot

If two features were chosen by the user, a graph is made in the same way as described when plotting a 1D scatter plot above. However this time, the y co-ordinate values are passed into the ‘scatter()’ function instead of an array’s of 0. Another difference is that the label of the y axis is displayed using the ‘set_ylabel()’ function and the values of the y axis were not hidden.

```
# Plotting a 2D scatter plot using x and y co-ordinates
elif(no_choices == 2):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(x, y, color=color)
    ax.legend(custom_lines, options)
    ax.set_xlabel(global_axes[0])
    ax.set_ylabel(global_axes[1])
```

Below is an example of a 2D Scatter plot illustrating the Petal Length and Sepal Width features of the dataset.



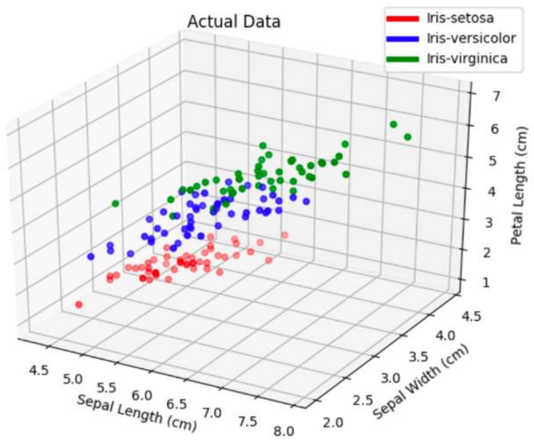
Plotting 3D scatter plot

If three features were chosen by the user, a graph is made in the same way as described when plotting a 2D scatter plot above. However this time, the ‘add_subplot()’ function has an additional argument which makes the projection of the graph 3D. Another difference is that in the ‘scatter()’

function, the z co-ordinate values are used and a label for the z axis is added to the graph using the 'set_zlabel()' function.

```
# Plotting a 3D scatter plot using x, y and z co-ordinates
else:
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(x, y, z, color=color)
    ax.legend(custom_lines, options)
    ax.set_xlabel(global_axes[0])
    ax.set_ylabel(global_axes[1])
    ax.set_zlabel(global_axes[2])
```

Below is an example of a 3D Scatter plot illustrating the Petal Length, Sepal Length and Petal Width feature of the dataset.



Artifact 2

The functions described in this section of the documentation were implemented to complete Artifact 2 of the project. These functions are responsible for creating k clusters by implementing the k-means clustering algorithm and then plotting the clusters in a scatter plot graph to compare the results. It is important to note the in the project a k value of 3 is used, however since this is a global variable it can be easily changed. It is also important to note that the same features that were chosen from 'askProp()' will be plotted – however this time their colors will represent the value they are in. This section will proceed by describing each of the functions and classes that were used.

Cluster class

The following class was used to define cluster objects. Each cluster object will have an array of points specific to the cluster where each point represents an array of its feature values (like in the Data class). The cluster object will also have an array of its points attributes and a variable that keeps track of the centroid of the cluster when new points are added.

```
class Cluster:
    def __init__(self,points,attributes,means):
        self.points = points
        self.attributes = attributes
        self.means = means
```

getMinAndMax() function

The following function finds the minimum and maximum values of each feature that the user selected in the ‘askProp()’ function. It functions by creating an empty array called ‘temp’ where each element represents a chosen feature and all of its values from the data points. For instance, if the user chose only one feature in the ‘askProp()’ function, the ‘temp’ array would have one element which is an array of all the values of this particular feature from the data points. If the user chose two features, the ‘temp’ array would have two elements which are arrays containing all the values of their respective feature.

```
#Create a temp array to store all values of a feature
temp = [[] for i in range(no_choices)]
#Access each point
for point in data.points:
    #Access each feature
    for i,value in enumerate(point):
        #Add the points feature value to the appropriate array
        temp[i].append(value)
```

This is done by looping through each point in the points array of the global data object and then accessing the stored features of the point. It then appends each of the features to their respective arrays in the ‘temp’ variable (the first stored feature will go in the first array of ‘temp’ and so on).

Next the function creates two arrays storing the minimum and maximum value of each of the features. Hence, the first element in the minimum array is the minimum value of the first feature and second element in the minimum array is the minimum value of the second feature and so on. The same can be applied to the maximum array. Both arrays are then returned at the end of the function.

```
#Returns an array of minimum/maximum values of each feature
temp_min = [min(temp[i]) for i in range(no_choices)]
temp_max = [max(temp[i]) for i in range(no_choices)]
return temp_min, temp_max
```

getNewMean() function

The following function creates one data point consisting of a random value for each of the chosen features that the user selected in the ‘askProp()’ function. It does this by getting an array of the minimum values and an array of maximum values for each chosen feature. It then creates an empty array of size respective to the number of chosen features. Next, the function iterates for each chosen feature and pushes a random value found between the minimum and maximum value of the feature, rounded to one decimal place. In the end, the function returns an array where each element is a random value of a chosen feature.

```
#Creates a data point of a random feature values
def getNewMean():
    temp_min, temp_max = getMinAndMax()
    means = [None] * no_choices
    # Find a random value for each chosen feature
    for i in range(no_choices):
        means[i] = round(rand.uniform(temp_min[i]+1, temp_max[i]-1),1)
    return means
```

updateMeans() function

The following function updates the centroid (or mean point) of a given cluster. It functions by storing the number of points in the cluster as the letter n. Next it re-calculates the centroid by looping through each feature value of the old centroid (which was the mean of the previous points) and updating it to a new mean using the formula below.

$$\mu_n = \frac{(n-1)\mu_{n-1} + x_n}{n}$$

where

- n = number of points in the cluster
- n-1 = old number of points in the cluster
- μ_n = mean of the values of a feature for the first n points
- μ_{n-1} = old mean of the values of a feature
- x_n = actual value of a feature for a newly added point

After each feature value is updated to represent the mean of their respective feature for all the points in the cluster, a new centroid for the cluster is set and the function returns.

```
def updateMeans(cluster,point):
    # Updates the mean point/centroid for a given cluster
    n = float(len(cluster.points))
    # Access each feature in the old centroid and updating it
    for i,mean in enumerate(cluster.means):
        cluster.means[i] = (mean*(n-1)+point[i])/float(n)
    return
```

euclidDistance() function

The following function finds the Euclidean distance between two points to determine their distance apart. It does this by implementing the following formula (depending on the number of dimensions) which describes the distance between points for n-dimensions.

$$d(p,q) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \cdots + (p_i-q_i)^2 + \cdots + (p_n-q_n)^2}$$

where **p** and **q** are two points and p_n and q_n are their matching co-ordinates on a plane (for instance, p_1 and q_1 are the x co-ordinates of p and q).

The number of dimensions needed are determined by the number of options the user chose in the 'askProp()' function. This is because each feature will be plotted and hence will have its own axis. Thus, the function checks the number of features that were chosen and then uses a 1-dimensional, 2-dimension or 3-dimensional Euclidian distance formula accordingly. Thus, each feature value of a point is considered to be a co-ordinate of the point in the Euclidian distance formula.

For instance, if the user chooses 3 features in the 'askProp()' function, a 3-dimensional Euclidean distance formula will be used and the first feature value of a point will represent the x-co-ordinate, the second will represent the y co-ordinate and the third will represent the z co-ordinate. Since users can only choose up to 3 features, this is the highest dimension that will be calculated.

```
#Eucledian distance where A and B are arrays of x,y,z co-ordinates (feature values)
def euclidDistance(A, B):
    if (no_choices == 1):
        return math.sqrt(math.pow(A[0] - B[0], 2))
    if (no_choices == 2):
        return math.sqrt(math.pow(A[0] - B[0], 2) + math.pow(A[1] - B[1], 2))
    if (no_choices == 3):
        return math.sqrt(math.pow(A[0] - B[0], 2) + math.pow(A[1] - B[1], 2) +
math.pow(A[2] - B[2], 2))
```

getAssignedCluster() function

The following function finds a cluster closest to a given point and returns the index of the found cluster. It works by iterating through each of the cluster objects and calculating the Euclidian distance between the point and the centroid of the cluster. Each time a smaller Euclidean distance between a point and cluster is found, a variable called 'min_value' stores the Euclidean distance value and the 'position' variable stores the position of the cluster.

For the first iteration in the function, the 'min-value' is set to the Euclidean distance the point and the first cluster to avoid any possible errors. At the end of the function, the position of the cluster object in clusters array which had the smallest Euclidean distance with the point is returned.

```
# Finds the nearest cluster of a given point and returns its index.
def getAssignedCluster(point,clusters):
    min_value,position = 0, 0
    #Iterate through each cluster
    for i,cluster in enumerate(clusters):
        #Find the distance between the point and current cluster
        euclid_distance = euclidDistance(point,cluster.means)
        #Set an initial minimum value
        if i==0:
            min_value = euclid_distance
        #Find a new minimum
        if euclid_distance < min_value:
            min_value = euclid_distance
            position = i
    return position
```

getCommonAttr() function

The following function returns the most common attribute in a given array of attributes. It works by creating a variable to store a 'Counter' container of the array. This container keeps track of how many times equivalent values are added, or in other words, it keeps track of the frequencies of each attribute. It then uses the 'most_common()' function to return an array containing the element with the highest count. Then the word is extracted from this array (note that it is stored with its frequency value), and an array of the most common word is created of the same length of the attributes array and returned. This function is used to assign a plant class to the cluster - by finding its most repeated attribute.

```
#Find the most common attribute in a cluster and return an array of it
def getCommonAttr(attributes):
    words = Counter(attributes)
    words.most_common(1)
    return [words.most_common(1)[0][0]] * len(attributes)
```

createKMClusters() function

The following function links together all the functions described above in order to perform K-Means clustering. The function creates k number of clusters using k means clustering and returns them. Please note that the k-value is determined by the global variable 'k' and can be changed at any time, but for this project the k-value was set to 3. The function works by creating an array of k new cluster objects with an empty array of points and attributes. Each cluster has a centroid (or mean) initialized using the 'getNewMean()' function which creates a random point according to the data and returns it.

Next, the function iterates through each point that was stored in the global data object and tries to find the nearest cluster to it using the 'getAssignedCluster()' function. It then adds the point to the points array of the cluster obtained from the ‘getAssignedCluster()’ function and it also adds the attribute to the point to the cluster’s array of attributes. Next the mean (or centroid) of the cluster is updated used the ‘updateMeans()’ function. This process continues for all the points stored in the global data object until all the points are assigned to a cluster.

```
#Returns k Clusters created by K-Means Clustering
def createKMClusters():
    #Create k new clusters of random means/centroids
    clusters = [Cluster([], [], getNewMean()) for i in range(k)]
    #Access each item in data and assign it to appropriate cluster.
    for j,item in enumerate(data.points):
        #Get the index of the nearest cluster.
        i = getAssignedCluster(item,clusters)
        #Add the point to the array of points in given cluster.
        clusters[i].points.append(item)
        #Add attribute
        clusters[i].attributes.append(data.attributes[j])
        #Update cluster mean
        updateMeans(clusters[i],item)
```

Next, the function iterates through each of the populated cluster objects and replaces it’s attributes array with the most common attribute of the cluster using the ‘getCommonAttr()’ function. Note that the function will first check if there are points in the cluster object since the ‘getCommonAttr()’ could crash the program if an empty array is given, hence an error message will be displayed if a cluster is not populated. However, this is just a safety measure and all clusters should be populated. At the end of the function, the array of cluster objects is returned. Please note that the clusters are plotted and displayed as a graph in the ‘start()’ function.

```
#Update attributes of cluster to most common
for i,cluster in enumerate(clusters):
    if(len(cluster.attributes) != 0):
        clusters[i].attributes = getCommonAttr(cluster.attributes)
    else:
        print('Error - cluster has no assigned points.\n')
return clusters
```

Artifact 3

The functions described in this section of the documentation were implemented to complete Artifact 2 of the project. These functions are responsible for the implementation of the KNN Clustering algorithm. Please note that the ‘doKNNClustering()’ function was implemented to perform KNN Clustering for multiple values of k (and training percentages) and tries to find best k and training percentage. The ‘KNNClustering()’ implements the KNN clustering algorithm for a single k-value and training percentage value. It is also important to note that the KNN Clustering was performed on data which was chosen by the user in the ‘askProp()’ section. This section will proceed by describing each of the functions and classes that were used.

splitData() function

The following function splits the global data object into a training set and an evaluation set. Please note that the data which will be split depending on what the features the user chose in the ‘askProp()’ function, hence each point in the data can have 1 – 3 feature values stored.

The function starts by creating two empty Data objects will be used to store the points and attributes for the training set and evaluation set. Next, the function calculates the number of points that will be added to the training set and stores the value in the variable ‘split’. This is done by multiplying the number of points by the training percentage which was passed into the function as the variable ‘percentage’ and rounding it to an integer value. Hence, the ‘split’ variable holds the value of the number of points which will be placed in the training set.

Next, the function iterates through each point and uses the ‘split’ variable to assign the specified amount of points and their attributes into the training set and the rest into the evaluation set. This method was used since the data was already shuffled in the ‘getData()’ function. For example, since there are 150 points in the dataset, if the training percentage was set to 90%, 135 points will be added to the training set and the remaining 15% will be added to the evaluation set. At the end of the function, the training set and evaluation set data objects are returned.

```
#Roughly splits the data according to a percentage - returns training and
evaluation data object.
def splitData(data,percentage):
    training,evaluation = Data([],[]),Data([],[])
    #Calculate how many elements are going into training set
    split = int(round(percentage * len(data.points)))
    for i,point in enumerate(data.points):
        if i < split:
            training.points.append(point)
            training.attributes.append(data.attributes[i])
        else:
            evaluation.points.append(point)
            evaluation.attributes.append(data.attributes[i])
    return training, evaluation
```

findNeighbours() function

The following function tries to find the k closest training points and their attributes from a given point from the evaluation set. The function starts by making a list of distances between the given point and all the points in the training set. It does this by iterating through each training point and calculating the Euclidian distance between it and the given point. It then creates an array storing the distance and the index of the training point which is then pushed into an array called ‘distances’. After this process is done for each training point, the elements in the distances array should represent the Euclidian distances between the testing points and the given point as well as the positions of the respective training points.

Next, the function uses the ‘sort()’ function to sort the ‘distances’ array by the Euclidian distance values of each element in ascending order. Then the function enters a loop for k times, so that it could append the k closest training points and attributes using the indexes that were previously stored in the ‘distances’ array. Since the ‘distances’ array was sorted so that the elements were ordered by their Euclidian distance in ascending order, the index values that were previously stored in the first k elements could be obtained and used to append the correct training points and their attributes to two arrays. Hence these two arrays store the closest three training points to the point passed into the function and their attributes. These two arrays are then returned.

```
def findNeighbours(test_point,training,k):
    distances = []
    #Making a list of distances between point and training points.
    for i,point in enumerate(training.points):
        euclid_distance=euclidDistance(test_point,point)
        #Add distance and Index of point in training set
        distances.append([euclid_distance,i])
    #Sort the array by the elements Euclidian Distance in ascending order
    distances.sort(key=lambda x: x[0])
    points,attributes = [],[]
    #Get points and attributes of k closest training points
    for i in range(k):
        points.append(training.points[distances[i][1]])
        attributes.append(training.attributes[distances[i][1]])
    return points, attributes
```

testAccuracy() function

The following return the percentage of how many predictions made by the KNN Clustering Algorithm were correct. The 'predictions' variable is an array of attributes that represent the predicted plant class of each evaluation point. The function loops through each attribute in the attributes array of the evaluation set and checks whether the prediction for the same point was true. If the prediction was the same as the actual result for the point, a variable called 'matched' is incremented, otherwise, the loop keeps iterating.

At the end of the function the number of matched predictions is divided by the number of points in the evaluation set and multiplied by 100 to obtain the accuracy of the KNN Clustering Algorithm (how many evaluation points it predicted correctly). The value is rounded by two and returned.

```
def testAccuracy(predictions, evaluation):
    matched = 0
    #Check if the data was predicted correctly
    for i,attr in enumerate(evaluation.attributes):
        if predictions[i] == attr:
            matched += 1
    return round((matched/float(len(evaluation.attributes))) * 100.0 , 2)
```

KNNClustering() function

The following function contains the implementation of the KNN clustering algorithm. The function starts by splitting the data into a training and evaluation set (or testing set) based on the training percentage that was given. Next, it iterates through each point in the evaluation set and finds the k closest points and their attributes from the training set using the 'findNeighbours()' function. Note that the k-value was passed into the function instead of using the global k-value.

Next, the most common attributes of the closest attributes is obtained using the 'getCommonAttr()' function which was also used for K-Means Clustering. This attribute is assumed to be the prediction of the attribute of the current point and is pushed into the predictions array. The following process is done for each point until the predictions array has all the predicted attributes of each of the evaluation points. In the end, the function gets the accuracy of the predictions using the 'testAccuracy()' function and then returns the accuracy.

```
def KNNClustering(k,percentage):
    #Split the data into a training and evaluation set
    training,evaluation = splitData(data,percentage)
    predictions = []
    #For each point in the evaluation set
    for i,point in enumerate(evaluation.points):
        #Find the closest points and their attributes from the training set
        neighbours, attributes = findNeighbours(point,training,k)
        #Find the most common attribute and add it to predictions
        attr = getCommonAttr(attributes)[0]
        predictions.append(attr)
    #Test the accuracy of the predictions and return
    accuracy = testAccuracy(predictions,evaluation)
    print('\tFor k = ' + repr(k) + ' and Training Percentage = ' + repr(percentage)
+ ' : Accuracy = ' + repr(accuracy) + '%')
    return accuracy
```

doKNNClustering() function

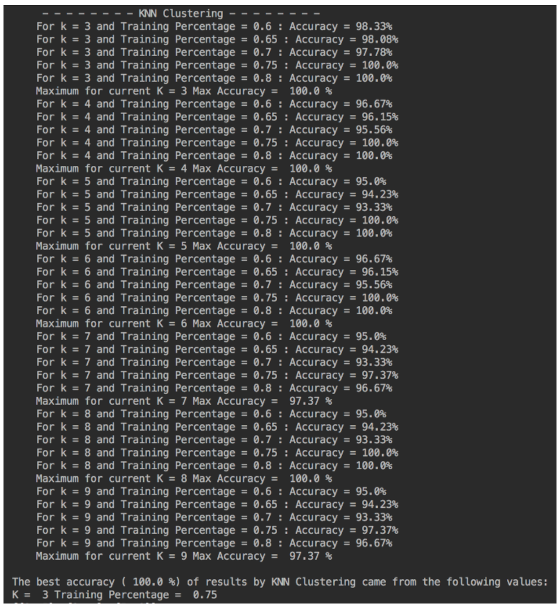
The following functions calls the 'KNNClustering()' method for different k-values and training percentages in order to determine the which k-value and training percentage give the best accuracy. Note that it was to set display results for k-values between the range of 3 – 9 and training percentages of 60%, 70%, 75% and 80% for this project but these values can be changed to suit the user.

It functions by iterating through each k-value by an enumeration of 1 and percentage by enumeration of 5. It passes the k-value and training percentage of each enumeration to the 'KNNClustering()' function and if the accuracy obtained is greater than the current highest one, the values storing the highest percentage value, k-value and accuracy are updated according to the values of the new highest accuracy. This is done for each training percentage iteration to keep track of the highest accuracy for each k-value. The same is also done for each k-value to keep track of the overall highest accuracy.

```
#Does KNN Clustering for multiple values of k and tries to find best k and training
percentage
def doKNNClustering():
    max_acc, max_k, max_train = 0.0, 0, 0.0
    print('\n\t - - - - - - - - KNN Clustering - - - - - - - - ')
    # Try a bunch of K's
    for i in range(3, 10):
        temp_acc, temp_k, temp_train = 0.0, 0, 0.0
        # Try a bunch of percentages
        for j in range(60, 81, 5):
            accuracy = KNNClustering(i, float(j) / 100)
            if (accuracy > temp_acc):
                temp_acc, temp_k, temp_train = accuracy, i, float(j) / 100
        print('\tMaximum for current K =', i, 'Max Accuracy = ', temp_acc, '%')
        if (temp_acc > max_acc):
            max_acc, max_k, max_train = temp_acc, temp_k, temp_train
```

At the end of the iterations, the highest accuracy and its k-value and training percentage are displayed to the user. Below is an example of the output displayed to the user.

```
print('\nThe best accuracy (', max_acc, '%) of results by KNN Clustering came from
the following values:')
print('K = ', max_k, 'Training Percentage = ', max_train)
```



```
         - - - - - - - - KNN Clustering - - - - - - - -
        For k = 3 and Training Percentage = 0.6 : Accuracy = 98.33%
        For k = 3 and Training Percentage = 0.65 : Accuracy = 98.08%
        For k = 3 and Training Percentage = 0.7 : Accuracy = 97.78%
        For k = 3 and Training Percentage = 0.75 : Accuracy = 100.0%
        For k = 3 and Training Percentage = 0.8 : Accuracy = 100.0%
        Maximum for current K = 3 Max Accuracy =  100.0 %
        For k = 4 and Training Percentage = 0.6 : Accuracy = 96.67%
        For k = 4 and Training Percentage = 0.65 : Accuracy = 96.15%
        For k = 4 and Training Percentage = 0.7 : Accuracy = 95.56%
        For k = 4 and Training Percentage = 0.75 : Accuracy = 100.0%
        For k = 4 and Training Percentage = 0.8 : Accuracy = 100.0%
        Maximum for current K = 4 Max Accuracy =  100.0 %
        For k = 5 and Training Percentage = 0.6 : Accuracy = 95.0%
        For k = 5 and Training Percentage = 0.65 : Accuracy = 94.23%
        For k = 5 and Training Percentage = 0.7 : Accuracy = 93.33%
        For k = 5 and Training Percentage = 0.75 : Accuracy = 100.0%
        For k = 5 and Training Percentage = 0.8 : Accuracy = 100.0%
        Maximum for current K = 5 Max Accuracy =  100.0 %
        For k = 6 and Training Percentage = 0.6 : Accuracy = 96.67%
        For k = 6 and Training Percentage = 0.65 : Accuracy = 96.15%
        For k = 6 and Training Percentage = 0.7 : Accuracy = 95.56%
        For k = 6 and Training Percentage = 0.75 : Accuracy = 100.0%
        For k = 6 and Training Percentage = 0.8 : Accuracy = 100.0%
        Maximum for current K = 6 Max Accuracy =  100.0 %
        For k = 7 and Training Percentage = 0.6 : Accuracy = 95.0%
        For k = 7 and Training Percentage = 0.65 : Accuracy = 94.23%
        For k = 7 and Training Percentage = 0.7 : Accuracy = 93.33%
        For k = 7 and Training Percentage = 0.75 : Accuracy = 97.37%
        For k = 7 and Training Percentage = 0.8 : Accuracy = 96.67%
        Maximum for current K = 7 Max Accuracy =  97.37 %
        For k = 8 and Training Percentage = 0.6 : Accuracy = 95.0%
        For k = 8 and Training Percentage = 0.65 : Accuracy = 94.23%
        For k = 8 and Training Percentage = 0.7 : Accuracy = 93.33%
        For k = 8 and Training Percentage = 0.75 : Accuracy = 100.0%
        For k = 8 and Training Percentage = 0.8 : Accuracy = 100.0%
        Maximum for current K = 8 Max Accuracy =  100.0 %
        For k = 9 and Training Percentage = 0.6 : Accuracy = 95.0%
        For k = 9 and Training Percentage = 0.65 : Accuracy = 94.23%
        For k = 9 and Training Percentage = 0.7 : Accuracy = 93.33%
        For k = 9 and Training Percentage = 0.75 : Accuracy = 97.37%
        For k = 9 and Training Percentage = 0.8 : Accuracy = 96.67%
        Maximum for current K = 9 Max Accuracy =  97.37 %

 The best accuracy ( 100.0 %) of results by KNN Clustering came from the following values:
 K =  3 Training Percentage =  0.75
```