BIG DATA MicroMasters

Major assignment 1: Your submission

This is your assignment template for BigDataX Major assignment 1. Save this document on your local machine and include all of your work within the relevant sections. Once you've completed all two parts of the assignment, upload the document via the submission area on the "Submit your assignment" page at the end of Major assignment 1.

You will need to include:

- **the answer to the questions**
- **all of your calculations/working that sufficiently justify your answers**

Your answers and calculations/working will assist the University of Adelaide academic staff member to assess your submission.

Quick links:

Major assignment 1: Part 2 Clustering

Major assignment 1: Part 3 PageRank

THE UNIVERSITY of ADELAIDE

BIG DATA MicroMasters

Major assignment 1: Part 2 Clustering

1. Your answer and all associated working/calculations to step 1 [4 points]

Your answer goes here:

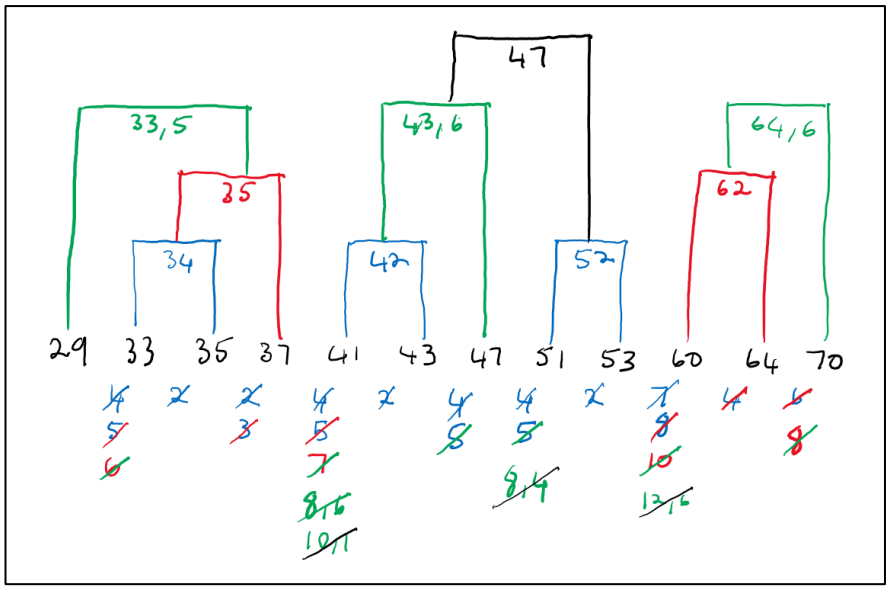

Figure 1. Clustering by closes centroid.

Your associated working/calculations:

- Clusters were manually calculated by clustering from left to right by the closes distance between cluster centroids.
- Centroids were calculated as the average of the points contained in a cluster *(mental arithmetic and pocket calculator where I doubted myself)*.
- Clustering was stopped after only 3 clusters remained.
- The manual process is error prone, thus as a bare minimum check a dendrogram was plotted with SciPy to visually confirm the clusters match.

```
data = [[i] for i in [29, 33, 35, 37, 41, 43, 47, 51, 53, 60, 64, 70]]
data_linkage = scipy.cluster.hierarchy.linkage(data, method="centroid")
dn = scipy.cluster.hierarchy.dendrogram(data_linkage, count_sort='descending',
labels=data)
```

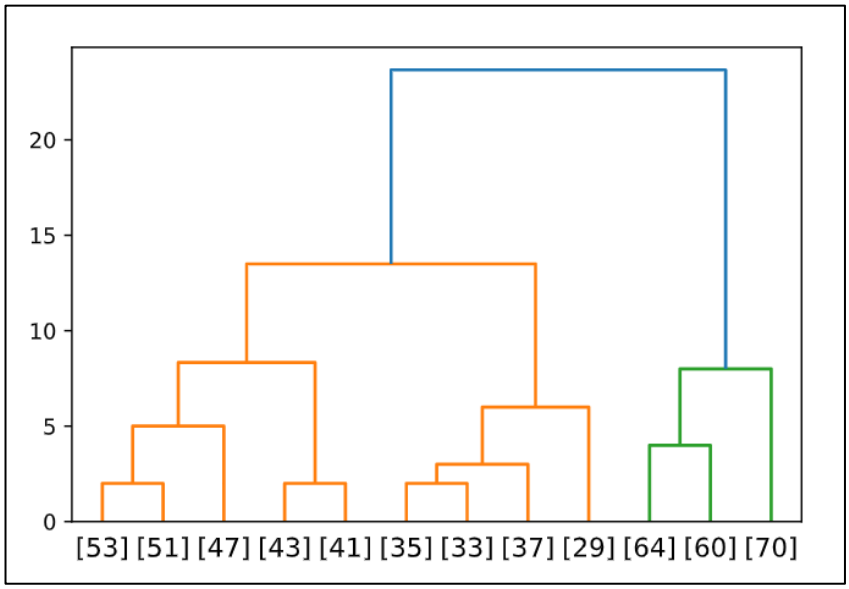

Figure 2. Closes centroid dendrogram.

2. Your answer and all associated working/calculations to step 2 [4 points]

Your answer goes here:

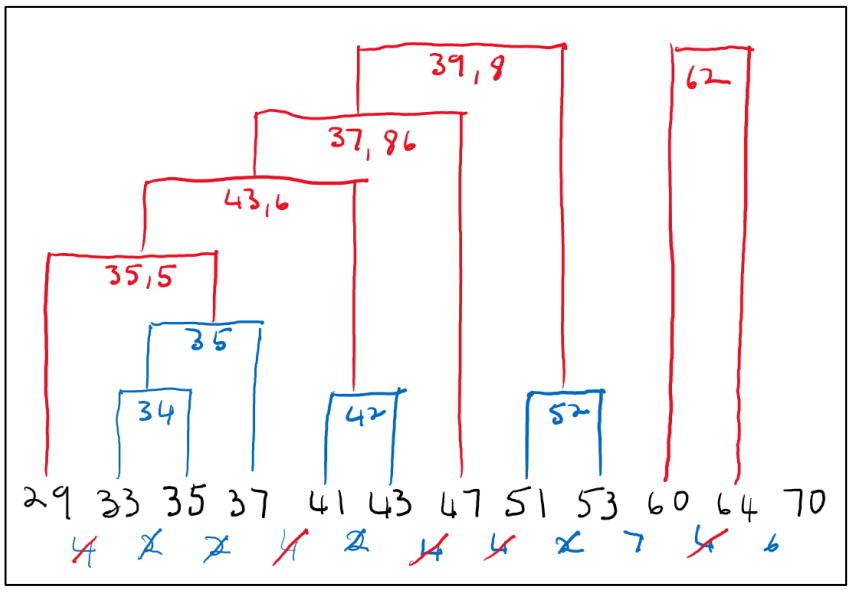

Figure 3. Clustering by minimum distance between any two points.

Your associated working/calculations:

- Clusters were manually calculated by clustering from left to right by the closes distance between two points in a cluster.
- Centroids were calculated as the average of the points contained in a cluster *(mental arithmetic and pocket calculator where I doubted myself)*.
- Clustering was stopped after only 3 clusters remained.
- The manual process is error prone, thus as a bare minimum check a dendrogram was plotted with SciPy to visually confirm the clusters match.

```
data = [[i] for i in [29, 33, 35, 37, 41, 43, 47, 51, 53, 60, 64, 70]]
data_linkage = scipy.cluster.hierarchy.linkage(data, method="single")
d = scipy.cluster.hierarchy.dendrogram(data_linkage, count_sort='descending', l
abels=data)
```

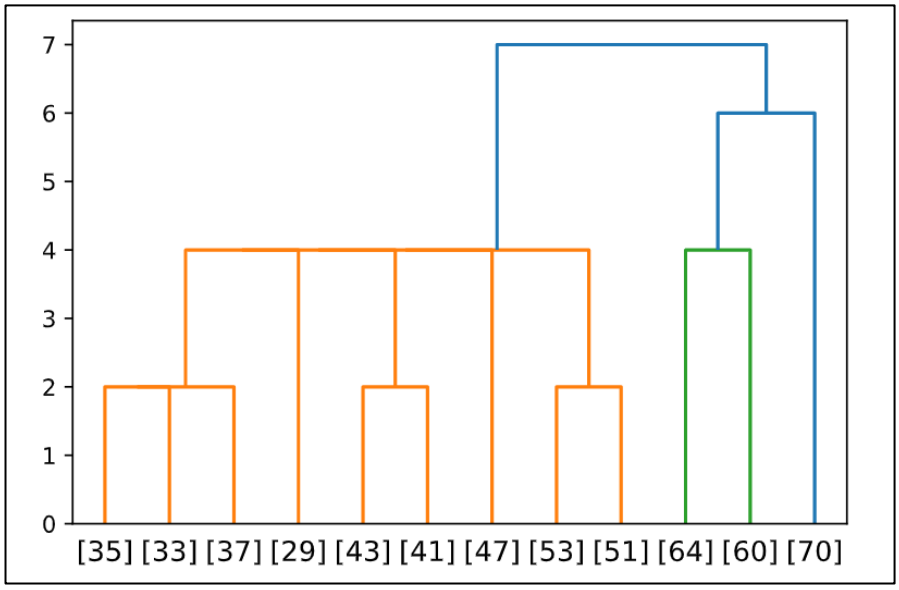

Figure 4. Closes two points dendrogram.

BIG DATA
MicroMasters

Major assignment 1: Part 3 PageRank

In the previous sections I had to continually re-work the manual calculations since there are so many opportunities for mistakes to creep in. By necessity the following class was created to aid in the calculations related to the PageRank algorithm:

```
class PageRank:
    """
    Implement the PageRank algorithm using matrix manipulation.
    """
    def __init__(self, edges):
        """
        Create a dafault instance of the class.

        Args:
            edges: The edges to create the page graph from.
        """
        self.graph = self.createGraph(edges)

    def createGraph(self, edges):
        """
        Construct a graph from the page edges.

        Args:
            edges: The edges to create the page graph from.

        Returns:
            The graph created from the input edges.
        """
        return gp.Graph.TupleList(edges, directed=True)


    def plotGraph(self, width=300, height=300, margin=15, layout='auto', edge_c
urved=True):
        """
        Plot the page graph.
        """
        return gp.plot(self.graph,
            vertex_label=self.graph.vs["name"],
            layout=layout,
            margin=margin,
            bbox=(0, 0, width, height),
```

THE UNIVERSITY of ADELAIDE

```
            edge_curved=edge_curved)

    def get_transitionmatrix(self):
        """
        Get the transation matrix of the graph.

        Returns:
            A numpy array containing the transition matrix.
        """
        # initialize the transition matrix with zeros
        node_count = len(self.graph.vs)
        transition_matrix = np.zeros([node_count, node_count])

        # get the in and out adjacency list
        out_adj = self.graph.get_adjlist(mode='out')
        in_adj = self.graph.get_adjlist(mode='in')

        # calculate the transition matrix
        for i in range(node_count):
            for j in range(node_count):
                # if there is no link from j to i, then mij equals zero
                if i not in out_adj[j]:
                    transition_matrix[i][j] = 0
                else: # mij equals 1/k, if there's a link from page j to i. k i
s the total number of outgoing links from j
                    transition_matrix[i][j] = 1.0 /  len(out_adj[j])


        return transition_matrix

    def get_initial_vector(self):
        """
        Get the initial pagerank vector v0.
        """
        node_count = len(self.graph.vs)

        vector = np.empty(node_count)
        vector.fill(1/node_count)

        return vector

    def get_distribution(self, n, beta=None):
```

```
        """
        Get the distribution after n steps.

        Args:
            n: The number of steps to perform.
            beta: The beta value to use to deal with spider traps.
        """
        M = self.get_transitionmatrix()
        v = self.get_initial_vector()

        for i in range(n):
            if beta == None:
                v = np.matmul(M, v)
            else:
                v = beta * np.matmul(M, v) + ((1-
beta) / len(v)) * np.ones(len(v))

        return v

    def get_pagerank(self):
        """
        Calculates the PageRank.

        Returns:
            A list with the PageRank values calculated from the input graph.
        """
        step = 1
        converged = False
        rankings = self.get_initial_vector()

        while not(converged):
            # get the new rankings
            new_rankings = self.get_distribution(step)

            # determine if the rankings have stabilized
            #print(new_rankings)
            converged = np.sum(np.absolute(rankings -  new_rankings)) < 1e-5
            rankings = new_rankings

            step = step + 1
            if step == 500:
                print('*** COULD NOT CONVERGE! ***')
```

```
                converged = True

        return rankings
```

BIG DATA MicroMasters

1. Your answers to Part 3a (Graph 1.) and all associated working/calculations [4 points]

```
graph_edges = [
    ['A', 'B'],
    ['A', 'C'],
    ['A', 'D'],
    ['B', 'A'],
    ['B', 'C'],
    ['C', 'A'],
    ['C', 'B'],
    ['C', 'D'],
    ['D', 'A'],
    ['D', 'C'],
]

pageRank = PageRank(graph_edges)
pageRank.plotGraph(width=200, height=200, layout='circle', edge_curved=False)
```

Transition matrix goes here:

```
pageRank.get_transitionmatrix()
```

```
array([[0.    , 0.5   , 0.3333, 0.5   ],
       [0.3333, 0.    , 0.3333, 0.    ],
       [0.3333, 0.5   , 0.    , 0.5   ],
       [0.3333, 0.    , 0.3333, 0.    ]])
```

Initial PageRank vector goes here:

```
pageRank.get_initial_vector()
```

```
array([0.25, 0.25, 0.25, 0.25])
```

THE UNIVERSITY of ADELAIDE

First matrix-vector multiplication goes here:

```
pageRank.get_distribution(1)
```

```
array([0.3333, 0.1667, 0.3333, 0.1667])
```

Second matrix-vector multiplication goes here:

```
pageRank.get_distribution(2)
```

```
array([0.2778, 0.2222, 0.2778, 0.2222])
```

Third matrix-vector multiplication goes here:

```
pageRank.get_distribution(3)
```

```
array([0.3148, 0.1852, 0.3148, 0.1852])
```

2. Your answer to Part 3b (Graph 2.) and explanation to justify your answer [2 points]

```
# create the graph
graph_edges = [
    ['A', 'B'],
    ['A', 'C'],
    ['A', 'D'],
    ['B', 'A'],
    ['B', 'C'],
    ['B', 'D'],
    ['C', 'A'],
    ['C', 'B'],
    ['C', 'D'],
    ['D', 'A'],
    ['D', 'B'],
    ['D', 'C']
]

pageRank = PageRank(graph_edges)
pageRank.plotGraph(width=200, height=200, layout='circle', edge_curved=False)
```

Your answer goes here:

```
pageRank.get_pagerank()
```

```
array([0.25, 0.25, 0.25, 0.25])
```

Your explanation to justify your answer goes here:

Even without performing any calculations one can intuitively see that the final PageRank will be equal to V_0 since the graph is unidirectional with every node connecting to all other nodes. Thus every page has an equal probability to be landed on by the hypothetical random surfer.

3. **Your answers to Part 3c (Graph 3.) and all associated working/calculations** [4 points]

```
# create the graph
graph_edges = [
    ['A', 'B'],
    ['A', 'C'],
    ['A', 'D'],
    ['B', 'A'],
    ['B', 'C'],
    ['B', 'F'],
    ['C', 'A'],
    ['C', 'B'],
    ['C', 'D'],
    ['C', 'E'],
    ['D', 'A'],
    ['D', 'C'],
    ['D', 'E']
]

pageRank = PageRank(graph_edges)
pageRank.plotGraph(width=200, height=200, layout='circle', edge_curved=False)
```

Transition matrix goes here:

```
pageRank.get_transitionmatrix()
```

```
array([[0.    , 0.3333, 0.25  , 0.3333, 0.    , 0.    ],
       [0.3333, 0.    , 0.25  , 0.    , 0.    , 0.    ],
       [0.3333, 0.3333, 0.    , 0.3333, 0.    , 0.    ],
       [0.3333, 0.    , 0.25  , 0.    , 0.    , 0.    ],
       [0.    , 0.3333, 0.    , 0.    , 0.    , 0.    ],
       [0.    , 0.    , 0.25  , 0.3333, 0.    , 0.    ]])
```

Initial PageRank vector goes here:

```
pageRank.get_initial_vector()
```

```
array([0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.1667])
```

First matrix-vector multiplication goes here:

```
pageRank.get_distribution(n=1, beta=0.8)
```

```
array([0.1556, 0.1111, 0.1667, 0.1111, 0.0778, 0.1111])
```

Second matrix-vector multiplication goes here:

```
pageRank.get_distribution(n=2, beta=0.8)
```

```
array([0.1259, 0.1081, 0.1341, 0.1081, 0.063 , 0.0963])
```

Third matrix-vector multiplication goes here:

```
pageRank.get_distribution(n=3, beta=0.8)
```

```
array([0.1178, 0.0937, 0.1246, 0.0937, 0.0622, 0.089 ])
```