Clustering

Manuel Mauch, Marius Köppel

DM 17/18

12. 12. 2017

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Inhaltsverzeichnis

JG|U

Agglomerative and Divisive Clustering

The goal of this task was to create three dentograms for each dataset with R and the provided algorithmus Agnes and Diana. First we will present this algorithmus and then we will discuss the dentograms of each dataset.

Agnes

- Blabal
- Blabal

Diana

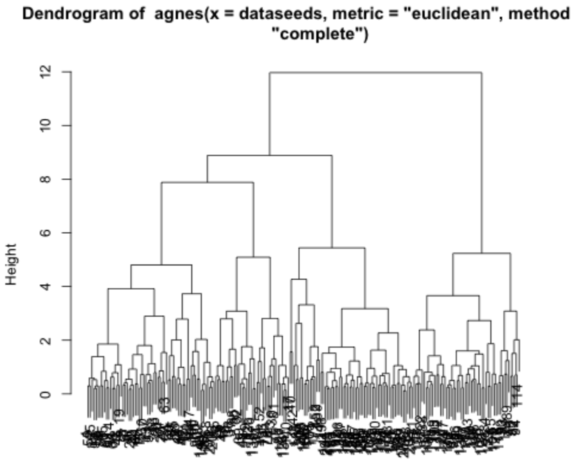
- Blabal
- Blabal

S2 from S-sets

For this dataset we were not able to create to plots, because it took to much time for proceed.

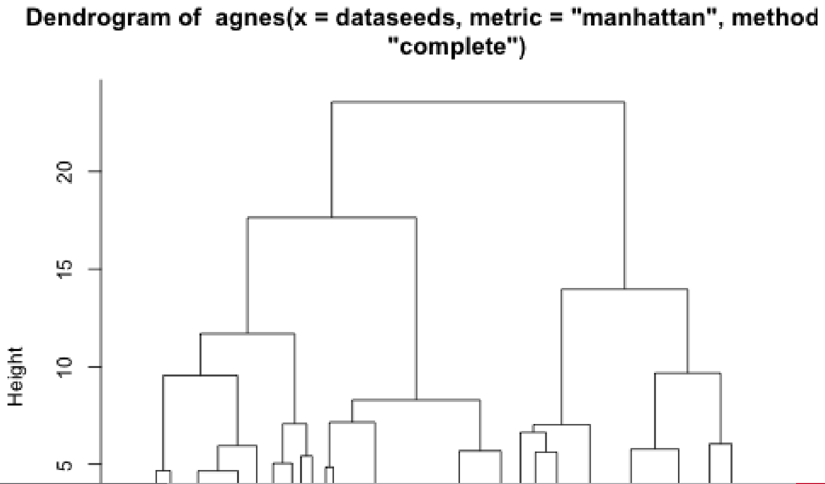
Seeds

Detaset Seeds



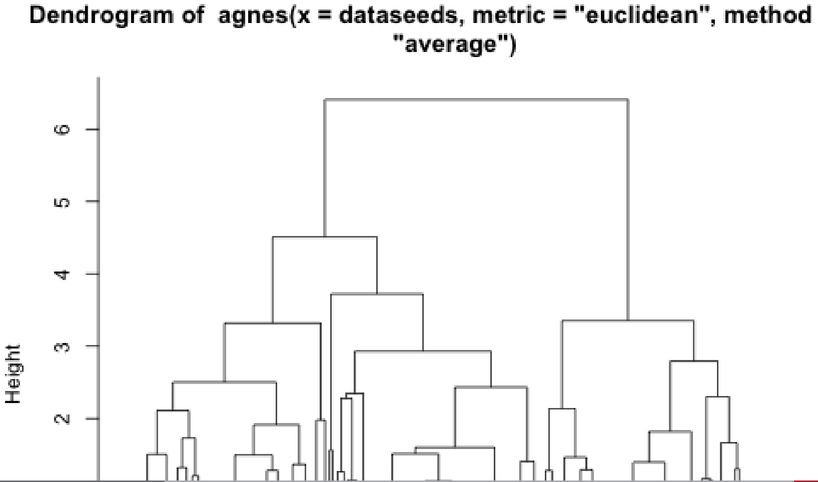
Seeds

Detaset Seeds



Seeds

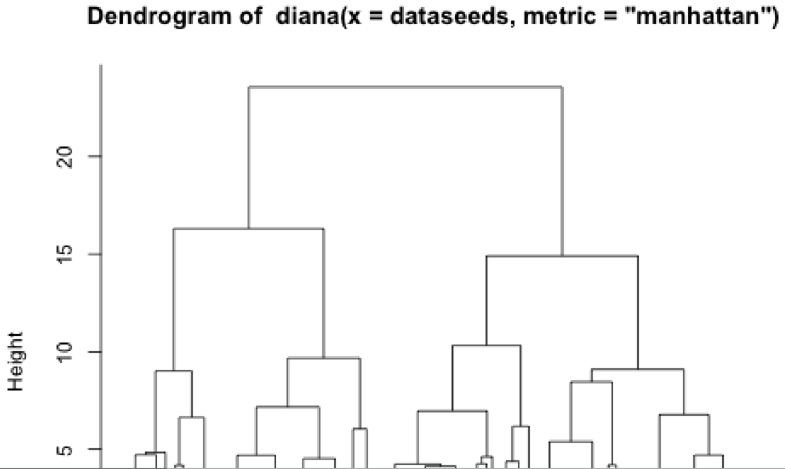
Detased Seeds



JG|U

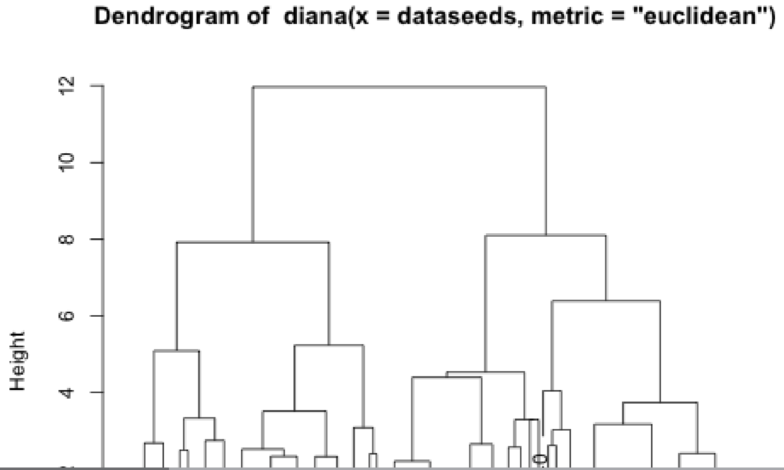
Seeds

Detaset Seeds



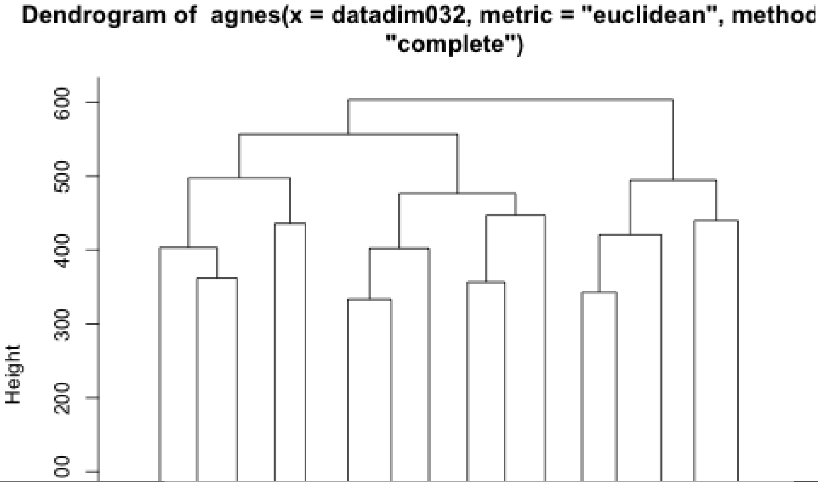
Seeds

Detaset Seeds



Dim032 from DIM-sets (high)

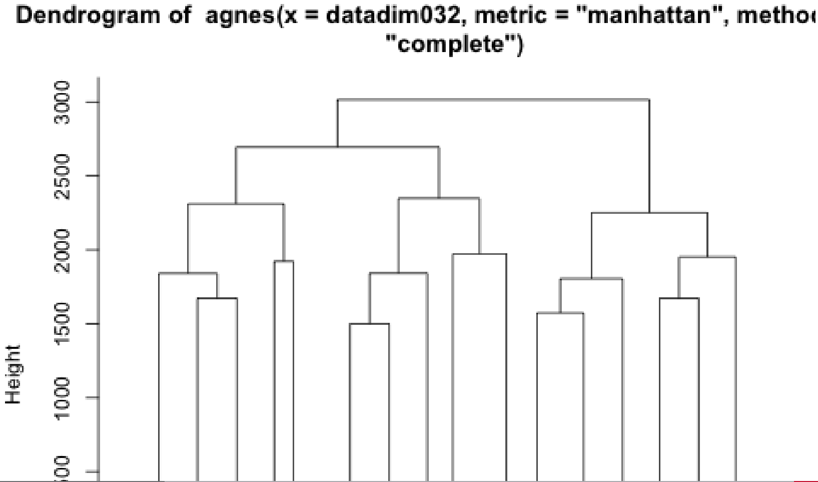
Detaset Dim032



JG|U

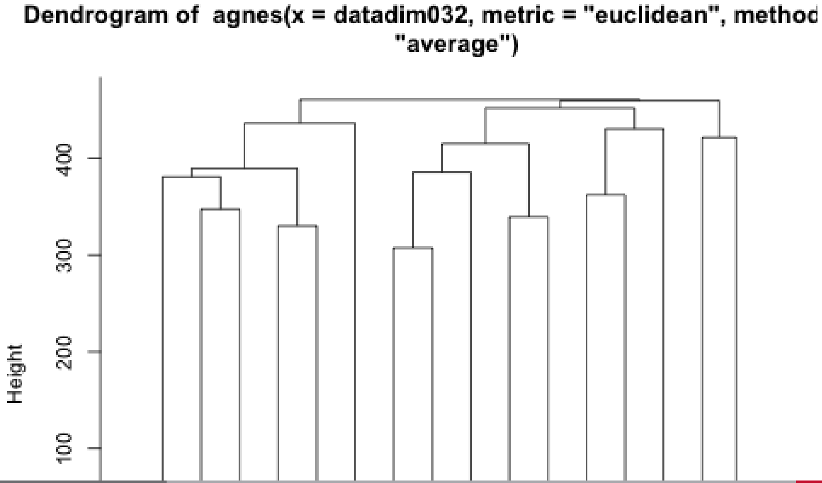
Dim032 from DIM-sets (high)

Detaset Dim032



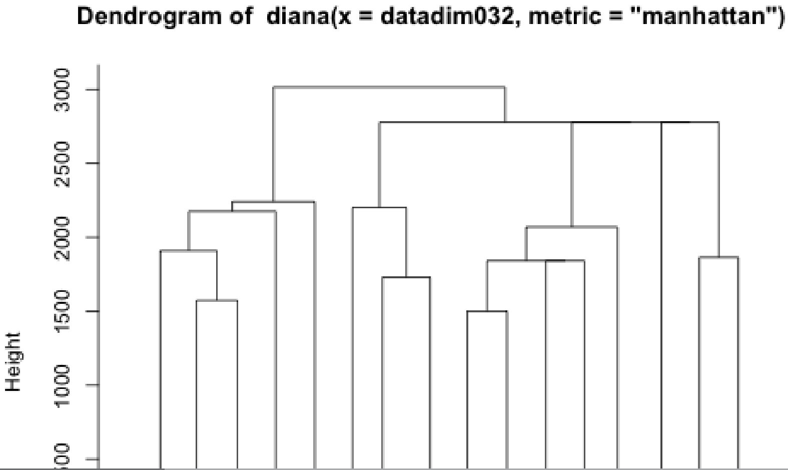
Dim032 from DIM-sets (high)

Detaset Dim032



Dim032 from DIM-sets (high)

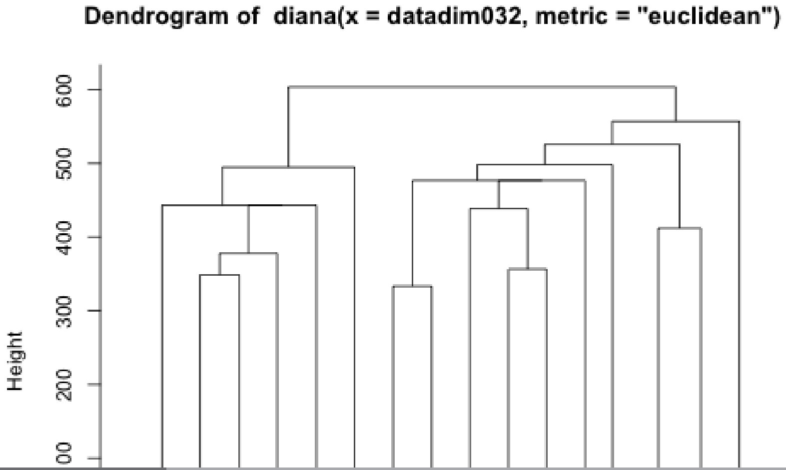
Detaset Dim032



JG|U

Dim032 from DIM-sets (high)

Detaset Dim032



Conclusion

TODO (MANUEL): Was ist hight in dentogram?, Wie gehen die Algorithmen und Interpretation der unterschiedlichen Teile.

JG|U

K-means

```
argsys = sys.argv
# Input of the program
filename = argsys[2]
maxk = argsys[1]
# Read the file
list_of_list = []
with open(filename,'r') as rf:
    for line in rf:
        tokens = line.split('\t')
        tokens = [float(t) for t in tokens]
        list_of_list.append(tokens)
```

JG|U

K-means

```python
# Generate a set with key=tCluster, value=point
true_cluster = {}
for i in range(nTrueClusters):
    true_cluster[i+1] = []
for key in true_cluster.keys():
    for point in list_of_list:
        if point[2] == key:
            true_cluster[key].append([point[0], point[1]])
```

K-means

```
# Iterate until maxk
for k in range(int(maxk)):
   # Generate random centers
    random_centers = []
    for i in range(0,int(k+1)-1):
        random_centers.append(random.choice(list_of_list))
    # Get distance for the random centers
    dic = getDistanceDic(list_of_list, random_centers)
    # Get current clusters
    current_cluster = getDicOfCurrentCluster(k+1, dic,
        list_of_list)
    # Get current centers
    current_centers = getCurrentCenters(current_cluster)
    # Set check for while loop true
    checkCenters = True
```

K-means

```
# In for-loop
while checkCenters:
     # Generate old_centers
       old_centers = [centers for centers in
           current_centers]
       # Get distance for the random centers
       dic = getDistanceDic(list_of_list, current_centers)
       # Get current clusters
       current_cluster = getDicOfCurrentCluster(k+1, dic,
           list_of_list)
       # Get current centers
       current_centers =
           getCurrentCenters(current_cluster)
       # Check while
       if old_centers == current_centers:
           checkCenters = False
```

K-means

```
# Get distance of current centers
def getDistanceDic(list_of_list, centers):
    dic = {}
    for i in range(len(list_of_list)):
        temp_distance = []
        for point in centers:
            temp_distance.append(sqrt((list_of_list[i][0]-point[0
                + (list_of_list[i][1]-point[1])**2))
        dic[i] = temp_distance
    return dic
```

K-means

```
# Get dic of current clusters
def getDicOfCurrentCluster(k, dic, list_of_list):
    current_cluster = {}
    for i in range(int(k)):
        current_cluster[i] = []
    for key in dic.keys():
        current_cluster[dic[key].index(min(dic[key]))].append([li
            list_of_list[key][1]])
    return current_cluster
```

K-means

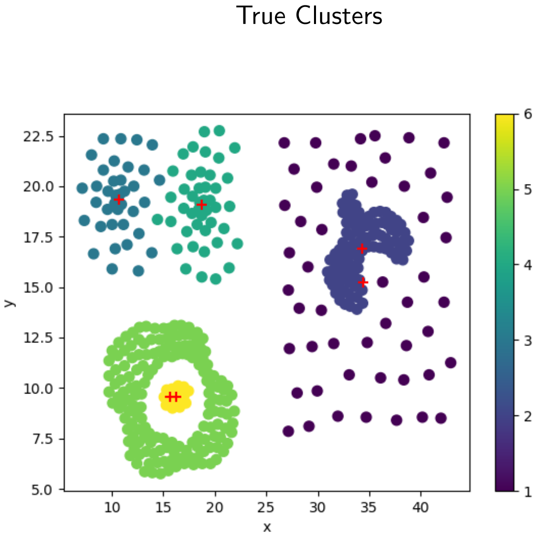
```
# Generate current centers
def getCurrentCenters(current_cluster):
    current_centers = []
    for key in current_cluster.keys():
        x, y = zip(*current_cluster[key])
        current_centers.append([sum(x) / len(x), sum(y) /
            len(y)])
    return current_centers
```

K-means

Data	RAND	norm. mutual info.	purity of clusters
jain.txt	2	6	1
compound.txt	3	3	1

Set S2 had no lable so it was not possible to calculate the best k

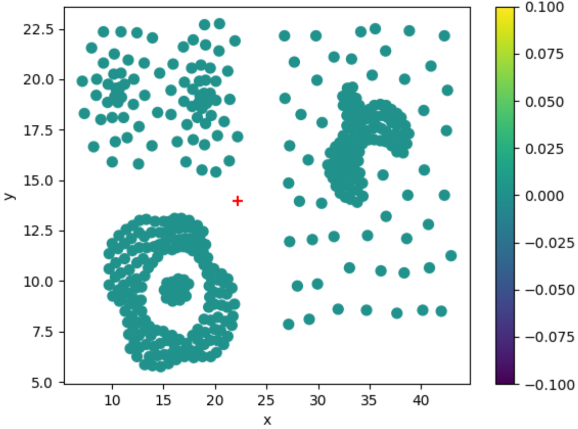
K-means


True Clusters
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
6
5
4
3
2
1

JG|U

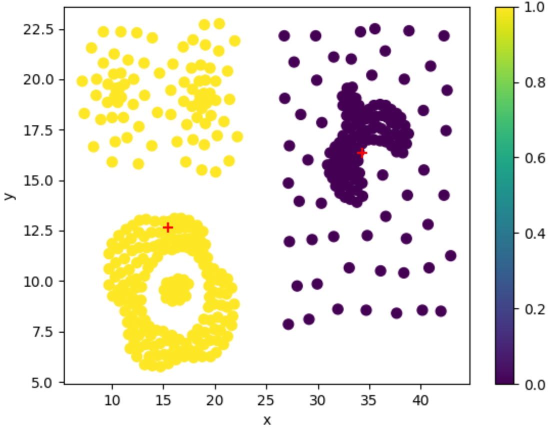
K-means

$$k = 1$$



K-means

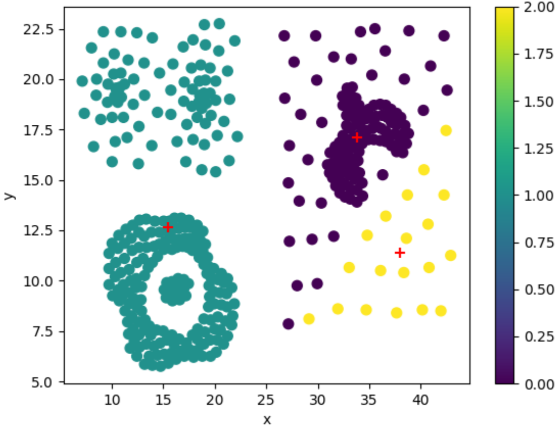
$k = 2$


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
1.0
0.8
0.6
0.4
0.2
0.0

JG|U

K-means

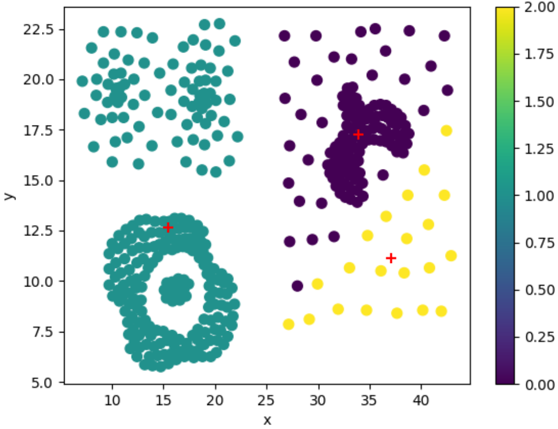
k = 3



JG|U

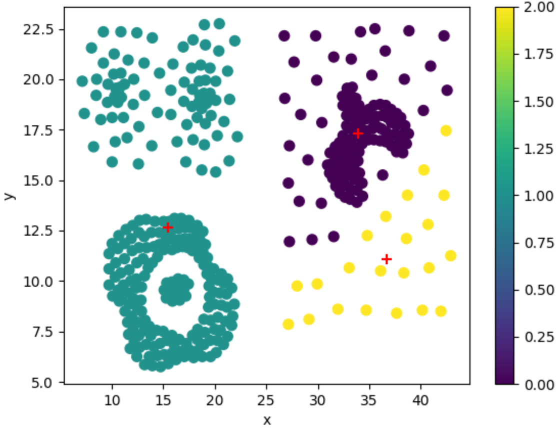
K-means

$$k = 3$$



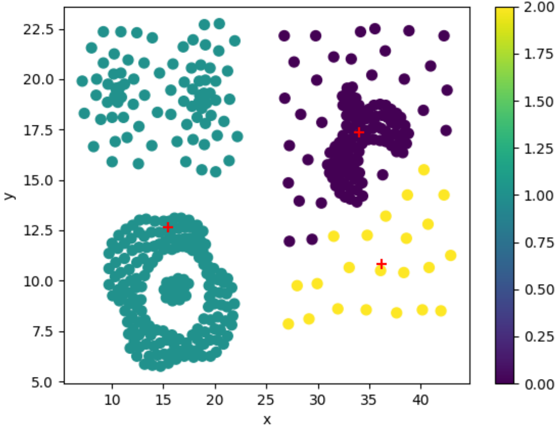
K-means

$k = 3$


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

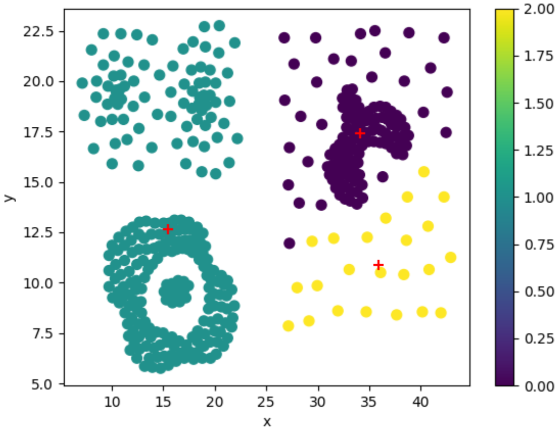
K-means

$k = 3$


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

K-means

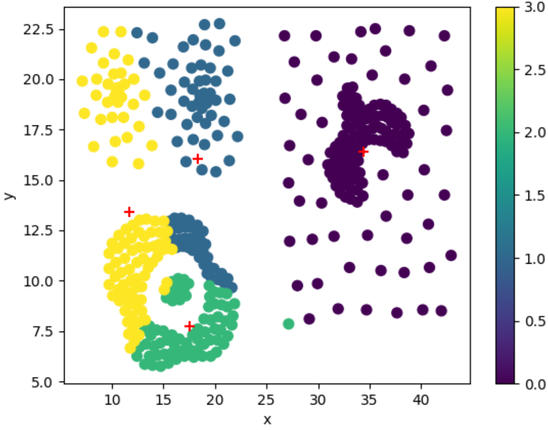
$k = 3$


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

JG|U

K-means

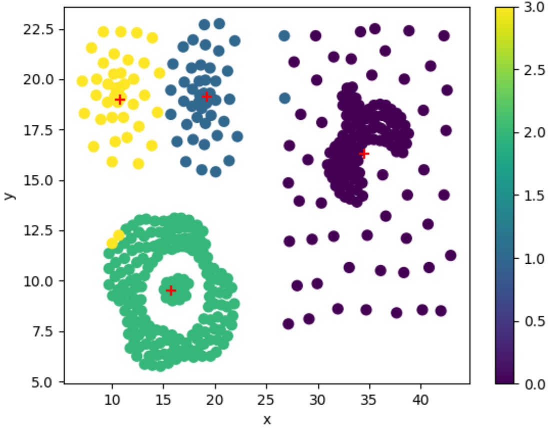
$\mathrm{k} = 4$



JG|U

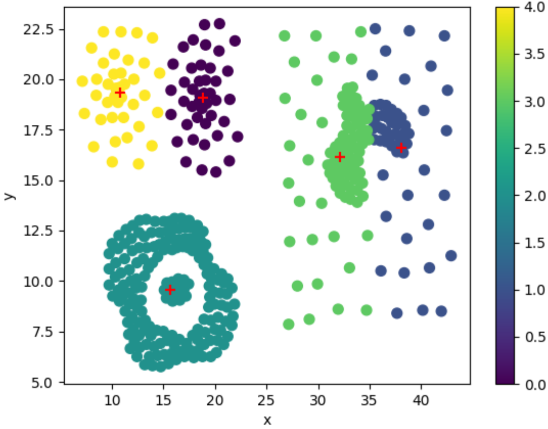
K-means

$k = 4$



K-means

$$k = 5$$


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
y
10
15
20
25
30
35
40
x
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

EM

```
# Read the date and set the parameters
lable, data = read("EM-data.csv")
parameters = {'mu1': 1., 'sig1': 1., 'mu2': 4., 'sig2':
    1., 'p': 0.5}
k = 2 # We had only 2 clusters here
# Run the main program
for i in range(20):
    print ("#########")
    print ("Step "+str(i))
    print (logLikelihood(data, k, parameters))
    print (parameters)
    print ("\n")
    wa, wb = eStep(data, parameters)
    parameters = mStep(wa, wb, data, parameters)
```

EM

```
# Function for validation
def logLikelihood(data, k, parameters):
    logLikeli = 0
    for x in data:
        logLikeli += np.log(parameters["p"] *
            gaus(parameters["sig1"], parameters["mu1"], x)
                                + parameters["p"] *
                                    gaus(parameters["sig2"],
                                    parameters["mu2"], x))
    return logLikeli
```

```
def gaus(sigma, mu, x):
    return 1./(np.sqrt(2. * math.pi) * sigma) *
        math.exp(-(x - mu)**2. / (2. * sigma**2.))
```

EM

```
def eStep(data, parameters):
  wa = []
  wb = []
  for i in range(len(data)):
      wai = gaus(parameters["sig1"], parameters["mu1"],
          data[i]) * parameters["p"] / \
           (gaus(parameters["sig1"], parameters["mu1"],
               data[i]) * parameters["p"] +
            gaus(parameters["sig2"], parameters["mu2"],
                data[i]) * parameters["p"])
      wa.append(wai)
      wb.append(1 - wai)
  return wa, wb
```

JG|U

EM

```
def mStep(wa, wb, data, parameters):
  parameters['mu1'] = getMu(data, wa)
  parameters['mu2'] = getMu(data, wb)
  parameters['sig1'] = getSig(data, wa, parameters['mu1'])
  parameters['sig2'] = getSig(data, wa, parameters['mu2'])
  return parameters
```

JG|U

EM

```
def getMu(data, ws):
    zaehler = 0
    nenner = 0
    for x, w in zip(data, ws):
        zaehler += float(x*w)
        nenner += float(w)
    return zaehler/nenner
```

EM

```
def getSig(data, ws, mu):
    zaehler = 0
    nenner = 0
    for x, w in zip(data, ws):
        zaehler += float(w * (x - mu)**2)
        nenner += float(w)
    return float(np.sqrt(zaehler/nenner))
```

JG|U

EM

sig1	sig2	mu1	mu2
1.32	7.91	16.93	24.73

Value afer 20 steps: -1193.21