k-means
Partitioning Around Medoids (PAM)

Aprendizagem de máquina

Henrique Fan

Universidade Federal do Pampa - UNIPAMPA

Introdução

Estes algoritmos são usados para particionar dados em grupos, com a tentativa de minimizar as distâncias entre os pontos do grupo e um ponto que representa o centro desse conjunto.

K-means

K-means

```
x, y = readFile(k, file)
c, cx, cy = createCentroids(x, y, k)
for i in range(50):
    cx, cy = calCentroids(x, y, c, cx, cy, k)
    c = updateCluster(x, y, c, cx, cy, k)
```

K-means

```
def createCentroids(x, y, k):

    cx = []
    cy = []

    #centroide começa como algum ponto aleatório
    for i in range(k):
        cx.append(x[random.randint(0, len(x))])
        cy.append(y[random.randint(0, len(y))])

    #pontos começam em um cluster aleatório
    c = []
    for i in range(len(x)):
        c.append(random.randint(0, k-1))

    return c, cx, cy
```

K-means

```
def calCentriods(x, y, c, cx, cy, k):

    for i in range(k): #todos os clusters
        a = 0
        sumX = 0.0
        sumY = 0.0
        for j in range(len(c)): #todos os pontos
            if i == c[j]:
                sumX += x[j]
                sumY += y[j]
                a += 1
        print(i, a)
        if a != 0:
            cx[i] = sumX/a
            cy[i] = sumY/a

    return cx, cy
```

K-means

```
def updateCluster(x, y, c, cx, cy, k):

    for i in range(len(c)): #todos os pontos
        m = float('inf')
        for j in range(k): #todos os centroides
            distance = d(x[i], y[i], cx[j], cy[j])
            if distance < m:
                # print(i, d(x[i], y[i], cx[j], cy[j]), j)
                m = distance
                c[i] = j
    return c
```

- **DISTÂNCIA EUCLIDIANA**

PAM

PAM

```
x, y = readFile(k, file)

c, mx, my = createMedoids(x, y, k)

c = updateCluster(x, y, c, mx, my, k)

allowableLossCost = 10

while allowableLossCost > 0:
    for i in range(0, k):
        for j in range(0, len(c)):
            if c[j] == i:
                if (mx[i] != x[j]) and (my[i] != y[j]):
                    mx_old = mx[i]
                    my_old = my[i]
                    previousCost = calCost(x, y, c, mx, my, k)
                    mx[i] = x[j]
                    my[i] = y[j]
                    c = updateCluster(x, y, c, mx, my, k)
                    newCost = calCost(x, y, c, mx, my, k)

                    if previousCost < newCost:
                        allowableLossCost -= 1
                        mx[i] = mx_old
                        my[i] = my_old
                        c = updateCluster(x, y, c, mx, my, k)
```

PAM

```
def createMedoids(x, y, k):

    mx = []
    my = []

    for i in range(0, k):
        a = random.randint(0, len(x))
        mx.append(x[a])
        my.append(y[a])

    #pontos começam em um cluster aleatório
    c = []
    for i in range(0, len(x)):
        c.append(random.randint(0, k-1))

    return c, mx, my
```

PAM

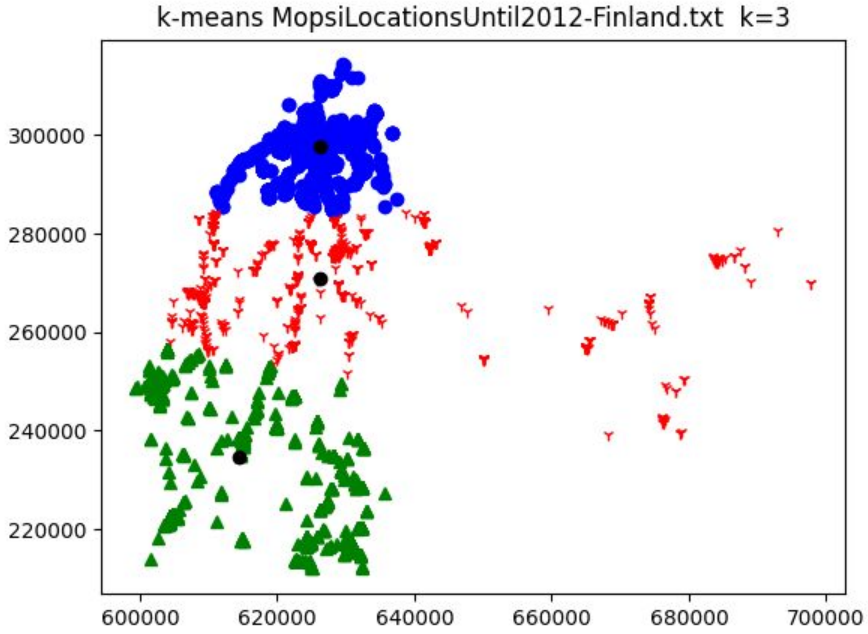
```
def updateCluster(x, y, c, mx, my, k):

    for i in range(0, len(c)): #todos os pontos
        m = float('inf')
        for j in range(0, k): #todos os medoides
            distance = d(x[i], y[i], mx[j], my[j])
            if distance < m:
                # print(i, d(x[i], y[i], cx[j], cy[j]), j)
                m = distance
                c[i] = j
    return c
```

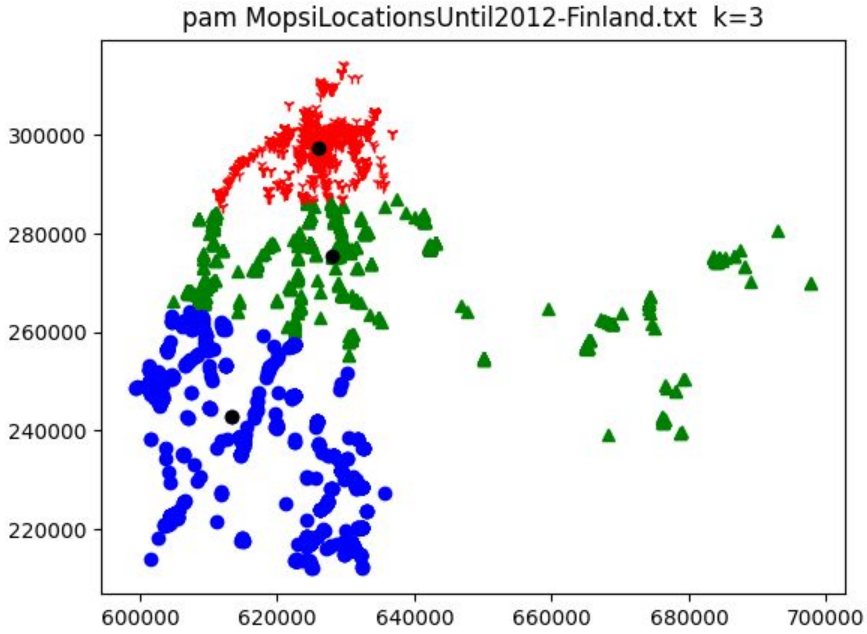
PAM

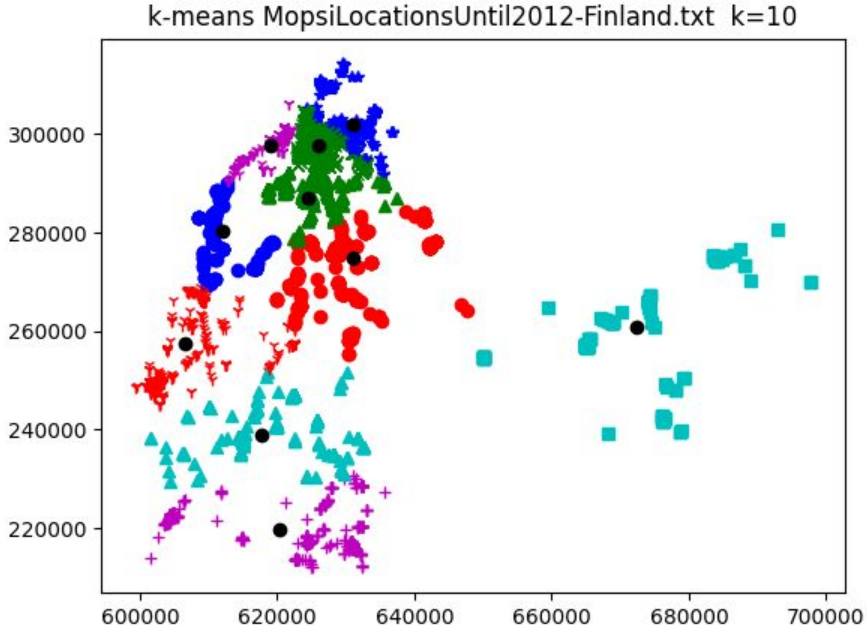
```
def calCost(x, y, c, mx, my, k):

    cost = 0.0
    for i in range(0, k):
        costMedoid = 0.0
        for j in range(len(c)):
            if c[j] == i:
                costMedoid += d(mx[i], my[i], x[j], y[j])
        cost += costMedoid

    return cost
```

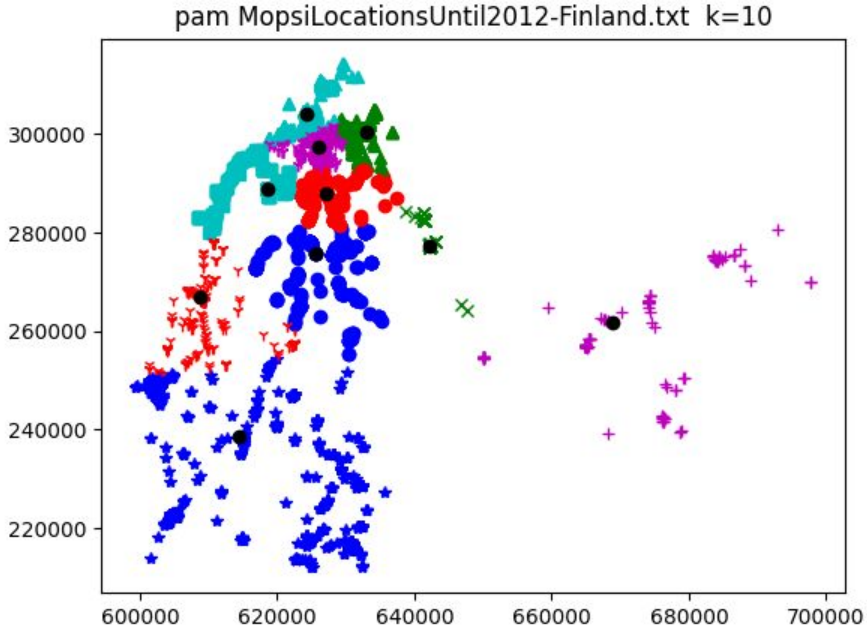
- **DISTÂNCIA EUCLIDIANA**

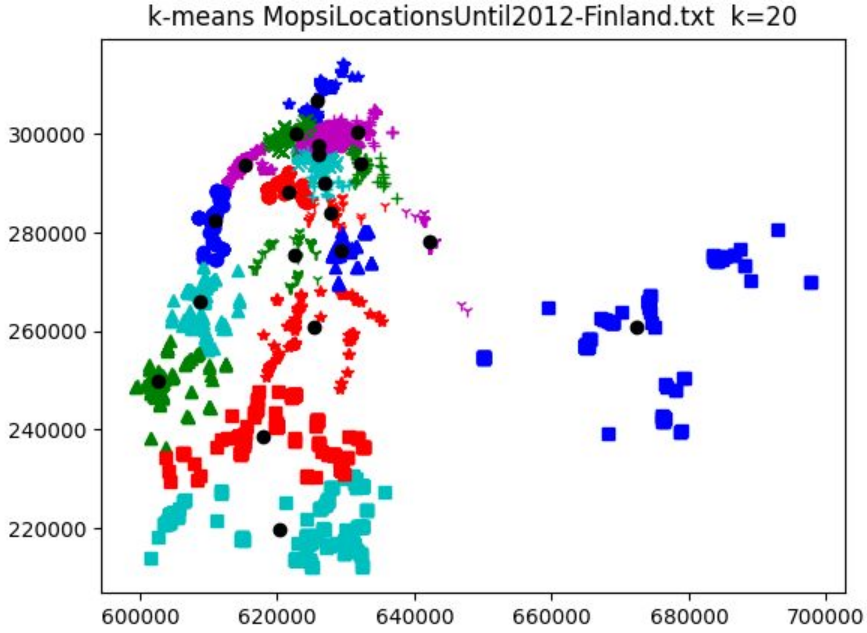
Resultados

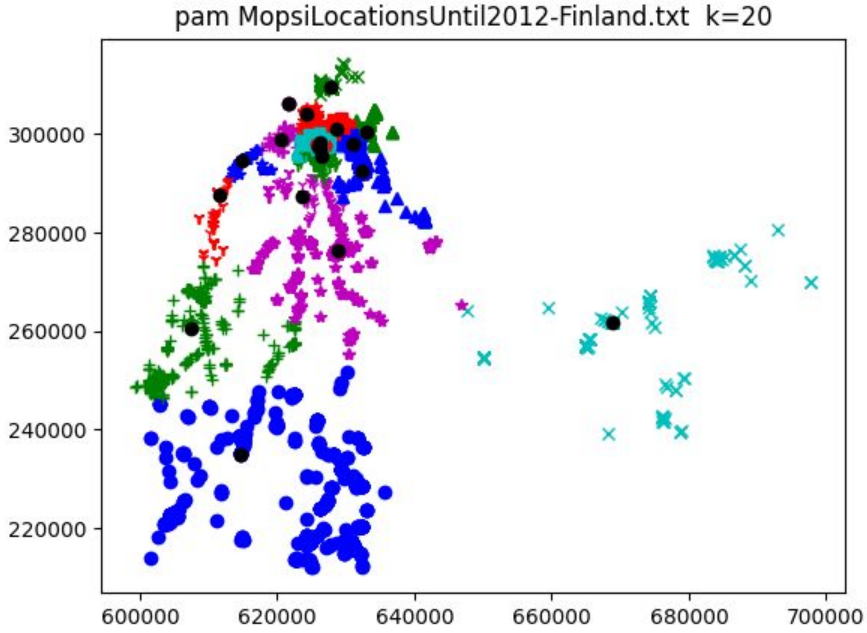

k-means MopsiLocationsUntil2012-Finland.txt k=3
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

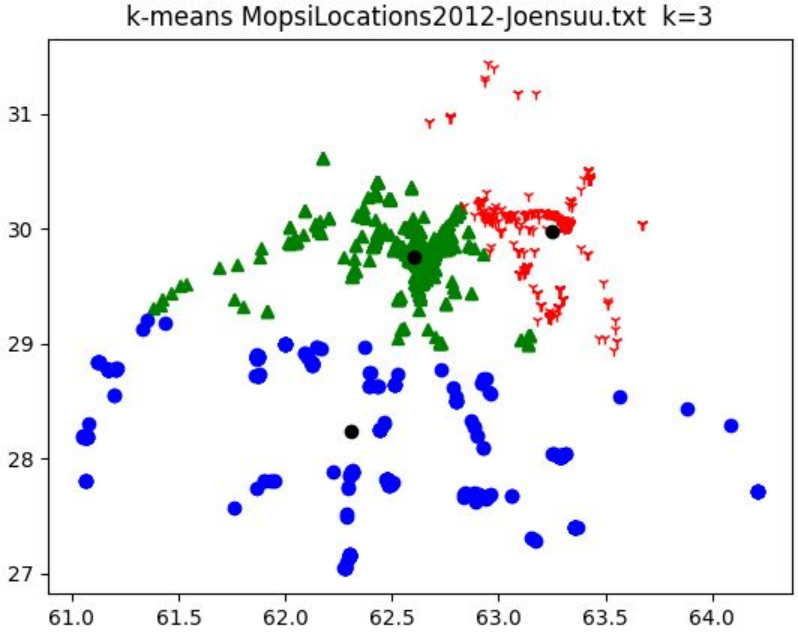

pam MopsiLocationsUntil2012-Finland.txt k=3
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

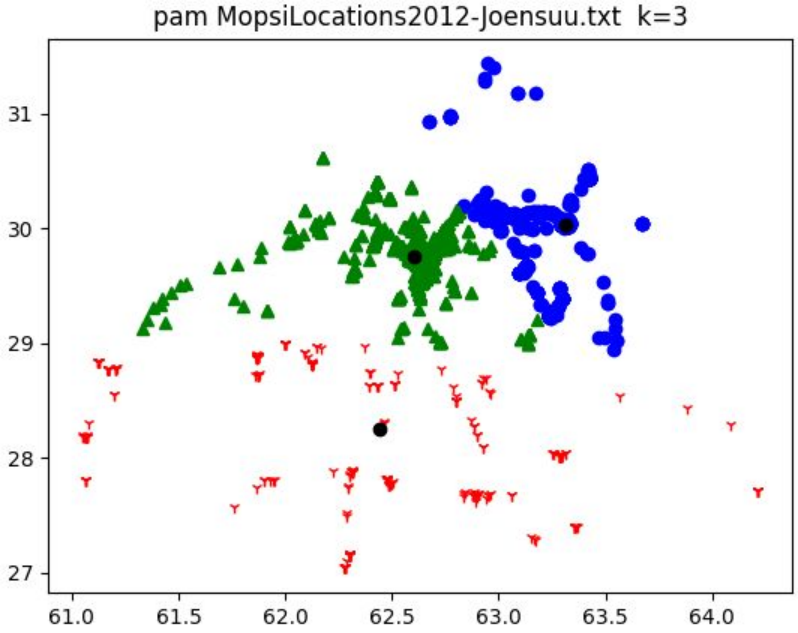

k-means MopsiLocationsUntil2012-Finland.txt k=10
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

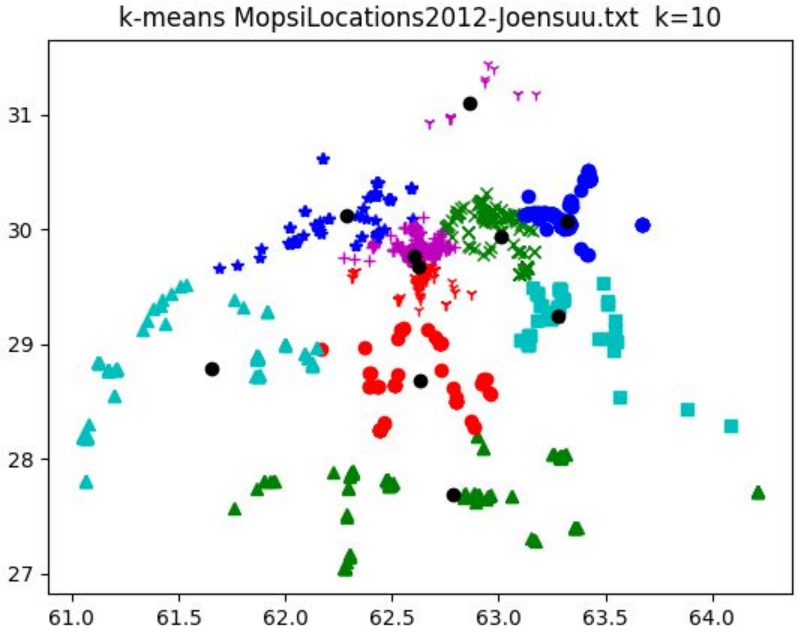

pam MopsiLocationsUntil2012-Finland.txt k=10
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

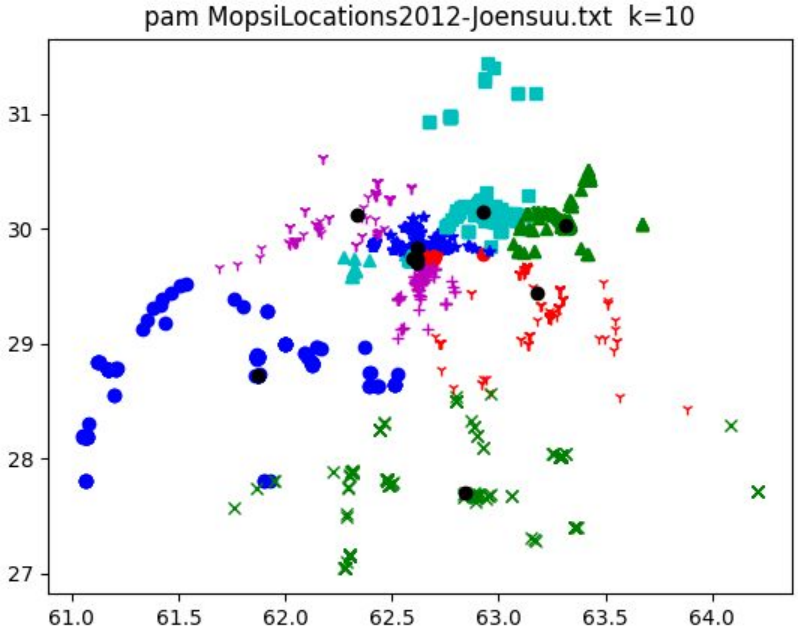

k-means MopsiLocationsUntil2012-Finland.txt k=20
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

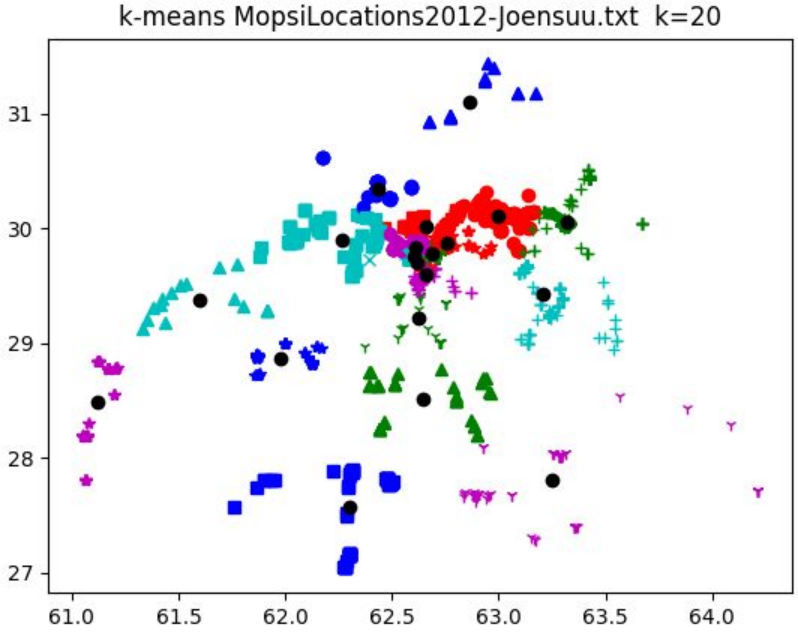

pam MopsiLocationsUntil2012-Finland.txt k=20
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

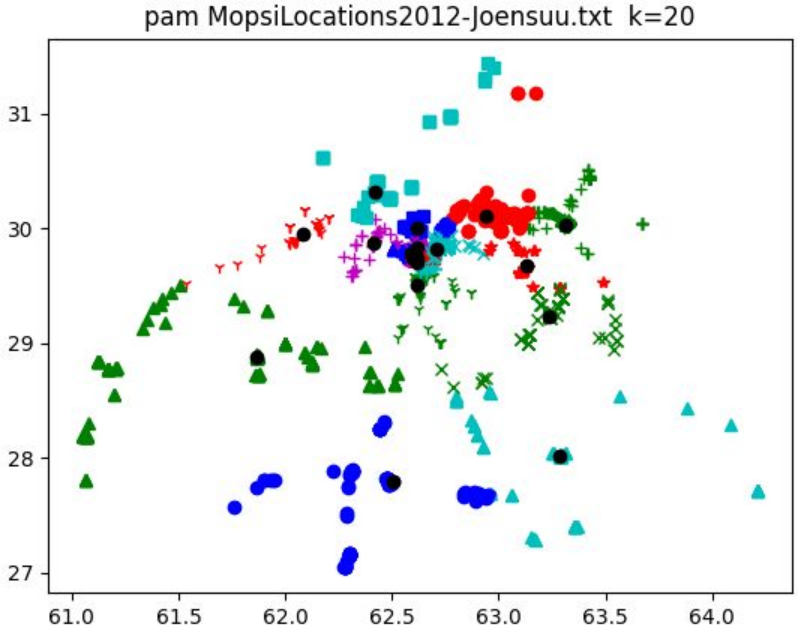

k-means MopsiLocations2012-Joensuu.txt k=3
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


pam MopsiLocations2012-Joensuu.txt k=3
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


k-means MopsiLocations2012-Joensuu.txt k=10
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


pam MopsiLocations2012-Joensuu.txt k=10
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


k-means MopsiLocations2012-Joensuu.txt k=20
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


pam MopsiLocations2012-Joensuu.txt k=20
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0

Obrigado!