IIT INDORE

1/12/2019

PROJECT REPORT

Optimizing Machine Learning algorithms in constrained environments (K-Means Clustering Algorithm)

Submitted by-

1) Seemandhar Jain
   (170001046)
2) Akshit Khatgarh
   (170001004)

Submitted To-

Dr. Kapil Ahuja

Associate Professor in C.S.E

. IIT INDORE

Aditya Anand Shastri

Siddharth Gupta

Introduction:

The more data volume increases in complexity and diversity, the harder it is to structure and manipulate them. Searching and labeling data with similar characteristics has become one of the biggest challenges in analyzing today's applications. Grouping data based on common characteristics we call clustering. Clustering algorithms fall into the unsupervised classification technique category. They classify a group of objects into subsets of groups based on similarities between them. The differences between groups have to be expressed clearly and clearly.

Clustering can be applied to a wide range of domains such as: marketing (market analysis and recommendations, methodological weaknesses), medicine (medical image segmentation), e-business (comments analysis on news portals) or e-learning (students' educational Prediction) performance).

There is no secret recipe for choosing the best clustering algorithm. The choice should be based on experimental studies and data descriptions, unless there is a clear mathematical model.

The standard K-Means algorithm represents one of the most popular unprocessed exclusive clustering algorithms. K-Means is based on reducing the average squared Euclidean distance between the data item and the center of the cluster (called centroid). The results of the algorithm are influenced by the initial centroids. Different initial configurations can give rise to different end groups. The center of the cluster is defined as the medium of the item in the cluster.

Standard K- means Method:

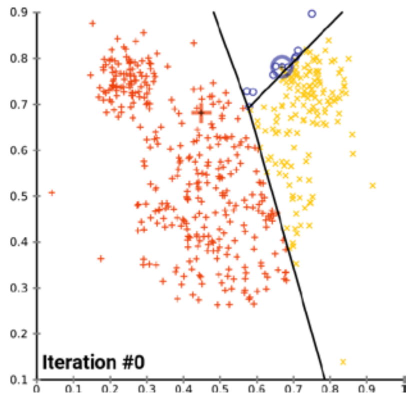
1. Choose k data objects representing the cluster centroids;

2. Assign each data object of the entire data set to the cluster having the closest centroid

3. Compute new centroid for each cluster, by averaging the data objects belonging to the cluster

4. If at least one of the centroids has changed, go to step 2, otherwise go to step 5

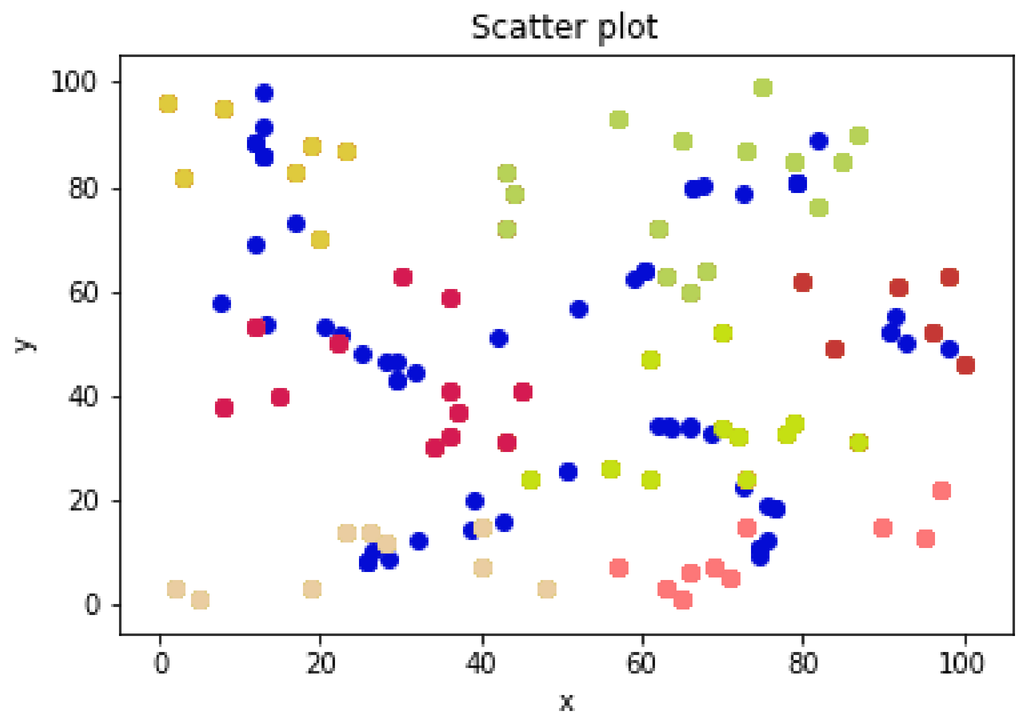
5. Output the clusters.

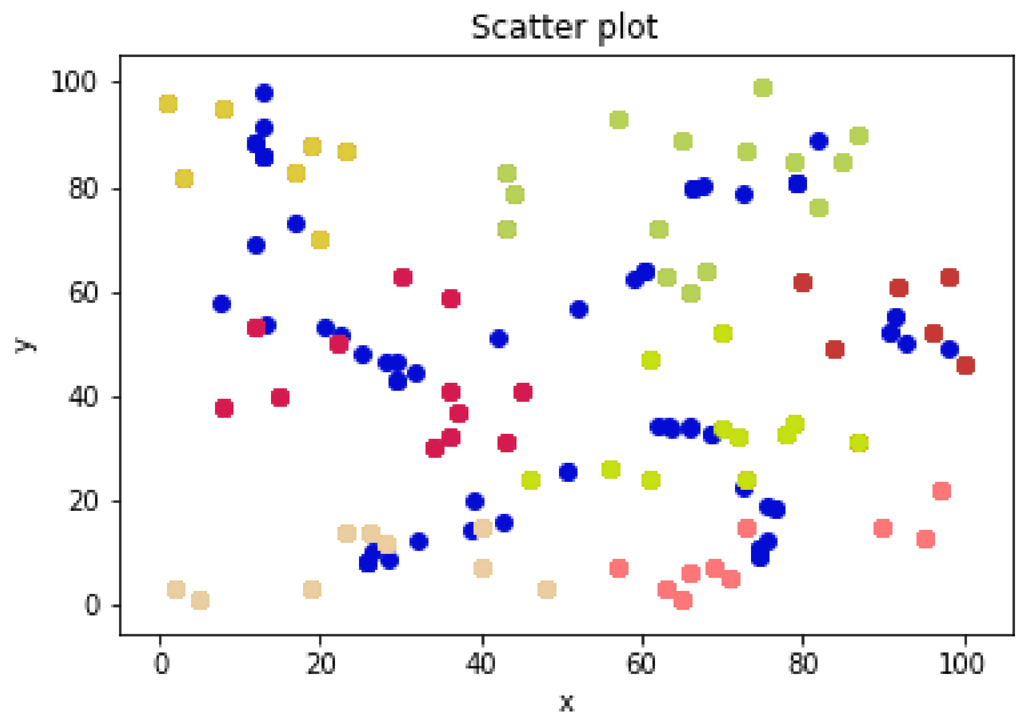


Code-

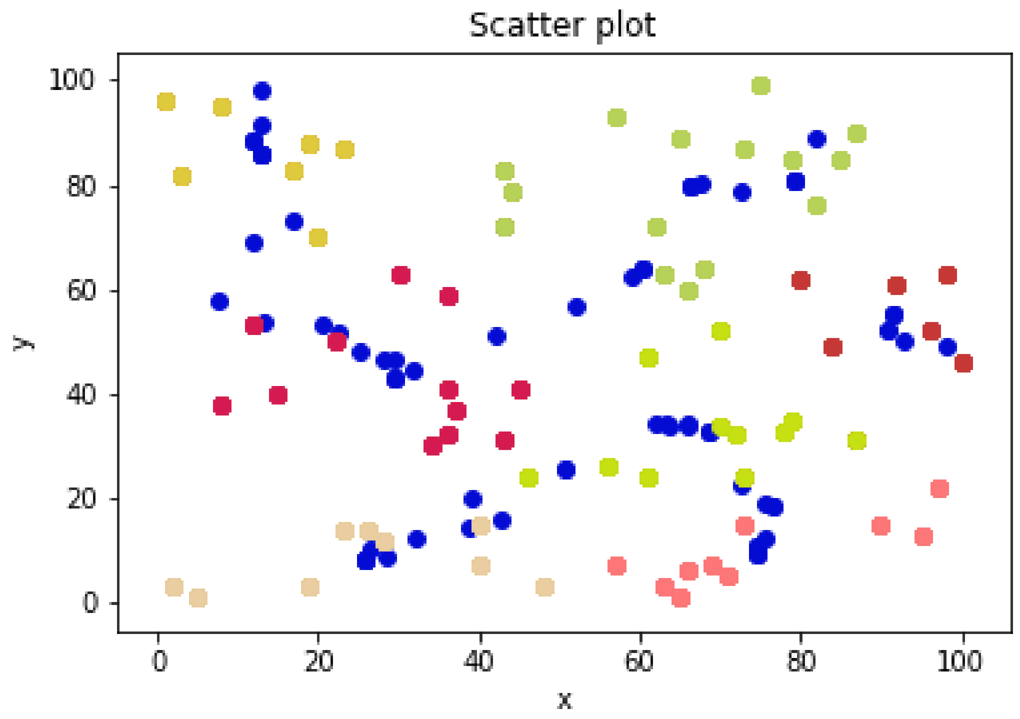
```
import numpy as np
import matplotlib.pyplot as plt
import random as r
x=[]
y=[]
cx=[]
cy=[]
n=int(input("Enter no. of coordinates"))
for i in range(0,n):
    x.append(r.randint(0,100))
    y.append(r.randint(0,100))
print(x,y)
plt.scatter(x, y,c='black')
plt.title('Scatter plot')
plt.xlabel('x')
plt.ylabel('y')
k=int(input("Enter value of k for k means clustering"))
for i in range(0,k):
    cx.append(r.randint(0,100))
    cy.append(r.randint(0,100))
def find(xc,yc,cx,cy):
    min=1000
    x=-1
    y=-1
    for i in range(0,k):
        if (cx[i]-xc)**2+(cy[i]-yc)**2<min:
            min=(cx[i]-xc)**2+(cy[i]-yc)**2
            x=cx[i]
            y=cy[i]
    return x,y
t=25
number_of_colors=k+1
color = ["#"+''.join([r.choice('0123456789ABCDEF') for j in range(6)])
             for i in range(number_of_colors)]
while(t):
    t-=1
    pointallocation={}
    for i in range(0,n):
        xc=x[i]
        yc=y[i]
        cxc,cyc=find(xc,yc,cx,cy)
        if (cxc,cyc) not in pointallocation:
            pointallocation[(cxc,cyc)]=[(xc,yc)]
        else:
            pointallocation[(cxc,cyc)].append((xc,yc))
    for i in range(0,k):
        plt.scatter(cx[i],cy[i],c=color[-1])
        for j in pointallocation[(cx[i],cy[i])]:
            plt.scatter(j[0],j[1],c=color[i])
    cx=[]
    cy=[]
    for i in pointallocation:
        xavg=0
        yavg=0
        for j in range(0,len(pointallocation[i])):
            xavg+=pointallocation[i][j][0]
            yavg+=pointallocation[i][j][1]
        xavg/=len(pointallocation[i])
        yavg/=len(pointallocation[i])
        cx.append(xavg)
        cy.append(yavg)

    plt.show()
```

Result-Itr1

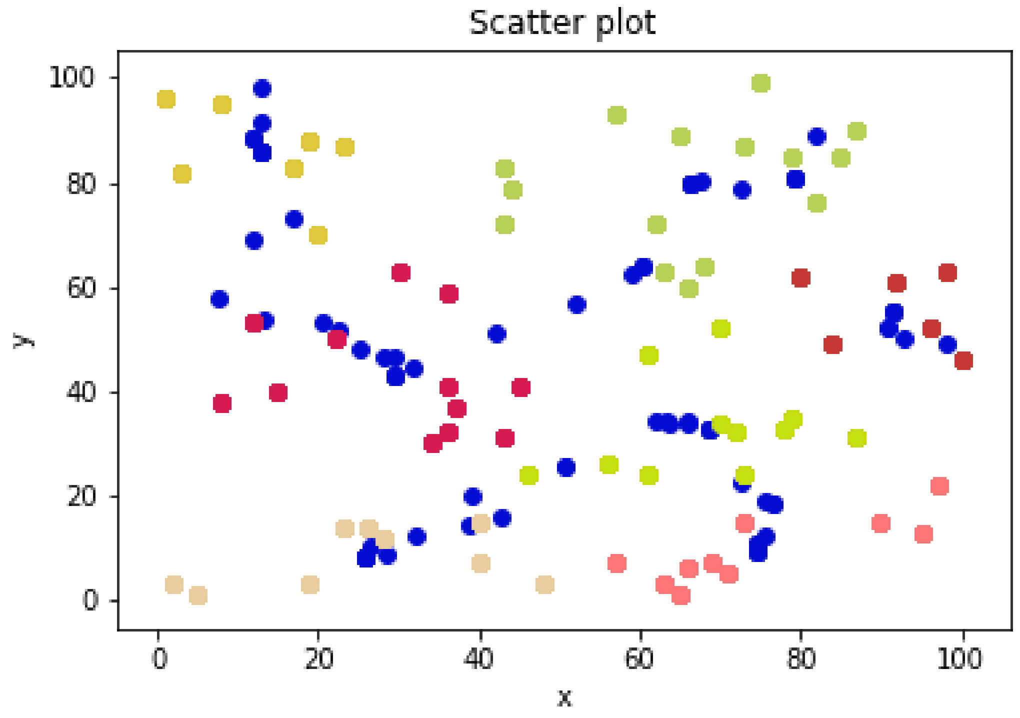

Scatter plot
y
x
0
20
40
60
80
100


Scatter plot
y
x
0
20
40
60
80
100

Itr3-

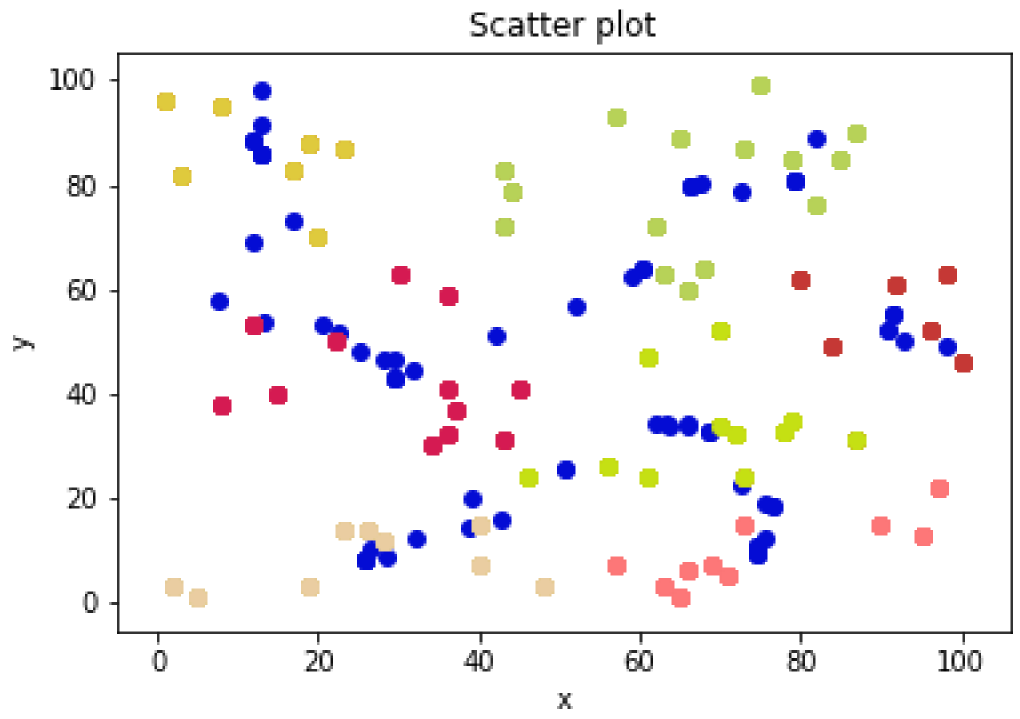


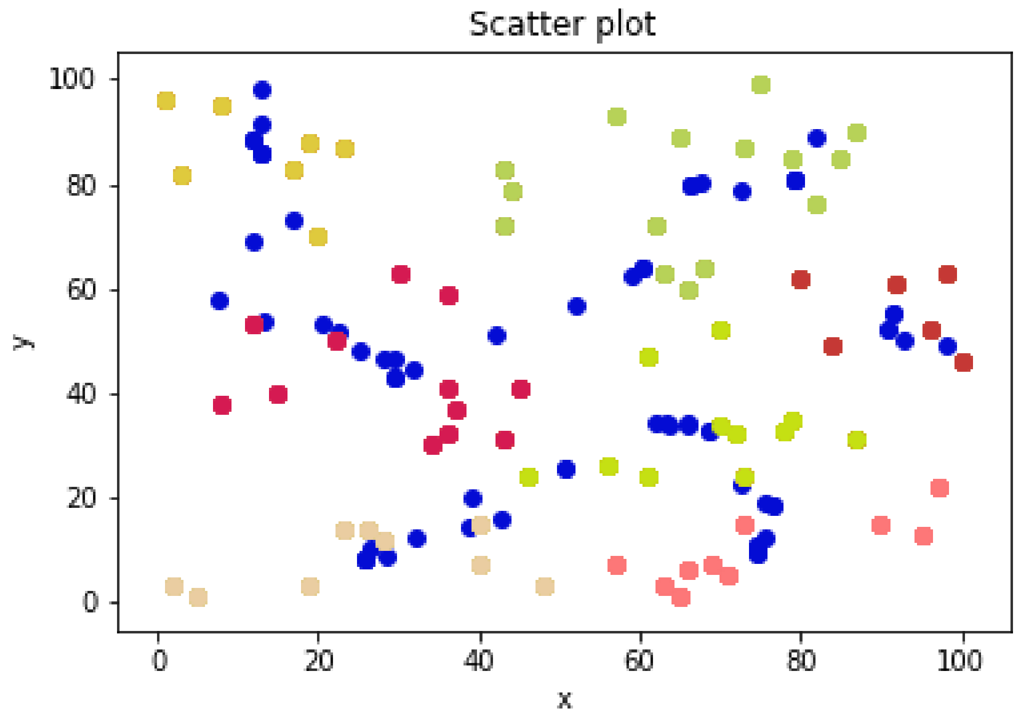
Scatter plot
y
x
0
20
40
60
80
100



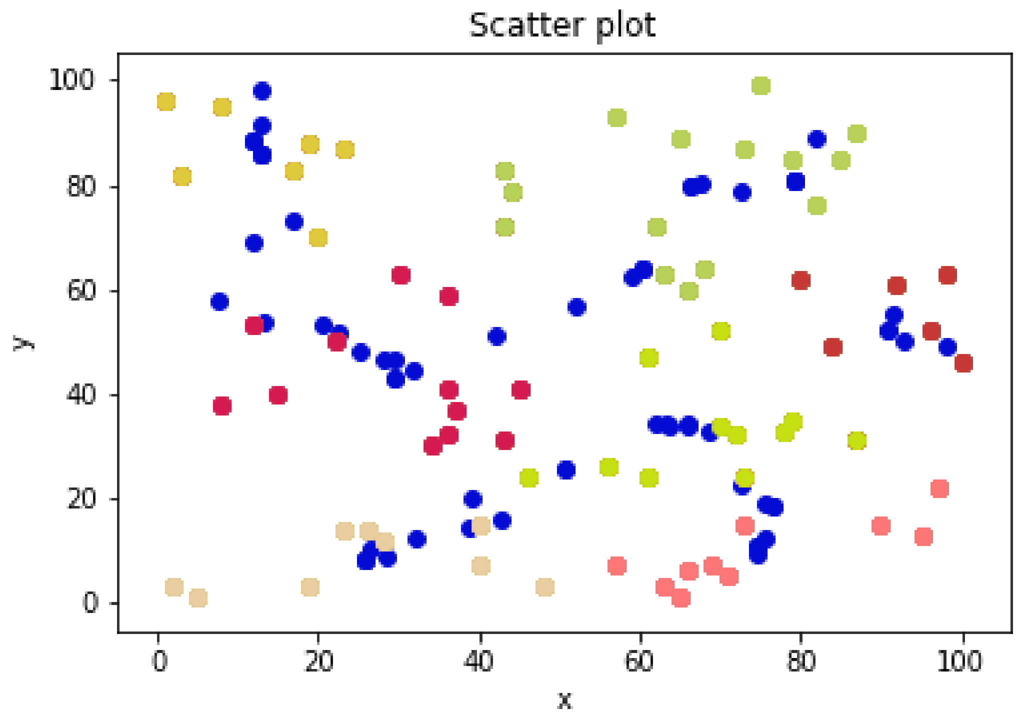
Scatter plot
y
x
0
20
40
60
80
100

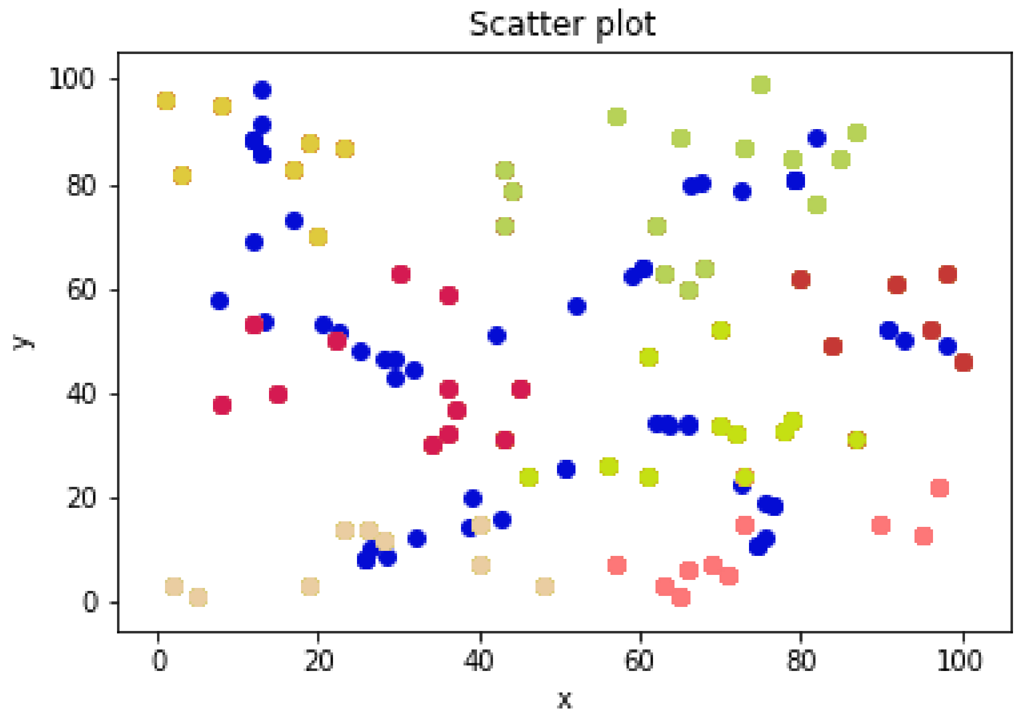
Itr5-


Scatter plot
y
x
0
20
40
60
80
100

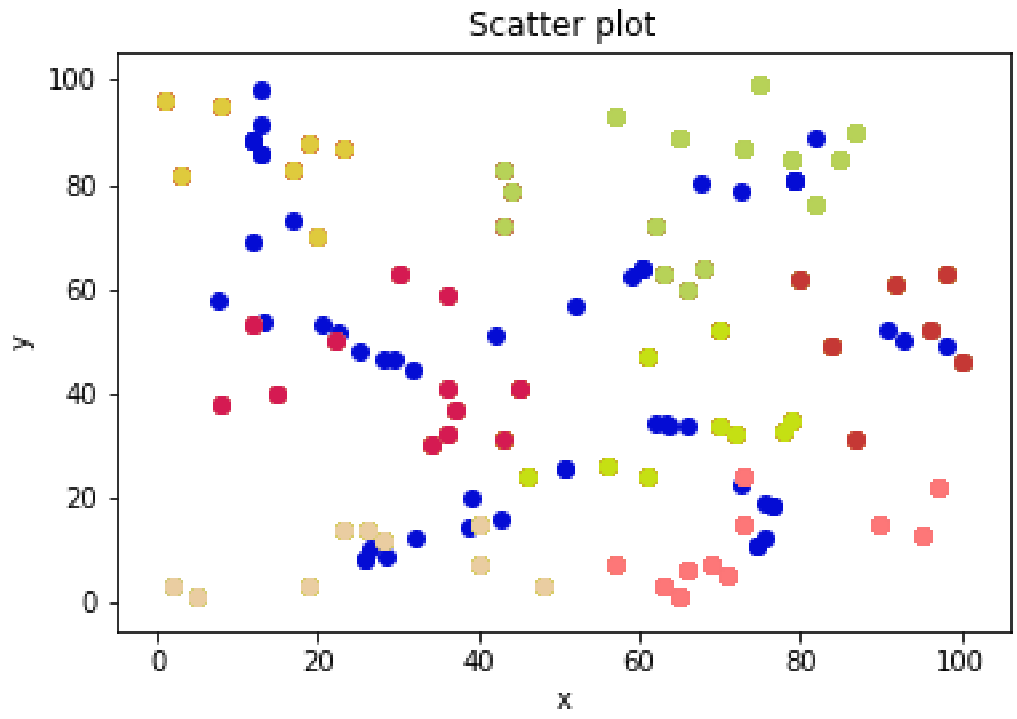

Scatter plot
y
x
0
20
40
60
80
100

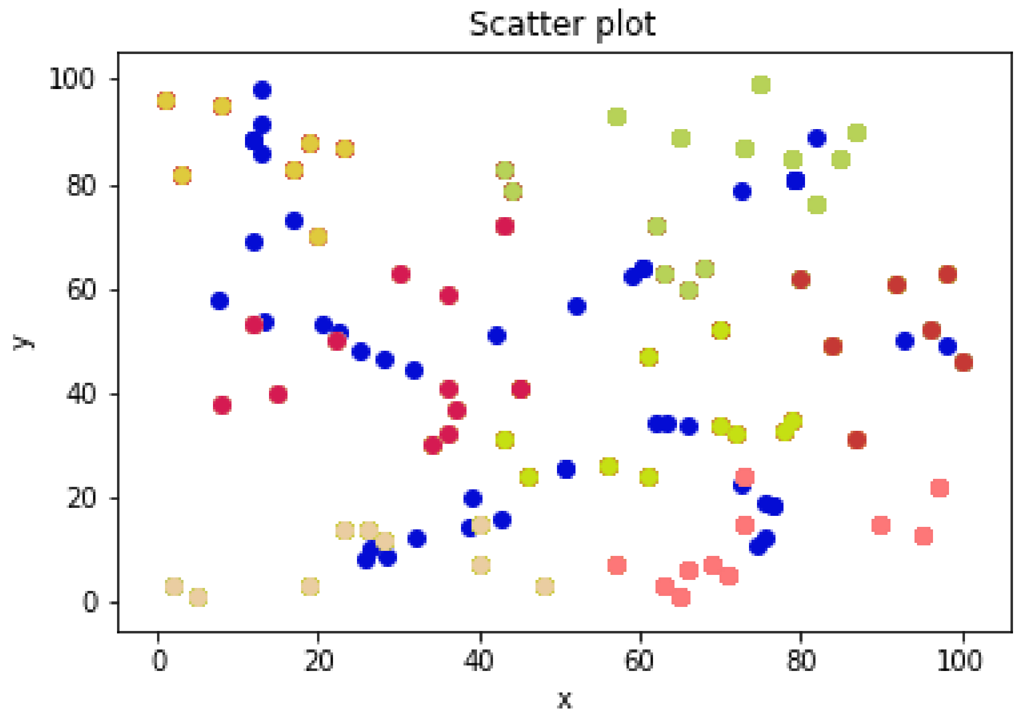
Itr7-


Scatter plot
y
x
0
20
40
60
80
100

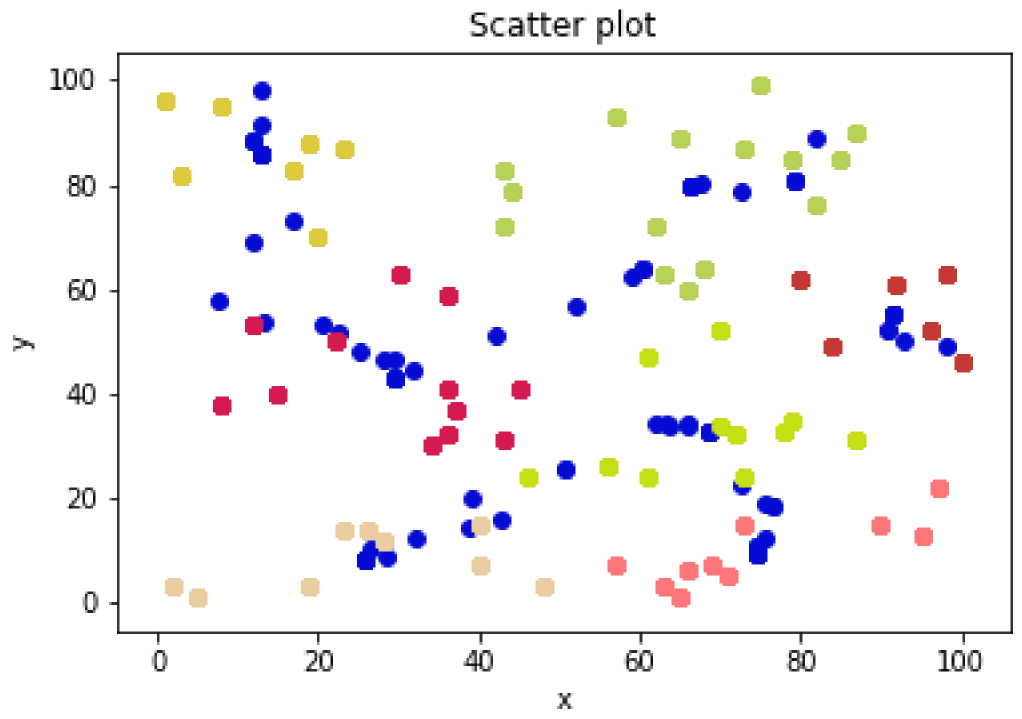

Scatter plot
y
x
0
20
40
60
80
100

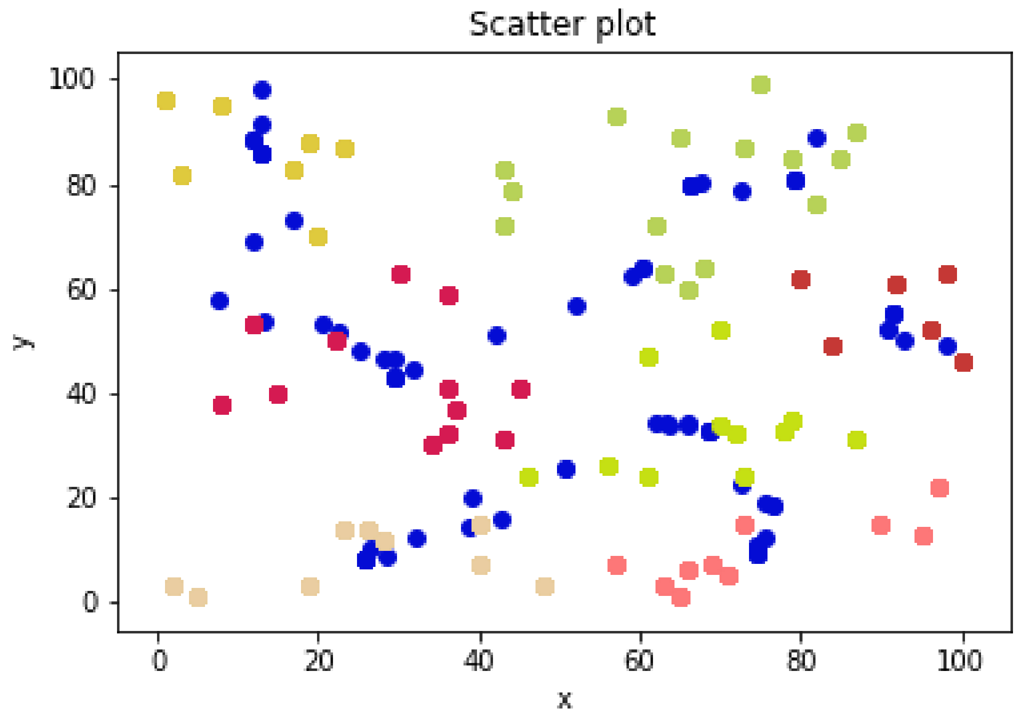
Itr9-


Scatter plot
y
x
0
20
40
60
80
100
0
20
40
60
80
100


Scatter plot
y
x
0
20
40
60
80
100
0
20
40
60
80
100

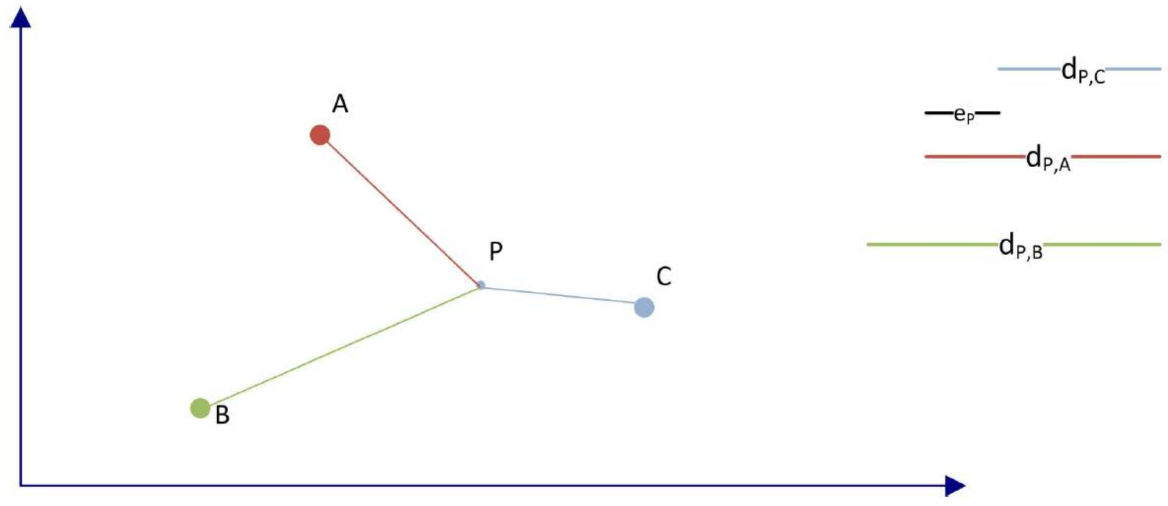
Itr11-


Scatter plot
y
x
0
20
40
60
80
100


Scatter plot
y
x
0
20
40
60
80
100

Optimization of k-means(Algorithm Design) :

Being able to determine which data object can be affected by a move can lead us to a very significant improvement on step number 2 because we no longer need to go to the entire data set, but the data A short list of objects we call a 'border' list). Before deciding which data objects should be placed on the 'border' list, we need to establish the criteria to be met by a data element so that it can be considered a 'border' element. Let us consider the Figure, which presupposes that I have to calculate.



Let P be a part of cluster C. All other points have been omitted for ease of presentation. Point P is part of cluster C because the distance from P to C (DPC) is less than the distance from A (DPA) and the distance from B (DPB). We want to know the distance of jump from point P to another cluster. This would clearly be:

EP = min (DPA-DPC, DPB - DPC) (1)

We have labeled the distance from P to the nearest edge as eP. We can say that point P is eP-away from switching clusters.

At the end of iteration i, the centroid must be updated based on the configuration of the new groups. Let us assume that nucleus A was moved to 'A', nucleus B was shifted to B and nucleus C was moved to C.

In the context shown in Figure 2, the worst case for point P would be: point C is far from P 'CC'. While point A passed. AA '| And point B passed. BB '| What would be the situation for P to live in Cluster C

Obviously it will be

eP> | CC '| + | AA '| (2)

And

ePs> | CC '| + | Bibi '| (3)

To simplify the algorithm a bit and reduce computations, we can mix conditions (2) and (3):

eP> 2 * max (| AA '|; | BB' | |; CC '|) (4)

That being said, we have found a way to determine if a point is part of the 'border' list. However, we are still not well because checking the inequality for each of our data elements at each iteration gives us back to where we started. To avoid such calculations, we can map all of our data elements in a wide interval to the value of E, as shown in Algorithm 2:

Algorithm 2 indicates groups with close values of E so that instead of visiting each data element and checking against their close distance, we can do so for the entire group. This agreement is the key to complete optimization. The WIDTH constant has a large effect on optimization. If the value of WIDTH is small, the number of intervals will increase and the workload involved can be significantly increased by checking and updating the interval at each iteration. If the value of WIDTH is large, the number of points increases for each interval, and even though the number of intervals decreases, the performance loss is evident when a larger interval is marked for re-visiting.

One can easily see that the quality of the proposed groups is not affected by the proposed optimization. At each iteration, the structure of the groups is exactly the same as we run standard K-Means.

Optimized K- Means algorithm:

1. Define constant WIDTH

2. Define intervals Ii = [i * WIDTH, (i+1) * WIDTH) and tag them with value i * WIDTH

3. Mark the entire data set to be visited

4. For each point to be visited

5. Compute e = min(dPCl - dPCw) where Cw is the center of the winner (closest) cluster and Cl, l=1..k, l6= w stands for all other centroids

6. Map all points with i * WIDTH < e < (i+1) * WIDTH to interval i * WIDTH where i is a positive integer

7. Compute new centroids Cj, where j=1..k and their maximum deviation D = max(|CjCj'|)

8. Update Ii's tag by subtracting 2 * D (points owned by this interval got closer to the edge by 2 * D)

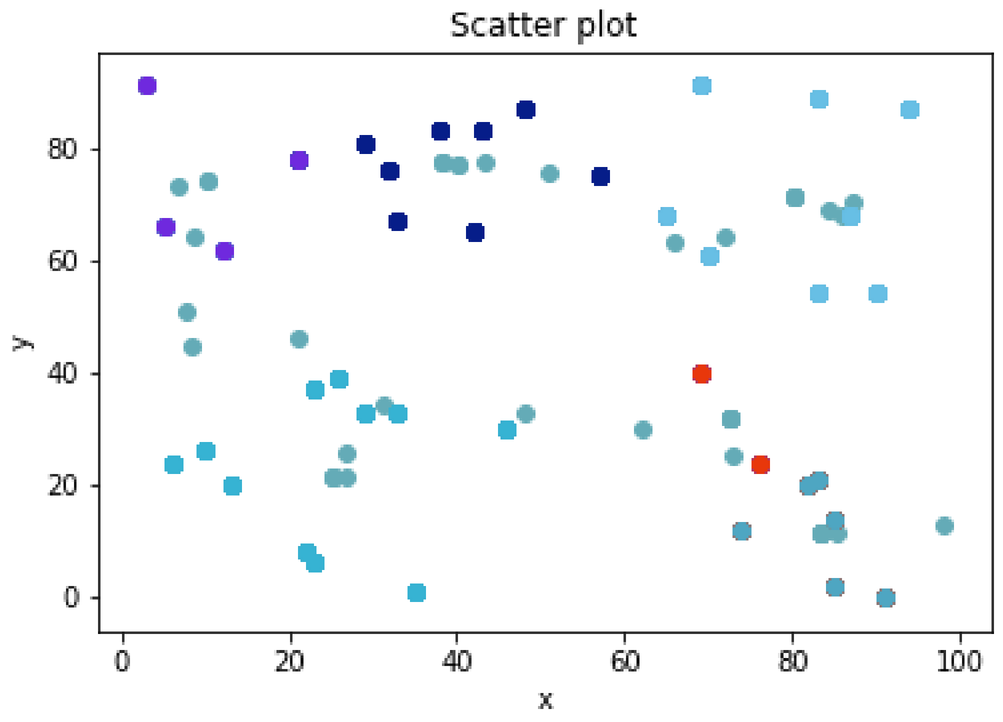
9. Pick up all points inside intervals whose tag is less or equal to 0, and go to 4 to revisit them

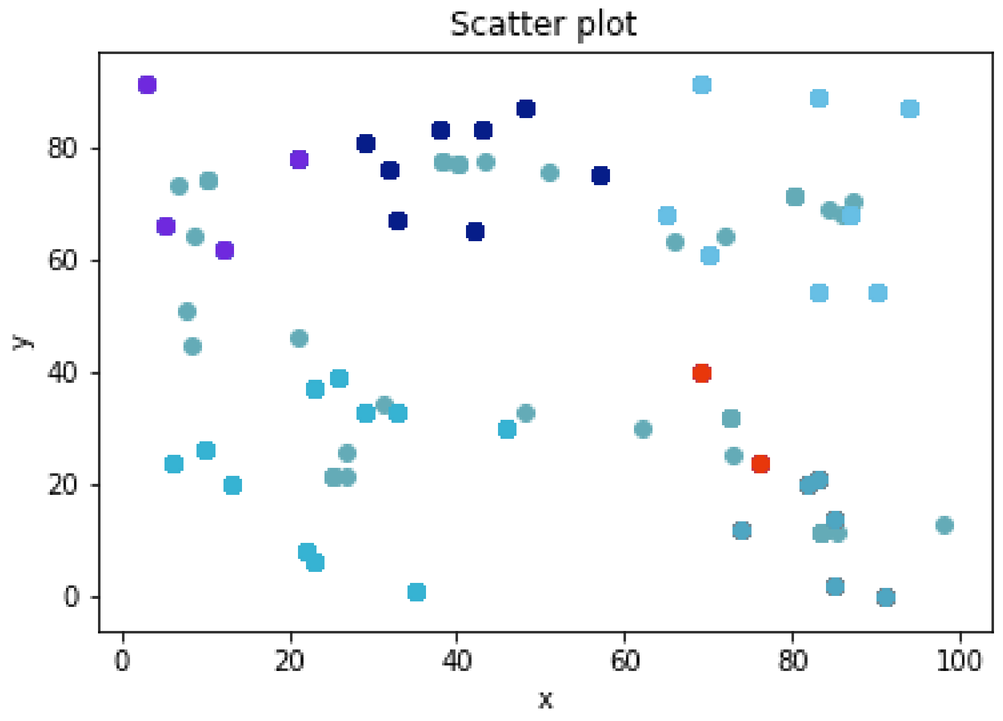
Code-

```
import copy
import math

import time
import random
import numpy as np
from sklearn import metrics
from sklearn import datasets
from numpy import genfromtxt
import numpy.random as nprand
import matplotlib.pyplot as plt
from profilehooks import profile
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import pairwise_distances

def reject_outliers(data, m = 2.):
    d = np.abs(data - np.median(data))
    len(data)
    mdev = np.median(d)
    s = d/mdev if mdev else 0.
    return data[s<m]

def populate_essentials(fname, hyper_to_d):
    rows = genfromtxt(fname, delimiter=',')
    n = len(rows)
    hypercube_dim_float = math.log(n,2)
    #print "hypercube_dim_float " + str(hypercube_dim_float)
    hypercube_dim = math.ceil(hypercube_dim_float)
    #print "hypercube_dim " + str(hypercube_dim)
    n = 0
    for row in rows:
        row = row[:-1]
        hyper_to_d[n] = np.array(row)
        n = n+1
    return (hypercube_dim, n)

# particles is the indices of the current particles

def initialize(num_clusters, hyper_to_d, particles_h_to_d, particles, hypercube_dim, velocity):
    for i in range(num_clusters):
        x = random.choice(hyper_to_d.keys())
        particles_h_to_d[x] = hyper_to_d[x]
        particles[i] =  x
        x = hypercube_dim
        a = random.randrange(0,x)
        velocity[i] = int(a)

# calculate the global fitness function
def global_fitness(particles, hyper_to_d):
    arr = []
    for i in xrange(len(particles)):
        for j in xrange(i, len(particles)):
            arr.append((hyper_to_d[particles[i]]-hyper_to_d[particles[j]])**2)
    return np.sum(arr)

# calculate the local fitness function
def local_fitness(p, particles_h_to_d):
    #p is an integer ie. index of the required particle
    key = particles_h_to_d[p]
    arr = []
    for p1 in particles_h_to_d.keys():
        arr.append((particles_h_to_d[p1] - key)**2)
    return np.sum(arr)

def move(particle, vel, hyper_to_d, particles_h_to_d, dim):
    '''d = np.array([particle])
    bin = (((d[:,None] & (1 << np.arange(dim)))) > 0).astype(int)
    ones = np.where(bin>0)[1]'''
    particle = bin(particle)[2:]
    diff1 = (int)(dim - len(particle))
    lis = ''
    for i in range(diff1):
        lis+='0'
    lis+=particle
    particle = lis

    if vel < 0:
        vel = -vel

    diff = nprand.randint(dim, size = vel)
    out = particle
    for i in diff:
        if out[i] == '0':
            out = out[:i] + '1' + out[i+1:]
        else:
            out = out[:i] + '0' + out[i+1:]
    num = int(out, 2)
    ones = np.where
    while num not in hyper_to_d and num not in particles_h_to_d:
        if vel < 0:
            vel = -vel
        diff = nprand.randint(dim, size = vel)
        out = particle
        for i in diff:
            if out[i] == '0':
                out = out[:i] + '1' + out[i+1:]
            else:
                out = out[:i] + '0' + out[i+1:]
        num = int(out,2)
    return int(out, 2)

def edit_distance(p1, p2, hypercube_dim):
    #arr is an array
    pos1 = bin(p1)[2:]
    pos2 = bin(p2)[2:]
    dist = abs(len(pos1) - len(pos2))
    for i in range(min(len(pos1), len(pos2))):
        if pos1[i] != pos2[i]:
            dist+=1
    return dist

def pso(iterations, num_clusters, num_particles, hyper_to_d, particles_h_to_d, hypercube_dim, particles, velocity):
    i =0
    gbest = np.zeros(num_clusters, dtype = "int")
    pbest = np.zeros(num_clusters, dtype = "int")

    np.copyto(gbest, particles)
    np.copyto(pbest, particles)

    best_value =  global_fitness(gbest, hyper_to_d)
    best_local_value = []
    bests = []

    for j in range(num_clusters):
        best_local_value.append(local_fitness(particles[j] , particles_h_to_d))

    while i < iterations:
            #print "particles[j] = " + str(particles[j])
            val = hyper_to_d[particles[j]]

            velocity[j] = int(velocity[j] + nprand.uniform(-1,1) * edit_distance(pbest[j], particles[j], hypercube_dim) +
nprand.uniform(-1,1) * edit_distance(gbest[j], particles[j], hypercube_dim))

            particles[j] = move(particles[j] , velocity[j], hyper_to_d, particles_h_to_d,hypercube_dim)

            particles_h_to_d[particles[j]] = hyper_to_d[particles[j]]
            gfit = global_fitness(particles, hyper_to_d)

            if(gfit > best_value):
                print "improved from " + str(best_value) + " to " + str(global_fitness(particles, hyper_to_d))
                best_value = gfit
                np.copyto(gbest, particles)
                new = np.zeros(num_clusters, dtype = "int")
                np.copyto(new, gbest)
                bests.append(new)

            pfit = local_fitness(particles[j], particles_h_to_d)
            if pfit > best_local_value[j]:
                #print "improved pbest"
                best_local_value[j] = pfit
                pbest[j] = particles[j]
        i = i+1
    return (bests, pbest, best_value)


def driver(fname, cluster, iterations):
    num_clusters = cluster
    hyper_to_d = {}
    particles_h_to_d = {}
    particles = np.zeros(num_clusters, dtype = 'int')
    hypercube_dim, num_particles = populate_essentials(fname, hyper_to_d)
    k = 0
    #print hyper_to_d
    velocity = np.zeros(num_clusters, dtype = 'int')


    initialize(num_clusters, hyper_to_d, particles_h_to_d, particles, hypercube_dim, velocity)

    best_value = global_fitness(particles, particles_h_to_d)
    start = time.time()
    (bests, pbest, best_value) = pso(iterations, num_clusters, num_particles, hyper_to_d,  particles_h_to_d,
hypercube_dim, particles, velocity)
    end = time.time()
    ti = end-start
    print "time to initialize = "
    print ti
    #print str(sum(best_local_value)/2) + " local value"

    f = open(fname)
    f.readline()  # skip the header
    X = np.genfromtxt(f, delimiter = ',')
    y = X[:,-1]
    y = y.astype(np.int)
    #print max(y)
    X = np.delete(X, -1, 1)

    # iris = datasets.load_iris()
    # X = iris.data
    # y = iris.target
    # print y

    estimators = {'kmeans++': KMeans(n_clusters=cluster, n_jobs = -1),
                  'k_means_bad_init': KMeans(n_clusters=cluster, n_init=10,
                                              init='random', n_jobs = -1)
                 }
      lis = []
      for j in bests[i]:
          lis.append(hyper_to_d[j])
      estimators["kmeans_pso_"+str(i)] =  KMeans(n_clusters=cluster, n_init = 1, init=np.asarray(lis), n_jobs = -1)

    #estimators['k_means_pso_last' + str(i)] =  KMeans(n_clusters=9, n_init = 1, init=np.asarray(hyper_to_d[]), n_jobs
= -1)

    #np.random.seed(5)
    fignum = 1
    for name, est in estimators.items():
        start = time.time()
        est.fit(X)
        end = time.time()
        labels = est.labels_
        print str(-est.score(X)) + " score of kmeans " + name
        tkm = end-start
        print tkm
        if name[7:10] == "pso":
            ti += tkm
        '''fig = plt.figure(name, figsize=(10,9))
        plt.clf()
        ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=40, azim=134)
        ax.scatter(X[:, 5], X[:, 6], X[:, 10], c=labels.astype(np.float))
        fignum = fignum + 1
    # Plot the ground truth
    fig = plt.figure("ground truth", figsize=(4, 3))
    plt.clf()
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=40, azim=134)
    plt.cla()
    for name, label in [('1', 1),
                        ('2', 2),
                        ('3', 3),
                        ('4', 4),]
        ax.text3D(X[y == label, 5].mean(),
                  X[y == label, 6].mean(),
                  X[y == label, 10].mean(), name,
                  horizontalalignment='center',
                  bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
    # Reorder the labels to have colors matching the cluster results
    #y = np.choose(y, [1, 2, 3, 4, 5, 6, 7, 8, 9]).astype(np.float)
    ax.scatter(X[:, 5], X[:, 6], X[:, 10], c=y)
    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('dim1')
    ax.set_ylabel('dim2')
    ax.set_zlabel('dim3')
    plt.show()'''
```

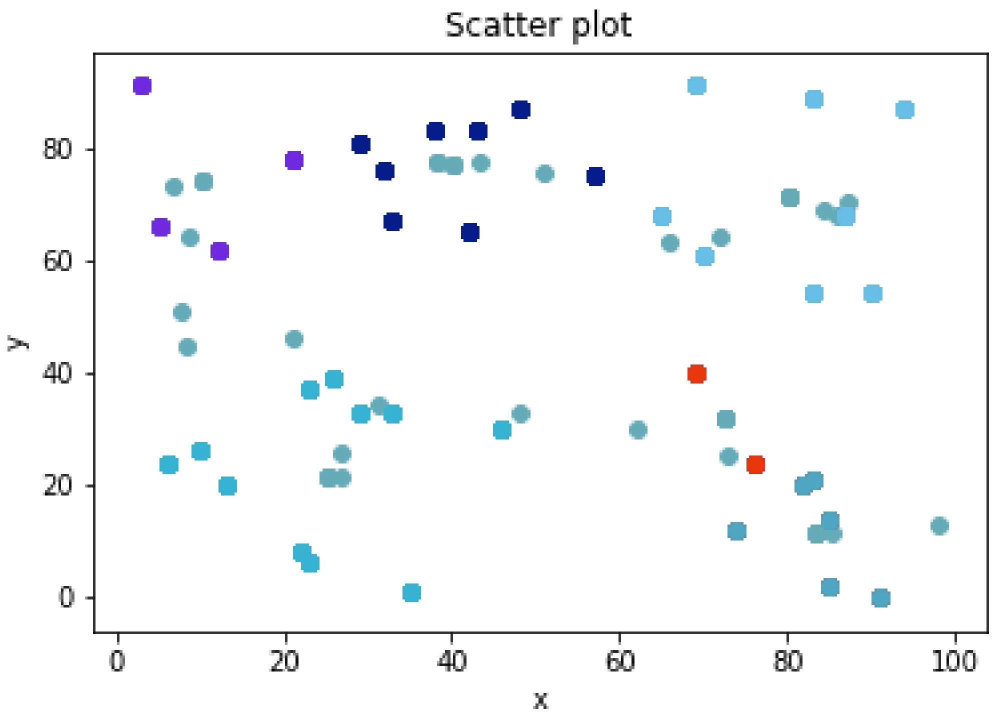
```
cluster = 6
iterations = 300
```

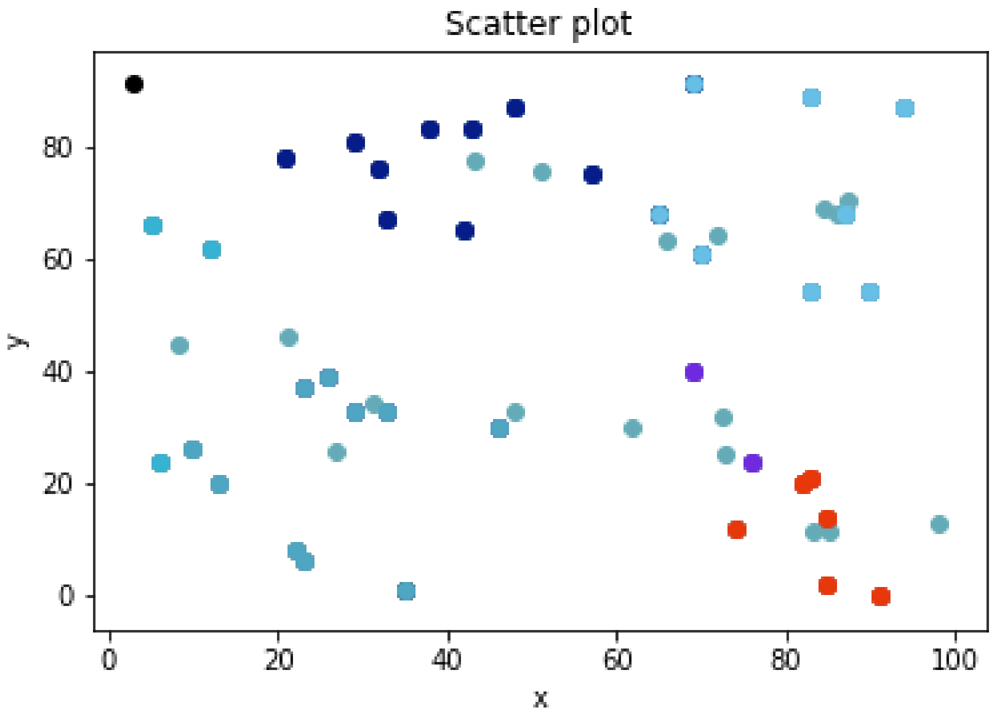
Result For Optimized Algorithm:itr-1



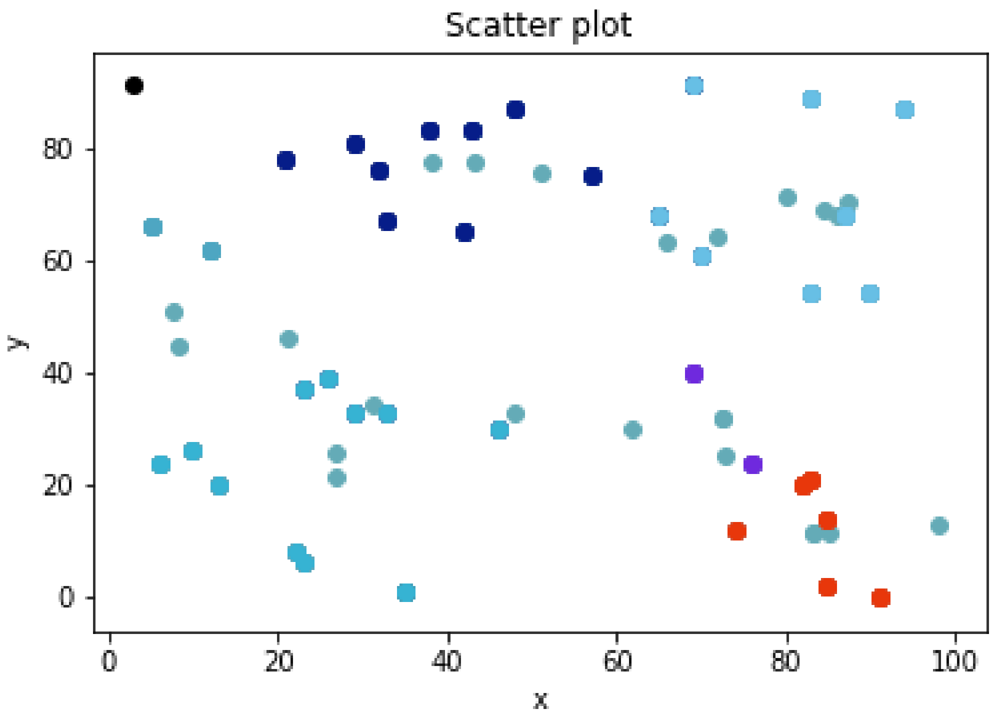


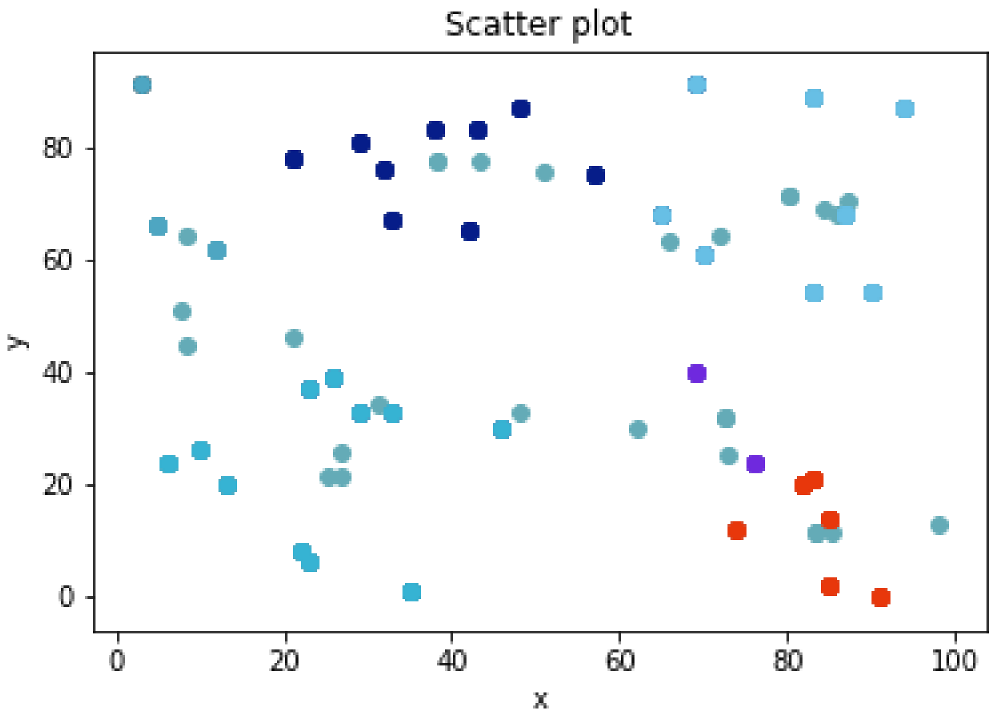
Itr3-


Scatter plot
y
x
0
20
40
60
80
100

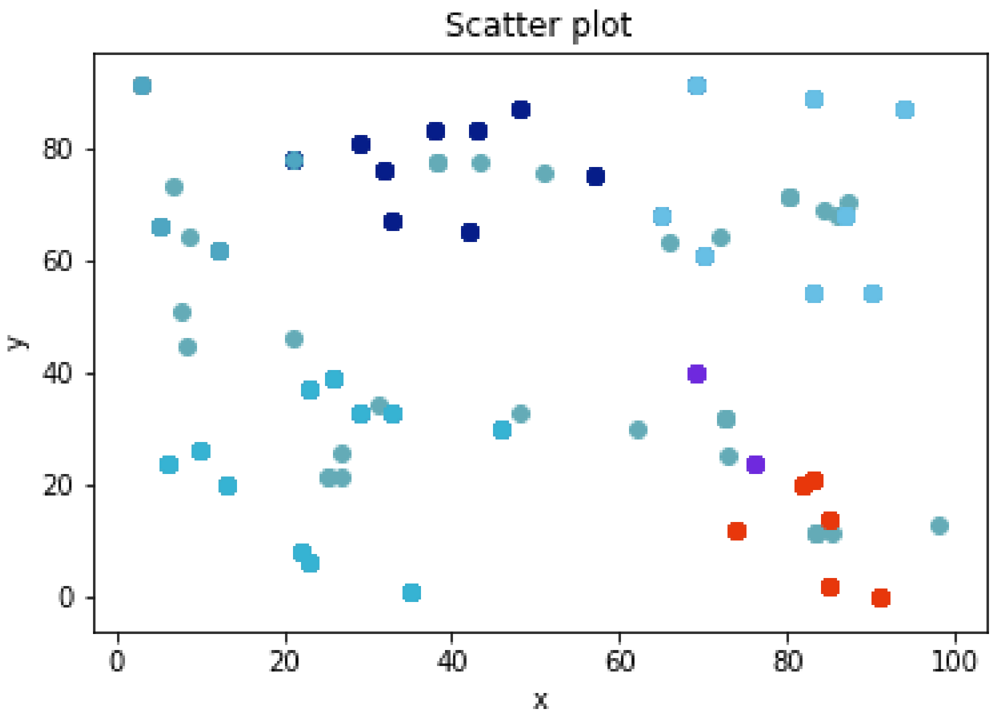

Scatter plot
y
x
0
20
40
60
80
100

Itr5-


Scatter plot
y
x
0
20
40
60
80
100


Scatter plot
y
x
0
20
40
60
80
100

Itr7-



Result-

Instead of 12 iteration in the basic algorithm of k-means, we are getting only 7 iteration to find the same result with the help of PCA algorithm and some optimization technique.

Which lead to decrease in 5 iteration when compared to the original algorithm.

Conclusion and future work:

The Project offers a customized version of the K-Means algorithm. Optimization refers to the running time. Optimization comes from a considerable lack of data space that recurs on each loop. The algorithm defines a 'border' region made up of points that are close enough to the edge of their cluster to cause them to switch clusters to move to the next centroids.

A prototype implementation of a domain specific data set has been evaluated. The implementation assumes that the data set is composed of 2D points. The data set has been generated using a uniform distribution generator.

The future work will focus on domain-independent implementation and evaluation of algorithms. The scalability of the algorithm as well as the data sensitivity (form and distribution) are to be analyzed for the purpose of drawing conclusions on what would be the best and worst environment (data and configuration) for the algorithm.

A natural question would be whether the algorithm could be improved. One can easily note that the width of the grouping interval can be a point of vulnerability for performance gains. The shorter the width, the more gaps to be checked; The greater the width, the more points must be checked when their spacing distances are close to the edge.

References:

[1] Dolnicar, S, Using cluster analysis for market segmentation – typical misconceptions, established methodological weaknesses and some recommendations for improvement, Australasian Journal of Market Research, 2003, 11(2), 5-12.

[2] Ng, H.P., Ong, S.H.; Foong, K.W.C.; Goh, P.S.; Nowinsky, W.L. - Medical Image Segmentation Using K-Means Clustering and Improved Watershed Algorithm, 7th IEEE Southwest Symposium on Image Analysis and Interpretation, March 26-28, 2006, Denver, Colorado, pages 61-66

[3] Hongwei Xie, Li Zhang ; Jingyu Sun ; Xueli Yu - Application of Kmeans Clustering Algorithms in News Comments - The International Conference on E-Business and E-Government, May 2010, Guangzhou, China, pages 451-454

[4] kK Oyelade, O. J, Oladipupo, O. O, Obagbuwa, I. C – Application of K-Means Clustering algorithm for prediction of Students' Academic Performance, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 1, 2010, pages 292–295

[5] Burdescu, D.D.; Mihaescu, M.C., "Enhancing the Assessment Environment within a Learning Management Systems," EUROCON, 2007. The International Conference on "Computer as a Tool", vol., no., pp.2438,2443, 9-12 Sept. 2007

[6] Chang Wen Chen, Jiebo Luo, Kevin J. Parker - Image Segmentation via Adaptive K-Mean Clustering and Knowledge-Based Morphological Operations with Biomedical Applications, IEEE Transactions on Image Processing, VOL. 7, NO. 12, DECEMBER 1998, pages 1673 - 1683

[7] T. Kanungo, D.M. Mount, K.D. Piatko, N.S. Netanyahu, R. Silverman, A. Y. Wu - An efficient k-means clustering algorithm: analysis and implementation, Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume:24, Issue: 7), pages 881-892, July 2002, ISSN 0162-8828