CS 6990

Fuzzy Clustering Project

By

Nisarg Dave

CS 6990: Computational Intelligence

Project 2

Introduction

The Fuzzy C-means algorithm is the advancement of the crisp c means. Instead of clustering the objects into the same groups, it concentrates on clustering based on soft boundaries. It manipulates the membership function using the cost based optimization. The optimization is based on the iterative process of calculating membership partition matrix.

1. Fuzzy C means algorithm:

Input parameters:

1. C - number of clusters
2. X - Data Matrix N_samples*N_features
3. U_up - Updating fn
4. m - Fuzziness parameter
5. Max - Maximum number of iterations allowed
6. Tol - tolerance factor for membership criterion

Designing the core function for performing FCM on 2D data,

```
function [prediction v] = fcm(c, X, m, metric, Max, tol)
[n, no] = size(X);
U = zeros([c, n]);
```

We need to select the initial centroids using three different conditions,

```
v = repmat(max(X), c, 1).*rand([c, no]);  %for random initialization
v = repmat(max(X), c, 1).*randn([c, no]);  %centers selected from Gaussian normal Distribution
r = randi([1 140],1,3);
v = [50,34,15,2;65,28,46,15;64,32,53,23]; %vectors from the existing data points
```

Calculating the U and V from the equation,

```
U = rand([c, n]);
for j = 1:n
    U(:, j) = U(:, j)./sum(U(:, j));
end
for i = 1:c
```

```
    v(i, :) = sum((X.*repmat(U(i, :)'.^m, 1, 4)),1)./sum(U(i, :).^m);
end
```

Using optimization using the FCM cost optimizer,
We are finding the Obj function value using the U and updated V index, we are updating the U and v values in iteration,

```
v_old = v;
delta = 1e4;
k = 0;
while  (k<Max & delta>tol)
   for i = 1:c
     for j = 1:n
       U(i, j) = 1/sum((metric(X(j, :), v(i, :))./metric(X(j, :), v)).^(2/(m-1)));
     end
   end
   for i = 1:c
      v(i, :) = sum((X.*repmat(U(i, :)'.^m, 1, 4)), 1)./sum(U(i, :).^m);
   end
v_new = v;
delta = max(max(abs(v_new-v_old)));
v_old = v;
```

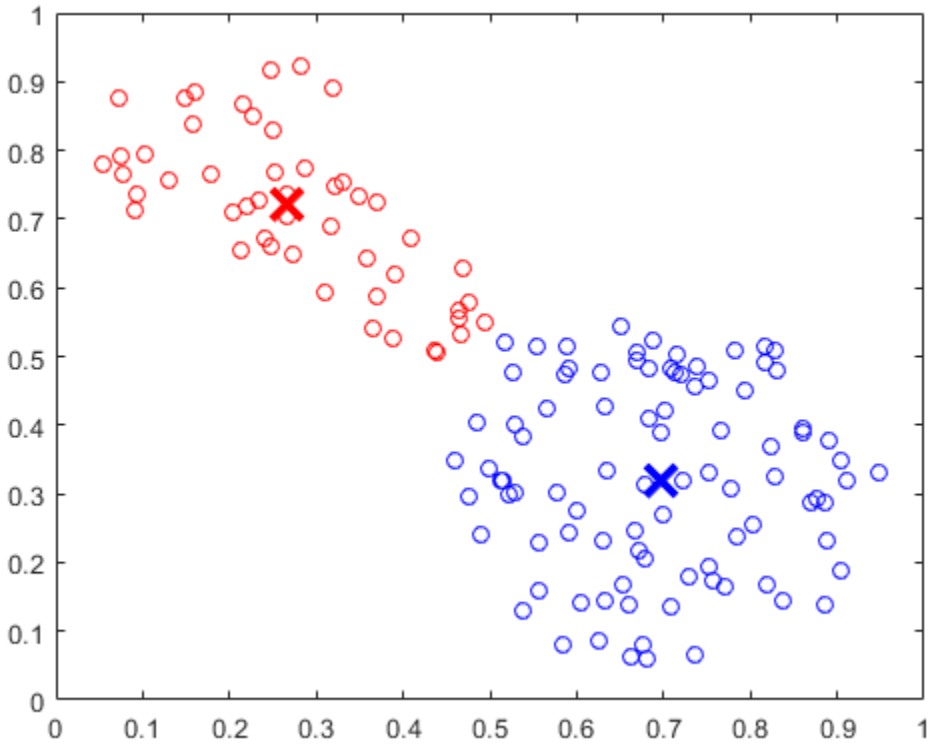
Here Metric is the method function for calculating the distance between the points, which finds the euclidean distance measure.

The last step is to perform the prediction and compare it for calculating overall accuracy measure.

```
prediction = zeros([1, n]);
for i = 1:n
  [M, prediction(i)]=max(U(:, i));
End
```

Testing of FCM on 2D Data:

I tested my algorithm on simple 2D data points with well defined clusters. The output is below,



2. GKFCM Implementation for simple data,

The further modification on FCM is using the nu for each iteration and using that value as a scaling parameter.

```
function result = GKFCM(data,param)

%checking the parameters given
f0=param.c;
X=data.X;

[N,n] = size(X);
[Nf0,nf0] = size(f0);
X1 = ones(N,1);
%default parameters
if exist('param.m')==1, m = param.m;else m = 2;end;
if exist('param.e')==1, e = param.m;else e = 1e-4;end;
if exist('param.ro')==1, rho=param.ro;
else
```

```
    if max(Nf0,nf0) == 1
        rho = ones(1,param.c);
    else
        rho = ones(1,size(f0,2));
    end
end
if exist('param.gamma')==1, gamma = param.gamma;else gamma = 0;end;
if exist('param.beta')==1, beta = param.beta;else beta = 1e15;end;
```

Here I did set the initial clusters and size for each matrix. The parameter options are for selecting the optimal rho value.

Here also I'm using three different initialization,

```
v = 2*(ones(c,1)*aa).*(rand(c,n)-0.5) + ones(c,1)*mm;  %From random distribution
v = 2*(ones(c,1)*aa).*(randn(c,n)-0.5) + ones(c,1)*mm; % for Gaussian normal distribution
v = [50,34,15,2;65,28,46,15;64,32,53,23]; %vector from the data points
```

GKFCM has major difference for calculating the partition matrix and using the covariance estimation for each GK step.
Here we are using Mahal distance for estimating the metric between data points, Mahal fn uses the more convex conculutional shape for estimating round edges between centers.

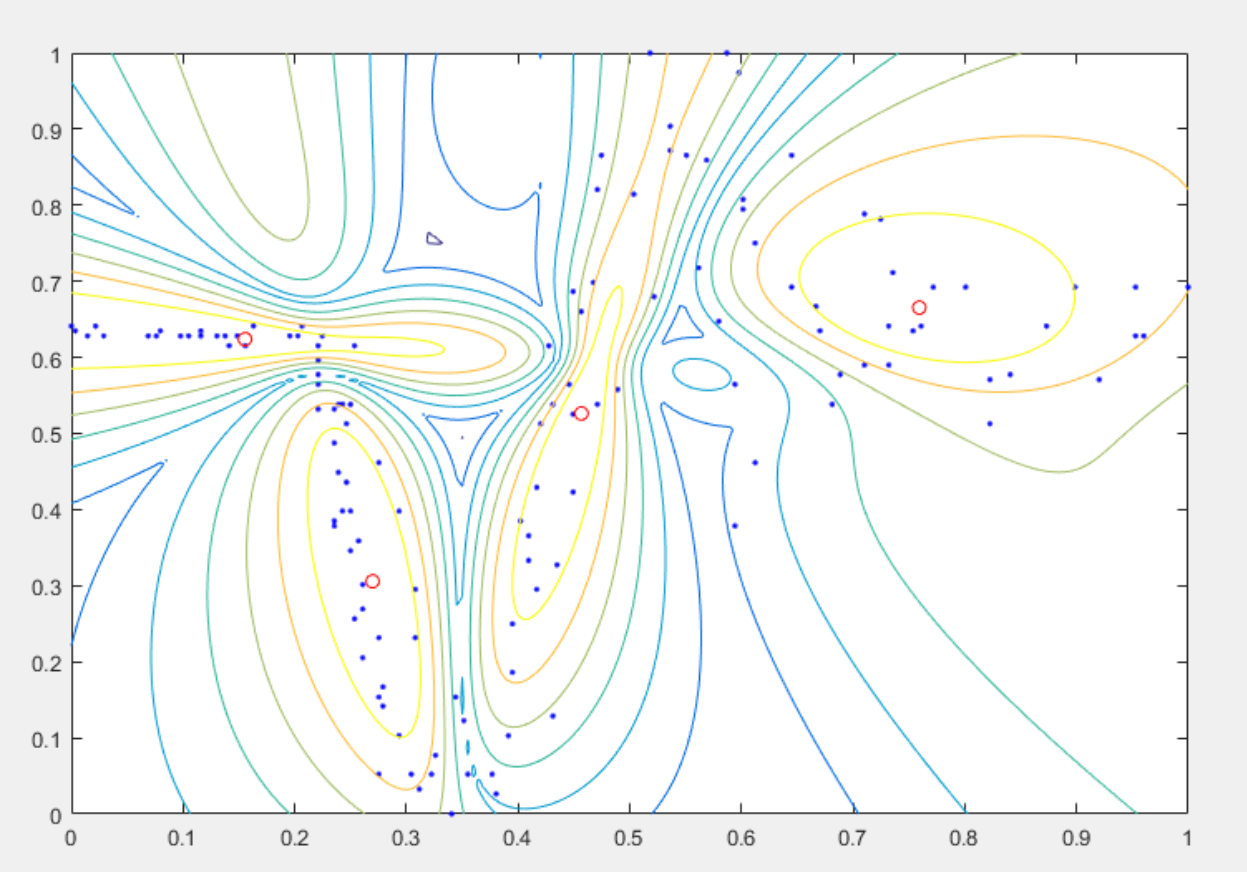
I tried two different way for doing that,

```
 M = (1/det(pinv(A))/rho(j))^(1/n)*pinv(A);
 Or %M(:,:,j) = (det(A)/rho(j)).^(1/n)*pinv(A);
```

Now we will calculate the same objective function using the cost estimator J(U,V),
Finally gkFCM returns the following things as a output,

1. Covariance matrix P
2. Eigenvectors
3. Eigenvalues
4. Norm-inducing matrix using distance metric
5. Overall cost estimation.

The testing on 2D data gave following result,



Total Iterations: 70
Cluster centers V,
0.759172650973751 0.665432967234065
0.155523958169872 0.623896228262697
0.456695319169521 0.526348843010176
0.269726674634449 0.305785297183463

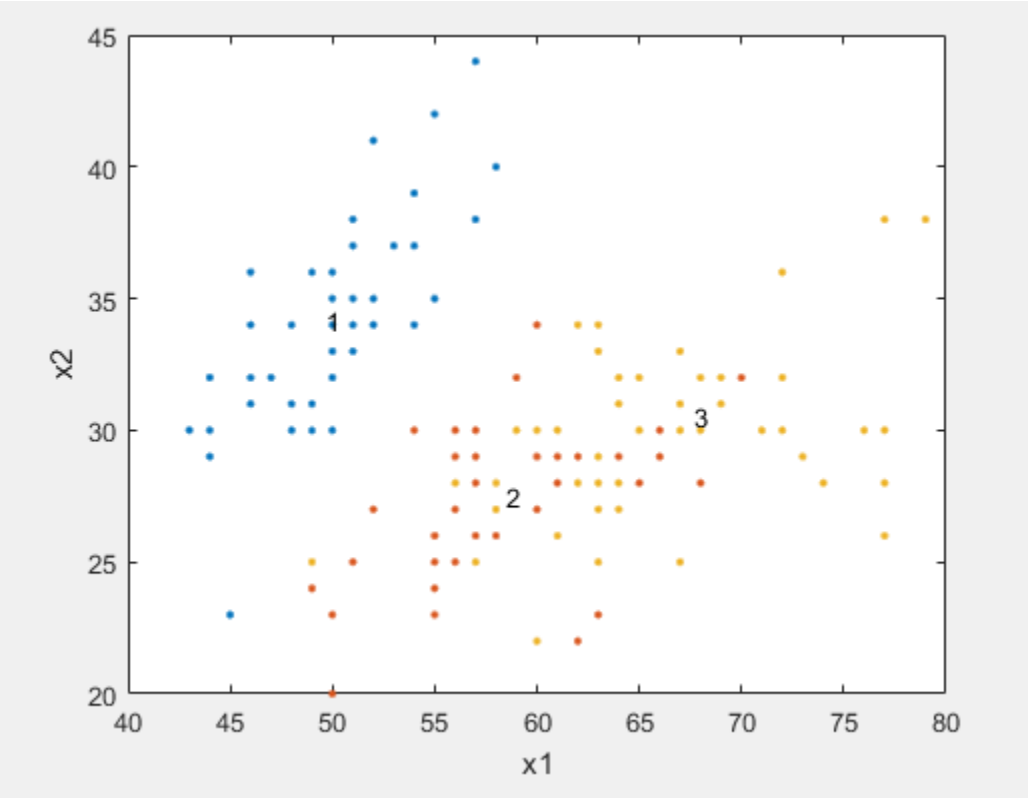
Possibilistic C means:

The possibilistic c means, initial step is to start first iteration with the simple FCM and then using the nu as step scaler. The value of nu is to be entered by user.
The fuzzy partitions and typicality are considered as the U & T initials.

```
mf = U.^expo; % MF matrix after exponential
%modification
tf=T.^nc;
tfo=(1-T).^nc;
center_new = (a.*mf+b.*tf)*data./((ones(size(data,2), 1)*sum(a.*mf.'+b.*tf.'))');
dist = distfcm(center, data);
```

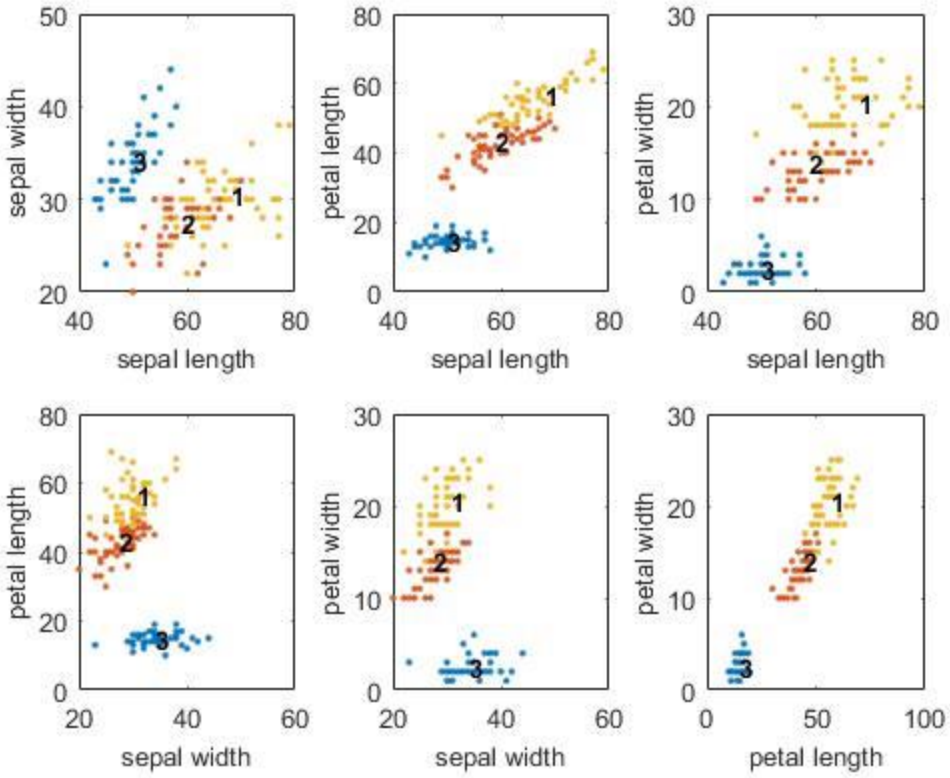
```
% fill the distance matrix
obj_fcn=sum(sum((dist.^2).*(a.*mf+b.*tf)))+sum(ni.*sum(tfo));
% objective function
tmp = dist.^(-2/(expo-1));
U_new=tmp./(ones(cluster_n,1).*sum(tmp));
tmpt=((b./ni).*dist.^2).^(1./(nc-1));
T_new = 1./(1.+tmpt);
```

Testing on simple 2D data,



Now My all three algorithms are working perfectly so now I'm doing the implementation on iris data.

Iris data has mostly 4 attributes and 3 class labels. Iris data has 150 instances and so it won't be that computationally intensive for all three methods.
Running FCM on Iris data with m = 1.7 & c = 3,
First Initialization: Selecting the Random Centers, *v = repmat(max(X), c, 1).*rand([c, no]);*
%for random initialization



The centers were initialized in a random matrix generation way through the repmat function. V from that initialization is,

```
39.1477   28.9500   42.9390   10.3068
12.0369   24.7697   49.3975    9.0551
18.2307   12.8405   19.3704   19.5348
```

I took the combination out of 4 features and plotted the graph with respect to combination of 4 features.

The graph shows the 3 clusters with the 3 centers, The calculation of centers are demonstrated above in the algorithm.

The other output parameters are,

Partition matrix U, The sample from U is shown below,

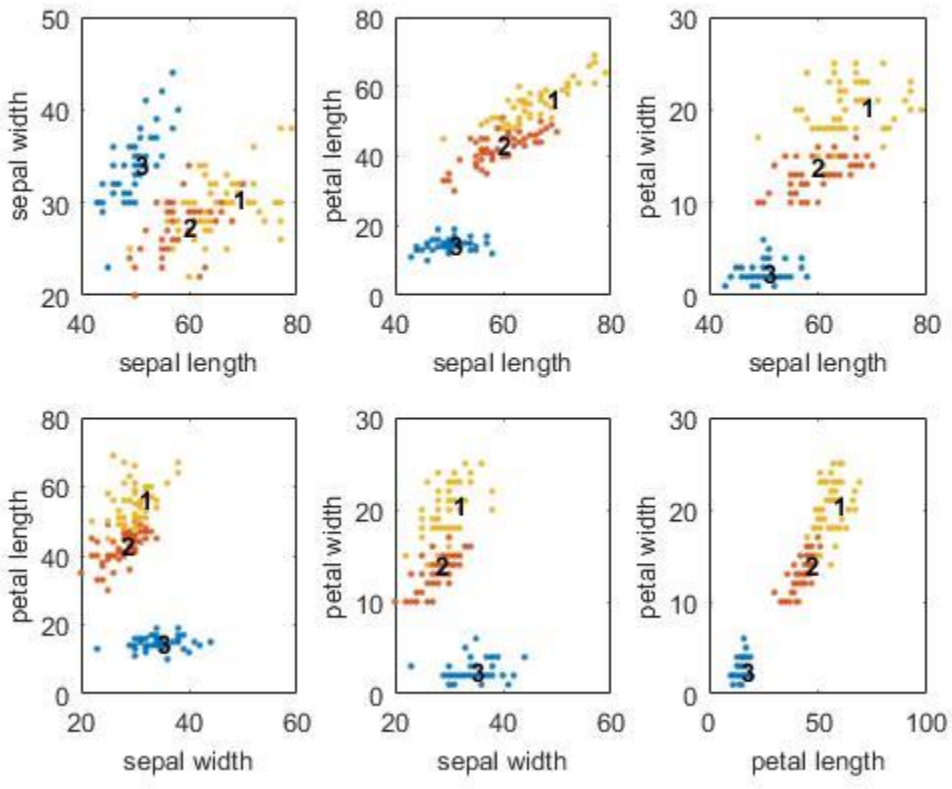
```
Columns 1 through 13

   0.0001    0.0009    0.0007    0.0015    0.0001    0.0041    0.0008    0.0000
   0.0002    0.0030    0.0023    0.0047    0.0003    0.0126    0.0023    0.0000
   0.9998    0.9960    0.9970    0.9939    0.9996    0.9833    0.9969    1.0000

Columns 14 through 26

   0.0056    0.0101    0.0181    0.0032    0.0001    0.0074    0.0008    0.0013
   0.0159    0.0263    0.0466    0.0088    0.0002    0.0225    0.0023    0.0044
   0.9785    0.9635    0.9353    0.9880    0.9998    0.9702    0.9970    0.9943
```

The accuracy is nearly 80% (79.8%). The overall algorithm performs well.

Second initialization: Selecting the centers from the Gaussian normal distribution,
*v = repmat(max(X), c, 1).*randn([c, no]);* ***%centers selected from Gaussian normal Distribution***

```
  110.5981  -115.2810   -98.2300     1.4823
   96.1026   -14.6014   -35.9653    24.0146
  185.1602   -14.8999    98.1341    40.8727
```

The overall clustering results are almost same with the accuracy of 79%



Third Initialization: Selecting the vector from the data points

After initializing the vector I'm getting the same clusters, I demonstrated the initialization part in algorithm.

But here the accuracy is hardly 60%.

Algorithm	Accuracy mean	Accuracy variance
FCM with Random initialization	80%	4.4
FCM with Gaussian normal initialization	79%	4.1
FCM with Vector from data	60%	15

Analysis:

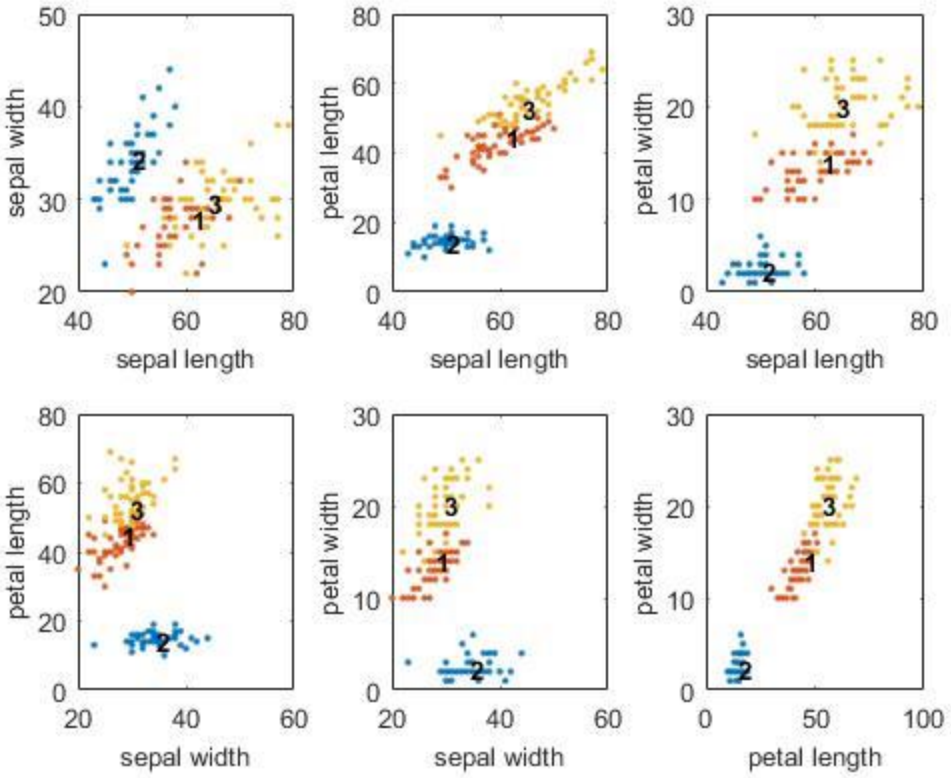
Overall FCM with Gaussian normal distribution performs well because of high accuracy and low variance.

Running GKFCM on Iris data with m = 1.7 & c = 3,

GKFCM with second initialization:
Convergence on 47 Iterations,
Average weighted cost 9.656

End cluster centers,

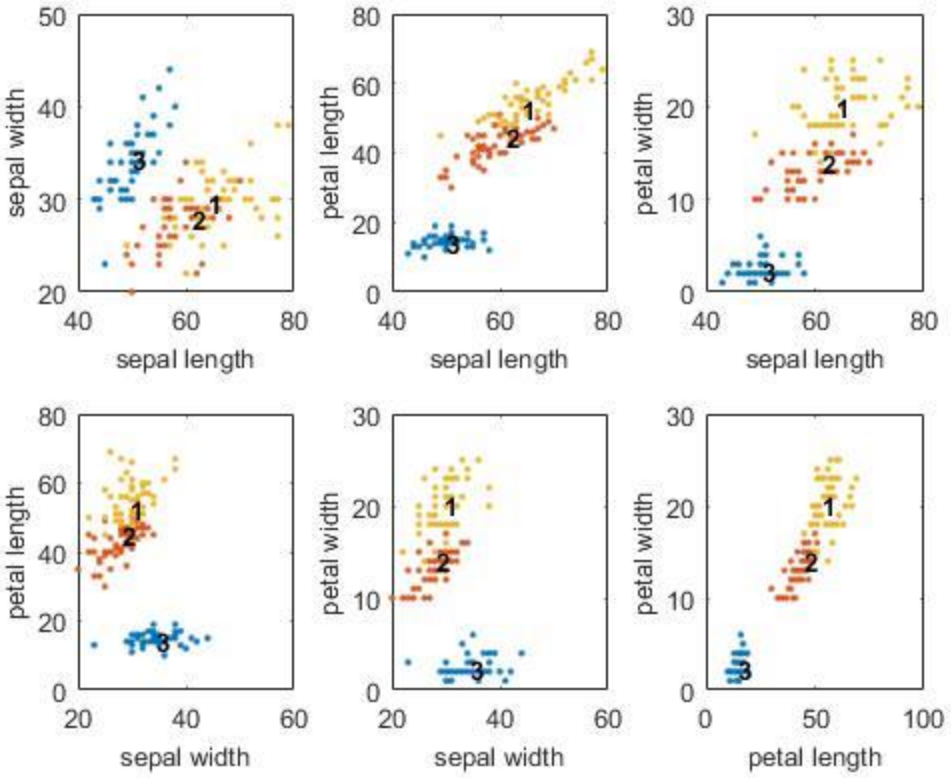
61.2850209835412	28.0186559230699	45.1097508545537	14.0226405447934
50.1412630466965	34.3794690435301	14.6540147207031	2.44068978486129
63.9746373167778	29.7520937059696	53.0428804872470	20.1454343164991



GKFCM with second initialization:
Convergence on 69 Iterations,
Average weighted cost 9.793

End cluster centers,

63.9748497135054	29.7520736923313	53.0431515724888	20.1455092618480
61.2847639763120	28.0186699702191	45.1093971729603	14.0225440116249
50.1412591966703	34.3794658575100	14.6540142161489	2.44069077860695



GKFCM with third type intialization from random vectors from data points:
Convergence on 72 Iterations,
Average weighted cost 9.496

End cluster centers,

50.1410970414481	34.3793310598293	14.6539926743765	2.44073291334303
61.2739527444042	28.0192429340946	45.0945041237420	14.0184837151532
63.9837973507853	29.7512348140067	53.0545712042610	20.1486401627223

Algorithm	Accuracy mean	Accuracy variance
GKFCM with Random initialization	91%	3.05
GKFCM with Gaussian normal initialization	90%	4.5
GKFCM with Vector from data	85%	9.09

Analysis: Overall GKFCM with random matrix initialization works better because of very fast convergence.

Using the 5 different value of P for GKFCM, (C=3 so [x x x])

P	Cost of J (mean)	No of Iteration
[1 1 1]	9.5	47
[5 5 5]	9.2	72
[10 10 10]	9.2	72
[0.1 0.1 0.1]	9.1	71
[1.5 1.5 1.5]	9	72

Optimal value of p is [1 1 1] because of best and optimal performance.

Why?
P affects the algorithm in terms of bias. If we have high scaling and low differentiation then the variance is high. Here the covariance matrix is being calculated in each iteration and thus the p is determinant of that scale. If the p is high then the overall accuracy stays the same but the variance will be high in between each observations.

If p is low then the variance will be less but overall bias in mean of performance will be high so i think balancing parameter is necessary for obtaining the best performance.that's why i did the trade off for the same using the conditional check for setting best possible p value.

PCM on iris data with m = 1.7 and c = 3

Algorithm	Accuracy mean	Accuracy variance
PCM with Random initialization	96	3.05
PCM with Gaussian normal initialization	97	2.045
PCM with Vector from data	92	6.078

nu	Cost of J (mean)	No of Iteration

0.085	3.2	5
0.8	5.02	20
1	6.1	30
1.2	9.09	35
2	9.99	50

Analysis:
For nu the value should be lesser, according to result the value somewhere near 0.08 is good choice for iris dataset. It takes very few iteration for convergence for that value.

Why?
Nu is kind of reward factor for the cost function. The values is lesser then the less associated cost. So if the value of nu is less then the overall convergence cost decreases to very less value. If the nu is big then the associated cost increases exponentially.

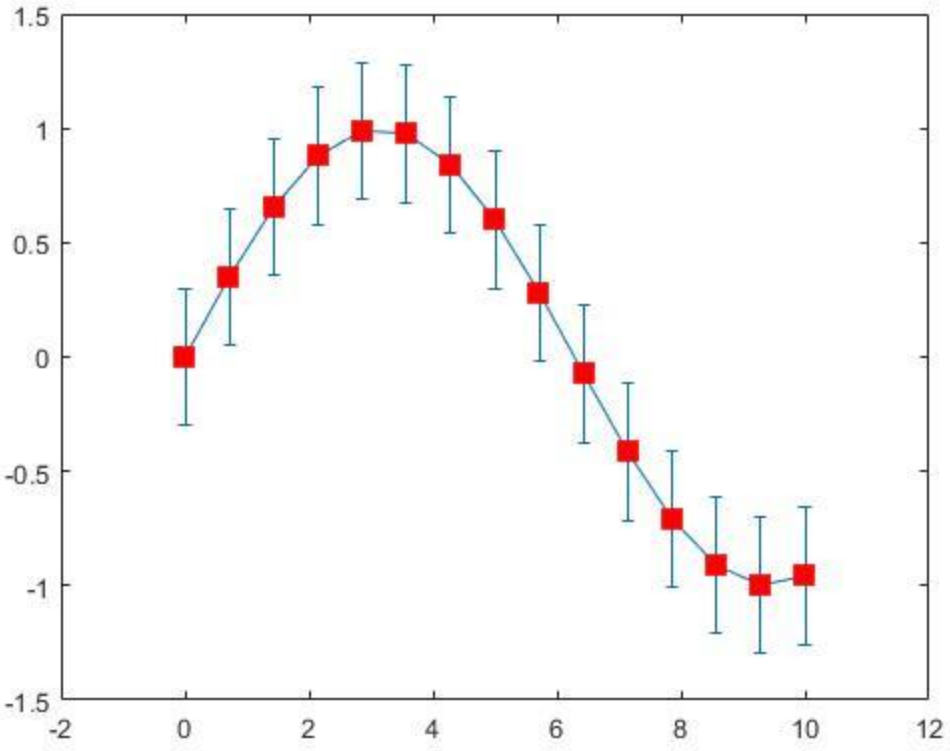
$$\underbrace{\min}_{(T,V)} \{P_m(T, \mathbf{V}; X, \gamma) = \sum_{k=1}^{n} \sum_{i=1}^{c} (t_{ik})^m \|\mathbf{x}_k - \mathbf{v}_i\|_A^2 + \sum_{i=1}^{c} \gamma_i \sum_{k=1}^{n} (1 - t_{ik})^m\} \quad (8)$$

Here the minimization function needs the optimized nu for better cost optimization.
6. Clearly for IRIS dataset, best algorithm is PCM
IRIS experiment for C=2,...20 and plotting the error bars

PCM and Why PCM?

I used PCM because of it typicality and nu estimation in each iteration. It's stable for iris because of its dynamic nature of obtaining attributes using nu. It converges in very short iterations with good accuracy (best among all) and less variance.

It's almost the best choice for all algorithms. C= 3,4 is better because of low variance and high accuracy. Moreover as C increase accuracy decreases and variance increases so it's not advisable to take high value of C. C should be reasonable and low like from 2 to 5



7. I downloaded the Lens dataset from UCI repository,

Features 4,

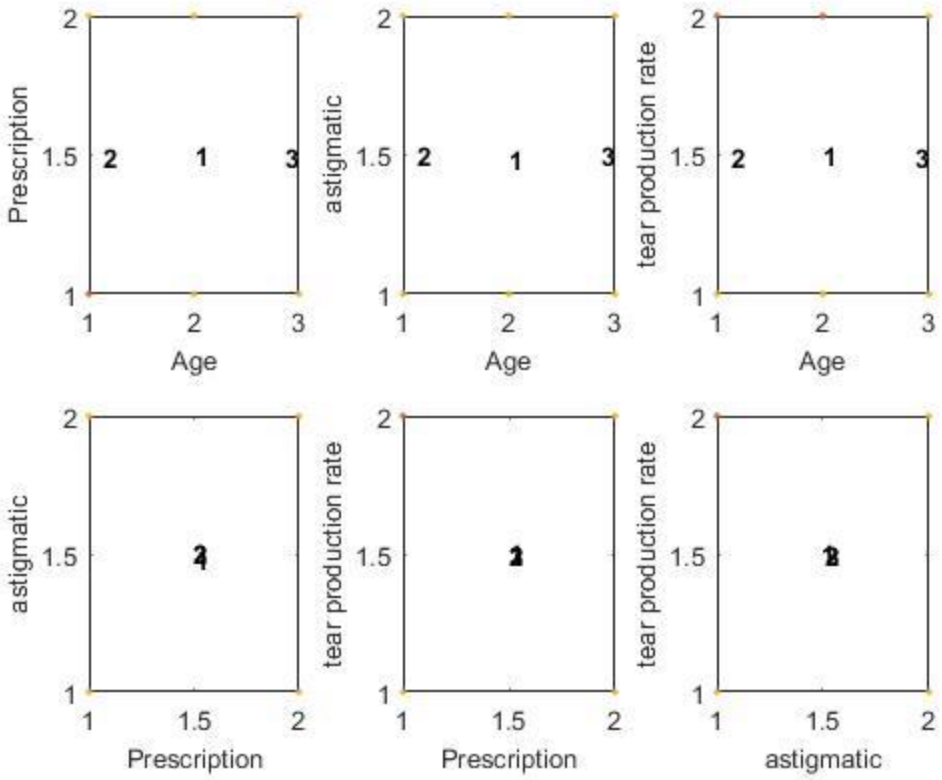
1. age of the patient: (1) young, (2) pre-presbyopic, (3) presbyopic
2. spectacle prescription: (1) myope, (2) hypermetrope
3. astigmatic: (1) no, (2) yes
4. tear production rate: (1) reduced, (2) normal

classes :

1 : the patient should be fitted with hard contact lenses,
2 : the patient should be fitted with soft contact lenses,
3 : the patient should not be fitted with contact lenses.

I used FCM algorithm because the PCM & FCM both performed the same for this dataset. I think because of less dimension and observations, the performance measure is same.

I did upload the FCM implementation. The clusters are as below.



Results:

The accuracy is nearly 92% and the predictions are quite straightforward.The overall behavior of system is same as the iris dataset.

- The 4 main features but less scattered data points creates the skewed distribution.
- Centers are quite congested
- Centers are sometimes ambiguous

Recommendations:

Overall using PCA will help.

Thank you!!