COMPM012 - Coursework 3

Esben A. Sørig

November 29, 2014

1 Practical

1.1 k-means implementation

```
function [clusterings, centers] = mykmeans(X, k)
    % Randomly initialize centers of clusters
    c = datasample(X, k, 'Replace', false);
    r = repmat(0, size(X, 1), k);
    oldr = 1; % something that is not equal to r initially

    dist = @(x, y) norm(x-y);

    % Loop as long as the clustering is changing
    while ~isequal(r, oldr)
        oldr = r;

        % Assign points to clusters
        for i = 1:size(X,1)
            cluster = 1;
            for j = 1:k
                if dist(X(i, :), c(j, :)) <  dist(X(i, :), c(cluster, :))
                    cluster = j;
                end
            end

            r(i, :) = [repmat(0, 1, cluster-1) 1 repmat(0, 1, k-cluster)];
        end

        % Update center positions
        for i = 1:k
            npoints = 0;
            c(i, :) = 0;
            for j = 1:size(r, 1)
```

```
                c(i, :) = c(i, :) + r(j, i)*X(j, :);
                npoints = npoints + r(j, i);
            end
            c(i, :) = c(i, :)/npoints;
        end
    end

    centers = c;
    % Output formatting (vector with cluster index for each row in X)
    clustering = repmat(0, size(X,1), 1);
    for i = 1:size(X,1)
        for j = 1:k
            if r(i, j) == 1
                clustering(i) = j;
                break;
            end
        end
    end

    clusterings = clustering;
end
```

1.2 k-means test on data generated from three gaussians

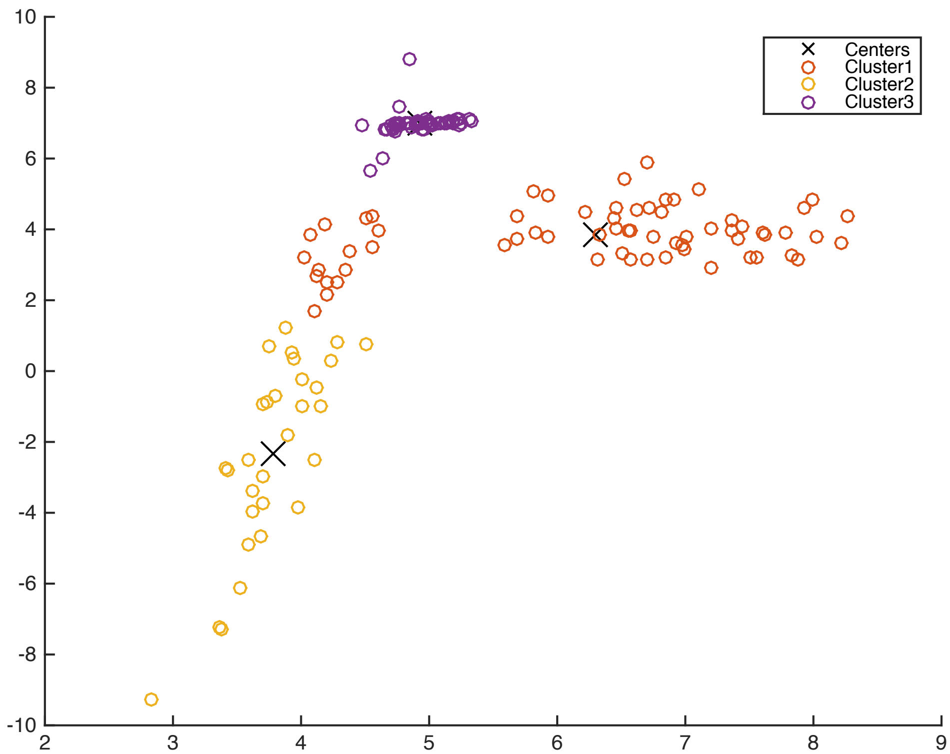
Firstly, we generate the data and run the k-means algorithm on it.

```
% Generate data
data = genData2;

% Put data for each cluster in a seperate list
cluster1 = [];
cluster2 = [];
cluster3 = [];
[clusterings, centers] = mykmeans(data, 3);

for i = 1:size(data, 1)
    if clusterings(i) == 1
        cluster1 = [cluster1; data(i, :)];
    end
    if clusterings(i) == 2
        cluster2 = [cluster2; data(i, :)];
    end
    if clusterings(i) == 3
        cluster3 = [cluster3; data(i, :)];
    end
end
```

We can now plot the clusters the algorithm has found

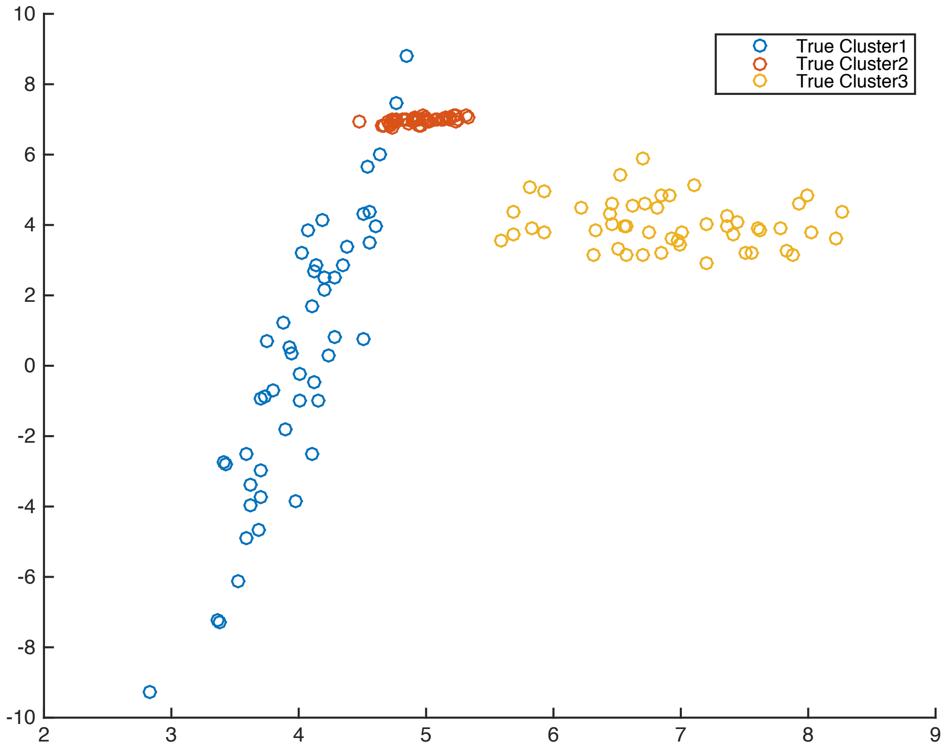
```
figure
```

```
hold on

scatter(centers(:, 1), centers(:, 2), 200, 'xblack');
scatter(cluster1(:, 1), cluster1(:, 2));
scatter(cluster2(:, 1), cluster2(:, 2));
scatter(cluster3(:, 1), cluster3(:, 2));
legend('Centers', 'Cluster1', 'Cluster2', 'Cluster3');
hold off
```



Comparing this to the true classifications

```
figure
hold on
scatter(data(1:50, 1), data(1:50, 2));
scatter(data(51:100, 1), data(51:100, 2));
scatter(data(101:150, 1), data(101:150, 2));
legend('True Cluster1', 'True Cluster2', 'True Cluster3');
hold off
```



We clearly get an intuition of how the k-means algorithm is performing on the data. Please see appendix 4.1 for the code that creates the animation of the convergence to a solution for this dataset.

We measure the mean error and standard deviation of the error of the clustering over a 100 runs of the algorithm.

```
%% Error measurements
errors = [];
for j = 1:100
    [clusterings, centers] = mykmeans(data, 3);

    % keep track of the classifications of each true cluster
    firstcluster = [0,0,0];
    secondcluster = [0,0,0];
    thirdcluster = [0,0,0];
    for i = 1:50
        firstcluster(clusterings(i)) = 1 + firstcluster(
    clusterings(i));
    end
    for i = 51:100
```

```
        secondcluster(clusterings(i)) = 1 + secondcluster(
    clusterings(i));
    end
    for i = 101:150
        thirdcluster(clusterings(i)) = 1 + thirdcluster(
    clusterings(i));
    end
    % We assume that the mode of the classifications of a true
    cluster is the class of the true cluster. Therefore, the
    error of a cluster is the frequency of other classifications
    appearing for that cluster.
    firstcluster = sort(firstcluster);
    secondcluster = sort(secondcluster);
    thirdcluster = sort(thirdcluster);
    misclassifications = firstcluster(1) + firstcluster(2) +
    secondcluster(1) + secondcluster(2) + thirdcluster(1) +
    thirdcluster(2);
    errors = [errors misclassifications/150];
end

meanerror = mean(errors)
stddeviationerror = std(errors)
```

Which returns:

```
meanerror =
    0.1438
stddeviationerror =
    0.0155
```

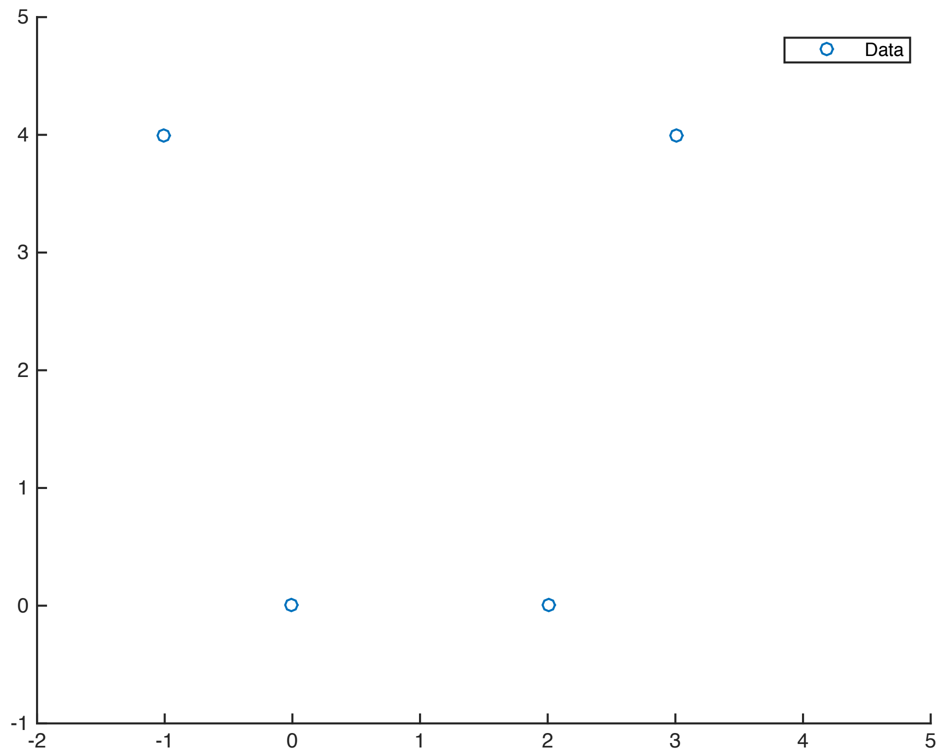
2 Questions

2.1 Dataset with local minima

We can easily construct a dataset on which our k-means algorithm has local minima. An example of such a dataset is the set $\{(0,0),(2,0),(-1,4),(3,4)\}$. This is clear if we plot the data.

```
X = [0, 0; 2,0; -1 4; 3 4];

hold on
axis([-2 5 -1 5]);
scatter(X(:, 1), X(:, 2));
```

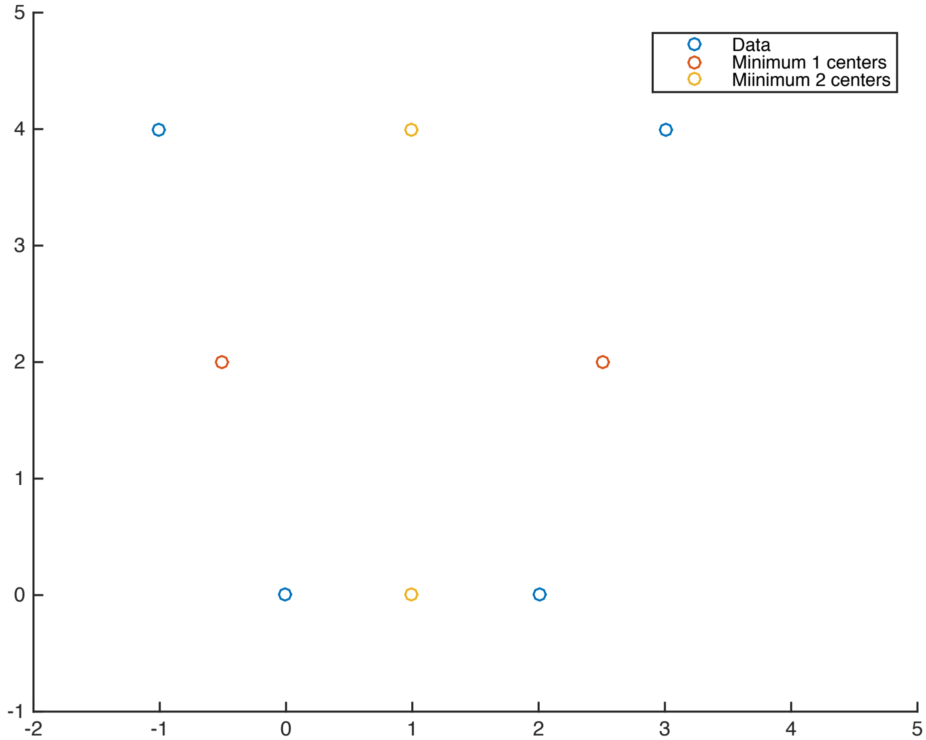
Which gives the plot:



Running k-means on this dataset will converge in one of two local minima. We run the algorithm twice and plot the centers of the clusters: (in order to get two different results, this code may have to be run several times)

```
[clustering1, centers1] = mykmeans(X, 2);
[clustering2, centers2] = mykmeans(X, 2);
legend('Data');

scatter(centers1(:, 1), centers1(:, 2));
scatter(centers2(:, 1), centers2(:, 2));

legend('Data', 'Minimum 1 centers', 'Miinimum 2 centers');
hold off
```

Which produces:



Here we clearly see the two local minima of the convergence on the four data points.

2.2 Argument that the centroid is the minimizer of the sum of squared distances

We can minimize the summed squared error

$$SSE = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} ||\mathbf{x} - \mathbf{c}_i||^2 = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{c}_i)^2 \tag{1}$$

for the k^{th} centroid by setting the derivative with respect to the k^{th} centroid equal to zero.

$$
\begin{aligned}
\frac{\delta}{\delta \mathbf{c}_k} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{c}_i)^2 &= \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \frac{\delta}{\delta \mathbf{c}_k} (\mathbf{x} - \mathbf{c}_i)^2 \\
&= \sum_{\mathbf{x} \in C_k} 2(\mathbf{x} - \mathbf{c}_k) = 0 \\
\implies \sum_{\mathbf{x} \in C_k} \mathbf{c}_k &= \sum_{\mathbf{x} \in C_k} \mathbf{x} \\
\implies \mathbf{c}_k &= \frac{1}{\sum_{\mathbf{x} \in C_k} 1} \sum_{\mathbf{x} \in C_k} \mathbf{x}
\end{aligned}
\tag{2}
$$

We see that the minimizer of $\mathbf{c}_k$ is equivalent to the centroid of the cluster.

2.3 Proof of convergence in finite amount of steps

We know that the k-means algorithm converges but we do not know if it does so in finitely or infinitely many steps. Here we show that the k-means algorithm does indeed converge in finitely many steps.

Proof: The k-means algorithm stops once there is no longer any improvement in the clustering. It will therefore never reach the same clustering twice. Each possible k-clustering of the input consists of k subsets. The number of subsets of the input is a finite number and the number of k subsets of the input is bounded above by this. Consequently, there is a finite number of possible clusterings. Hence the k-means algorithm will terminate after a finite number of iterations.

3 Extension

3.1 k-means segmentation

The objective of the algorithm is to find the segmentation $\{i_1, ..., i_{k-1}\}$ of a sequence of points $\{\mathbf{x}_1, ..., \mathbf{x}_l\} \in \mathbb{R}^n$ that is the global minimum of the following optimisation problem:

$$
\underset{i_1,...,i_{k-1};\mathbf{c}_1,...,\mathbf{c}_k}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{p=i_{j-1}+1}^{i_j} ||\mathbf{x}_p - \mathbf{c}_j||^2 \tag{3}
$$

The algorithm must run in polynomial time in l, n, and k. We start by defining the error function as it will be useful when searching for minima of in the error.

```
function e = error(delimiters, X)
    e = 0;
    bounddelimiters = [0 delimiters size(X, 1)];
    for j = 2:size(bounddelimiters, 2)
        indeces = (1+bounddelimiters(j-1)):bounddelimiters(j);
        segment = X(indeces, :);
        centroid = sum(segment)/size(segment, 1);
        for k = 1:size(segment, 1)
            e = e + norm(segment(k, :)-centroid)^2;
        end
    end
end
```

We can now proceed to design the segmentation algorithm. Consider the set of global minimum delimiters $\mathbf{i}^* = \{i_1^*, ..., i_{k-1}^*\}$. Observe that each i_j^* is the delimiter that given $\mathbf{i}^* \setminus \{i_j^*\}$ minimises the error of $\mathbf{i}^*$. That is, each i_j^* is the delimiter that minimises the error when we add it to the set of the other delimiters. We can use this observation to construct a polynomial time algorithm.

The idea is to start with an empty set of optimal delimiters, i.e. $\mathbf{i}_0^* = \emptyset$ where $\mathbf{i}_n^*$ denotes the set of n optimal delimiters (which produces $n+1$ segments). From our observation it follows that $\mathbf{i}_n^* = \mathbf{i}_{n-1}^* \cup \{i_n^*\}$ where i_n^* is the delimiter that minimises the error of $\mathbf{i}_{n-1}^* \cup \{i_n\}$. This easily translates to a recursive algorithm:

```
function delimiters = segmentation(X, k)
    delimitercount = k-1;
    if delimitercount <= 0
        delimiters = [];
        return;
    end

    delimiters = segmentation(X, k-1);
    mindelimiters = [];
    for i = 1:(size(X,1) - 1)
        newdelimiters = sort([delimiters i]);
        if error(newdelimiters, X) < error(mindelimiters, X)
            mindelimiters = newdelimiters;
        end
    end
    delimiters = mindelimiters;
end
```

Informal argument of time complexity: The running time is polynomial in l, n, and k. The the recursive call is performed $k-1$ times. At each recursive call, a loop iterates l times, at each iteration calling the error function. The error function has two nested loops iterating over the k segments delimiters and each of the, on average, l/k points in each segment. It is assumed that the summation function used in the error function has a complexity of $O(l \times n)$. Without going in further detail it seems that the running time is around $O(k \times n \times l^2)$ (may be different under certain conditions such as $k \times \log k > n \times l$ in which case sorting the delimiters has higher complexity than calculating the error) and definitely not larger than polynomial in k, n, and l.

3.2 (p, k)-means

4 Appendix

4.1 Movie generation code

```
%% Movie
X = data;
k = 3;
% Randomly initialize centers of clusters
c = datasample(X, k, 'Replace', false);
r = repmat(0, size(X, 1), k);
oldr = 1; % something that is not equal to r initially
movie = [];

dist = @(x, y) norm(x-y);

% Loop as long as the clustering is changing
while ~isequal(r, oldr)
    oldr = r;

    % Assign points to clusters
    for i = 1:size(X,1)
        cluster = 1;
        for j = 1:k
            if dist(X(i, :), c(j, :)) <  dist(X(i, :), c(cluster, :))
                cluster = j;
            end
        end

        r(i, :) = [repmat(0, 1, cluster-1) 1 repmat(0, 1, k-cluster)];
```

```
    end

    %MOVIE GENERATION
    % Split the data into the three clusters
    cluster1 = [];
    cluster2 = [];
    cluster3 = [];

    for i = 1:size(X, 1)
        if r(i,1) == 1
            cluster1 = [cluster1; X(i, :)];
        end
        if r(i,2) == 1
            cluster2 = [cluster2; X(i, :)];
        end
        if r(i,3) == 1
            cluster3 = [cluster3; X(i, :)];
        end
    end

    % Plot the centers and the three clusters
    hold on
    scatter(c(:, 1), c(:, 2), 200, 'xblack');
    if size(cluster1, 1) ~= 0
        scatter(cluster1(:, 1), cluster1(:, 2));
    end
    if size(cluster2, 1) ~= 0
        scatter(cluster2(:, 1), cluster2(:, 2));
    end
    if size(cluster3, 1) ~= 0
        scatter(cluster3(:, 1), cluster3(:, 2));
    end
    legend('Centers', 'Cluster1', 'Cluster2', 'Cluster3');
    hold off
    movie = [movie getframe];
    clf;
    % MOVIE GENERATION OVER

    % Update center positions
    for i = 1:k
        npoints = 0;
        c(i, :) = 0;
        for j = 1:size(r, 1)
            c(i, :) = c(i, :) + r(j, i)*X(j, :);
            npoints = npoints + r(j, i);
        end
        c(i, :) = c(i, :)/npoints;
    end
end
```

```
movie2avi(movie, 'k-means.avi', 'fps', 1);
```