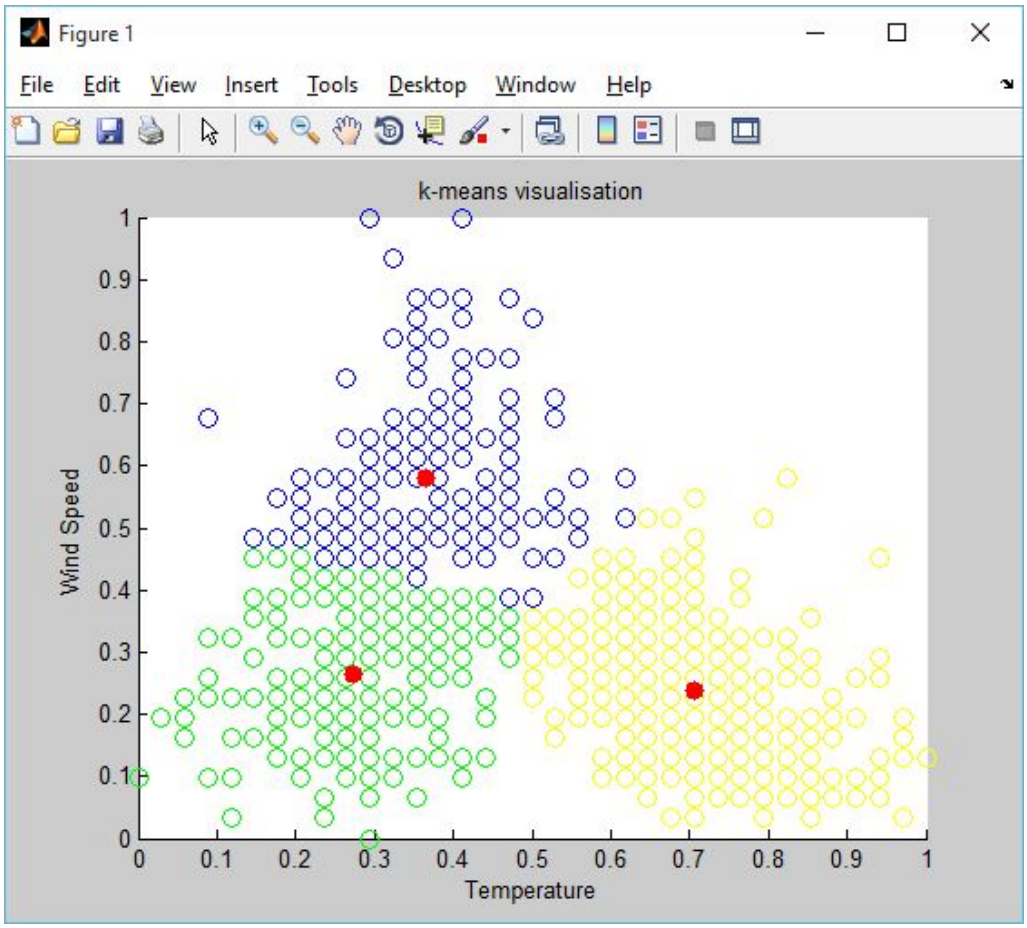
K-means Clustering

In this task we were asked to create an implementation of a k-means algorithm function to be used in matlab using my own programming knowledge and my understanding of the algorithm. In this report I will be talking about what I have learned during this task including the functionality and workings of the algorithm as well as the advantages and disadvantages of it usage drawing from the data set provided in the lab sessions. I will then discuss other available clustering algorithms including agglomerative hierarchical and the advantages, disadvantages and performance of said algorithms vs each other and against k-means.

A clustering algorithm is used in cluster analysis to aid in grouping a set of objects in that each object in the same group are more similar to each other than those in other groups.(we call these groups clusters)

Cluster algorithms can be used to identify data points that share similar attributes, most commonly when graphing two sets of data against each other. For example, In this task I primarily looked at the correlation between Wind speed and temperature when developing my kmeans function. Clustering algorithms can also be used for for more complex applications containing 3+ datasets however for the sake of visualisation I have decided to use only 2 sets, allowing me to display my results on a 2d graph, an example of the results that can be outputted with my k means function (can be seen in the appendix at the end of this report) can be seen below using the data set supplied in labs, specifically the wind speed and temperature columns using 3 centroids (indicated in red).



K-means functionality

In this section I will be discussing the workings of k-means, specifically my implementation of the function.

K-means is an example of centroid based, this means that each of the clusters are represented by a central vector, which I will be referring to as the centroid. This may or may not be a point which is a member of the data set, in the case of my kmeans implementation the centroid is a separate entity although there exists k means implementations that use random data points as the initial point of the centroid, my centroids are simply randomly placed within the bounds of the normalised data when the function is initialised. In the case of my implementation, I will be using a separate point as the centroid, which will be overlaid as a red dot on the visualisation of the data, as seen in the above example.

My k-means is a fairly simple concept as it is centered around 2 main subroutines but firstly, there is a small amount of preprocessing involved. the data inputted is required to be normalised, this allows for a better comparison between 2 different data types such as wind speed and temperature. Once the data is normalised, I create random coordinates within the bounds of the data to act as the initial points for the centroids. Another option would be to use random pre existing samples as the initial points, but for the sake of having zero bias, I have opted to go with the former method.

I then begin the iterations of the 2 main subroutines the first is the generation of a partition matrix which is used to note which points are allocated to which cluster based on distance,this matrix is then used in the second subroutine, which realigns the centroids to the center of the cluster of point allocated to it. These steps are then repeated until no more changes are made or the max iteration limit is reached(this is defined by the user). Below is a very simple pseudo code for my kmeans function

```
import dataSet, K
create K centroids with random coordinates

f1()  //initiates partition matrix in correspondence to initial
      //centroid locations

WHILE a change in the matrix is detected OR max iterations hit
      f2()
      f1()

function f1
      For each point in dataSet
            Find closest centroid
            Record in partition matrix

function f2
      FOR each centroid
            find closest points via matrix
            Move centroid to average location of points

plotData()
```

My K-means implementation is therefore, by definition, a recursive function as it calls itself until a condition is met(no more changes are detected).

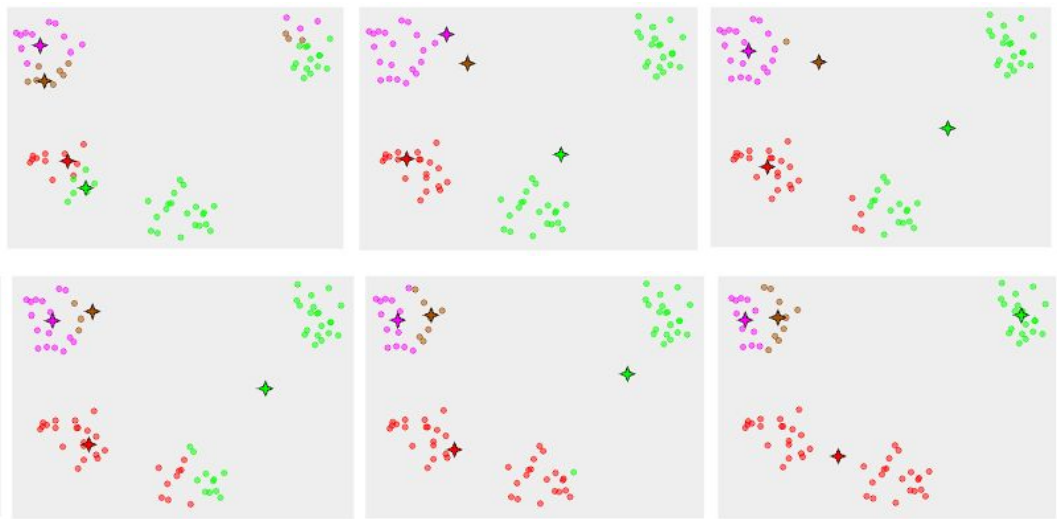
Multiple times during this report I have referred to a *partition matrix*, this is an efficient way of storing which point is allocated to which centroid, a simple example of a k-means partition matrix can be seen below, this example contains 2 centroids representing a total of 7 points.

```
partitionMatrix =

centroid 1    0    0    0    1    0    1    0
centroid 2    1    1    1    0    1    0    1

                data points 1-7
```

this clearly shows that points 1,2,3,5 and 7 are allocated to centroid 2 while points 4 and 6 are allocated to centroid 1. This matrix can then aid the function when finding the average location of the points assigned to each centroid as you would simply scan each entry and at every point there is a ‘1’, you would refer to that entry the data set, eventually calculating the average, allowing you to move the centroid to a new centered location.

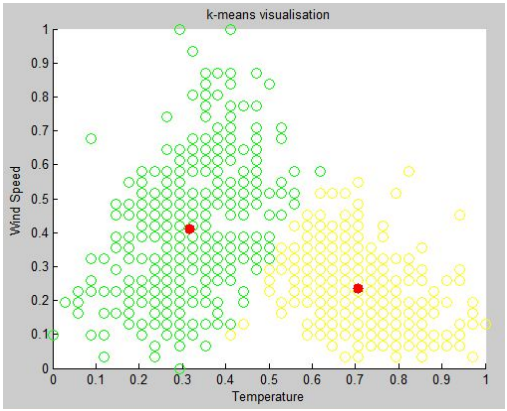
The centroid is truly centered when no more changes to the matrix are made, thus concluding the algorithm.

Below is an example[1] of the progression of k means algorithm, the stars representing the centroids, the coloured points representing the data points and which cluster they belong to.



[1] Mirkes, EM. "K-means and K-medoids applet." *University of Leicester* (2011).

Conclusions draw from lab data set



Looking at the case study we were provided in labs, we can use k means to aid us in analysing data samples, in this example, we'll look at the correlation between wind speed and temperature. at first look, this data seems to be fairly centralised, with few immediately noticeable clusters, however, after trying differing numbers of centroids I believe certain conclusions can be drawn from this data.

It appears to me that there are 2 main groups in this example, the first(green) cluster appears to trend upwards to a point, we see that as the temperature increases, the maximum threshold for wind speed does also. However, what we see in the yellow cluster indicates that when high temperatures occur, it is unlikely that they will be accompanied by high wind speed resulting in sizable subset of data points occurring at high temperatures and but wind speed. This result was consistent amongst all tests I carried out, Although I did note differing numbers of iterations of the main function loop, this will be a result of the randomly placed centroids during preprocessing.

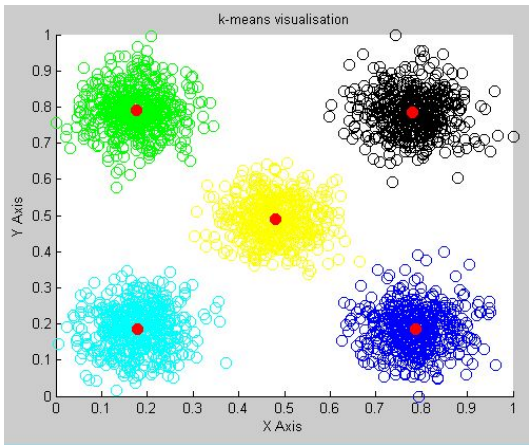
Advantages and disadvantages of K-means Algorithm

With there being many clustering algorithms available, it is valuable to understand the advantages and disadvantages of k means over other types of cluster analysis. This section we will be looking at the inherent advantages associated with k means as well as the disadvantages before comparing the method with other methods.

Advantages

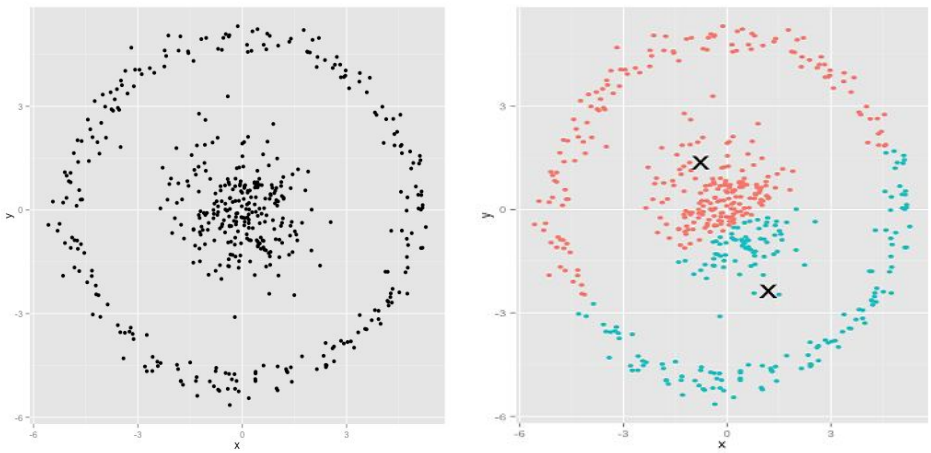
- Low complexity
    - k-means operates as the complexity O(N) 'order of N where n is number of samples',
    - This is very quick in comparison to other clustering algorithms (which I will be discussing later in this report). This is of crucial importance to systems that rely on speed and efficiency as a key performance indicator. This benefit becomes more obvious as the number of samples increases
- gives best results on distinctly separated data
    - When data is distinctly separate, k means is very quick and accurate, an example of this is shown below using a custom data set I created and using my kmeans algorithm.

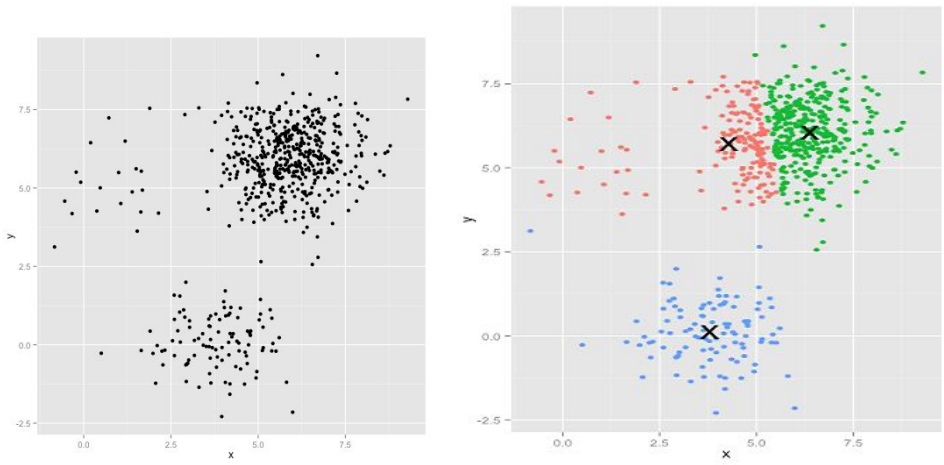
    - This example took only 4 iterations of the main loop, at a complexity of only O(N) this is incredibly quick.
- easy to implement
    - The implementation of k means is fairly simple, not involving many complex functions or difficult mathematics, making it superb example for simplistic cluster analysis done at great speed.



a

Disadvantages

- Initial centroid distribution/partition can affect performance
    - As centroids are randomly distributed in k means (excluding the method of assigning centroids to preexisting samples, which is only partially random), the placement of the centroids can play a large role in the speed of the system.
    - for example, if by chance the centroids are placed exactly or nearly exactly on the eventual final location, it may only take 1 or 2 iterations to complete. Conversely if they are placed far away from the final points, or if the centroids are placed very close together it may take longer to complete the function.
    - This may prove worrisome for time based systems as you can be unsure how long it will take to complete.
- poor at handling of non spherical data
    - by non spherical I mean data values that are not tightly clusters and of fairly uniform distribution, the best way to describe this issue is with an example, below is a distribution of values alongside the k means cluster of the distribution
    - 
    - To the human eye the 2 groups of points seem obvious however k means struggles to differentiate groups that surround each other.

    - This cluster would however be detected using agglomerative hierarchical clustering (discussed later)
- poor handling of uneven/noisy data
    - due to how kmeans weights the value of points, areas of high concentration are values more, this causes some issues when identifying clusters, see the example below.
    - 
    - As you can see, the more concentrated cluster has been priority in the algorithm despite the data clusters being fairly obvious to the human eye.
- outlying centroid can fail to register
    - This is an issue i've found when using smaller, more spread out data sets, there is a small chance that a centroid is initially placed too far from any points to be classed as the closest to any particular data value causing no data points to ever be assigned to that centroid, meaning it will never move and never be assigned to any cluster.
    - This can be countered by initially assigning your centroids to pre existing data values as discussed earlier.
- use of exclusive assignment
    - If there are two overlapping clusters, k means will often be unable to resolve the presence of two clusters at all, preferring to group them as one.

Other clustering algorithms

There exists many other clustering algorithms, in this final section we will be discussing a few, along with the advantages and disadvantages of each.

Agglomerative Hierarchical

The goal of agglomerative hierarchical clustering is to ensure that all nearby points end up in the same cluster, this can therefore be used to counter some of the issues mentioned earlier in the disadvantages of k-means, specifically when dealing with non spherical data.
Hierarchal is 4 step process and consists of the following

1. assign each point to its own cluster
2. find the closest 2 points and combine them into the same cluster
3. work out distances between new cluster and each of the old cluster

4. repeat steps 2 and 3 until all items are in a single cluster

Now that all values are joined, you can decide the level of clustering you wish using a dendrogram.

Advantages

- does not require information relating to the number of clusters.
- Gives better results than k-means in some cases.
    - This includes non spherical data clusters or clusters that are of an irregular shape
    - Many of the disadvantages mentioned in relation to k means clusters can be resolved with agglomerative hierarchal clustering

Disadvantages

- Very poor complexity of $O(n^2 \log n)$ with n being the number of data values makes large data sets extremely grueling to get through

When comparing hierarchical clustering to k-means you can make a few conclusions, firstly hierarchical can be considered a little more flexible, being able to process many different distributions of clusters unlike k means which can sometimes be stumped by more elaborate clusters (as mentioned earlier). However, the poor complexity means that processing large amounts of data can take a huge amount of time longer than using alternatives such as kmeans.

K Nearest Neighbor (KNN)

K nearest neighbor is in some ways similar to the agglomerative hierarchical method is that in order to assign a point to a cluster you must look to nearby points. It works by assigning a number ‘K’, K is the number of surrounding points that are checked, K must not be an even number and it must not be a must not be a multiple of the number of clusters. once you’ve checked the appropriate number of surrounding points you decide which cluster to join based on which is most common. For example if you have a K value of 3 and the 3 closest surrounding points are members of clusters A, A and B, your point will be assigned to the A cluster.

Advantages

- High performance within noisy data sets
- Good complexity O(N)

Disadvantages

- Useage with large data sets results in a long processing time due to having to have to check multiple points for each point in the data set.
- Many outlying data points may be allocated to what a human would perceive as the wrong cluster if K value is too low, however increasing value only makes it slower.
- Struggles with non spherical distributions, much like K-Means.

Conclusion

During this project I feel I have greatly improved my knowledge of not only kmeans, but a collection of different clustering algorithms. I have expanded my understanding of the functionality and applications of such algorithms. For example, I now understand how the ease of implementation associated with k-means, alongside its impressive O(N) complexity make it an attractive choice for many applications, however it's inability to handle complex clusters limits it for some purposes, meanwhile other algorithms can contend with this but can only do so in an inefficient manner, such as hierarchical clustering.

I also feel I have a better concept of the use of clustering algorithms in the context of artificial intelligence. K-Means is a good example of a function that can analyze and adapt without the need for human input. This is a crucial concept when considering self learning systems that can process independently of human interaction.

Finally, I am able to see this project as a springboard in terms of moving on to moving on to more complex concepts within the artificial intelligence module, giving me a good understanding of how these systems operate and the key concepts included in creating Intelligent systems.

Appendix

My k means function:

```
function [] = cw1_woodruffe_daniel(X,K, maxIt)
%Implementation of kmean function
%takes in X(dataset), K(number of clusters) and maxIt(maximum itterations,
%to prevent infinate looping under rare secumstances
%example call : cw1_woodruffe_daniel(data,4,100)

data = normal(X);    %normalise data
means = rand(K,2);   %initialise kmeans
means

partitionMatrix = zeros(K,size(X,1)); %initiallises enpty distance matrix

[data,means,partitionMatrix] = updateMatrix(data,means,partitionMatrix); %first alocation of matrix
tempMeans = []; %creates empty matrix for means to be checked against


iteration=0;
while isequal(tempMeans,means)==0&&iteration<maxIt     %if temp means == means then no change was made on last iteration, break
    tempMeans = means;  %update temp means
    [data,means,partitionMatrix] = updateMeans(data,means,partitionMatrix);         %reassign means
    [data,means,partitionMatrix] = updateMatrix(data,means,partitionMatrix);        %update the partion matrix
    iteration=(iteration+1); %logs iteration
    iteration
end
partitionMatrix
means
[data,means,partitionMatrix] = plotMean(data,means,partitionMatrix,X);        %plot results
end
```

```
function [data,means,partitionMatrix] = updateMatrix(data,means,partitionMatrix)
%updates the partition matrix according to the means

partitionMatrix = zeros( size(means,1),size(data,1) ); %clears zpartition matrix

for dataIndex=1:size(data,1) %for each data value

    closestMean = [];      %distance to closest centroid
    closestMeanDist = []; %distance to closest centroid

    for j=1:size(means,1)   %for each centroid
        i = [data(dataIndex,1:2);means(j,:)];   %these 2 lines work out the distance between the data value and the selected centroid
        v = pdist(i,'euclidean');

        if j == 1   %if its the first centroid checked it must be the closest so far
            closestMean = 1;
            closestMeanDist = v;    %distance of closest centroid
        else
            if v<closestMeanDist        %if centroid is closer than previous closest
                closestMean = j;        %set as new closest centroid
                closestMeanDist = v;    %log distance for next check
            end
        end
    end
    partitionMatrix(closestMean,dataIndex) = 1; %update datavalue entry with closest centroid
end
end
```

```
function [data,means,partitionMatrix] = updateMeans(data,means,partitionMatrix)
%updates means according to locations of asigned points

for n=1:size(means,1)   %for the number of centroids
    %loop finds all points assigned to n centroid and plots a '1' in the
    %matrix

    myX = [];
    myY = [];

    for j=1:length(partitionMatrix)     %for every data value
        if partitionMatrix(n,j)==1      %if selected datavalue is assigned to selected centroid
            myX(end+1)=data(j,1);       %log the location
            myY(end+1)=data(j,2);
        end
    end

    means(n,1) = mean(myX);     %find the average location of all the assigned points
    means(n,2) = mean(myY);     %set the centroid to this average location
end

end
```

```
function [data,means,partitionMatrix] = plotMean(data,means,partitionMatrix,X)
%plots data

for n=1:size(means,1)
    for j=1:length(partitionMatrix)
        if partitionMatrix(n,j)==1
            switch n
                case 1
                    scatter(data(j,1),data(j,2),80,'green');
                    hold on;
                case 2
                    scatter(data(j,1),data(j,2),80,'yellow');
                    hold on;
                case 3
                    scatter(data(j,1),data(j,2),80,'blue');
                    hold on;
                case 4
                    scatter(data(j,1),data(j,2),80,'black');
                    hold on;
                case 5
                    scatter(data(j,1),data(j,2),80,'cyan');
                    hold on;
                otherwise
                    warning('too many centroids, cant handle the colours')
            end
        end
    end
end

scatter(means(:,1),means(:,2),80,'red','filled');   %adds centroids

title('k-means visualisation');
xlabel('Temperature');
ylabel('Wind Speed');
%axis([min(X(:,1)) max(X(:,1)) min(X(:,2)) max(X(:,2))]);
end
```

MATrix LABoratory length : 4365 lines : 120 Ln : 111 Col : 1 S