COMP3003 Machine Learning

Exercise 1: Unsupervised machine learning techniques

Reece Davies [10572794]

Plymouth University

2020

Summary

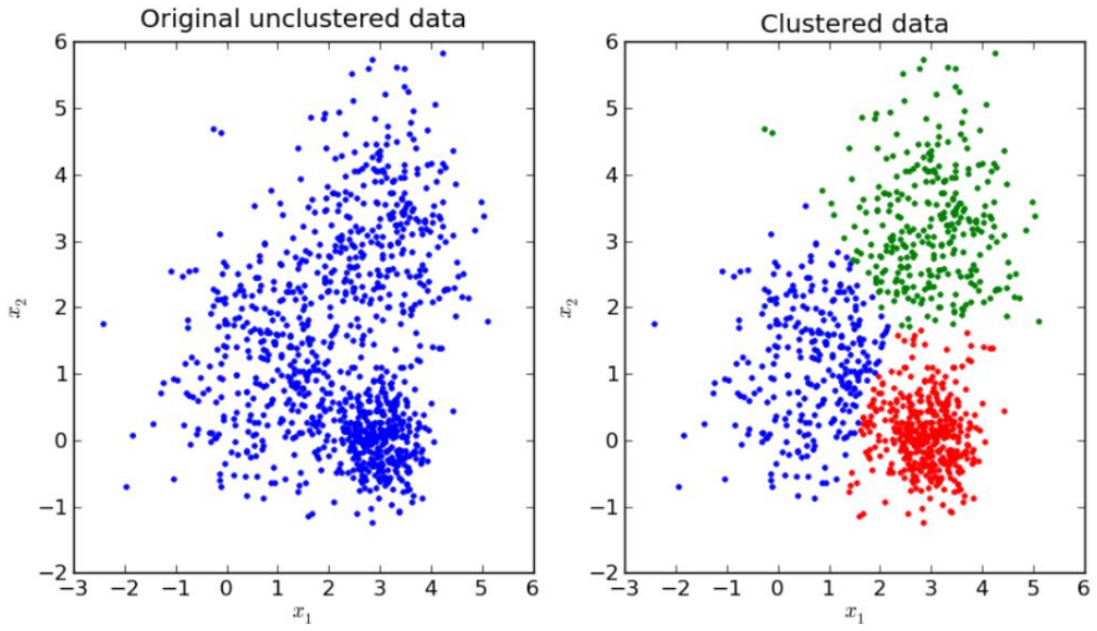
This document will analyse and explain how unsupervised machine learning can be implemented in real world scenarios, specifically with use of the technique 'Kmeans clustering'. Examples will be given of specific problems and why unsupervised machine learning techniques would prove to be beneficial in creating a solution. Furthermore, Kmeans will be implemented in the software 'MATLAB' to demonstrate the process of unsupervised machine learning with a designated dataset; code, diagrams, and results will be used in correlation to this.

Literature Review

Define Unsupervised Machine Learning

Unsupervised machine learning algorithms analyse the patterns from a dataset without being given any indication of what the values have been labelled as, or what outcome should be derived. The algorithm will independently detect patterns in the training dataset and use it as a reference for identifying similar patterns in the testing dataset. Furthermore, the primary objective of using unsupervised learning over supervised learning, is to gain a better understanding of the overall structure of the data. However, unsupervised learning can be somewhat unpredictable when compared to other learning techniques as there is no correct answer to support the testing dataset.

Unsupervised learning can be categorised into to two main categories: clustering, and association. Clustering is used to discover the inherent groups of the data by identifying recognisable patterns that define each class. Some methods of clustering are as follows: 'hierarchical clustering', 'k-means', 'mixture models', 'DBSCAN', and 'OPTICS algorithm'. Association on the other hand, is used for discovering key relations between separate objects in a large dataset by pinpointing items that frequently occur together. This document will focus on the application of Kmeans clustering: a centroid based algorithm that groups the dataset into separate clusters by calculating the distance assigned to a designated point in each cluster, known as a centroid. Each cluster is associated with a specific centroid, where the quantity of centroids is defined by 'K'. All data points will be assigned to a specific cluster, which is computed based on the set distance between the data point and all available centroids. The centroid will again recompute each cluster because of the different values that come into the group; and is again repeated until there are no changes to the output.



Real world example 1: Email spam filter

Kmeans clustering has proven to be an effective tool in identifying spam emails; businesses implement Kmeans algorithms in the user's mailbox, where it flags whether a recently received email is considered spam. The algorithm analyses the contents of the different sections of the email – header, body, and sender – which are grouped together into separate clusters. These groups are then classified as 'spam' or 'genuine'. Each email is categorised by implementing the method 'vector determination'. The primary goal of using machine learning in a spam filter is to improve the reliability of identifying spam, and therefore means it will be less likely to flag a genuine email as 'spam', specifically those coming from an automated newsletter system.

Real world example 2: Anomaly detection for fraudulent activity

Due to the rapid growth in electronic ecommerce services, there has been an immense increase of fraudulent activity, specifically in the banking industry. Fraud is the term used to describe the act of criminal deception committed specifically to secure illegitimate personal gain. In this case, fraud applies to unauthorised activity taken place in electronic payment systems. As a result of the ease of use and time efficiency of online banking, customers prefer to use an online payment method as opposed to face-to-face transactions. This of course means fraudulent transactions have also increased and therefore techniques need to be implemented to detect such crimes.

By making use of Kmeans clustering algorithms, it is possible to analyse customers' activity in search for anomalies that may potentially be acts of fraud. The Kmeans algorithm makes use of the transactions' details and groups them based on their shared similarities. These groups would most likely be dependent on the risk or probability of a particular transaction being associated with an act of fraud. For instance: K is defined as 3 and each transaction is categorised as 'low risk', 'medium risk', or 'high risk'. Furthermore, Kmeans can also be applied to detect other types of fraud or anomaly based searches, such as insurance claims, data mining for healthcare fraud, network traffic anomalies, etc.

Implementation of Kmeans in MATLAB

Generating the dataset

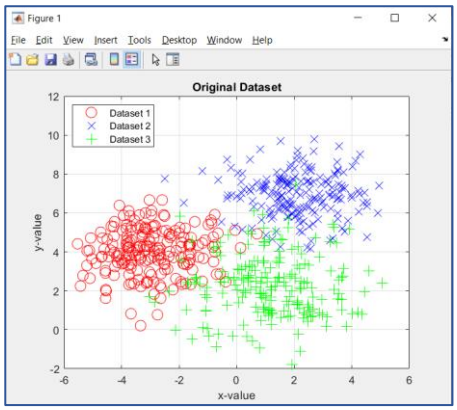
The specification demanded that the dataset was a 2D uncorrelated Gaussian distributed training dataset with a total of six hundred individual points for three separate clusters. The three clusters have been declared as "**data1**", "**data2**", and "**data3**" alongside with a designated mean and standard deviation variable. These variables have been structured in such a manner to ensure that the clusters remain separate from one another but contain certain datapoints that overlap with neighbouring clusters.

To specify – the mean controls the centre of the cluster, whilst the standard deviation is responsible for how dispersed the data is in relation to the mean's value. The larger the standard deviation, the more dispersed the datapoints will be from the mean; if the standard deviation is a smaller value however, the datapoints will be more concentrated. Each cluster is generated by multiplying the standard deviation with a 2-by-200 matrix of normally distributed random variables, which is made by using the MATLAB "**randn**" function and adding the result to a 1-by-200 array that contains duplicate values of the specified mean, that is generated with the use of the MATLAB "**repmat**" function.

```
% A.) Generate a 2D uncorrelated Gaussian distributed training dataset
samples = 200;
Mean1 = [-3; 4];
Std1 = 1.25;
Mean2 = [2; 7];
Std2 = 1.25;
Mean3 = [1; 2];
Std3 = 1.5;

data1 = Std1 * randn(2, samples) + repmat(Mean1, 1, samples);
data2 = Std2 * randn(2, samples) + repmat(Mean2, 1, samples);
data3 = Std3 * randn(2, samples) + repmat(Mean3, 1, samples);
```

Subsequently, this dataset was plotted on a two-dimension plane using the MATLAB "**plot**" function, as stated in the assignment brief. The three clusters have been given separate colours and shapes to help the user distinguish one cluster from another.

```
% Plot original training dataset
figure;
plot(data1(1, :), data1(2,:),'ro', 'MarkerSize', 10);
hold on;
plot(data2(1,:), data2(2,:), 'bx', 'MarkerSize', 10);
hold on;
plot(data3(1,:), data3(2,:), 'g+', 'MarkerSize', 10);
legend('Dataset 1', 'Dataset 2', 'Dataset 3', 'location', 'NW');
title('Original Dataset');
hold off;
xlabel('x-value');
ylabel('y-value');
xlim([-6 6]);
ylim([-2 12]);
grid on;
```

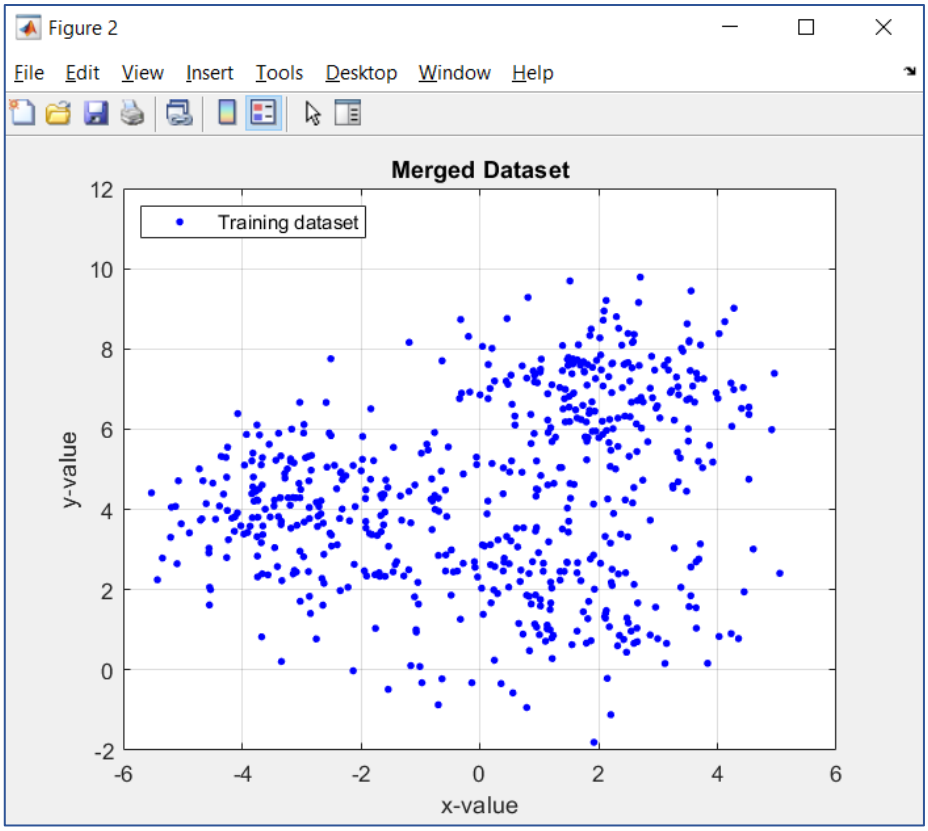


Concatenating the training datasets into a single dataset

```
% B.) Concatenate the training datasets into a single dataset
trainData = [data1 data2 data3];

% Plot merged dataset
figure;
plot(trainData(1, :), trainData(2,:), 'b.', 'MarkerSize', 10);
legend('Training dataset','location', 'NW');
title('Merged Dataset');
xlabel('x-value');
ylabel('y-value');
xlim([-6 6]);
ylim([-2 12]);
grid on;
```

The three separate clusters needed to be concatenated into a single merged dataset, as the Kmeans algorithm will need to be trained with one dataset of n number of points. This was simply created by joining the three clusters into a single matrix which was declared as "**trainData**". As a result, "**trainData**" becomes a matrix consisting of two rows and six hundred columns, with the first two-hundred columns consisting of the values from the first cluster, the second two-hundred from the second cluster, and the third two-hundred from the third cluster, respectively.

This new dataset was then plotted onto a two dimensional plane, similarly to when plotting the three clusters. Of course, the identity of each cluster is no longer shown on the graph as separate colours because this figure only consists of one dataset.



Implement Kmeans clustering

```
% C.) Implement Kmeans clustering from first principles
X = trainData';
K = 3;
max_iterations = 10;
```

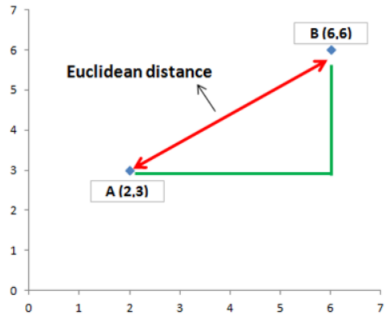
$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^{T}$$

Once the three clusters were integrated into a single dataset, it needed to be implemented into the Kmeans algorithm. Firstly, the "**trainData**" matrix was required to be transposed in order for the algorithm to function correctly; this was done by creating a new variable "**X**" that equals to the transpose of "**trainData**". To specify – transposing a matrix interchanges the row and column index for each element, and thus the end result is a matrix where the rows and columns have been replaced with one another. The variable "**K**" represents the number of clusters the dataset must be categorised into, which has been declared as 3 in this scenario. Another variable, "**max_iterations**", is to specify the number of times that the centroid of a particular cluster is calculated, with each iteration creating a more accurate outcome.

```
% Create centroids, based on the value of K
randidx = randperm(size(X,1));
centroids = X(randidx(1:K), :);

total_dist = zeros(K,1);

for i = 1:max_iterations...
```

Once K had been declared, the centroids for each cluster needed to be calculated; this was declared within the "**centroids**" variable, which becomes a 3x2 matrix. A 'for' loop is created and is executed ten times, with the use of the "**max_iterations**" variable; all centroid calculations are performed from within this loop.

Kmeans functions as an iterative method based on the "Expectation Maximisation algorithm" and once the number of clusters has been determined, the algorithm must calculate the distance between all the datapoints and each centroid. Firstly, the algorithm calculates which cluster each datapoint must be assigned to. A new 'for' loop is created and is executed for every datapoint in the training dataset, therefore it will be run six hundred times, as stated with the variable "**m**". The minimum distance is calculated between the chosen datapoint and a specific centroid, which is done with the use of the "Euclidian Distance" formula.



$$\textit{Euclidean distance}\ (a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

```
K = size(centroids, 1);
indices = zeros(size(X,1), 1);
m = size(X,1);

% Calculate which cluster each datapoint must be assigned to
for p = 1:m

    chosen_cluster = 1;
    min_dist = (X(p,:) - centroids(1,:)) * (X(p,:) - centroids(1,:))';

    for j = 2:K
        dist = sum((X(p,:) - centroids(j,:)) .^ 2);
        if (dist < min_dist)
            min_dist = dist;
            chosen_cluster = j;
        end
    end

    indices(p) = chosen_cluster;
end
```

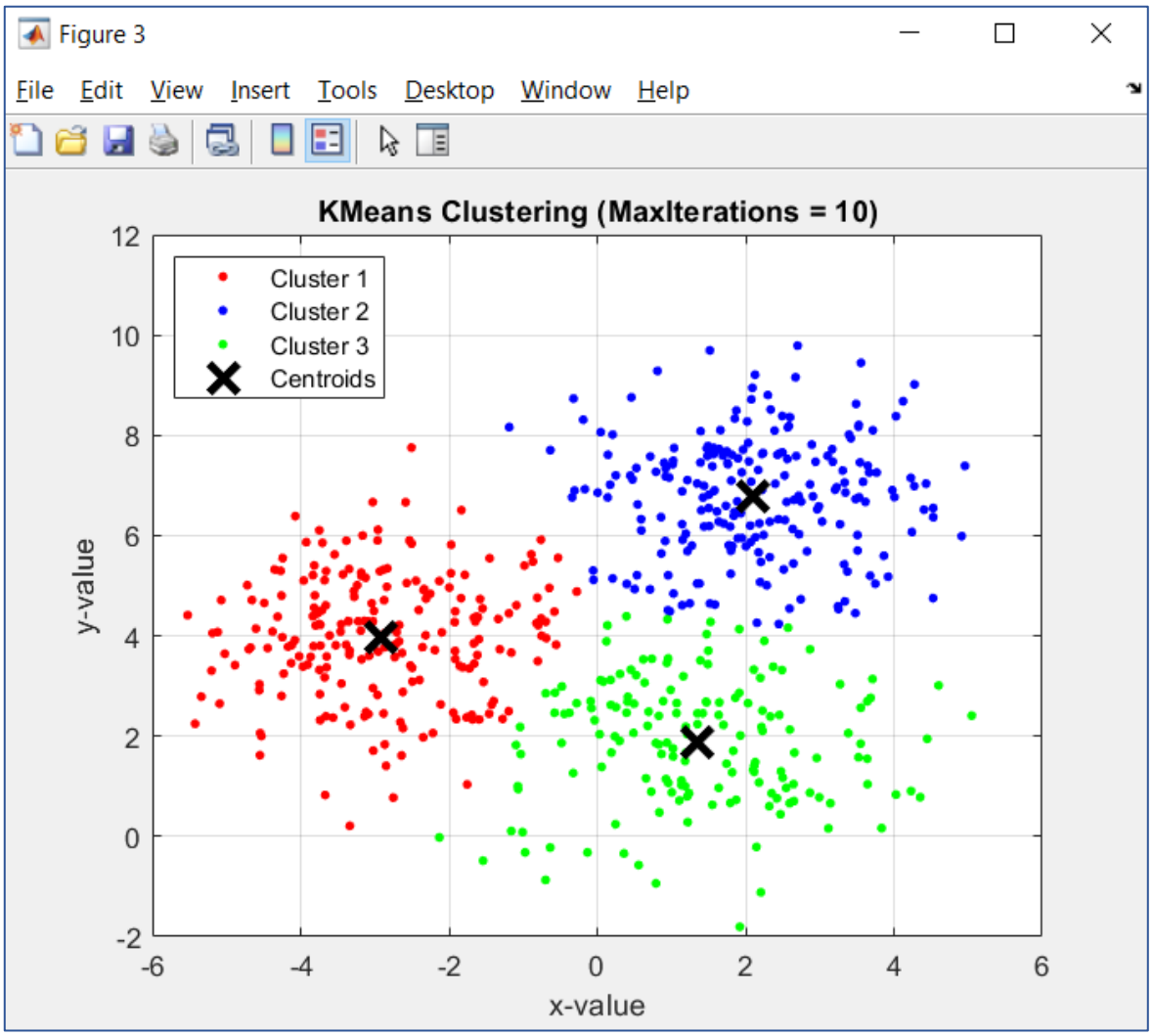
A nested 'for' loop is declared and is executed for the remaining centroids. The distance between a datapoint and the given centroid is then calculated from within this loop and assigned to the "**dist**" variable. If this distance is smaller than the minimum distance previously calculatd, the minimum distance variable is then assigned to this new value as the datapoint is closer to this specific centroid. Subsequently, the datapoint is assigned to the cluster that is dependent on the chosen centroid.

```
[m, n] = size(X);
centroids = zeros(K,n);
distances = 0;

% Calculate the centroids for each cluster
for q = 1:K
    xi = X(indices == q, :);
    ck = size(xi, 1);
    centroids(q,:) = mean(xi);
    t = (xi - mean(xi)).^2;

    % Calculate the total distance within each cluster
    distances = distances + sum(sqrt(t(:,1) + t(:,2)));
end

total_dist(i) = distances;
```

Once the different clusters have been declared, the algorithm will calculate the centroids for each cluster. With each iteration, the chosen centroid becomes more accurate to the centre of the cluster. Of course, the initial centroid may potentially be miscalculated and thus based on the outer edge of the cluster, which in turn would hinder the classification process; this is why the centroids are repeatedly calculated. Furthermore, the total distance is also calculated if the user intended to create a plot for the learning rate for this particular calculation.

To demonstrate the final results of the Kmeans algorithm, it has been plotted onto a two dimensional plan with each cluster being a separate colour, alongside with the centroids being marked as black '**X**'s.

```
% Plot Kmeans clustering implementation
figure;
message = sprintf('KMeans Clustering (MaxIterations = %d)', max_iterations);
plot(trainData(1, indices == 1),trainData(2, indices == 1),'r.', 'MarkerSize', 10);% for 1st cluster
hold on;
plot(trainData(1, indices == 2), trainData(2, indices == 2), 'b.', 'MarkerSize', 10); %for 2nd cluster
hold on;
plot(trainData(1, indices == 3), trainData(2, indices == 3), 'g.', 'MarkerSize', 10); %for 3rd cluster
hold on;
plot(centroids(1,1), centroids(1,2),'kx', 'MarkerSize',15,'LineWidth', 3); %for the 1st centroid
hold on;
plot(centroids(2,1), centroids(2,2),'kx', 'MarkerSize', 15, 'LineWidth', 3); %for the 2nd centroid
hold on;
plot(centroids(3,1), centroids(3,2),'kx', 'MarkerSize', 15, 'LineWidth', 3); %for the 3rd centroid
legend('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroids', 'location', 'NW');
title(message);
hold off;
xlabel('x-value');
ylabel('y-value');
xlim([-6 6]);
ylim([-2 12]);
grid on;
```



```
% Create centroids, based on the value of K
randidx = randperm(size(X,1));
centroids = X(randidx(1:K), :);

total_dist = zeros(K,1);

for i = 1:max_iterations

    K = size(centroids, 1);
    indices = zeros(size(X,1), 1);
    m = size(X,1);

    % Calculate which cluster each datapoint must be assigned to
    for p = 1:m

        chosen_cluster = 1;
        min_dist = (X(p,:) - centroids(1,:)) * (X(p,:) - centroids(1,:))';

        for j = 2:K
            dist = sum((X(p,:) - centroids(j,:)) .^ 2);
            if (dist < min_dist)
                min_dist = dist;
                chosen_cluster = j;
            end
        end

        indices(p) = chosen_cluster;
    end

    [m, n] = size(X);
    centroids = zeros(K,n);
    distances = 0;

    % Calculate the centroids for each cluster
    for q = 1:K
        xi = X(indices == q, :);
        ck = size(xi, 1);
        centroids(q,:) = mean(xi);
        t = (xi - mean(xi)).^2;

        % Calculate the total distance within each cluster
        distances = distances + sum(sqrt(t(:,1) + t(:,2)));
    end

    total_dist(i) = distances;
end
```

Applying the Kmeans algorithm to a testing dataset

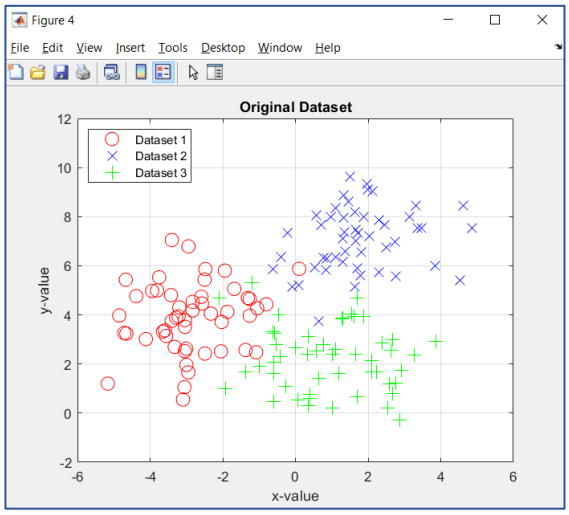
Once the Kmeans algorithm has calculated the clusters with the use of a training dataset, additional datasets can be applied to categorise the points provided with said data. To demonstrate this, a new dataset is generated that has the same gaussian distribution as a training dataset, however only consisting of three clusters of fifty values. Furthermore, the three data matrices are concatenated into a single merged dataset, which is done in the same manner as when the training dataset was generated.

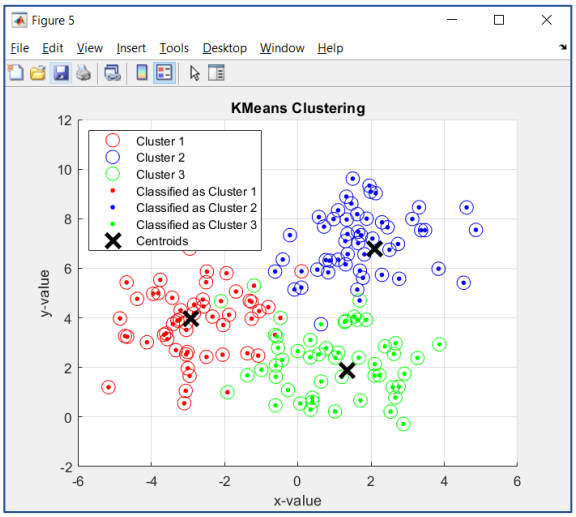
```
% D.) Generate a testing dataset, with the same Gaussian distribution
samples = 50;
Mean1 = [-3; 4];
Std1 = 1.25;
Mean2 = [2; 7];
Std2 = 1.25;
Mean3 = [1; 2];
Std3 = 1.5;

data1 = Std1 * randn(2, samples) + repmat(Mean1, 1, samples);
data2 = Std2 * randn(2, samples) + repmat(Mean2, 1, samples);
data3 = Std3 * randn(2, samples) + repmat(Mean3, 1, samples);

% E.) Concatenate the testing datasets into a single dataset
testData = [data1 data2 data3];
```

```
% F.) Assign new data to the existing clusters
X = testData';
K = size(centroids, 1);
indices = zeros(size(X,1), 1);
m = size(X,1);

% Calculate closest centroid for each datapoint (which cluster each
% datapoint must be assigned to)
for p = 1:m

    chosen_cluster = 1;
    min_dist = (X(p,:) - centroids(1,:)) * (X(p,:) - centroids(1,:))';

    for j = 2:K
        dist = sum((X(p,:) - centroids(j,:)) .^ 2);
        if (dist < min_dist)
            min_dist = dist;
            chosen_cluster = j;
        end
    end

    indices(p) = chosen_cluster;
end
```

To test this new data, the algorithm needed to loop through all datapoints in the provided dataset and calculate the distance between said datapoint and all centroids; the datapoint is then assigned to a specific cluster, based on which centroid is closest. The test results are then plotted on a new figure showing what the datapoints have been categorised as, alongside with its original cluster.

```
% Plot Kmeans clustering implementation with training dataset
figure;
message = sprintf('KMeans Clustering');
hold on;

plot(X(1:50, 1),X(1:50, 2),'ro', 'MarkerSize', 10); % for 1st cluster
plot(X(51:100, 1),X(51:100, 2),'bo', 'MarkerSize', 10); % for 2nd cluster
plot(X(101:150, 1),X(101:150, 2),'go', 'MarkerSize', 10); % for 1st cluster

plot(testData(1, indices == 1),testData(2, indices == 1),'r.', 'MarkerSize', 10); % for 1st cluster
plot(testData(1, indices == 2), testData(2, indices == 2), 'b.', 'MarkerSize', 10); %for 2nd cluster
plot(testData(1, indices == 3), testData(2, indices == 3), 'g.', 'MarkerSize', 10); %for 3rd cluster

plot(centroids(1,1), centroids(1,2),'kx', 'MarkerSize',15,'LineWidth', 3); %for the 1st centroid
plot(centroids(2,1), centroids(2,2),'kx', 'MarkerSize', 15, 'LineWidth', 3); %for the 2nd centroid
plot(centroids(3,1), centroids(3,2),'kx', 'MarkerSize', 15, 'LineWidth', 3); %for the 3rd centroid

legend('Cluster 1', 'Cluster 2', 'Cluster 3', 'Classified as Cluster 1', 'Classified as Cluster 2', ...
    'Classified as Cluster 3', 'Centroids', 'location', 'NW');
title(message);
hold off;
xlabel('x-value');
ylabel('y-value');
xlim([-6 6]);
ylim([-2 12]);
grid on;
```





Conclusion

By analysing the results represented in "**figure 5**", it is clear that some datapoints have been misclassified, with a total of seven datapoints being wrongly assigned to a neighbouring cluster. This is due to its value being significantly different from the centroid calculated for its original cluster. In addition, one major issue of the Kmeans algorithm is that when the data consists of many outliers, the calculated centroids can be inaccurate in comparison to the true centre of the cluster. This becomes particularly problematic when the data from neighbouring clusters have very similar values which therefore causes many overlaps in the different clusters. Furthermore, when the provided data consists of varying sizes or density, the algorithm finds it difficult to categorise such data.

Another issue that Kmeans contains, is that the user has to specify the value K manually, which can become especially troublesome when K is required to be a high value. This is because Kmeans is dependent on its initial values; when K is low, this problem can be mitigated by recalculating the centroids and clusters n number of times. However, as K increases, more advanced versions of Kmeans are required to properly calculate the centroids of each cluster. Nevertheless, the Kmeans algorithm is simple, easy to implement, and can be scaled to categorise large datasets.

References

Guru99. (2020). Unsupervised Machine Learning: What is, Algorithms, Example. Guru99. Viewed 23rd Nov 2020. <https://www.guru99.com/unsupervised-machine-learning.html>.

DataRobot. (2020). Unsupervised Machine Learning. DataRobot. Viewed 23rd Nov 2020. <https://www.datarobot.com/wiki/unsupervised-machine-learning/>.

Sharma, P. (2019). The Most Comprehensive Guide to K-Means Clustering You'll Ever Need. Analytics Vidhya. Viewed 23rd Nov 2020. < https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/>.

Whittaker, C. (2019). 7 Innovative Uses of Clustering Algorithms in the Real World. Datafloq. Viewed 26th Nov 2020. <https://datafloq.com/read/7-innovative-uses-of-clustering-algorithms/6224>.

Guda, Kavitha. (2016). International Journal of Computer Engineering In Research Trends: An Effective algorithm for Spam Filtering and Cluster Formation [online]. pp. 659-666. Viewed 27th Nov 2020. <https://ijcert.org/ems/ijcert_papers/V3I1210.pdf>.

Chougule, P. Thakare, A. Kale, P. Gole, M. Nanekar, P. (2015). International Journal of Computer Science and Information Technologies: Genetic K-means Algorithm for Credit Card Fraud Detection [online]. p.p. 1724-1727. Viewed 1st Dec 2020. <http://ijcsit.com/docs/Volume%206/vol6issue02/ijcsit20150602177.pdf>.

(2014). International Journal of Computer Applications: Fraud Detection in Credit Card by Clustering Approach [online]. p.p. 29-32. Viewed 1st Dec 2020. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.1195&rep=rep1&type=pdf>.

MathWorks. Kmeans. MathWorks. Viewed 3rd Dec 2020. <https://uk.mathworks.com/help/stats/kmeans.html>.

Yildirim, S. (2020). K-Means Clustering — Explained. Towards Data Science. Viewed 3rd Dec 2020. <https://towardsdatascience.com/k-means-clustering-explained-4528df86a120>.

Sustaina. (2018). Steps to calculate centroids in cluster using K-means clustering algorithm. Data Science Central. Viewed 6th Dec 2020. <https://www.datasciencecentral.com/profiles/blogs/steps-to-calculate-centroids-in-cluster-using-k-means-clustering>.

Mirzani, A. (2014). Statistical Analysis of Microarray Data Clustering using NMF, Spectral Clustering, Kmeans, and GMM. Washington, D.C. IEEE Comput. Soc. Press.