KENNESAW STATE
UNIVERSITY

CS 4267
MACHINE LEARNING

PROJECT 1

UNSUPERVISED LEARNING

INSTRUCTOR

Emin Mary Abraham

Franck DIPANDA
001049149

1. ABSTRACT

In this project, I implemented the K-Means clustering algorithm without using MATLAB's built-in kmeans function. The algorithm was applied to the kmtest and iris datasets to study the impact of normalization and initialization. Having previously completed a project involving image segmentation and clustering in MATLAB during the summer as part of a Machine Vision course, I was already familiar with MATLAB's environment and data handling. This helped accelerate implementation and debugging. The final results show that normalization improves convergence and cluster stability, and that the best run aligns closely with the actual Iris species distribution.

2. TEST RESULTS

2.1 Clustering with K-means algorithm for kmtest dataset

The kmtest dataset was clustered using K = 2, 3, 4, and 5 both with and without z-score normalization.

Without normalization, clusters were unevenly distributed due to varying feature scales. After applying normalization, clusters showed clear separation between groups.

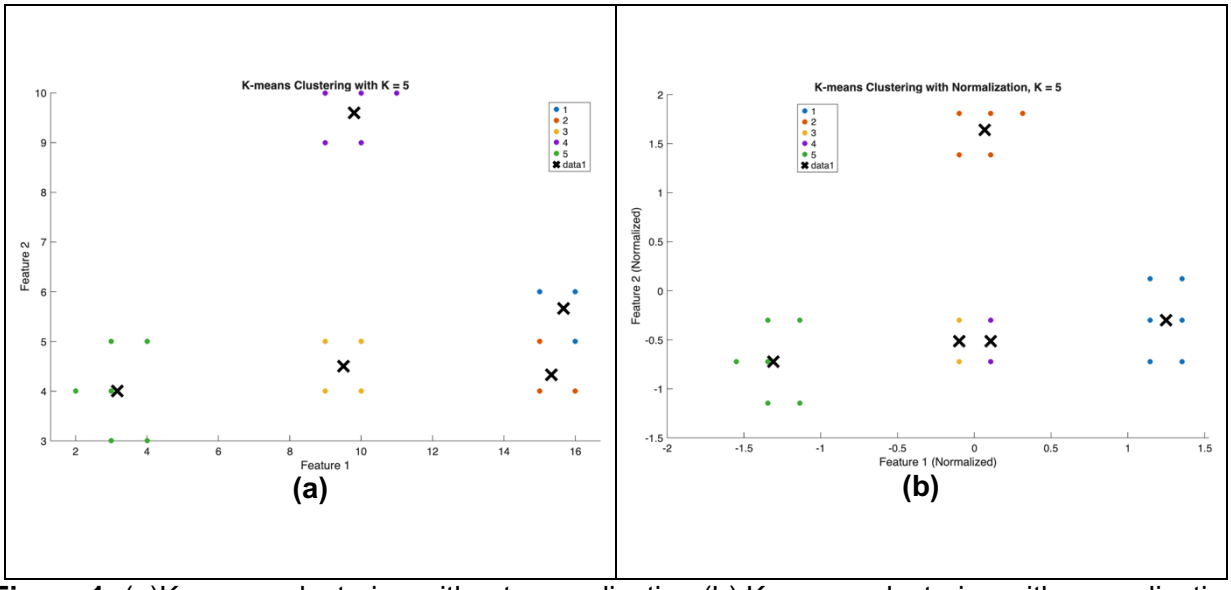

Figure 1: (a)K-means clustering without normalization (b) K-means clustering with normalization

2.2 Test Results for Clustering with K-means algorithm for iris dataset

The iris dataset was clustered using K = 3. The algorithm was executed five times with different random seeds to observe variation in results. The best and worst results were computed from the distortion values, based on total distance from the clusters.

The best run produced well defined clusters for Setosa, Versicolor, and Virginica, whereas the worst run showed overlap between Versicolor and Virginica.

The distance matrix between actual and best cluster centers confirms a close match between the predicted and actual species groups.

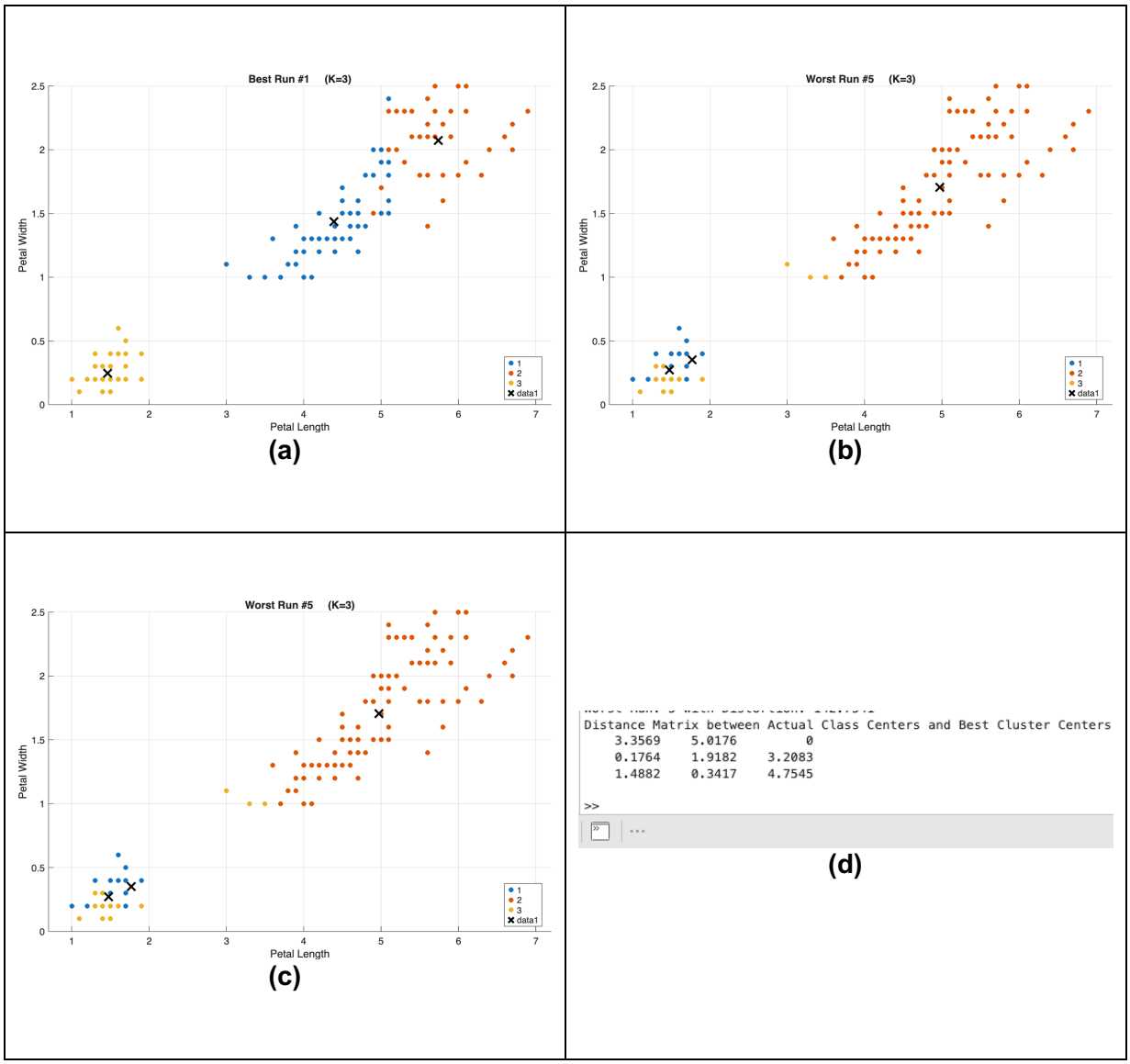

Figure 2: (a) Best run, (b) Worst run, (c) Original Iris classes, (d) Distance between Actual and Predicted Cluster Centers.

3. CODES

3.1 Code for the Main script

```
% Name: Franck Dipanda
% Number: 001047147
% Project 1


clc; clear; close all


% Load the datasets
km_data = readmatrix('kmtest (1).csv');
iris_data = readmatrix('iris .csv');

% Display the first few rows of each dataset to verify loading
disp('kmtest shape ->');
disp(size(km_data));
disp('first rows of kmtest:');
disp(head(km_data));
disp('iris shape ->');
disp(size(iris_data));
disp('first rows of iris:');
disp(head(iris_data));

% Plotting the datasets to ensure they loaded correctly

figure;
scatter(km_data(:, 1), km_data(:, 2), 25, 'filled');
title('kmtest Data');
xlabel('Feature 1');
ylabel('Feature 2');
grid on

figure;
scatter(iris_data(:, 3), iris_data(:, 4), 25, 'filled');
title('Iris Data');
xlabel('Petal Length');
ylabel('Petal Width');
grid on;
```

3.2 Code for K-means algorithm

```
% Name: Franck Dipanda
% Number: 001047147
% Project 1


function [centers, labels] = kMeans(X, k, max_iter)

if nargin < 3
    max_iter = 100; % Set default maximum iterations if not provided
end
```

```
n=size(X,1);
d=size(X,2);

% Randomly select our k points for the initial centroids
randomIndices = randperm(n, k); % Randomly select k unique indices
centers = X(randomIndices, :);   % Initialize centroids using the selected
points

labels = zeros(n, 1); % Initialize labels for each data point
for iter = 1:max_iter
    % Assign labels based on the closest centroid
    distances = pdist2(X, centers); % Compute distances from points to
centroids
    [~, labels] = min(distances, [], 2); % Assign labels based on closest
centroid
    new_centers = zeros(k,d);

% Update centroids by calculating the mean of the assigned points
for j = 1:k
    pts = X(labels == j, :);
    if ~isempty(pts)
        new_centers(j, :) = mean(pts, 1); % Update the centroid to the mean of
assigned points
    else
        new_centers(j, :) = centers(j, :); % Keep the old centroid if no
points are assigned
    end
end

% Check for convergence
if all(abs(new_centers - centers) < 1e-6, 'all')
    disp(['Converged at Iteration ', num2str(iter)])
    break; % Exit the loop if convergence is achieved
end
centers=new_centers;
end
end
```

3.3 Code for K-means implementation on kmtest dataset

```
% Name: Franck Dipanda
% Number: 001047147
% Project 1


% K-means on kmtest dataset (no normalization)
clc; clear; close all

% Load the kmtest dataset
km_data = readmatrix('kmtest (1).csv');
```

```
% Set K values
K_values = [2 3 4 5];
% Perform K-means clustering for each K value and store results

fprintf('\n Running K-means without normalization \n')
for k = K_values
    [idx, centroids] = kMeans(km_data, k);
    fprintf('K = %d: Cluster centers calculated.\n', k);

% Plotting the clusters
figure;
    gscatter(km_data(:, 1), km_data(:, 2), idx);
    hold on;
    plot(centroids(:, 1), centroids(:, 2), 'kx', 'MarkerSize', 15,
'LineWidth', 3);
    title(['K-means Clustering with K = ' num2str(k)]);
    xlabel('Feature 1');
    ylabel('Feature 2');
    hold off;


end

% K-means on kmtest dataset (z-score normalization)
fprintf('\n Running K-means with normalization \n')

% Normalize the data using z-score normalization
normalized_data = zscore(km_data);
for k = K_values
    [centroids, idx] = kMeans(normalized_data, k);
    fprintf('K = %d: Cluster centers calculated.\n', k);

    % Plotting the clusters
    figure;
    gscatter(normalized_data(:, 1), normalized_data(:, 2), idx);
    hold on;
    plot(centroids(:, 1), centroids(:, 2), 'kx', 'MarkerSize', 15,
'LineWidth', 3);
    title(['K-means Clustering with Normalization, K = ' num2str(k)]);
    xlabel('Feature 1 (Normalized)');
    ylabel('Feature 2 (Normalized)');
    hold off;
end
```

3.4 Code for K-means implementation on iris dataset

```
% Name: Franck Dipanda
% Number: 001047147
```

```
% Project 1


% K-means on iris dataset

fprintf('\n Running K-means on Iris dataset (K=3) \n');

% Load the iris dataset
iris_data = readmatrix('iris .csv');
% Extract features for clustering
features = iris_data(:, 1:4);

runs = 5; %run 5 times
k =3;
results = cell(runs, 2); %store both centers and labels

for i = 1:runs
    rng(i);  % randomizes seed for each iteration
    [clusterCenters, clusterLabels] = kMeans(features, k);
    results{i, 1} = clusterCenters;
    results{i, 2} = clusterLabels;
end

% Analyze the results and compute the best and worst clustering runs

X = features;

distortions = zeros(runs,1);

for i = 1:runs
    centers = results{i,1};
    labels = results{i,2};
    d=0;

    for j = 1:k
        cluster_pts = X(labels ==j, :);
        if ~isempty(cluster_pts)
            d = d +sum(sum((cluster_pts - centers(j,:)).^2));
        end
    end

    distortions(i) = d; % store the distortion
end

[bestDistortion, bestRun] = min(distortions); % find the best clustering run
[worstDistortion, worstRun] = max(distortions); % find the worst clustering
run


% Display the results
fprintf('Best Run: %d with Distortion: %.4f\n', bestRun, bestDistortion);
fprintf('Worst Run: %d with Distortion: %.4f\n', worstRun, worstDistortion);

% Plot the distortions for the bes & worst runs
```

```
bestCenters = results{bestRun, 1};
bestLabels = results{bestRun, 2};

worstCenters = results{worstRun,1};
worstLabels = results{worstRun,2};

figure
gscatter(X(:,3), X(:,4), bestLabels)
hold on
plot(bestCenters(:,3), bestCenters(:,4), 'kx', 'MarkerSize',12,'LineWidth',2)
title(sprintf('Best Run #%d     (K=3)', bestRun))
xlabel('Petal Length'); ylabel('Petal Width'); grid on

figure
gscatter(X(:,3), X(:,4), worstLabels)
hold on
plot(worstCenters(:,3), worstCenters(:,4), 'kx',
'MarkerSize',12,'LineWidth',2)
title(sprintf('Worst Run #%d     (K=3)', worstRun))
xlabel('Petal Length'); ylabel('Petal Width'); grid on


% Plot the original label to compare

orig_labels = iris_data(:,5);
figure
gscatter(X(:,3), X(:,4), orig_labels)
title('Original Iris Labels (Attributes 3 & 4)')
xlabel('Petal Length'); ylabel('Petal Width'); grid on


% Finally, finding the distance between best cluster centers and actual
% class centers

actual_labels = iris_data(:,5); %they are disclosed on the 5th column

unique_labels = unique(actual_labels);
num_classes = numel(unique_labels);

% Calculate the actual class centers for comparison
actualCenters = zeros(num_classes, size(X,2));

for i = 1:num_classes
    class_pts = X(actual_labels == unique_labels(i), :);
    actualCenters(i, :) = mean(class_pts, 1); % calculate the mean for each
class
end

distMatrix = pdist2(actualCenters, bestCenters);

% Display the distance matrix for analysis
disp('Distance Matrix between Actual Class Centers and Best Cluster
Centers:');
disp(distMatrix);
```

Acknowledgement:

I would like to thank **Dr Mahmut Karakaya** for providing the template for project report. I also wish to acknowledge the resources I used, which assisted my understanding of clustering concepts and implementation:

“Query regarding k-means clustering in MATLAB”, Stack Overflow, accessed at: https://stackoverflow.com/questions/35359104/query-regarding-k-means-clustering-in-matlab

“K-Means Clustering with Python”, W3Schools, accessed at: https://www.w3schools.com/python/python_ml_k-means.asp