Maral Nourimand
23.11.2023
Data Mining Exercise Week 04

Question #1

```
% Load the dataset from the Excel file
power_data = xlsread('Tetuan City power consumption.csv');

% Select the fifth case
query_case = power_data(5, :);

% Calculate Euclidean distance (excluding the query case)
euclidean_distances = sqrt(sum((power_data - query_case).^2, 2));
% Set distance to query case as Inf to exclude it from finding min
euclidean_distances(5) = Inf;

% Calculate Manhattan distance (excluding the query case)
manhattan_distances = sum(abs(power_data - query_case), 2);
% Set distance to query case as Inf to exclude it from finding min
manhattan_distances(5) = Inf;

% Calculate Cosine distance (excluding the query case)
cosine_distances = 1 - sum(power_data .* repmat(query_case, size(power_data, 1), 1),
2) ...
    ./ (sqrt(sum(power_data.^2, 2)) * sqrt(sum(query_case.^2)));
% Set distance to query case as Inf to exclude it from finding min
cosine_distances(5) = Inf;

% Find the indices of the nearest neighbors
[~, euclidean_nearest_neighbor_index] = min(euclidean_distances);
[~, manhattan_nearest_neighbor_index] = min(manhattan_distances);
[~, cosine_nearest_neighbor_index] = min(cosine_distances);

% Display the results
disp('Euclidean Nearest Neighbor:');
disp(['Index: ' num2str(euclidean_nearest_neighbor_index)]);
disp('Manhattan Nearest Neighbor:');
disp(['Index: ' num2str(manhattan_nearest_neighbor_index)]);
disp('Cosine Nearest Neighbor:');
disp(['Index: ' num2str(cosine_nearest_neighbor_index)]);
```

query_case =

5.921	75.7	0.081	0.048	0.085	27335.7	17872.34	18442.41

Euclidean Nearest Neighbor:

Index: 32719

0.995877533188383 0.808736606119517 0.999994125098691 0.999675403006193 0.999763943925655 -32726.2608169623 -13919.3998340420 -12174.1249693616

Manhattan Nearest Neighbor:

Index: 32719

Cosine Nearest Neighbor:

Index: 1011

14.2700000000000 77.7000000000000 0.0870000000000000 0.0330000000000000 0.122000000000000 26831.3924100000 17536.7781200000 18060.7228900000

Question #2

```
% Normalize the variables
normalized_power_data = zscore(power_data);

% Select the fifth case from the normalized data
query_case_normalized = normalized_power_data(5, :);

% Calculate Euclidean distance (excluding the query case)
euclidean_distances_normalized = sqrt(sum((normalized_power_data -
query_case_normalized).^2, 2));
euclidean_distances_normalized(5) = Inf;


% Calculate Manhattan distance (excluding the query case)
manhattan_distances_normalized = sum(abs(normalized_power_data -
query_case_normalized), 2);
manhattan_distances_normalized(5) = Inf;


% Calculate Cosine distance (excluding the query case)
cosine_distances_normalized = 1 - sum(normalized_power_data .*
repmat(query_case_normalized, size(normalized_power_data, 1), 1), 2) ...
    ./ (sqrt(sum(normalized_power_data.^2, 2)) *
sqrt(sum(query_case_normalized.^2)));
cosine_distances_normalized(5) = Inf;

% Find the indices of the nearest neighbors
[~, euclidean_nearest_neighbor_index_normalized] =
min(euclidean_distances_normalized)
[~, manhattan_nearest_neighbor_index_normalized] =
min(manhattan_distances_normalized)
```

```
[~, cosine_nearest_neighbor_index_normalized] = min(cosine_distances_normalized)
```

```
euclidean_nearest_neighbor_index_normalized =

     6


manhattan_nearest_neighbor_index_normalized =

     6


cosine_nearest_neighbor_index_normalized =

     6
```

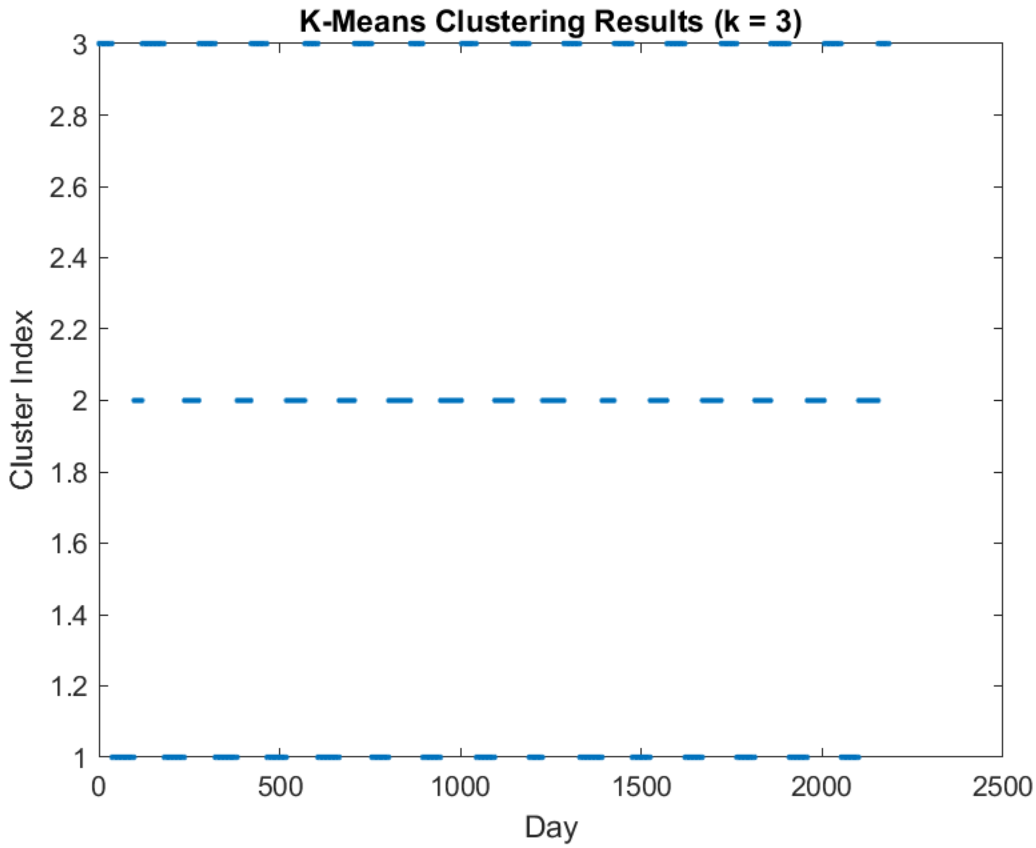
Question #3

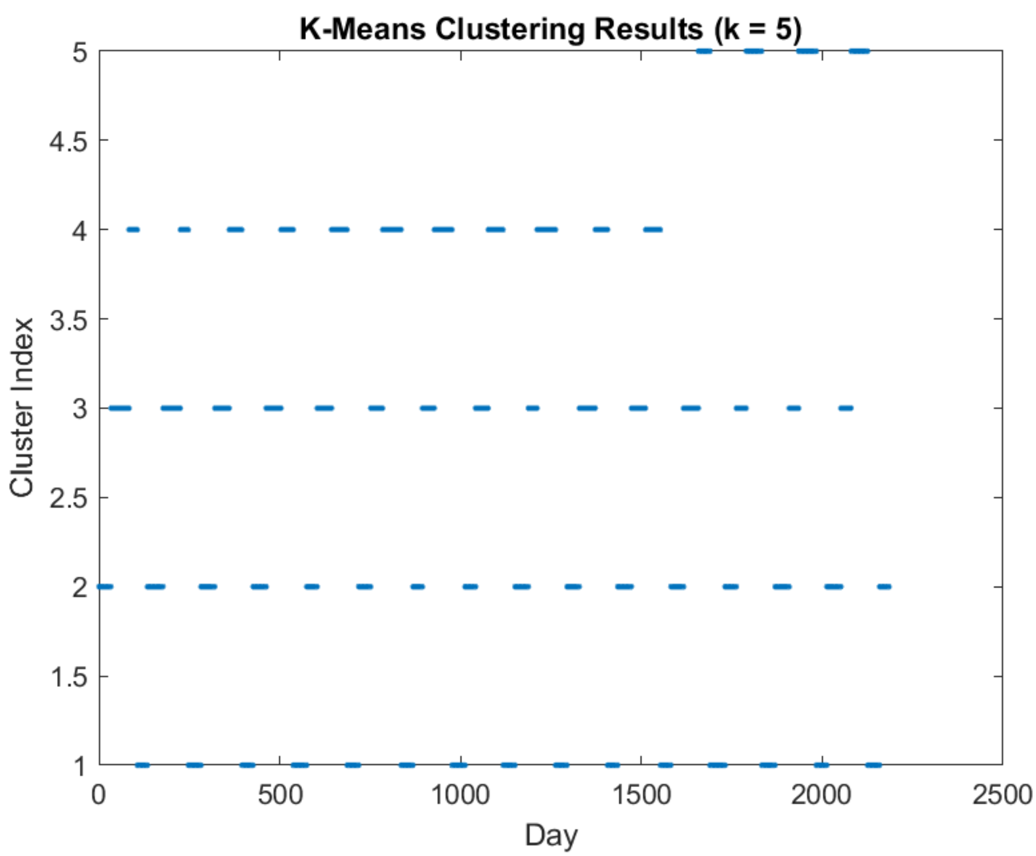
```
% Scale the variables to the interval [0, 1]
scaled_power_data = (power_data - min(power_data)) ./ (max(power_data) -
min(power_data));

% Select the fifth case from the scaled data
query_case_scaled = scaled_power_data(5, :);

% Calculate Euclidean distance (excluding the query case)
euclidean_distances_scaled = sqrt(sum((scaled_power_data - query_case_scaled).^2,
2));
euclidean_distances_scaled(5) = Inf;

% Calculate Manhattan distance (excluding the query case)
manhattan_distances_scaled = sum(abs(scaled_power_data - query_case_scaled), 2);
manhattan_distances_scaled(5) = Inf;

% Calculate Cosine distance (excluding the query case)
cosine_distances_scaled = 1 - sum(scaled_power_data .* repmat(query_case_scaled,
size(scaled_power_data, 1), 1), 2) ...
    ./ (sqrt(sum(scaled_power_data.^2, 2)) * sqrt(sum(query_case_scaled.^2)));
cosine_distances_scaled(5) = Inf;

% Find the indices of the nearest neighbors
[~, euclidean_nearest_neighbor_index_scaled] = min(euclidean_distances_scaled)
[~, manhattan_nearest_neighbor_index_scaled] = min(manhattan_distances_scaled)
[~, cosine_nearest_neighbor_index_scaled] = min(cosine_distances_scaled)
```

```
euclidean_nearest_neighbor_index_scaled =

   6

manhattan_nearest_neighbor_index_scaled =

   6

cosine_nearest_neighbor_index_scaled =

     3460
```

Question #4

```
% Extract the temperature data (temperature is the first column)
temperature_data = power_data(:, 1);

% Reshape the temperature data to have daily measurements
daily_temperature_data = reshape(temperature_data, [], 24);

% Choose the number of clusters (k values)
k_values = [3, 5];

% Perform k-means clustering for each k value
for k = k_values
    % Perform k-means clustering
    [idx, centers] = kmeans(daily_temperature_data, k);

    % Display the results
    % disp(['Results for k = ' num2str(k)]);
    % disp('Cluster Centers:');
    % disp(centers);

    % Visualize the cluster assignments
    figure;
    plot(idx, '.');
    title(['K-Means Clustering Results (k = ' num2str(k) ')']);
    xlabel('Day');
    ylabel('Cluster Index');
end
```


K-Means Clustering Results (k = 3)
Cluster Index
Day
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
500
1000
1500
2000
2500


K-Means Clustering Results (k = 5)
Cluster Index
Day
5
4.5
4
3.5
3
2.5
2
1.5
1
0
500
1000
1500
2000
2500

Question #5

```
% Load the Iris dataset
iris_data = dlmread('Iris.txt');

% Extract relevant data
running_number = iris_data(:, 1);
% Exclude the running number and class columns
features = iris_data(:, 2:end-1);
classes = iris_data(:, end);

% Separate training and test data
training_data = [];
training_labels = [];
test_data = [];
test_labels = [];

for i = 1:3
    class_data = features(classes == i, :);

    % Training data: First 40 cases
    training_data = [training_data; class_data(1:40, :)];
    training_labels = [training_labels; repmat(i, 40, 1)];

    % Test data: Remaining 10 cases
    test_data = [test_data; class_data(41:end, :)];
    test_labels = [test_labels; repmat(i, 10, 1)];
end

% Perform k-means clustering on training data
num_clusters = 3;
[idx, cluster_centers] = kmeans(training_data, num_clusters);

% Classify using Euclidean distance and nearest neighbor for k-means
predicted_labels_kmeans = zeros(size(test_data, 1), 1);

for i = 1:size(test_data, 1)  % 30 test cases
    distances = sqrt(sum((cluster_centers - test_data(i, :)).^2, 2));
    [~, min_idx] = min(distances);
    predicted_labels_kmeans(i) = min_idx;
end

% Calculate accuracy for k-means
accuracy_kmeans = sum(predicted_labels_kmeans == test_labels) / length(test_labels) *
100
```

accuracy_kmeans =

40.3333%

Question #6

```
% Identify highly correlated variables
correlation_matrix = corr(features);
[max_corr, idx] = max(correlation_matrix(:));
[row, col] = ind2sub(size(correlation_matrix), idx);

% Replace highly correlated variables with their mean
reduced_features = features;
reduced_features(:, col) = mean(features(:, [col, row]), 2);
reduced_features(:, row) = [];  % Remove the column with higher correlation

% Separate training and test data for reduced variables
training_data = [];
training_labels = [];
test_data = [];
test_labels = [];

for i = 1:3
    class_data = reduced_features(classes == i, :);

    % Training data: First 40 cases
    training_data = [training_data; class_data(1:40, :)];
    training_labels = [training_labels; repmat(i, 40, 1)];

    % Test data: Remaining 10 cases
    test_data = [test_data; class_data(41:end, :)];
    test_labels = [test_labels; repmat(i, 10, 1)];
end

% Perform k-means clustering on the training data
num_clusters = 3;
[idx, cluster_centers] = kmeans(training_data, num_clusters);

% Classify the test data using k-means clusters and calculate accuracy
predicted_labels_kmeans = zeros(size(test_data, 1), 1);

for i = 1:size(test_data, 1)
    distances = sqrt(sum((cluster_centers - test_data(i, :)).^2, 2));
    [~, min_idx] = min(distances);
    predicted_labels_kmeans(i) = min_idx;
end

% Calculate accuracy for k-means
accuracy_kmeans_reduced = sum(predicted_labels_kmeans == test_labels) /
length(test_labels) * 100
```

accuracy_kmeans_reduced =

36.6667%

Question #7

```
% Apply Principal Component Analysis (PCA)
[coeff, score, latent] = pca(features);

% Determine the number of components to retain (e.g., retain 2 components)
num_components = 2;
reduced_features = score(:, 1:num_components);

% Separate training and test data
training_data = [];
training_labels = [];
test_data = [];
test_labels = [];

for i = 1:3
    class_data = reduced_features(classes == i, :);

    % Training data: First 40 cases
    training_data = [training_data; class_data(1:40, :)];
    training_labels = [training_labels; repmat(i, 40, 1)];

    % Test data: Remaining 10 cases
    test_data = [test_data; class_data(41:end, :)];
    test_labels = [test_labels; repmat(i, 10, 1)];
end

% Perform k-means clustering on the training data
num_clusters = 3;
[idx, cluster_centers] = kmeans(training_data, num_clusters);

% Classify the test data using k-means clusters and calculate accuracy
predicted_labels_kmeans = zeros(size(test_data, 1), 1);

for i = 1:size(test_data, 1)
    distances = sqrt(sum((cluster_centers - test_data(i, :)).^2, 2));
    [~, min_idx] = min(distances);
    predicted_labels_kmeans(i) = min_idx;
end

% Calculate accuracy for k-means
accuracy_kmeans_PCA = sum(predicted_labels_kmeans == test_labels) /
length(test_labels) * 100
```

accuracy_kmeans_PCA =

23.3333%

For Questions #5, #6 and #7 I used function kmeans(). Later, I understood that in every round of run the accuracy for classification differs and The differences in accuracy between runs could be due to the random initialization of the k-means algorithm. The k-means algorithm starts with random initial cluster centers, and the final results can vary depending on the initial random seed.
So, I added `rng` function to set the random seed before running k-means.
After that the results of accuracy were carved on stone as following. It is a bit strange but I couldn't find any explanation unfortunately:

accuracy_kmeans =

0%

accuracy_kmeans_reduced =

96.6667%

accuracy_kmeans_PCA =

90 %