Linear Algebra

Vector Part 3: Vector Applications

Automotive Intelligence Lab.

Contents

- Correlation and cosine similarity
- Time series filtering and feature detection
- k-means clustering
- Summary
- Exercise

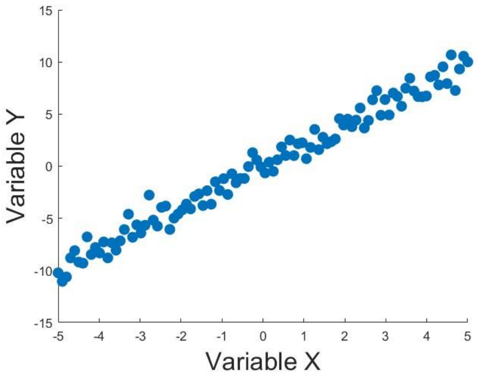
Correlation and Cosine Similarity

Correlation Coefficient

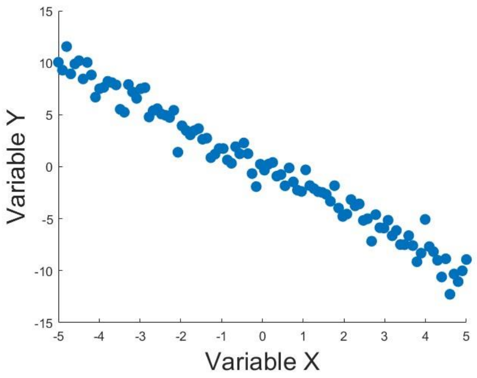
- **A Single number that quantifies the linear relationship between two variables.**

- **Correlation coefficients range is from -1 to $+1$.**
  - -1 is indicating a perfect negative relationship.
  - $+1$ is indicating a perfect positive relationship.
  - 0 is indicating $\boxed{\text{no}}$ relationship.

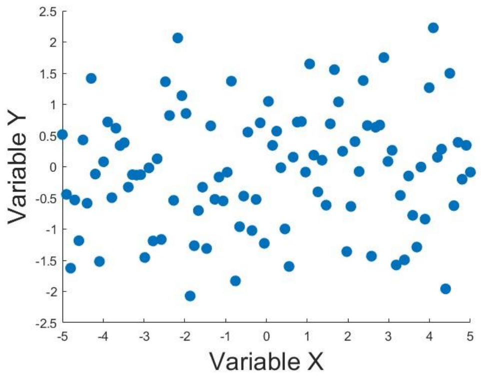
- **Nonlinear relationships can exist even if their correlation is zero.**



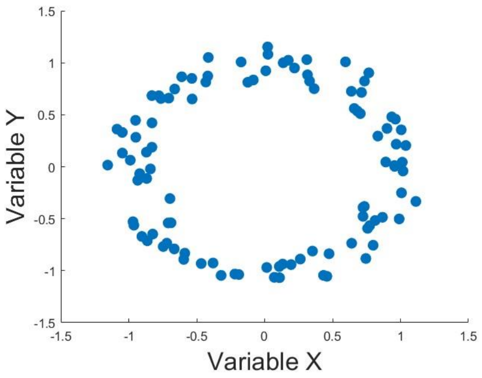
Positive correlation



Negative correlation

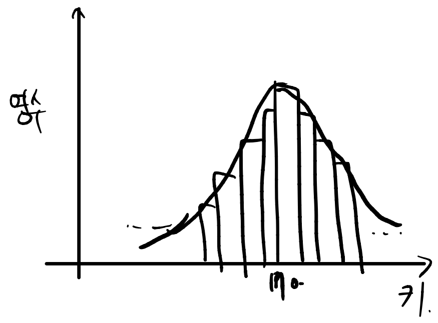


Zero correlation



Zero correlation
(nonlinear relationships)

Formula of Pearson Correlation Coefficient



- **To make the correlation coefficient fall within the range of -1 to $+1$, normalizations are required.**
  - Mean center each variable
    - Mean centering means to subtract the average value from each data value.
  - Divide the dot product by the product of the vector norms.
    - This divisive normalization cancels the measurement units and scales the maximum possible correlation magnitude to $|1|$.
    - In Eq 1. $\bar{x}$ is the mean value of x
    - In Eq 2. $\tilde{x}$ is the mean-centered version of x.
    - In Eq 2. , if the variables are unit normed such that $\|x\| = \|y\| = 1$ $\left(\|x\| = \sqrt{x^T x}\right)$, then their correlation equals their dot product.
    - Eq 2. is a simplification under the assumption that the variables have already been mean centered.

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Eq 1. Formula for Pearson correlation coefficient

$$\rho = \frac{\tilde{x}^T \tilde{y}}{\|\tilde{x}\|\,\|\tilde{y}\|}$$

내적.

Eq 2. Pearson correlation expressed in the parlance of linear algebra

Cosine Similarity

Correlation is not only way to assess similarity between two variables.

- ▶ Cosine similarity is another method.
  - Cosine similarity

$$\cos(\theta_{x,y}) = \frac{\alpha = x^T y}{||x||\ ||y||}$$

Eq 1. Formula for cosine similarity

- α is the dot product between x and y.

- Cosine similarity utilizes the cosine of the angle between two vectors in the dot product space to measure the degree of similarity between the vectors.
  - If angle is 0°, cosine value is 1 ➔ Completely identical vectors.
  - If angle is 180°, cosine value is -1. ➔ Completely opposite vectors.
  - If angle is in range of $0^\circ - 180^\circ$, cosine value is less than 1.

- Cosine similarity range : $\boxed{[-1, +1]}$

Difference between Correlation and Cosine Similarity

- **Pearson correlation and cosine similarity represent the linear relationship between two variables.**
  - They are based on the dot product which is a linear operation.

- **Pearson correlation and cosine similarity can give different results for the same data.**
  - They start from different assumptions.
  - For the variables $[0, 1, 2, 3]$ and $[100,\ 101,\ 102,\ 103]$
    - Pearson correlated : $\boxed{1}$
      - Changes in one variable are exactly mirrored in the other variable.
      - It doesn't matter that one variable has lager numerical values.
    - Cosine similarity : $\boxed{0.808}$
      - They are not same numerical scale, so they are not perfectly related.
  - Neither measure is incorrect nor better than the other.
    - Different statistical methods make different assumptions about data .
    - Those assumptions have implications for the results and for proper interpretation.

Time Series Filtering and Feature Detection

Time Series Filtering

Feature Detection Method by using dot product

- Mechanism of filtering
  - Compute the dot product between the kernel and the time series signal.
- Filtering usually requires local feature detection.
  - Kernel is typically much shorter than the entire time series.
    - Computing the dot product between the kernel and a short snippet of the data of the same length as the kernel is required.
    - This procedure produces one time point in the filtered signal, and then the kernel is moved one time step to the right to compute the dot product with a different (overlapping) signal segment.
    - This procedure is called convolution.

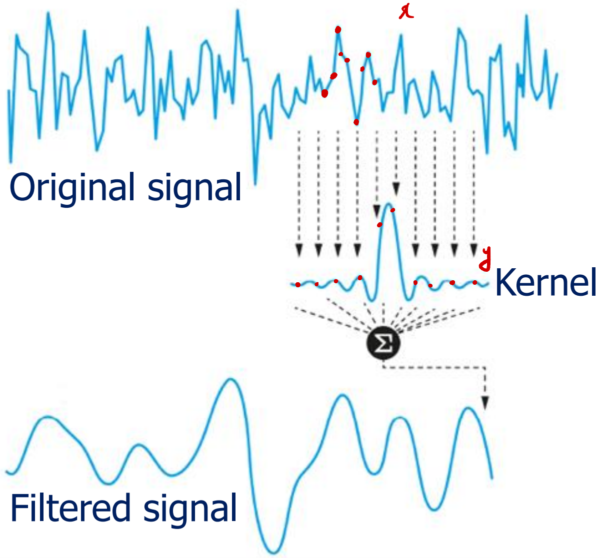
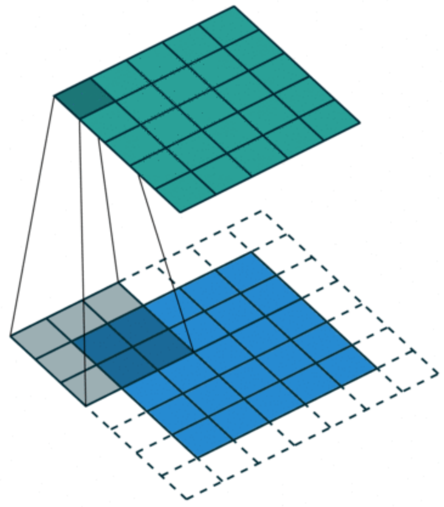


Illustration of time series filtering

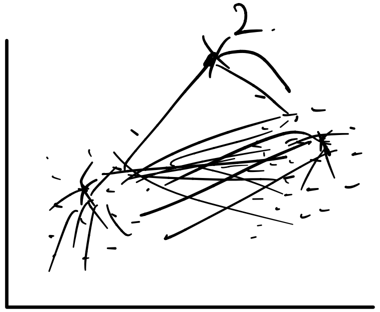


convolution neural network (CNN)

k-means clustering

k-Means Clustering Algorithm

- **An unsupervised method of classifying data into a small number of groups or categories.**
- **Based on minimizing distance to the group center.**
- **An important analysis method in machine learning.**
- 정해줘야 함.
  k–means clustering algorithm.
  1. Initialize k centroids as random points in the data space. (each centroid is a class or category.)
  2. Compute the Euclidean distance between each observation and centroid.
  3. Assign each data observation to the group with the closest centroid.
  4. Update each centroid as the average of all data observations assigned to that centroid.
  5. Repeat steps $2.-4.$ until a convergence criteria is satisfied, or for N iterations.



Step 1: Initialize k Centroids in the Data Space

- k is a parameter of k-means clustering.
  - Here, fix $k = 3$.
- 중점.
- Randomly select k data samples to be centroids.
- The data are contained 150 observations and 2 features.

Code Exercise of k-Means Clustering (1)

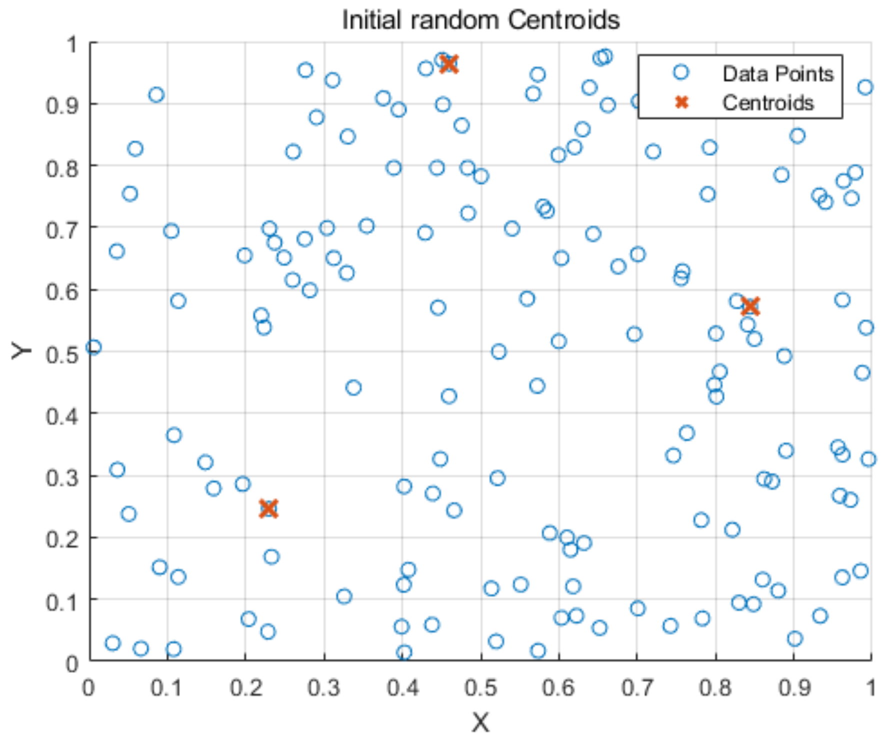
Initialize k centroids as random points in the data space.

- Code Exercise (04_01)

Today's Code Exercise is sequential process, so do not clear the workspace !

```
% Clear previous data and figure
clc; clear; close all;
% Generate 150 vectors each with 2 random elements
data = rand(150, 2); % Each row is a 2D vector, totaling 150 vectors

% Declare the variable k
k = 3;

% Extract k initial centroids from data
ridx = randperm(size(data, 1), k); % Randomly select k unique indices
centroids = data(ridx, :); % Select the rows (vectors) at these indices to be centroids

% Visualization
figure; % Create a new figure window
hold on; % Hold on to draw multiple graphic objects on the same axes

% Plot the data points
scatter(data(:,1), data(:,2), 'o'); % Use 'o' marker to plot data points

% Plot the centroids
scatter(centroids(:,1), centroids(:,2), 100, 'x', 'LineWidth', 2);

title('Initial random Centroids');
xlabel('X');
ylabel('Y');
legend('Data Points', 'Centroids', 'Location', 'best');
grid on; % Turn on the grid
hold off; % Finish drawing
```

Source code



Source code result

Step 2: Compute the Euclidean Distance

- **For one data observation and centroid, Euclidean distance is computed as Eq 1..**
- **An example of how linear algebra often looks different in equations compared to in code.**
  - Think about why the $\boxed{\text{Square root}}$ in Euclidean distance is missing from the code.

$$\boldsymbol{\delta_{i,j}} = \sqrt{(\boldsymbol{d_i^x} - \boldsymbol{c_j^x})^2 + (\boldsymbol{d_i^y} - \boldsymbol{c_j^y})^2}$$

Eq 1. Euclidean distance between one data observation and centroid

$\delta_{i,j}$: The distance from data observation i to centroid j
d_i^x : The feature x of the i th data observation
c_j^x : The x-axis coordinate of centroid j

```
% Calculate the distances
for ci = 1:k
    dists(:, ci) = sum((data - centroids(ci, :)).^2, 2);
    % .^2 is element-wise square function
end
```

Source code

Step 3: Assign Data to the Group with Minimum Distance

- **Code implemented in Matlab.**
- **Return to the inconsistency between the formula and its code.**
  - Distance and squared distance are monotonically related, so both give the same answer.
  - Adding the square root operation increases code Complexity and Computation. time.

```
% Find the minimum distance and the corresponding centroid for each data point
[minDists, assignment] = min(dists, [], 2);
```

Source code

Code Exercise of k-Means Clustering (2)

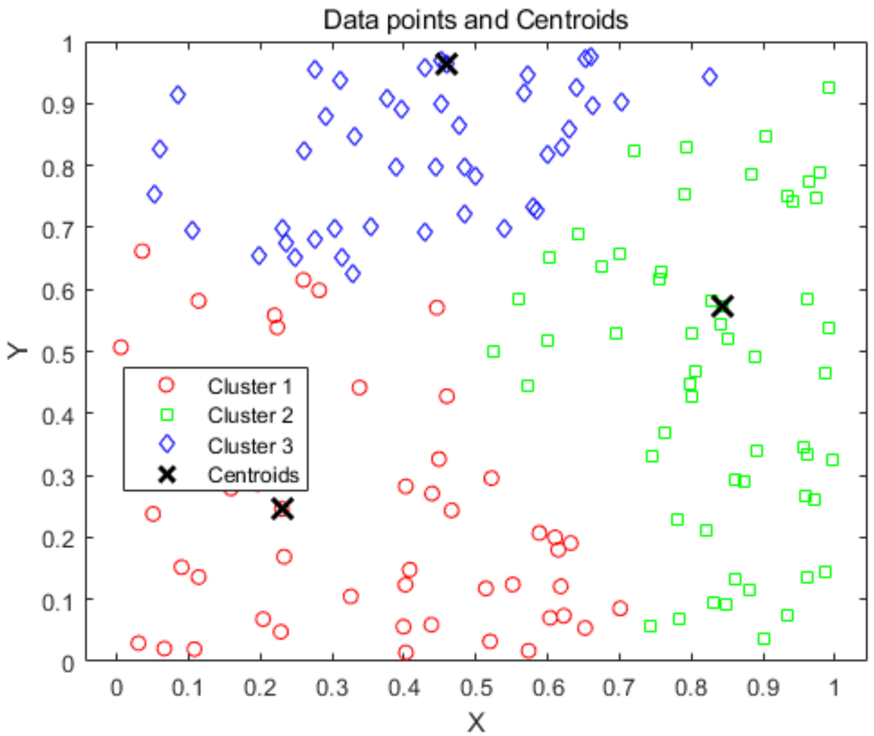
■ Compute the distance and assign to minimum distance group.

▶ Code Exercise (04_02)

Today's Code Exercise is sequential process, so do not clear the workspace !

```
%% You have to run "AILAB_LA_Exercise_04_01_dist.m" before !
dists = zeros(size(data, 1), k);

% Calculate the squared distances
for ci = 1:k
    % Compute the squared Euclidean distance from each point to each centroid
    % and store the results in the 'dists' matrix
    dists(:, ci) = sum((data - centroids(ci, :)).^2, 2);
end

% Find the minimum distance and the corresponding centroid for each data point
[minDists, assignment] = min(dists, [], 2);

% Create a new figure for visualization
figure;

% Plot each data point, colored by the index of its closest centroid
gscatter(data(:,1), data(:,2), assignment, 'rgb', 'osd');
hold on;

% Plot centroids
plot(centroids(:,1), centroids(:,2), 'kx', 'MarkerSize', 12, 'LineWidth', 2);

% Add title and labels
title('Data points and Centroids');
xlabel('X');
ylabel('Y');
legend([arrayfun(@(x) ['Cluster ' num2str(x)], unique(assignment), 'UniformOutput', false);
'Centroids']);
hold off;
```

Source code



Source code result

Step 4: Recompute the Centroids

- Loop over the k clusters, find all data points assigned to each cluster.
- The [ ] of all data points within the group are new centroids.

```
% Recompute centroids
newCentroids = zeros(size(centroids));
for ci = 1:k
    % Calculate the mean of all points assigned to centroid ci
    newCentroids(ci, :) = mean(data(assignment == ci, :), 1);
end
```

Source code

Code Exercise of k-Means Clustering (3)

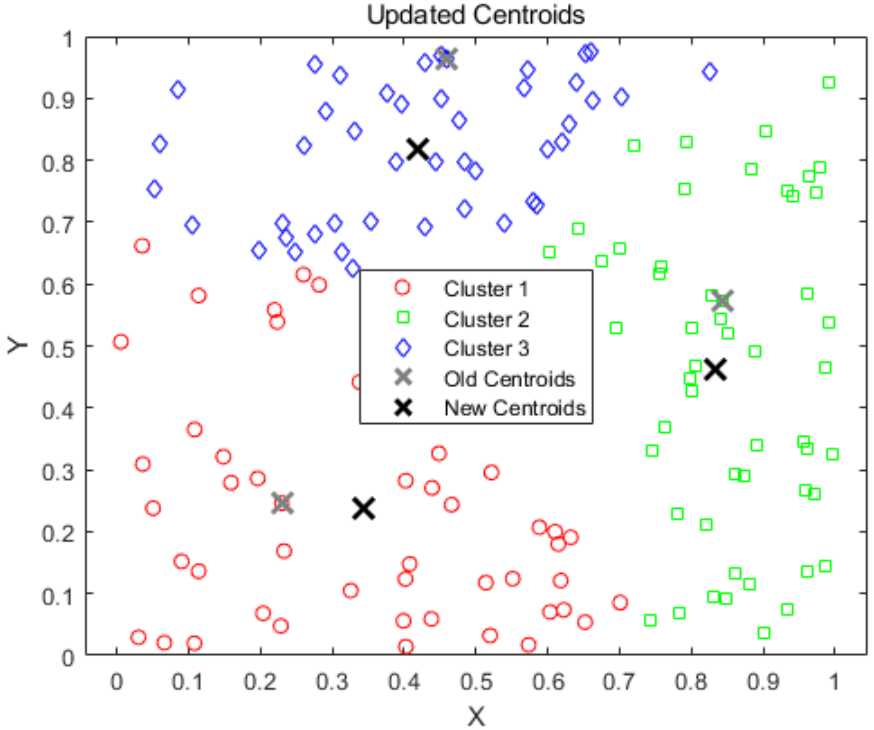
Recompute the centroids as the mean of all data points within the group.

▶ Code Exercise (04_03)

Today's Code Exercise is sequential process, so do not clear the workspace !

```
%% You have to run "AILAB_LA_Exercise_04_02_dist.m" before !

% Recalculate centroids
newCentroids = zeros(size(centroids));
for ci = 1:k
    % Calculate the mean of all points assigned to centroid ci
    newCentroids(ci, :) = mean(data(assignment == ci, :), 1);
end

% Create a new figure for updated visualization
figure;

% Plot each data point, colored by the index of its closest centroid
gscatter(data(:,1), data(:,2), assignment, 'rgb', 'osd');
hold on;

% Plot old centroids with transparent (faded) x marks
plot(centroids(:,1), centroids(:,2), 'x', 'MarkerSize', 12, 'LineWidth', 2, 'Color', [0.5 0.5 0.5
0.5]);

% Plot new centroids
plot(newCentroids(:,1), newCentroids(:,2), 'kx', 'MarkerSize', 12, 'LineWidth', 2);

% Add title and labels
title('Updated Centroids');
xlabel('X');
ylabel('Y');
legend([arrayfun(@(x) ['Cluster ' num2str(x)], unique(assignment), 'UniformOutput', false); 'Old
Centroids'; 'New Centroids']);
hold off;
```

Source code

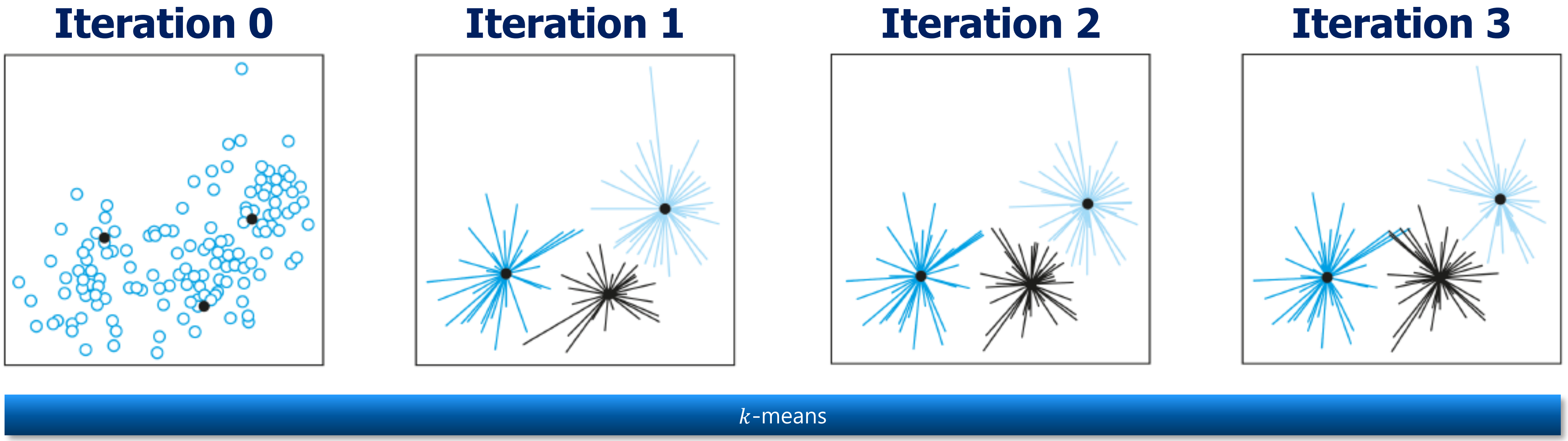


Source code result

Step 5: Put the Previous Steps into a Loop

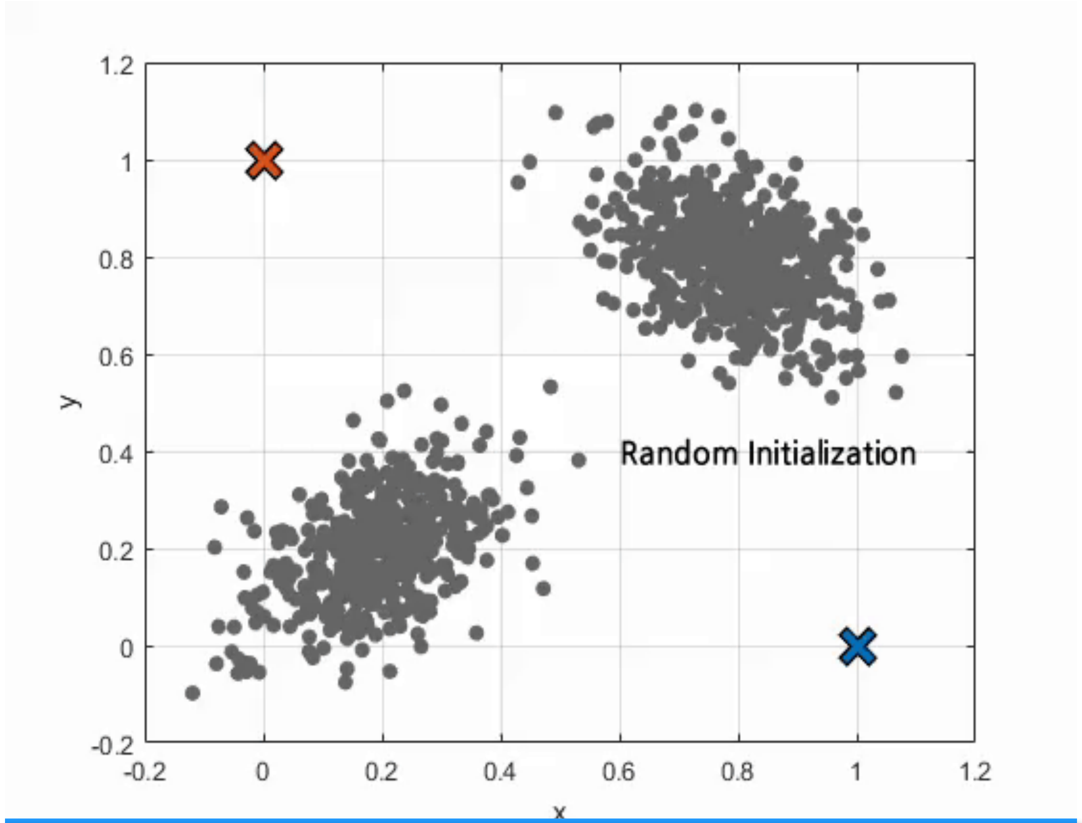
- **The iterations continue until a stopping criteria is reached.**
  - E.g., that the cluster centroids are no longer moving around.

- **Example of k-means clustering.**
  - The four panels show the initial random cluster centroids. (Iteration 0)
  - Their locations are updated after each of three iterations.



Visual Materials of k-Means Clustering

- **Process of k-means clustering.**
  - https://angeloyeo.github.io/2021/02/07/k_means.html



Process of performing k-means algorithm

Exercise

Correlation Exercises 1

- **Write a MATLAB function code**
  - input : takes two vectors
  - output : two number
    - the Pearson correlation coefficient and the cosine similarity value.
- **Write code that follows the formulas presented in this chapter.**

Correlation Exercises 2

1. **Create a variable containing the integers 0 - 3.**
   - *var1* = [0,1,2,3]
2. **Create** *for loop* **that plus offset in** *var1.*
   - *offset* = range (from -50 to 50)
   - *var2 = var1 + offset* list에 저장
3. **Save Pearson correlation Cosine similarity between** *var1* **and** *var2* **in** *for loop.*
4. **Create line plot showing how the correlation and cosine similarity are affected by the mean offset.**
   - Hint. The result must follow Figure 1.

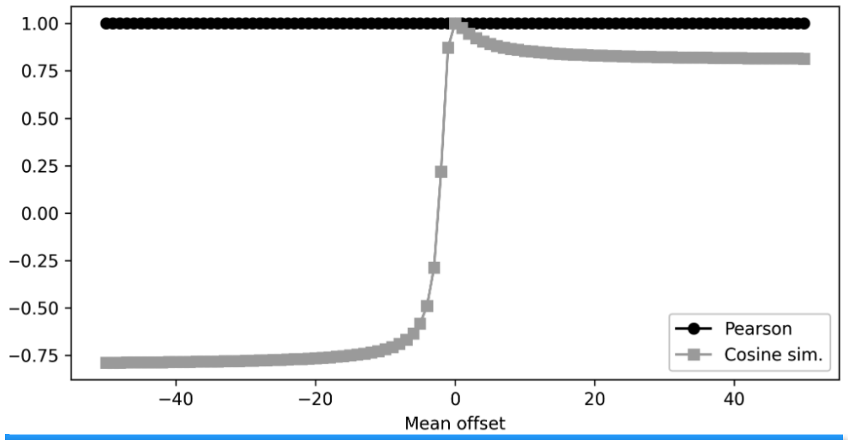


Figure 1. Results of exercise

Filtering and Feature Detection Exercises 1

Build an edge detector.

- the kernel for an edge detector : [-1 +1].
- Write code that creates these two time series (Fig A & B).
  - The signal we'll work with is a plateau function.
  - show the kernel(Fig A.) and the signal(Fig B.).
- Write a for loop over the time points in the signal. At each time point, compute the dot product between the kernel and a segment of the time series data that has the same length as the kernel. You should produce a plot that looks like graph C in Figure 1.
- Hint: The dot product of that kernel with a snippet of a time series signal wit constant value (e.g., [10 10]) is 0. But that dot product is large when the signal has a steep change (e.g., [1 10] would be produce a dot product of 9.)

Fro

kernel [-1; 1.5]

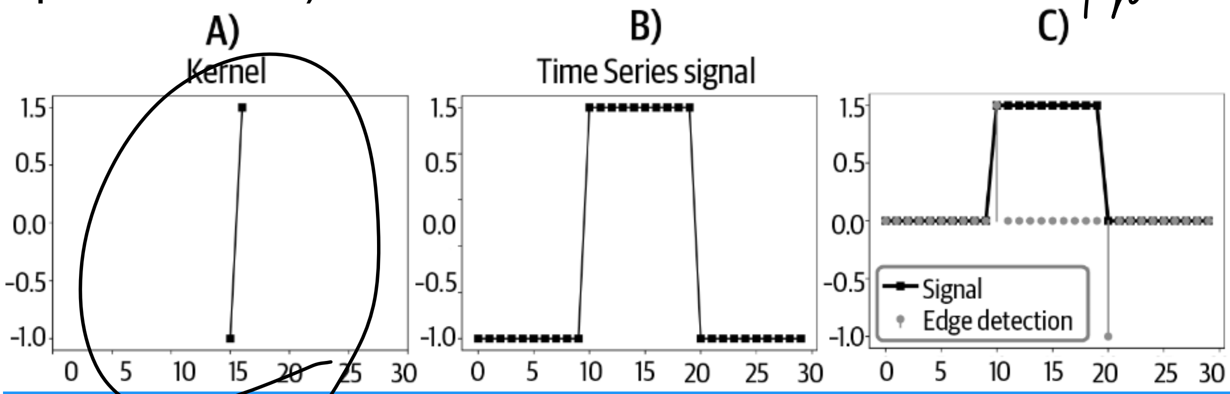
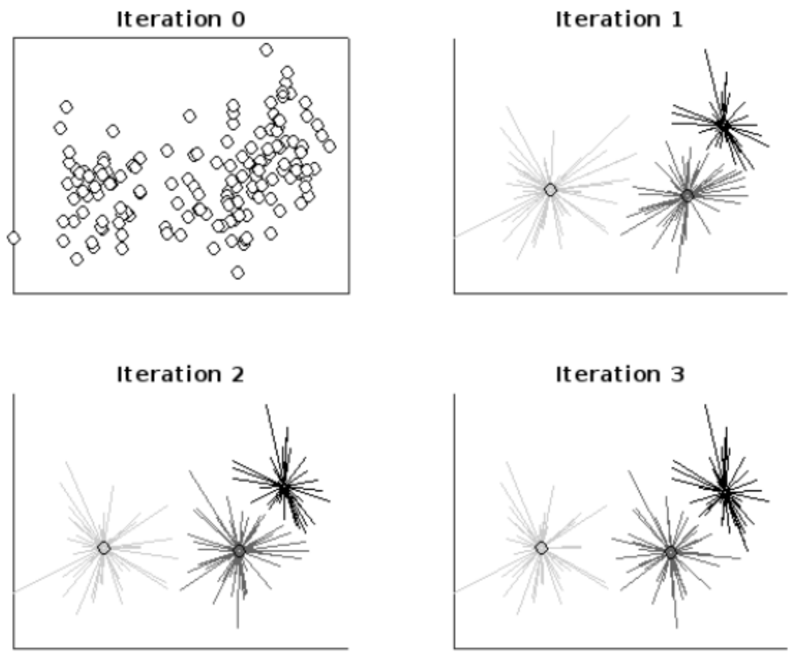


Figure 1. Results of exercise 5

k-means exercises 1

- Without generating new data, rerun the k-means code several times using $k = 3$ to see whether the resulting clusters are similar. Do the final cluster assignments generally seem similar even though the centroids are randomly selected?
  - Source code is attached in next page.



Result of clustering (point distribution can be different)

k-means exercises 1

```matlab
nPerClust = 50;

% Blur around centroid (std units)
blur = 1;

% XY centroid locations
A = [  1, 1 ];
B = [ -3, 1 ];
C = [  3, 3 ];

% Generate data
a = [ A(1)+randn(nPerClust,1)*blur , A(2)+randn(nPerClust,1)*blur ];
b = [ B(1)+randn(nPerClust,1)*blur , B(2)+randn(nPerClust,1)*blur ];
c = [ C(1)+randn(nPerClust,1)*blur , C(2)+randn(nPerClust,1)*blur ];

% Concatenate into a matrix
data = [a; b; c];

% Plot data
figure;
plot(data(:,1), data(:,2), 'ko', 'MarkerFaceColor', 'w');
title('Raw (preclustered) data');
xticks([]);
yticks([]);

% Number of clusters
k = 3;

% Randomly select cluster centers from the data
ridx = randperm(size(data, 1), k);
centroids = data(ridx, :);

% Setup the figure
figure;
lineColors = [0, 0, 0; .4, .4, .4; .8, .8, .8]; % Different shades of gray for each

% Plot data with initial random cluster centroids
subplot(2, 2, 1);
plot(data(:, 1), data(:, 2), 'ko', 'MarkerFaceColor', 'w'); hold on;
plot(centroids(:, 1), centroids(:, 2), 'ko'); hold off;
title('Iteration 0');
set(gca, 'XTick', [], 'YTick', []);

% Loop over iterations
for iteri = 1:3
    % fill here (start)
    % Step 1: Compute distances from each point to each centroid

    % Step 2: Assign to group based on minimum distance

    % Step 3: Recompute centroids
    % fill here (end)
    % Plotting
    subplot(2, 2, iteri+1);
    hold on;
    for i = 1:length(data)
        plot([data(i, 1), centroids(groupidx(i), 1)], [data(i, 2), 
centroids(groupidx(i), 2)], 'Color', lineColors(groupidx(i), :));
    end
    plot(centroids(:, 1), centroids(:, 2), 'ko');
    hold off;
    title(sprintf('Iteration %d', iteri));
    set(gca, 'XTick', [], 'YTick', []);
end

% Save the figure
saveas(gcf, 'Figure_03_03.png');
```

Source code

THANK YOU
FOR YOUR ATTENTION

```
% Scalars
l1 = 1;
l2 = 2;
l3 = -3;

% Vectors
v1 = [4, 5, 1];
v2 = [-4, 0, -4];
v3 = [1, 3, 2];

% Linear weighted combination
linCombo1 = l1 * v1 + l2 * v2 + l3 * v3;
disp(linCombo1);
```

Original code

```
% Scalars
l1 = 1;
l2 = 2;
l3 = -3;

% Vectors
v1 = [4, 5, 1];
v2 = [-4, 0, -4];
v3 = [1, 3, 2];

% Scalars and vectors organized into arrays
scalars =
vectors =

% Initialize the linear combination
linCombo2 =

% Implement linear weighted combination using a loop
for i = 1:length(scalars)
    linCombo2 =
end

% Confirm it's the same answer as above
disp(linCombo2);
```

Generated code