K-Means Clustering

Group 3

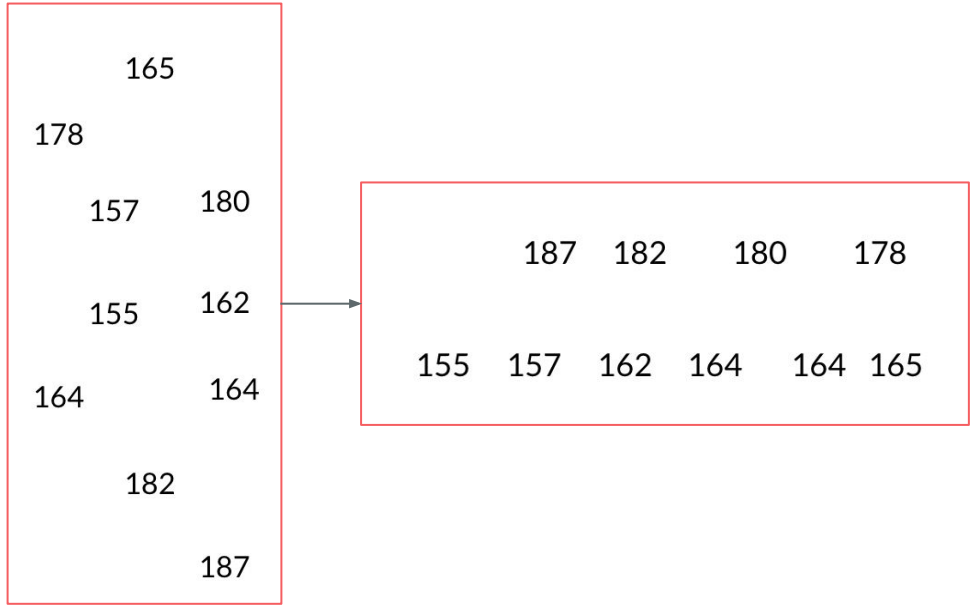
Outline

- Introduction
- Algorithm - Theoretical Approach
- Implementation
- Applications
- Specific algorithm for image compression
- Drawbacks

Introduction

In plain English....

- Objective of K-means is to group similar data points together and discover underlying patterns.
- The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster



Vocab

- Clusters = collection of data points aggregated together
- K = number of clusters

→ again, K-means clustering algorithm tries to group similar items in the form of clusters. The number of groups is represented by K.

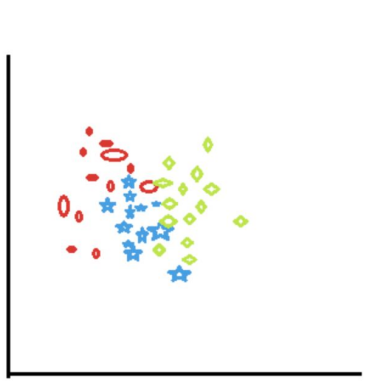


fig 1: before applying k-means clustering

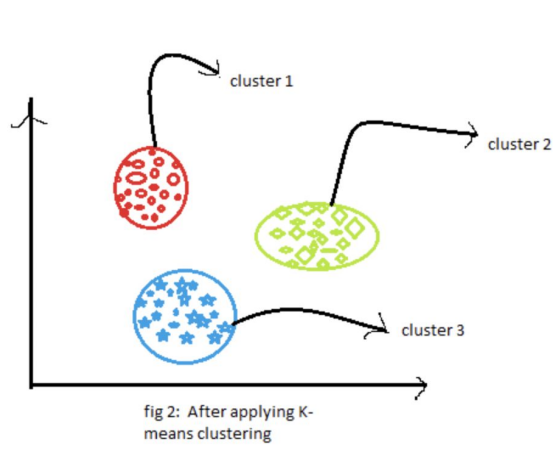
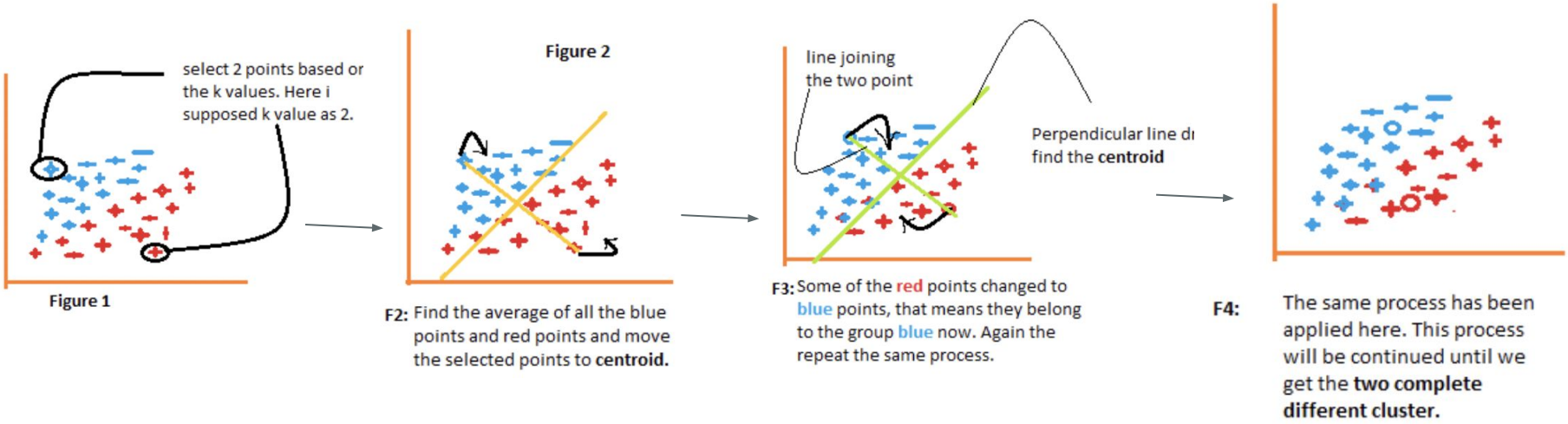


fig 2: After applying K-means clustering

Cit. Analytics Vidhya

How does it work?

1. Select the K-values
2. Initialize the centroids
3. Select the group and find the average

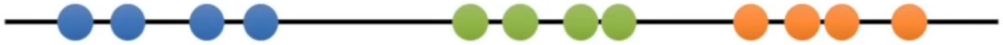


Cit. Analytics Vidhya

Algorithm -
Theoretical Approach

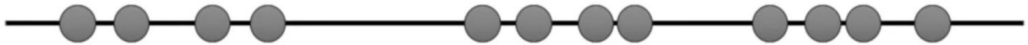
K-Means Clustering Algorithm

Imagine you have some data that are measurements from 3 different types of cells or tumors or penguins. In this case the data make 3 relatively obvious clusters.

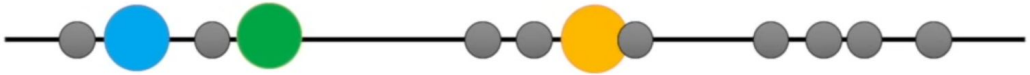


But, if we want a computer to do it for us we can use K-means clustering:

Step 1: We start with a raw data and define the number of clusters that we want for our data. This is “K” in “K-means clustering”.

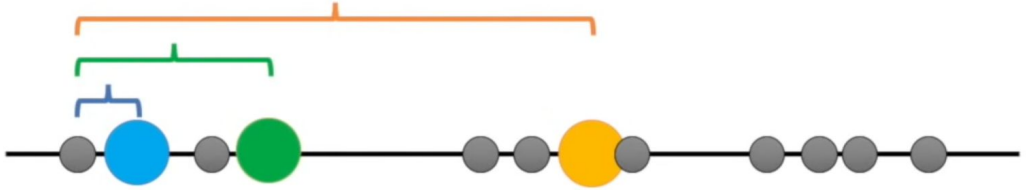


Step 2: Randomly select 3 distinct data points.

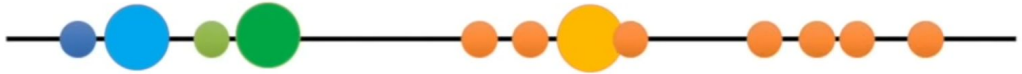


K-Means Clustering Algorithm

Step 3: measure the distance between each point and 3 initial clusters.



Step 4: assign each point to their nearest cluster.



Step 5:calculate the mean of each cluster.

We repeat steps 1-4 using the mean values instead of cluster values.

K-Means Clustering Algorithm

→ Quality of the clustering can be assessed by measuring variation in each cluster.

→ K-means will do the whole process over and over with different starting points until:

- There is no change in Centroid values because the clustering has been successful.
- The defined number of iterations has been achieved.

→ Each time we add a new cluster the total variation within each cluster is smaller than before.

→ There is a high reduction in variation with K=3 compared to K=2, but after K=3 the variation does not go down as quickly.

→ K-means assumes spherical shapes of clusters and doesn't work well when clusters are in different shapes such as elliptical clusters.

Implementation

K-Means Clustering Implementation

[1]:
```
import numpy as np
from numpy.linalg import norm
```

Importing libraries

[2]:
```
class Kmeans:
    '''Implementing Kmeans algorithm.'''
```

Kmeans class - encapsulates the K-means clustering logic.

K-Means Clustering Implementation

```
def __init__(self, n_clusters, max_iter=100, random_state=123):
    self.n_clusters = n_clusters
    self.max_iter = max_iter
    self.random_state = random_state
```

Constructor (‘**__init__**’ method) :
- The constructor initializes the K-means object with three parameters:

- ‘**n_clusters**’: The number of clusters to create.

- ‘**max_iter**’: The maximum number of iterations for the algorithm (default is 100).

- ‘**random_state**’: A seed for random number generation to ensure reproducibility (default is 123).

```
def initializ_centroids(self, X):
    np.random.RandomState(self.random_state)
    random_idx = np.random.permutation(X.shape[0])
    centroids = X[random_idx[:self.n_clusters]]
    return centroids
```

Initialization of Centroids (‘**initializ_centroids**’ method):

-This method initializes the cluster centroids randomly by permuting the indices of the data points and selecting the first ‘**n_clusters**’ data points as centroids.

K-Means Clustering Implementation

```
def compute_centroids(self, X, labels):
    centroids = np.zeros((self.n_clusters, X.shape[1]))
    for k in range(self.n_clusters):
        centroids[k, :] = np.mean(X[labels == k, :], axis=0)
    return centroids
```

Compute Centroids (‘**compute_centroids**’ method):

- This method calculates the new centroids for each cluster by taking the mean of the data points assigned to that cluster.

```
def compute_distance(self, X, centroids):
    distance = np.zeros((X.shape[0], self.n_clusters))
    for k in range(self.n_clusters):
        row_norm = norm(X - centroids[k, :], axis=1)
        distance[:, k] = np.square(row_norm)
    return distance
```

Compute Distance (‘**compute_distance**’ method):

- This method computes the squared Euclidean distance between each data point and all cluster centroids.

K-Means Clustering Implementation

```
def find_closest_cluster(self, distance):
    return np.argmin(distance, axis=1)
```

Find Closest Cluster (‘**find_closest_cluster**’ method):

- This method identifies the closest cluster for each data point by finding the index of the cluster with the minimum distance.

```
def compute_sse(self, X, labels, centroids):
    distance = np.zeros(X.shape[0])
    for k in range(self.n_clusters):
        distance[labels == k] = norm(X[labels == k] - centroids[k], axis=1)
    return np.sum(np.square(distance))
```

Compute Sum of Squared Errors (SSE) (‘**compute_sse**’ method):

- SSE is a measure of the quality of a clustering. This method computes the sum of squared distances between data points and their assigned cluster centroids.

K-Means Clustering Implementation

```
def fit(self, X):
    self.centroids = self.initializ_centroids(X)
    for i in range(self.max_iter):
        old_centroids = self.centroids
        distance = self.compute_distance(X, old_centroids)
        self.labels = self.find_closest_cluster(distance)
        self.centroids = self.compute_centroids(X, self.labels)
        if np.all(old_centroids == self.centroids):
            break
    self.error = self.compute_sse(X, self.labels, self.centroids)
```

Fit ('**fit**' method):

- The '**fit**' method performs the K-means clustering on the input data 'X'.

- It initializes centroids, iteratively assigns data points to clusters, updates centroids, and stops when the centroids no longer change significantly or when the maximum number of iterations is reached.

```
def predict(self, X):
    distance = self.compute_distance(X, self.centroids)
    return self.find_closest_cluster(distance)
```

Predict ('**predict**' method):

- The '**predict**' method assigns new data points to the nearest cluster based on the centroids learned during the fitting process. It returns cluster labels for the new data.

Applications

Applications of K-means Clustering

Customer Segmentation

K-means clustering is widely used in market segmentation to identify distinct groups of customers based on their demographic, psychographic, and behavioural attributes. This allows businesses to personalize their marketing strategies and enhance customer satisfaction.

Image Compression

By applying K-means clustering to pixel values, images can be compressed by reducing the number of colors used. This not only reduces storage space but also speeds up image processing and transmission without significant loss of visual quality.

Anomaly Detection

K-means clustering can help identify outliers in data. By defining normal clusters, any data point that does not belong to a cluster can be considered an anomaly. This is useful in fraud detection, network intrusion detection, and other anomaly detection applications.

Use Cases of K-means Clustering in Industry

Retail

In the retail industry, k-means clustering is used for store layout optimization, inventory management, and customer segmentation for targeted promotions.

Finance

Financial institutions apply k-means clustering for credit risk assessment, fraud detection, and portfolio analysis to identify investment opportunities.

Healthcare

K-means clustering aids in patient segmentation, disease diagnosis, and drug discovery, allowing healthcare providers to deliver personalized treatments and improve patient outcomes.

Image Compression

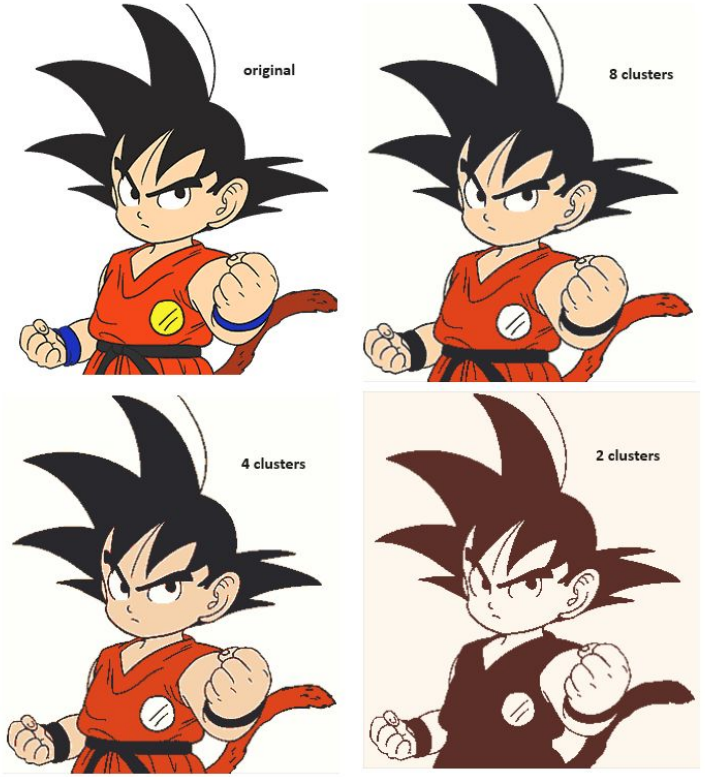
Image compression using K-means

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

def read_image():
    img = cv2.imread('goku.png')

    # Convert the image from BGR to RGB
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Scaling the image so that the values are in the range of 0 to 1
    img = img / 255.0

    return img

def initialize_means(img, clusters):
    # reshaping it or flattening it into a 2d matrix
    points = img.reshape((-1, img.shape[2]))
    m, n = points.shape

    # means is the array of assumed means or centroids.
    means = np.zeros((clusters, n))

    # random initialization of means.
    for i in range(clusters):
        rand_indices = np.random.choice(m, size=10, replace=False)
        means[i] = np.mean(points[rand_indices], axis=0)

    return points, means

# Function- To measure the euclidean distance (distance formula)
def distance(x1, y1, x2, y2):
    dist = np.square(x1 - x2) + np.square(y1 - y2)
    dist = np.sqrt(dist)
    return dist
```

```python
def k_means(points, means, clusters):
    iterations = 10
    m, n = points.shape

    # these are the index values that correspond to the cluster to
    # which each pixel belongs to.
    index = np.zeros(m)

    # k-means algorithm.
    while iterations > 0:
        for j in range(m):
            # initialize minimum value to a large value
            min_dist = float('inf')
            temp = None

            for k in range(clusters):
                x1, y1 = points[j, 0], points[j, 1]
                x2, y2 = means[k, 0], means[k, 1]

                if distance(x1, y1, x2, y2) <= min_dist:
                    min_dist = distance(x1, y1, x2, y2)
                    temp = k
                    index[j] = k

        for k in range(clusters):
            cluster_points = points[index == k]
            if len(cluster_points) > 0:
                means[k] = np.mean(cluster_points, axis=0)

        iterations -= 1

    return means, index
```

Image compression using K-means

RESULTS:



Drawbacks

Drawbacks

Choosing k manually.

Use the “Loss vs. Clusters” plot to find the optimal (k).

Being dependent on initial values.

For a low k, you can mitigate this dependence by running k-means several times with different initial values and picking the best result. As k increases, you need advanced versions of k-means to pick better values of the initial centroids (called **k-means seeding**).

Clustering data of varying sizes and density.

k-means has trouble clustering data where clusters are of varying sizes and density..

Clustering outliers.

Centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. Consider removing or clipping outliers before clustering.

Scaling with number of dimensions.

As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. Reduce dimensionality either by using **PCA** on the feature data, or by using “spectral clustering” to modify the clustering algorithm as explained below.