EP 4130 / PH 6130
Data Science Analysis
Analysis of K-means Clustering

Sanket Ranade - EE19BTECH11012
Pranav Kumar Kota - EE19BTECH11051
Indian Institute of Technology, Hyderabad

Contents

1 Abstract

In this paper, we analyse application of K-means clustering in digit classification and image compression and study it's performance on different structures of data.

2 Introduction

2.1 Theory

• K-means Clustering is an Unsupervised Learning algorithm, which groups unlabeled data into different clusters.

• Clustering of data points is based on a similarity measure such as euclidean distance between points.

• It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.

• The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

2.2 Algorithm

Algorithm 1 K-means Clustering

Require: Data Vectors $\{x_n\}_{n=1}^N$, number of clusters K
  for $n \leftarrow 1...N$ **do** ▷ Initialize all of the responsibilities
    $r_n \leftarrow [0,0,...,0]$ ▷ Zero out the responsibilities
    $k' \leftarrow$ RandomInteger(1,K) ▷ Make one of them randomly one to initialize
    $r_{nk'} = 1$
  end for
  repeat
    for $k \leftarrow 1...K$ **do** ▷ Loop over the clusters
      $N_k \leftarrow \sum_{n=1}^N r_{nk}$ ▷ Compute the number assigned to cluster k
      $\mu_k \leftarrow \frac{1}{N_k} \sum_{n=1} N r_{nk} x_n$ ▷ Compute the mean of the kth cluster
    end for
    for $n \leftarrow 1...N$ **do** ▷ Loop over the data
      $r_n \leftarrow [0,0,...,0]$ ▷ Zero out the responsibilities
      $k' \leftarrow \arg\, min_k \; ||x_n - \mu_k||^2$ ▷ Find the closest mean
      $r_{nk'} = 1$
    end for
  until none of the r_n change
  return assignments $\{r_n\}_{n=1}^N$ for each datum, and cluster means $\{\mu_k\}_{k=1}^K$

3 Mathematical background

• X is the set of input vectors

• r_n := one-hot encoded vector denoting class of input x_n, for $x \in X$

• μ_k := Mean of vectors $x : x \in \mathscr{C}^k$

• Loss function: $\mathrm{J}\left(\{\mathrm{r}_n\}_{n=1}^N, \{\mu_k\}_{k=1}^K\right)$

• Given these parameters and the data vectors x_n, One intuition is that good settings of r_n and μ_k will be those in which as many of the data as possible can be near their assigned μ_k.

• This fits well with the compression view of K-means: if each of the x_n was replaced by the appropriate μ_k, then better solutions will have this error be very small on average. We write this as an objective function in terms of the r_n and μ_k:

$$J\left(\{r_n\}_{n=1}^{N}, \{\mu_k\}_{k=1}^{K}\right) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n - \mu_k||_2^2 \tag{1}$$

• Here, we assume that the distance is euclidian. This function sums up the squared distances between each example and the prototype it belongs to.

• The K-means algorithm minimizes alternating between 1) minimizing each of the r_n, and 2) minimizing each of the μ_k.

• **Minimizing the r_n:** The r_n only appears in one of the outer sums. There are only K possible values for r_n and so we can minimize it (holding everything else fixed) by choosing r_{nk} = 1 for the cluster that has the smallest distance:

$$r_{nk} = \begin{cases} 1, & \text{k = argmin}_{k'}||x_n - \mu_k||_2^2 \\ 0, & \text{otherwise} \end{cases}$$

• **Minimizing the μ_k:** Having fixed everything else, we note that each μ_k only depends on one of the parts of the inner sums. We can think about the objective function written in terms of only one of these:

$$J(\mu_k) = \sum_{n=1}^{N} r_{nk}||x_n - \mu_k||_2^2 \tag{2}$$

$$J(\mu_k) = \sum_{n=1}^{N} r_{nk}(x_n - \mu_k)^T(x_n - \mu_k) \tag{3}$$

To minimize, we differentiate this objective with respect to μk , set the resulting gradient to zero, and then solve for μ_k

$$\nabla_{\mu_k} J(\mu_k) = \nabla_{\mu_k} \sum_{n=1}^{N} r_{nk}(x_n - \mu_k)^T(x_n - \mu_k) \tag{4}$$

$$\nabla_{\mu_k} J(\mu_k) = \sum_{n=1}^{N} r_{nk}\nabla_{\mu_k}(x_n - \mu_k)^T(x_n - \mu_k) \tag{5}$$

On simplifying, we get

$$\nabla_{\mu_k} J(\mu_k) = -2\sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0 \tag{6}$$

$$\sum_{n=1}^{N} r_{nk} x_n = \mu_k \sum_{n=1}^{N} r_{nk} \tag{7}$$

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}} \tag{8}$$

• Thus, for Euclidean distances anyway, taking the average of the assigned data gives the mean that minimizes the distortion (holding everything else fixed).

• It can also useful to think of the K-means clustering algorithm as finding a Voronoi partition (partitioning a plane into regions close to each of a given set of objects) of the data space.

4 Practical Applications

4.1 Digits - Classification

We apply the K means algorithm to the Digits dataset, which is a collection of images of handwritten digits. We set the clusters as 10, and find the cluster centers.

Figure 1: Visualising the cluster centers

```
# importing all the required libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.metrics import accuracy_score, PCA, load_digits
from sklearn.metrics import confusion_matrix
from scipy.stats import mode

# loading the dataset into variable
digits = load_digits()

# use the k-means algorithm to estimate clusters
estimate = KMeans(n_clusters=10)
clusters = estimate.fit_predict(digits.data)
estimate.cluster_centers_.shape

# printing the images
fig = plt.figure(figsize=(8,3))
for i in range(10):
    ax = fig.add_subplot(2,5,1+i,xticks=[],yticks=[])
    # display
    ax.imshow(estimate.cluster_centers_[i].reshape((8,8)), cmap
   =plt.cm.binary)

# assigning actual labels to obtained cluster labels
labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]


# visualising clusters using PCA (Principle Component Analysis)
X = PCA(2).fit_transform(digits.data)

# setting arguments
kwargs = dict(cmap = plt.cm.get_cmap('rainbow', 10), edgecolor=
   'none', alpha=0.6)

# subplots
fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax[0].scatter(X[:, 0], X[:, 1], c=labels, **kwargs)
ax[0].set_title('Learned cluster labels')

```

```
ax[1].scatter(X[:, 0], X[:, 1], c=digits.target, **kwargs)
ax[1].set_title('True labels');

# printing accuracy of cluster predictions
accuracy_score(digits.target, labels)

# confusion matrix
print(confusion_matrix(digits.target, labels))

# plotting the confusion matrix
plt.imshow(confusion_matrix(digits.target, labels),cmap='Greens
    ', interpolation='nearest')
plt.colorbar()
plt.grid(False)
plt.ylabel('true')
plt.xlabel('predicted');
```

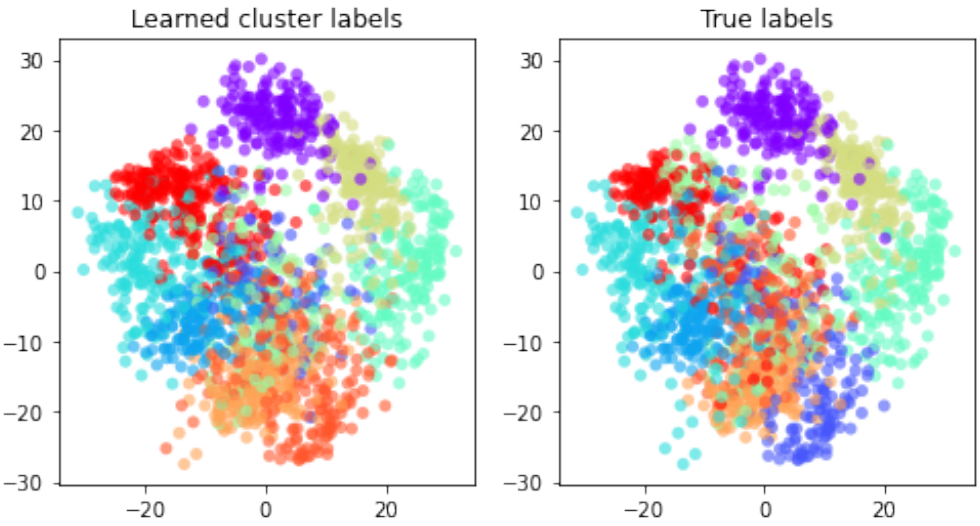


Figure 2: PCA visualisation of clusters

We observe an accuracy of about 80%

The confusion matrix for the class labels is:

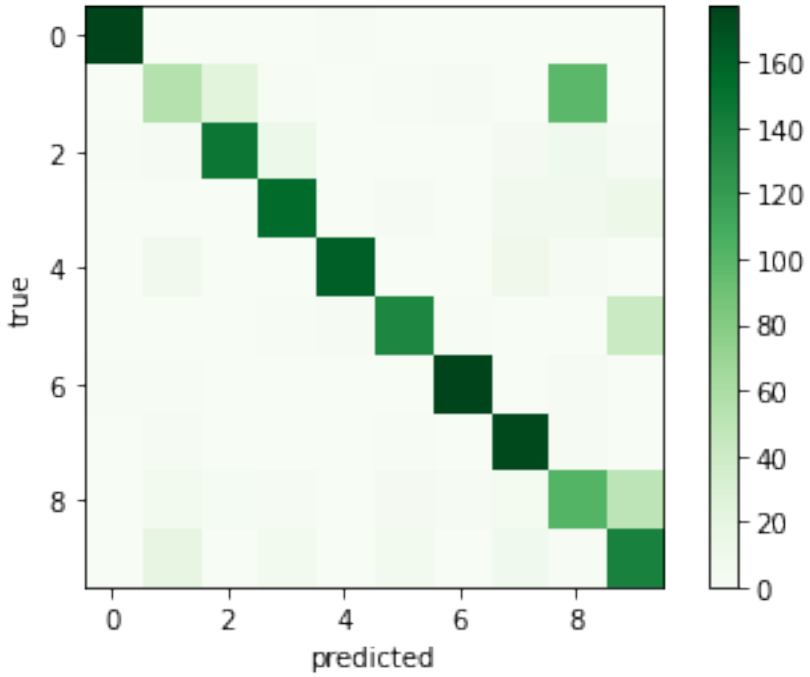


Figure 3: Confusion chart

4.2 Image Compression

Using K means, we cluster the entire pixel space of the input image into k colour clusters.

```
# import libraries
import os
import sys
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Defining the required functions for segmentation
def initialize_K_centroids(X, K):
    m = len(X)
    return X[np.random.choice(m, K, replace=False), :]

def find_closest_centroids(X, centroids):
    m = len(X)
    c = np.zeros(m)
    for i in range(m):
        # Find distances
```

```
        distances = np.linalg.norm(X[i] - centroids, axis=1)

        # Assign closest cluster to c[i]
        c[i] = np.argmin(distances)

    return c

# function for computing means
def compute_means(X, idx, K):
    _, n = X.shape
    centroids = np.zeros((K, n))
    for k in range(K):
        examples = X[np.where(idx == k)]
        mean = [np.mean(column) for column in examples.T]
        centroids[k] = mean
    return centroids

# function for find k-means
def find_k_means(X, K, max_iters=10):
    centroids = initialize_K_centroids(X, K)
    previous_centroids = centroids
    for _ in range(max_iters):
        idx = find_closest_centroids(X, centroids)
        centroids = compute_means(X, idx, K)
        if (centroids == previous_centroids).all():
            # The centroids aren't moving anymore.
            return centroids
        else:
            previous_centroids = centroids

    return centroids, idx

# loading image from local machine
def Load_image(path):
    image = Image.open(path)
    return np.asarray(image) / 255

# Importing the image
image_path = '/Users/pranav/Desktop/Image compress k Means/
   River/image.png'
image = Load_image(image_path)
w, h, d = image.shape

# Reshaping the image to feed into kmeans algorithm
X = image.reshape((w * h, d))
```

```
# the desired number of colors in the compressed image
K = 20

# obtaining clusters
colors, _ = find_k_means(X, K, max_iters=10)
idx = find_closest_centroids(X, colors)

idx = np.array(idx, dtype=np.uint8)
X_reconstructed = np.array(colors[idx, :] * 255, dtype=np.uint8
    ).reshape((w, h, d))

compressed_image = Image.fromarray(X_reconstructed)

# saving the compressed image
compressed_image.save('/Users/pranav/Desktop/Image compress k
    Means/River/out.png')

# displaying the image
plt.figure(figsize=(10,10))
plt.imshow(compressed_image)
plt.show()
```

The obtained compressed image:

(a) Original Image (b) Compressed Image

5 Analysis of the method

5.1 Digits - Classification

- The accuracy of predictions is around 80%
- There exist other supervised learning algorithms, which give much better accuracy on predictions. However, for a basic unsupervised algorithm this is still impressive.

5.2 Image Compression

In the image we have analysed, the original image was of size $364 \times 364 \times 3$

- Using the following snippet, we can compute the number of unique colours our image contains, and the corresponding pixel values.

```
colors, counts = np.unique(image.reshape(-1, 3), axis=0,
   return_counts=True)
print(colors.shape)
print(counts.shape)
'''
output - (57019, 3)
         (57019, )
'''
```

- The colour space has been compressed from 57019 to 20. This is a significant reduction in data required for representation.
- However, the quality of image as perceived by humans has also dropped. We see that brightness of some patches has been diminished and gradation is not smooth anymore.

We now look at the effect of K on compression quality:

(a) K = 5

(b) K = 10

(c) K = 15

(d) K = 20

(e) K = 25

(f) K = 30

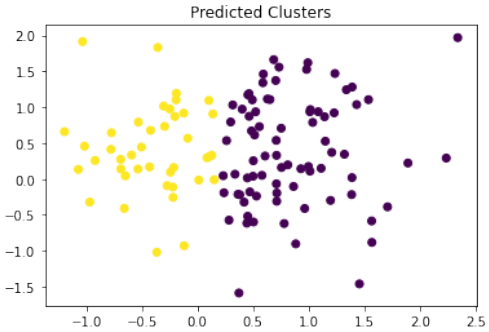
- As K increases, the image quality increases

5.3 General

In this section, we will analyse the result of choosing the wrong K, having intermixed datasets, and having outliers.

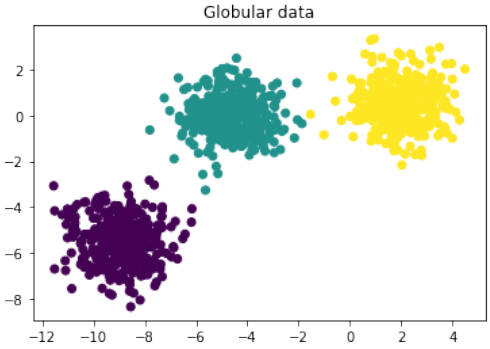
- The following is the result of K-means clustering in a mixed dataset.

(a) Test distribution

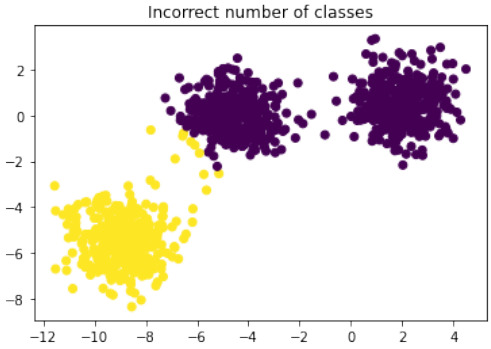


(b) Result of K-means; $k = 2$

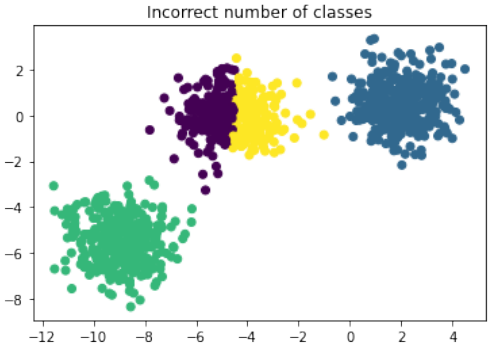
- The following is the result of clustering with the wrong value of K:



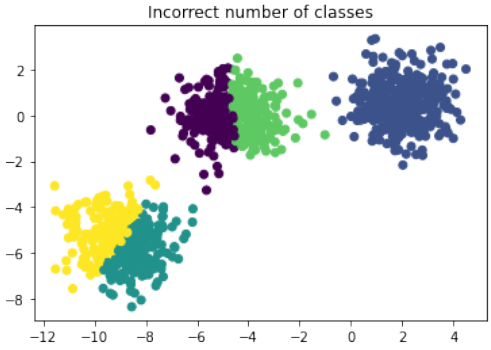
(a) Test distribution



(b) Result of K-means; $k = 2$

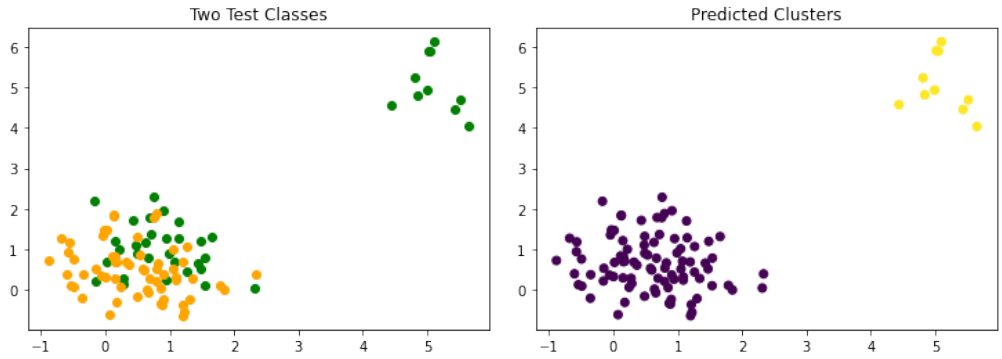


(c) Result of K-means; $k = 4$



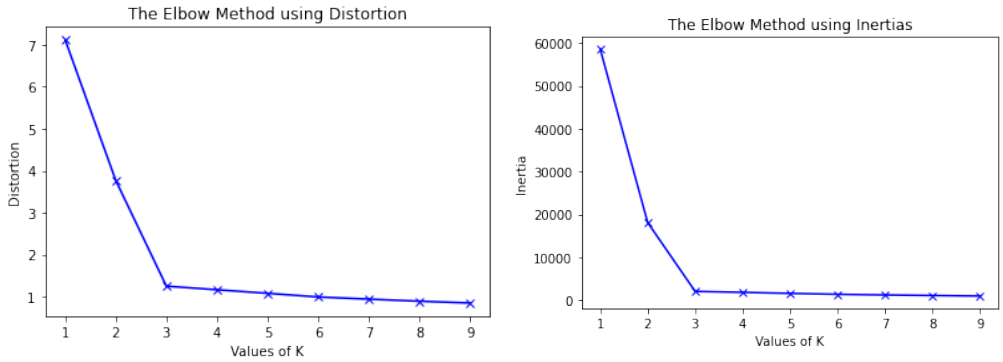
(d) Result of K-means; $k = 5$

- The following is the result of K-means clustering in the presence of outliers.

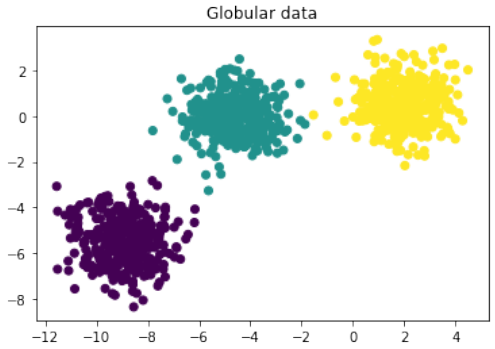


(a) Test distribution (b) Result of K-means; $k = 2$

- We now show the elbow method to obtain the optimal value for K, for the given dataset.
  - Distortion: It is calculated as the average of the squared distances from the cluster centers of the respective clusters. Typically, the Euclidean distance metric is used.
  - Inertia: It is the sum of squared distances of samples to their closest cluster center.



(a) Distortion (b) Inertia



(c) Dataset

- From the distortion and inertial plots, we can infer that the optimal value for K=3, as is clear from the dataset shown

6 Summary

We summarise some of its advantages and limitations below.

6.1 Advantages

• **Simple to implement:**
This algorithm is fairly simple to implement as compared to other clustering algorithms.

• **Guaranteed convergence:**
Since it is guaranteed to converge to a set of centroids defining clusters, it is very powerful as a tool for data analysis.

• **Adaptability:**
This algorithm is well adaptable to new datasets.

6.2 Limitations

• **Shape of the data:**
K-means algorithm analyses the underlying structure of the data well, if clusters have are globular. Since the algorithm tries to cluster classes compactly, it performs poorly when data is highly correlated.

• **Knowledge of hyper-parameter K:**
To run the clustering algorithm, one has to define K.Hence, it is a difficult task to decide it's value, and one has to use techniques like the elbow method to find an optimal K.

• **Impact of outliers:**
Centroids can be dragged by outliers, or outliers are assigned their own cluster instead of being ignored. So, the outliers have to be truncated from the data manually, otherwise the performance will be poor.

7 Code repository

All the code used for this project can be found at Github.

8 References

- K-means Clustering : Wikipedia
- Image Segmentation : Wikipedia
- Voronoi Diagram : Wikipedia
- Analysis of K-means Clustering : Google Developers