Reykjavik University

Master Degree in Computer Science

Course of Data Mining and Machine Learning

K-Means

Author

Davide Parente

Instructor

Eyjólfur Ingi Ásgeirsson

06. November 2022

1 Section 1.6

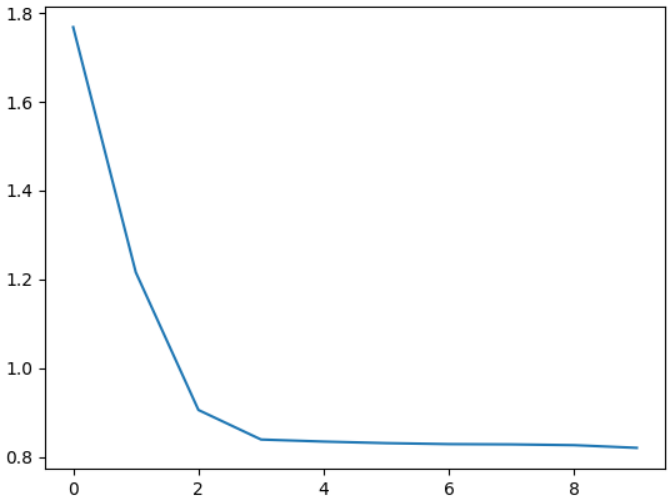

Figure 1: 1_6_1.png

2 Section 1.7

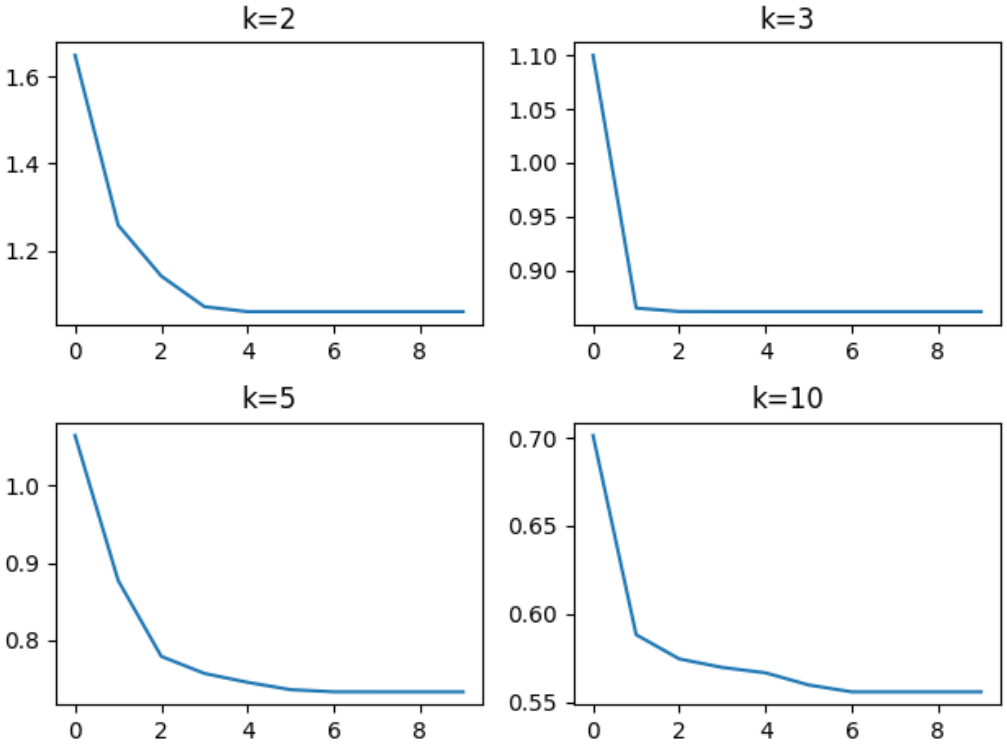

Figure 2: 1_7_1.png

3 Section 1.8

The objective function is the value that indicates how well the algorithm is performing. In the k-means algorithm is not clear how to choose the value of K which indicates the number of clusters.
The k-Means algorithm works in this way:

- Inizializate the prototypes, then iterate between:
  1. E-step: assign each data point to a cluster based on a metric between it and the prototype
  2. M-step: update the prototype to be the cluster means, based on the new assignments
- These iterations will be interrupted based on distinct criteria such as the number of steps, the reach of a threshold, and so forth.

These two steps have as a goal to minimize the value of J,*cost function*, E-steps will minimize it with respect to r_{nk} while M-step respect to μ_k.
The numbers of Clusters in K-Means have clearly defined variables that do not overlap. The clusters must have at least one item and items must not overlap in the clusters. Higher numbers of the clusters favor accuracy since the features are narrowed down to specific ones. Data may be clustered using the features of the items[1].
As can be seen from the graphs above, the algorithm performs better, achieving a lower J in the case of 10 clusters. However, it must be considered that clusters have a computation cost, and therefore considering only this metric as an evaluation is erroneous. So, in this case, the value of clusters that performs better on the measure J is *k=10*.
By setting the sample value equal to the cluster value, theoretically each cluster should be able to contain one sample. For example in the case of the Iris dataset, each cluster should contain one and only one type of flower. Assigning n=k in the case of the Iris Dataset may be a good choice because the different classes have few features and are well clustered. Nevertheless, the accuracy obtained is still not optimal (≈ 0.83).
In conclusion, I think that setting $k=n$ is a choice that must be done with respect to the dataset. Nevertheless, I think is not a good approach to setting k just in base to n, the best approach to find the number of clusters is to have a comparison between different results got from different values of k.
An example of $k=n$ that can't work is with breast cancer dataset where too many features act.

[1] https://studycorgi.com/number-of-clusters-in-k-means-clustering/

4 Section 1.10

The accuracy and confusion matrix follows:

```
Accuracy: 0.8466666666666667
Confusion Matrix:
[[50.  0.  0.]
 [ 0. 45.  5.]
 [ 0. 18. 32.]]
```

Figure 3: Results

5 Section 2

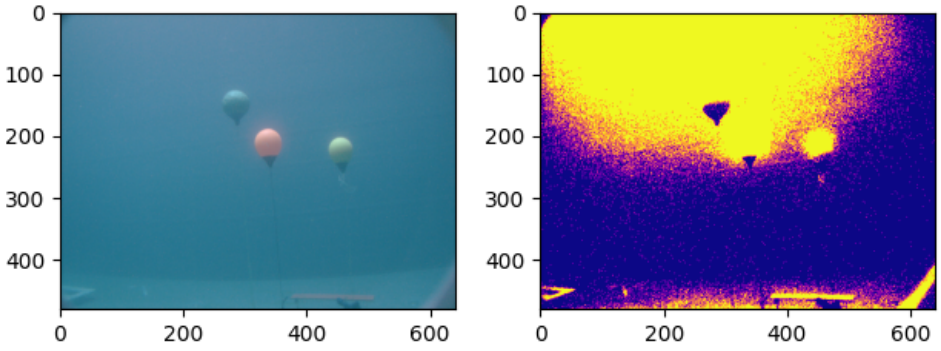

Figure 4: 2_1_1.png

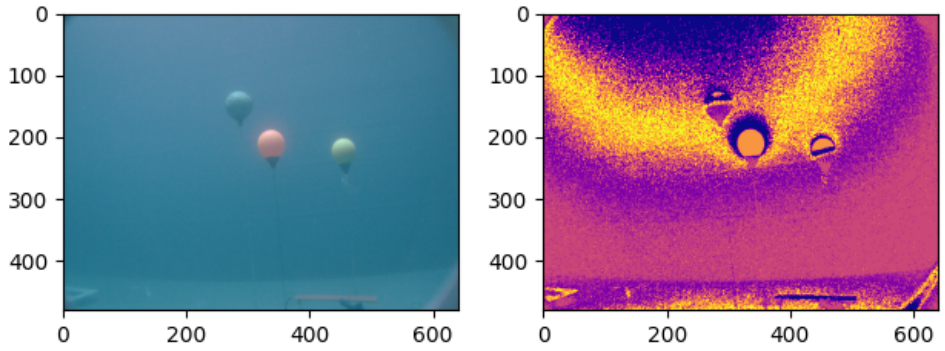

Figure 5: 2_1_2.png

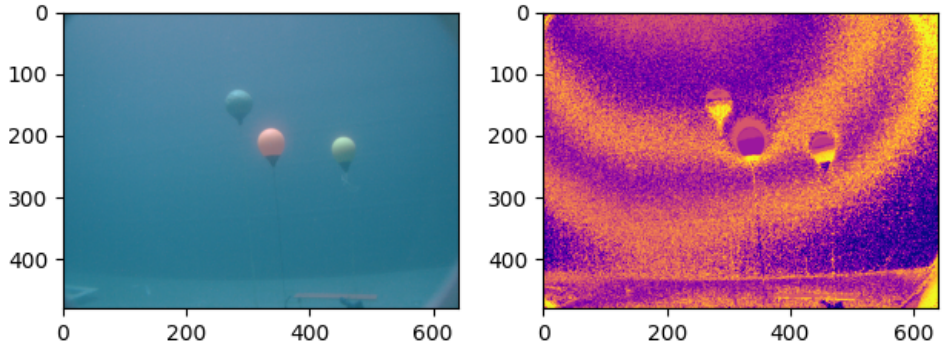

Figure 6: 2_1_3.png

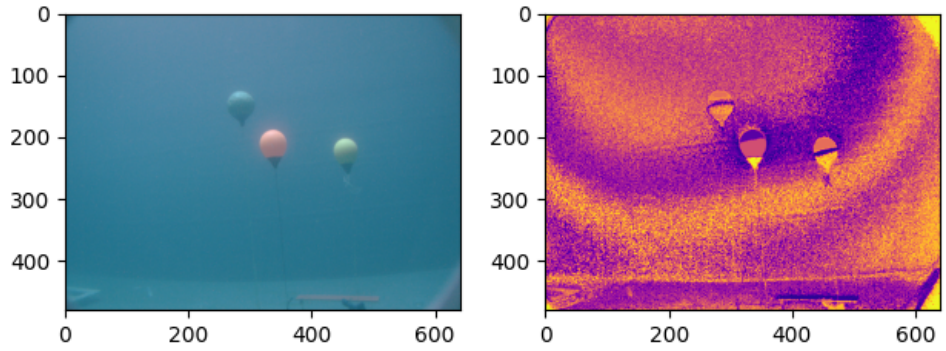

Figure 7: 2_1_4.png

6 Section 3.1

A Gaussian mixture model (GMM) is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. A GMM can be used for data clustering, density estimation, and many other applications.
The Gaussian mixture model is a generalization of the k-means clustering algorithm. Instead of having a fixed number of clusters, the GMM algorithm models the data as a mixture of a variable number of Gaussian distributions. This algorithm is a generalization of the k-means algorithm. Instead of having a fixed number of clusters, the GMM algorithm models the data as a mixture of a variable number of Gaussian distributions.

The GMM algorithm starts by randomly initializing the parameters of the Gaussian distributions. Then, it assigns each data point to the Gaussian distribution that has the highest probability of generating that data point. Finally, it updates the parameters of the Gaussian distributions by taking the maximum likelihood estimate of the data.

The mixing coefficient are probabilities so they are in the interval [0,1].

$$\forall k : \pi_k \geq 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$

The purpose of this value is to assign the data in the best cluster reducing performing the total likelihood. The marginal distribution over z is specified also in terms of the mixing coefficient.
These values are updated at each iteration of the M-step and has one of the most important role because indicates the probability to take a specific Gaussian to describe a point. It indicates the percentage of influence of a gaussian in the final function.

7 Section 3.2

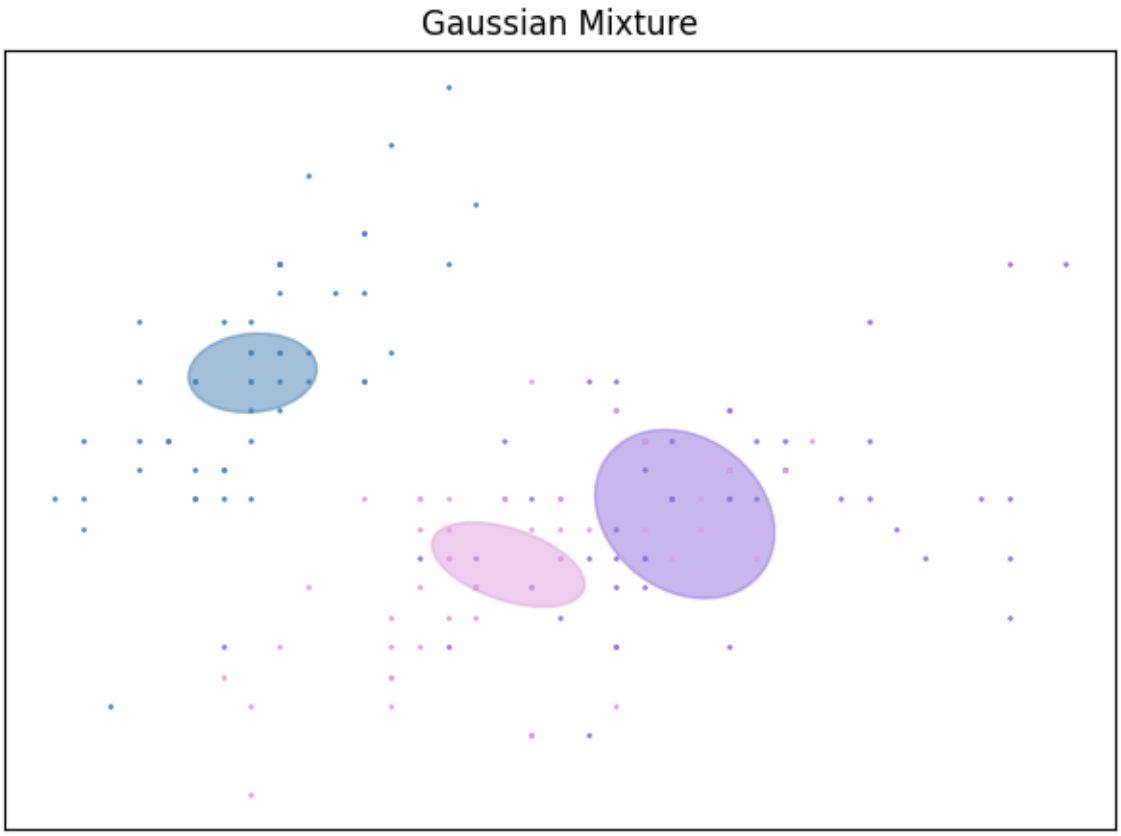

Figure 8: 3_2_1.png

8 Indep

The template.py file with the function *kmeans_predict(X: np.ndarray,t: np.ndarray,classes: list,num_its: int)* only allows us to change the dataset and the number of iterations. What about the number of clusters? In my function I have associated the number of classes with the number of clusters, but this is a major limitation.
In the function where we tested the values of k, we only evaluated the change in J but did not plot it. So I have developed a new function to test different value of k and also plot it to see more in detail the results.
The function I wrote follows:

```
def indep1(
    X: np.ndarray,
    t: np.ndarray,
    k: int, #Number of clusters
    classes: list,
    num_its: int
    ) -> np.ndarray:
    Mu,R,Js= k_means(X,k,num_its)
    bestCluster = np.zeros((k,len(classes)))
    colors=['blue','orange','green','olive','purple','brown','pink','yellow','gray','cyan']
    for i in range(t.shape[0]):
        bestCluster[np.argmax(R[:][i])][t[i]]+=1
    ClusterFinal=np.zeros(k)
    for i in range(k):
        ClusterFinal[i]=np.argmax(bestCluster[i])
    prediction= np.zeros(R.shape[0])
    for i in range(R.shape[0]):
        x, y = X[i,0],X[i,2]
        prediction[i]=ClusterFinal[np.argmax(R[i])]
        if(prediction[i]==t[i]):
            plt.scatter(x, y,c=colors[np.argmax(R[i])],linewidths=2)
        else:
            plt.scatter(x, y,c=colors[np.argmax(R[i])],edgecolors='red',linewidths=2)
    print("Accuracy:",accuracy_score(t,prediction))
    print("Confusion matrix:")
    print(confusion_matrix(t,prediction))
    plt.savefig("indep.png")
    return prediction
```

Figure 9: Code for independent part

The measure generated calling the function with k=10 (*i.e indep1(X,y,10,c,10)* follows:

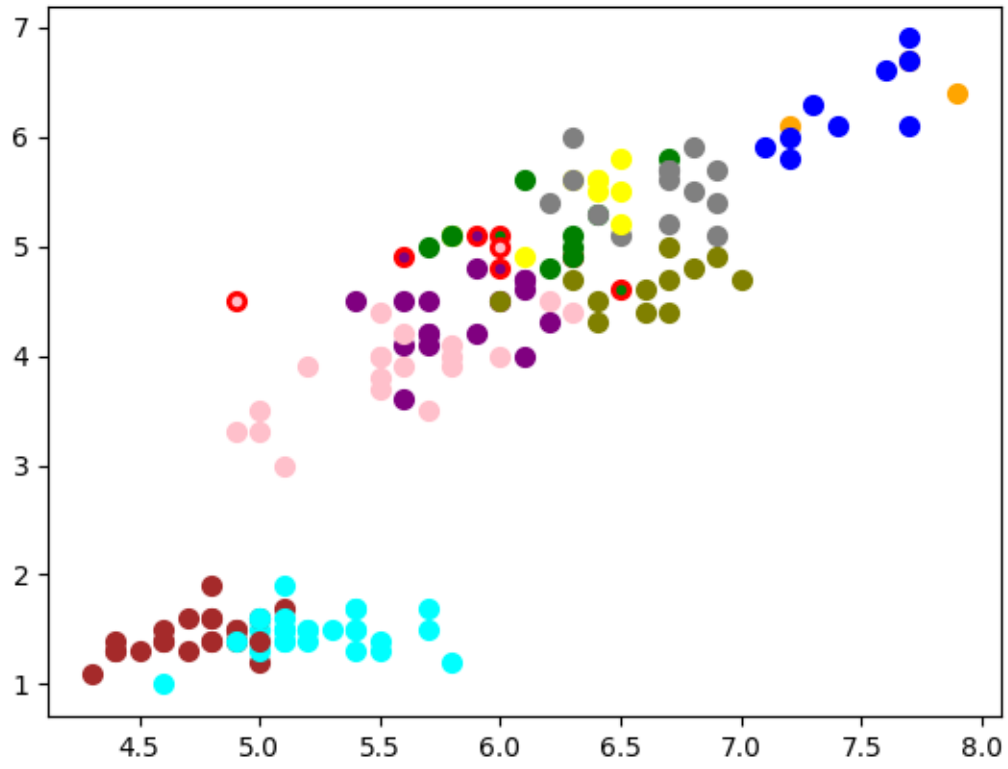

Figure 10: Plot

The point with the red edge are the flower misclassified. Each cluster has a different color. The confusion matrix and accuracy of this plot follows:

```
Accuracy: 0.9466666666666667
Confusion matrix:
[[50  0  0]
 [ 0 47  3]
 [ 0  5 45]]
```

Figure 11: Values

These results are really good because we increase our previous accuracy of 0.10.
Also I wrote another function, without the plotting phase, just to check how accuracy could be improved by greatly increasing the number of clusters.

The results I got follows:

```
Accuracy: 0.96
Confusion matrix:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

Figure 12: 50 cluster

```
Accuracy: 0.9733333333333334
Confusion matrix:
[[49  1  0]
 [ 0 49  1]
 [ 0  2 48]]
```

Figure 13: 80 cluster

I did not plot the result because having 50 or 80 different colours to represent each cluster could be confusing.

In the end, I did the test of point two on a new image.

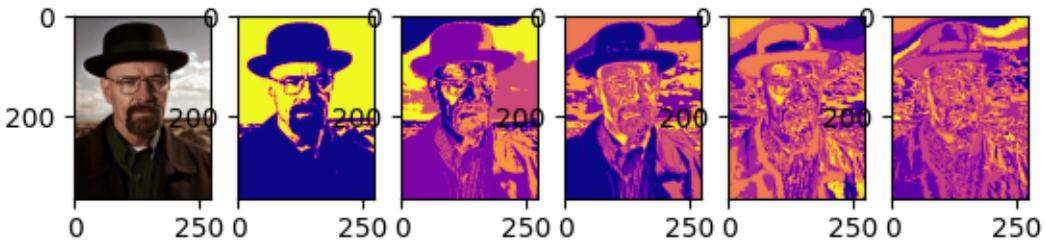

Figure 14: Respectively 2,5,10,20,30 clusters