Clustering Optimization using k-means

Preface

In this paper, we shall demonstrate two approaches for optimizing the k-means algorithm for clustering.

The problem of clustering is a good example of an optimization problem. The problem is, first, how to divide a set of n elements into a given number of k groups. We have many ways to do this, and we need to find the optimal grouping, assigning to each group its best matching elements.

Moreover, in real-life problems, k itself is unknown and needs to be optimized by the machine itself. We can theoretically declare the whole set of elements as one cluster, or consider each element as its own cluster, but that would be pointless. Thus, we need to find a way to optimize the choice of the number k, assuming the first problem itself (finding the optimal grouping, for a given k) has an optimal solution.

There are two kinds of clustering that we would like to consider, which may affect the algorithm that we run,

1. Clustering of scattered details

In this kind of clustering, we try to identify clusters from details that are scattered along the image. This can have various applications, such as **Military Intelligence** (clusters of troops, aircraft, armored vehicles), **Nature Science** (clustered structures of birds, insects, fish), and many more.

2. Identification of objects as clusters

In this kind of clustering, we try to identify large objects, from samples found in the image. A major example of this kind of clustering would be face recognition, where we have many features in some image, and we try to isolate and identify the face of a person, or of several people.

The basic k-means algorithm

One of the classic algorithms for clustering is the k-means algorithm.
This algorithm is based on a very simple concept of acquiring initial data, then adjusting this data until the algorithm stables. This algorithm is called k-means, because we are trying to find $k \in \mathbb{N}$ clusters, out of $n \in \mathbb{N}$ elements, which are supposed to give the optimal clustering because each cluster has a center point, which is the mean of all the points that are grouped together in this cluster. We call this cluster, or the center point of this cluster, a **centroid**.

The basic description of this algorithm, for a given k, is,

1. Initializing Initialize a set of k centroids within the pixels of the image.
2. Association For each element, calculate its distance from the center of each cluster, find the centroid with the minimal distance from this element, and associate the element to this centroid.
3. Recalculation Using the association of the elements, calculate the center point of each centroid again, by taking the mean point of all its associated elements.
4. Iteration Iterate steps 2 and 3 until the algorithm turns stable, that is, there are no more moves of associated elements between centroids.

Algorithm 1 Calculate k-means

```
Require:
Ensure:
  $r \leftarrow true$

  $L \leftarrow size(samples)$

  while $r$ is $true$ do
    $a \leftarrow 0$

    $i \leftarrow 0$
    while $i < L$ do
      $s \leftarrow samples[i]$

      $f \leftarrow null$
      $m \leftarrow null$

      $j \leftarrow 0$
      while $j < k$ do
        $c \leftarrow centroids[j]$

        $dx \leftarrow s.x - c.x$
        $dy \leftarrow s.y - c.y$
        $d2 \leftarrow dx^2 + dy^2$

        if $m$ is null or $d2 < m$ then
          $m \leftarrow d2$
          $f \leftarrow c$
        end if

        if $s.c \neq f$ then
          $a \leftarrow a + 1$
        end if

        $j \leftarrow j + 1$
      end while

      $i \leftarrow i + 1$
    end while

    if $a < 1$ then
      $r \leftarrow false$
    end if

  end while
```

This algorithm, as described, is promised to converge, that is, to achieve a stable state, where the stopping condition (no more moves between centroids) is satisfied.

The proof of convergence is given by the basic observation that the number of grouping options, for a given number of k clusters, out of n elements, is obviously finite, and that on each step, we get a better score on the clustering. A full proof can be found at [1]

Proposition The k-means algorithm does not necessarily give an optimal solution, for a given k
Explanation The given proof only proves that if we start from some initial setting of the system, we are sure to converge, at some point. However, this convergence is to a local optimum only, because, if we start from a different setting, we may converge to another local optimum, possibly to the best existing solution, which we shall refer to as the global optimum.

The Elbow method
Recall from above, one major problem that we have, with the k-means algorithm, is that the native algorithm requires an input number of k, for running. We also recall that the total number of ways to group n points to $1 \leq k \leq n$ clustering is given by the Bell number, which is a sum of the Stirling number, for each k, thus significantly larger.

To automatically choose the optimal number of k, we need a way to compare the scores of different values of k, running under the same conditions. This brings us to an optimization method, called the **Elbow Method**. The basic concept of this method is that we can compute some score on each k, and then present this score as a function of k.

Taking a range of k values, that is, $\{k_1, k_2, k_3, \ldots, k_m\}$, we shall observe that for the lower values of k, the function is decreasing rapidly, while after a certain value of k, the function is taking a significant turn, from a high (negative) slope, into a nearly asymptotic graph. This means that for less than the optimum k value, our clusters are too large, and for more than the optimum, the more k clusters we calculate, in the k-means algorithm running, we do not add any improvement for the clustering, but exactly the opposite, meaning the output clusters will split the real clusters in the image, and not give us any beneficial clustering information.

In other words, the optimal number of k is the turning point of the graph, from the high slope to the asymptote. This is why it is called "elbow" because it resembles a folded arm and the elbow that is the outmost point in the arm. So, if we can compute different ranges of numbers, for different maximal k values, $\{m_1, m_2, m_3, \ldots, m_l\}$, and we get the same elbow (that is, a specific value of k), for each m_i, then we have the optimal number of k, for this image. We can even assume that the optimum will move up and down, but will maintain some boundaries, from which we can take the average k, with or without some weight or probability considerations.

WCSS

The score we are calculating, for the elbow method, is <u>WCSS</u> (Within-Cluster Sum of Square),
which means, we calculate the squared distances of all the elements from their associated centroids,
That is,
$WCSS(k) := \sum_{j=1}^{k} \sum_{i=1}^{m_j} (c_j - p_{j_i})^2 = \sum_{j=1}^{k} \sum_{i=1}^{m_j} (x_j - x_{j_i})^2 + (y_j - y_{j_i})^2$
Where
m_j - the number of elements associated to centroid j
$c_j = (x_j, y_j)$ - the center point of centroid j
$p_{j_i} = (x_{j_i}, y_{j_i})$ - the point i of centroid j

So, at first glance, we are looking for a k value that will yield the minimal WCSS.
However, we can observe, as mentioned before, that we are not looking for a minimal value, but rather for the optimal value. Indeed, if we continue to calculate the k-means for higher values of k,
The WCSS will decrease to the minimum, without giving us any benefit, but actually ruining the clustering.
It is a trivial observation since we can always take $k = n$, that is, declare each element as a separate cluster, thus obtaining $WCSS(k = n) := \sum_{j=1}^{k} \sum_{i=1}^{m_j} 0 = \sum_{j=1}^{k} 0 = 0$,
which is clearly the minimal WCSS possible, while it is obvious that this clustering result is absolutely wrong.
Going back to the elbow method from above, we need to construct an algorithm that will automatically determine the optimal k values, by finding the elbow. So, basically, our algorithm outline will look like this,

1. set k, an initial number of desired clusters
2. calculate k centroids, using the k-means algorithm from above
3. calculate $WCSS(k)$, if it is the optimal value, return k
4. if $k = n$, return k. otherwise, increment k, and repeat step 3.

So, this algorithm can run, theoretically, until $k = n$.
However, we can observe that this is not exactly the desired algorithm, because,
how do we know that we have achieved the elbow, that is, the optimal value?

So, we edit this algorithm in a slightly different manner,
1. set s, the number of successful optimal k calculations
2. set t, the tolerance for optimal k values.
3. set q, a number representing a quant of k values to calculate
4. set k, an initial number of desired clusters
5. calculate k centroids, using the k-means algorithm from above
6. calculate $WCSS(k)$, and store it in an array
7. if k=0 mod q, calculate the optimal k using all the stored WCSS values and store it in an array
8. if the array of optimal k values has s identical values of k, or if their difference is in the range of t, we take their value (in case they are identical), or some average between them, and return this value as the final k value.
9. if $k = n$, return k. otherwise, increment k, and repeat step 5.

Algorithm 2 Calculate ekbow for k-means

```
Require:
  $q$: quant of $k$ for optimum calculations
  $s$: number of successful elbow calculations
  $t$: tolerance of $k$ value
Ensure:
  $L \leftarrow size(samples)$

  $a \leftarrow$ float[]
  $b \leftarrow$ integer[]

  $r \leftarrow true$

  while $r$ is $true$ do
    $a \leftarrow 0$

    $i \leftarrow 0$
    while $i < L$ do
      $s \leftarrow samples[i]$

      $f \leftarrow null$
      $m \leftarrow null$

      $j \leftarrow 0$
      while $j < k$ do
        $c \leftarrow centroids[j]$

        $dx \leftarrow s.x - c.x$
        $dy \leftarrow s.y - c.y$
        $d2 \leftarrow dx^2 + dy^2$

        if $m$ is null or $d2 < m$ then
          $m \leftarrow d2$
          $f \leftarrow c$
        end if

        if $s.c \neq f$ then
          $a \leftarrow a + 1$
        end if
```

```
        $j \leftarrow j + 1$
      end while

      $i \leftarrow i + 1$
    end while
```