Algorithms in Bioinformatics

Joris van Steenbrugge, `Assignment 5`

1 Cluster the data in the file 2dtest.csv, depicted in Fig. 1, into three clusters and print the output. Run multiple times k-means on this dataset (with k=3)

(a) Do you always get the same result

The results are definitely not always the same. With k=3, often the points are contained in 3 clusters but sometimes only in 2. There is also variability in the cluster index (e.g. cluster 1 and cluster 2 or cluster 0 and cluster 1 etc.)

t

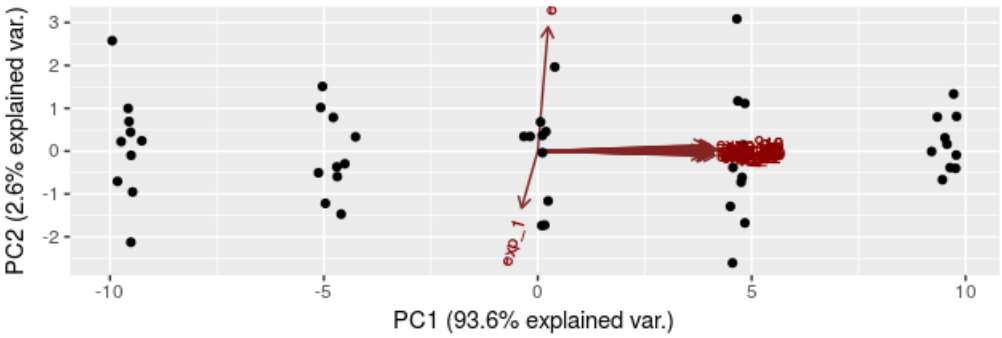
(b) How many iterations are needed for converge

There are on average 2 or 3 iterations needed, but sometimes more.

2 The data in the file LargeSet_1.csv represents measurements of 50 genes in 25 conditions. Run k-means multiple times for k $\in$ { 2, 3, 4, 5, 6 }

(a) What k do you think results in a more consistent clustering? Justify your answer.

I was unsure how to estimate that with so many dimensions, therefore I created a PCA plot. Based on this PCA plot, if I interpret it correctly, I think 5 clusters is the most consistent clustering number.



(b) How many iterations are needed for convergence?

There are on average 4 iterations needed (numbers differ).

3 The data in the file LargeSet_2.csv is depicted in Fig. 2. It contains 400 points in two dimensions. Data are arrange so that the first 200 are in one cluster and the second 200 in another clusters. Run the k-means algorithm with k=2 multiple times. Most likely you will notice that it fails to recover the correct cluster structure. Explain why do you think this is happening.

(a) The correct cluster structure is indeed not fully recovered but the vast majority of the data-points ends up in the correct cluster. I think it depends on where the centroid for each cluster starts. if the centroid for the green cluster in Fig 2 has a high x value, some data-points from the red cluster with a high x value might get assigned to the wrong cluster.

4 Is the algorithm deterministic? Justify the answer

(a) This algorithm does not seem to be deterministic because there are random factors involved. The first centroids for each cluster are chosen by selecting a data-point at random. Therefore the result is not always the same, and thus is the algorithm not deterministic.

5 What is the memory complexity of the algorithm you implemented

(a) For this implementation, the dataset is loaded from a file and stored in memory (d and np). For each data-point a clustering number is stored which is a fixed amount of memory for each data-point. It does not matter how many clusters (k) there are. Therefore the memory complexity seems to be $O(d * np)$

6 How does the running time depend on k, d and np (number of points) ? How do these affect *Niter* (the number of iterations)

(a) The number of dimensions seem to be related in a way that when np increases by one, the running time increases by the number of dimensions, i.e. $O(np * d)$. I do not expect the size of k to impact running time as significantly as np or d.

7 (Optional) Approximate convergence: up to now the iteration on k-means was finished when the centroids did not move between iterations. However, it could be convenient to allow a slight tolerance, so that if the centroids move less than a certain tolerance we consider that the algorithm has converged. Develop your own approach to address this issue. Explain your algorithm and implement it in your code.

(a) My approach to approximately converge the algorithm is by the calculating the absolute difference for each dimension in a centroid with the respective dimensions in the previous centroid of the same cluster. Then the absolute differences are divided by the full previous centroid value of each dimension. Based on an arbitrary cutoff value (e.g. 2%) the clustering algorithm can approximately converge. The algorithm is additionally displayed in **Algorithm 1**

Algorithm 1 Approximate convergence

```
1: procedure ApproxConv(K-means)
2:     for each centroid do
3:         c ← centroid
4:         for each dimension in c do
5:             abs_diff ← c_dim − prev_c_dim
6:             percentage ← abs_diff/prev_c_dim
7:             if percentage < cutoff then
8:                 return True
9:             else
10:                return False
```