Improving K-means with VNS

K-means algorithm produces a local solution which depends on the initial placement of centroids or initial split of data into clusters [5]. Depending on the initial partition, assignment of observations into classes by K-means and the solution quality may differ quite much from run to run. However, once the initial partition of observations into clusters has been selected, the algorithm will produce a deterministic result (conditional on that initial partition).

Multiple K-means initialization methods exist. [5] mentions about 30 methods of K-means initialization which can be grouped into:

- Random, e.g., random selection of observations in the dataset as initial centroids or random splitting of observations into clusters.
- Deterministic that seek to find centroids that would represent dissimilar regions of data.

Random initialization methods are fast, but due to variability of results they produce, multiple K-means runs from different initial centroids/partitions are needed [5]. Based on those runs, the result with the smallest within-cluster sum of squared errors is selected as the final solution. This is a so-called multi-start approach which works quite well on small datasets.

Deterministic initialization methods require more time than random, but as they need just a single run of K-means, they offer substantial time savings when clustering large data sets [5].

We used random selection of observations in the dataset as initial centroids to initialize the K-means algorithm. The reason behind such an initialization is that random selection is likely to pick centroids from dense regions of rather homogenous observations. However, no mechanism exists to guarantee against choosing outliers or points that are too close to each other as initial centroids.

Even though such an initialization calls for the multi-start approach, we did not implement it in our algorithm. Instead, we decided to improve it with VNS algorithm that would ‘push’ K-means out of the local optimum found so far to find an even better solution.

We consulted articles [2] and [4] to explore possible implementations of VNS on K-means. We took approach described in [2] as an inspiration and VNS algorithm we implemented uses the following components:

1. Shaking best centroid(s) obtained so far
2. Local search (K-means)
3. Neighborhood change

We define neighborhood by the number of centroids we shake (k):
N1: k=1 (if only one centroid is shaken),
N2: k=2 (if two centroids are shaken),
….
Nk: k=K (if all centroids are shaken).

The higher k, the more randomness we introduce into our best solution before running local search again. When k=K, this is equivalent to running K-means from scratch again.

VNS algorithm we implemented can be described as follows:

1. Let K-means converge and produce the first solution, compute its within cluster sum of squared errors and set it to best solution (SSE_best).
2. Take k=1 centroids from the best solution available so far, shake them
3. Local search - run K-means again with shaken centroids
4. If improvement found (SSE_new < SSE_best) – keep new centroids, update SSE_best, set k=1 (fall back to N1)
5. Else: neighborhood change - go to neighbourhood k+1
6. Repeat 3-5 until k=K (number of clusters)
7. Repeat 2-6 until the max number of iterations allowed for VNS reached.
8. Return the best solution.

Shaking centroids in our VNS implementation means randomly selecting centroid(s) from the best available solution and replacing them with random observation(s) from the data set. We used the following shaking function:

```
Require: X (dataset), k – number of centroids to shake, C – centroids from the best K-means
run
Function Shaking (X,k,C):
    1. C shuffled <- Shuffle randomly list of centroids C
    2. X shuffled <- Shuffle randomly observations in data set X
    3. Replace first k centroids from C shuffled by the first k observations from X shuffled
    4. Return new centroids C'
```

Our VNS function then combines shaking, local search (as presented by K-means) and neighbourhood change into one algorithm:

```
Require: X - dataset, K – number of clusters, C – centroids from the best K-means run
(having SSE_best), SSE_best – lowest within-cluster sum of squared errors obtained so far,
max_iter – max number of VNS runs through all neighbourhoods.

Function VNS (X, K, C, SSE_best, max_iter)
counter = 1
Repeat
|   k = 1  Neighbourhood index
|       Repeat
|       |       C' <- Shaking(X,k,C)  Shaking centroids
|       |       C'', SSE(C'') <- K-means(X,C')    Local search
|       |       if SSE(C'') < SSE_best:  Comparing with the best solution
|       |               C <- C'',  SSE_best = SSE(C''),  k = 1
|        |        else: k = k+1  **Neighborhood change**
|        Until k = K
|        counter <- counter + 1
Until counter = max_iter
Return **C, SSE_best, cluster attribution for each observation in X**
```

Please see Appendix 2 for the Python code of the VNS algorithm, implemented on the basic K-means algorithm described above. The code can also be found in the file **Basic_Kmeans_with_VNS.py**. **Basic_Kmeans_VNS_with_testing.py** contains the same code plus some testing of the algorithms on our data set.

References

[1] Data set:
https://archive.ics.uci.edu/ml/datasets/abalone?fbclid=IwAR17vaqIW8hmWdw9vGN4E0KqxE4vpkEg-bPRC_y0edyqT3Xd6eH5q8n8gvI

[2] Abdulrahman Alguwaizani, Pierre Hansen, Nenad Mladenovic´, Eric Ngai. Variable neighborhood search for harmonic means clustering, in *Applied Mathematical Modelling 35 (2011) 2688–2694.*

[3] Adibi, M. A., & Shahrabi, J. (2014). A clustering-based modified variable neighborhood search algorithm for a dynamic job shop scheduling problem, in *The International Journal of Advanced Manufacturing Technology*, *70(9-12), 1955-1961.*

[4] E. Camby, G. Caporossi, S. Perron. A parallel algorithm using VNS with shared memory and message passing interface for community detection in complex networks, in *Les Cahiers du GERAD, February 2018.*

[5] M. Emre Celebi, Hassan A. Kingravi, Patricio A. Vela. A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm, in *Expert Systems with Applications, 40(1): 200–210, 2013.*