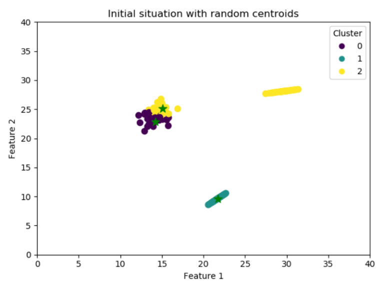
All source code is hosted here.

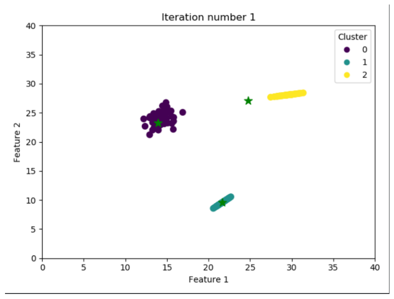
Problem 2.1 (K-Means)

a) You can see the code of k-means implementation here

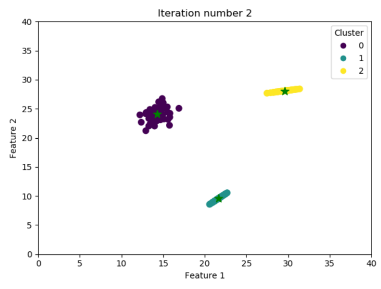
b) Here is shown the clustering obtained with k = 3 and 3 iterations.



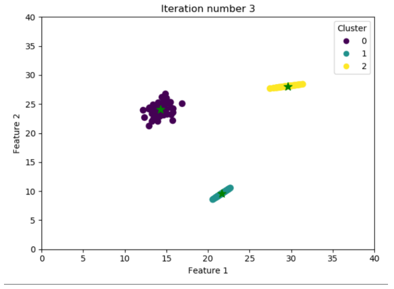
(a) Initial random centroids



(b) First iteration



(c) Second iteration

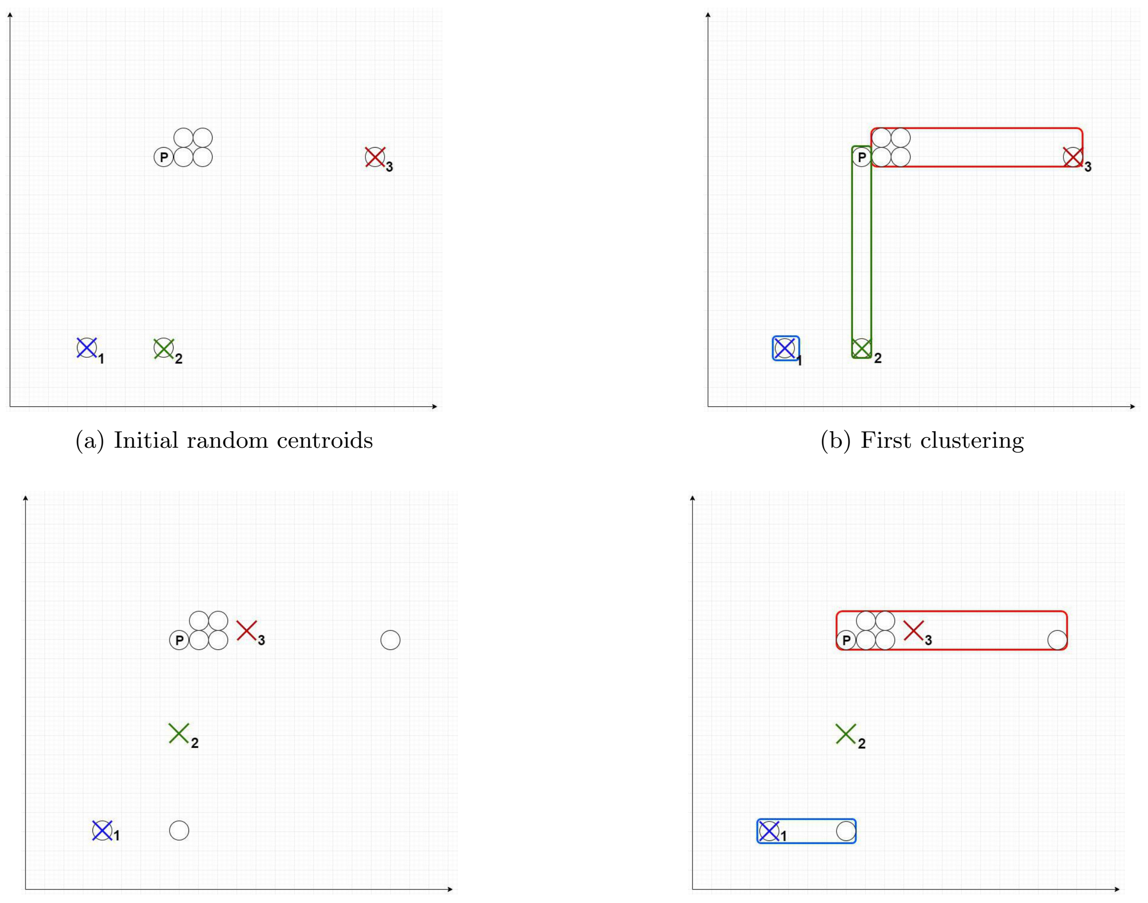


(d) Third iteration

Abbildung 1: Clustering obtained with k = 3 and 3 iterations

c) In general, using K-Means clustering algorithm, we can have the situation of an empty cluster for different reasons. The trivial ones are when the number of clusters exceed the number of the instances and when the first random centroids are not instances of the dataset and are too far from them. In our case the second circumstance is avoided choosing the first random centroids in the dataset, so that at least one instance is included in each cluster initially. Anyway, it's still possible to have an empty cluster how we can see in next example.

Suppose to have a dataset with eight instances whit these coordinates: $X_1 = (4,3)$, $X_2 = (8,3)$,$X_3 = (8,13)$, $X_4 = (9,13)$, $X_5 = (9,14)$, $X_6 = (10,13)$, $X_7 = (10,14)$ and $X_8 = (19,13)$ and that $k = 3$. Hypothesize that the first random centroids are: $C_1 = X_1$, $C_2 = X_2$ e $C_3 = X_8$ (2a). Thus the corresponding clusters are those showed in 2b. Now the algorithm calculates the new centroids for the three clusters: $C_1 = X_1$, $C_2 = (8,8)$ and $C_3 = (11.4, 13,4)$ (2c). At this point, how we can see in 2d, the centroid C_2 is no longer closer to any point than the others two, so the corresponding cluster is empty.



(a) Initial random centroids

(b) First clustering

(c) New centroids

(d) New clusters

Abbildung 2: Example of empty cluster

Problem 2.2 (C-Index)

a) You can see the code of k-means implementation here
C-index algorithm implementation:

```python
P = [(X[i], X[j]) for i in range(len(X)) for j in range(i +
    1,len(X))]
Q = [(X[i], X[j]) for i in range(len(X)) for j in range(i + 1,
    len(X)) if clusters[i] == clusters[j]]
Scl = sum([np.linalg.norm(np.array(i[0]) - np.array(i[1])) for i in
    Q])
q = len(Q)
P.sort(key=lambda x: np.linalg.norm(np.array(x[0]) -
    np.array(x[1])))
Smin = sum([np.linalg.norm(np.array(i[0]) - np.array(i[1])) for i
    in P[:q]])
Smax = sum([np.linalg.norm(np.array(i[0]) - np.array(i[1])) for i
    in P[(len(P) - q):]])
return (Scl - Smin) / (Smax - Smin)
```

b) Minimal and average C-index values versus k

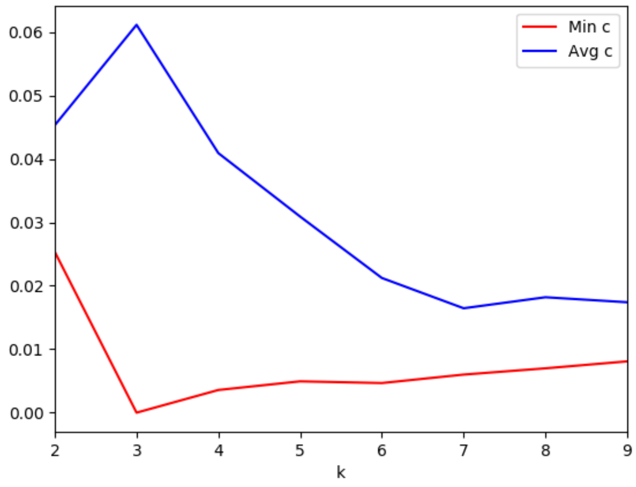


Abbildung 3: Minimal and average C-Index values versus k

c) The results shows clearly that we need at least 3 cluster to achive the best clustering.
Based on the minimal C-values the best choice is 3, but there the average is very high. The reason for this is clustering into 3 cluster is not always successfull like on the image 4.

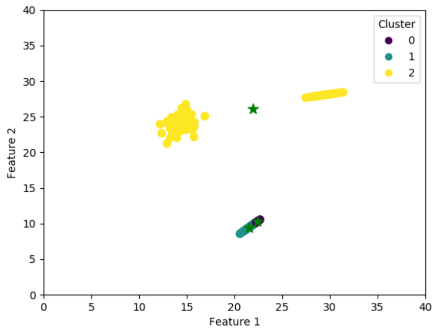


Abbildung 4: Bad clustering with k==3

This can occur when the starting centroid selection is not good. Although there is no empty cluster, the clustering is still bad. The C-index is very high: 0.1479
However clustering into 3 cluster can result the best clustering with the smalles C-index value: around 0.
Using more clusters will decrease the chance of a totally bad clustering (This is shown by the decreasing average curve), but the minimum C-index will increase, so our clustering will be not as good as the k = 3 scenario. This is also generally true, when we are using bigger k than the true cluster number.
In this concreate case 3 is the best solution.