Data Mining Summer Term 2020

Institute of Neural Information Processing
PD Dr. F. Schwenker

Assignment 4 (Submission until June 2, 2020)

Exercise 1 (3 points): k-means Clustering [Pen and Paper]

We learned in the last assignment how we can retrieve a certain number of partitions from the hierarchical clustering by undoing the last merges until we have k clusters. In this assignment, we are looking at k-means clustering where the data is directly partitioned into k clusters. This is especially useful if we are not interested in the hierarchical information but rather want to find k prototypes which are good representatives for our dataset.

More precisely, we are using Lloyd's algorithm from the k-means clustering family[1]. In order to get used to the basic concepts behind the algorithm, we are going to apply it first on a small dataset in a visual fashion. The implementation is postponed until Exercise 2. The two-dimensional dataset is shown in Figure 1 and the points are defined as

$$\boldsymbol{p}_1 = \begin{pmatrix}0\\0\end{pmatrix}, \quad \boldsymbol{p}_2 = \begin{pmatrix}1\\0\end{pmatrix}, \quad \boldsymbol{p}_3 = \begin{pmatrix}1.5\\0.8\end{pmatrix} \quad \text{and} \quad \boldsymbol{p}_4 = \begin{pmatrix}1.5\\1\end{pmatrix}. \tag{1}$$

Abstractly speaking, the k-means algorithm starts with an initial partition and iteratively improves it by re-calculating the cluster centres and re-assigning each point to its nearest cluster centre. There are several approaches to get the initial clusters. Here, we use $k = 2$ and simply define randomly generated points from our feature space as our initial clusters

$$\boldsymbol{c}_1(0) = \begin{pmatrix}1.35\\0.7\end{pmatrix} \quad \text{and} \quad \boldsymbol{c}_2(0) = \begin{pmatrix}1.6\\1.1\end{pmatrix}. \tag{2}$$

$\boldsymbol{c}_i(t)$ denotes the location of the i-th cluster centre in the iteration t.

1. Following this approach, the first step is to assign each data point to its nearest cluster centre. Perform this step by drawing a line for each data point $\boldsymbol{p}_i$ to one of the two cluster centres $\boldsymbol{c}_1$ or $\boldsymbol{c}_2$ in Figure 1. Since we do not want to spend to much time with distance calculations, it is sufficient to perform this task solely on the visual information from the plot.

2. We now have our first two clusters. The next step is to re-calculate the cluster centres $\boldsymbol{c}_i$ based on the data points which are assigned to them. Calculate $\boldsymbol{c}_1(1)$ and $\boldsymbol{c}_2(1)$ and draw both into Figure 2.

3. The algorithm now repeats the last two steps until a defined number of maximal steps $t_{\max}$ is reached or the cluster centres do not change anymore. Repeat the re-assigning and the re-calculation of the cluster centres until you think that the algorithm converged and explain why this is the case.

[1] Note that due to its popularity it is common to refer only to the k-means algorithm implicitly meaning that Lloyd's algorithm is used.

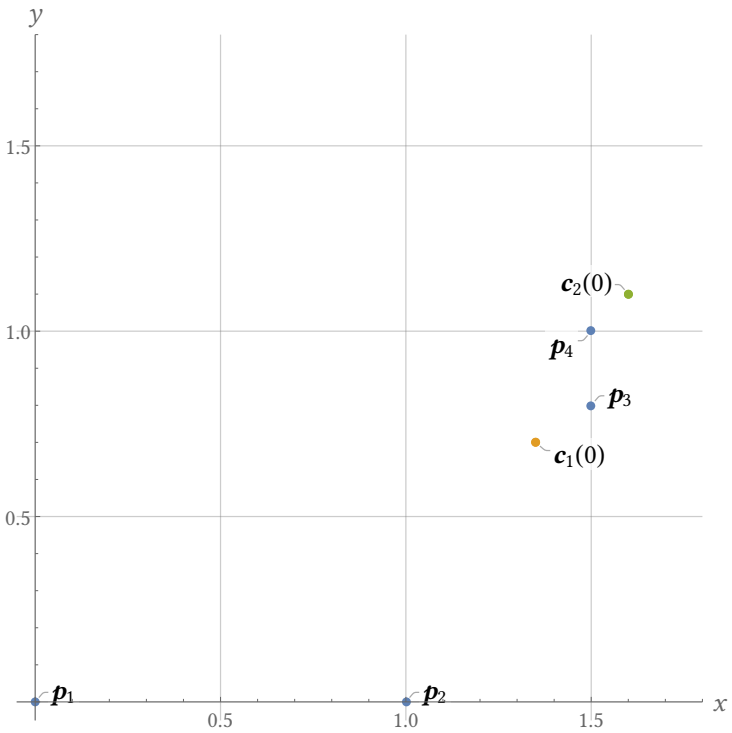

Figure 1: Data points (Equation 1) and initial cluster centres (Equation 2).

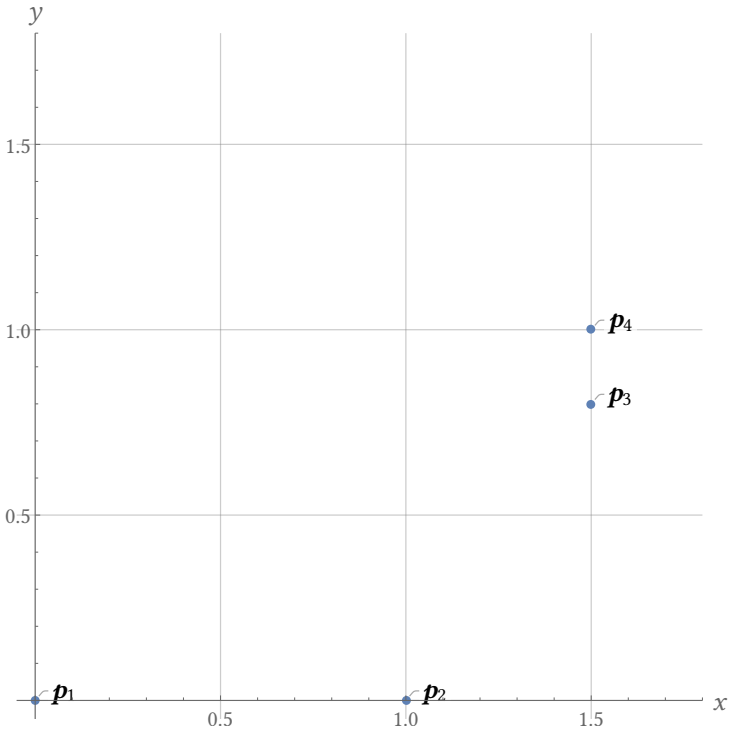

Figure 2: Re-print of the data points of Equation 1.

Exercise 2 (3 points): k-means Clustering [Python]

In this exercise, we want to implement the k-means algorithm and apply it on a larger two-dimensional dataset. We choose $k = 3$ and in order to get a numerical indicator of how good our clusters are, we evaluate the variance after each iteration. What is more, we like to see how the algorithm approaches a local minimum over iteration time. For this, we keep track of the prototypes, cluster assignments as well as the variance $D_{\text{var}}(t)$ and show the result with the help of the `ipywidgets` module.

We are using a slightly different initialization technique than before. Instead of generating random points in the complete feature space, we randomly select existing data points and elect them to be the first prototypes. This has the advantage that our cluster centres tend to be closer to the relevant range of our dataset.

Use the file `kMeans.ipynb` as a basis for your implementation which already loads the dataset, other initializations and code to plot the result. For the test script, please pay also attention to Table 1.

1. Implement a function `reassign(prototypes)` which finds the nearest neighbour for each data point and returns a list of cluster assignments (an index to the corresponding prototype for each data point). The class `NearestNeighbors` from the `sklearn.neighbors` package is helpful for this task.
2. Implement a function `recalculate(indices)` which calculates and returns new prototypes based on the cluster assignments (indices).
3. Implement a function `total_variance(indices, prototypes)` which calculates the variance $D_{\text{var}}(t)$ for one iteration (defined by the cluster assignments and the prototypes). The function `distance.cdist` from the `scipy.spatial` package can be used to calculate pairwise distances between two sets of points.
4. Begin with an initial iteration $t = 0$:
   a) Randomly select $k = 3$ prototypes from the data points. `np.random.randint` might come in useful here.
   b) Perform the first assignment based on the prototypes.
5. Now, run the k-means algorithm over 20 iterations $t = 1, 2, \ldots, 20$ (you don't need to implement any additional stop criterion). There are basically three steps in each iteration:
   a) Re-calculate the prototypes based on the points which are currently assigned to it.
   b) Re-assign each data point to its nearest cluster centre.
   c) Calculate the variance $D_{\text{var}}(t)$ based on the current clustering result.
6. Use the given visualization code to show your result. You can use the slider to control the iteration time t.

Table 1: Variable names for the notebook `kMeans.ipynb` (Exercise 2). You have to name your variables exactly as noted here so that the test script can check their content.

Name	Type	Description
`cluster_prototypes`	List	Cluster prototypes $\boldsymbol{c}_i(t)$ per iteration stored in a 3×2 matrix (rows denote prototypes, columns feature dimensions).
`cluster_indices`	List	Cluster assignments per iteration stored as an array of indices (0, 1 or 2).
`variances`	List	Variance values $D_{\text{var}}(t)$ per iteration.

Exercise 3 (4 points): *k*-means++ Clustering [Pen and Paper]

In the vanilla implementation of the *k*-means algorithm, we have the burden of initializing the *k* prototypes in the beginning. Starting with a random partition, random points in the feature space or *k* randomly selected data points are common approaches to tackle this problem. However, they all have the problem that the distribution of the data is not considered and it may be that the initialization is inefficient. This, in turn, can lead to a bad clustering result where we end up in a local minimum which is far apart from the global minimum.

Arthur and Vassilvitskii [1] proposed one possible solution to this problem by using a different initialization technique for the *k* prototypes (the rest of the *k*-means algorithm stays the same). The general idea is to select data points as prototypes which are far apart from previously selected clusters. This takes the distribution of the dataset into account by, roughly speaking, sampling prototypes where they are required. The authors give mathematical proof and show simulations that *k*-means++ outperforms vanilla *k*-means in terms of both speed and accuracy.

The goal of this exercise is to understand the problem of random initialization in the vanilla *k*-means algorithm and the solution proposed by Arthur and Vassilvitskii.

1. We want to use the online animation[2] by Naftali Harris implementing the vanilla *k*-means algorithm to understand the problem of random initialization. Select I'll Choose to manually pick the cluster centres and use the Gaussian Mixture dataset. Place two prototypes in the top point cloud and one prototype in one of the point clouds in the bottom. Then, run the algorithm. Note that you need to manually update the centroids and re-assign the points until convergence.

    a) What problem can you observe? Note: if you don't see a problem, choose different prototypes until you get a bad clustering result. Append a screenshot of your initial prototypes and the final result in your solution.

[2] https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

b) How is this problem related to a bad initialization of the cluster prototypes? Why do further iterations not help to overcome the problem?

2. We now want to explore the initialization procedure of the k-means++ algorithm based on the one-dimensional example data in Figure 3 which are defined as

$$p_1 = 1, \quad p_2 = 3, \quad p_3 = 4, \quad p_4 = 6 \quad \text{and} \quad p_5 = 10.$$

We want to find $k = 3$ prototypes.

a) For the first prototype, one data point is selected uniformly at random. Let's suppose that p_2 wins this race and hence we have our first prototype $c_1 = p_2$. The next step is to compute the distance for all points to its nearest cluster centre. Since we only have one, this reduces to $d_i = \|p_i - c_1\|^2$ (note that the squared Euclidean distance is used here). Compute the distance d_i for all points p_i.

b) For the second prototype c_2, we prefer data points which are further away from the existing cluster centre c_1. More precisely, we select the second prototype at random not uniformly but rather with a probability of ($n = 5$ here)

$$P_i = \frac{d_i}{\sum_{j=1}^{n} d_j}.$$

Calculate the probability P_i for each data point p_i and draw the discrete probability distribution (distance d_i on the x-axis and the probability P_i on the y-axis). Suppose that a random number generator which selects points p_i based on the weights P_i chooses $c_2 = p_5$ as the prototype for the second cluster.

c) We now repeat the previous procedure until we have the desired number of prototypes. That is, we first calculate the new distances of each data point to its nearest cluster centre

$$d_i = \min\{\|p_i - c_1\|^2, \|p_i - c_2\|^2, \ldots\}$$

and then select a new point at random with probability P_i.

i. Calculate the distance values d_i in this iteration.

ii. If we asked a number generator 14 times to pick a data point as third cluster centre c_3, how often would each data point p_i be chosen?

3. To see the k-means++ initialization in action for a two-dimensional dataset, watch this video[3] by Curtis Northcutt where the probability P_i is visualized via the size of the circles.

a) Explain why it is possible that for the third cluster centre not the point with the largest circle is chosen.

[3] https://www.youtube.com/watch?v=BIQDlmZDuf8

b) As an alternative of working with the probabilities P_i, we could also simply select the point with the highest distance d_i. Does this sound like a good idea to you? What might be an argument against this choice and work with probabilities instead?

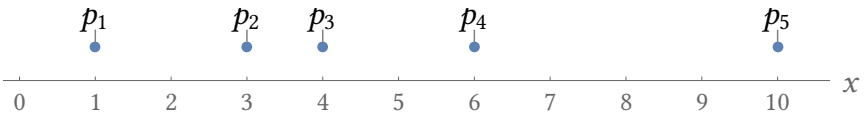

Figure 3: Example data used to explore the k-means++ algorithm.

References

[1] David Arthur and Sergei Vassilvitskii. "K-means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 978-0-898716-24-5. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.