Data Mining Summer Term 2020

Institute of Neural Information Processing
PD Dr. F. Schwenker

Assignment 8 (Submission until June 30, 2020)

Exercise 1 (5 points): Clustering of Words [Python]

Words may be a bit problematic at the first glance because they are nominal-scaled data with each word standing for itself without any meaningful distance metric. Luckily, a solution to this problem already exists and is called a *word embedding*. Each word of a vocabulary is embedded in a numerical vector from a continuous vector space which we can use to perform all the calculations we have used so far. The values in these vectors (one per word) do not have any specific meaning. However, the relationship between the vectors, e.g. the distance between two points, does have a meaning and relates to the semantic of the involved words. In this exercise, we are using word embeddings as an example and apply again some of our techniques learned previously.

How exactly the word embeddings are created is a topic on its own and we are not covering it here. Just to give you a general idea: the context of each word is analysed based on a large text base (e.g. Wikipedia) and from the neighbouring relationship between the words a vector representation is derived[1].

Another nice thing is that we don't have to create the word embeddings ourselves. Many pre-trained models in different languages exist which are ready to use. They mainly differ in the number of words in the corpus, vector dimension and the sources used for training. Here, we are playing with a small example corpus consisting of 9999 words embedded in 50-dimensional vectors.

Please use the file `ClusteringWords.ipynb` attached to this exercise as a basis for your implementation. It already loads the corpus and shows some basic usage. For the test script, please pay also attention to Table 1.

1. Apply k-means clustering (`sklearn.cluster.KMeans`) to all embeddings in the corpus. The tricky question here is how to select the number of clusters k. You can test different values for this parameter but since we want to visualize the clusters later, you should, for brevity, set it to a low number (use $k = 20$ for the test script).
2. We can't visualize our embedding space directly since the dimensionality is too high. Therefore, we need to project the vectors first into a lower-dimensioned feature space. Luckily, we already know about t-SNE which is very well suited for this task (`sklearn.manifold.TSNE`). Create two-dimensional projections for the 50-dimensional word embeddings. Note: even for this small dataset, the execution might take awhile.

[1] The article *Deep Learning, NLP, and Representations* by Christopher Olah is recommended for interested readers.

3. For a better visualization, it is nice to have a representative for each cluster, i.e. one word which describes the topic of the cluster. Go through each cluster and select the word which is located closest to the corresponding cluster centre. Store the word as well as the corresponding projections (the similarity is calculated in the feature space but for the visualization, we are only interested in the coordinates in the projection space). Hint: the embedding model has a `similar_by_vector` method which is helpful for this task.

4. It is now time to visualize our clustering result. Display the projections labelled with the corresponding word, colour each point based on the assigned cluster[2] and show the representative word for each cluster.

   If we displayed a label for each data point, we would get a very cluttered result as there are just too many words. Instead, a better idea is to include some kind of interaction and show the word labels only on mouse hover. However, this kind of interactivity is hard to achieve with matplotlib so we take the opportunity and use Plotly as visualization library. It is specially designed to create interactive charts.

   A possible result of what we want to achieve can be seen in Figure 1. You don't have to reproduce the exact same figure; it is sufficient when the clustering result becomes visible. If you like, you can also extend the figure with any functionality you want.

   a) Plot every point of each cluster and make sure that the words appear on mouse hover (`fig.add_scattergl`).

   b) Display the representative words permanently as annotations so that you can get a better overview of the clusters.

5. Explore the word embeddings and see if you can find something interesting. Note that you can zoom in to a specific area, hide a cluster (single click on the legend name) or show only points from one cluster (double click on the legend name).

   a) Explain why points from a cluster may be scattered around in the projection space even though they all belong to the same cluster.

   b) When a point is further away from the representative word in the projection space, does this necessarily mean that the same is true for points in the feature space?

6. We have seen that it isn't so clear how to choose the number of clusters k. Due to the high dimensionality of the dataset, we just don't know a-priori what a natural number for k may be. After all, it isn't even clear that the dataset has a specific number of k clusters. A possible workaround for this problem is to take a different point of view and analyse the hierarchy information by applying agglomerative clustering techniques.

[2] Just to be clear: we visualize the projected points but label them according to the clustering result obtained from the original feature space.

a) We first need a subset of the words since the whole list would be too much to display in any dendrogram. Select a representative word of your choice as a basis and then find 30 similar words for it (via the same similarity measure as used before). Keep the list of similar words as well as the corresponding embeddings.

b) Apply agglomerative clustering on the extracted set of embeddings by using the `linkage` function from the `scipy.cluster.hierarchy` package. Use complete-linkage as inter-cluster distance and set the `optimal_ordering` parameter to `True`.

c) Show your result in a dendrogram. Use the selected words as labels. Figure 2 shows the result based on the cluster "graduate".

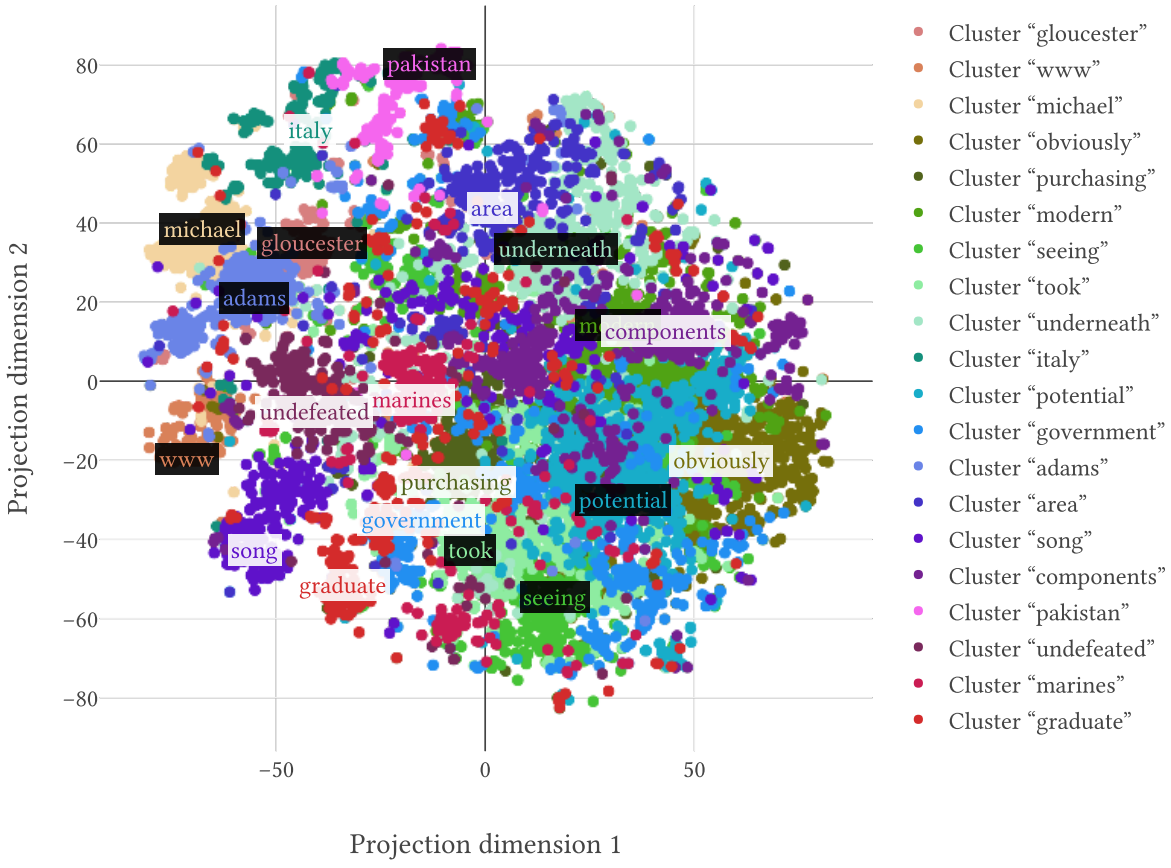

Figure 1: Clustering result of the word embeddings shown as t-SNE plot. The highlighted words are nearest to the respective cluster centroids.

Exercise 2 (5 points): Neural Gas [Pen and Paper]

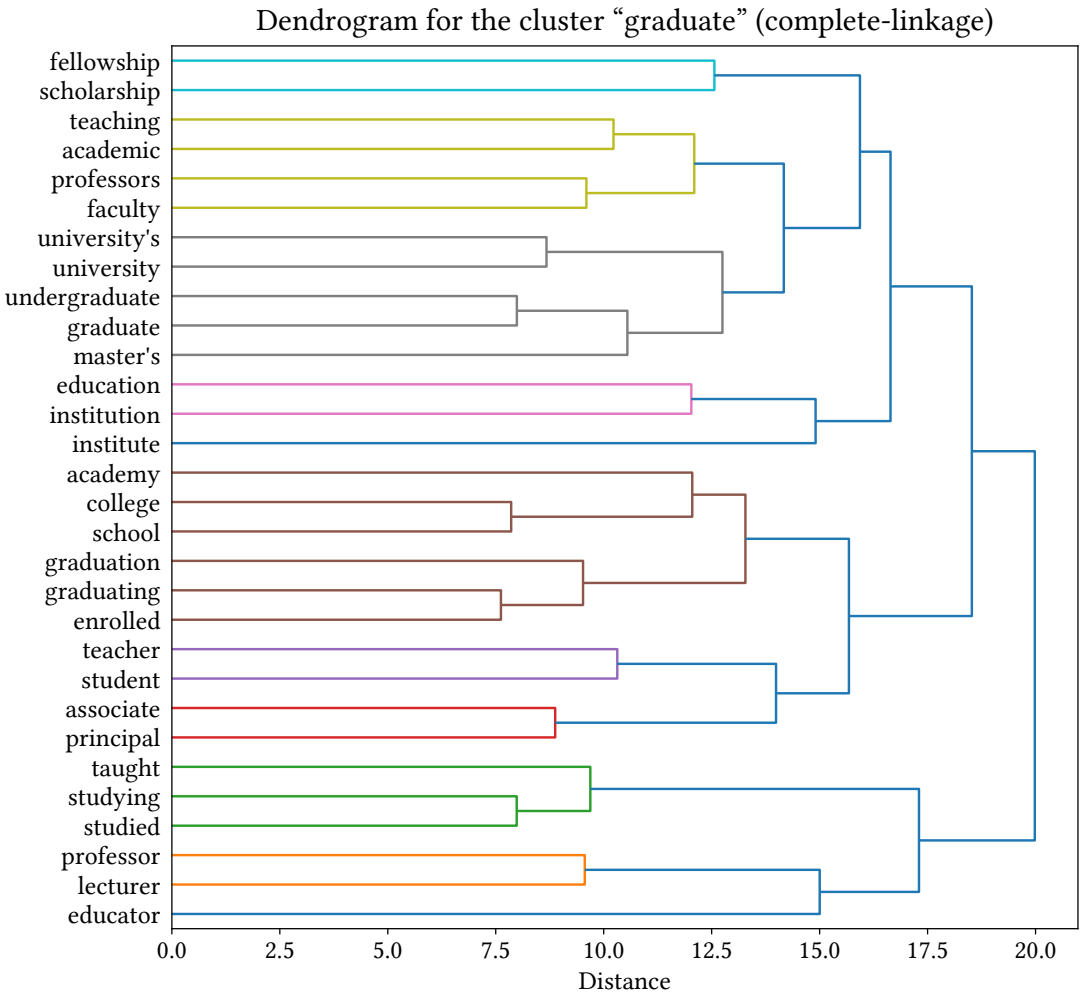

Figure 2: Hierarchical clustering result for similar words of a specific cluster.

Crisp and fuzzy k-means clustering are extremes where either only one data point has an influence on the nearest cluster prototype or all points influence all prototypes. Here, we want to introduce the neural gas algorithm [1] which works somewhere in-between. It is related to the family of competitive algorithms and does also work in an incremental fashion. However, not only the winning prototype gets adjusted but also more distant ones and the update process works differently compared to the fuzzy method. We also have to specify the number of clusters k in advance.

The key idea is to define a ranking across all cluster centres $\boldsymbol{c}_j$ based on the distances to the current data point $\boldsymbol{p}$. The ranks for each prototype $\boldsymbol{c}_j$ are in the range $r_j \in \{0, 1, \dots, k-1\}$ with the nearest cluster centre being on rank 0, the second nearest on rank 1 and so forth. As distance metric, we are using the squared Euclidean distance.

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

Name	Type	Description
`projections`	Matrix	The projections obtained from *t*-SNE.
`centroid_words`	List	Representative words for each cluster. Each index corresponds to a cluster label.
`centroid_projections`	List	Corresponding projections for the representative words.
`similar_words`	List	Similar words for the selected cluster.
`similar_vectors`	Matrix	Corresponding embeddings for the similar words.
`tree`	Matrix	Hierarchical information obtained when applying agglomerative clustering on the example cluster (similar words).

Based on the ranks $r_j(t)$ of the iteration t, the incremental update rule for all cluster prototypes $\boldsymbol{c}_j(t)$ and an arbitrarily chosen data point $\boldsymbol{p}$ is defined as

$$\boldsymbol{c}_j(t) = \boldsymbol{c}_j(t-1) + \eta(t-1) \cdot e^{-\frac{r_j(t-1)}{\lambda(t-1)}} \cdot \Big(\boldsymbol{p} - \boldsymbol{c}_j(t-1)\Big), \quad j = 1, 2, \ldots, k; t = 1, 2, \ldots, t_{\max}. \tag{1}$$

Note that in each step every prototype is updated. We repeat applying this rule until a defined number of $t_{\max}$ iterations while always randomly selecting a data point $\boldsymbol{p}$. It is also possible to select the data points $\boldsymbol{p}$ in a round-robin fashion and this happens to be the procedure we want to use here.

$\eta(t)$ are learning rates which decay to zero, i.e. the updates get smaller and smaller with increasing iteration time t. With η_{start} and η_{end} we can construct a sequence of learning rates as

$$\eta(t) = \eta_{\text{start}} \cdot \left(\frac{\eta_{\text{end}}}{\eta_{\text{start}}}\right)^{\frac{t}{t_{\max}}}. \tag{2}$$

This sequence begins with $\eta(0) = \eta_{\text{start}}$ and ends with $\eta(t_{\max}) = \eta_{\text{end}}$ (hence the name of the variables).

The sequence $\lambda(t)$ controls the extent to which higher-ranked clusters get updated in the iteration t. To put it differently, it controls how much influence a point $\boldsymbol{p}$ has on its neighbouring cluster centres $\boldsymbol{c}_j$ whereby the neighbourhood is only defined by the rank r_j. Similar to $\eta(t)$, it decays to zero, i.e. the influence of a point $\boldsymbol{p}$ decreases with increasing iterations, and is defined with the parameters λ_{start} and λ_{end} as

$$\lambda(t) = \lambda_{\text{start}} \cdot \left(\frac{\lambda_{\text{end}}}{\lambda_{\text{start}}}\right)^{\frac{t}{t_{\max}}}. \tag{3}$$

As with previous algorithms, we initialize the prototypes $\boldsymbol{c}_j$ at random in our feature space. It is possible that the corresponding error surface contains a lot of (false) local minima and we might end up in a non-optimal solution due to the poor initialization. This is an issue we also saw in the clustering algorithms of the k-means family. The idea of the decaying parameter $\lambda(t)$ now is that in the beginning many prototypes get updated and that this provides a way of jumping over the local minima. After traversing this dangerous first part, we can fine-tune the result by adjusting fewer and fewer prototypes at a time.

Here, we want to learn in rates from $\eta_{\text{start}} = 1$ to $\eta_{\text{end}} = 0.001$ and begin with a neighbourhood influence of $\lambda_{\text{start}} = 2$ until we reach $\lambda_{\text{end}} = 0.01$ in $t_{\text{max}} = 200$ iterations. Figure 3 shows a plot of both sequences.

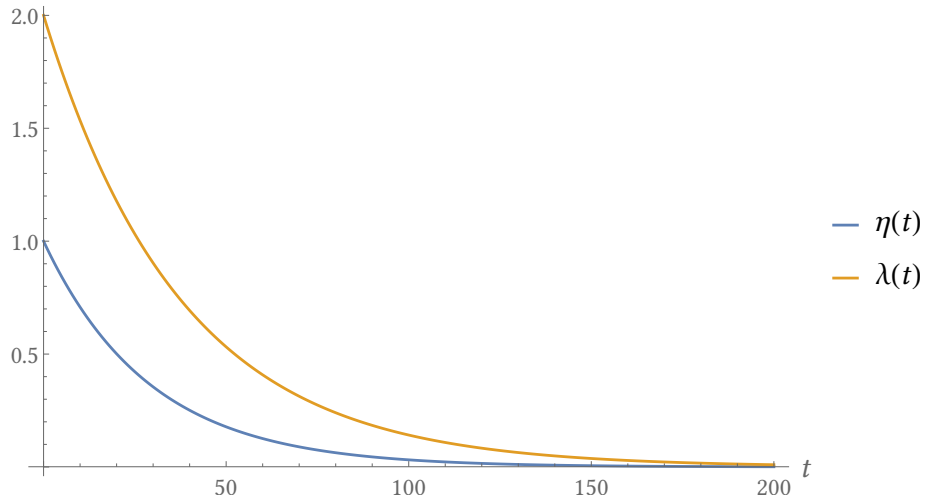

Figure 3: Decay sequence of the learning rate $\eta(t)$ (Equation 2) and the neighbourhood influence $\lambda(t)$ (Equation 3).

We are using the same two-dimensional dataset as we used in the fuzzy k-means exercise

$$\boldsymbol{p}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{p}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \boldsymbol{p}_3 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{p}_4 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}. \tag{4}$$

The $k = 2$ initial cluster centres are defined as

$$\boldsymbol{c}_1(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{c}_2(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{5}$$

This starting situation is also shown in Figure 4. We present the data points in the order as defined by Equation 4, i.e. $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_1, \ldots$

1. Let's apply the update rule of Equation 1 to the data point $\boldsymbol{p}_1$.

    a) We first need to find the current ranking $r_j(0)$ of the initial situation $t = 0$. Determine the rank $r_1(0)$ for the first $\boldsymbol{c}_1(0)$ and the rank $r_2(0)$ for the second prototype $\boldsymbol{c}_2(0)$. It is sufficient to perform this task solely on a visual basis.

b) Calculate the values for the two parameters $\eta(0)$ and $\lambda(0)$.

c) Calculate the weight $e^{-r_j(0)/\lambda(0)}$ of the neighbourhood influence for both prototypes.

d) Now, we have all the ingredients together to perform the update step of Equation 1. Calculate $\boldsymbol{c}_1(1)$ as well as $\boldsymbol{c}_2(1)$ and draw both into Figure 4

2. Perform another iteration by calculating $\boldsymbol{c}_1(2)$ and $\boldsymbol{c}_2(2)$ while using the data point $\boldsymbol{p}_2$. Again, draw the result into Figure 4. The final result after $t_{\max}$ steps can be seen in Figure 5.

3. The neighbourhood influence $\lambda(t)$ is an important parameter of the neural gas algorithm. It is useful to explore its meaning a bit.

   a) We first want to know how the weight $e^{-r_j(t-1)/\lambda(t-1)}$ behaves for the sequence $\lambda(t)$. This is shown in Figure 6 for different rank values. Sketch how the curve for the rank $r = 0$ looks like.

   b) The curves shown in Figure 6 depend also on the starting value which we set to $\lambda_{\text{start}} = 2$ here. Sketch how the curve for the rank value $r = 1$ looks like if we use a higher starting value $\lambda_{\text{start}} > 2$.

   c) Let's assume for a moment that the factor $\lambda(t)$ does not depend on t anymore and is fixed instead.

      i. What happens if we set λ to a very small value? Compute the limit

      $$\lim_{\lambda \to 0+} e^{-\frac{r}{\lambda}}, \quad r = 0, 1, 2, 3$$

      and explain to which algorithm Equation 1 reduces to in this case. Hint: maybe you can find a formula in the script which looks similar.

      ii. The other extreme is to set λ to a very large value. To see what happens then, compute the limit

      $$\lim_{\lambda \to \infty} e^{-\frac{r}{\lambda}}, \quad r = 0, 1, 2, 3$$

      and explain what this choice implies. Is this a useful setting?

4. Explain why it is reasonable that the learning rate $\eta(t)$ decays to zero.

5. Figure 7 shows a new situation with one data point $\boldsymbol{p}_1$ and three cluster prototypes $\boldsymbol{c}_j$. If we apply Equation 1 to this situation, we get a change $\Delta\boldsymbol{c}_j$ for each prototype which denotes the movement towards $\boldsymbol{p}_1$. Mark the area where we could move the second prototype $\boldsymbol{c}_2$ (before applying Equation 1) so that the change for the other prototypes $\Delta\boldsymbol{c}_{j\neq 2}$ remains the same (no calculations required).

References

[1] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "'Neural-gas' network for vector quantization and its application to time-series prediction". In: *IEEE Transactions on Neural Networks* 4.4 (July 1993), pp. 558–569. ISSN: 1045-9227. DOI: 10.1109/72.238311.

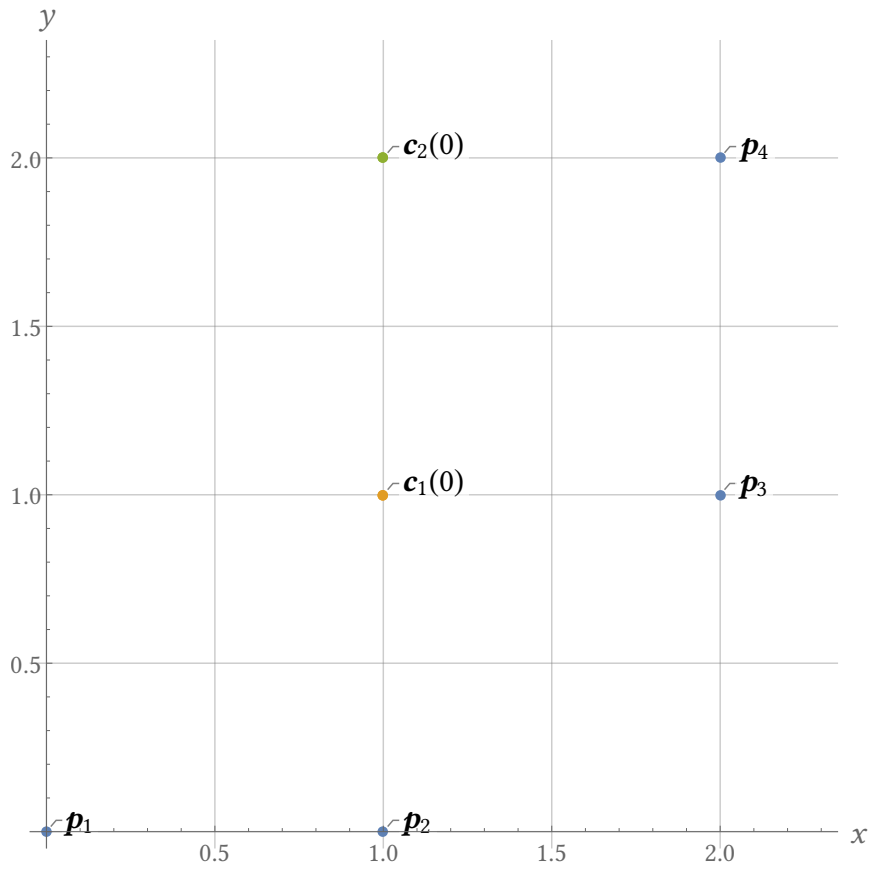

Figure 4: Starting situation of the example dataset used in Exercise 2. Shown are the data points from Equation 4 and the initial clusters from Equation 5.

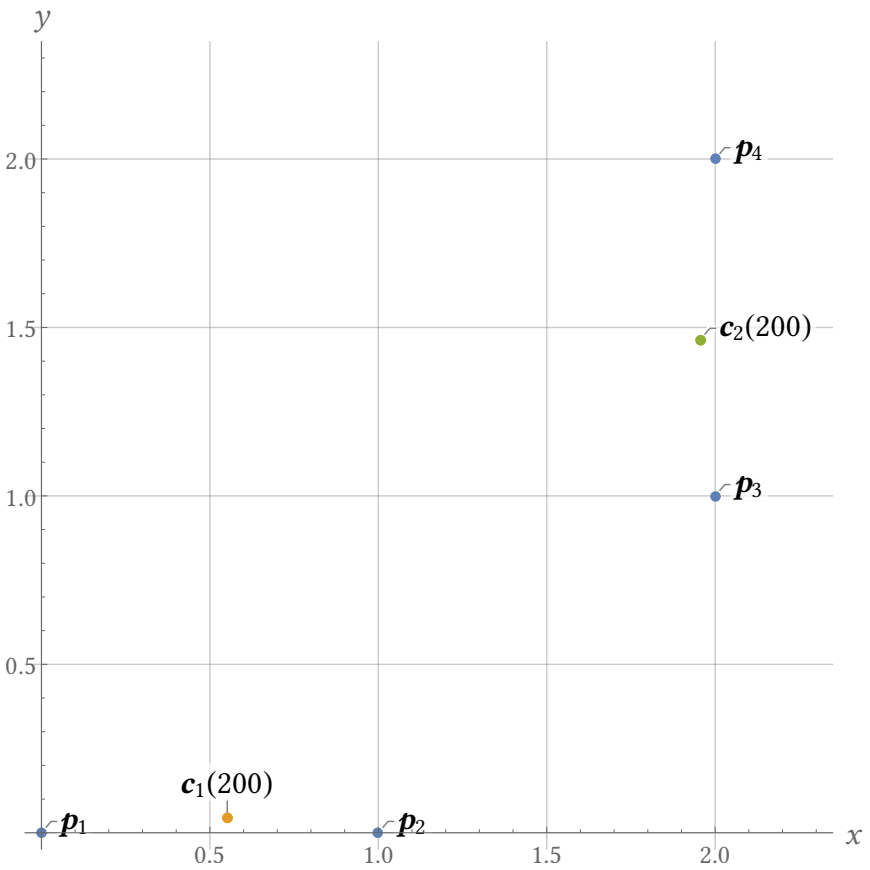

Figure 5: Result of the neural gas algorithm after $t_{\max} = 200$ iterations.

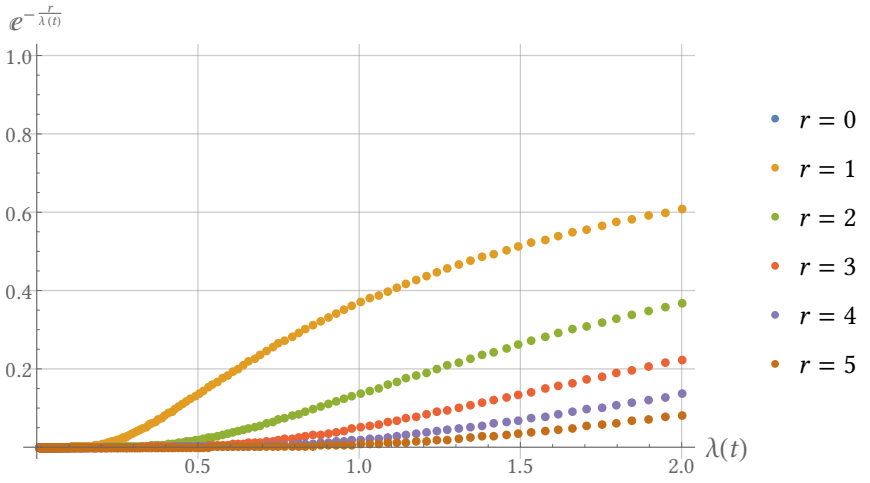

Figure 6: Plot of the weights $e^{-r_j(t-1)/\lambda(t-1)}$ for the neighbourhood influence sequence $\lambda(t)$. For comparison, parametric curves for different ranks r are shown (even more than we use here). The curve for $r = 0$ is missing.

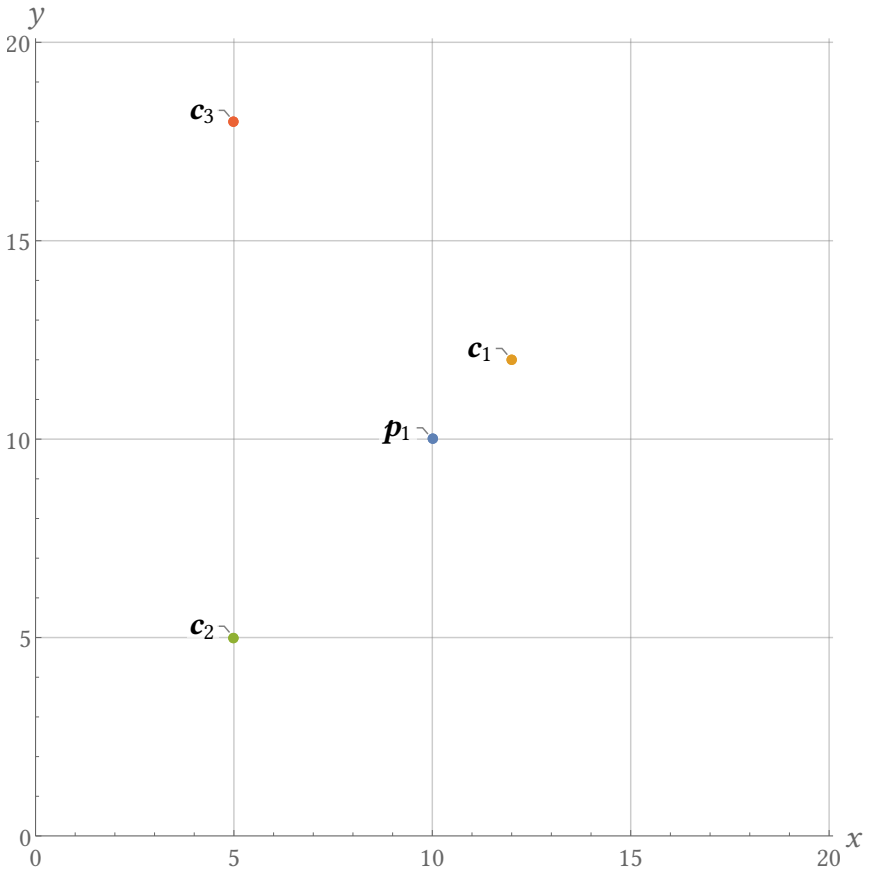

Figure 7: New situation with one data point $\boldsymbol{p}_1$ and three clusters prototypes $\boldsymbol{c}_j$.