CLUSTERING OF THE SELF ORGANIZING MAP - SEMINAR REPORT

Halbig Roland

University of Erlangen-Nürnberg, Germany
roland.halbig@math.stud.uni-erlangen.de

ABSTRACT

The aim of this article was to verify the results presented in [1]. In respect to the 'two-level-approach' using k-means, those results could not be verified. Instead, the complexity considerations of the Algorithm could be refined and we show that it should be slower than it is proposed in [1]. A discussion of the difficulties that arise if one wants to measure the quality of a Self Organizing Map concludes this work.

Index Terms— Self Organizing Map, Algorithm, Implementation, Computational Complexity, XOR-dataset

1. INTRODUCTION

Classification in high-dimensional feature spaces can be very time consuming (c.f. curse of dimensionality). The idea of a SOM is to reduce this dimensionality by choosing lower-dimensional representations such that the distribution of the high-dimensional data is mapped. One could say that a SOM is a approximation of the real data's topological distribution. Ideally the SOM can be thought as a topology-preserving mapping.

The most common SOM is the two-dimensional map where the representative nodes are ordered in a rectangular grid. This is mainly due to implementational reasons, as well as it is usefull for visualization of high-dimensional data. However, also three-dimensional SOMS can be used in real applications (c.f. [2]).

In this work we will restrict ourself to the 2D-case.

This paper is ordered as follows: First, a simple SOM-algorithm is presented as pseudo-code. Then, implementation details will be discussed and possible alternatives will be named considering some adaptions for improving fitting performance and convergance in general.

As a next step, we will analyze the computational complexity and critically discuss the advantage of the so-called 'two-level-approach' presented in [1] for k-means clustering.

Finally, some results will be presented which highlighten some difficulties which arise when one wants to evaluate the quality of a SOM.

1.1. Self Organizing Maps

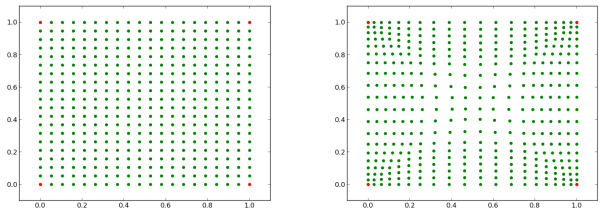
Definition: SOM

Let $\mathcal{I} \subset \mathbb{N}$ be a finite index set. A **Self Organizing Map** (SOM) is defined as a set

$$SOM_{\mathcal{I}} = \{(N_\lambda, r_\lambda) \in R^m \times \mathbb{R}^d \,|\, \lambda \in \mathcal{I}\}$$

of prototype vectors $N_\lambda \in \mathbb{R}^m$ and their position $r_\lambda \in \mathbb{R}^d$ in a grid.

Fig. 1. The XOR-dataset



1.2. Algorithm

For this algorithm it is assumed, that SOM consists of a rectangular grid, i.e. $\mathcal{L} = \{(i,j) \in I \times J\}$.

1. Initialize the prototype vectors N_λ of the SOM, set the counter $t = 0$
2. Iterate steps 2-6 while $t < tmax$
3. Choose a sample $X_s \in \mathbb{R}^m$
4. Calculate the **Best Matching Unit** N_b with $b = \arg\min_{\lambda \in \mathcal{I}} \left\{ \|X_s - N_\lambda\|_{L^2(\mathbb{R}^m)} \right\}$
5. Adjust all nodes N_λ in the proximity of N_b: $N_\lambda^{t+1} = N_\lambda^t + \alpha(t) h_{b,\lambda}(t) \left(X_s - N_\lambda^t\right)$
6. Set $t = t + 1$ and adjust both **kernel radius** σ and **learning rate** α.

The functions above are defined as follows:

$$h_{b,\lambda}(t) = \exp\left[-\frac{\|r_b - r_\lambda\|_{L^2(\mathbb{R}^d)}}{2\sigma^2(t)}\right]$$

$$\sigma(t+1) = \sigma_0 \exp\left[-\frac{t}{t_{max}} \log\left(\sigma(t)+1\right)\right]$$

$$\alpha(t+1) = \alpha_0 \exp\left[-\frac{t}{t_{max}} \log\left(\alpha(t)+1\right)\right]$$

In figure 1 the influence of the gaussiann kernel radius after one iteration over all four samples is shown with initial σ_0 being half the grid's size.

2. IMPLEMENTATION

The implementation was done in python on basis of an old version of the code by Kyle Dickerson ([3]). The main effort was to keep the code simple an adjustable to custom needs. For this sake the SOM's nodes are not implemented as a 2D-array containing the m

coordinates of the prototype vectors, but as a 1D-Array containing this information. In order to use this data structure as if it was a 2D grid, the 'getIndex' method is used. Vice versa, another 1D-array is used to save the index pair within the 2D grid.

The advantage of this approach is that it is very easy to adjust the SOM to a hexagonal or a 3D grid, for example. One only has to change the 'getIndex' function, the initialization and the getNeighborhood function. On the other hand, finding the best matching unit can be done very efficiently using numpy.sum and numpy.argmin.

Initialization can easily be done using random values in about the range of the sample data. As described in [1], for better convergence in the beginning of the training, principal component analysis is used. The two main components are identified and the prototype vectors are initialized equidistantly along the hyperplane they span.

Due to the implementation of the grid, as described above, the neighborhood of the best matching unit can easily be gotten. One only needs to find the indices of the neighboring nodes in the grid. Also the removal of nodes from the result data which is also mentioned in [1] is easily done using an index dicionary which saves all available indices. By returning only those nodes, which are named in the index dicitonary, the data structure of numpy array.

Another very good way to improve the quality of the SOM is, in each iteration step, to repeat the node adjustment for more than one sample. This is equivalent to the choosing stepwise decreasing adaption functions. As quality increases, however, performance suffers a lot. If k samples are chosen in each iteration step, the algorithm will take k-times as long. This approach can also be applied with the training of the already trained SOM (c.f. [1]). Beginning with a rough training phase choosing many samples in each iteration, and continuing with a much smaller radius and less samples.

It should be noted that the choice of decay functions for the kernel radius and the learning rate are not characteristic for a SOM, but it is rather a parameter. For some data, linear decay is more performative, for other exponentional decay is. Using more than one sample in each iteration, as described above, can also be viewed as using a piecewise constant decay function on one sample in each iteration. Also multiple training can be viewed in this light: It is merely the piecewise definition of the two decaying functions of two iteration epochs.

2.1. Computational Complexity

The computational complexity depends on the number of nodes $|\mathcal{I}| = N^2$, the number of iterations t_{max} and the number of samples l that are evaluated in each iteration step, the dimension m, the kernel function h, the initial kernel radius σ_0 and the choice of the kernel decrease $\sigma(t)$.

The number of nodes plays a fundamental role when one is computing the best matching unit and the update rule.

If the kernel function is only applied to nodes within the square of width $\sigma(t) := \sigma_0 e^{-\frac{t}{t_{max}}}$, using $\sigma_0 = \frac{N}{3}$ we get a complexity of

$$\begin{aligned}\mathcal{O}\left(l\,m\,|\mathcal{I}|\sum_{t=1}^{t_max}\sigma(t)^2\right) &= \mathcal{O}\left(l\,m\,|\mathcal{I}|\,\sigma_0^2\sum_{t=1}^{t_{max}} e^{-\frac{2t}{t_{max}}}\right)\\ &= \mathcal{O}\left(l\,m\,|\mathcal{I}|\,\sigma_0^2\left[\frac{1}{2}t_{max}(1-e^{-1})\right]^2\right)\\ &= \mathcal{O}\left(\frac{l}{18}\,m\,N^4\,t_{max}\right)\end{aligned}$$

It is important to note that the complexity of the som does not explicitly depend on the number of sample vectors M. However, concerning the randomization in step 3 of the algorithm, one should choose t_max such that it is at least as large as $\frac{M}{l}$ since otherwise some of the samples will not be considered. Indeed, in order to achieve good results it should be considerably larger that this. Therfore, we get a effective complexity of $\mathcal{O}\left(m\,M\,N^4\right)$.

If we compare this result to [1], section II.C, we can say when it will be useful to use the SOM as a preprocessing step for the k-means clustering. et al.Vesanto [1] state the computational complexity of a k-means is $\mathcal{O}\left(M\,\sum_{k=2}^{C_{max}} k\right)$. Using SOM, this complexity would reduce to $\mathcal{O}\left(M\,N^4 + N\,\sum_{k=2}^{C_{max}} k\right)$, if we ignore the dimension's influence. It is to note that this complexity is reasonably higher than $\mathcal{O}\left(M\,N + N\,\sum_{k=2}^{C_{max}} k\right)$ which is given in [1].

Considering complexity results in [4], however, the reduction is indeed more remarkable: Ignoring the variance term, $\mathcal{O}\left(M^{34}k^{34}m^8\log^4(M)\right)$ reduces to $\mathcal{O}\left(m\,M\,N^4 + N^{34}k^{34}\log^4(N)\right)$.

Finally, note that if a 1D-SOM is used with N^2 nodes, the SOM's complexity reduces even further to $\mathcal{O}\left(m\,M\,N^3\right)$.

3. EVALUATION

The main problem evaluating the SOM is that it is a preprocessing step which does not give us the clustering of the data, but a approximation of it. Therefore there doesn't exist an effective measure whether a SOM is good or not.

The potential function $E(t) = \sum_k^N \sum_{\lambda\in\mathcal{I}} h_{\lambda^*,\lambda}(t)\,\|X_k - N_\lambda\|^2_{L^2(\mathbb{R}^m)}$ with the best matching index $\lambda^* = \arg\min_{\lambda\in\mathcal{I}}\left\{\|X_k - N_\lambda\|^2_{L^2(\mathbb{R}^m)}\right\}$ is mimimized by the SOM algorithm (c.f. [5]). It can also be viewed as a gradient descent of the sample function $E_s(t) = \sum_{\lambda\in\mathcal{I}} h_{\lambda^*,\lambda}(t)\,\|X_k - N_\lambda\|^2_{L^2(\mathbb{R}^m)}$.

It is easy to see as $\lim_{t\to t_{max}} h_{\lambda^*,\lambda}(t) = \delta_{\lambda^*,\lambda}$ with dirac delta δ, that $\lim_{t\to t_{max}} E_s(t) = \|X_k - N_{\lambda^*}\|^2_{L^2(\mathbb{R}^m)}$.

Summed over all samples, this is equivalent to the residual $E = \sum_k^N \min_{\lambda\in\mathcal{I}} \|X_k - N_\lambda\|^2_{L^2(\mathbb{R}^m)}$ proposed in [6].

Note also the similarity to the residual of the k-means $E_K = \sum_{j=1}\sum_{i\in S_j}\|x_i - \mu_j\|^2$ where S_j are all nodes which are closer to μ_j than to all other μ (c.f.[7]). [5] explains the realtionship as follows: '[The] difference is that in the SOM the distance of each input from the reference vectors instead of just the closest one is taken into accoung, weighted by the neighborhood kernel h. Thus the SOM functions as a converntional clustering algorithm if the width of the neighborhood kernel is zero.'([5], p.25.)

4. RESULTS

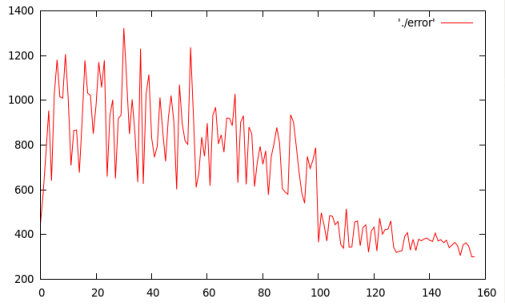
The problem with optaining results was that we could not decide wether a SOM was actually good, or not. Despite the discussion above, measuring the error actually is very inaccurate and the resiudal oscillates a lot.

AS can be seen in figure 2, the error also does not necessarily converge to zero in one iteration epoch.

5. CONCLUSION

Considering the difficulties we had describing the quality of the Self Organizing Map, we are sceptical wether it is a good tool for cluster analysis. Statistical tools, such as Gaussian Mixture Models can be far more useful.

Fig. 2. Error plot for the XOR-dataset



However, those models, as well as k-means, do get in trouble in very hihj dimensions. This is the core strength of the SOM, that the dimension does not play a large role for it. It can also give a first idea what is happening in the data set.

In this work, only data sets of a couple of thousand samples were taken into considerations because this was the scale applied in [1]. For larger datasets the results can be better.

Also, the complexity statements in [1] could not be verified, instead we gave a complexity measure based on our implementation which implies a much slower performance.

6. REFERENCES

[1] Esa Alhoniemi Juha Vesanto, "Clustering of the self-organizing map," 2000.

[2] Jorge M. L. Gorricha and Victor J. A. S. Lobo, "On the use of three-dimensional self-organizing maps for visualizing clusters in geo-referenced data," Year ?

[3] Kyle Dickerson, "sompy.py," 2008.

[4] Arthur D.; Manthey B.; Roeglin H., "k-means has polynomial smoothed complexity," 2009.

[5] Samuel Kaski, "Data exploration using self-organizing maps," 1997.

[6] A. Sperduti J. J. Steil, "Indices to evaluate self-organizing maps for structures," 2009.

[7] Edo Liberty, "k-means clustering," .