Examining the Behaviour of the Evolving Tree

Jussi Pakkanen
Laboratory of Computer and Information Science
Helsinki University of Technology
Finland
jussi.pakkanen@hut.fi

Abstract - *We analyze the Evolving Tree, which is a tree-shaped hierarchical neural network. Especially we try to shed light on its growth process and performance when compared to classical methods. Several visualizations done with Sammon's mapping show that while the Evolving Tree describes the data set very differently its performance is comparable to the classical methods.*

Key words - hierarchical clustering, tree-shaped neural networks, Evolving Tree, defect images

1 Introduction

The self-organizing map (SOM) is a very popular data analysis method [4]. Like all systems it has some features which are nonoptimal for certain applications. These include the predefined size of the grid and also relatively slow training times when dealing with large maps.
We have developed a new neural system which attempts to solve these drawbacks. It is called the Evolving Tree (ETree) [6]. Its main design principles have been simplicity and scalability to very large problems. In this paper be examine its behaviour with large data sets and compare it to several classical algorithms.

2 The Evolving Tree

In this section we briefly describe the ETree algorithm. We refer the readers interested in details on the basic algorithm to [6]. We also describe a new method for controlling the complexity of the system.
The basic building blocks of ETree are the same as in the SOM. We have nodes which have prototype vectors. We also use the same training formulas as the SOM. The difference is that we use these blocks in a very different way. The basic idea is to use a tree topology as opposed to the grid structure of SOM. The leaf nodes are used for data analysis while the trunk nodes maintain an efficient search tree to the leaf nodes. This makes SOM's most time consuming operation, finding the best matching unit (BMU), very fast. This is illustrated in the left image in Figure 1.
Another important part of SOM is the neighborhood. It measures the distance between two nodes along the grid. We use an equivalent metric called the *tree distance*. It computes

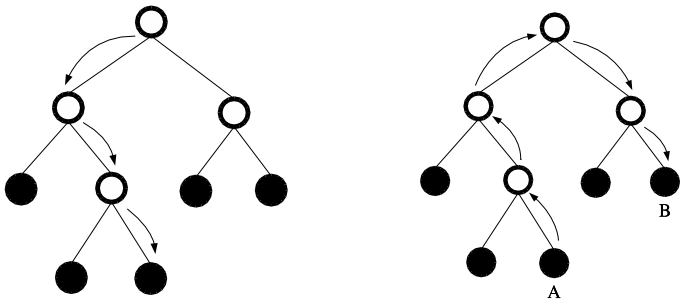

Figure 1: Fundamental operations of the Evolving Tree. The left image shows how the BMU is found. The right image demonstrates how the tree distance between two nodes is calculated.

the distance between two leaf nodes along the the search tree, as can be seen in the second image in Figure 1. With these we can train the tree structure in the same way as in the SOM. The last missing piece for a working system is a method for growing the tree. This is done by counting how many times each node has been the BMU. When this value reaches a pre-specified threshold, we split the node. The leaf node is transformed into a trunk node by giving it some child nodes.
This training is usually performed for some amount of epochs and then stopped. Selecting the epoch count beforehand is very difficult, and usually leads to nonoptimal trees. To get around this, we used an algorithm based on weight decay [1]. As mentioned above, every node contains a counter, which says how many times it has been the BMU. After every epoch these counters are multiplied by a constant that is less than one. This inhibits the growth by decreasing the amount of splits. When we discover that the tree has grown only very little, such as under 5%, in one epoch we stop the training.

3 Experiments

We have used two different kinds of data sets for our experiments. The first data set consists of three different MPEG-7 features [5]:

Homogeneous texture descriptor filters the image with a bank of orientation and scale tuned filters that are modeled using Gabor functions. The feature vectors have 62 elements.

Edge histogram calculates the amount of vertical, horizontal, 45 degree, 135 degree and non-directional edges in 16 sub-images of the picture. The resulting feature vectors have 80 elements.

Color structure slides a structuring element over the image, the numbers of positions where the element contains each particular color is recorded and used as a descriptor. Its vectors have 32 elements.

These were calculated for approximately 1300 paper surface defect images [7], which is a moderate size for analysis. This data set represents actual industrial data. There are a total

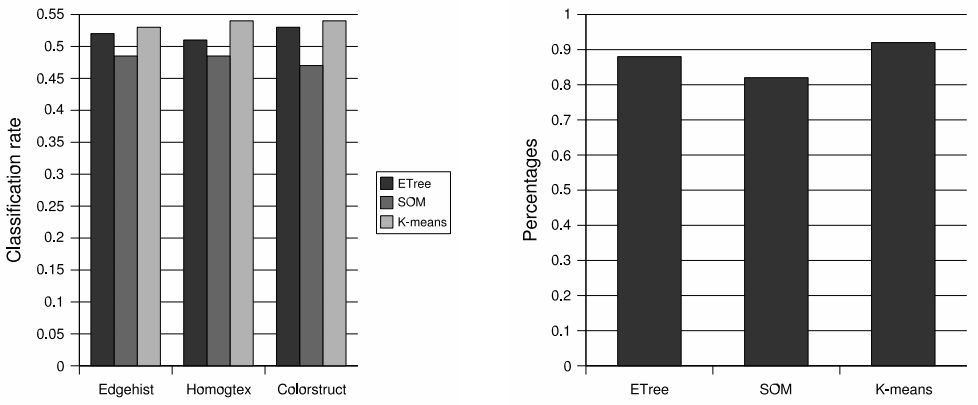

Figure 2: Classification rates for ETree, SOM and k-means for paper defect images (left) and handwritten digits (right).

of 14 different classes which are fuzzy and overlapping and therefore extremely difficult to classify. Most classes had 100 images, except for one with 70 and another one with 30 images. The other data set consists of 18 000 handwritten digits. The digit images were normalized to 32×32 pixels with 256 gray scale values. 38 principal components were obtained using PCA and used for our experiments [3]. This is a relatively large classified database. Comparing results to the MPEG-7 feature databases shows how different algorithms scale to larger problems.

The tested algorithms are ETree, SOM and k-means clustering. They are all based on similar algorithmic steps. SOM and k-means are well established methods and therefore form a good basis for comparison. The sizes of the SOMs were automatically selected by the SOM Toolbox[1], which was used for the experiments. An identical amount of clusters was used for k-means. The amount of nodes in ETree cannot be exactly defined, but in almost every case ETree had slightly fewer nodes than SOM and k-means.

3.1 Classification results

First we examined how the three different algorithms perform in a straightforward classification task. This was done by deciding class labels for the nodes with majority voting. The results can be seen in Figure 2. For paper defects the classification rates are quite similar for this difficult data set. SOM is worst for every feature. ETree comes second while being very close to the performance levels of k-means clustering.

For the handwritten digits the overall results are the same. The relative performance difference of ETree and k-means is slightly larger than in the paper databases. SOM performs clearly worst of the three.

The overall result of these classification experiments is that ETree's performance is comparable to similar classical methods. It does not quite match k-means on performance, but it runs a lot faster. In these experiments the training time for ETree was only about one tenth of the other methods.

[1] `http://www.cis.hut.fi/projects/somtoolbox/`

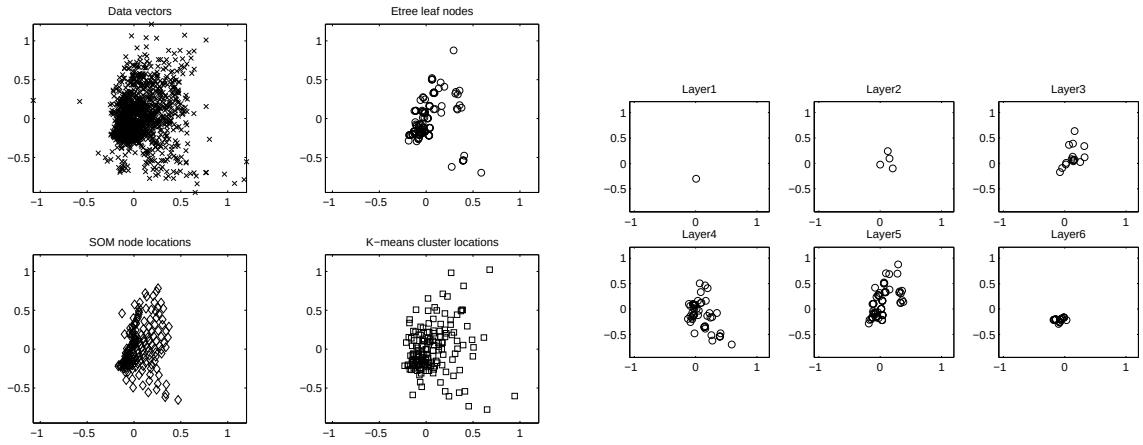

Figure 3: Data and node vector locations obtained with Sammon's mapping. The second shows the locations of ETree's nodes per layer.

3.2 Visualization experiments

To obtain further insight into the behaviour of different algorithms, we did visualization experiments. Since the Evolving Tree is designed for large dimensional data, we cannot plot them directly. We took the data and projected it to two dimensions using Sammon's mapping [8]. This preserves the overall shape of the data cloud by preserving the distances between vectors. Sammon's mapping is nonlinear and lossy, so the images it produces are not 100% accurate, but they are quite suitable for qualitative data analysis.

These experiments are only done on the paper defect data. The reason is the very large size of the digit database. Squeezing 13 000 data vectors into a relatively small figure produces results that are at best murky and at worst unreadable. Since these images would yield very little insight to the data, we have concentrated on the smaller data set.

The first picture in Figure 3 shows the locations of data vectors, ETree and SOM nodes, and k-means cluster centers. We have used the homogeneous texture data set which has 1300 vectors. Almost all of them are in the dark clump in the middle of the picture. Only roughly one to two hundred vectors are further away. Every algorithm has its own way of distributing nodes between these two areas.

First we look at the Evolving Tree, which focuses very strongly on the dense area. There are only a few nodes in the sparse data areas, but they follow the overall shape of the data quite well. The two furthest nodes cover only a few data vectors which are most likely outliers.

The SOM's rigid grid structure can be seen in the third picture as the nodes form regular structures. SOM focuses less on the dense areas than ETree. SOM's training formulas also prevent it from covering the entire data set, it is not as spread out as ETree. This is known as the *border effect* of SOM. Another consequence of the training is that there are a lot of nodes in the semi-sparse area up and right of the center. This area is likely overrepresented.

Last we have k-means which seems to have best captured the overall shape of the data cloud. There are, however, several outlier clusters which lie very far from the data cloud. These are not shown on the image due to scaling issues. Another potential drawback is that k-means seems to put more weight on the sparse areas than the data blob in the center. This behaviour

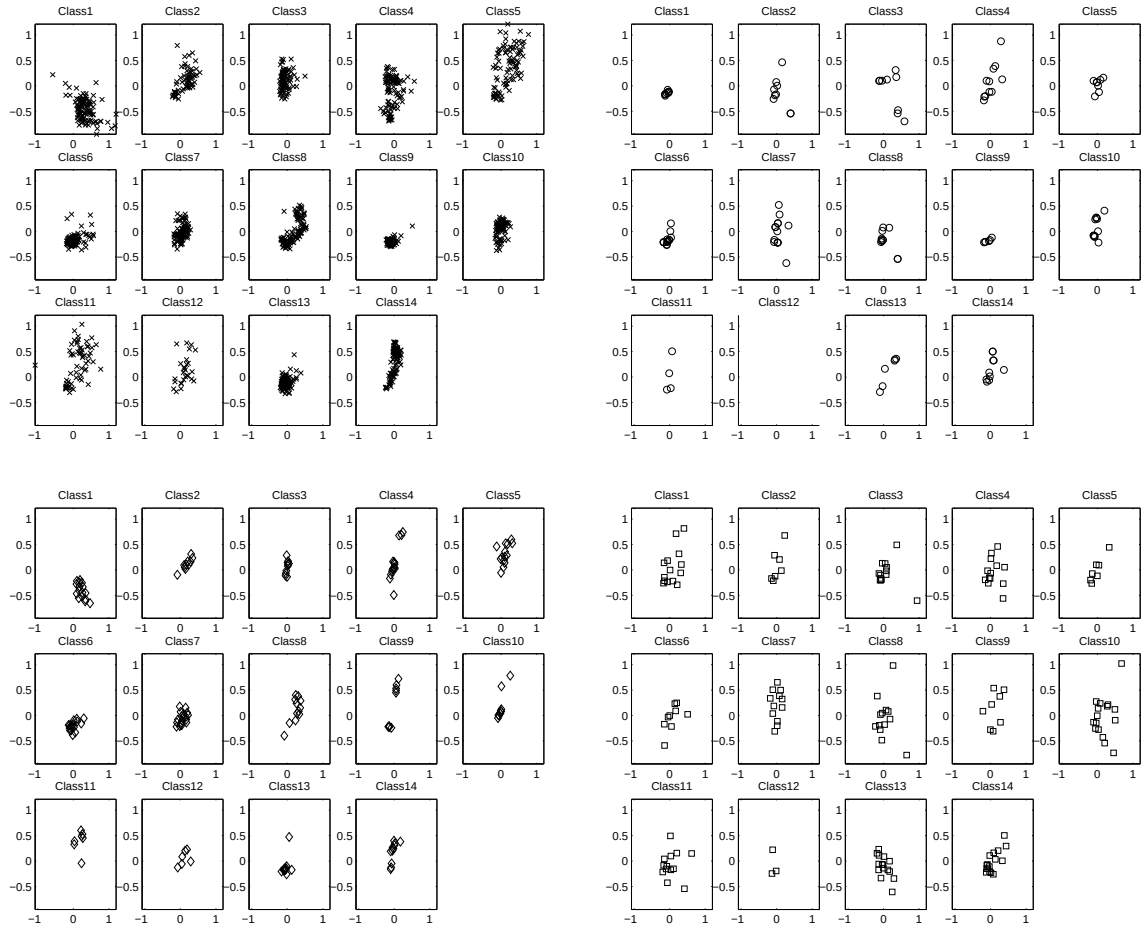

Figure 4: Sammon projected locations of data vectors (top left) and nodes for ETree (top right), SOM (bottom left) and k-means (bottom right) per class.

can be justified in this case since k-means obtains the best classification results.
The second image set in Figure 3 shows the locations of ETree nodes per layer. They show how the tree grows outwards as the training progresses. The first two layers are rather plain, but they demonstrate how every node splits into four. The next ones are more interesting. They show how the tree grows outward layer by layer. The large amount of splits at the dense central area can also be seen in every layer. Finally at layer six the tree only grows at the dense area. Weight decay prevents the outmost branches from growing. This behaviour is exactly as was predicted in the algorithm description.

Leaf node locations per class

Figure 4 shows the locations of Sammon projected data vectors for each of the 14 classes. There are no clear cluster differences and many of the classes overlap significantly. Given this very difficult data set, it is not surprising that none of the algorithms works properly for all cases. Class 12 demonstrates the very difficult nature of the data set. The current ETree

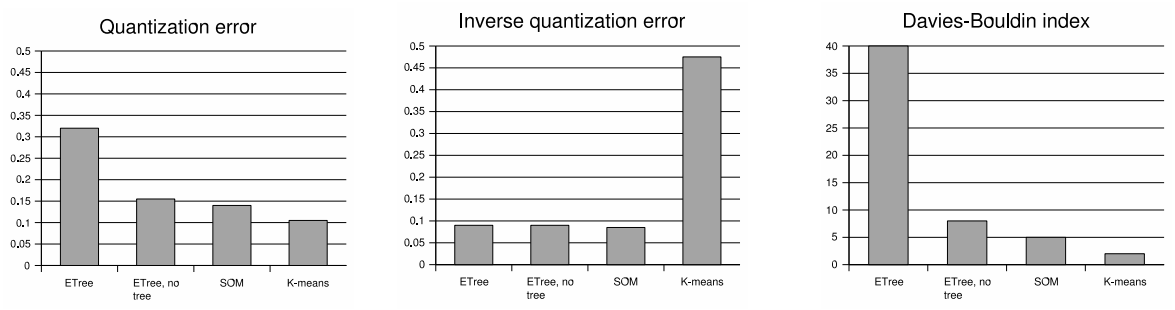

Figure 5: Regular and inverse quantization errors and the Davies-Bouldin index for the Evolving Tree with and without the tree, SOM and k-means, respectively.

does not have a single vector in that class. This is unfortunate, but somwhat expected, since the defects in this class are almost identical to class 11 and there are only about 30 of them. ETree assigns nodes to these classes in some runs but not in others. The overall shape of each class is somewhat captured, but careful analysis locates several anomalies.
Take for example class 9, which is a small clump just below the middle area. All algorithms find this clump quite well, but both SOM and k-means have spurious nodes over the main cluster. Similarly class 5 is elongated to the top and right, but only SOM finds it. Overall the results mirror the above results for the total data set. The Evolving Tree has the tightest node locations, k-means is very spread out and SOM stands between these two.

3.3 Quality of clustering

One very common way of measuring different clustering algorithms is *quantization error*. It is usually defined as the average distance from a data vector to its nearest codebook vector. Given two similar methods, the one with the smaller quantization error is usually the better one. Figure 5 shows the quantization error for the regular ETree, for ETree without the search tree, for SOM, and k-means clustering. ETree without the tree simply means discarding the trunk nodes after training and using the leaf nodes as quantizers like in k-means clusters. The homogeneous texture data set was used for these experiments. K-means and SOM obtain noticeably smaller quantization errors, but the situation is not as straightforward.
SOM and k-means are flat data structures, meaning that all nodes perform an identical task. The Evolving Tree, on the other hand, uses a hierarchal divide and conquer approach. The nodes at the top layers partition the data space with hyperplanes. In any non-trivial problem the clusters are non-separable so the partitioning will inevitably split some clusters non-optimally. This increases quantization error. This is apparent in the second column, removing the search tree halves the quantization error. This is the trade-off of a hierarchical structure: the ability to do fast operations causes some losses in accuracy.
This kind of error is inevitable in almost all hierarchical schemes. Therefore they cannot obtain error levels of flat algorithms. The interesting question, then, is how much larger the error is. If the partitions are chosen badly, the error differences can grow quite large. The Evolving Tree does rather well in this regard. The error is only about three times as large as with k-means. This is a satisfactory result, especially since the classification performances of

the different algorithms are very close to each other.
Quantization error measures the average distance from a data vector to a node. Adding nodes to an existing network can only decrease the error, no matter where in the data space they are placed. An efficient system has as few nodes as possible. To examine the efficiency we calculated the *inverse quantization error*, that is, the average distance from a node to its nearest data vector. If this value is much larger than normal quantization error, it implies the existance of outlier analysis nodes which may hinder computational efficiency. The results can be seen in the second image in Figure 5.
Etree and SOM have the smallest errors, while k-means' error is the largest. A large portion of k-means' error most likely comes from the outlier points mentioned in the previous chapter. In other experiments the inverse error was a lot smaller, some times even the smallest of the three. This shows that k-means is susceptible to outlier nodes. These don't usually matter, but they reduce the overall efficiency. Conversely ETree and SOM stay very close to the data cloud.
Finally we calculated the *Davies–Bouldin index* [2] for the different methods. It is one of the most common measures for examining clustering validity. Its basic assumption is that a good clustering consists of compact clusters that are well-separated. It is calculated using the following formula:

$$\mathrm{DBI} = \frac{1}{n}\sum_{i=1}^{n}\max_{i\neq j}\left(\frac{S_n(C_i)+S_n(C_j)}{S(C_i,C_j)}\right).$$

Here $S_n(C_i)$ is the average distance of data vectors to the cluster center in class C_i. $S(C_i, C_j)$ is the distance between the centers of clusters C_i and C_j. A good clustering as described above should minimize this function.
The results are in the rightmost image in Figure 5. K-means obtains the best results. This is quite understandable since the DB-index favors compact clusters that are well separated. This is what the k-means algorithm is trying to achive as well. A much more interesting result is the search tree's effect on the Evolving Tree. Using only the leaf nodes gives very roughly the same results as the SOM, but the entire tree's index value is a whole lot bigger. This indicates that the clusters it forms are most likely elongated with non-spherical shapes. Since the data is very fuzzy and overlapping, this is quite natural.
It is interesting that even though these algorithms describe the data in very different ways, their classification percentages are almost equal. We suspect this is also due to the structure of the data. Some portions of the data space are easy to analyze, but a significant portion is fuzzy. Exact classification in these areas is extremely difficult or maybe even impossible. In contrast there are several different ways to analyze the area that are very unlike each other, but still"good enough". This leads to the contrasting result images.

4 Conclusions

We have examined the performance of the ETree and compared it to SOM and k-means clustering. We have found that its perfromance is on the same level as the other systems, but that it performs noticeably faster. Visual experimentation on a very difficult data set has shown that all the systems analyze it very differently. Still, the end results are surprisingly similar.

The results obtained here suggest that ETree can be succesfully applied to difficult large scale problems. Especially those that have been too slow to analyze with classical methods. Our future work includes testing the system with a data base of millions of data vectors.

Acknowledgements

We would like to thank the Technology Development Centre of Finland (TEKES's grant 40102/04) and our industrial partner ABB Oy (J. Rauhamaa) for funding this research. We would also like to thank Petri Turkulainen for coding and running some of the experiments.

References

[1] Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(4):224–227, 1979.

[3] Lasse Holmström, Petri Koistinen, Jorma Laaksonen, and Erkki Oja. Comparison of neural and statistical classifiers — theory and practice. Research Reports A13, Rolf Nevanlinna Institute, Helsinki, 1996.

[4] Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, 3. extended edition, 2001.

[5] B. S. Manjunath, Philippe Salembier, and Thomas Sikora, editors. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons Ltd., 2002.

[6] Jussi Pakkanen, Jukka Iivarinen, and Erkki Oja. The Evolving Tree — a novel self-organizing network for data analysis. *Neural Processing Letters*, 20(3):199–211, December 2004.

[7] Jussi Pakkanen, Antti Ilvesmäki, and Jukka Iivarinen. Defect image classification and retrieval with MPEG-7 descriptors. In Josef Bigun and Tomas Gustavsson, editors, *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 349–355, Göteborg, Sweden, June 29–July 2 2003. Springer-Verlag.

[8] John W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, May 1969.