Visualization on a Closed Surface

Walter A. Kosters and Jeroen F. J. Laros

Leiden Institute of Advanced Computer Science (LIACS)
Universiteit Leiden, The Netherlands
jlaros@liacs.nl

Abstract

We propose a visualization algorithm that, given a set of points in high-dimensional space, will produce an image projected on a 2-dimensional torus. The algorithm is a push and pull oriented technique which uses a sigmoid-like function to correct the pairwise distances. We describe how to make use of the torus property and show that using a torus is a generalization of employing the standard closed unit square. Experiments show the merits of the method.

1 Introduction

In many situations one wants to cluster and/or visualize data [7]. In this paper we will describe a method to visualize a perhaps large set of data points on a 2-dimensional surface: a *torus* U. So we start with a finite set of n data points $\{p_1, p_2, \ldots, p_n\}$. We use some given metric d to compute the distance $d_{ij} = d(p_i, p_j)$ between p_i and p_j ($i, j \in \{1, 2, \ldots, n\}$), which yields a symmetric matrix $D = (d_{ij})_{i,j=1}^{n}$. This matrix D will be the basis for our further actions. Its entries will be referred to as the *desired distances*. Our goal is to obtain points $\{p'_1, p'_2, \ldots, p'_n\}$ (the so-called *current points*) in U, in such a way that the distance between p'_i and p'_j in U (the *current distance*) resembles d_{ij}, the desired distance between p_i and p_j, as much as possible for $i, j \in \{1, 2, \ldots, n\}$. The total sum of the absolute differences between the current distance and the desired distance is therefore minimised. Together, the current points constitute the *current configuration*. Once this configuration is established, it can be used for all sorts of clustering purposes.

Our algorithm repeatedly takes two current points, and pushes them together or pulls them apart with a *correction factor*, depending on the relation between desired and current distance. We use an *inflation factor* and a *correction multiplier* to improve the current configuration. Note that the distances in U do not change when one rotates, mirrors or translates all points. Since our method makes use of random elements, visualizations might be the same under rotation, mirroring or translation, but it is also possible that they are actually different.

There are many methods that perform a dimension reduction. We mention Multi Dimensional Scaling (MDS, see [1, 4]) and Principal Component Analysis (PCA, see [4]) as two well-known statistical methods. Other methods include several types of (competitive) neural networks, such as Kohonen's Self Organizing Maps (SOMs, see [4]) and vector quantization (again, see [4]). A comparison of all these methods is beyond the scope of this paper (e.g., see [2]), we just mention two issues. First, our method is intuitive, very fast and requires no complicated mathematical operations, such as matrix inversion; it is suited for large datasets. Second, it is possible to add, remove or alter data points when the algorithm is running, and even trace individual points. It is even possible to visually inspect the current configuration in each iteration. Finally, the use of the torus appears to be both natural and easy to describe; it also performs better than the previously used closed unit square (with boundary, cf. [6]), but still has all its merits. Notice that when using a 0.5×0.5 sub-square of U, one has this situation as a special case.

In Section 2 we define the torus, and mention some alternative topologies. The method is described in Section 3. Section 4 has experiments, and we conclude in Section 5.

2 The torus

In this section we mention some issues concerning our surface. We first give the proper definitions, and also point out a few difficulties that might arise, and some other possibilities.
The surface is basically the unit square U in $\mathbb{R}^2$, with sides identified in such a way that it topologically is a *torus*: left and right boundary are identified, and so are top and bottom boundary, see Figure 1 below. The resulting surface has no boundaries. As distance between two points a and b in U we just take the minimum of the ordinary Euclidean distances between a and any point from $\{b+(k,\ell) \mid k,\ell \in \{-1,0,1\}\}$. This surface will be referred to as "the" torus. Note that the distance is *not* the one that arises when a torus is embedded in $\mathbb{R}^3$ in the usual way (as a doughnut). In our case a visualization as a unit square is more appropriate, remembering that left-hand and right-hand side are near to one another (and also top side and bottom side).

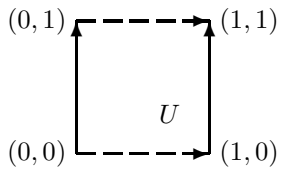

Figure 1: Unit square with sides identified: the torus U.

As specified above, the surface we use is not the standard 2-dimensional unit square in the Euclidean space $\mathbb{R}^2$, but a 2-dimensional torus. The main advantage of using such a manifold is that there are more degrees of freedom in such a space.
A disadvantage of using a torus is that it is impossible to contract every circle to a point, and thus there are configurations possible where clusters are wrapped around the torus and thus might get stuck in a "local minimum". A solution to this is to use a sphere (where each circle can be contracted to a point), but the projection of a globe onto a flat 2-dimensional space gives a distorted image (just like the map of the earth, the polar regions usually appear much larger than they actually are).
Another way to prevent the potential wrapping around the torus is to use a non-random initialization. If all points are initialized in one (small) area, the process will most likely not result in a configuration where wrapping is an issue. This can even be forced by placing a maximum distance (determined by the circumference of the torus) on the correction part of the algorithm.
There are more possibilities for such surfaces, like the non-orientable Klein bottle (obtained when identifying the dotted arrows from Figure 1 in opposite direction) or the real projective plane, but from all these, the metric on a torus (as specified above) is most like the standard Euclidean one, so it is natural to choose this object.

3 Algorithm

The algorithm we use is a *push and pull* oriented one, where the correction factor depends on the difference between the desired distance d_{desired} and the current distance d_{current}. This current distance, or rather its square, between two points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ from U can be efficiently computed by:

$d_{\text{current}}\,((x_1,y_1),(x_2,y_2)) ::$
  var $x_3 \leftarrow x_2$;
  var $y_3 \leftarrow y_2$;
  if $x_1 - x_2 > 0.5$ **then** $x_3 \leftarrow x_3 + 1.0$;
  if $x_1 - x_2 < -0.5$ **then** $x_3 \leftarrow x_3 - 1.0$;
  if $y_1 - y_2 > 0.5$ **then** $y_3 \leftarrow y_3 + 1.0$;
  if $y_1 - y_2 < -0.5$ **then** $y_3 \leftarrow y_3 - 1.0$;
  return $(x_1 - x_3)^2 + (y_1 - y_3)^2$;

Quadratic distance between points in U

The point $b' = (x_3, y_3)$ is the (or more precisely, a) point from $\{b + (k, \ell) \mid k, \ell \in \{-1, 0, 1\}\}$ that realizes the shortest distance to a. This point will also be used later on. The maximal quadratic distance between any two points from U equals 0.5. (We will omit the word "quadratic" in the sequel.)
Instead of a linear or a constant function (of the current distance) to calculate the amount of correction, we can and will use a sigmoid-like function, or rather a family of functions. This function must adhere to some simple constraints, enumerated below. So we want a function $f = f_{d_{\text{desired}}}$ which is defined on $[0, 0.5]$, where 0.5 is the maximum distance between two points (on the torus). We must have, with $0 < d_{\text{desired}} < 0.5$ fixed:

- $f(0) = \rho$
- $f(0.5) = -\rho$
- $f(d_{\text{desired}}) = 0$

Here $\rho \in (0, 1]$ is the so-called *correction multiplier*. So when the current distance is as desired, f has value 0 — and so has the correction. The resulting correction factor $corrfac$ equals $f(d_{\text{current}})$. If $d_{\text{desired}} = 0$, we make it slightly larger; similarly, if $d_{\text{desired}} = 0.5$, we make it slightly smaller.
We will use

$$f_{d_{\text{desired}}}(x) = \begin{cases} \rho \cos(\pi \log_t(2x(t-1)+1)) & \text{if } d_{\text{desired}} \neq 0.25 \\ \rho \cos(\pi 2x) & \text{if } d_{\text{desired}} = 0.25 \end{cases}$$

where $t = (1 - 1/(2d_{\text{desired}}))^2$; this function satisfies all the constraints. Figure 2 depicts $f_{0.1}$ and $f_{0.25}$, with $\rho = 1$.
The reason we choose a function like this, is because the correction of a point will be large when the error of that point is large. Only when the error is close to zero, the correction will be small. Other functions like suitable sigmoids will have the same behaviour, and could also be employed.

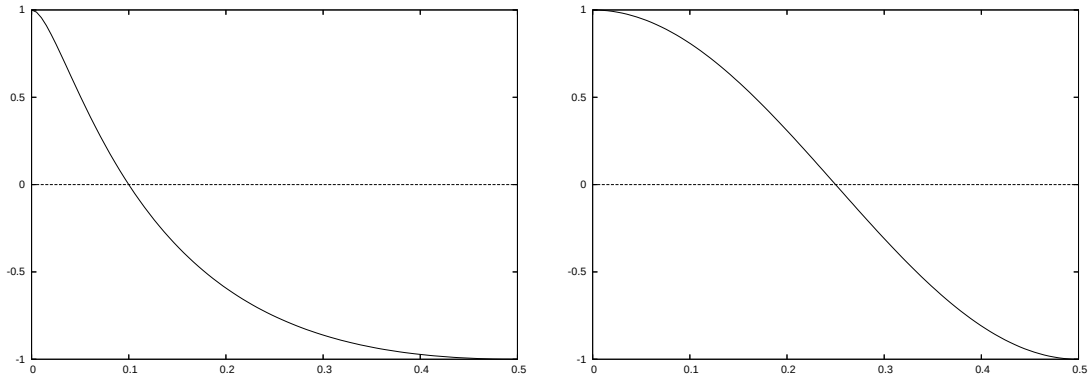

Figure 2: $f_{d_{\text{desired}}}$ with $d_{\text{desired}} = 0.1$ and $d_{\text{desired}} = 0.25$, $\rho = 1$.

Now suppose we want to "push and pull" two given points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ in U; we first compute $b' = (x_3, y_3)$ as in the distance calculation of d_{current} above. Then the coordinates x_1 and y_1 of a are updated through

$$\begin{aligned} x_1 &\leftarrow x_1 + corrfac \cdot |d_{\text{desired}} - d_{\text{current}}| \cdot (x_1 - x_3) \;/\; 2 \\ y_1 &\leftarrow y_1 + corrfac \cdot |d_{\text{desired}} - d_{\text{current}}| \cdot (y_1 - y_3) \;/\; 2 \end{aligned} \tag{1}$$

A positive $corrfac$ corresponds with pushing apart, a negative one with pulling together. In a similar way, the coordinates x_3 and y_3 of b are updated in parallel. If a coordinate becomes smaller than 0, we add 1, and if it becomes larger than 1, we subtract 1. Together we will refer to this as Equation (1).
The basic structure of the algorithm is as follows:

initialize all current points in a small region of U
while not $Ready$ **do**
    update all pairs (in arbitrary order) with Equation (1)

The push and pull algorithm

The algorithm terminates when the mean error and the standard deviation no longer substantially change, or are low enough. The error is defined in the usual way: we just compute

$$\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} |d_{\text{desired}}(i,j) - d_{\text{current}}(i,j)|$$

where $d_{\text{desired}}(i,j)$ and $d_{\text{current}}(i,j)$ are the desired and current distance between data points p_i and p_j, respectively.
Of course, the algorithm might get stuck in a local optimum. And it is also possible that the final error remains high or keeps fluctuating, especially if the data is distributed in a complicated way. So far, we have not experienced this in practical situations, where convergence was always achieved.
We now introduce the inflation factor σ, and secondly comment on the correction multiplier ρ.

3.1 Inflation factor

The *inflation factor* $\sigma > 0$ can be used in the following way: Equation (1) is changed to

$$x_1 \leftarrow x_1 + \mathit{corrfac} \cdot |\sigma \cdot d_{\text{desired}} - d_{\text{current}}| \cdot (x_1 - x_3) \;/\; 2. \tag{2}$$

This can be useful in several ways. If, for example, all distances are between 0 and 0.2, one might argue that it is useful to multiply these distances by 2.5 to get a better spreading. This argument is especially valid if the resulting clustering cannot be realized in the plane, but can be embedded on a torus. Inflation with the right factor can make the overall error drop to zero in this case, while using the original distances will always result in a non-zero overall error.
Even if all distances are between 0 and 0.5, inflation or deflation can still be beneficial. For example, the input data can be such that inflation or deflation will result in the correct clustering of a large part of the input, while not using an inflation factor will result in a much higher overall error. An example of such input data would be a torus that is scaled between 0 and 0.2, with a few points outside this region. Normal clustering would result in a flat image where the points outside the torus region would have correct distances to the torus region, but with the correct inflation factor, the torus will be mapped on the entire space, and the few points outside the region will be misclustered. This results in a clustering where the overall error is small.
In practice we often take $\sigma = 1$.

3.2 Correction multiplier

The correction multiplier ρ is a parameter which controls the aggressiveness of the correction function. Initially this factor is set to 1, but for data that can not be embedded in the plane, lowering this factor can be beneficial.
If, for example, most of the distances are near the maximum, the correction function will push them so far apart, that they are pushed towards other points at he other side of the torus. This can result in the rapid fluctuation between two or more stable states. These states are probably not the global minimum for the clustering error, and therefore not the end result we desire. Increasing the correction multiplier will counter this effect.

4 Experiments

In this section we describe several experiments, both on synthetic and real data. The experiments are of an exploratory nature. We try to give a good impression of the merits of the algorithm.
We start with a synthetic dataset. In the left-hand picture of Figure 3 we see the original data points (in a "flat" 2D plane), from which a distance matrix is derived to serve as test data for the visualization algorithm. In this picture we see seven spheres of which three are unique: the topmost two and the one in the center. The other four spheres are copies of one another. The total number of points is 700 and all distances are between 0.0 and 0.5.

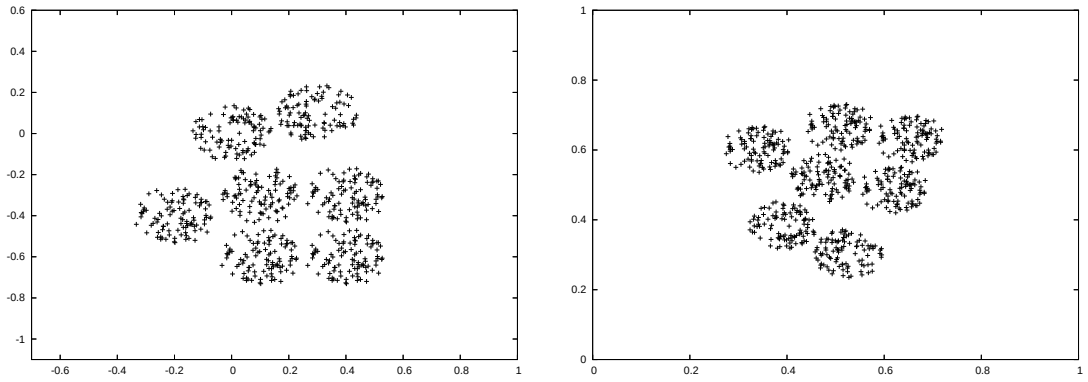

Figure 3: Original data in the 2D plane (left) and visualization on the torus (right).

After only a few iterations of our algorithm, the right-hand picture of Figure 3 appears. Notice how it resembles the input data, except for a mirroring and a rotation. All distances are preserved almost perfectly. Remember that only the pairwise distances were used by the algorithm. The mean error in this picture is 0.00004 and the standard deviation is 0.00003. As a final remark, "flat data" will always cluster within a sub-square of size 0.5×0.5.

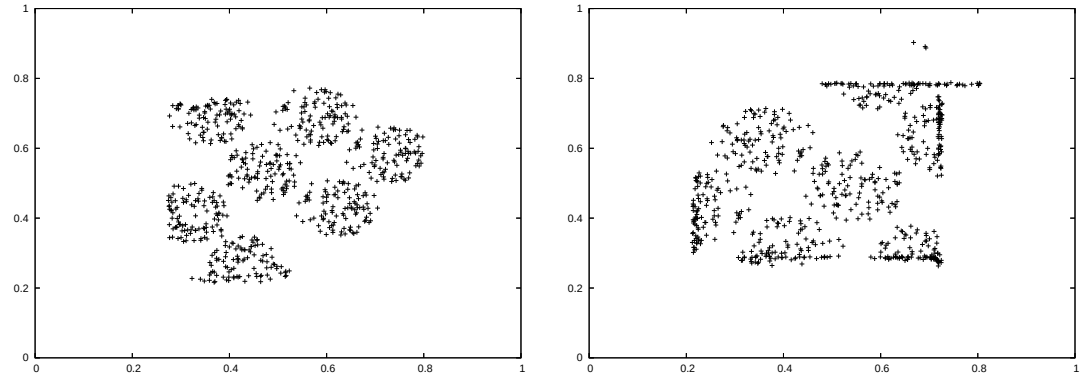

Figure 4: Visualization of flat data on the torus with an invalid inflation factor.

In Figure 4 we see the same test data, except that the distances have been multiplied by a factor 1.5 in the left-hand picture, and by 2.6 in the right-hand one. This results in a non-correct embedding, since the maximum distance in this space is 0.5. The effects can be seen in Figure 4, in particular in the right-hand picture. In both pictures a translation has been applied in order to center most points. Though the full 1.0×1.0 square has been used, most current points reside in the smaller 0.5×0.5 square, as is clearly visible in the right-hand picture.
The top-left sphere is forced closer to the bottom-left one than is possible. This results in the flattening of the spheres at the outermost edges. This effect can be explained by considering the overall error (which is minimized). By a local deformation, the overall error is kept small. The effect can also be seen (to a lesser

extent) in the middle-left sphere. Notice that the effect is absent in the top-central sphere because of the void at the bottom-center of the picture (there is nothing to collide with at the other side of the torus).

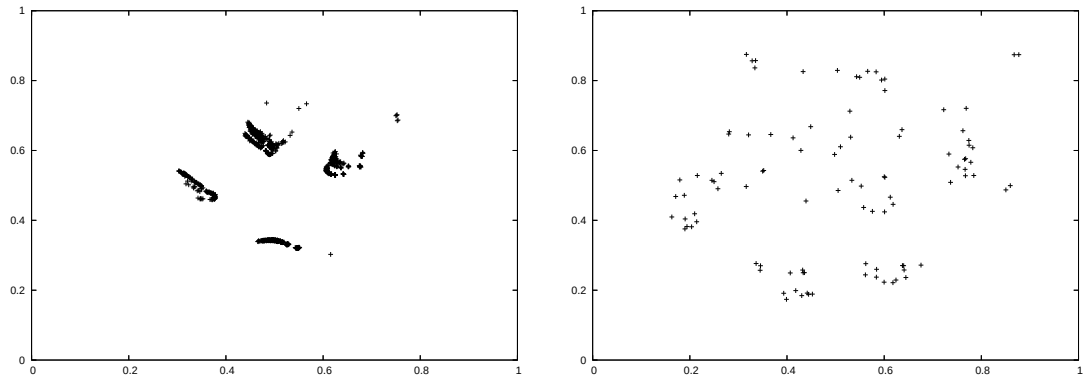

Figure 5: Visualization of criminals, non flat case. Left: with categories; right: without categories.

In Figure 5, left, we see a visualization of real data. We have taken a database of $1{,}000$ criminal records and divided the crimes into three categories (light, middle, heavy): each record has three integers, describing the number of crimes in the respective categories. The distance measure we use is one defined on multisets and is described in [5]. It basically averages the absolute differences between the numbers of crimes. We mention that MDS gives a similar picture.
The resulting matrix cannot be embedded in the plane, but it almost could, since the mean error is relatively small 0.00494 and so is the standard deviation 0.00529. We refer to this type of situation as a "non flat case". An indication that the data is almost flat, is that the clustering stays within the 0.5×0.5 sub-square, and inflation increases the error. There are four main clusters in this picture, where:

- The leftmost one consists of criminals that have committed relatively light crimes. They all fall into the categories light and middle.
- The top one consists of all-rounders, they have all committed crimes in all categories.
- The rightmost one consists of criminals that have only committed light and heavy crimes, nothing in between.
- The bottom one consists of criminals that have only committed light crimes, all of them fall into the category light.

Then there is a very small cluster in the top-right corner of the picture, this is a cluster of people who have only committed heavy crimes. This is apparently non-standard behaviour for a criminal. There are a few other isolated points in this picture, they all are people with a strange criminal record.
In Figure 5, right, we see the clustering of 100 criminals based upon the same distance measure as in Figure 5, but now we do *not* categorize the crimes; here the records have 80 attributes. The result is a scattered image (largely due to the lack of similarity), occupying a large part of the unit square, and only a few local clusters. We make use of inflation factor $\sigma = 2$ and correction multiplier $\rho = 1/16$ here, to produce the picture with a mean error of 0.02636 and a standard deviation of 0.01786. All visualizations are obtained within a few seconds.
Finally, we show an example from chemistry. The dataset we use, the so-called `4069.no_aro` dataset, contains 4,069 molecules (graphs); from this we extracted a lattice containing the 2,149 most frequent subgraphs. These are grouped into 298 structural related patterns occurring in the same molecules using methods presented in [3], resulting in a 298 by 298 distance matrix; the distance between graphs is based on the number of co-occurrences.
Figure 6 shows two visualizations. The left-hand picture has mean 0.03488 and standard deviation 0.03117, with parameters $\rho = 0.048$ and $\sigma = 1.1$; the right-hand picture has mean 0.05029 and standard deviation 0.03200, with parameters $\rho = 0.031$ and $\sigma = 0.5$. The latter picture is what we would have gotten when we had used a bounded unit square. The first picture gives a better embedding, with a lower error. The groups that pop up can be used by a biologist to investigate biological activity.

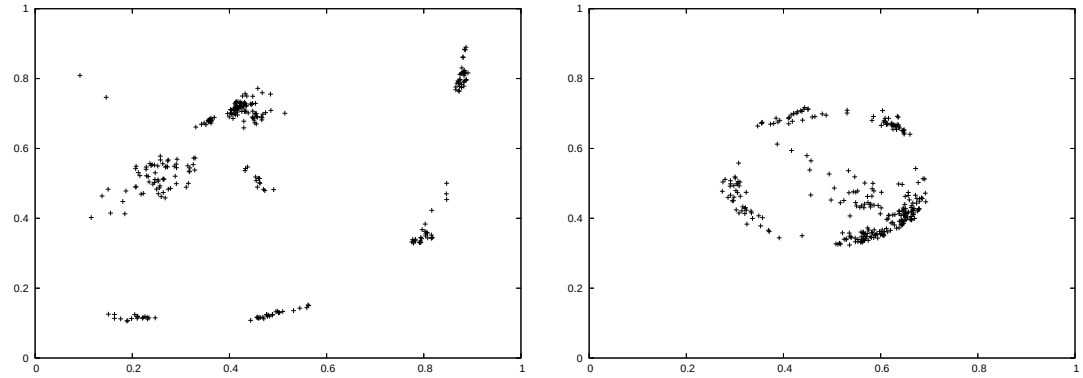

Figure 6: Two visualization of a dataset with molecules; left: good embedding; right: inferior embedding, in fact the torus property is used but not exploited.

5 Conclusions and further research

We conclude that our algorithm is able of giving adequate visualizations on the torus. Starting from a set of data points and their pairwise distances, it quickly provides an embedding on this surface. The algorithm is fast, flexible and easy to use, for instance for clustering purposes.
The method was originally developed for the analysis of criminal records (see Section 4), and performs quite well in this case, but it also appears to be applicable in other fields. We would like to compare its performance with that of existing methods.
For further research, we would like to examine other topologies, such as a sphere. Yet another possibility is to somehow fix current points, once they have reached a good position with respect to *many* other points. And finally, the online addition of new points.

Acknowledgements

This research is part of the DALE (Data Assistance for Law Enforcement) project as financed in the ToKeN program from the Netherlands Organization for Scientific Research (NWO) under grant number 634.000.430.
The authors would like to thank Timo Krul for an insightful simplification of the distance algorithm.

References

[1] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 1997.

[2] I.K. Fodor. A survey of dimension reduction techniques. Technical Report, Lawrence Livermore National Laboratory, 2002, `http://www.llnl.gov/tid/lof/documents/pdf/240921.pdf`.

[3] E.H de Graaf, J.N. Kok and W.A. Kosters. Improving the exploration of graph mining results with clustering. In *Proc. 4th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI 2007)*, Athens, Greece, to appear.

[4] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[5] W.A. Kosters and J.F.J. Laros. Metrics for mining multisets. In *Proc. Twenty-seventh SGAI International Conference on Artificial Intelligence (AI-2007)*, Cambridge, UK, to appear.

[6] W.A. Kosters and M.C. van Wezel. Competitive neural networks for customer choice models. In J. Segovia, P.S. Szczepaniak and M. Niedzwiedzinski, editors, *E-Commerce and Intelligent Methods*, Studies in Fuzziness and Soft Computing 105, Springer, pages 41–60, 2002.

[7] P.N. Tan, M. Steinbach and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.