Interactive Exploration of Fuzzy Clusters Using Neighborgrams

Bernd Wiswedel, David E. Patterson, and Michael R. Berthold

Abstract—We describe an interactive method to generate a set of fuzzy clusters for a given data set. The presented method is best suited for applications where the focus of analysis lies on a model for the minority class or for small- to medium-size data sets. The clustering algorithm creates one-dimensional models of the neighborhood of a set of patterns by constructing cluster candidates for each pattern of interest and then choosing the best subset of clusters that form a global model of the data. The accompanying visualization of these neighborhoods allows the user to interact with the clustering process by selecting, discarding, or fine-tuning potential cluster candidates. Clusters can be crisp or fuzzy and the latter leads to a substantial improvement of the classification accuracy. We demonstrate the performance of the underlying algorithm on several data sets from the StatLog project.

I. INTRODUCTION

The analysis of large data sets has gained considerable interest over the past decade or so. Modern technology allows to record vast amounts of data that need to be analyzed in order to reveal interesting relationships. Methods from diverse disciplines have been combined to help analyze such data sets, an introduction to the most commonly used methods can be found in [1]. In recent years, techniques to visualize data or models that summarize the data have emerged that allow interactive exploration of data or models, [2] summarizes some of these approaches.

Many different approaches to clustering have been proposed in the past. Most methods iterate over the available data in order to minimize a particular objective function. Most of these algorithms attempt to find a cluster center by continuously adjusting the location of a cluster representative, many of the well-known fuzzy clustering algorithms are of this type ([3] summarizes many of them). Adjusting the cluster centers in an iterative manner (often performed by gradient descent procedures) makes sense if we deal with huge amounts of data. The computational effort is reasonable – at least in comparison to, for example, the training of a multi layer perceptron. However, the final solution is often suboptimal and in addition, the number of clusters must be known in advance.

Approaches which find the cluster centers directly have also been proposed. One example is the DDA algorithm, a constructive training algorithm for Probabilistic Neural Networks [4]. The algorithm picks the cluster centers directly from the training patterns. A new cluster is introduced whenever the current network cannot model the newly encountered pattern. The new cluster is positioned at this pattern. This approach does not find optimal cluster centers, since the positions of the cluster centers depend on the order in which the training patterns are processed but it is straightforward to use and very fast.

The algorithm described here belongs to the latter category of algorithms. But rather than using a heuristic or greedy algorithm to select example patterns as cluster representatives, the proposed method analyzes the neighborhood of each cluster candidate and picks the optimal cluster representative directly. Such a complete and hence computationally expensive approach obviously only works for all classes of a medium sized data set or—in case of very large data sets—to model a minority class of interest. However, in many applications the focus of analysis is on a class with few patterns only, a minority class. Such data can be found, for instance, in drug discovery research. Here, huge amounts of data are generated for example in High Throughput Screening, but only very few compounds really are of interest to the biochemist. Therefore it is of prime interest to find clusters that model a small but interesting subset of data extremely well.

This paper mainly focuses on this aspect.

This paper deals with a representation of such data by computing a Neighborgram for each example of the minority class. A Neighborgram is a summarization of the neighborhood of a pattern, which allows an interpretable view on data. The proposed algorithm then finds clusters in a set of such Neighborgrams based on an optimality criteria. Since Neighborgrams are easy to interpret, the algorithm can also be used to visually suggest clusters to the user, who might be interested in influencing the clustering process in order to inject his/her expert knowledge. So the clustering can be done fully automatically¹, interactively, or even completely manually. Furthermore, Neighborgrams are distance based, which makes them applicable to data sets where only distances between patterns are known.

This paper is organized as follows. We first introduce the concept of Neighborgrams and describe the basic algorithm. Section III discusses experiments on some publicly available datasets from the StatLog project and demonstrates that the fully automated clustering algorithm achieves results comparable to those of state-of-the-art techniques. We continue by sketching the usefulness of the visual clustering and conclude with a discussion of possible extensions and future directions of research.

¹This paper mainly focuses on this aspect.
II. NEIGHBORGRAM CLUSTERING

In this section we first introduce the concept of Neighborgrams, the underlying data structure of the proposed clustering algorithm. Afterwards we describe the clustering algorithm itself along with a discussion of different membership functions.

We assume a set of training patterns \( T \) with \( |T| = M \) instances for which distances, \( d(x_i, x_j) \), \( i, j \in \{1, \ldots, M\} \), are given\(^2\). Each example is assigned to one of \( C \) classes, \( c(x_i) = k, 1 \leq k \leq C \).

A. Neighborgrams

A Neighborgram is a one-dimensional model of the neighborhood of a chosen pattern, which we will call the centroid. Depending on the distance other patterns have to the centroid they are mapped into the Neighborgram. Therefore, a Neighborgram summarizes the neighborhood of one pattern. One can imagine a Neighborgram as a ray onto which the neighborhood of the centroid is plotted. Obviously, if all patterns were mapped onto the ray it would become crowded and the visualization would lose its clarity. Therefore we introduce a parameter \( R \) which indicates the number of patterns stored in a Neighborgram. Those \( R \) stored items are therefore the \( R \)-closest neighbors to the centroid. For this reason a Neighborgram can also be considered as an ordered list of length \( R \).

\[
\text{NG}_i = [x_{i_1}, \ldots, x_{i_R}] ,
\]

The list \( \text{NG}_i \) is sorted according to the distance of pattern \( x_{i_r} \) to the center vector \( x_i \):

\[
\forall r : 2 \leq r \leq R \land d(x_i, x_{i_{(r-1)}}) \leq d(x_i, x_{i_r}) ,
\]

and the patterns in the Neighborgram are the closest neighbors of the centroid:

\[
\exists r : r > R \land d(x_i, x_{i_r}) < d(x_i, x_{i_R}) .
\]

Note that \( i_1 = i \), because \( d(x_i, x_i) = 0 \) for all \( i \), that is, each pattern is closest to itself.

Obviously in case of large data sets the computation of Neighborgrams for each training pattern is excessively time and memory consuming. As noted earlier, however, the main target of the algorithm discussed here are problems where one (or several) minority class(es) are of prime interest. The computation of Neighborgrams for all these patterns is then of complexity \( O(R \times M \times M') \), where \( M' \) indicates the number of examples of the minority class(es), i.e. \( M' \ll M \) in case of large data sets. This complexity estimate was derived as follows: for each pattern \( O(M) \) and for each Neighborgram \( O(M') \) do an Insertion Sort into a list of \( R \) patterns \( O(R) \). If the size \( R \) of the Neighborgrams is closer to the overall number of pattern \( M \) it might make more sense to use a more efficient sorting scheme but for large data sets usually \( R \ll M \) holds and an insertion sort is sufficiently efficient.

example 2.1: Figure 1 shows an example with three patterns of the positive class (empty circles) and four negative patterns (solid circles). The Manhattan distance is used to determine distances between patterns, which can be calculated by counting vertical or horizontal steps that have to be taken to move along the grid. Between \( \otimes \) and \( \oplus \) there are two steps in horizontal direction and one in vertical direction. Therefore the distance is 3. The three Neighborgrams on the right show the neighborhoods of the respective pattern for \( R = 5 \). The lists defining the Neighborgrams can also be written as:

\[
\begin{align*}
\text{NG}_\otimes &= [\otimes, \oplus, \otimes, \otimes, \otimes] \\
\text{NG}_\oplus &= [\otimes, \otimes, \oplus, \oplus, \otimes] \\
\text{NG}_\ominus &= [\oplus, \oplus, \otimes, \otimes, \otimes]
\end{align*}
\]

We will continue to use this example in the following section to illustrate the basic algorithm.

B. The Basic Clustering Algorithm

The key idea underlying the clustering algorithm is that each pattern for which a Neighborgram has been built is regarded as a potential cluster center. The objective of the algorithm is to rank Neighborgrams in order to find the “best” cluster at each step. The result is a subset of all possible clusters.

The algorithm can be summarized as follows:

1) Compute a cluster candidate for each Neighborgram,
2) rank cluster candidates and add the best one as a cluster,
3) remove all patterns covered by this cluster, and
4) start over at step 1, unless certain stopping criteria are fulfilled.

Obviously it needs to be defined what a cluster candidate is, how these candidates can be ranked, and what “removing covered patterns” really means. In addition, the termination criterion has to be specified. In order to do this, let us first define a few properties of Neighborgrams.

1) Parameter and Properties: In section II-A an ordered list has been suggested as a representation of a Neighborgram. This list contains patterns which are ordered due to the exact distance. The length of the list is determined by the parameter \( R \).

\[
\text{NG}_i = [x_{i_1}, \ldots, x_{i_R}] 
\]

Note that it is incorrect to say that \( x_{i_r} \) is \( r \)-th nearest neighbor since we allow patterns to have the same index. The parameter \( r \) is called depth.

The main parameters to describe a cluster candidate are the following:

- Purity II: The purity of a Neighborgram is the ratio of the number of patterns belonging to the same class as
the centroid to the number of patterns encountered up to a certain depth \( r \). The purity is a measure how many “good” vs. “bad” patterns a certain neighborhood around the centroid contains.

\[
\Pi_i(r) = \frac{\{(x_{l,r} \in NG_i \mid 1 \leq r' < r \land c(x_{l,r}) = c(x_i)\}\}}{\{(x_{l,r} \in NG_i \mid 1 \leq r' \leq r\}\}}
\]

**example 2.2:** For Figure 1 the purity for the pattern \( \ominus \) depending on the depth \( r \) would be: \( \Pi_\ominus(1) = 1 \), \( \Pi_\ominus(2) = \frac{1}{2} \), \( \Pi_\ominus(3) = \frac{2}{3} \), and \( \Pi_\ominus(4) = \frac{3}{4} \).

- **Optimal Depth** \( \Omega \): The optimal depth is the maximal depth where the purity for all depths \( r \) less than or equal to \( \Omega \) is greater than or equal to a given threshold \( p_{min} \). Note that it is straightforward to derive the corresponding radius from a given depth, that is \( d(x_i, x_{l,r}) \).

\[
\Omega_i(p_{min}) = \max \{ r \mid 1 \leq r' \leq r \land \Pi_i(r') \geq p_{min} \}
\]

**example 2.3:** In Figure 1 we find for pattern \( \ominus \) and \( p_{min} = 1.0 \): \( \Omega_\ominus(1.0) = 2 \). For \( p_{min} = \frac{2}{3} \) we get \( \Omega_\ominus(\frac{2}{3}) = 3 \).

- **Coverage** \( \Gamma \): The default coverage of a cluster with a certain depth \( r \) determines how many positive patterns it “explains”, that is, the number of positive patterns that fall within its radius:

\[
\Gamma_i(r) = \{(x_{l,r} \in NG_i \mid 1 \leq r' \leq r \land c(x_{l,r}) = c(x_i)\}\}
\]

**example 2.4:** For pattern \( \ominus \) in Figure 1 the coverage is: \( \Gamma_\ominus(1) = 1 \), \( \Gamma_\ominus(2) = 1 \), \( \Gamma_\ominus(3) = 2 \), and \( \Gamma_\ominus(4) = 3 \).

- **Minimum Size** \( \Lambda \): The minimum size a cluster must have. Computing purity as described above has the disadvantage that for noisy data sets many clusters will not extend as far as they could because an early encounter of a pattern of the wrong class will set the optimal depth \( \Omega \) very close to the centroid. To avoid this, we introduce a parameter \( \Lambda \), which allows to specify a minimum number of patterns in a neighborhood before the purity \( \Pi \) and the optimal depth \( \Omega \) are determined. Early experiments with noisy data sets have shown a decrease in number of clusters and better generalization ability.

Furthermore, there is a parameter for the overall coverage \( \Psi \) which determines the termination criterion for the algorithm. It is the sum of the coverages of the chosen clusters. Once this threshold is reached, the algorithm stops.

### C. Cluster Candidates and the Clustering Algorithm

We can now specify clearer what we mean by “ranking clusters” and “removing covered patterns”. Starting from a user-defined value for parameter purity \( \Pi \) we can compute values for parameters optimal depth \( \Omega \) and coverage \( \Gamma \) for each cluster. The best cluster is the one with the highest coverage. This cluster “covers” all patterns that are within its radius. These patterns are then discarded from the data set and the cluster-selection process can start again, based on the reduced set of patterns to be explained. Table I summarizes the basic algorithm.

It is obvious that the basic algorithm sketched here is very strict — a pattern will be completely removed from any further consideration as soon as it falls within the optimal radius for just one single cluster. This effect might be desirable for patterns lying close to the center of the new cluster but it will reduce accuracy in areas further away from the cluster center. We therefore introduce the notion of Partial Coverage using fuzzy membership functions, which allows us to model a degree of membership of a particular pattern to a cluster. The next section will present the used membership functions.

### D. Membership Functions

The idea underlying the partial coverage is that each cluster is modeled by a fuzzy membership function. This function has its maximum at the centroid and declines towards the cluster boundaries. The Coverage is then determined using these degrees of membership and patterns are removed to a high degree towards the inner areas of a cluster and to a lesser degree towards the outer bounds. The following figures show the four membership functions we used. Note that the rectangular membership function corresponds to the basic algorithm discussed above: patterns are covered only with degrees of 0 or 1, and are hence removed completely when covered.

In order to describe a cluster by means of a membership function we first need to introduce three radii which will help to isolate regions of the neighborhood.

- \( r_1 \) stands for the radius of the last pattern with \( \Pi = 1 \) (last known perfect), \( r_1 = \max \{ r \mid \Pi_i(r) = 1 \} \).
- \( r_2 \) represents the last pattern with \( \Pi \geq p_{min} \) (last known good), i. e., \( r_2 = \max \{ r \mid 1 \leq r' \leq r \land \Pi_i(r') \geq p_{min} \} \).
- \( r_3 \) describes the first pattern for which \( \Pi < p_{min} \) (first known bad), i. e., \( r_3 = \max \{ r \mid 1 \leq r' \leq r - 1 \land \Pi_i(r') \geq p_{min} \} \).

These radii are sufficient to describe the following commonly used membership functions:

<table>
<thead>
<tr>
<th>TABLE I The basic Neighborgram Clustering Algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ( \forall x_i: c(x_i) = \text{minority class} \Rightarrow \text{compute NG}_i )</td>
</tr>
<tr>
<td>2) ( \forallNG_i ): compute ( \Omega_i )</td>
</tr>
<tr>
<td>3) ( \forallNG_i ): compute ( \Gamma_i(\Omega_i) )</td>
</tr>
<tr>
<td>4) ( s := 0 )</td>
</tr>
<tr>
<td>5) while ( s &lt; \Psi )</td>
</tr>
<tr>
<td>6) ( i_{best} = \arg \max { \Gamma_i(\Omega_i) } )</td>
</tr>
<tr>
<td>7) add ( NG_{i_{best}} ) to list of clusters,</td>
</tr>
<tr>
<td>8) add ( \Gamma_{i_{best}}(\Omega_{i_{best}}) ) to ( s )</td>
</tr>
<tr>
<td>9) remove them from all Neighborgrams ( NG_i )</td>
</tr>
<tr>
<td>10) ( \forallNG_i ): recompute ( \Gamma_i(\Omega_i) )</td>
</tr>
<tr>
<td>11) end while</td>
</tr>
</tbody>
</table>
The **Rectangular** Membership Function.

\[
\mu(d) = \begin{cases} 
1, & \text{if } 0 \leq d \leq \frac{r_2 + r_3}{2} \\
0, & \text{otherwise}
\end{cases}
\]

The **Trapezoidal** Membership Function.

\[
\mu(d) = \begin{cases} 
1, & \text{if } 0 \leq d \leq r_1 \\
\frac{r_3 - d}{r_3 - r_1}, & \text{if } r_1 < d \leq r_3 \\
0, & \text{otherwise}
\end{cases}
\]

The **Triangular** Membership Function.

\[
\mu(d) = \begin{cases} 
\frac{r_3 - d}{r_3}, & \text{if } 0 \leq d \leq r_3 \\
0, & \text{otherwise}
\end{cases}
\]

The **Gaussian** Membership Function.

\[
\mu(d) = \exp \left( -\frac{d^2}{\sigma^2} \right) \quad \text{with} \quad \sigma^2 = -\frac{r_3^2}{\ln(\theta)}
\]

While the expansion of the rectangular, trapezoidal, and triangular membership function is only determined by the three radii, the gaussian membership function has an additional parameter \(\theta\) which defines the shape of the membership function. The corresponding radius to \(\theta\), \(r_\theta\), equals the radius \(r_3\). For a purity \(\Pi\) equal to 1, the parameter \(\theta\) determines the maximum degree of membership of an incorrect class for other patterns in the training data.

Using these fuzzy membership functions the clustering algorithm works slightly differently.

First a degree of exposure (or “being uncovered”), \(\eta \in [0, 1]\), needs to be assigned to each pattern of the minority class. At the beginning this value is initialized to 1.0 for each pattern, that is, each pattern still needs to be covered completely. Subsequently this value will be decreased during clustering. A new cluster which (partly) covers a pattern will reduce this value by the amount by which it covers the sample. However, obviously a pattern can only be covered until \(\eta = 0\). Let \(\eta(x)\) be a pattern’s degree of exposure and \(\mu_{\text{Cluster}}(d(x, x_i))\) the degree of membership to the cluster. Then the Partial Coverage, \(\Phi\), of a cluster is defined as:

\[
\Phi_i(\Omega_i) = \sum_{x_{l,r} \in \Omega_i} \min \{ \eta(x_{l,r}), \mu_{\text{Cluster}}(d(x_{l,r}, x_i)) \}
\]

The introduction of this concept of partial coverage improved the clustering substantially — as we will discuss later in section III. The new algorithm is shown in table II. A list of covered patterns needs to be created in conjunction with the individual degrees of coverage. Steps (8) and (9) of the basic algorithm are modified to serve the purpose of partial coverage.

### Table II

*The extended Neighogram Clustering Algorithm.*

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>(\forall x_i: c(x_i)) is minority class (\Rightarrow) compute (\text{NG}_i)</td>
</tr>
<tr>
<td>2)</td>
<td>(\forall x_i: c(x_i)) is minority class (\Rightarrow) store (\eta(x_i) = 1)</td>
</tr>
<tr>
<td>3)</td>
<td>(\forall \text{NG}_i): compute (\Omega_i)</td>
</tr>
<tr>
<td>4)</td>
<td>(\forall \text{NG}_i): compute (\Phi_i(\Omega_i))</td>
</tr>
<tr>
<td>5)</td>
<td>(\psi := 0)</td>
</tr>
<tr>
<td>6)</td>
<td>while (\psi &lt; \eta)</td>
</tr>
<tr>
<td>7)</td>
<td>(i_{\text{best}} = \arg \max_{i} { \Phi_i(\Omega_i) })</td>
</tr>
<tr>
<td>8)</td>
<td>add (\text{NG}<em>{i</em>{\text{best}}}) to list of clusters, add (\Phi_{i_{\text{best}}}(\Omega_{i_{\text{best}}})) to (\psi)</td>
</tr>
<tr>
<td>9)</td>
<td>recompute (\eta) for each pattern and</td>
</tr>
<tr>
<td>10)</td>
<td>hide them according to their (\eta) from all Neighograms (\text{NG}_i)</td>
</tr>
<tr>
<td>11)</td>
<td>(\forall \text{NG}_i): recompute (\Phi_i(\Omega_i))</td>
</tr>
<tr>
<td>12)</td>
<td>end while</td>
</tr>
</tbody>
</table>

**III. Results**

We used four standard data sets from the StatLog project [5] to evaluate the generalization ability of the clusters generated by the presented algorithm. In order to be able to compare the new method to published results, we generated clusters for all classes, i.e. for each class we built Neighograms to discriminate against all other classes. The classification outcome (the predicted class) was determined by the maximum of the sum of the degrees of membership from all Neighograms of all classes. Note that for all membership functions except the gaussian one it is possible to have an output of 0.0 for all classes, in which case the default class was predicted, that is, the class with the highest apriori occurrence frequency.

### A. Experiments

From the StatLog project [5] the four data sets listed in Table III were taken. We followed the procedure there and
TABLE III

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>#classes</th>
<th>Size (trn—tst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SatImage</td>
<td>36</td>
<td>6</td>
<td>(4435, 2000)</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>2</td>
<td>(768, 12-fold)</td>
</tr>
<tr>
<td>Segment</td>
<td>19</td>
<td>7</td>
<td>(2310, 10-fold)</td>
</tr>
<tr>
<td>DNA</td>
<td>180 (binary)</td>
<td>3</td>
<td>(2000, 1186)</td>
</tr>
</tbody>
</table>

TABLE IV
MISCLASSIFICATION RATES FOR THE DATA SETS IN TABLE III FOR A NUMBER OF STANDARD ALGORITHMS (TOP) TOGETHER WITH RESULTS FOR THE NEIGHBORGRAM CLASSIFIER (BOTTOM) USING THE MAXIMUM OF THE SUM OF THE DEGREES OF MEMBERSHIP TO DETERMINE THE PREDICTED CLASS (SEE TEXT FOR DETAILS). SHOWN ARE MISCLASSIFICATION RATES IN PERCENT.

<table>
<thead>
<tr>
<th>Name</th>
<th>c4.5</th>
<th>kNN</th>
<th>MLP</th>
<th>PNN</th>
<th>DDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SatImage</td>
<td>15.0</td>
<td>9.4</td>
<td>13.9</td>
<td>9.8</td>
<td>8.9</td>
</tr>
<tr>
<td>Diabetes</td>
<td>27.0</td>
<td>32.4</td>
<td>24.8</td>
<td>24.9</td>
<td>24.1</td>
</tr>
<tr>
<td>Segment</td>
<td>4.0</td>
<td>7.7</td>
<td>5.4</td>
<td>3.5</td>
<td>3.9</td>
</tr>
<tr>
<td>DNA</td>
<td>7.6</td>
<td>14.6</td>
<td>8.8</td>
<td>10.5</td>
<td>16.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>NG-T</th>
<th>NG-R</th>
<th>NG-Δ</th>
<th>NG-G 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SatImage</td>
<td>14.4</td>
<td>15.9</td>
<td>9.9</td>
<td>8.1 (1.8 · 10^{-5})</td>
</tr>
<tr>
<td>Diabetes</td>
<td>29.8</td>
<td>31.4</td>
<td>27.1</td>
<td>24.4 (3.0 · 10^{-1})</td>
</tr>
<tr>
<td>Segment</td>
<td>8.1</td>
<td>9.7</td>
<td>3.9</td>
<td>3.5 (1.0 · 10^{-6})</td>
</tr>
<tr>
<td>DNA</td>
<td>25.7</td>
<td>28.0</td>
<td>19.1</td>
<td>16.6 (1.0 · 10^{-12})</td>
</tr>
</tbody>
</table>

either used the specified division in training– and test data or applied n–fold cross–validation.

Results for the Neighborgram Classifier are shown in Table IV along with results for the decision tree algorithm c4.5, k-nearest neighbor, and a multi layer perceptron (all results from [5]). In addition we list results for Probabilistic Neural Networks [6], trained by the original algorithm (PNN) where each neuron covers one training example and generated constructively through the DDA algorithm [4]. The second table shows the classification error for various membership functions (NG-T: trapezoidal, NG-R: rectangular, NG-Δ: triangular, NG-G (θ): gaussian with respective parameters) of the Neighborgram Classifier.

Note how in three cases (SatImage, Diabetes and Segment) the classification accuracies of the triangular Neighborgram Classifier (NG-Δ) and the gaussian one (NG-G) compare nicely to the other methods (and also the remainder of the algorithms reported in the StatLog project). For the DNA data set, however, the NG Classifier performs somewhat worse no matter what membership function is used. This is mostly due to the restrictive, local nature of the clusters used by the NG Clustering Algorithm. In case of the DNA data set the generated NG clusters report no class for over 20% of the test patterns, indicating an insufficient coverage of the feature space. Considering that this a binary problem, it is not surprising that a distance based method fails to perform well.

\footnote{All results are using a summed degree of membership maximum class decision making process (Sum).}

\footnote{The parameter θ is given in parenthesis.}

As before, whenever the classifier did not generate a prediction, the default class was used.

Using the proposed method as a standard clustering algorithm clearly achieves satisfactory results and could therefore be used as an automatic classifier. However, as we have mentioned before, these Neighborgrams can also be used to visualize cluster candidates. In the following we briefly discuss this property.

IV. VISUAL CLUSTERING

One focus of our research was on a very interesting extension: The fact that a Neighborgram requires only one dimension, i.e. the distance to the centroid, offers the opportunity to visualize the corresponding neighborhood. In addition we can also project the membership function for a given purity \( p_{\text{min}} \) onto the Neighborgram, so the user is able to select and fine-tune potential cluster candidates. In conjunction with the proposed technique described above, this results in a system which suggests potential clusters to the user who then can evaluate the clusters’ interestingness. In [7] it is demonstrated how the algorithm and the corresponding visualization routine as can also be seen from the mediocre results by the other local methods, kNN and DDA.

Note that the Neighborgram Classifier using Gaussian Membership Functions shows superior performance to almost all other algorithms, especially the neighborhood based algorithms, which is due to the better, i.e. non-greedy selection of cluster centers. In addition, the Gaussian Membership functions always produces a non-zero output and therefore predict a class. However, in most cases a classifier that also produces a “don’t know” answer is preferable, since they allow to defer an obviously uncertain classification to an expert or to another system. In the following we will concentrate on the other membership functions instead, also because they allow for a more intuitive visualization.

Additional experiments were conducted to investigate how the used classification computation affected the outcome. Table V list the results on the same four data sets for the triangular membership function and three different ways to determine the predicted class.

- standard (Max): i.e. use maximum degree of membership (unweighted!),
- sum (Sum): sum up degrees of membership for each class and determine maximum, and
- weighted sum (wSum): sum up all weighted degrees of membership for each class and determine maximum.

As before, whenever the classifier did not generate a prediction, the default class was used.

Using the proposed method as a standard clustering algorithm clearly achieves satisfactory results and could therefore be used as an automatic classifier. However, as we have mentioned before, these Neighborgrams can also be used to visualize cluster candidates. In the following we briefly discuss this property.
work on the Iris data set and the usefulness of the visual clustering concept is shown on a real world data set from NIH.

**example 4.1:** Figure 2 shows two Neighborgrams for patterns of the Iris data set with the corresponding cluster membership function. In case that two or more patterns are too close to each other so that they would overlap we decided to stack them on top of each other, so patterns can be individually selected and are highlighted in other Neighborgrams — or even other views — as well. The vertical axes therefore has no geometrical meaning, it is simply used to avoid overlaps. In turn, for the displayed membership function the vertical axes does have a meaning, i.e. the degree of membership. Note how the quality of these two cluster becomes visible immediately. The user can now select individual clusters and change their membership functions.

Fig. 2. Two Neighborgrams built for patterns of the Iris data set. Two classes (grey and black) can be distinguished. See [7] for further details.

V. DISCUSSION

A couple of issues that we do not have space to discuss in detail, but that are worth being touched upon at least briefly are listed in the following.

A. Binning Neighborgrams

Obviously for only few hundreds of patterns in each Neighborgram it is possible to plot all patterns individually. For larger neighborhoods it is preferable to bin the neighborhood and just display how many patterns of each class are encountered in each bin. We have experimented with this type of display as well but for all our applications smaller neighborhoods have shown to be sufficient to find good clusters, which in domains centered around chemical descriptors does not come as a surprise as larger neighborhoods have been shown to show not much predictive ability in most cases anyway [8].

B. Fuzzy Class Membership

In some applications class information is not as exact as the example above or the presented benchmark data sets seem to suggest. Here fuzzifying the class information as well could allow to build better clusters. The purity of a cluster candidate would then be computed based on the degree of membership to the correct vs. conflicting class. We have not yet implemented this feature but believe that it offers promise for less perfect data, such as activity information from high throughput screening of drug candidates.

C. Parallel Universes

Visual Clustering and Clustering using Neighborgrams are both very promising concepts. The Neighborgram approach in particular has one interesting extension that we are currently exploring. When looking at the algorithm in Table I one notes that the clusters don’t “interact” directly. Besides the fact that a chosen cluster removes covered patterns from consideration there is no obvious need for two clusters to be based on the same distance function or even be built in the same feature space. This leads to the notion of **Parallel Universes**, where we can find clusters in different feature spaces in parallel. Especially for data sets that involve structural descriptions of molecules it is hardly ever known which descriptor is optimal for a particular problem at hand. We can then modify the clustering algorithm to investigate all descriptor spaces in parallel and choose the best cluster among all universes in parallel. Covered patterns will then be removed from all universes and the result is a set of clusters, spread out over different descriptor spaces.

VI. CONCLUSIONS

We have presented a method to build clusters based on Neighborgrams, which model the local distribution of patterns for each potential cluster candidate. The underlying cluster algorithm has two main parameters, these are the purity, which determines the boundary of a cluster candidate, and a termination criterion. Both parameters are easy to interpret and therefore not very crucial.

We used four publicly available data sets from the StatLog project to show that the algorithm achieves satisfactory classification accuracy and we proved that the introduction of fuzzy boundaries increases the performance of the cluster algorithm substantially, so that the results are comparable to other state-of-the-art techniques.

The accompanying visualization provides means to explore the proposed cluster selection and enables the user to inject domain knowledge into the clustering process, as discussed in more detail in [7].

REFERENCES